Embedded Software and Motor Control Libraries for PXR40xx

User's Guide

Document Number: PXR40XXMCLUG
Rev. 1.0

Contents

Section number **Title** **Page**

Chapter 1

1.1 License Agreement....101

Chapter 2
Introduction

2.1 Architecture Overview....109
2.2 General Information....110
2.3 Multiple Implementation Support....111
2.3.1 Global Configuration Option....112
2.3.2 Additional Parameter Option....113
2.3.3 API Postfix Option....113
2.4 Supported Compilers....113
2.5 Matlab Integration....115
2.6 Installation....116
2.7 Library File Structure....118
2.8 Integration Assumption....120
2.9 Library Integration into a Green Hills Multi Development Environment....120
2.10 Library Integration into a Wind River Compiler Environment....123
2.11 Library Integration into a Freescale CodeWarrior Classic IDE....126
2.12 Library Integration into a Freescale CodeWarrior Eclipse IDE....128
2.13 MC Library Testing....132
2.13.1 MCLib Testing based on the MATLAB Simulink Toolbox....133
2.13.2 MCLib target-in-loop Testing based on the SFIO Toolbox....134

Chapter 3

3.1 Function Index....137

Chapter 4
API References

4.1 Function GDFLIB_FilterFIRInit_F32....149
4.1.1 Declaration....149
4.1.2 Arguments....149
4.1.3 Return....149
4.1.4 Description....150
4.1.5 Re-entrancy....150
4.1.6 Code Example....150
4.2 Function GDFLIB_FilterFIR_F32....152
4.2.1 Declaration....152
4.2.2 Arguments....152
4.2.3 Return....152
4.2.4 Description....152
4.2.5 Re-entrancy....153
4.2.6 Code Example....153
4.3 Function GDFLIB_FilterFIRInit_F16....155
4.3.1 Declaration....155
4.3.2 Arguments....155
4.3.3 Return....155
4.3.4 Description....156
4.3.5 Re-entrancy....156
4.3.6 Code Example....156
4.4 Function GDFLIB_FilterFIR_F16....158
4.4.1 Declaration....158
4.4.2 Arguments....158
4.4.3 Return....158
4.4.4 Description....158
4.4.5 Re-entrancy....159

4.4.6 Code Example....159
4.5 Function GDFLIB_FilterFIRInit_FLT....161
4.5.1 Declaration....161
4.5.2 Arguments....161
4.5.3 Return....161
4.5.4 Description....162
4.5.5 Re-entrancy....162
4.5.6 Code Example....162
4.6 Function GDFLIB_FilterFIR_FLT....164
4.6.1 Declaration....164
4.6.2 Arguments....164
4.6.3 Return....164
4.6.4 Description....164
4.6.5 Re-entrancy....165
4.6.6 Code Example....165
4.7 Function GDFLIB_FilterIIR1Init_F32....167
4.7.1 Declaration....167
4.7.2 Arguments....167
4.7.3 Return....167
4.7.4 Description....167
4.7.5 Re-entrancy....167
4.7.6 Code Example....168
4.8 Function GDFLIB_FilterIIR1_F32....168
4.8.1 Declaration....168
4.8.2 Arguments....168
4.8.3 Return....169
4.8.4 Description....169
4.8.5 Re-entrancy....171
4.8.6 Code Example....171

4.9 Function GDFLIB_FilterIIR1Init_F16....171
4.9.1 Declaration....172
4.9.2 Arguments....172
4.9.3 Return....172
4.9.4 Description....172
4.9.5 Re-entrancy....172
4.9.6 Code Example....172
4.10 Function GDFLIB_FilterIIR1_F16....173
4.10.1 Declaration....173
4.10.2 Arguments....173
4.10.3 Return....173
4.10.4 Description....174
4.10.5 Re-entrancy....175
4.10.6 Code Example....176
4.11 Function GDFLIB_FilterIIR1Init_FLT....176
4.11.1 Declaration....176
4.11.2 Arguments....176
4.11.3 Return....177
4.11.4 Description....177
4.11.5 Re-entrancy....177
4.11.6 Code Example....177
4.12 Function GDFLIB_FilterIIR1_FLT....178
4.12.1 Declaration....178
4.12.2 Arguments....178
4.12.3 Return....178
4.12.4 Description....178
4.12.5 Re-entrancy....180
4.12.6 Code Example....180

4.13 Function GDFLIB_FilterIIR2Init_F32....181
4.13.1 Declaration....181
4.13.2 Arguments....181
4.13.3 Return....181
4.13.4 Description....181
4.13.5 Re-entrancy....182
4.13.6 Code Example....182
4.14 Function GDFLIB_FilterIIR2_F32....182
4.14.1 Declaration....182
4.14.2 Arguments....183
4.14.3 Return....183
4.14.4 Description....183
4.14.5 Re-entrancy....185
4.14.6 Code Example....185
4.15 Function GDFLIB_FilterIIR2Init_F16....186
4.15.1 Declaration....186
4.15.2 Arguments....186
4.15.3 Return....186
4.15.4 Description....186
4.15.5 Re-entrancy....187
4.15.6 Code Example....187
4.16 Function GDFLIB_FilterIIR2_F16....187
4.16.1 Declaration....187
4.16.2 Arguments....188
4.16.3 Return....188
4.16.4 Description....188
4.16.5 Re-entrancy....190
4.16.6 Code Example....190

4.17 Function GDFLIB_FilterIIR2Init_FLT....191
4.17.1 Declaration....191
4.17.2 Arguments....191
4.17.3 Return....191
4.17.4 Description....191
4.17.5 Re-entrancy....192
4.17.6 Code Example....192
4.18 Function GDFLIB_FilterIIR2_FLT....192
4.18.1 Declaration....192
4.18.2 Arguments....193
4.18.3 Return....193
4.18.4 Description....193
4.18.5 Re-entrancy....195
4.18.6 Code Example....195
4.19 Function GDFLIB_FilterMAInit_F32....196
4.19.1 Declaration....196
4.19.2 Arguments....196
4.19.3 Return....196
4.19.4 Description....196
4.19.5 Re-entrancy....197
4.19.6 Code Example....197
4.20 Function GDFLIB_FilterMA_F32....197
4.20.1 Declaration....197
4.20.2 Arguments....197
4.20.3 Return....198
4.20.4 Description....198
4.20.5 Re-entrancy....199
4.20.6 Code Example....199

4.21 Function GDFLIB_FilterMAInit_F16....200
4.21.1 Declaration....200
4.21.2 Arguments....200
4.21.3 Return....200
4.21.4 Description....200
4.21.5 Re-entrancy....201
4.21.6 Code Example....201
4.22 Function GDFLIB_FilterMA_F16....201
4.22.1 Declaration....201
4.22.2 Arguments....201
4.22.3 Return....202
4.22.4 Description....202
4.22.5 Re-entrancy....203
4.22.6 Code Example....203
4.23 Function GDFLIB_FilterMAInit_FLT....204
4.23.1 Declaration....204
4.23.2 Arguments....204
4.23.3 Return....204
4.23.4 Description....204
4.23.5 Re-entrancy....205
4.23.6 Code Example....205
4.24 Function GDFLIB_FilterMA_FLT....205
4.24.1 Declaration....205
4.24.2 Arguments....205
4.24.3 Return....206
4.24.4 Description....206
4.24.5 Re-entrancy....207
4.24.6 Code Example....207

4.25 Function GFLIB_Acos_F32....208
4.25.1 Declaration....208
4.25.2 Arguments....208
4.25.3 Return....208
4.25.4 Description....208
4.25.5 Re-entrancy....212
4.25.6 Code Example....212
4.26 Function GFLIB_Acos_F16....212
4.26.1 Declaration....212
4.26.2 Arguments....212
4.26.3 Return....213
4.26.4 Description....213
4.26.5 Re-entrancy....216
4.26.6 Code Example....216
4.27 Function GFLIB_Acos_FLT....217
4.27.1 Declaration....217
4.27.2 Arguments....217
4.27.3 Return....217
4.27.4 Description....217
4.27.5 Re-entrancy....220
4.27.6 Code Example....220
4.28 Function GFLIB_Asin_F32....221
4.28.1 Declaration....221
4.28.2 Arguments....221
4.28.3 Return....221
4.28.4 Description....221
4.28.5 Re-entrancy....225
4.28.6 Code Example....225

4.29 Function GFLIB_Asin_F16....225
4.29.1 Declaration....225
4.29.2 Arguments....225
4.29.3 Return....226
4.29.4 Description....226
4.29.5 Re-entrancy....229
4.29.6 Code Example....229
4.30 Function GFLIB_Asin_FLT....229
4.30.1 Declaration....229
4.30.2 Arguments....230
4.30.3 Return....230
4.30.4 Description....230
4.30.5 Re-entrancy....233
4.30.6 Code Example....233
4.31 Function GFLIB_Atan_F32....233
4.31.1 Declaration....234
4.31.2 Arguments....234
4.31.3 Return....234
4.31.4 Description....234
4.31.5 Re-entrancy....237
4.31.6 Code Example....237
4.32 Function GFLIB_Atan_F16....238
4.32.1 Declaration....238
4.32.2 Arguments....238
4.32.3 Return....238
4.32.4 Description....238
4.32.5 Re-entrancy....241
4.32.6 Code Example....242

4.33 Function GFLIB_Atan_FLT....242
4.33.1 Declaration....242
4.33.2 Arguments....242
4.33.3 Return....243
4.33.4 Description....243
4.33.5 Re-entrancy....246
4.33.6 Code Example....246
4.34 Function GFLIB_AtanYX_F32....246
4.34.1 Declaration....246
4.34.2 Arguments....246
4.34.3 Return....247
4.34.4 Description....247
4.34.5 Re-entrancy....248
4.34.6 Code Example....248
4.35 Function GFLIB_AtanYX_F16....248
4.35.1 Declaration....248
4.35.2 Arguments....249
4.35.3 Return....249
4.35.4 Description....249
4.35.5 Re-entrancy....250
4.35.6 Code Example....250
4.36 Function GFLIB_AtanYX_FLT....250
4.36.1 Declaration....251
4.36.2 Arguments....251
4.36.3 Return....251
4.36.4 Description....251
4.36.5 Re-entrancy....252
4.36.6 Code Example....252

4.37 Function GFLIB_AtanYXShifted_F32....252
4.37.1 Declaration....252
4.37.2 Arguments....253
4.37.3 Return....253
4.37.4 Description....253
4.37.5 Re-entrancy....257
4.37.6 Code Example....257
4.38 Function GFLIB_AtanYXShifted_F16....258
4.38.1 Declaration....258
4.38.2 Arguments....258
4.38.3 Return....258
4.38.4 Description....258
4.38.5 Re-entrancy....262
4.38.6 Code Example....262
4.39 Function GFLIB_AtanYXShifted_FLT....263
4.39.1 Declaration....263
4.39.2 Arguments....263
4.39.3 Return....263
4.39.4 Description....264
4.39.5 Re-entrancy....266
4.39.6 Code Example....266
4.40 Function GFLIB_ControllerPIp_F32....267
4.40.1 Declaration....267
4.40.2 Arguments....267
4.40.3 Return....268
4.40.4 Description....268
4.40.5 Re-entrancy....271
4.40.6 Code Example....271

4.41 Function GFLIB_ControllerPIp_F16....271
4.41.1 Declaration....272
4.41.2 Arguments....272
4.41.3 Return....272
4.41.4 Description....272
4.41.5 Re-entrancy....275
4.41.6 Code Example....275
4.42 Function GFLIB_ControllerPIp_FLT....276
4.42.1 Declaration....276
4.42.2 Arguments....276
4.42.3 Return....276
4.42.4 Description....276
4.42.5 Re-entrancy....278
4.42.6 Code Example....278
4.43 Function GFLIB_ControllerPIpAW_F32....279
4.43.1 Declaration....279
4.43.2 Arguments....279
4.43.3 Return....279
4.43.4 Description....279
4.43.5 Re-entrancy....283
4.43.6 Code Example....283
4.44 Function GFLIB_ControllerPIpAW_F16....284
4.44.1 Declaration....284
4.44.2 Arguments....284
4.44.3 Return....284
4.44.4 Description....284
4.44.5 Re-entrancy....288
4.44.6 Code Example....288

4.45 Function GFLIB_ControllerPIpAW_FLT....289
4.45.1 Declaration....289
4.45.2 Arguments....289
4.45.3 Return....289
4.45.4 Description....289
4.45.5 Re-entrancy....291
4.45.6 Code Example....292
4.46 Function GFLIB_ControllerPIr_F32....292
4.46.1 Declaration....292
4.46.2 Arguments....292
4.46.3 Return....293
4.46.4 Description....293
4.46.5 Re-entrancy....296
4.46.6 Code Example....296
4.47 Function GFLIB_ControllerPIr_F16....297
4.47.1 Declaration....297
4.47.2 Arguments....297
4.47.3 Return....297
4.47.4 Description....297
4.47.5 Re-entrancy....300
4.47.6 Code Example....300
4.48 Function GFLIB_ControllerPIr_FLT....301
4.48.1 Declaration....301
4.48.2 Arguments....301
4.48.3 Return....301
4.48.4 Description....302
4.48.5 Re-entrancy....303
4.48.6 Code Example....303

4.49 Function GFLIB_ControllerPIrAW_F32....303
4.49.1 Declaration....304
4.49.2 Arguments....304
4.49.3 Return....304
4.49.4 Description....304
4.49.5 Re-entrancy....308
4.49.6 Code Example....308
4.50 Function GFLIB_ControllerPIrAW_F16....308
4.50.1 Declaration....309
4.50.2 Arguments....309
4.50.3 Return....309
4.50.4 Description....309
4.50.5 Re-entrancy....313
4.50.6 Code Example....313
4.51 Function GFLIB_ControllerPIrAW_FLT....313
4.51.1 Declaration....313
4.51.2 Arguments....314
4.51.3 Return....314
4.51.4 Description....314
4.51.5 Re-entrancy....316
4.51.6 Code Example....316
4.52 Function GFLIB_Cos_F32....316
4.52.1 Declaration....316
4.52.2 Arguments....317
4.52.3 Return....317
4.52.4 Description....317
4.52.5 Re-entrancy....321
4.52.6 Code Example....321

4.53 Function GFLIB_Cos_F16....321
4.53.1 Declaration....321
4.53.2 Arguments....321
4.53.3 Return....322
4.53.4 Description....322
4.53.5 Re-entrancy....326
4.53.6 Code Example....326
4.54 Function GFLIB_Cos_FLT....326
4.54.1 Declaration....326
4.54.2 Arguments....326
4.54.3 Return....327
4.54.4 Description....327
4.54.5 Re-entrancy....330
4.54.6 Code Example....330
4.55 Function GFLIB_Hyst_F32....331
4.55.1 Declaration....331
4.55.2 Arguments....331
4.55.3 Return....331
4.55.4 Description....331
4.55.5 Re-entrancy....332
4.55.6 Code Example....332
4.56 Function GFLIB_Hyst_F16....333
4.56.1 Declaration....333
4.56.2 Arguments....333
4.56.3 Return....333
4.56.4 Description....334
4.56.5 Re-entrancy....335
4.56.6 Code Example....335

4.57 Function GFLIB_Hyst_FLT....335
4.57.1 Declaration....336
4.57.2 Arguments....336
4.57.3 Return....336
4.57.4 Description....336
4.57.5 Re-entrancy....337
4.57.6 Code Example....337
4.58 Function GFLIB_IntegratorTR_F32....338
4.58.1 Declaration....338
4.58.2 Arguments....338
4.58.3 Return....338
4.58.4 Description....339
4.58.5 Re-entrancy....340
4.58.6 Code Example....340
4.59 Function GFLIB_IntegratorTR_F16....341
4.59.1 Declaration....341
4.59.2 Arguments....341
4.59.3 Return....341
4.59.4 Description....342
4.59.5 Re-entrancy....343
4.59.6 Code Example....343
4.60 Function GFLIB_IntegratorTR_FLT....344
4.60.1 Declaration....344
4.60.2 Arguments....344
4.60.3 Return....344
4.60.4 Description....345
4.60.5 Re-entrancy....345
4.60.6 Code Example....346

4.61 Function GFLIB_Limit_F32....346
4.61.1 Declaration....346
4.61.2 Arguments....346
4.61.3 Return....347
4.61.4 Description....347
4.61.5 Re-entrancy....347
4.61.6 Code Example....347
4.62 Function GFLIB_Limit_F16....348
4.62.1 Declaration....348
4.62.2 Arguments....348
4.62.3 Return....348
4.62.4 Description....349
4.62.5 Re-entrancy....349
4.62.6 Code Example....349
4.63 Function GFLIB_Limit_FLT....350
4.63.1 Declaration....350
4.63.2 Arguments....350
4.63.3 Return....350
4.63.4 Description....350
4.63.5 Re-entrancy....351
4.63.6 Code Example....351
4.64 Function GFLIB_LowerLimit_F32....351
4.64.1 Declaration....351
4.64.2 Arguments....351
4.64.3 Return....352
4.64.4 Description....352
4.64.5 Re-entrancy....352
4.64.6 Code Example....352

4.65 Function GFLIB_LowerLimit_F16....353
4.65.1 Declaration....353
4.65.2 Arguments....353
4.65.3 Return....353
4.65.4 Description....353
4.65.5 Re-entrancy....354
4.65.6 Code Example....354
4.66 Function GFLIB_LowerLimit_FLT....354
4.66.1 Declaration....354
4.66.2 Arguments....355
4.66.3 Return....355
4.66.4 Description....355
4.66.5 Re-entrancy....355
4.66.6 Code Example....355
4.67 Function GFLIB_Lut1D_F32....356
4.67.1 Declaration....356
4.67.2 Arguments....356
4.67.3 Return....356
4.67.4 Description....356
4.67.5 Re-entrancy....359
4.67.6 Code Example....360
4.68 Function GFLIB_Lut1D_F16....360
4.68.1 Declaration....360
4.68.2 Arguments....360
4.68.3 Return....361
4.68.4 Description....361
4.68.5 Re-entrancy....364
4.68.6 Code Example....364

4.69 Function GFLIB_Lut1D_FLT....364
4.69.1 Declaration....365
4.69.2 Arguments....365
4.69.3 Return....365
4.69.4 Description....365
4.69.5 Re-entrancy....368
4.69.6 Code Example....368
4.70 Function GFLIB_Lut2D_F32....368
4.70.1 Declaration....368
4.70.2 Arguments....368
4.70.3 Return....369
4.70.4 Description....369
4.70.5 Re-entrancy....374
4.70.6 Code Example....374
4.71 Function GFLIB_Lut2D_F16....375
4.71.1 Declaration....375
4.71.2 Arguments....375
4.71.3 Return....375
4.71.4 Description....375
4.71.5 Re-entrancy....380
4.71.6 Code Example....380
4.72 Function GFLIB_Lut2D_FLT....381
4.72.1 Declaration....381
4.72.2 Arguments....381
4.72.3 Return....382
4.72.4 Description....382
4.72.5 Re-entrancy....386
4.72.6 Code Example....386

4.73 Function GFLIB_Ramp_F32....387
4.73.1 Declaration....387
4.73.2 Arguments....387
4.73.3 Return....388
4.73.4 Description....388
4.73.5 Re-entrancy....388
4.73.6 Code Example....389
4.74 Function GFLIB_Ramp_F16....389
4.74.1 Declaration....389
4.74.2 Arguments....389
4.74.3 Return....390
4.74.4 Description....390
4.74.5 Re-entrancy....391
4.74.6 Code Example....391
4.75 Function GFLIB_Ramp_FLT....392
4.75.1 Declaration....392
4.75.2 Arguments....392
4.75.3 Return....392
4.75.4 Description....392
4.75.5 Re-entrancy....393
4.75.6 Code Example....393
4.76 Function GFLIB_Sign_F32....394
4.76.1 Declaration....394
4.76.2 Arguments....394
4.76.3 Return....394
4.76.4 Description....395
4.76.5 Re-entrancy....395
4.76.6 Code Example....395

4.77 Function GFLIB_Sign_F16....396
4.77.1 Declaration....396
4.77.2 Arguments....396
4.77.3 Return....396
4.77.4 Description....396
4.77.5 Re-entrancy....397
4.77.6 Code Example....397
4.78 Function GFLIB_Sign_FLT....397
4.78.1 Declaration....398
4.78.2 Arguments....398
4.78.3 Return....398
4.78.4 Description....398
4.78.5 Re-entrancy....399
4.78.6 Code Example....399
4.79 Function GFLIB_Sin_F32....399
4.79.1 Declaration....399
4.79.2 Arguments....399
4.79.3 Return....400
4.79.4 Description....400
4.79.5 Re-entrancy....404
4.79.6 Code Example....404
4.80 Function GFLIB_Sin_F16....404
4.80.1 Declaration....404
4.80.2 Arguments....404
4.80.3 Return....405
4.80.4 Description....405
4.80.5 Re-entrancy....408
4.80.6 Code Example....408

4.81 Function GFLIB_Sin_FLT....409
4.81.1 Declaration....409
4.81.2 Arguments....409
4.81.3 Return....409
4.81.4 Description....409
4.81.5 Re-entrancy....412
4.81.6 Code Example....412
4.82 Function GFLIB_Sqrt_F32....412
4.82.1 Declaration....412
4.82.2 Arguments....413
4.82.3 Return....413
4.82.4 Description....413
4.82.5 Re-entrancy....414
4.82.6 Code Example....414
4.83 Function GFLIB_Sqrt_F16....415
4.83.1 Declaration....415
4.83.2 Arguments....415
4.83.3 Return....415
4.83.4 Description....415
4.83.5 Re-entrancy....416
4.83.6 Code Example....416
4.84 Function GFLIB_Sqrt_FLT....417
4.84.1 Declaration....417
4.84.2 Arguments....417
4.84.3 Return....417
4.84.4 Description....417
4.84.5 Re-entrancy....419
4.84.6 Code Example....419

4.85 Function GFLIB_Tan_F32....420
4.85.1 Declaration....420
4.85.2 Arguments....420
4.85.3 Return....420
4.85.4 Description....420
4.85.5 Re-entrancy....424
4.85.6 Code Example....424
4.86 Function GFLIB_Tan_F16....424
4.86.1 Declaration....424
4.86.2 Arguments....425
4.86.3 Return....425
4.86.4 Description....425
4.86.5 Re-entrancy....429
4.86.6 Code Example....429
4.87 Function GFLIB_Tan_FLT....429
4.87.1 Declaration....429
4.87.2 Arguments....429
4.87.3 Return....430
4.87.4 Description....430
4.87.5 Re-entrancy....434
4.87.6 Code Example....434
4.88 Function GFLIB_UpperLimit_F32....434
4.88.1 Declaration....434
4.88.2 Arguments....435
4.88.3 Return....435
4.88.4 Description....435
4.88.5 Re-entrancy....435
4.88.6 Code Example....435

4.89 Function GFLIB_UpperLimit_F16....436
4.89.1 Declaration....436
4.89.2 Arguments....436
4.89.3 Return....436
4.89.4 Description....436
4.89.5 Re-entrancy....437
4.89.6 Code Example....437
4.90 Function GFLIB_UpperLimit_FLT....437
4.90.1 Declaration....437
4.90.2 Arguments....438
4.90.3 Return....438
4.90.4 Description....438
4.90.5 Re-entrancy....438
4.90.6 Code Example....438
4.91 Function GFLIB_VectorLimit_F32....439
4.91.1 Declaration....439
4.91.2 Arguments....439
4.91.3 Return....439
4.91.4 Description....440
4.91.5 Re-entrancy....441
4.91.6 Code Example....441
4.92 Function GFLIB_VectorLimit_F16....442
4.92.1 Declaration....442
4.92.2 Arguments....442
4.92.3 Return....443
4.92.4 Description....443
4.92.5 Re-entrancy....444
4.92.6 Code Example....445

4.93 Function GFLIB_VectorLimit_FLT....445
4.93.1 Declaration....445
4.93.2 Arguments....446
4.93.3 Return....446
4.93.4 Description....446
4.93.5 Re-entrancy....447
4.93.6 Code Example....448
4.94 Function GMCLIB_Clark_F32....448
4.94.1 Declaration....448
4.94.2 Arguments....448
4.94.3 Return....449
4.94.4 Description....449
4.94.5 Re-entrancy....449
4.94.6 Code Example....450
4.95 Function GMCLIB_Clark_F16....450
4.95.1 Declaration....450
4.95.2 Arguments....451
4.95.3 Return....451
4.95.4 Description....451
4.95.5 Re-entrancy....451
4.95.6 Code Example....452
4.96 Function GMCLIB_Clark_FLT....452
4.96.1 Declaration....452
4.96.2 Arguments....453
4.96.3 Return....453
4.96.4 Description....453
4.96.5 Re-entrancy....453
4.96.6 Code Example....454

4.97 Function GMCLIB_ClarkInv_F32....454
4.97.1 Declaration....454
4.97.2 Arguments....454
4.97.3 Return....455
4.97.4 Description....455
4.97.5 Re-entrancy....456
4.97.6 Code Example....456
4.98 Function GMCLIB_ClarkInv_F16....456
4.98.1 Declaration....456
4.98.2 Arguments....457
4.98.3 Return....457
4.98.4 Description....457
4.98.5 Re-entrancy....458
4.98.6 Code Example....458
4.99 Function GMCLIB_ClarkInv_FLT....458
4.99.1 Declaration....458
4.99.2 Arguments....459
4.99.3 Return....459
4.99.4 Description....459
4.99.5 Re-entrancy....460
4.99.6 Code Example....460
4.100 Function GMCLIB_DecouplingPMSM_F32....460
4.100.1 Declaration....461
4.100.2 Arguments....461
4.100.3 Return....461
4.100.4 Description....461
4.100.5 Re-entrancy....465
4.100.6 Code Example....465

4.101 Function GMCLIB_DecouplingPMSM_F16....466
4.101.1 Declaration....466
4.101.2 Arguments....466
4.101.3 Return....467
4.101.4 Description....467
4.101.5 Re-entrancy....471
4.101.6 Code Example....471
4.102 Function GMCLIB_DecouplingPMSM_FLT....472
4.102.1 Declaration....472
4.102.2 Arguments....472
4.102.3 Return....472
4.102.4 Description....473
4.102.5 Re-entrancy....475
4.102.6 Code Example....475
4.103 Function GMCLIB_ElimDcBusRip_F32....475
4.103.1 Declaration....475
4.103.2 Arguments....476
4.103.3 Return....476
4.103.4 Description....476
4.103.5 Re-entrancy....478
4.103.6 Code Example....478
4.104 Function GMCLIB_ElimDcBusRip_F16....479
4.104.1 Declaration....479
4.104.2 Arguments....479
4.104.3 Return....480
4.104.4 Description....480
4.104.5 Re-entrancy....482
4.104.6 Code Example....482

4.105 Function GMCLIB_ElimDcBusRip_FLT....483
4.105.1 Declaration....483
4.105.2 Arguments....483
4.105.3 Return....484
4.105.4 Description....484
4.105.5 Re-entrancy....486
4.105.6 Code Example....486
4.106 Function GMCLIB_Park_F32....486
4.106.1 Declaration....487
4.106.2 Arguments....487
4.106.3 Return....487
4.106.4 Description....487
4.106.5 Re-entrancy....488
4.106.6 Code Example....488
4.107 Function GMCLIB_Park_F16....488
4.107.1 Declaration....489
4.107.2 Arguments....489
4.107.3 Return....489
4.107.4 Description....489
4.107.5 Re-entrancy....490
4.107.6 Code Example....490
4.108 Function GMCLIB_Park_FLT....490
4.108.1 Declaration....491
4.108.2 Arguments....491
4.108.3 Return....491
4.108.4 Description....491
4.108.5 Re-entrancy....492
4.108.6 Code Example....492

4.109 Function GMCLIB_ParkInv_F32....492
4.109.1 Declaration....493
4.109.2 Arguments....493
4.109.3 Return....493
4.109.4 Description....493
4.109.5 Re-entrancy....494
4.109.6 Code Example....494
4.110 Function GMCLIB_ParkInv_F16....494
4.110.1 Declaration....495
4.110.2 Arguments....495
4.110.3 Return....495
4.110.4 Description....495
4.110.5 Re-entrancy....496
4.110.6 Code Example....496
4.111 Function GMCLIB_ParkInv_FLT....496
4.111.1 Declaration....497
4.111.2 Arguments....497
4.111.3 Return....497
4.111.4 Description....497
4.111.5 Re-entrancy....498
4.111.6 Code Example....498
4.112 Function GMCLIB_SvmStd_F32....498
4.112.1 Declaration....499
4.112.2 Arguments....499
4.112.3 Return....499
4.112.4 Description....499
4.112.5 Re-entrancy....512
4.112.6 Code Example....512

4.113 Function GMCLIB_SvmStd_F16....513
4.113.1 Declaration....513
4.113.2 Arguments....513
4.113.3 Return....513
4.113.4 Description....514
4.113.5 Re-entrancy....526
4.113.6 Code Example....526
4.114 Function GMCLIB_SvmStd_FLT....527
4.114.1 Declaration....527
4.114.2 Arguments....527
4.114.3 Return....527
4.114.4 Description....528
4.114.5 Re-entrancy....541
4.114.6 Code Example....541
4.115 Function MLIB_Abs_F32....541
4.115.1 Declaration....542
4.115.2 Arguments....542
4.115.3 Return....542
4.115.4 Description....542
4.115.5 Re-entrancy....543
4.115.6 Code Example....543
4.115.7 Re-entrancy....543
4.115.8 Code Example....544
4.116 Function MLIB_Abs_F16....544
4.116.1 Declaration....544
4.116.2 Arguments....544
4.116.3 Return....545
4.116.4 Description....545
4.116.5 Re-entrancy....545

4.116.6 Code Example....545
4.116.7 Re-entrancy....546
4.116.8 Code Example....546
4.117 Function MLIB_Abs_FLT....547
4.117.1 Declaration....547
4.117.2 Arguments....547
4.117.3 Return....547
4.117.4 Description....547
4.117.5 Re-entrancy....548
4.117.6 Code Example....548
4.117.7 Re-entrancy....549
4.117.8 Code Example....549
4.118 Function MLIB_AbsSat_F32....549
4.118.1 Declaration....549
4.118.2 Arguments....549
4.118.3 Return....550
4.118.4 Description....550
4.118.5 Re-entrancy....550
4.118.6 Code Example....550
4.119 Function MLIB_AbsSat_F16....551
4.119.1 Declaration....551
4.119.2 Arguments....551
4.119.3 Return....551
4.119.4 Description....551
4.119.5 Re-entrancy....552
4.119.6 Code Example....552
4.120 Function MLIB_Add_F32....553
4.120.1 Declaration....553
4.120.2 Arguments....553

4.120.3 Return....553
4.120.4 Description....553
4.120.5 Re-entrancy:....554
4.120.6 Code Example:....554
4.120.7 Re-entrancy:....555
4.120.8 Code Example:....555
4.121 Function MLIB_Add_F16....555
4.121.1 Declaration....555
4.121.2 Arguments....555
4.121.3 Return....556
4.121.4 Description....556
4.121.5 Re-entrancy....556
4.121.6 Code Example....557
4.121.7 Re-entrancy....557
4.121.8 Code Example....557
4.122 Function MLIB_Add_FLT....558
4.122.1 Declaration....558
4.122.2 Arguments....558
4.122.3 Return....558
4.122.4 Description....558
4.122.5 Re-entrancy....559
4.122.6 Code Example....559
4.122.7 Re-entrancy....560
4.122.8 Code Example....560
4.123 Function MLIB_AddSat_F32....560
4.123.1 Declaration....561
4.123.2 Arguments....561
4.123.3 Return....561
4.123.4 Description....561

4.123.5 Re-entrancy....562
4.123.6 Code Example....562
4.123.7 Re-entrancy....563
4.123.8 Code Example....563
4.124 Function MLIB_AddSat_F16....563
4.124.1 Declaration....563
4.124.2 Arguments....563
4.124.3 Return....564
4.124.4 Description....564
4.124.5 Re-entrancy....564
4.124.6 Code Example....565
4.124.7 Re-entrancy....565
4.124.8 Code Example....565
4.125 Function MLIB_Convert_F32F16....566
4.125.1 Declaration....566
4.125.2 Arguments....566
4.125.3 Return....566
4.125.4 Description....567
4.125.5 Re-entrancy....567
4.125.6 Code Example....567
4.126 Function MLIB_Convert_F32FLT....568
4.126.1 Declaration....568
4.126.2 Arguments....568
4.126.3 Return....568
4.126.4 Description....568
4.126.5 Re-entrancy....569
4.126.6 Code Example....569
4.127 Function MLIB_Convert_F16F32....570
4.127.1 Declaration....570

4.127.2 Arguments....570
4.127.3 Return....570
4.127.4 Description....570
4.127.5 Re-entrancy....571
4.127.6 Code Example....571
4.128 Function MLIB_Convert_F16FLT....572
4.128.1 Declaration....572
4.128.2 Arguments....572
4.128.3 Return....572
4.128.4 Description....572
4.128.5 Re-entrancy....573
4.128.6 Code Example....573
4.129 Function MLIB_Convert_FLTF16....573
4.129.1 Declaration....573
4.129.2 Arguments....574
4.129.3 Return....574
4.129.4 Description....574
4.129.5 Re-entrancy....574
4.129.6 Code Example....575
4.130 Function MLIB_Convert_FLTF32....575
4.130.1 Declaration....575
4.130.2 Arguments....575
4.130.3 Return....576
4.130.4 Description....576
4.130.5 Re-entrancy....576
4.130.6 Code Example....577
4.131 Function MLIB_ConvertPU_F32F16....577
4.131.1 Declaration....577
4.131.2 Arguments....577

4.131.3 Return....577
4.131.4 Description....578
4.131.5 Re-entrancy....578
4.131.6 Code Example....578
4.132 Function MLIB_ConvertPU_F32FLT....579
4.132.1 Declaration....579
4.132.2 Arguments....579
4.132.3 Return....579
4.132.4 Description....579
4.132.5 Re-entrancy....580
4.132.6 Code Example....580
4.133 Function MLIB_ConvertPU_F16F32....580
4.133.1 Declaration....580
4.133.2 Arguments....580
4.133.3 Return....580
4.133.4 Description....581
4.133.5 Re-entrancy....581
4.133.6 Code Example....581
4.134 Function MLIB_ConvertPU_F16FLT....582
4.134.1 Declaration....582
4.134.2 Arguments....582
4.134.3 Return....582
4.134.4 Description....582
4.134.5 Re-entrancy....583
4.134.6 Code Example....583
4.135 Function MLIB_ConvertPU_FLTF16....583
4.135.1 Declaration....583
4.135.2 Arguments....583
4.135.3 Return....583

4.135.4 Description....584
4.135.5 Re-entrancy....584
4.135.6 Code Example....584
4.136 Function MLIB_ConvertPU_FLTF32....585
4.136.1 Declaration....585
4.136.2 Arguments....585
4.136.3 Return....585
4.136.4 Description....585
4.136.5 Re-entrancy....585
4.136.6 Code Example....586
4.137 Function MLIB_Div_F32....586
4.137.1 Declaration....586
4.137.2 Arguments....586
4.137.3 Return....586
4.137.4 Description....587
4.137.5 Re-entrancy....587
4.137.6 Code Example....587
4.138 Function MLIB_Div_F16....588
4.138.1 Declaration....588
4.138.2 Arguments....588
4.138.3 Return....588
4.138.4 Description....588
4.138.5 Re-entrancy....589
4.138.6 Code Example....589
4.139 Function MLIB_Div_FLT....590
4.139.1 Declaration....590
4.139.2 Arguments....590
4.139.3 Return....590
4.139.4 Description....590

4.139.5 Re-entrancy....591
4.139.6 Code Example....591
4.140 Function MLIB_DivSat_F32....591
4.140.1 Declaration....591
4.140.2 Arguments....592
4.140.3 Return....592
4.140.4 Description....592
4.140.5 Re-entrancy....592
4.140.6 Code Example....592
4.141 Function MLIB_DivSat_F16....593
4.141.1 Declaration....593
4.141.2 Arguments....593
4.141.3 Return....593
4.141.4 Description....594
4.141.5 Re-entrancy....594
4.141.6 Code Example....594
4.142 Function MLIB_Mac_F32....595
4.142.1 Declaration....595
4.142.2 Arguments....595
4.142.3 Return....595
4.142.4 Description....595
4.142.5 Re-entrancy....596
4.142.6 Code Example....596
4.142.7 Re-entrancy....597
4.142.8 Code Example....597
4.143 Function MLIB_Mac_F32F16F16....598
4.143.1 Declaration....598
4.143.2 Arguments....598
4.143.3 Return....598

4.143.4 Description....598
4.143.5 Re-entrancy....599
4.143.6 Code Example....599
4.143.7 Re-entrancy....600
4.143.8 Code Example....600
4.144 Function MLIB_Mac_F16....601
4.144.1 Declaration....601
4.144.2 Arguments....601
4.144.3 Return....601
4.144.4 Description....601
4.144.5 Re-entrancy....602
4.144.6 Code Example....602
4.144.7 Re-entrancy....603
4.144.8 Code Example....603
4.145 Function MLIB_Mac_FLT....604
4.145.1 Declaration....604
4.145.2 Arguments....604
4.145.3 Return....604
4.145.4 Description....604
4.145.5 Re-entrancy....605
4.145.6 Code Example....605
4.145.7 Re-entrancy....606
4.145.8 Code Example....606
4.146 Function MLIB_MacSat_F32....607
4.146.1 Declaration....607
4.146.2 Arguments....607
4.146.3 Return....607
4.146.4 Description....607
4.146.5 Re-entrancy....608

4.146.6 Code Example....608
4.147 Function MLIB_MacSat_F32F16F16....609
4.147.1 Declaration....609
4.147.2 Arguments....609
4.147.3 Return....609
4.147.4 Description....609
4.147.5 Re-entrancy....610
4.147.6 Code Example....610
4.148 Function MLIB_MacSat_F16....610
4.148.1 Declaration....610
4.148.2 Arguments....610
4.148.3 Return....611
4.148.4 Description....611
4.148.5 Re-entrancy....611
4.148.6 Code Example....611
4.149 Function MLIB_Mul_F32....612
4.149.1 Declaration....612
4.149.2 Arguments....612
4.149.3 Return....613
4.149.4 Description....613
4.149.5 Re-entrancy....613
4.149.6 Code Example....613
4.149.7 Re-entrancy....614
4.149.8 Code Example....614
4.150 Function MLIB_Mul_F32F16F16....615
4.150.1 Declaration....615
4.150.2 Arguments....615
4.150.3 Return....615
4.150.4 Description....616

4.150.5 Re-entrancy....616
4.150.6 Code Example....616
4.150.7 Re-entrancy....617
4.150.8 Code Example....617
4.151 Function MLIB_Mul_F16....617
4.151.1 Declaration....617
4.151.2 Arguments....618
4.151.3 Return....618
4.151.4 Description....618
4.151.5 Re-entrancy....618
4.151.6 Code Example....619
4.151.7 Re-entrancy....619
4.151.8 Code Example....619
4.152 Function MLIB_Mul_FLT....620
4.152.1 Declaration....620
4.152.2 Arguments....620
4.152.3 Return....620
4.152.4 Description....621
4.152.5 Re-entrancy....621
4.152.6 Code Example....621
4.152.7 Re-entrancy....622
4.152.8 Code Example....622
4.153 Function MLIB_MulSat_F32....623
4.153.1 Declaration....623
4.153.2 Arguments....623
4.153.3 Return....623
4.153.4 Description....623
4.153.5 Re-entrancy....624
4.153.6 Code Example....624

4.153.7 Re-entrancy....625
4.153.8 Code Example....625
4.154 Function MLIB_MulSat_F32F16F16....625
4.154.1 Declaration....625
4.154.2 Arguments....626
4.154.3 Return....626
4.154.4 Description....626
4.154.5 Re-entrancy....627
4.154.6 Code Example....627
4.154.7 Re-entrancy....627
4.154.8 Code Example....628
4.155 Function MLIB_MulSat_F16....628
4.155.1 Declaration....628
4.155.2 Arguments....628
4.155.3 Return....628
4.155.4 Description....629
4.155.5 Re-entrancy....629
4.155.6 Code Example....629
4.155.7 Re-entrancy....630
4.155.8 Code Example....630
4.156 Function MLIB_Neg_F32....631
4.156.1 Declaration....631
4.156.2 Arguments....631
4.156.3 Return....631
4.156.4 Description....631
4.156.5 Re-entrancy....632
4.156.6 Code Example....632
4.157 Function MLIB_Neg_F16....632
4.157.1 Declaration....632

4.157.2 Arguments....633
4.157.3 Return....633
4.157.4 Description....633
4.157.5 Re-entrancy....633
4.157.6 Code Example....633
4.158 Function MLIB_Neg_FLT....634
4.158.1 Declaration....634
4.158.2 Arguments....634
4.158.3 Return....634
4.158.4 Description....635
4.158.5 Re-entrancy....635
4.158.6 Code Example....635
4.159 Function MLIB_NegSat_F32....636
4.159.1 Declaration....636
4.159.2 Arguments....636
4.159.3 Return....636
4.159.4 Description....636
4.159.5 Re-entrancy....637
4.159.6 Code Example....637
4.160 Function MLIB_NegSat_F16....637
4.160.1 Declaration....637
4.160.2 Arguments....637
4.160.3 Return....638
4.160.4 Description....638
4.160.5 Re-entrancy....638
4.160.6 Code Example....638
4.161 Function MLIB_Norm_F32....639
4.161.1 Declaration....639
4.161.2 Arguments....639

4.161.3 Return....639
4.161.4 Description....640
4.161.5 Re-entrancy....640
4.161.6 Code Example....640
4.162 Function MLIB_Norm_F16....640
4.162.1 Declaration....640
4.162.2 Arguments....641
4.162.3 Return....641
4.162.4 Description....641
4.162.5 Re-entrancy....641
4.162.6 Code Example....641
4.163 Function MLIB_Round_F32....642
4.163.1 Declaration....642
4.163.2 Arguments....642
4.163.3 Return....642
4.163.4 Description....642
4.163.5 Re-entrancy....643
4.163.6 Code Example....643
4.164 Function MLIB_Round_F16....643
4.164.1 Declaration....644
4.164.2 Arguments....644
4.164.3 Return....644
4.164.4 Description....644
4.164.5 Re-entrancy....644
4.164.6 Code Example....645
4.165 Function MLIB_ShBi_F32....645
4.165.1 Declaration....645
4.165.2 Arguments....645
4.165.3 Return....646

4.165.4 Description....646
4.165.5 Re-entrancy....646
4.165.6 Code Example....646
4.166 Function MLIB_ShBi_F16....647
4.166.1 Declaration....647
4.166.2 Arguments....647
4.166.3 Return....647
4.166.4 Description....647
4.166.5 Re-entrancy....647
4.166.6 Code Example....648
4.167 Function MLIB_ShBiSat_F32....648
4.167.1 Declaration....648
4.167.2 Arguments....648
4.167.3 Return....649
4.167.4 Description....649
4.167.5 Re-entrancy....649
4.167.6 Code Example....649
4.168 Function MLIB_ShBiSat_F16....650
4.168.1 Declaration....650
4.168.2 Arguments....650
4.168.3 Return....650
4.168.4 Description....650
4.168.5 Re-entrancy....651
4.168.6 Code Example....651
4.169 Function MLIB_ShL_F32....651
4.169.1 Declaration....652
4.169.2 Arguments....652
4.169.3 Return....652
4.169.4 Description....652

4.169.5 Re-entrancy....652
4.169.6 Code Example....652
4.170 Function MLIB_ShL_F16....653
4.170.1 Declaration....653
4.170.2 Arguments....653
4.170.3 Return....653
4.170.4 Description....654
4.170.5 Re-entrancy....654
4.170.6 Code Example....654
4.171 Function MLIB_ShLSat_F32....655
4.171.1 Declaration....655
4.171.2 Arguments....655
4.171.3 Return....655
4.171.4 Description....655
4.171.5 Re-entrancy....656
4.171.6 Code Example....656
4.172 Function MLIB_ShLSat_F16....656
4.172.1 Declaration....656
4.172.2 Arguments....656
4.172.3 Return....657
4.172.4 Description....657
4.172.5 Re-entrancy....657
4.172.6 Code Example....657
4.173 Function MLIB_ShR_F32....658
4.173.1 Declaration....658
4.173.2 Arguments....658
4.173.3 Return....658
4.173.4 Description....658
4.173.5 Re-entrancy....659

4.173.6 Code Example....659
4.174 Function MLIB_ShR_F16....659
4.174.1 Declaration....659
4.174.2 Arguments....660
4.174.3 Return....660
4.174.4 Description....660
4.174.5 Re-entrancy....660
4.174.6 Code Example....660
4.175 Function MLIB_Sub_F32....661
4.175.1 Declaration....661
4.175.2 Arguments....661
4.175.3 Return....661
4.175.4 Description....661
4.175.5 Re-entrancy....662
4.175.6 Code Example....662
4.175.7 Re-entrancy....663
4.175.8 Code Example....663
4.176 Function MLIB_Sub_F16....664
4.176.1 Declaration....664
4.176.2 Arguments....664
4.176.3 Return....664
4.176.4 Description....664
4.176.5 Re-entrancy....665
4.176.6 Code Example....665
4.176.7 Re-entrancy....666
4.176.8 Code Example....666
4.177 Function MLIB_Sub_FLT....666
4.177.1 Declaration....666
4.177.2 Arguments....666

4.177.3 Return....667
4.177.4 Description....667
4.177.5 Re-entrancy....667
4.177.6 Code Example....668
4.177.7 Re-entrancy....668
4.177.8 Code Example....668
4.178 Function MLIB_SubSat_F32....669
4.178.1 Declaration....669
4.178.2 Arguments....669
4.178.3 Return....669
4.178.4 Description....670
4.178.5 Re-entrancy....670
4.178.6 Code Example....670
4.179 Function MLIB_SubSat_F16....671
4.179.1 Declaration....671
4.179.2 Arguments....671
4.179.3 Return....671
4.179.4 Description....671
4.179.5 Re-entrancy....672
4.179.6 Code Example....672
4.180 Function MLIB_VMac_F32....673
4.180.1 Declaration....673
4.180.2 Arguments....673
4.180.3 Return....673
4.180.4 Description....673
4.180.5 Re-entrancy....674
4.180.6 Code Example....674
4.180.7 Re-entrancy....675
4.180.8 Code Example....675

4.181 Function MLIB_VMac_F32F16F16....676
4.181.1 Declaration....676
4.181.2 Arguments....676
4.181.3 Return....676
4.181.4 Description....677
4.181.5 Re-entrancy....677
4.181.6 Code Example....677
4.181.7 Re-entrancy....678
4.181.8 Code Example....678
4.182 Function MLIB_VMac_F16....679
4.182.1 Declaration....679
4.182.2 Arguments....679
4.182.3 Return....679
4.182.4 Description....679
4.182.5 Re-entrancy....680
4.182.6 Code Example....680
4.182.7 Re-entrancy....681
4.182.8 Code Example....681
4.183 Function MLIB_VMac_FLT....682
4.183.1 Declaration....682
4.183.2 Arguments....682
4.183.3 Return....682
4.183.4 Description....682
4.183.5 Re-entrancy....683
4.183.6 Code Example....683
4.183.7 Re-entrancy....684
4.183.8 Code Example....684
4.184 Function MCLIB_GetVersion....685
4.184.1 Declaration....685

4.184.2 Return....685
4.184.3 Description....685
4.184.4 Reentrancy....685

Chapter 5

5.1 Typedefs Index....687

Chapter 6
Compound Data Types

6.1 GDFLIB_FILTER_IIR1_COEFF_T_F16....692
6.1.1 Description....692
6.1.2 Compound Type Members....692
6.2 GDFLIB_FILTER_IIR1_COEFF_T_F32....692
6.2.1 Description....693
6.2.2 Compound Type Members....693
6.3 GDFLIB_FILTER_IIR1_COEFF_T_FLT....693
6.3.1 Description....693
6.3.2 Compound Type Members....693
6.4 GDFLIB_FILTER_IIR1_T_F16....694
6.4.1 Description....694
6.4.2 Compound Type Members....694
6.5 GDFLIB_FILTER_IIR1_T_F32....694
6.5.1 Description....694
6.5.2 Compound Type Members....694
6.6 GDFLIB_FILTER_IIR1_T_FLT....695
6.6.1 Description....695
6.6.2 Compound Type Members....695
6.7 GDFLIB_FILTER_IIR2_COEFF_T_F16....695
6.7.1 Description....695
6.7.2 Compound Type Members....695

6.8 GDFLIB_FILTER_IIR2_COEFF_T_F32....696
6.8.1 Description....696
6.8.2 Compound Type Members....696
6.9 GDFLIB_FILTER_IIR2_COEFF_T_FLT....696
6.9.1 Description....696
6.9.2 Compound Type Members....697
6.10 GDFLIB_FILTER_IIR2_T_F16....697
6.10.1 Description....697
6.10.2 Compound Type Members....697
6.11 GDFLIB_FILTER_IIR2_T_F32....697
6.11.1 Description....697
6.11.2 Compound Type Members....698
6.12 GDFLIB_FILTER_IIR2_T_FLT....698
6.12.1 Description....698
6.12.2 Compound Type Members....698
6.13 GDFLIB_FILTER_MA_T_F16....698
6.13.1 Description....699
6.13.2 Compound Type Members....699
6.14 GDFLIB_FILTER_MA_T_F32....699
6.14.1 Description....699
6.14.2 Compound Type Members....699
6.15 GDFLIB_FILTER_MA_T_FLT....699
6.15.1 Description....700
6.15.2 Compound Type Members....700
6.16 GDFLIB_FILTERFIR_PARAM_T_F16....700
6.16.1 Description....700
6.16.2 Compound Type Members....700
6.17 GDFLIB_FILTERFIR_PARAM_T_F32....700
6.17.1 Description....701

6.17.2 Compound Type Members....701
6.18 GDFLIB_FILTERFIR_PARAM_T_FLT....701
6.18.1 Description....701
6.18.2 Compound Type Members....701
6.19 GDFLIB_FILTERFIR_STATE_T_F16....701
6.19.1 Description....702
6.19.2 Compound Type Members....702
6.20 GDFLIB_FILTERFIR_STATE_T_F32....702
6.20.1 Description....702
6.20.2 Compound Type Members....702
6.21 GDFLIB_FILTERFIR_STATE_T_FLT....702
6.21.1 Description....703
6.21.2 Compound Type Members....703
6.22 GFLIB_ACOS_T_F16....703
6.22.1 Description....703
6.22.2 Compound Type Members....703
6.23 GFLIB_ACOS_T_F32....703
6.23.1 Description....704
6.23.2 Compound Type Members....704
6.24 GFLIB_ACOS_T_FLT....704
6.24.1 Description....704
6.24.2 Compound Type Members....704
6.25 GFLIB_ACOS_TAYLOR_COEF_T_F16....704
6.25.1 Description....705
6.25.2 Compound Type Members....705
6.26 GFLIB_ACOS_TAYLOR_COEF_T_F32....705
6.26.1 Description....705
6.26.2 Compound Type Members....705

6.27 GFLIB_ASIN_T_F16....705
6.27.1 Description....705
6.27.2 Compound Type Members....706
6.28 GFLIB_ASIN_T_F32....706
6.28.1 Description....706
6.28.2 Compound Type Members....706
6.29 GFLIB_ASIN_T_FLT....706
6.29.1 Description....706
6.29.2 Compound Type Members....707
6.30 GFLIB_ASIN_TAYLOR_COEF_T_F16....707
6.30.1 Description....707
6.30.2 Compound Type Members....707
6.31 GFLIB_ASIN_TAYLOR_COEF_T_F32....707
6.31.1 Description....707
6.31.2 Compound Type Members....707
6.32 GFLIB_ATAN_T_F16....708
6.32.1 Description....708
6.32.2 Compound Type Members....708
6.33 GFLIB_ATAN_T_F32....708
6.33.1 Description....708
6.33.2 Compound Type Members....708
6.34 GFLIB_ATAN_T_FLT....709
6.34.1 Description....709
6.34.2 Compound Type Members....709
6.35 GFLIB_ATAN_TAYLOR_COEF_T_F16....709
6.35.1 Description....709
6.35.2 Compound Type Members....709
6.36 GFLIB_ATAN_TAYLOR_COEF_T_F32....710
6.36.1 Description....710

6.36.2 Compound Type Members....710
6.37 GFLIB_ATANYXSHIFTED_T_F16....710
6.37.1 Description....710
6.37.2 Compound Type Members....710
6.38 GFLIB_ATANYXSHIFTED_T_F32....711
6.38.1 Description....711
6.38.2 Compound Type Members....711
6.39 GFLIB_ATANYXSHIFTED_T_FLT....711
6.39.1 Description....711
6.39.2 Compound Type Members....712
6.40 GFLIB_CONTROLLER_PI_P_T_F16....712
6.40.1 Description....712
6.40.2 Compound Type Members....712
6.41 GFLIB_CONTROLLER_PI_P_T_F32....713
6.41.1 Description....713
6.41.2 Compound Type Members....713
6.42 GFLIB_CONTROLLER_PI_P_T_FLT....713
6.42.1 Description....713
6.42.2 Compound Type Members....713
6.43 GFLIB_CONTROLLER_PI_R_T_F16....714
6.43.1 Description....714
6.43.2 Compound Type Members....714
6.44 GFLIB_CONTROLLER_PI_R_T_F32....714
6.44.1 Description....715
6.44.2 Compound Type Members....715
6.45 GFLIB_CONTROLLER_PI_R_T_FLT....715
6.45.1 Description....715
6.45.2 Compound Type Members....715

6.46 GFLIB_CONTROLLER_PIAW_P_T_F16....716
6.46.1 Description....716
6.46.2 Compound Type Members....716
6.47 GFLIB_CONTROLLER_PIAW_P_T_F32....717
6.47.1 Description....717
6.47.2 Compound Type Members....717
6.48 GFLIB_CONTROLLER_PIAW_P_T_FLT....718
6.48.1 Description....718
6.48.2 Compound Type Members....718
6.49 GFLIB_CONTROLLER_PIAW_R_T_F16....718
6.49.1 Description....718
6.49.2 Compound Type Members....719
6.50 GFLIB_CONTROLLER_PIAW_R_T_F32....719
6.50.1 Description....719
6.50.2 Compound Type Members....719
6.51 GFLIB_CONTROLLER_PIAW_R_T_FLT....720
6.51.1 Description....720
6.51.2 Compound Type Members....720
6.52 GFLIB_COS_T_F16....721
6.52.1 Description....721
6.52.2 Compound Type Members....721
6.53 GFLIB_COS_T_F32....721
6.53.1 Description....721
6.53.2 Compound Type Members....721
6.54 GFLIB_COS_T_FLT....722
6.54.1 Description....722
6.54.2 Compound Type Members....722
6.55 GFLIB_HYST_T_F16....722
6.55.1 Description....722

6.55.2 Compound Type Members....722
6.56 GFLIB_HYST_T_F32....723
6.56.1 Description....723
6.56.2 Compound Type Members....723
6.57 GFLIB_HYST_T_FLT....723
6.57.1 Description....723
6.57.2 Compound Type Members....724
6.58 GFLIB_INTEGRATOR_TR_T_F16....724
6.58.1 Description....724
6.58.2 Compound Type Members....724
6.59 GFLIB_INTEGRATOR_TR_T_F32....724
6.59.1 Description....725
6.59.2 Compound Type Members....725
6.60 GFLIB_INTEGRATOR_TR_T_FLT....725
6.60.1 Description....725
6.60.2 Compound Type Members....725
6.61 GFLIB_LIMIT_T_F16....726
6.61.1 Description....726
6.61.2 Compound Type Members....726
6.62 GFLIB_LIMIT_T_F32....726
6.62.1 Description....726
6.62.2 Compound Type Members....726
6.63 GFLIB_LIMIT_T_FLT....726
6.63.1 Description....727
6.63.2 Compound Type Members....727
6.64 GFLIB_LOWERLIMIT_T_F16....727
6.64.1 Description....727
6.64.2 Compound Type Members....727

6.65 GFLIB_LOWERLIMIT_T_F32....727
6.65.1 Description....727
6.65.2 Compound Type Members....728
6.66 GFLIB_LOWERLIMIT_T_FLT....728
6.66.1 Description....728
6.66.2 Compound Type Members....728
6.67 GFLIB_LUT1D_T_F16....728
6.67.1 Description....728
6.67.2 Compound Type Members....729
6.68 GFLIB_LUT1D_T_F32....729
6.68.1 Description....729
6.68.2 Compound Type Members....729
6.69 GFLIB_LUT1D_T_FLT....729
6.69.1 Description....729
6.69.2 Compound Type Members....730
6.70 GFLIB_LUT2D_T_F16....730
6.70.1 Description....730
6.70.2 Compound Type Members....730
6.71 GFLIB_LUT2D_T_F32....730
6.71.1 Description....731
6.71.2 Compound Type Members....731
6.72 GFLIB_LUT2D_T_FLT....731
6.72.1 Description....731
6.72.2 Compound Type Members....731
6.73 GFLIB_RAMP_T_F16....732
6.73.1 Description....732
6.73.2 Compound Type Members....732
6.74 GFLIB_RAMP_T_F32....732
6.74.1 Description....732

6.74.2 Compound Type Members....732
6.75 GFLIB_RAMP_T_FLT....733
6.75.1 Description....733
6.75.2 Compound Type Members....733
6.76 GFLIB_SIN_T_F16....733
6.76.1 Description....733
6.76.2 Compound Type Members....733
6.77 GFLIB_SIN_T_F32....734
6.77.1 Description....734
6.77.2 Compound Type Members....734
6.78 GFLIB_SIN_T_FLT....734
6.78.1 Description....734
6.78.2 Compound Type Members....734
6.79 GFLIB_TAN_T_F16....735
6.79.1 Description....735
6.79.2 Compound Type Members....735
6.80 GFLIB_TAN_T_F32....735
6.80.1 Description....735
6.80.2 Compound Type Members....736
6.81 GFLIB_TAN_T_FLT....736
6.81.1 Description....736
6.81.2 Compound Type Members....736
6.82 GFLIB_TAN_TAYLOR_COEF_T_F16....736
6.82.1 Description....736
6.82.2 Compound Type Members....736
6.83 GFLIB_TAN_TAYLOR_COEF_T_F32....737
6.83.1 Description....737
6.83.2 Compound Type Members....737

6.84 GFLIB_UPPERLIMIT_T_F16....737
6.84.1 Description....737
6.84.2 Compound Type Members....737
6.85 GFLIB_UPPERLIMIT_T_F32....738
6.85.1 Description....738
6.85.2 Compound Type Members....738
6.86 GFLIB_UPPERLIMIT_T_FLT....738
6.86.1 Description....738
6.86.2 Compound Type Members....738
6.87 GFLIB_VECTORLIMIT_T_F16....739
6.87.1 Description....739
6.87.2 Compound Type Members....739
6.88 GFLIB_VECTORLIMIT_T_F32....739
6.88.1 Description....739
6.88.2 Compound Type Members....739
6.89 GFLIB_VECTORLIMIT_T_FLT....740
6.89.1 Description....740
6.89.2 Compound Type Members....740
6.90 GMCLIB_DECOUPLINGPMSM_T_F16....740
6.90.1 Description....740
6.90.2 Compound Type Members....740
6.91 GMCLIB_DECOUPLINGPMSM_T_F32....741
6.91.1 Description....741
6.91.2 Compound Type Members....741
6.92 GMCLIB_DECOUPLINGPMSM_T_FLT....741
6.92.1 Description....741
6.92.2 Compound Type Members....742
6.93 GMCLIB_ELIMDCBUSRIP_T_F16....742
6.93.1 Description....742

6.93.2 Compound Type Members....742
6.94 GMCLIB_ELIMDCBUSRIP_T_F32....742
6.94.1 Description....742
6.94.2 Compound Type Members....743
6.95 GMCLIB_ELIMDCBUSRIP_T_FLT....743
6.95.1 Description....743
6.95.2 Compound Type Members....743
6.96 MCLIB_VERSION_T....743
6.96.1 Description....743
6.96.2 Compound Type Members....744
6.97 SWLIBS_2Syst_F16....744
6.97.1 Description....744
6.97.2 Compound Type Members....744
6.98 SWLIBS_2Syst_F32....744
6.98.1 Description....744
6.98.2 Compound Type Members....744
6.99 SWLIBS_2Syst_FLT....745
6.99.1 Description....745
6.99.2 Compound Type Members....745
6.100 SWLIBS_3Syst_F16....745
6.100.1 Description....745
6.100.2 Compound Type Members....745
6.101 SWLIBS_3Syst_F32....746
6.101.1 Description....746
6.101.2 Compound Type Members....746
6.102 SWLIBS_3Syst_FLT....746
6.102.1 Description....746
6.102.2 Compound Type Members....746

Chapter 7

7.1 Macro Definitions....749
7.1.1 Macro Definitions Overview....749

Chapter 8
Macro References

8.1 Define GDFLIB_FilterFIRInit....763
8.1.1 Macro Definition....763
8.1.2 Description....763
8.2 Define GDFLIB_FilterFIR....763
8.2.1 Macro Definition....763
8.2.2 Description....763
8.3 Define GDFLIB_FILTERFIR_PARAM_T....764
8.3.1 Macro Definition....764
8.3.2 Description....764
8.4 Define GDFLIB_FILTERFIR_PARAM_T....764
8.4.1 Macro Definition....764
8.4.2 Description....764
8.5 Define GDFLIB_FILTERFIR_PARAM_T....764
8.5.1 Macro Definition....765
8.5.2 Description....765
8.6 Define GDFLIB_FILTERFIR_STATE_T....765
8.6.1 Macro Definition....765
8.6.2 Description....765
8.7 Define GDFLIB_FILTERFIR_STATE_T....765
8.7.1 Macro Definition....766
8.7.2 Description....766
8.8 Define GDFLIB_FILTERFIR_STATE_T....765
8.8.1 Macro Definition....766
8.8.2 Description....766

8.9 Define GDFLIB_FilterIIR1Init....767
8.9.1 Macro Definition....767
8.9.2 Description....767
8.10 Define GDFLIB_FilterIIR1....767
8.10.1 Macro Definition....767
8.10.2 Description....767
8.11 Define GDFLIB_FILTER_IIR1_T....767
8.11.1 Macro Definition....768
8.11.2 Description....768
8.12 Define GDFLIB_FILTER_IIR1_T....767
8.12.1 Macro Definition....768
8.12.2 Description....768
8.13 Define GDFLIB_FILTER_IIR1_T....767
8.13.1 Macro Definition....769
8.13.2 Description....769
8.14 Define GDFLIB_FILTER_IIR1_DEFAULT....769
8.14.1 Macro Definition....769
8.14.2 Description....769
8.15 Define GDFLIB_FILTER_IIR1_DEFAULT....769
8.15.1 Macro Definition....770
8.15.2 Description....770
8.16 Define GDFLIB_FILTER_IIR1_DEFAULT....769
8.16.1 Macro Definition....770
8.16.2 Description....770
8.17 Define GDFLIB_FILTER_IIR1_DEFAULT_F32....771
8.17.1 Macro Definition....771
8.17.2 Description....771
8.18 Define GDFLIB_FILTER_IIR1_DEFAULT_F16....771
8.18.1 Macro Definition....771

8.18.2 Description....771
8.19 Define GDFLIB_FILTER_IIR1_DEFAULT_FLT....771
8.19.1 Macro Definition....772
8.19.2 Description....772
8.20 Define GDFLIB_FilterIIR2Init....772
8.20.1 Macro Definition....772
8.20.2 Description....772
8.21 Define GDFLIB_FilterIIR2....772
8.21.1 Macro Definition....772
8.21.2 Description....772
8.22 Define GDFLIB_FILTER_IIR2_T....773
8.22.1 Macro Definition....773
8.22.2 Description....773
8.23 Define GDFLIB_FILTER_IIR2_T....773
8.23.1 Macro Definition....773
8.23.2 Description....773
8.24 Define GDFLIB_FILTER_IIR2_T....773
8.24.1 Macro Definition....774
8.24.2 Description....774
8.25 Define GDFLIB_FILTER_IIR2_DEFAULT....774
8.25.1 Macro Definition....774
8.25.2 Description....774
8.26 Define GDFLIB_FILTER_IIR2_DEFAULT....774
8.26.1 Macro Definition....775
8.26.2 Description....775
8.27 Define GDFLIB_FILTER_IIR2_DEFAULT....774
8.27.1 Macro Definition....776
8.27.2 Description....776

8.28 Define GDFLIB_FILTER_IIR2_DEFAULT_F32....776
8.28.1 Macro Definition....776
8.28.2 Description....776
8.29 Define GDFLIB_FILTER_IIR2_DEFAULT_F16....776
8.29.1 Macro Definition....777
8.29.2 Description....777
8.30 Define GDFLIB_FILTER_IIR2_DEFAULT_FLT....777
8.30.1 Macro Definition....777
8.30.2 Description....777
8.31 Define GDFLIB_FilterMAInit....777
8.31.1 Macro Definition....777
8.31.2 Description....777
8.32 Define GDFLIB_FilterMA....778
8.32.1 Macro Definition....778
8.32.2 Description....778
8.33 Define GDFLIB_FILTER_MA_T....778
8.33.1 Macro Definition....778
8.33.2 Description....778
8.34 Define GDFLIB_FILTER_MA_T....778
8.34.1 Macro Definition....779
8.34.2 Description....779
8.35 Define GDFLIB_FILTER_MA_T....778
8.35.1 Macro Definition....779
8.35.2 Description....779
8.36 Define GDFLIB_FILTER_MA_DEFAULT....779
8.36.1 Macro Definition....780
8.36.2 Description....780
8.37 Define GDFLIB_FILTER_MA_DEFAULT....779
8.37.1 Macro Definition....780

8.37.2 Description....780
8.38 Define GDFLIB_FILTER_MA_DEFAULT....779
8.38.1 Macro Definition....781
8.38.2 Description....781
8.39 Define GDFLIB_FILTER_MA_DEFAULT_F32....781
8.39.1 Macro Definition....781
8.39.2 Description....781
8.40 Define GDFLIB_FILTER_MA_DEFAULT_F16....782
8.40.1 Macro Definition....782
8.40.2 Description....782
8.41 Define GDFLIB_FILTER_MA_DEFAULT_FLT....782
8.41.1 Macro Definition....782
8.41.2 Description....782
8.42 Define GFLIB_Acos....782
8.42.1 Macro Definition....782
8.42.2 Description....783
8.43 Define GFLIB_ACOS_T....783
8.43.1 Macro Definition....783
8.43.2 Description....783
8.44 Define GFLIB_ACOS_T....783
8.44.1 Macro Definition....783
8.44.2 Description....783
8.45 Define GFLIB_ACOS_T....783
8.45.1 Macro Definition....784
8.45.2 Description....784
8.46 Define GFLIB_ACOS_DEFAULT....784
8.46.1 Macro Definition....784
8.46.2 Description....785

8.47 Define GFLIB_ACOS_DEFAULT....784
8.47.1 Macro Definition....785
8.47.2 Description....785
8.48 Define GFLIB_ACOS_DEFAULT....784
8.48.1 Macro Definition....785
8.48.2 Description....786
8.49 Define GFLIB_ACOS_DEFAULT_F32....786
8.49.1 Macro Definition....786
8.49.2 Description....786
8.50 Define GFLIB_ACOS_DEFAULT_F16....786
8.50.1 Macro Definition....786
8.50.2 Description....786
8.51 Define GFLIB_ACOS_DEFAULT_FLT....787
8.51.1 Macro Definition....787
8.51.2 Description....787
8.52 Define GFLIB_Asin....787
8.52.1 Macro Definition....787
8.52.2 Description....787
8.53 Define GFLIB_ASIN_T....787
8.53.1 Macro Definition....787
8.53.2 Description....787
8.54 Define GFLIB_ASIN_T....787
8.54.1 Macro Definition....788
8.54.2 Description....788
8.55 Define GFLIB_ASIN_T....787
8.55.1 Macro Definition....788
8.55.2 Description....789
8.56 Define GFLIB_ASIN_DEFAULT....789
8.56.1 Macro Definition....789

8.56.2 Description....789
8.57 Define GFLIB_ASIN_DEFAULT....789
8.57.1 Macro Definition....789
8.57.2 Description....790
8.58 Define GFLIB_ASIN_DEFAULT....789
8.58.1 Macro Definition....790
8.58.2 Description....790
8.59 Define GFLIB_ASIN_DEFAULT_F32....790
8.59.1 Macro Definition....790
8.59.2 Description....791
8.60 Define GFLIB_ASIN_DEFAULT_F16....791
8.60.1 Macro Definition....791
8.60.2 Description....791
8.61 Define GFLIB_ASIN_DEFAULT_FLT....791
8.61.1 Macro Definition....791
8.61.2 Description....791
8.62 Define GFLIB_Atan....791
8.62.1 Macro Definition....791
8.62.2 Description....792
8.63 Define GFLIB_ATAN_T....792
8.63.1 Macro Definition....792
8.63.2 Description....792
8.64 Define GFLIB_ATAN_T....792
8.64.1 Macro Definition....792
8.64.2 Description....792
8.65 Define GFLIB_ATAN_T....792
8.65.1 Macro Definition....793
8.65.2 Description....793

8.66 Define GFLIB_ATAN_DEFAULT....793
8.66.1 Macro Definition....793
8.66.2 Description....794
8.67 Define GFLIB_ATAN_DEFAULT....793
8.67.1 Macro Definition....794
8.67.2 Description....794
8.68 Define GFLIB_ATAN_DEFAULT....793
8.68.1 Macro Definition....794
8.68.2 Description....795
8.69 Define GFLIB_ATAN_DEFAULT_F32....795
8.69.1 Macro Definition....795
8.69.2 Description....795
8.70 Define GFLIB_ATAN_DEFAULT_F16....795
8.70.1 Macro Definition....795
8.70.2 Description....795
8.71 Define GFLIB_ATAN_DEFAULT_FLT....796
8.71.1 Macro Definition....796
8.71.2 Description....796
8.72 Define GFLIB_AtanYX....796
8.72.1 Macro Definition....796
8.72.2 Description....796
8.73 Define GFLIB_AtanYXShifted....796
8.73.1 Macro Definition....796
8.73.2 Description....797
8.74 Define GFLIB_ATANYXSHIFTED_T....797
8.74.1 Macro Definition....797
8.74.2 Description....797
8.75 Define GFLIB_ATANYXSHIFTED_T....797
8.75.1 Macro Definition....797

8.75.2 Description....797
8.76 Define GFLIB_ATANYXSHIFTED_T....797
8.76.1 Macro Definition....798
8.76.2 Description....798
8.77 Define GFLIB_ControllerPIp....798
8.77.1 Macro Definition....799
8.77.2 Description....799
8.78 Define GFLIB_CONTROLLER_PI_P_T....799
8.78.1 Macro Definition....799
8.78.2 Description....799
8.79 Define GFLIB_CONTROLLER_PI_P_T....799
8.79.1 Macro Definition....800
8.79.2 Description....800
8.80 Define GFLIB_CONTROLLER_PI_P_T....799
8.80.1 Macro Definition....800
8.80.2 Description....800
8.81 Define GFLIB_CONTROLLER_PI_P_DEFAULT....801
8.81.1 Macro Definition....801
8.81.2 Description....801
8.82 Define GFLIB_CONTROLLER_PI_P_DEFAULT....801
8.82.1 Macro Definition....801
8.82.2 Description....802
8.83 Define GFLIB_CONTROLLER_PI_P_DEFAULT....801
8.83.1 Macro Definition....802
8.83.2 Description....802
8.84 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F32....802
8.84.1 Macro Definition....803
8.84.2 Description....803

8.85 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F16....803
8.85.1 Macro Definition....803
8.85.2 Description....803
8.86 Define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT....803
8.86.1 Macro Definition....803
8.86.2 Description....803
8.87 Define GFLIB_ControllerPIpAW....804
8.87.1 Macro Definition....804
8.87.2 Description....804
8.88 Define GFLIB_CONTROLLER_PIAW_P_T....804
8.88.1 Macro Definition....804
8.88.2 Description....804
8.89 Define GFLIB_CONTROLLER_PIAW_P_T....804
8.89.1 Macro Definition....805
8.89.2 Description....805
8.90 Define GFLIB_CONTROLLER_PIAW_P_T....804
8.90.1 Macro Definition....805
8.90.2 Description....805
8.91 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT....806
8.91.1 Macro Definition....806
8.91.2 Description....806
8.92 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT....806
8.92.1 Macro Definition....807
8.92.2 Description....807
8.93 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT....806
8.93.1 Macro Definition....807
8.93.2 Description....807
8.94 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32....808
8.94.1 Macro Definition....808

8.94.2 Description....808
8.95 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16....808
8.95.1 Macro Definition....808
8.95.2 Description....808
8.96 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT....808
8.96.1 Macro Definition....809
8.96.2 Description....809
8.97 Define GFLIB_ControllerPIr....809
8.97.1 Macro Definition....809
8.97.2 Description....809
8.98 Define GFLIB_CONTROLLER_PI_R_T....809
8.98.1 Macro Definition....809
8.98.2 Description....809
8.99 Define GFLIB_CONTROLLER_PI_R_T....809
8.99.1 Macro Definition....810
8.99.2 Description....810
8.100 Define GFLIB_CONTROLLER_PI_R_T....809
8.100.1 Macro Definition....811
8.100.2 Description....811
8.101 Define GFLIB_CONTROLLER_PI_R_DEFAULT....811
8.101.1 Macro Definition....811
8.101.2 Description....811
8.102 Define GFLIB_CONTROLLER_PI_R_DEFAULT....811
8.102.1 Macro Definition....812
8.102.2 Description....812
8.103 Define GFLIB_CONTROLLER_PI_R_DEFAULT....811
8.103.1 Macro Definition....812
8.103.2 Description....813

8.104 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F32....813
8.104.1 Macro Definition....813
8.104.2 Description....813
8.105 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F16....813
8.105.1 Macro Definition....813
8.105.2 Description....813
8.106 Define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT....814
8.106.1 Macro Definition....814
8.106.2 Description....814
8.107 Define GFLIB_ControllerPIrAW....814
8.107.1 Macro Definition....814
8.107.2 Description....814
8.108 Define GFLIB_CONTROLLER_PIAW_R_T....814
8.108.1 Macro Definition....815
8.108.2 Description....815
8.109 Define GFLIB_CONTROLLER_PIAW_R_T....814
8.109.1 Macro Definition....815
8.109.2 Description....815
8.110 Define GFLIB_CONTROLLER_PIAW_R_T....814
8.110.1 Macro Definition....816
8.110.2 Description....816
8.111 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT....816
8.111.1 Macro Definition....816
8.111.2 Description....816
8.112 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT....816
8.112.1 Macro Definition....817
8.112.2 Description....817
8.113 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT....816
8.113.1 Macro Definition....818

8.113.2 Description....818
8.114 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32....818
8.114.1 Macro Definition....818
8.114.2 Description....818
8.115 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16....818
8.115.1 Macro Definition....819
8.115.2 Description....819
8.116 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT....819
8.116.1 Macro Definition....819
8.116.2 Description....819
8.117 Define GFLIB_Cos....819
8.117.1 Macro Definition....819
8.117.2 Description....819
8.118 Define GFLIB_COS_T....820
8.118.1 Macro Definition....820
8.118.2 Description....820
8.119 Define GFLIB_COS_T....820
8.119.1 Macro Definition....820
8.119.2 Description....820
8.120 Define GFLIB_COS_T....820
8.120.1 Macro Definition....821
8.120.2 Description....821
8.121 Define GFLIB_COS_DEFAULT....821
8.121.1 Macro Definition....821
8.121.2 Description....821
8.122 Define GFLIB_COS_DEFAULT....821
8.122.1 Macro Definition....822
8.122.2 Description....822

8.123 Define GFLIB_COS_DEFAULT....821
8.123.1 Macro Definition....822
8.123.2 Description....822
8.124 Define GFLIB_COS_DEFAULT_F32....823
8.124.1 Macro Definition....823
8.124.2 Description....823
8.125 Define GFLIB_COS_DEFAULT_F16....823
8.125.1 Macro Definition....823
8.125.2 Description....823
8.126 Define GFLIB_COS_DEFAULT_FLT....823
8.126.1 Macro Definition....823
8.126.2 Description....823
8.127 Define GFLIB_Hyst....824
8.127.1 Macro Definition....824
8.127.2 Description....824
8.128 Define GFLIB_HYST_T....824
8.128.1 Macro Definition....824
8.128.2 Description....824
8.129 Define GFLIB_HYST_T....824
8.129.1 Macro Definition....825
8.129.2 Description....825
8.130 Define GFLIB_HYST_T....824
8.130.1 Macro Definition....825
8.130.2 Description....825
8.131 Define GFLIB_HYST_DEFAULT....826
8.131.1 Macro Definition....826
8.131.2 Description....826
8.132 Define GFLIB_HYST_DEFAULT....826
8.132.1 Macro Definition....826

8.132.2 Description....826
8.133 Define GFLIB_HYST_DEFAULT....826
8.133.1 Macro Definition....827
8.133.2 Description....827
8.134 Define GFLIB_HYST_DEFAULT_F32....827
8.134.1 Macro Definition....827
8.134.2 Description....827
8.135 Define GFLIB_HYST_DEFAULT_F16....827
8.135.1 Macro Definition....828
8.135.2 Description....828
8.136 Define GFLIB_HYST_DEFAULT_FLT....828
8.136.1 Macro Definition....828
8.136.2 Description....828
8.137 Define GFLIB_IntegratorTR....828
8.137.1 Macro Definition....828
8.137.2 Description....828
8.138 Define GFLIB_INTEGRATOR_TR_T....829
8.138.1 Macro Definition....829
8.138.2 Description....829
8.139 Define GFLIB_INTEGRATOR_TR_T....829
8.139.1 Macro Definition....829
8.139.2 Description....829
8.140 Define GFLIB_INTEGRATOR_TR_T....829
8.140.1 Macro Definition....830
8.140.2 Description....830
8.141 Define GFLIB_INTEGRATOR_TR_DEFAULT....830
8.141.1 Macro Definition....830
8.141.2 Description....831

8.142 Define GFLIB_INTEGRATOR_TR_DEFAULT....830
8.142.1 Macro Definition....831
8.142.2 Description....831
8.143 Define GFLIB_INTEGRATOR_TR_DEFAULT....830
8.143.1 Macro Definition....832
8.143.2 Description....832
8.144 Define GFLIB_INTEGRATOR_TR_DEFAULT_F32....832
8.144.1 Macro Definition....832
8.144.2 Description....832
8.145 Define GFLIB_INTEGRATOR_TR_DEFAULT_F16....832
8.145.1 Macro Definition....833
8.145.2 Description....833
8.146 Define GFLIB_INTEGRATOR_TR_DEFAULT_FLT....833
8.146.1 Macro Definition....833
8.146.2 Description....833
8.147 Define GFLIB_Limit....833
8.147.1 Macro Definition....833
8.147.2 Description....833
8.148 Define GFLIB_LIMIT_T....834
8.148.1 Macro Definition....834
8.148.2 Description....834
8.149 Define GFLIB_LIMIT_T....834
8.149.1 Macro Definition....834
8.149.2 Description....834
8.150 Define GFLIB_LIMIT_T....834
8.150.1 Macro Definition....835
8.150.2 Description....835
8.151 Define GFLIB_LIMIT_DEFAULT....835
8.151.1 Macro Definition....835

8.151.2 Description....835
8.152 Define GFLIB_LIMIT_DEFAULT....835
8.152.1 Macro Definition....836
8.152.2 Description....836
8.153 Define GFLIB_LIMIT_DEFAULT....835
8.153.1 Macro Definition....836
8.153.2 Description....836
8.154 Define GFLIB_LIMIT_DEFAULT_F32....837
8.154.1 Macro Definition....837
8.154.2 Description....837
8.155 Define GFLIB_LIMIT_DEFAULT_F16....837
8.155.1 Macro Definition....837
8.155.2 Description....837
8.156 Define GFLIB_LIMIT_DEFAULT_FLT....837
8.156.1 Macro Definition....837
8.156.2 Description....837
8.157 Define GFLIB_LowerLimit....838
8.157.1 Macro Definition....838
8.157.2 Description....838
8.158 Define GFLIB_LOWERLIMIT_T....838
8.158.1 Macro Definition....838
8.158.2 Description....838
8.159 Define GFLIB_LOWERLIMIT_T....838
8.159.1 Macro Definition....839
8.159.2 Description....839
8.160 Define GFLIB_LOWERLIMIT_T....838
8.160.1 Macro Definition....839
8.160.2 Description....839

8.161 Define GFLIB_LOWERLIMIT_DEFAULT....840
8.161.1 Macro Definition....840
8.161.2 Description....840
8.162 Define GFLIB_LOWERLIMIT_DEFAULT....840
8.162.1 Macro Definition....840
8.162.2 Description....840
8.163 Define GFLIB_LOWERLIMIT_DEFAULT....840
8.163.1 Macro Definition....841
8.163.2 Description....841
8.164 Define GFLIB_LOWERLIMIT_DEFAULT_F32....841
8.164.1 Macro Definition....841
8.164.2 Description....842
8.165 Define GFLIB_LOWERLIMIT_DEFAULT_F16....842
8.165.1 Macro Definition....842
8.165.2 Description....842
8.166 Define GFLIB_LOWERLIMIT_DEFAULT_FLT....842
8.166.1 Macro Definition....842
8.166.2 Description....842
8.167 Define GFLIB_Lut1D....842
8.167.1 Macro Definition....842
8.167.2 Description....843
8.168 Define GFLIB_LUT1D_T....843
8.168.1 Macro Definition....843
8.168.2 Description....843
8.169 Define GFLIB_LUT1D_T....843
8.169.1 Macro Definition....843
8.169.2 Description....843
8.170 Define GFLIB_LUT1D_T....843
8.170.1 Macro Definition....844

8.170.2 Description....844
8.171 Define GFLIB_LUT1D_DEFAULT....844
8.171.1 Macro Definition....844
8.171.2 Description....845
8.172 Define GFLIB_LUT1D_DEFAULT....844
8.172.1 Macro Definition....845
8.172.2 Description....845
8.173 Define GFLIB_LUT1D_DEFAULT....844
8.173.1 Macro Definition....845
8.173.2 Description....846
8.174 Define GFLIB_LUT1D_DEFAULT_F32....846
8.174.1 Macro Definition....846
8.174.2 Description....846
8.175 Define GFLIB_LUT1D_DEFAULT_F16....846
8.175.1 Macro Definition....846
8.175.2 Description....846
8.176 Define GFLIB_LUT1D_DEFAULT_FLT....847
8.176.1 Macro Definition....847
8.176.2 Description....847
8.177 Define GFLIB_Lut2D....847
8.177.1 Macro Definition....847
8.177.2 Description....847
8.178 Define GFLIB_LUT2D_T....847
8.178.1 Macro Definition....847
8.178.2 Description....847
8.179 Define GFLIB_LUT2D_T....847
8.179.1 Macro Definition....848
8.179.2 Description....848

8.180 Define GFLIB_LUT2D_T....847
8.180.1 Macro Definition....848
8.180.2 Description....849
8.181 Define GFLIB_LUT2D_DEFAULT....849
8.181.1 Macro Definition....849
8.181.2 Description....849
8.182 Define GFLIB_LUT2D_DEFAULT....849
8.182.1 Macro Definition....849
8.182.2 Description....850
8.183 Define GFLIB_LUT2D_DEFAULT....849
8.183.1 Macro Definition....850
8.183.2 Description....850
8.184 Define GFLIB_LUT2D_DEFAULT_F32....850
8.184.1 Macro Definition....850
8.184.2 Description....851
8.185 Define GFLIB_LUT2D_DEFAULT_F16....851
8.185.1 Macro Definition....851
8.185.2 Description....851
8.186 Define GFLIB_LUT2D_DEFAULT_FLT....851
8.186.1 Macro Definition....851
8.186.2 Description....851
8.187 Define GFLIB_Ramp....851
8.187.1 Macro Definition....851
8.187.2 Description....852
8.188 Define GFLIB_RAMP_T....852
8.188.1 Macro Definition....852
8.188.2 Description....852
8.189 Define GFLIB_RAMP_T....852
8.189.1 Macro Definition....852

8.189.2 Description....853
8.190 Define GFLIB_RAMP_T....852
8.190.1 Macro Definition....853
8.190.2 Description....853
8.191 Define GFLIB_RAMP_DEFAULT....853
8.191.1 Macro Definition....853
8.191.2 Description....854
8.192 Define GFLIB_RAMP_DEFAULT....853
8.192.1 Macro Definition....854
8.192.2 Description....854
8.193 Define GFLIB_RAMP_DEFAULT....853
8.193.1 Macro Definition....854
8.193.2 Description....855
8.194 Define GFLIB_RAMP_DEFAULT_F32....855
8.194.1 Macro Definition....855
8.194.2 Description....855
8.195 Define GFLIB_RAMP_DEFAULT_F16....855
8.195.1 Macro Definition....855
8.195.2 Description....855
8.196 Define GFLIB_RAMP_DEFAULT_FLT....856
8.196.1 Macro Definition....856
8.196.2 Description....856
8.197 Define GFLIB_Sign....856
8.197.1 Macro Definition....856
8.197.2 Description....856
8.198 Define GFLIB_Sin....856
8.198.1 Macro Definition....856
8.198.2 Description....856

8.199 Define GFLIB_SIN_T....857
8.199.1 Macro Definition....857
8.199.2 Description....857
8.200 Define GFLIB_SIN_T....857
8.200.1 Macro Definition....857
8.200.2 Description....857
8.201 Define GFLIB_SIN_T....857
8.201.1 Macro Definition....858
8.201.2 Description....858
8.202 Define GFLIB_SIN_DEFAULT....858
8.202.1 Macro Definition....858
8.202.2 Description....858
8.203 Define GFLIB_SIN_DEFAULT....858
8.203.1 Macro Definition....859
8.203.2 Description....859
8.204 Define GFLIB_SIN_DEFAULT....858
8.204.1 Macro Definition....859
8.204.2 Description....859
8.205 Define GFLIB_SIN_DEFAULT_F32....860
8.205.1 Macro Definition....860
8.205.2 Description....860
8.206 Define GFLIB_SIN_DEFAULT_F16....860
8.206.1 Macro Definition....860
8.206.2 Description....860
8.207 Define GFLIB_SIN_DEFAULT_FLT....860
8.207.1 Macro Definition....860
8.207.2 Description....861
8.208 Define GFLIB_Sqrt....861
8.208.1 Macro Definition....861

8.208.2 Description....861
8.209 Define GFLIB_TAN_LIMIT_FLT....861
8.209.1 Macro Definition....861
8.209.2 Description....861
8.210 Define GFLIB_Tan....862
8.210.1 Macro Definition....862
8.210.2 Description....862
8.211 Define GFLIB_TAN_T....862
8.211.1 Macro Definition....862
8.211.2 Description....862
8.212 Define GFLIB_TAN_T....862
8.212.1 Macro Definition....863
8.212.2 Description....863
8.213 Define GFLIB_TAN_T....862
8.213.1 Macro Definition....863
8.213.2 Description....863
8.214 Define GFLIB_TAN_DEFAULT....863
8.214.1 Macro Definition....864
8.214.2 Description....864
8.215 Define GFLIB_TAN_DEFAULT....863
8.215.1 Macro Definition....864
8.215.2 Description....864
8.216 Define GFLIB_TAN_DEFAULT....863
8.216.1 Macro Definition....865
8.216.2 Description....865
8.217 Define GFLIB_TAN_DEFAULT_F32....865
8.217.1 Macro Definition....865
8.217.2 Description....865

8.218 Define GFLIB_TAN_DEFAULT_F16....865
8.218.1 Macro Definition....865
8.218.2 Description....866
8.219 Define GFLIB_TAN_DEFAULT_FLT....866
8.219.1 Macro Definition....866
8.219.2 Description....866
8.220 Define GFLIB_UpperLimit....866
8.220.1 Macro Definition....866
8.220.2 Description....866
8.221 Define GFLIB_UPPERLIMIT_T....866
8.221.1 Macro Definition....866
8.221.2 Description....867
8.222 Define GFLIB_UPPERLIMIT_T....866
8.222.1 Macro Definition....867
8.222.2 Description....867
8.223 Define GFLIB_UPPERLIMIT_T....866
8.223.1 Macro Definition....868
8.223.2 Description....868
8.224 Define GFLIB_UPPERLIMIT_DEFAULT....868
8.224.1 Macro Definition....868
8.224.2 Description....868
8.225 Define GFLIB_UPPERLIMIT_DEFAULT....868
8.225.1 Macro Definition....869
8.225.2 Description....869
8.226 Define GFLIB_UPPERLIMIT_DEFAULT....868
8.226.1 Macro Definition....869
8.226.2 Description....869
8.227 Define GFLIB_UPPERLIMIT_DEFAULT_F32....870
8.227.1 Macro Definition....870

8.227.2 Description....870
8.228 Define GFLIB_UPPERLIMIT_DEFAULT_F16....870
8.228.1 Macro Definition....870
8.228.2 Description....870
8.229 Define GFLIB_UPPERLIMIT_DEFAULT_FLT....870
8.229.1 Macro Definition....871
8.229.2 Description....871
8.230 Define GFLIB_VectorLimit....871
8.230.1 Macro Definition....871
8.230.2 Description....871
8.231 Define GFLIB_VECTORLIMIT_T....871
8.231.1 Macro Definition....871
8.231.2 Description....871
8.232 Define GFLIB_VECTORLIMIT_T....871
8.232.1 Macro Definition....872
8.232.2 Description....872
8.233 Define GFLIB_VECTORLIMIT_T....871
8.233.1 Macro Definition....872
8.233.2 Description....873
8.234 Define GFLIB_VECTORLIMIT_DEFAULT....873
8.234.1 Macro Definition....873
8.234.2 Description....873
8.235 Define GFLIB_VECTORLIMIT_DEFAULT....873
8.235.1 Macro Definition....873
8.235.2 Description....874
8.236 Define GFLIB_VECTORLIMIT_DEFAULT....873
8.236.1 Macro Definition....874
8.236.2 Description....874

8.237 Define GFLIB_VECTORLIMIT_DEFAULT_F32....875
8.237.1 Macro Definition....875
8.237.2 Description....875
8.238 Define GFLIB_VECTORLIMIT_DEFAULT_F16....875
8.238.1 Macro Definition....875
8.238.2 Description....875
8.239 Define GFLIB_VECTORLIMIT_DEFAULT_FLT....875
8.239.1 Macro Definition....875
8.239.2 Description....875
8.240 Define GMCLIB_Clark....876
8.240.1 Macro Definition....876
8.240.2 Description....876
8.241 Define GMCLIB_ClarkInv....876
8.241.1 Macro Definition....876
8.241.2 Description....876
8.242 Define GMCLIB_DecouplingPMSM....876
8.242.1 Macro Definition....876
8.242.2 Description....877
8.243 Define GMCLIB_DECOUPLINGPMSM_T....877
8.243.1 Macro Definition....877
8.243.2 Description....877
8.244 Define GMCLIB_DECOUPLINGPMSM_T....877
8.244.1 Macro Definition....877
8.244.2 Description....878
8.245 Define GMCLIB_DECOUPLINGPMSM_T....877
8.245.1 Macro Definition....878
8.245.2 Description....878
8.246 Define GMCLIB_DECOUPLINGPMSM_DEFAULT....879
8.246.1 Macro Definition....879

8.246.2 Description....879
8.247 Define GMCLIB_DECOUPLINGPMSM_DEFAULT....879
8.247.1 Macro Definition....879
8.247.2 Description....879
8.248 Define GMCLIB_DECOUPLINGPMSM_DEFAULT....879
8.248.1 Macro Definition....880
8.248.2 Description....880
8.249 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32....880
8.249.1 Macro Definition....880
8.249.2 Description....881
8.250 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16....881
8.250.1 Macro Definition....881
8.250.2 Description....881
8.251 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT....881
8.251.1 Macro Definition....881
8.251.2 Description....881
8.252 Define GMCLIB_ElimDcBusRip....881
8.252.1 Macro Definition....881
8.252.2 Description....882
8.253 Define GMCLIB_ELIMDCBUSRIP_T....882
8.253.1 Macro Definition....882
8.253.2 Description....882
8.254 Define GMCLIB_ELIMDCBUSRIP_T....882
8.254.1 Macro Definition....882
8.254.2 Description....883
8.255 Define GMCLIB_ELIMDCBUSRIP_T....882
8.255.1 Macro Definition....883
8.255.2 Description....883

8.256 Define GMCLIB_ELIMDCBUSRIP_DEFAULT....884
8.256.1 Macro Definition....884
8.256.2 Description....884
8.257 Define GMCLIB_ELIMDCBUSRIP_DEFAULT....884
8.257.1 Macro Definition....884
8.257.2 Description....884
8.258 Define GMCLIB_ELIMDCBUSRIP_DEFAULT....884
8.258.1 Macro Definition....885
8.258.2 Description....885
8.259 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32....885
8.259.1 Macro Definition....885
8.259.2 Description....886
8.260 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16....886
8.260.1 Macro Definition....886
8.260.2 Description....886
8.261 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT....886
8.261.1 Macro Definition....886
8.261.2 Description....886
8.262 Define GMCLIB_Park....886
8.262.1 Macro Definition....886
8.262.2 Description....887
8.263 Define GMCLIB_ParkInv....887
8.263.1 Macro Definition....887
8.263.2 Description....887
8.264 Define GMCLIB_SvmStd....887
8.264.1 Macro Definition....887
8.264.2 Description....887
8.265 Define MLIB_Abs....888
8.265.1 Macro Definition....888

8.265.2 Description....888
8.266 Define MLIB_AbsSat....888
8.266.1 Macro Definition....888
8.266.2 Description....888
8.267 Define MLIB_Add....888
8.267.1 Macro Definition....888
8.267.2 Description....888
8.268 Define MLIB_AddSat....889
8.268.1 Macro Definition....889
8.268.2 Description....889
8.269 Define MLIB_Convert....889
8.269.1 Macro Definition....889
8.269.2 Description....889
8.270 Define MLIB_ConvertPU....889
8.270.1 Macro Definition....889
8.270.2 Description....890
8.271 Define MLIB_Div....890
8.271.1 Macro Definition....890
8.271.2 Description....890
8.272 Define MLIB_DivSat....890
8.272.1 Macro Definition....890
8.272.2 Description....890
8.273 Define MLIB_Mac....890
8.273.1 Macro Definition....891
8.273.2 Description....891
8.274 Define MLIB_MacSat....891
8.274.1 Macro Definition....891
8.274.2 Description....891

8.275 Define MLIB_Mul....891
8.275.1 Macro Definition....891
8.275.2 Description....891
8.276 Define MLIB_MulSat....892
8.276.1 Macro Definition....892
8.276.2 Description....892
8.277 Define MLIB_Neg....892
8.277.1 Macro Definition....892
8.277.2 Description....892
8.278 Define MLIB_NegSat....892
8.278.1 Macro Definition....892
8.278.2 Description....893
8.279 Define MLIB_Norm....893
8.279.1 Macro Definition....893
8.279.2 Description....893
8.280 Define MLIB_Round....893
8.280.1 Macro Definition....893
8.280.2 Description....893
8.281 Define MLIB_ShBi....894
8.281.1 Macro Definition....894
8.281.2 Description....894
8.282 Define MLIB_ShBiSat....894
8.282.1 Macro Definition....894
8.282.2 Description....894
8.283 Define MLIB_ShL....894
8.283.1 Macro Definition....894
8.283.2 Description....895
8.284 Define MLIB_ShLSat....895
8.284.1 Macro Definition....895

8.284.2 Description....895
8.285 Define MLIB_ShR....895
8.285.1 Macro Definition....895
8.285.2 Description....895
8.286 Define MLIB_Sub....895
8.286.1 Macro Definition....896
8.286.2 Description....896
8.287 Define MLIB_SubSat....896
8.287.1 Macro Definition....896
8.287.2 Description....896
8.288 Define MLIB_VMac....896
8.288.1 Macro Definition....896
8.288.2 Description....896
8.289 Define MCLIB_VERSION....897
8.289.1 Macro Definition....897
8.289.2 Description....897
8.290 Define MCLIB_STD_ON....897
8.290.1 Macro Definition....897
8.290.2 Description....897
8.291 Define MCLIB_STD_OFF....897
8.291.1 Macro Definition....897
8.291.2 Description....897
8.292 Define F32....898
8.292.1 Macro Definition....898
8.292.2 Description....898
8.293 Define F16....898
8.293.1 Macro Definition....898
8.293.2 Description....898

8.294 Define FLT....898
8.294.1 Macro Definition....898
8.294.2 Description....898
8.295 Define MCLIB_DEFAULT_IMPLEMENTATION_F32....899
8.295.1 Macro Definition....899
8.295.2 Description....899
8.296 Define MCLIB_DEFAULT_IMPLEMENTATION_F16....899
8.296.1 Macro Definition....899
8.296.2 Description....899
8.297 Define MCLIB_DEFAULT_IMPLEMENTATION_FLT....899
8.297.1 Macro Definition....899
8.297.2 Description....899
8.298 Define MCLIB_SUPPORT_F32....900
8.298.1 Macro Definition....900
8.298.2 Description....900
8.299 Define MCLIB_SUPPORT_F16....900
8.299.1 Macro Definition....900
8.299.2 Description....900
8.300 Define MCLIB_SUPPORT_FLT....900
8.300.1 Macro Definition....900
8.300.2 Description....901
8.301 Define MCLIB_SUPPORTED_IMPLEMENTATION....901
8.301.1 Macro Definition....901
8.301.2 Description....901
8.302 Define MCLIB_DEFAULT_IMPLEMENTATION....901
8.302.1 Macro Definition....901
8.302.2 Description....901
8.303 Define inline....902
8.303.1 Macro Definition....902

8.303.2 Description....902
8.304 Define SFRACT_MIN....902
8.304.1 Macro Definition....902
8.304.2 Description....902
8.305 Define SFRACT_MAX....902
8.305.1 Macro Definition....902
8.305.2 Description....903
8.306 Define FRACT_MIN....903
8.306.1 Macro Definition....903
8.306.2 Description....903
8.307 Define FRACT_MAX....903
8.307.1 Macro Definition....903
8.307.2 Description....903
8.308 Define FRAC32_0_5....903
8.308.1 Macro Definition....904
8.308.2 Description....904
8.309 Define FRAC16_0_5....904
8.309.1 Macro Definition....904
8.309.2 Description....904
8.310 Define FRAC32_0_25....904
8.310.1 Macro Definition....904
8.310.2 Description....904
8.311 Define FRAC16_0_25....905
8.311.1 Macro Definition....905
8.311.2 Description....905
8.312 Define INT16_MAX....905
8.312.1 Macro Definition....905
8.312.2 Description....905

8.313 Define INT16_MIN....905
8.313.1 Macro Definition....905
8.313.2 Description....906
8.314 Define INT32_MAX....906
8.314.1 Macro Definition....906
8.314.2 Description....906
8.315 Define INT32_MIN....906
8.315.1 Macro Definition....906
8.315.2 Description....906
8.316 Define FLOAT_MIN....907
8.316.1 Macro Definition....907
8.316.2 Description....907
8.317 Define FLOAT_MAX....907
8.317.1 Macro Definition....907
8.317.2 Description....907
8.318 Define INT16TOINT32....907
8.318.1 Macro Definition....907
8.318.2 Description....907
8.319 Define INT32TOINT16....908
8.319.1 Macro Definition....908
8.319.2 Description....908
8.320 Define INT32TOINT64....908
8.320.1 Macro Definition....908
8.320.2 Description....908
8.321 Define INT64TOINT32....908
8.321.1 Macro Definition....908
8.321.2 Description....909
8.322 Define F16TOINT16....909
8.322.1 Macro Definition....909

8.322.2 Description....909
8.323 Define F32TOINT16....909
8.323.1 Macro Definition....909
8.323.2 Description....909
8.324 Define F64TOINT16....910
8.324.1 Macro Definition....910
8.324.2 Description....910
8.325 Define F16TOINT32....910
8.325.1 Macro Definition....910
8.325.2 Description....910
8.326 Define F32TOINT32....910
8.326.1 Macro Definition....910
8.326.2 Description....911
8.327 Define F64TOINT32....911
8.327.1 Macro Definition....911
8.327.2 Description....911
8.328 Define F16TOINT64....911
8.328.1 Macro Definition....911
8.328.2 Description....911
8.329 Define F32TOINT64....911
8.329.1 Macro Definition....912
8.329.2 Description....912
8.330 Define F64TOINT64....912
8.330.1 Macro Definition....912
8.330.2 Description....912
8.331 Define INT16TOF16....912
8.331.1 Macro Definition....912
8.331.2 Description....912

8.332 Define INT16TOF32....913
8.332.1 Macro Definition....913
8.332.2 Description....913
8.333 Define INT32TOF16....913
8.333.1 Macro Definition....913
8.333.2 Description....913
8.334 Define INT32TOF32....913
8.334.1 Macro Definition....913
8.334.2 Description....914
8.335 Define INT64TOF16....914
8.335.1 Macro Definition....914
8.335.2 Description....914
8.336 Define INT64TOF32....914
8.336.1 Macro Definition....914
8.336.2 Description....914
8.337 Define F16_1_DIVBY_SQRT3....914
8.337.1 Macro Definition....915
8.337.2 Description....915
8.338 Define F32_1_DIVBY_SQRT3....915
8.338.1 Macro Definition....915
8.338.2 Description....915
8.339 Define F16_SQRT3_DIVBY_2....915
8.339.1 Macro Definition....915
8.339.2 Description....915
8.340 Define F32_SQRT3_DIVBY_2....916
8.340.1 Macro Definition....916
8.340.2 Description....916
8.341 Define F16_SQRT2_DIVBY_2....916
8.341.1 Macro Definition....916

8.341.2 Description....916
8.342 Define F32_SQRT2_DIVBY_2....916
8.342.1 Macro Definition....917
8.342.2 Description....917
8.343 Define FRAC16....917
8.343.1 Macro Definition....917
8.343.2 Description....917
8.344 Define FRAC32....917
8.344.1 Macro Definition....917
8.344.2 Description....917
8.345 Define FLOAT_DIVBY_SQRT3....918
8.345.1 Macro Definition....918
8.345.2 Description....918
8.346 Define FLOAT_SQRT3_DIVBY_2....918
8.346.1 Macro Definition....918
8.346.2 Description....918
8.347 Define FLOAT_SQRT3_DIVBY_4....918
8.347.1 Macro Definition....918
8.347.2 Description....919
8.348 Define FLOAT_SQRT3_DIVBY_4_CORRECTION....919
8.348.1 Macro Definition....919
8.348.2 Description....919
8.349 Define FLOAT_2_PI....919
8.349.1 Macro Definition....919
8.349.2 Description....919
8.350 Define FLOAT_PI....919
8.350.1 Macro Definition....920
8.350.2 Description....920

8.351 Define FLOAT_PI_DIVBY_2....920
8.351.1 Macro Definition....920
8.351.2 Description....920
8.352 Define FLOAT_PI_SINGLE_CORRECTION....920
8.352.1 Macro Definition....920
8.352.2 Description....920
8.353 Define FLOAT_PI_CORRECTION....921
8.353.1 Macro Definition....921
8.353.2 Description....921
8.354 Define FLOAT_PI_DIVBY_4....921
8.354.1 Macro Definition....921
8.354.2 Description....921
8.355 Define FLOAT_4_DIVBY_PI....921
8.355.1 Macro Definition....921
8.355.2 Description....922
8.356 Define FLOAT_0_5....922
8.356.1 Macro Definition....922
8.356.2 Description....922
8.357 Define FLOAT_PLUS_1....922
8.357.1 Macro Definition....922
8.357.2 Description....922
8.358 Define FLOAT_MINUS_1....922
8.358.1 Macro Definition....923
8.358.2 Description....923
8.359 Define MCLIB_VERSION_DEFAULT....923
8.359.1 Macro Definition....923
8.359.2 Description....923
8.360 Define MCLIB_MCID_SIZE....923
8.360.1 Macro Definition....923

8.360.2 Description....923
8.361 Define MCLIB_MCVERSION_SIZE....924
8.361.1 Macro Definition....924
8.361.2 Description....924
8.362 Define MCLIB_MCIMPLEMENTATION_SIZE....924
8.362.1 Macro Definition....924
8.362.2 Description....924
8.363 Define MCLIB_ID....924
8.363.1 Macro Definition....924
8.363.2 Description....924

Chapter 1

1.1 License Agreement

IMPORTANT: Read the following Freescale Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, you indicate that you accept the terms of this Agreement. You may then install the software.

FREESCALE SOFTWARE LICENSE AGREEMENT: This is a legal agreement between you (either as an individual or as an authorized representative of your employer) and Freescale Inc. It concerns your rights to use this file and any accompanying written materials (the "Software"). In consideration for Freescale allowing you to access the Software, you are agreeing to be bound by the terms of this Agreement. If you do not agree to all of the terms of this Agreement, do not download the Software. If you change your mind later, stop using the Software and delete all copies of the Software in your possession or control. Any copies of the Software that you have already distributed, where permitted, and do not destroy will continue to be governed by this Agreement. Your prior use will also continue to be governed by this Agreement.

LICENSE GRANT: Your rights to use this Software vary depending upon the type of license model you ordered from Freescale. The type of license will be set forth in the Freescale Quotation document which formed the basis of your order or, with respect to evaluation licenses, in the correspondence with Freescale confirming your request for a copy of the Software for evaluation. If you do not remember this information, you may contact the party from whom you obtained the Software or us at www.freescale.com.

Definitions: For purposes of this Agreement the following terms are defined as set forth below: "Authorized System" means Licensee's ECU product containing a Freescale processor or other semiconductor product supplied directly or indirectly from Freescale. "Confidential Information" means any information disclosed by Freescale to the Licensee and, if disclosed in writing or in some other tangible form, that is marked at the time of disclosure as being "Confidential" or "Proprietary" or with words of similar import; provided, that Software provided in source code format will be deemed Confidential Information whether or not identified as such in writing or otherwise. Confidential Information does not include any information that: (a) is, or becomes, publicly known

through no wrongful act on the Licensee's part; (b) is already known to the Licensee, or becomes lawfully known to the Licensee without restriction on disclosure; or (c) is independently developed by the Licensee. "Customer Product Line" means the customer product line specified in the Quotation Document. "Customer Target Project" means the one customer Target Project for one automotive vehicle manufacturer specified in the Quotation Document. "Limited Derivative Works" means Licensee may make derivative works only to the extent the copyrighted material provided by Freescale is not modified. For the avoidance of doubt, Licensee explicitly has the right to: 1) add source code to source code provided by Freescale without modification to the original source code; 2) compile any source code into object code, or 3) integrate the Freescale source code into its ECU without modifying the original source code. "Quotation Document" means the Freescale document offering the goods and/or services which formed the basis of your order. "Target Product" means the Freescale processor or other semiconductor product set forth in the Quotation Document. "Target Product Family" means the Freescale processor family or other semiconductor product family set forth in the Quotation Document.

I. Non-Production License Models: a. Evaluation License Freescale grants to you, free of charge, a personal, non-transferable, non-exclusive, revocable, royalty-free, license to use the Software for ninety (90) consecutive days from the date of your acceptance as indicated by clicking the "I accept" button below, internally, solely for the purpose of performing evaluation and testing of the Software, solely for automotive use, and solely on a Freescale Target Product. This license does not include the right to reproduce, distribute, sell, lease, sublicense or transfer all or any part of the Software, or to use the Software on non-Freescale integrated circuit devices, or to use the Software for any production purposes whatsoever. Freescale reserves all rights not expressly granted in this Agreement. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control. b. Development License: Subject to payment of the fees set forth in the Quotation Document, Freescale grants to you a personal, non-transferable, non-exclusive, revocable, license for two years from the date of your acceptance as indicated by clicking the "I accept" button below, unless a different license term is specified in the Quotation Document ("Term"), (1) to use the Software on one Target Product Family for execution on a maximum of 500 sample Authorized Systems, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Licensed Software (4) to distribute the Software and Limited Derivative Works thereof to automotive customers, and (5) to sublicense to automotive customers the right to use the distributed Software as included within the Authorized System solely for purposes of testing the Authorized System during the Term. Freescale reserves all rights not expressly granted in this Agreement. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this

Agreement, and require that you stop using and delete all copies of the Software in your possession or control. You are solely responsible for systems you design using the Software.

II. Production License Models:

a. Project License: Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you the non-exclusive, non-transferable right (1) to use the Software in the Target Project on the Target Product, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

b. Product Line License: Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family in one Customer Product Line, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

c. Family Multi-Project License: Subject to payment of the fees set forth in the Quotation Document, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law

specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

d. Volume License: Subject to payment of the fees set forth in the Quotation Document and the prior taking of a valid Development License for the same Target Product Family, exclusively in conjunction with Licensee's development and sale of an Authorized System, Freescale grants to you, the non-exclusive, non-transferable right (1) to use the Software in one Target Product Family on a maximum of the number of Authorized Systems set forth Quotation Document, (2) to reproduce the Software, (3) to prepare Limited Derivative Works of the Software, (4) to distribute the Software and Limited Derivative Works thereof as part of an Authorized System, and (5) to sublicense to others the right to use the distributed Software as included within the Authorized System. You must prohibit your sublicensees from translating, reverse engineering, decompiling, or disassembling the Software except to the extent applicable law specifically prohibits such restriction. If you violate any of the terms or restrictions of this Agreement, Freescale may immediately terminate this Agreement, and require that you stop using and delete all copies of the Software in your possession or control.

COPYRIGHT: The Software is licensed to you, not sold. Freescale owns the Software, and United States copyright laws and international treaty provisions protect the Software. Therefore, you must treat the Software like any other copyrighted material (e.g. a book or musical recording). You may not use or copy the Software for any other purpose than what is described in this Agreement. Except as expressly provided herein, Freescale does not grant to you any express or implied rights under any Freescale or third party patents, copyrights, trademarks, or trade secrets. Additionally, you must reproduce and apply any copyright or other proprietary rights notices included on or embedded in the Software to any copies or derivative works made thereof, in whole or in part, if any.

SUPPORT: Except as otherwise agreed by the parties under separate agreement, Freescale is NOT obligated to provide any support, upgrades or new releases of the Software. If you wish, you may contact Freescale and report problems and provide suggestions regarding the Software. Freescale has no obligation whatsoever to respond in any way to such a problem report or suggestion. Freescale may make changes to the Software at any time, without any obligation to notify or provide updated versions of the Software to you.

NO WARRANTY. TO THE MAXIMUM EXTENT PERMITTED BY LAW, FREESCALE EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. YOU ASSUME THE

ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE, OR ANY SYSTEMS YOU DESIGN USING THE SOFTWARE (IF ANY). NOTHING IN THIS AGREEMENT MAY BE CONSTRUED AS A WARRANTY OR REPRESENTATION BY FREESCALE THAT THE SOFTWARE OR ANY DERIVATIVE WORK DEVELOPED WITH OR INCORPORATING THE SOFTWARE WILL BE FREE FROM INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS OF THIRD PARTIES.

INDEMNITY: You agree to fully defend and indemnify Freescale from any and all claims, liabilities, and costs (including reasonable attorney's fees) related to (1) your use (including your sublicensee's use, if permitted) of the Software or (2) your violation of the terms and conditions of this Agreement.

LIMITATION OF LIABILITY. IN NO EVENT WILL FREESCALE BE LIABLE, WHETHER IN CONTRACT, TORT, OR OTHERWISE, FOR ANY INCIDENTAL, SPECIAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR ANY LOSS OF USE, LOSS OF TIME, INCONVENIENCE, COMMERCIAL LOSS, OR LOST PROFITS, SAVINGS, OR REVENUES TO THE FULL EXTENT SUCH MAY BE DISCLAIMED BY LAW.

COMPLIANCE WITH LAWS; EXPORT RESTRICTIONS: You must use the Software in accordance with all applicable U.S. laws, regulations and statutes. You agree that neither you nor your licensees (if any) intend to or will, directly or indirectly, export or transmit the Software to any country in violation of U.S. export restrictions.

GOVERNMENT USE: Use of the Software and any corresponding documentation, if any, is provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(l) and (2) of the Commercial Computer Software--Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Freescale, Inc., 6501 William Cannon Drive West, Austin, TX, 78735.

HIGH RISK ACTIVITIES: You acknowledge that the Software is not fault tolerant and is not designed, manufactured or intended by Freescale for incorporation into products intended for use or resale in on-line control equipment in hazardous, dangerous to life or potentially life-threatening environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems, in which the failure of products could lead directly to death, personal injury or severe physical or environmental damage ("High Risk Activities"). You specifically represent and warrant that you will not use the Software or any derivative work of the Software for High Risk Activities.

CHOICE OF LAW; VENUE; LIMITATIONS: You agree that the statutes and laws of the United States and the State of Texas, USA, without regard to conflicts of laws principles, will apply to all matters relating to this Agreement or the Software, and you agree that any litigation will be subject to the exclusive jurisdiction of the state or federal courts in Texas, USA. You agree that regardless of any statute or law to the contrary, any claim or cause of action arising out of or related to this Agreement or the Software must be filed within one (1) year after such claim or cause of action arose or be forever barred.

PRODUCT LABELING: You are not authorized to use any Freescale trademarks, brand names, or logos.

ENTIRE AGREEMENT: This Agreement constitutes the entire agreement between you and Freescale regarding the subject matter of this Agreement, and supersedes all prior communications, negotiations, understandings, agreements or representations, either written or oral, if any. This Agreement may only be amended in written form, executed by you and Freescale.

SEVERABILITY: If any provision of this Agreement is held for any reason to be invalid or unenforceable, then the remaining provisions of this Agreement will be unimpaired and, unless a modification or replacement of the invalid or unenforceable provision is further held to deprive you or Freescale of a material benefit, in which case the Agreement will immediately terminate, the invalid or unenforceable provision will be replaced with a provision that is valid and enforceable and that comes closest to the intention underlying the invalid or unenforceable provision.

NO WAIVER: The waiver by Freescale of any breach of any provision of this Agreement will not operate or be construed as a waiver of any other or a subsequent breach of the same or a different provision.

CONFIDENTIAL INFORMATION: The Licensee agrees to retain the Confidential Information in confidence perpetually, with respect to Software in source code form (human readable), or for a period of 3 years from the date of receipt of the Confidential Information, with respect to all other Confidential Information. During this period the Licensee may not disclose Freescale's Confidential Information to any third party except where such disclosure is necessary in order to achieve the purpose of the license because the automotive customer requires disclosure in conjunction with the testing or purchase of an Authorized System and who have executed written agreements obligating them to protect such Confidential Information. During this period Licensee will also limit dissemination of the Confidential Information to its employees or contractors who have a need to know of the Confidential Information and who have executed written agreements obligating them to protect such Confidential Information, and may not use the Freescale's Confidential Information for any purpose not authorized by this Agreement. The Licensee further agrees to use the same degree of care, but no less than a reasonable degree of care, with Freescale's Confidential Information as it would with its own

confidential information. The Licensee may disclose Confidential Information as required by a court or under operation of law or order provided that the Licensee notifies Freescale of such requirement prior to disclosure, that the Licensee discloses only that information required, and that the Licensee allows Freescale the opportunity to object to such court or other legal body requiring such disclosure.

Freescale Automotive Software License Agreement, v1.02, June 2011

Chapter 2
Introduction

The aim of this document is to describe the Embedded Software and Motor Control Libraries for MPXR40xx devices. It describes the components of the library, its behavior and interaction, the API and steps needed to integrate the library into the customer project.

2.1 Architecture Overview

The Embedded Software and Motor Control Libraries for MPXR40xx consists of four sub-libraries, functionally connected as depicted in Figure 2-1.

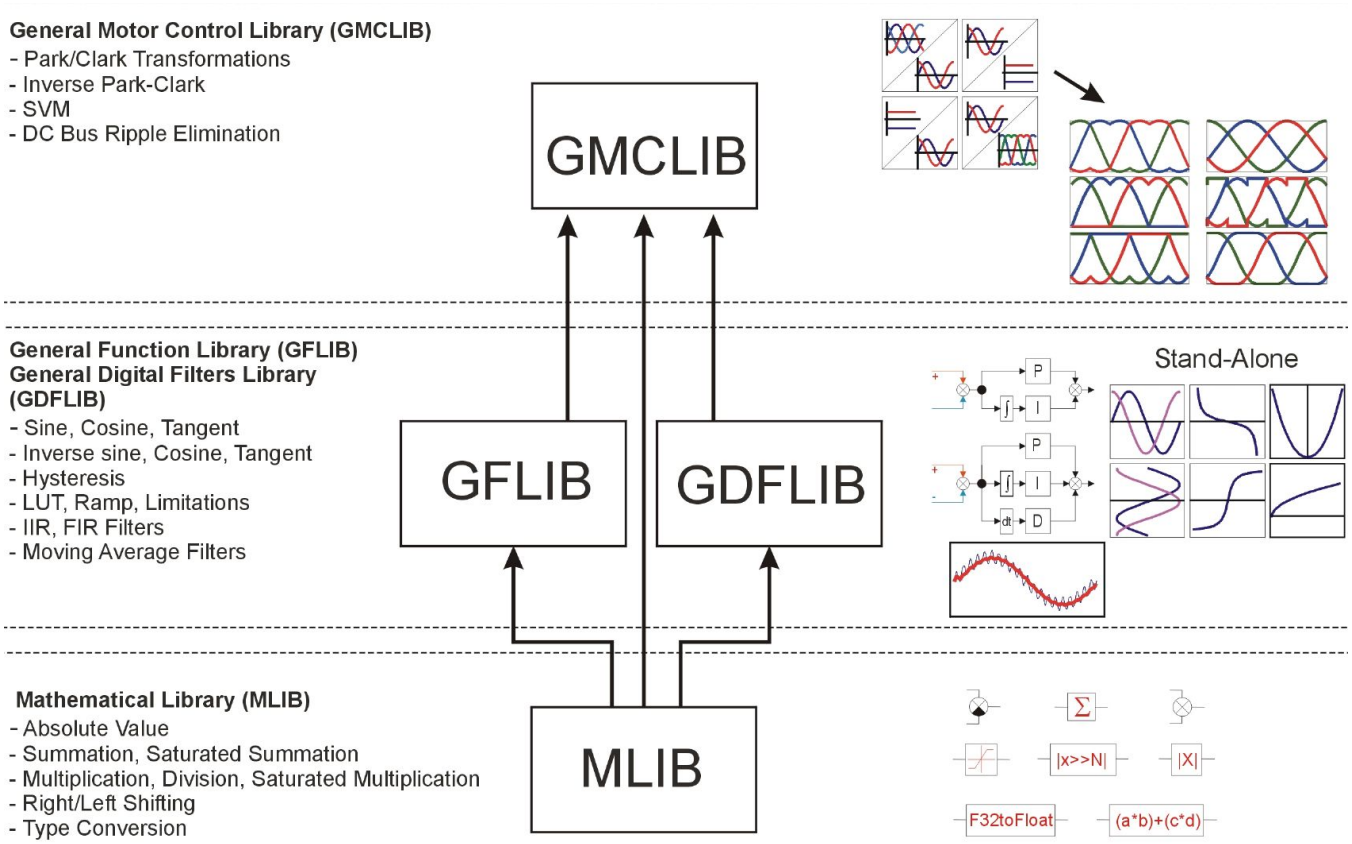

Figure 2-1. MCLib components structure

The Embedded Software and Motor Control Libraries for MPXR40xx sub-libraries are as follows:

- **Mathematical Function Library (MLIB)** - comprising basic mathematical operations such as addition, multiplication, etc.
- **General Function Library (GFLIB)** - comprising basic trigonometric and general math functions such as sine, cosine, tan, hysteresis, limit, etc.
- **General Digital Filters Library (GDFLIB)** - comprising digital IIR and FIR filters designed to be used in a motor control application
- **General Motor Control Library (GMCLIB)** - comprising standard algorithms used for motor control such as Clarke/Park transformations, Space Vector Modulation, etc.

As can be seen in Figure 2-1, the Embedded Software and Motor Control Libraries for MPXR40xx libraries form the layer architecture where all upper libraries utilize the functions from MLIB library. This concept is a key factor for mathematical operations abstractions allowing to support the highly target-optimized variants.

2.2 General Information

The Embedded Software and Motor Control Libraries for MPXR40xx was developed to support three major implementations: fixed-point 32-bit fractional, fixed-point 16-bit fractional, and single precision floating point. With exception of those functions where the mathematical principle limits the input or output values, these values are considered to be in the following limits:

- **Fixed-point 32-bit fractional**: <-1; $1-2^{-31}$> in Q1.31 format and with minimum positive normalized value 2^{-31}.
- **Fixed-point 16-bit fractional**: <-1; $1-2^{-15}$> in Q1.15 format and with minimum positive normalized value 2^{-15}.
- **Single precision floating point**: <-2^{128}; 2^{128}> and with minimum positive normalized value 2^{-128}.

Also those functions which are not relevant for particular implementation, e.g. saturated functions or shifting for single precision floating point implementation, are not delivered with this package. For detailed information about available functions please refer to Function Index. The Embedded Software and Motor Control Libraries for MPXR40xx was tested using two different test methods. To test the precision of each function implementation, the testing based on Matlab reference models was used. This release was tested using the Matlab R2009a version. To test the implementation on the embedded side, the target-in-loop testing was performed on the Embedded Software and Motor Control Libraries for MPXR40xx. This release was tested using the SFIO toolbox version 2.2 and Matlab R2009a version.

2.3 Multiple Implementation Support

In order to allow the user to utilize arbitrary implementation within one user application without any limitations, three different function call methods are supported in the Embedded Software and Motor Control Libraries for MPXR40xx:

- Global configuration option
- Additional parameter option
- API postfix option

Each of these method calls the API postfix function. Thus, for each implementation (32-bit fixed-point, 16-bit fixed point and single precision floating point) only one function is available within the package. This approach is based on ANSI-C99 ISO/IEC 9899:1999 function overloading.

By default the support of all implementations is turned off, thus the error message *"Define at least one supported implementation in SWLIBS_Config.h file."* is displayed during the compilation if no implementation is selected, preventing the user application building. Following are the macro definitions enabling or disabling the implementation support:

- **MCLIB_SUPPORT_F32** for 32-bit fixed-point implementation support selection
- **MCLIB_SUPPORT_F16** for 16-bit fixed-point implementation support selection
- **MCLIB_SUPPORT_FLT** for single precision floating point implementation support selection

These macros are defined in the SWLIBS_Config.h file located in Common directory of the Embedded Software and Motor Control Libraries for MPXR40xx installation destination. To enable the support of each individual implementation the relevant macro definition has to be set to *MCLIB_STD_ON*.

2.3.1 Global Configuration Option

This function call supports the user legacy applications, which were based on older version of Motor Control Library. Prior to any Embedded Software and Motor Control Libraries for MPXR40xx function call using the Global configuration option, the *MCLIB_DEFAULT_IMPLEMENTATION* macro definition has to be setup properly. This macro definition is not defined by default thus the error message "*Define default implementation in SWLIBS_Config.h file.*" is displayed during the compilation, preventing the user application building.

The *MCLIB_DEFAULT_IMPLEMENTATION* macro is defined in the *SWLIBS_Config.h* file located in Common directory of the Embedded Software and Motor Control Libraries for MPXR40xx installation destination. The *MCLIB_DEFAULT_IMPLEMENTATION* can be defined as the one of the following supported implementations:

- **MCLIB_DEFAULT_IMPLEMENTATION_F32** for 32-bit fixed-point implementation
- **MCLIB_DEFAULT_IMPLEMENTATION_F16** for 16-bit fixed-point implementation
- **MCLIB_DEFAULT_IMPLEMENTATION_FLT** for single precision floating point implementation

After proper definition of *MCLIB_DEFAULT_IMPLEMENTATION* macro the Embedded Software and Motor Control Libraries for MPXR40xx functions can be called using standard legacy API convention, e.g. *GFLIB_Sin(x)*.

For example if the *MCLIB_DEFAULT_IMPLEMENTATION* macro definition is set to *MCLIB_DEFAULT_IMPLEMENTATION_F32*, the 32-bit fixed-point implementation of sine function is invoked after the *GFLIB_Sin(x)* API call. Note that all standard legacy API calls will invoke the 32-bit fixed-point implementation in this example.

2.3.2 Additional Parameter Option

In order to support the free selection of used implementation in the user application while keeping the function name same as in standard legacy API approach, the additional parameter option is implemented in the Embedded Software and Motor Control Libraries for MPXR40xx. In this option the additional parameter is used to distinguish which implementation shall be invoked. There are three possible switches selecting the implementation:

- **F32** for 32-bit fixed-point implementation
- **F16** for 16-bit fixed-point implementation
- **FLT** for single precision floating point implementation

For example, if the user application needs to invoke the 16-bit fixed-point implementation of sine function, the *GFLIB_Sin(x, F16)* API call needs to be used. Note that there is possibility to call any implementation of the functions in user application without any limitation.

2.3.3 API Postfix Option

In order to support the free selection of used implementation in the user application while keeping the number of parameters same as in standard legacy API approach, the API postfix option is implemented in the Embedded Software and Motor Control Libraries for MPXR40xx. In this option the implementation postfix is used to distinguish which implementation shall be invoked. There are three possible API postfixes selecting the implementation:

- **F32** for 32-bit fixed-point implementation
- **F16** for 16-bit fixed-point implementation
- **FLT** for single precision floating point implementation

For example, if the user application needs to invoke the single precision floating point implementation of sine function, the *GFLIB_Sin_FLT* API call needs to be used. Note that there is possibility to call any implementation of the functions in user application without any limitation.

2.4 Supported Compilers

The Embedded Software and Motor Control Libraries for MPXR40xx is written in ANSI-C99 ISO/IEC 9899:1999 standard language. The library was built and tested using the following compilers:

1. GreenHills MULTI for PPC version 5.2.4
2. Wind River Compiler version 5.9.1
3. CodeWarrior Development Studio for Freescale MPC55xx, MPC56xx Classic IDE, version 2.9
4. CodeWarrior Development Studio for Freescale MPC55xx, MPC56xx Eclipse IDE, version 10.2

The library is delivered in a library module "*MCLIB_MPXR40xx_v0.91.a*" for the Green Hills compiler. The library module is located in "*C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\ghs*" folder (considering the default installation path).

For the Wind River compiler, the "*MCLIB_MPXR40xx_v0.91.a*" library module is delivered within the installation package. The library module is located in "*C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\diab*" folder (considering the default installation path).

As Freescale is providing two version of CodeWarrior, there are two different sets of library modules available for the CodeWarrior compiler. For the CodeWarrior Classic IDE, two library modules "*MCLIB_MPXR40xx_v0.91.\$suffix\$*" are delivered. Based on representation of the char data type, two library modules are available:

- "*MCLIB_MPXR40xx_v0.91.PPCEABI.V.SC.a*" for signed char data type
- "*MCLIB_MPXR40xx_v0.91.PPCEABI.V.UC.a*" for unsigned char data type

These library modules are located in "*C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\cw*" folder (considering the default installation path).

For the CodeWarrior Eclipse IDE, two library modules "*MCLIB_MPXR40xx_v0.91.\$suffix\$*" are delivered. Based on representation of the char data type, two library modules are available:

- "*MCLIB_MPXR40xx_v0.91_<MCU_Core>_VLE_SC_<FP_Option>.a*" for signed char data type
- "*MCLIB_MPXR40xx_v0.91_<MCU_Core>_VLE_UC_<FP_Option>.a*" for unsigned char data type

These library modules are located in "*C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\cw10x*" folder (considering the default installation path).

Together with the pre-compiled library modules, these are all the necessary header files. The interfaces to the algorithms included in this library have been combined into a single public interface header file for each respective sub-library, i.e. *mlib.h*, *gflib.h*, *gdflib.h*

and *gmclib.h*. This was done to simplify the number of files required for inclusion by application programs. Refer to the specific algorithm sections of this document for details on the software Application Programming Interface (API), definitions and functionality provided for the individual algorithms.

2.5 Matlab Integration

In addition to the Embedded Software and Motor Control Libraries for MPXR40xx library modules, the Bit Accurate Models (BAM) are delivered in the installation package. These models can be used in the Matlab Simulink Toolbox to model the behavior of each function in real implementation. Each model consists of these three files:

1. <libID>_<funcID>.BAM (e.g. *GFLIB_Acos.BAM*): Contains the Bit Accurate Model (BAM)) which can be included in the user Matlab Simulink model and refers to the S-function C file, and the S-function executable file.
2. <libID>_<funcID>_SF.c (e.g. *GFLIB_Acos_SF.c*): The S-function C file that calls a simple legacy function during the simulation.
3. <libID>_<funcID>_SF.mexw32 (e.g. *GFLIB_Acos_SF.mexw32*): Contains the compiled MATLAB S-function executable file created from the function C source file.

All delivered functions are provided with Bit Accurate models in all applicable implementation versions. The user is thus responsible for selecting appropriate version of the Bit Accurate Model. To include the Bit Accurate Model in the user Simulink model, simply copy the BAM into the Simulink model. Note that the BAM parameters need to be properly set up. The structure of the Matlab files delivered with the Embedded Software and Motor Control Libraries for MPXR40xx is depicted in Figure 2-2.

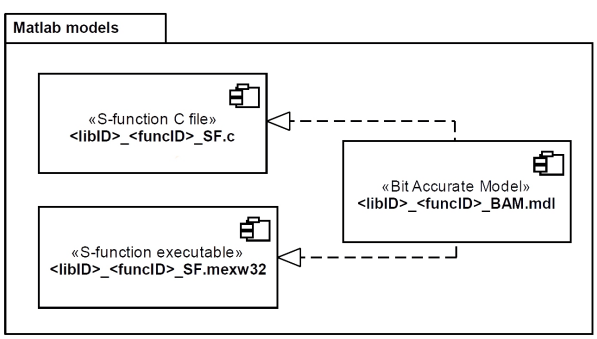

Figure 2-2. MCLib Matlab file structure

An example of Matlab Simulink model utilizing the Embedded Software and Motor Control Libraries for MPXR40xx Bit Accurate Models is depicted in Figure 2-3. Note that this schema is for the illustration only.

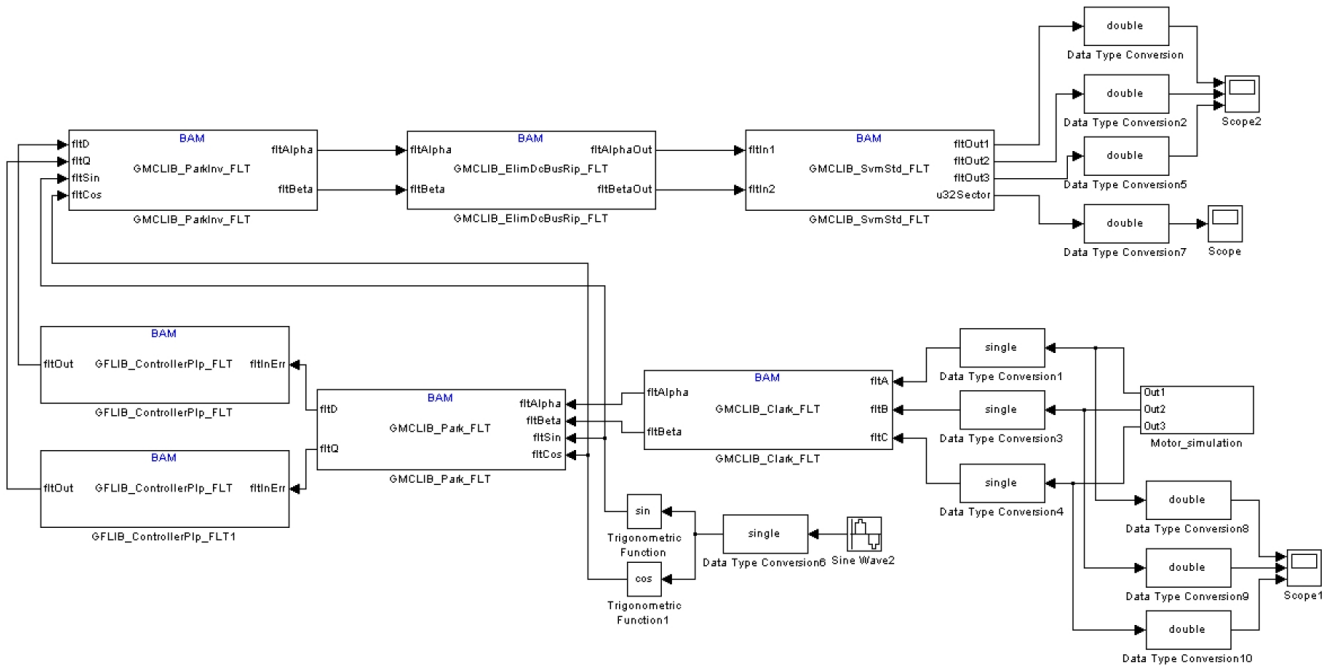

Figure 2-3. Example of Matlab Simulink model utilizing the Bit Accurate Models

2.6 Installation

The Embedded Software and Motor Control Libraries for MPXR40xx is delivered as a single executable file. To install the Embedded Software and Motor Control Libraries for MPXR40xx on a user computer, it is necessary to run the installation file and follow these steps:

1. Accept the license agreement

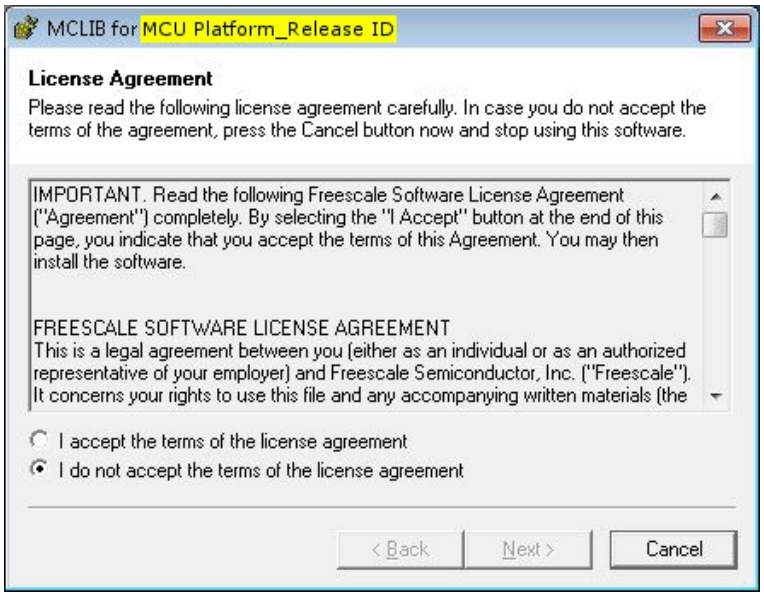

Figure 2-4. MCLib installation - step 1. Highlighted "releaseID" identifies the actual release number, which is MCLIB for MPXR40xx

2. Select the destination directory

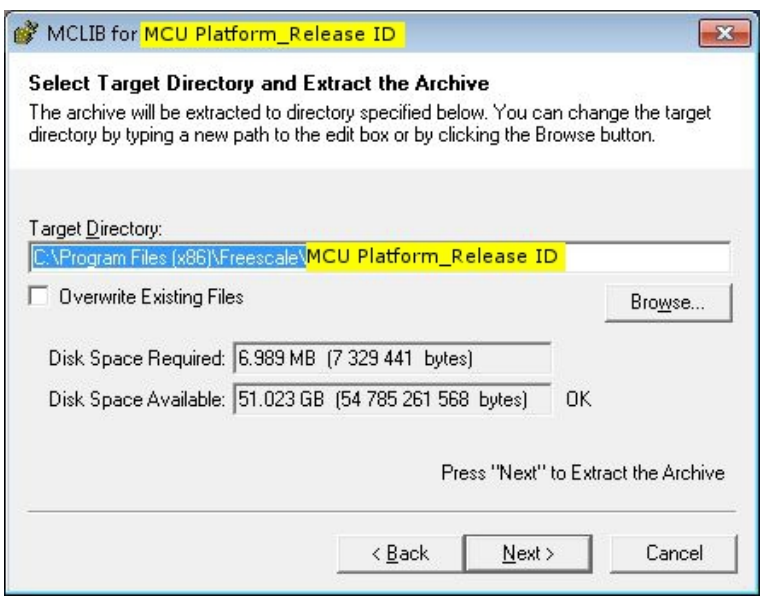

Figure 2-5. MCLib installation - step 2. Highlighted "releaseID" identifies the actual release number, which is MCLIB for MPXR40xx

3. Select Finish to end the installation

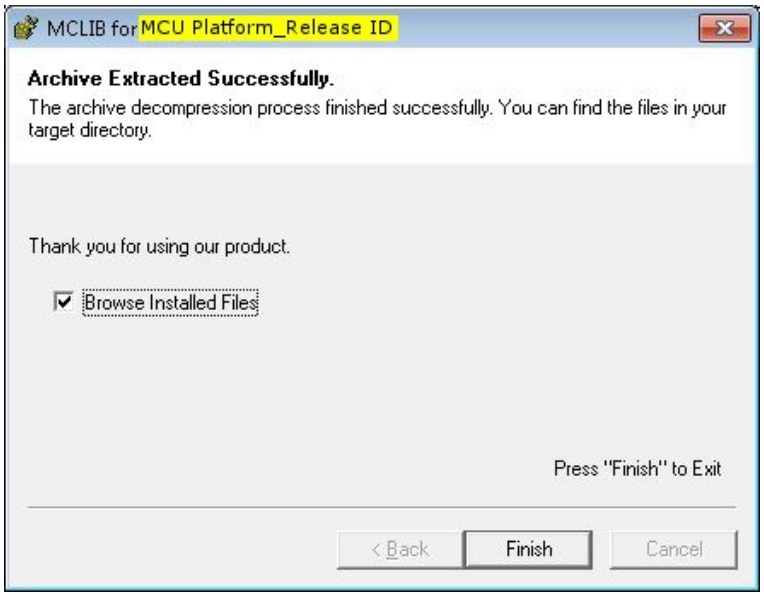

Figure 2-6. MCLib installation - step 3. Highlighted "releaseID" identifies the actual release number, which is MCLIB for MPXR40xx

2.7 Library File Structure

After a successful installation, the Embedded Software and Motor Control Libraries for MPXR40xx is added by default into the "*C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91*" subfolder. This folder will contain other nested subfolders and files required by the Embedded Software and Motor Control Libraries for MPXR40xx, as shown in Figure 2-7.

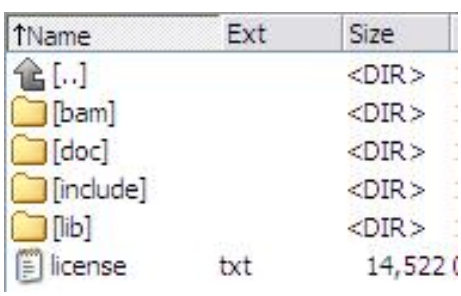

Figure 2-7. MCLib directory structure

A list of the installed directories/files, and their brief description, is given below:

- **BAM** - contains Bit Accurate Models of all the functions for Matlab/Simulink
- **doc** - contains the User Manual
- **include** - contains all the header files, including the master header files of each library to be included in the user application
- **lib** - contains the compiled library file to be included in the user application

In order to integrate the Embedded Software and Motor Control Libraries for MPXR40xx into a new Green Hills compiler based project, the steps described in Library Integration into a Green Hills Multi Development Environment section must be performed.

In order to integrate the Embedded Software and Motor Control Libraries for MPXR40xx into a new Wind River Compiler based project, the steps described in section Library Integration into a Wind River Compiler Environment must be performed.

Similarly, for integration of the Embedded Software and Motor Control Libraries for MPXR40xx into a new classic CodeWarrior based project, the steps described in section Library Integration into a Freescale CodeWarrior Classic IDE must be performed.

And finally, for integration of the Embedded Software and Motor Control Libraries for MPXR40xx into a new Eclipse-based CodeWarrior based project, the steps described in section Library Integration into a Freescale CodeWarrior Eclipse IDE must be performed.

The header files structure of the Embedded Software and Motor Control Libraries for MPXR40xx is depicted in Figure 2-8.

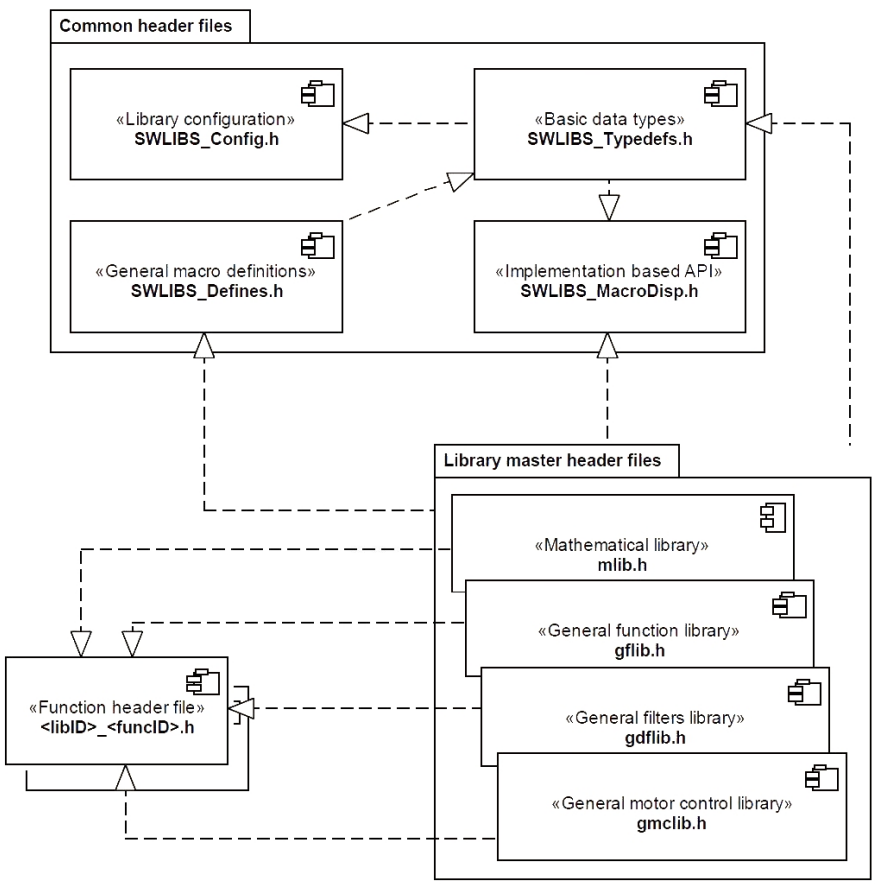

Figure 2-8. MCLib header file structure

2.8 Integration Assumption

In order to successfully integrate the Embedded Software and Motor Control Libraries for MPXR40xx to the customer application, the following assumptions need to be considered:

1. The C-99 language have to be enabled in the user application (please refer to User manual of your compiler to enable this feature).
2. In case the SPE module is available on target MCU, the SPE module have to be enabled even in case no single precision floating point library functions are utilized in the user application.
3. The pre-compiled library of the Embedded Software and Motor Control Libraries for MPXR40xx was compiled using the VLE option, thus the VLE has to be enabled in case the pre-compiled version of the Embedded Software and Motor Control Libraries for MPXR40xx is used in the user application and the VLE option is available on target MCU.
4. In case the floating point unit is available on target MCU, the single precision floating point HW support has to be switched on (please refer to User manual of your compiler to enable this feature).
5. In case the Freescale CodeWarrior is used either in Classic IDE or Eclipse IDE version, the C++ extension has to be switched off.

2.9 Library Integration into a Green Hills Multi Development Environment

The Embedded Software and Motor Control Libraries for MPXR40xx is added into a new Green Hills Multi project using the following steps:

1. Open a new empty C project in the Green Hills Multi IDE. See the Green Hills Multi user manual for instructions.
2. Once you have successfully created and opened a new C project, right click on the project file *.gpj in the GHS Multi Project Manager. Select <*Set Build Options...*> from the pop-up menu, as shown in Figure 2-9

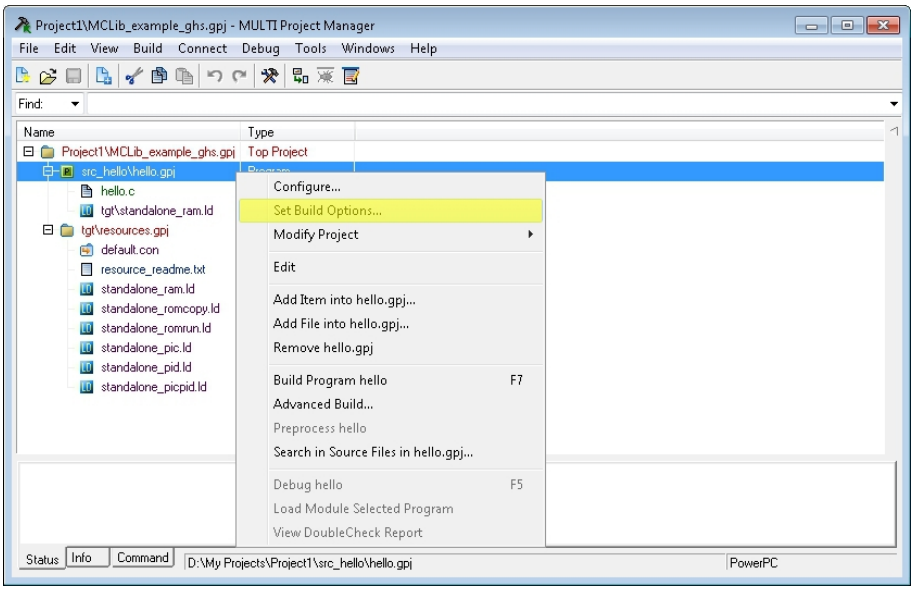

Figure 2-9. Project build options

3. In the <*Basic Options*> tab of the <*Build Options*> window, expand section <Project> and double click on the item <*Include Directories (-I)*>. Here, the directory where the builder should look for all the project header files, including the library header files, shall be specified as shown in Figure 2-10. Considering default settings, the following path shall be added: "*C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\include.* "

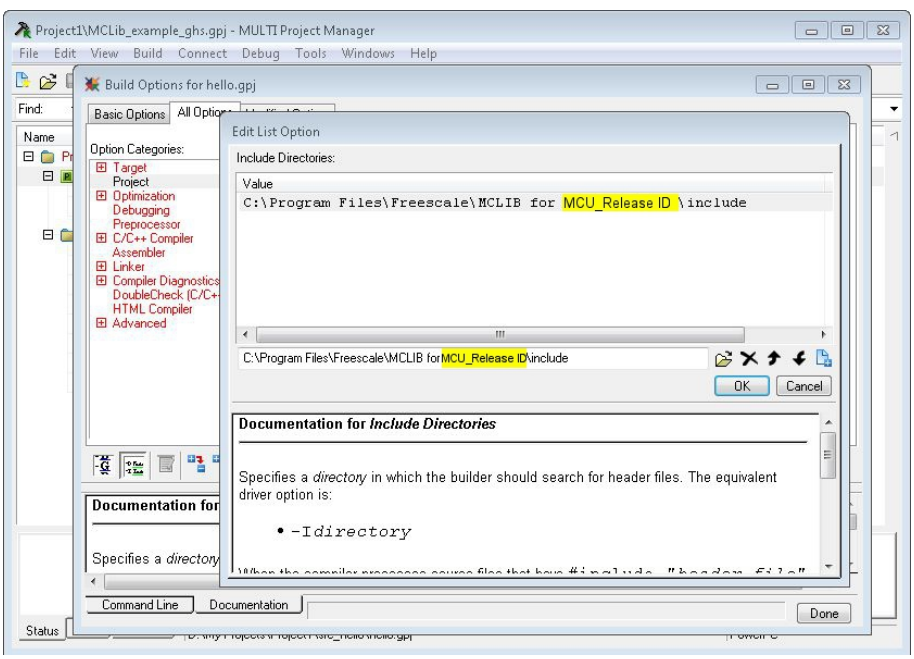

Figure 2-10. Adding a path to the library header files

4. While still in the <*Project*> section, double click on <*Libraries (-l)*> and enter a path and a pre-compiled library object file into the dialogue box, as shown in Figure 2-11.

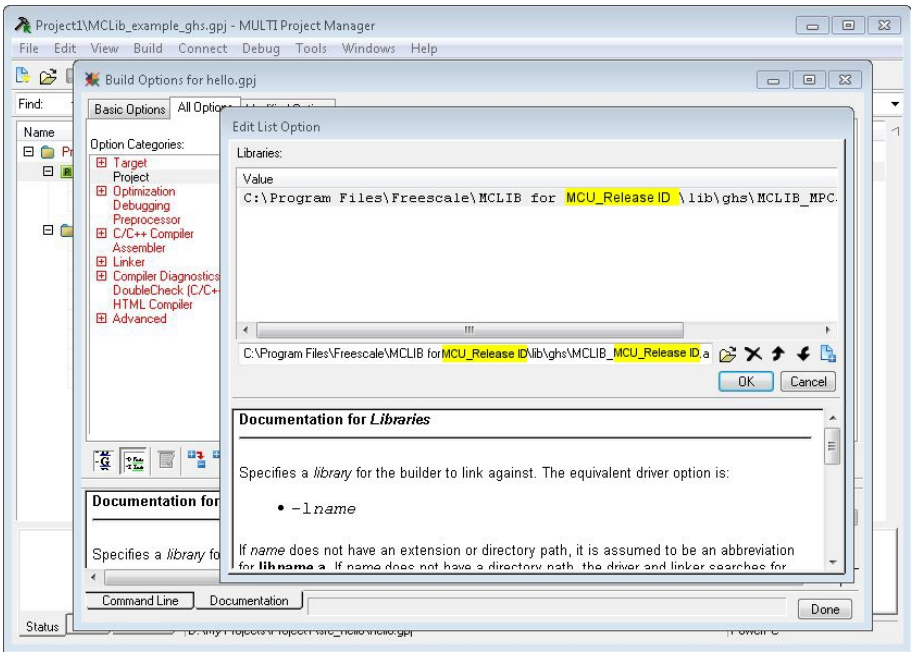

Figure 2-11. Adding a path to the library object files

Considering default settings, you should add *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\ghs\MCLIB_MPXR40xx_v0.91.a"*

5. In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive *\#include "<libID>.h"*, where <libID> can be *gflib* or *gdflib* or *gmclib*, depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Embedded Software and Motor Control Libraries for MPXR40xx integration into any user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "*mlib.h*" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the implementation based API call. Remember that by default there is no default implementation selected in the "*SWLIBS_Config.h*" thus the error message shall be displayed during the compilation requesting the default implementation selection.

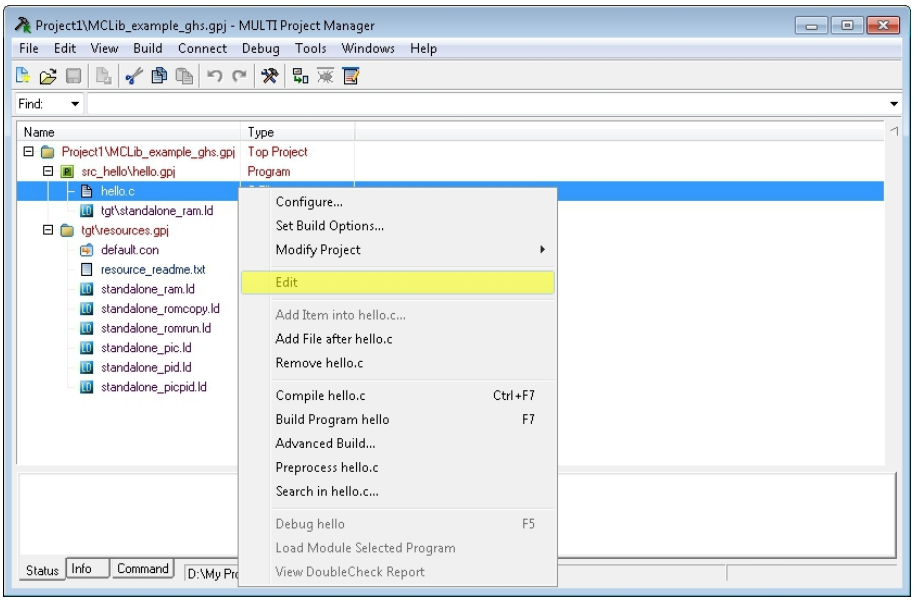

Figure 2-12. Opening the C editor for editing the application source code

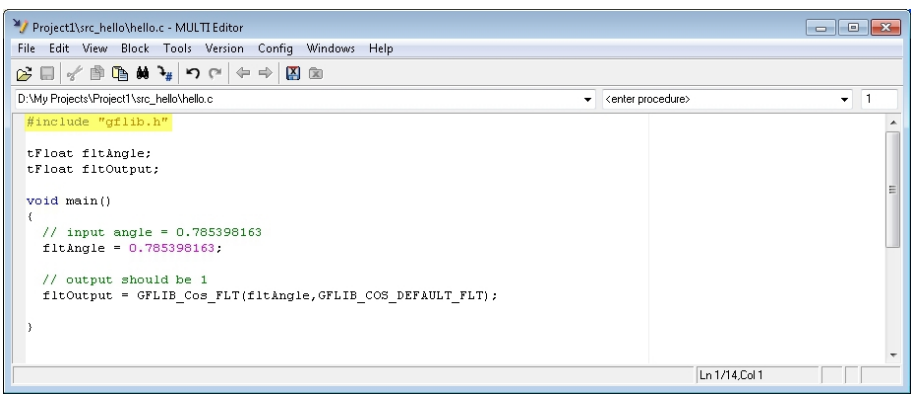

Figure 2-13. Using the pre-processor directive to include library master header files

At this point, the Embedded Software and Motor Control Libraries for MPXR40xx is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

2.10 Library Integration into a Wind River Compiler Environment

As the Wind River Compiler is provided with the Eclipse based IDE called Wind River Workbench, the following instruction will describe the steps needed to add the Embedded Software and Motor Control Libraries for MPXR40xx into a new project. All described steps assume that there is an existing Wind River Workbench project ("*MCLib_example_diab*" name is used in the pictures below).

In order to integrate the Embedded Software and Motor Control Libraries for MPXR40xx into an existing Wind River Workbench project, it is necessary to provide the Wind River Workbench IDE with access paths to the Embedded Software and Motor Control Libraries for MPXR40xx files. The following files shall be added to the user project:

- Library binary files located in the directory *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\diab"* (note: this is the default location and may be modified during library installation)
- Header files located in the directory *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\include"* (note: this is the default location and may be modified during library installation)

To add these files, choose your project in <*Project Explorer*> in the left tab, and from the <*Project*> menu select <*Properties*> as described in Figure 2-14.

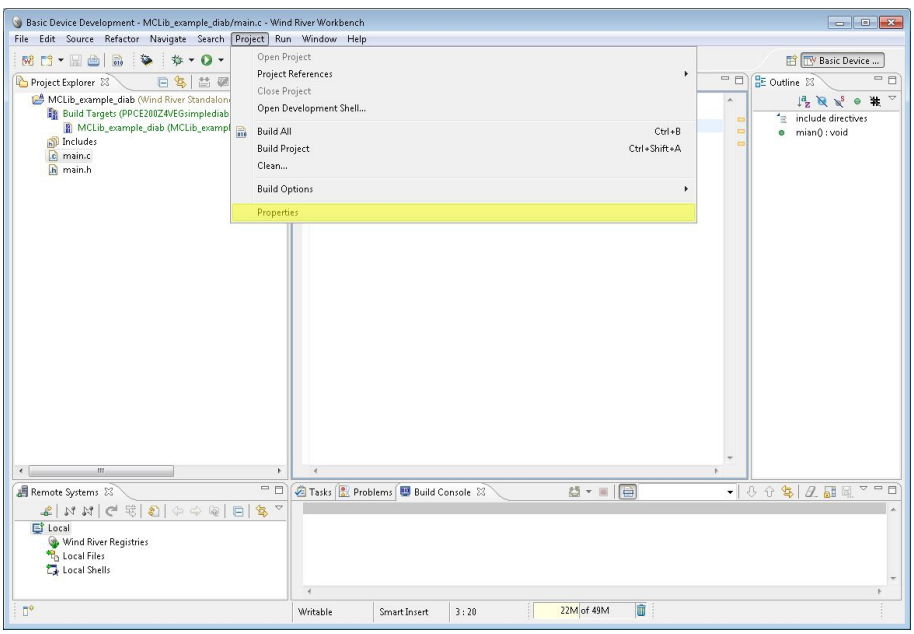

Figure 2-14. Selection of the project property in the Wind River Workbench

In the *Properties* window, select *Build Properties* from the left-hand list and choose the *Build Path* tab as described in Figure 2-15.

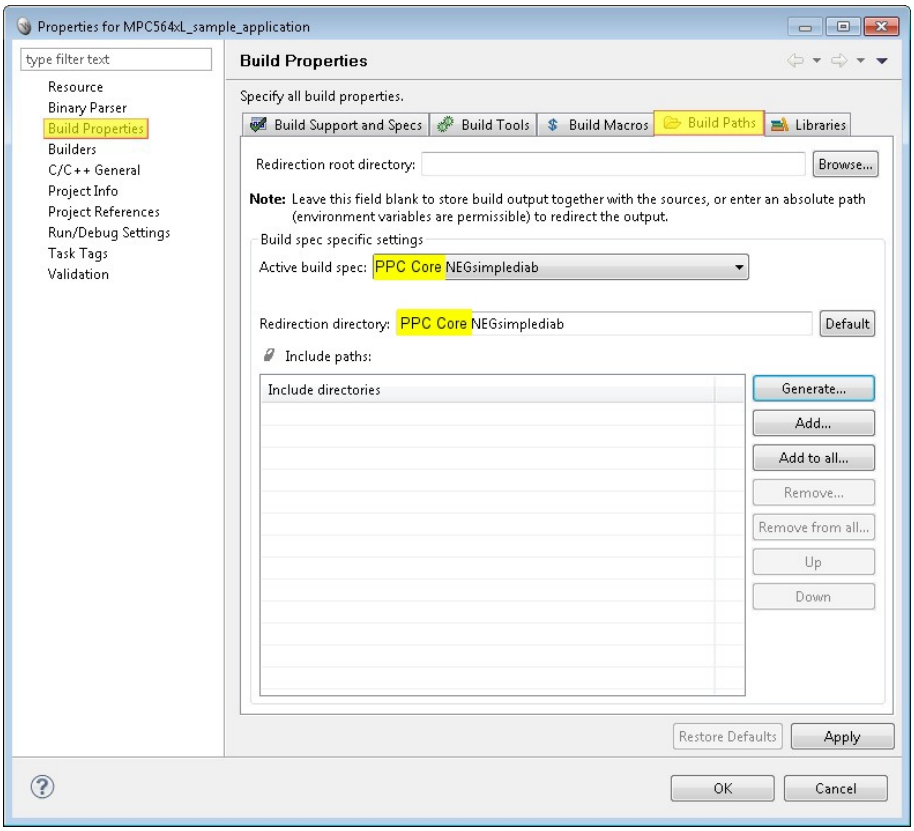

Figure 2-15. Selecting the include paths in the Wind River Workbench

By selecting the <*Add..*> button, add the directories, where the builder should look for all the header files, including the library file. Considering the default settings, the paths shown in Figure 2-16 and Figure 2-17 shall be added.

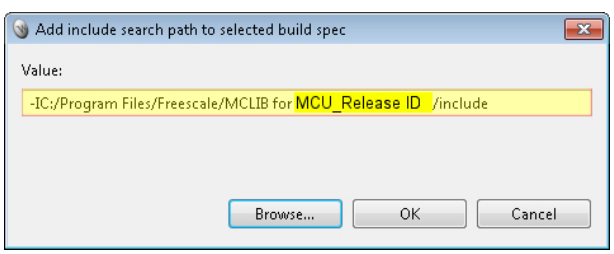

Figure 2-16. Including the header files directory to the Wind River Workbench

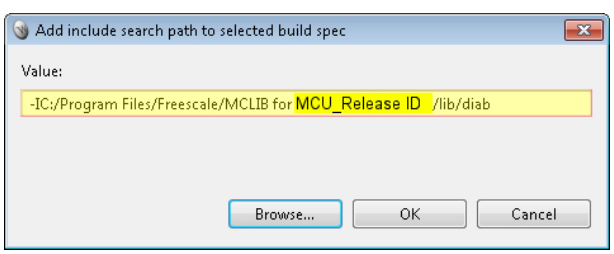

Figure 2-17. Including the library file directory to the Wind River Workbench

In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive \#*include "<libID>.h"*, where <libID> can be *gflib* or *gdflib* or *gmclib*, depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Embedded Software and Motor Control Libraries for MPXR40xx integration into any user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "*mlib.h*" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the implementation based API call. Remember that by default there is no default implementation selected in the "*SWLIBS_Config.h*" thus the error message shall be displayed during the compilation requesting the default implementation selection.

At this point, the Embedded Software and Motor Control Libraries for MPXR40xx is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

2.11 Library Integration into a Freescale CodeWarrior Classic IDE

The release provides two library versions for the CodeWarrior environment. The library versions are different in the compilation options that were set while the library was compiled:

- The library file with the suffix: *.PPCEABI.V.SC.a*, compilation options used: Power Architecture Embedded ABI, VLE instructions set, char type is signed.
- The library file with the suffix: *.PPCEABI.V.UC.a*, compilation options used: Power Architecture EABI, VLE instruction set, char type is unsigned.

The selection of the library version should follow those compilation options set for the CodeWarrior project. If the Embedded Software and Motor Control Libraries for MPXR40xx options and project options are different, warnings would be reported.

In order to use the Embedded Software and Motor Control Libraries for MPXR40xx within a CodeWarrior GUI environment, it is necessary to provide the CodeWarrior GUI with information on access paths and to add the appropriate Embedded Software and Motor Control Libraries for MPXR40xx files to the project. The following files shall be added to the project:

- Library binary files located in the directory *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\cw"* (note: this is the default location and may be modified during library installation)
- Header files located in the directory *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\include"* (note: this is the default location and may be modified during library installation)

An example of the procedure of how to add Embedded Software and Motor Control Libraries for MPXR40xx files to a CodeWarrior project is described in the step-by-step instructions below:

1. Create a new or open an existing CodeWarrior project, for which Embedded Software and Motor Control Libraries for MPXR40xx use is intended.
2. Add the library binary files and the header files to the CodeWarrior project. Files can be added by the use of mouse drag-and-drop or by selecting the <*Add Files*> action from the mouse right button menu in the project window. The CodeWarrior GUI may open a new window with information on new paths added to the project Access Paths list.
3. Now the Embedded Software and Motor Control Libraries for MPXR40xx is ready for use.

After completing the procedure, the CodeWarrior project window may look like that in Figure 2-18.

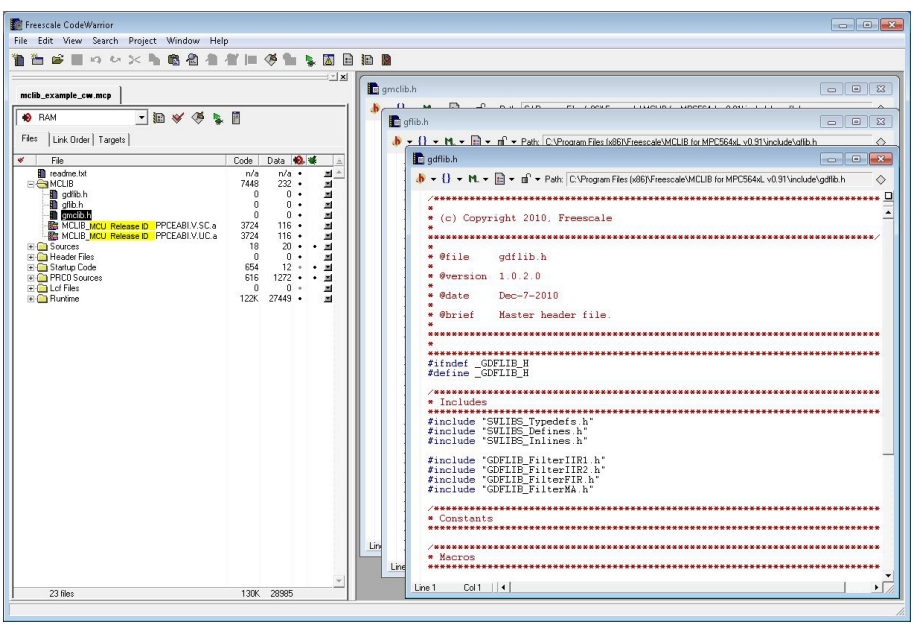

Figure 2-18. MCLib integration into a CodeWarrior project

In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive \#*include "<libID>.h"*, where <libID> can be *gflib* or *gdflib* or *gmclib*, depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Embedded Software and Motor Control Libraries for MPXR40xx integration into any user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "*mlib.h*" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the implementation based API call.

Remember that by default there is no default implementation selected in the "*SWLIBS_Config.h*" thus the error message shall be displayed during the compilation requesting the default implementation selection.

At this point, the Embedded Software and Motor Control Libraries for MPXR40xx is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

2.12 Library Integration into a Freescale CodeWarrior Eclipse IDE

The release provides two library versions for the Eclipse-based CodeWarrior environment. The library versions are different in the compilation options that were set while the library was compiled:

- The library file with the suffix: *<MCU_Core>_VLE_SC_<FP_Option>.a*, compilation options used: Power Architecture Embedded ABI, VLE instructions set, char type is signed.
- The library file with the suffix: *<MCU_Core>_VLE_UC_<FP_Option>.a*, compilation options used: Power Architecture EABI, VLE instruction set, char type is unsigned.

The selection of the library version should follow those compilation options set for the CodeWarrior Eclipse IDE project. If the Embedded Software and Motor Control Libraries for MPXR40xx options and project options are different, warnings would be reported. All described steps assume that there is an existing Eclipse project ("*mclib_example_cw10x*" name is used on below pictures).

In order to use the Embedded Software and Motor Control Libraries for MPXR40xx within a Eclipse-based CodeWarrior GUI environment, it is necessary to provide the Eclipse-based CodeWarrior GUI with access paths to the Embedded Software and Motor Control Libraries for MPXR40xx files. The following files shall be added to the user project:

- Library binary files located in the directory *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\lib\cw10x"* (note: this is the default location and may be modified during library installation)
- Header files located in the directory *"C:\Program Files\Freescale\MCLIB for MPXR40xx v0.91\include"* (note: this is the default location and may be modified during library installation)

To add these files, select your project in the <*Project Explorer*> in the left tab, and from the <*Project*> menu select the <*Properties*> option as described in Figure 2-19.

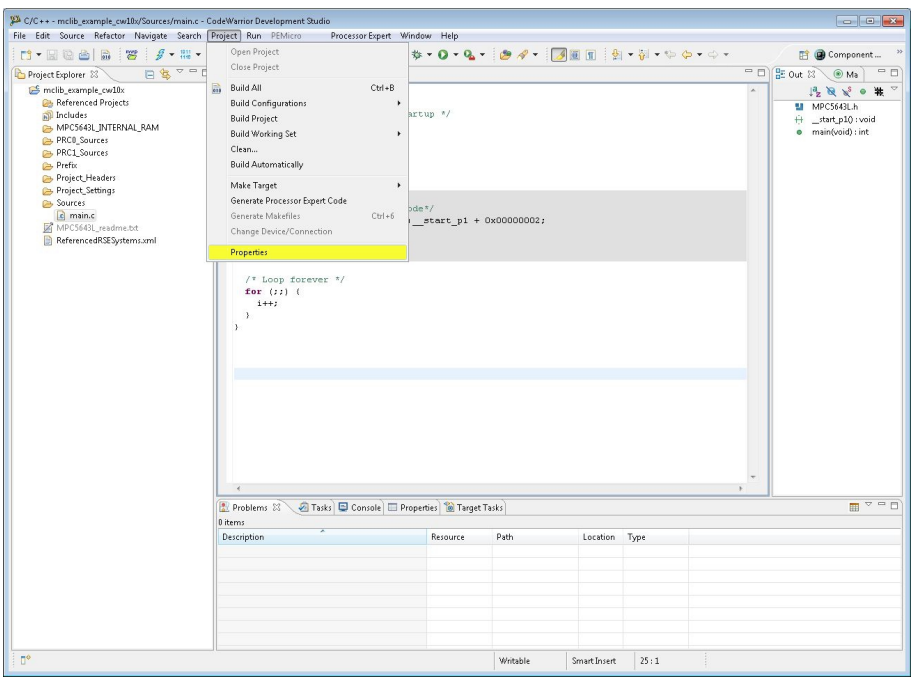

Figure 2-19. Selection of the project property in the CodeWarrior Eclipse IDE

In the <*Properties*> window, select the <*Paths and Symbols*> from the left-hand list and choose the <*Includes*> tab and <*C Source File*> under the <*Languages*> section, as described in Figure 2-20.

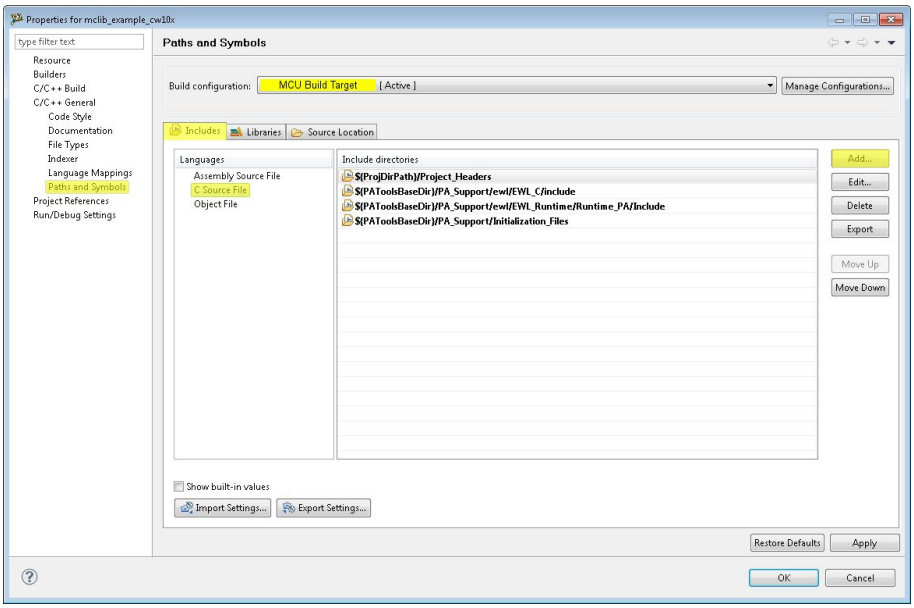

Figure 2-20. Selecting the include paths in the CodeWarrior Eclipse IDE

By selecting the <*Add..*> button, add the directories, where the builder should look for all the header files. Considering the default settings, the path shown in Figure 2-21 shall be added.

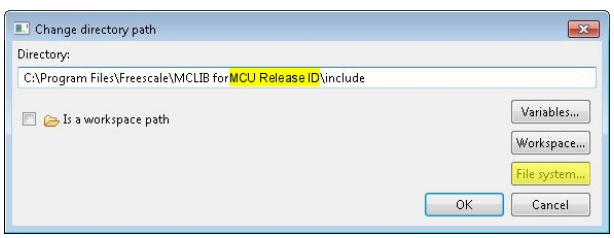

Figure 2-21. Adding the header files path to the CodeWarrior Eclipse IDE project

After adding of the Embedded Software and Motor Control Libraries for MPXR40xx header files path, the library object file shall be add to the Eclipse-based CodeWarrior project. In the <*Properties*> window, select the <*Paths and Symbols*> from the left-hand list and choose the <*Libraries*> tab, as described in Figure 2-22.

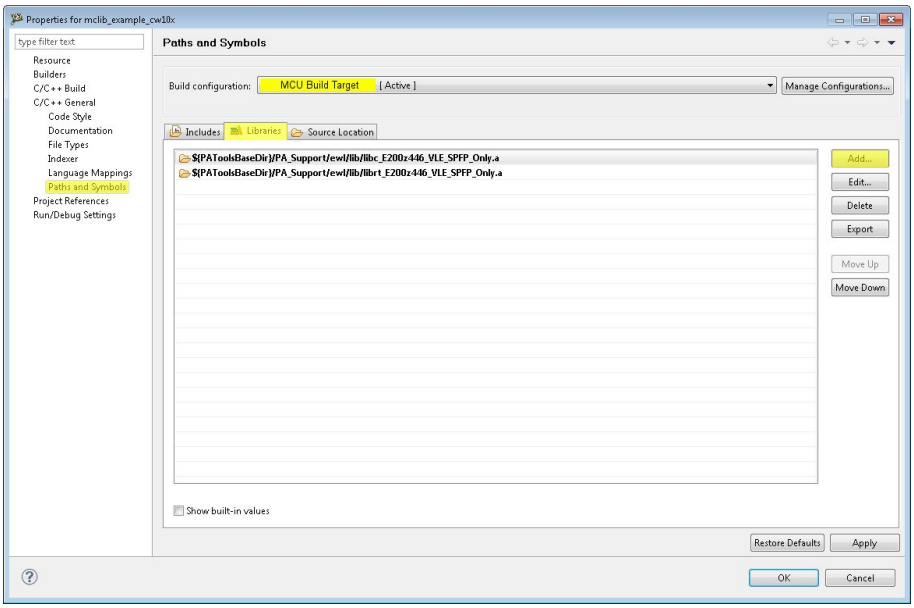

Figure 2-22. Selecting the library object in the CodeWarrior Eclipse IDE

By selecting the <*Add..*> button, add the library object file (consider the project setting of char data type representation). Considering the default settings, the path shown in Figure 2-23 shall be added.

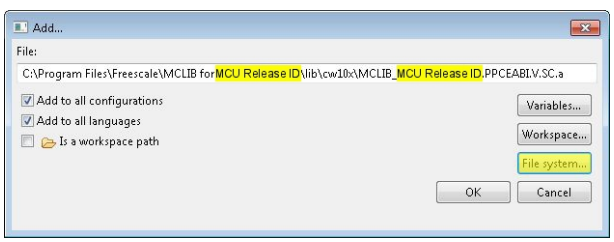

Figure 2-23. Adding the library object file to the CodeWarrior Eclipse IDE project

In order to use the library functions, the library master header files must be included into the application source code. This is done using the pre-processor directive *#include "<libID>.h"*, where <libID> can be *gflib* or *gdflib* or *gmclib*, depending on which library is to be employed.

The master header files contain several additional header files that are needed for the Embedded Software and Motor Control Libraries for MPXR40xx integration into any user application. They include the "*SWLIBS_Typedefs.h*" header file which contains all general purpose data type definitions, the "mlib.h" header file containing all general math functions, the "*SWLIBS_Defines.h*" file containing common macro definitions and the "*SWLIBS_MacroDisp.h*" allowing the implementation based API call.

Remember that by default there is no default implementation selected in the "*SWLIBS_Config.h*" thus the error message shall be displayed during the compilation requesting the default implementation selection.

At this point, the Embedded Software and Motor Control Libraries for MPXR40xx is linked with the user project file, and hence the library functions can be exploited and flawlessly compiled/linked with the user application.

2.13 MC Library Testing

In order to validate the implementation of the Embedded Software and Motor Control Libraries for MPXR40xx, the comparison of results from the MATLAB Reference Model and outputs from the tested library function is used. To ensure the Embedded Software and Motor Control Libraries for MPXR40xx precision, two test methods are used:

- Matlab Simulink Toolbox based testing (refer to MCLib Testing based on the MATLAB Simulink Toolbox } for more details).
- Target-in-loop based testing (refer to MCLib target-in-loop Testing based on the SFIO Toolbox for detailed information).

The Figure 2-24 shows the testing principle:

- **Input vector** represents the test vector which enters simultaneously into the Reference Model and into the Unit Under Test (UUT).
- **Reference Model (RM)** implements the real model of the UUT. For simple functions, the models are a part of the MATLAB Simulink Toolbox. Advanced functions such as filters or controllers had been designed separately.
- By the type of test method used, the **Unit Under Test (UUT)** may be:
  - **Bit Accurate Model (BAM)** - the "C" implementation of the tested function compiled in the MATLAB environment. The compilation result, called the binary MEX-file, is a dynamically-linked subroutine that the MATLAB interpreter can load and execute.
  - **SFIO Model** represents the tested function running directly on the target MCU.

Results from the UUT and Reference Model are saved in the final report, together with the calculated error which is simply the difference between the output value from the Reference Model and the output value from the UUT, recalculated to an equivalent precision. The equivalent precision is different based on the Embedded Software and Motor Control Libraries for MPXR40xx implementation. For the 32-bit fixed-point and single precision floating point implementations the output precision is recalculated to +/-1LSB in 16-bit arithmetic for non-approximation functions and +/-3LSB in 16-bit arithmetic for approximation functions. For the 16-bit fixed-point implementation the output precision is recalculated to +/-1LSB in 12-bit arithmetic for non-approximation functions and +/-3LSB for approximation functions in 12-bit arithmetic.

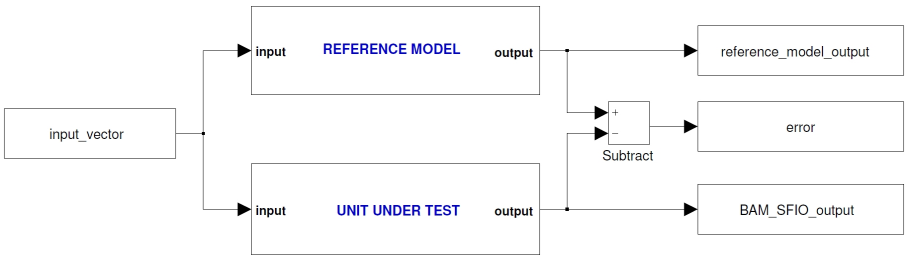

Figure 2-24. Principle of MCLib testing

In order to test the UUT under all conditions, three types of test vector sets are used:

- Deterministic vectors - a specifically defined set of input values over the entire input range.
- Stochastic vectors - a pseudo-randomly generated set of values (non-deterministic values fully covering the input range).
- Boundary vectors - a set of input values for which the potential weaknesses of the tested function are expected. This test is performed only on functions where these limit conditions might occur.

Each function is considered tested if the required accuracy during deterministic, stochastic and boundary tests has been achieved. The following two subchapters MCLib Testing based on the MATLAB Simulink Toolbox and MCLib target-in-loop Testing based on the SFIO Toolbox describe the differences between MCLIB testing based on BAM models and target-in-loop testing based on SFIO models.

2.13.1 MCLib Testing based on the MATLAB Simulink Toolbox

An example of the testing principle based on the BAM is depicted in the Clark transformation function (Figure 2-25). The Bit Accurate Model contains the binary MEX-file built from the GMCLIB_Clark function using the MATLAB compiler. This file is called inside the BAM model, see Figure 2-26. The Reference Model of the Clark transformation is not included in the MATLAB Simulink Toolbox and hence its mathematical representation had to be created. A detailed scheme of the Clark RM is in Figure 2-27.

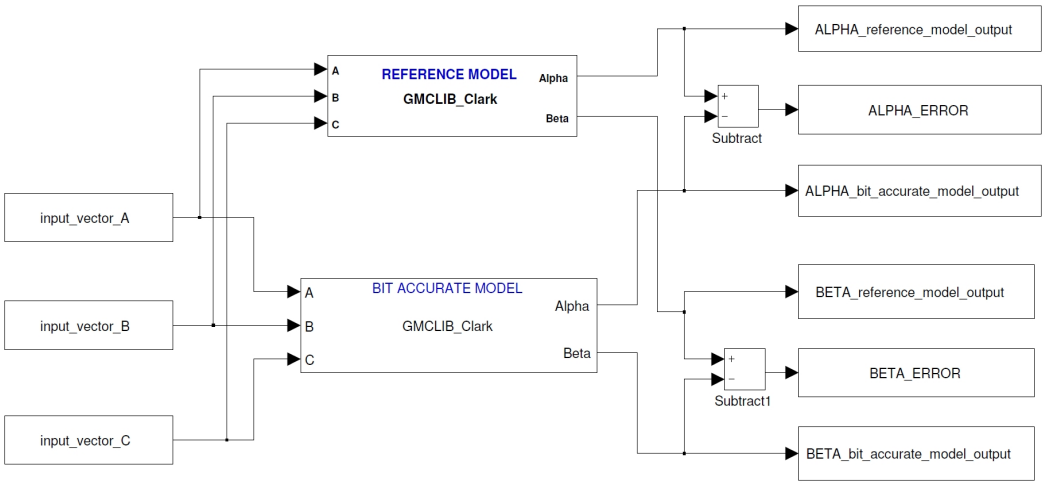

Figure 2-25. Testing of the GMCLIB_Clark function based on the MATLAB Simulink Toolbox

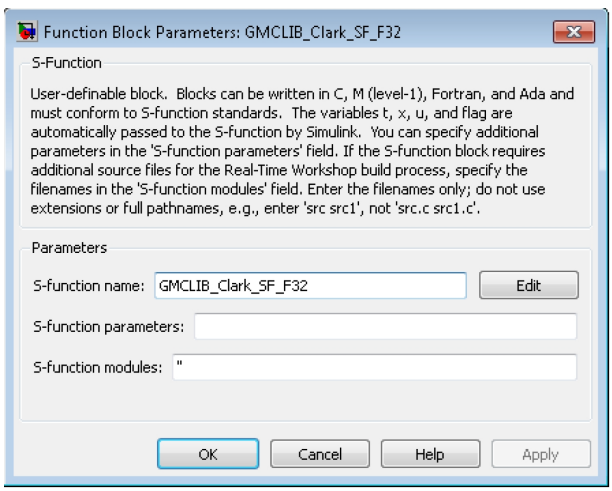

Figure 2-26. Bit Accurate Model parameters of the Clark transformation

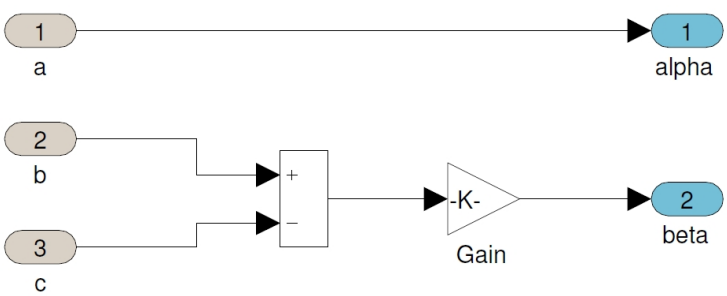

Figure 2-27. Reference Model of the GMCLIB_Clark function

2.13.2 MCLib target-in-loop Testing based on the SFIO Toolbox

The testing method in Figure 2-28 is the similar to that described in the previous chapter with exception that the BAM model is replaced by the SFIO model. The SFIO Toolbox realizes the bridge between Matlab and the Embedded target. During testing, the function GMCLIB_Clark is called directly from the application running on the target MCU. Unlike testing based on MATLAB, the target-in-loop method verifies that the implementation of the Embedded Software and Motor Control Libraries for MPXR40xx functions works correctly on the target MCU. Moreover, the SFIO application running on the processor is used to measure performance of the functions.

The SFIO block Set-up allows the setting of communication parameters which are common to the whole scheme.

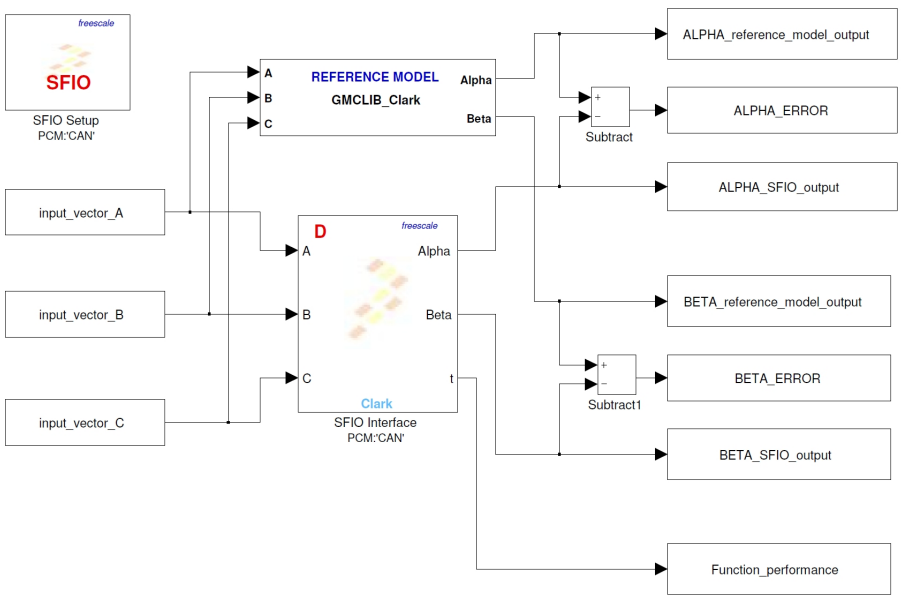

Figure 2-28. Target-in-loop testing example based on the SFIO Toolbox

Chapter 3

3.1 Function Index

Table 3-1. Quick function reference

Type	Name	Arguments
void	GDFLIB_FilterFIRInit_F16	const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState tFrac16 * pInBuf
void	GDFLIB_FilterFIRInit_F32	const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam GDFLIB_FILTERFIR_STATE_T_F32 *const pState tFrac32 * pInBuf
void	GDFLIB_FilterFIRInit_FLT	const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam GDFLIB_FILTERFIR_STATE_T_FLT *const pState tFloat * pInBuf
tFrac16	GDFLIB_FilterFIR_F16	tFrac16 f16In const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam GDFLIB_FILTERFIR_STATE_T_F16 *const pState
tFrac32	GDFLIB_FilterFIR_F32	tFrac32 f32In const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam GDFLIB_FILTERFIR_STATE_T_F32 *const pState
tFloat	GDFLIB_FilterFIR_FLT	tFloat fltIn const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam GDFLIB_FILTERFIR_STATE_T_FLT *const pState
void	GDFLIB_FilterIIR1Init_F16	GDFLIB_FILTER_IIR1_T_F16 *const pParam
void	GDFLIB_FilterIIR1Init_F32	GDFLIB_FILTER_IIR1_T_F32 *const pParam
void	GDFLIB_FilterIIR1Init_FLT	GDFLIB_FILTER_IIR1_T_FLT *const pParam
tFrac16	GDFLIB_FilterIIR1_F16	tFrac16 f16In GDFLIB_FILTER_IIR1_T_F16 *const pParam
tFrac32	GDFLIB_FilterIIR1_F32	tFrac32 f32In GDFLIB_FILTER_IIR1_T_F32 *const pParam
tFloat	GDFLIB_FilterIIR1_FLT	tFloat fltIn GDFLIB_FILTER_IIR1_T_FLT *const pParam
void	GDFLIB_FilterIIR2Init_F16	GDFLIB_FILTER_IIR2_T_F16 *const pParam

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
void	GDFLIB_FilterIIR2Init_F32	GDFLIB_FILTER_IIR2_T_F32 *const pParam
void	GDFLIB_FilterIIR2Init_FLT	GDFLIB_FILTER_IIR2_T_FLT *const pParam
tFrac16	GDFLIB_FilterIIR2_F16	tFrac16 f16In GDFLIB_FILTER_IIR2_T_F16 *const pParam
tFrac32	GDFLIB_FilterIIR2_F32	tFrac32 f32In GDFLIB_FILTER_IIR2_T_F32 *const pParam
tFloat	GDFLIB_FilterIIR2_FLT	tFloat fltIn GDFLIB_FILTER_IIR2_T_FLT *const pParam
void	GDFLIB_FilterMAInit_F16	GDFLIB_FILTER_MA_T_F16 * pParam
void	GDFLIB_FilterMAInit_F32	GDFLIB_FILTER_MA_T_F32 * pParam
void	GDFLIB_FilterMAInit_FLT	GDFLIB_FILTER_MA_T_FLT * pParam
tFrac16	GDFLIB_FilterMA_F16	tFrac16 f16In GDFLIB_FILTER_MA_T_F16 * pParam
tFrac32	GDFLIB_FilterMA_F32	tFrac32 f32In GDFLIB_FILTER_MA_T_F32 * pParam
tFloat	GDFLIB_FilterMA_FLT	tFloat fltIn GDFLIB_FILTER_MA_T_FLT * pParam
tFrac16	GFLIB_Acos_F16	tFrac16 f16In const GFLIB_ACOS_T_F16 *const pParam
tFrac32	GFLIB_Acos_F32	tFrac32 f32In const GFLIB_ACOS_T_F32 *const pParam
tFloat	GFLIB_Acos_FLT	tFloat fltIn const GFLIB_ACOS_T_FLT *const pParam
tFrac16	GFLIB_Asin_F16	tFrac16 f16In const GFLIB_ASIN_T_F16 *const pParam
tFrac32	GFLIB_Asin_F32	tFrac32 f32In const GFLIB_ASIN_T_F32 *const pParam
tFloat	GFLIB_Asin_FLT	tFloat fltIn const GFLIB_ASIN_T_FLT *const pParam
tFrac16	GFLIB_AtanYXShifted_F16	tFrac16 f16InY tFrac16 f16InX GFLIB_ATANYXSHIFTED_T_F16 * pParam
tFrac32	GFLIB_AtanYXShifted_F32	tFrac32 f32InY tFrac32 f32InX GFLIB_ATANYXSHIFTED_T_F32 * pParam
tFloat	GFLIB_AtanYXShifted_FLT	tFloat fltInY tFloat fltInX GFLIB_ATANYXSHIFTED_T_FLT * pParam
tFrac16	GFLIB_AtanYX_F16	tFrac16 f16InY tFrac16 f16InX
tFrac32	GFLIB_AtanYX_F32	tFrac32 f32InY tFrac32 f32InX
tFloat	GFLIB_AtanYX_FLT	tFloat fltInY tFloat fltInX
tFrac16	GFLIB_Atan_F16	tFrac16 f16In const GFLIB_ATAN_T_F16 *const pParam

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFrac32	**GFLIB_Atan_F32**	tFrac32 f32In const GFLIB_ATAN_T_F32 *const pParam
tFloat	**GFLIB_Atan_FLT**	tFloat fltIn const GFLIB_ATAN_T_FLT *const pParam
tFrac16	**GFLIB_ControllerPIpAW_F16**	tFrac16 f16InErr GFLIB_CONTROLLER_PIAW_P_T_F16 *const pParam
tFrac32	**GFLIB_ControllerPIpAW_F32**	tFrac32 f32InErr GFLIB_CONTROLLER_PIAW_P_T_F32 *const pParam
tFloat	**GFLIB_ControllerPIpAW_FLT**	tFloat fltInErr GFLIB_CONTROLLER_PIAW_P_T_FLT *const pParam
tFrac16	**GFLIB_ControllerPIp_F16**	tFrac16 f16InErr GFLIB_CONTROLLER_PI_P_T_F16 *const pParam
tFrac32	**GFLIB_ControllerPIp_F32**	tFrac32 f32InErr GFLIB_CONTROLLER_PI_P_T_F32 *const pParam
tFloat	**GFLIB_ControllerPIp_FLT**	tFloat fltInErr GFLIB_CONTROLLER_PI_P_T_FLT *const pParam
tFrac16	**GFLIB_ControllerPIrAW_F16**	tFrac16 f16InErr GFLIB_CONTROLLER_PIAW_R_T_F16 *const pParam
tFrac32	**GFLIB_ControllerPIrAW_F32**	tFrac32 f32InErr GFLIB_CONTROLLER_PIAW_R_T_F32 *const pParam
tFloat	**GFLIB_ControllerPIrAW_FLT**	tFloat fltInErr GFLIB_CONTROLLER_PIAW_R_T_FLT *const pParam
tFrac16	**GFLIB_ControllerPIr_F16**	tFrac16 f16InErr GFLIB_CONTROLLER_PI_R_T_F16 *const pParam
tFrac32	**GFLIB_ControllerPIr_F32**	tFrac32 f32InErr GFLIB_CONTROLLER_PI_R_T_F32 *const pParam
tFloat	**GFLIB_ControllerPIr_FLT**	tFloat fltInErr GFLIB_CONTROLLER_PI_R_T_FLT *const pParam
tFrac16	**GFLIB_Cos_F16**	tFrac16 f16In const GFLIB_COS_T_F16 *const pParam
tFrac32	**GFLIB_Cos_F32**	tFrac32 f32In const GFLIB_COS_T_F32 *const pParam
tFloat	**GFLIB_Cos_FLT**	tFloat fltIn const GFLIB_COS_T_FLT *const pParam
tFrac16	**GFLIB_Hyst_F16**	tFrac16 f16In GFLIB_HYST_T_F16 *const pParam
tFrac32	**GFLIB_Hyst_F32**	tFrac32 f32In GFLIB_HYST_T_F32 *const pParam
tFloat	**GFLIB_Hyst_FLT**	tFloat fltIn GFLIB_HYST_T_FLT *const pParam

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFrac16	**GFLIB_IntegratorTR_F16**	tFrac16 f16In GFLIB_INTEGRATOR_TR_T_F16 *const pParam
tFrac32	**GFLIB_IntegratorTR_F32**	tFrac32 f32In GFLIB_INTEGRATOR_TR_T_F32 *const pParam
tFloat	**GFLIB_IntegratorTR_FLT**	tFloat fltIn GFLIB_INTEGRATOR_TR_T_FLT *const pParam
tFrac16	**GFLIB_Limit_F16**	tFrac16 f16In const GFLIB_LIMIT_T_F16 *const pParam
tFrac32	**GFLIB_Limit_F32**	tFrac32 f32In const GFLIB_LIMIT_T_F32 *const pParam
tFloat	**GFLIB_Limit_FLT**	tFloat fltIn const GFLIB_LIMIT_T_FLT *const pParam
tFrac16	**GFLIB_LowerLimit_F16**	tFrac16 f16In const GFLIB_LOWERLIMIT_T_F16 *const pParam
tFrac32	**GFLIB_LowerLimit_F32**	tFrac32 f32In const GFLIB_LOWERLIMIT_T_F32 *const pParam
tFloat	**GFLIB_LowerLimit_FLT**	tFloat fltIn const GFLIB_LOWERLIMIT_T_FLT *const pParam
tFrac16	**GFLIB_Lut1D_F16**	tFrac16 f16In const GFLIB_LUT1D_T_F16 *const pParam
tFrac32	**GFLIB_Lut1D_F32**	tFrac32 f32In const GFLIB_LUT1D_T_F32 *const pParam
tFloat	**GFLIB_Lut1D_FLT**	tFloat fltIn const GFLIB_LUT1D_T_FLT *const pParam
tFrac16	**GFLIB_Lut2D_F16**	tFrac16 f16In1 tFrac16 f16In2 const GFLIB_LUT2D_T_F16 *const pParam
tFrac32	**GFLIB_Lut2D_F32**	tFrac32 f32In1 tFrac32 f32In2 const GFLIB_LUT2D_T_F32 *const pParam
tFloat	**GFLIB_Lut2D_FLT**	tFloat fltIn1 tFloat fltIn2 const GFLIB_LUT2D_T_FLT *const pParam
tFrac16	**GFLIB_Ramp_F16**	tFrac16 f16In GFLIB_RAMP_T_F16 *const pParam
tFrac32	**GFLIB_Ramp_F32**	tFrac32 f32In GFLIB_RAMP_T_F32 *const pParam
tFloat	**GFLIB_Ramp_FLT**	tFloat fltIn GFLIB_RAMP_T_FLT *const pParam
tFrac16	**GFLIB_Sign_F16**	tFrac16 f16In
tFrac32	**GFLIB_Sign_F32**	tFrac32 f32In
tFloat	**GFLIB_Sign_FLT**	tFloat fltIn
tFrac16	**GFLIB_Sin_F16**	tFrac16 f16In const GFLIB_SIN_T_F16 *const pParam
tFrac32	**GFLIB_Sin_F32**	tFrac32 f32In const GFLIB_SIN_T_F32 *const pParam

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFloat	**GFLIB_Sin_FLT**	tFloat fltIn const GFLIB_SIN_T_FLT *const pParam
tFrac16	**GFLIB_Sqrt_F16**	tFrac16 f16In
tFrac32	**GFLIB_Sqrt_F32**	tFrac32 f32In
tFloat	**GFLIB_Sqrt_FLT**	tFloat fltIn
tFrac16	**GFLIB_Tan_F16**	tFrac16 f16In const GFLIB_TAN_T_F16 *const pParam
tFrac32	**GFLIB_Tan_F32**	tFrac32 f32In const GFLIB_TAN_T_F32 *const pParam
tFloat	**GFLIB_Tan_FLT**	tFloat fltIn const GFLIB_TAN_T_FLT *const pParam
tFrac16	**GFLIB_UpperLimit_F16**	tFrac16 f16In const GFLIB_UPPERLIMIT_T_F16 *const pParam
tFrac32	**GFLIB_UpperLimit_F32**	tFrac32 f32In const GFLIB_UPPERLIMIT_T_F32 *const pParam
tFloat	**GFLIB_UpperLimit_FLT**	tFloat fltIn const GFLIB_UPPERLIMIT_T_FLT *const pParam
tBool	**GFLIB_VectorLimit_F16**	const SWLIBS_2Syst_F16 *const pIn SWLIBS_2Syst_F16 *const pOut const GFLIB_VECTORLIMIT_T_F16 *const pParam
tBool	**GFLIB_VectorLimit_F32**	const SWLIBS_2Syst_F32 *const pIn SWLIBS_2Syst_F32 *const pOut const GFLIB_VECTORLIMIT_T_F32 *const pParam
tBool	**GFLIB_VectorLimit_FLT**	const SWLIBS_2Syst_FLT *const pIn SWLIBS_2Syst_FLT *const pOut const GFLIB_VECTORLIMIT_T_FLT *const pParam
void	**GMCLIB_ClarkInv_F16**	const SWLIBS_2Syst_F16 *const pIn SWLIBS_3Syst_F16 *const pOut
void	**GMCLIB_ClarkInv_F32**	const SWLIBS_2Syst_F32 *const pIn SWLIBS_3Syst_F32 *const pOut
void	**GMCLIB_ClarkInv_FLT**	const SWLIBS_2Syst_FLT *const pIn SWLIBS_3Syst_FLT *const pOut
void	**GMCLIB_Clark_F16**	const SWLIBS_3Syst_F16 *const pIn SWLIBS_2Syst_F16 *const pOut
void	**GMCLIB_Clark_F32**	const SWLIBS_3Syst_F32 *const pIn SWLIBS_2Syst_F32 *const pOut
void	**GMCLIB_Clark_FLT**	const SWLIBS_3Syst_FLT *const pIn SWLIBS_2Syst_FLT *const pOut
void	**GMCLIB_DecouplingPMSM_F16**	SWLIBS_2Syst_F16 *const pUdqDec const SWLIBS_2Syst_F16 *const pUdq const SWLIBS_2Syst_F16 *const pIdq tFrac16 f16AngularVel const GMCLIB_DECOUPLINGPMSM_T_F16 *const pParam

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
void	GMCLIB_DecouplingPMSM_F32	SWLIBS_2Syst_F32 *const pUdqDec const SWLIBS_2Syst_F32 *const pUdq const SWLIBS_2Syst_F32 *const pIdq tFrac32 f32AngularVel const GMCLIB_DECOUPLINGPMSM_T_F32 *const pParam
void	GMCLIB_DecouplingPMSM_FLT	SWLIBS_2Syst_FLT *const pUdqDec const SWLIBS_2Syst_FLT *const pUdq const SWLIBS_2Syst_FLT *const pIdq tFloat fltAngularVel const GMCLIB_DECOUPLINGPMSM_T_FLT *const pParam
void	GMCLIB_ElimDcBusRip_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pIn const GMCLIB_ELIMDCBUSRIP_T_F16 *const pParam
void	GMCLIB_ElimDcBusRip_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pIn const GMCLIB_ELIMDCBUSRIP_T_F32 *const pParam
void	GMCLIB_ElimDcBusRip_FLT	SWLIBS_2Syst_FLT *const pOut const SWLIBS_2Syst_FLT *const pIn const GMCLIB_ELIMDCBUSRIP_T_FLT *const pParam
void	GMCLIB_ParkInv_F16	SWLIBS_2Syst_F16 *const pOut const SWLIBS_2Syst_F16 *const pInAngle const SWLIBS_2Syst_F16 *const pIn
void	GMCLIB_ParkInv_F32	SWLIBS_2Syst_F32 *const pOut const SWLIBS_2Syst_F32 *const pInAngle const SWLIBS_2Syst_F32 *const pIn
void	GMCLIB_ParkInv_FLT	SWLIBS_2Syst_FLT *const pOut const SWLIBS_2Syst_FLT *const pInAngle const SWLIBS_2Syst_FLT *const pIn
void	GMCLIB_Park_F16	SWLIBS_2Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pInAngle const SWLIBS_2Syst_F16 *const pIn
void	GMCLIB_Park_F32	SWLIBS_2Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pInAngle const SWLIBS_2Syst_F32 *const pIn
void	GMCLIB_Park_FLT	SWLIBS_2Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pInAngle const SWLIBS_2Syst_FLT *const pIn
tU16	GMCLIB_SvmStd_F16	SWLIBS_3Syst_F16 * pOut const SWLIBS_2Syst_F16 *const pIn
tU32	GMCLIB_SvmStd_F32	SWLIBS_3Syst_F32 * pOut const SWLIBS_2Syst_F32 *const pIn
tU32	GMCLIB_SvmStd_FLT	SWLIBS_3Syst_FLT * pOut const SWLIBS_2Syst_FLT *const pIn

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
const MCLIB_VERSION_T *	**MCLIB_GetVersion**	void
tFrac16	**MLIB_AbsSat_F16**	register tFrac16 f16In
tFrac32	**MLIB_AbsSat_F32**	register tFrac32 f32In
tFrac16	**MLIB_Abs_F16**	register tFrac16 f16In register tFrac16 f16In
tFrac32	**MLIB_Abs_F32**	register tFrac32 f32In register tFrac32 f32In
tFloat	**MLIB_Abs_FLT**	register tFloat fltIn register tFloat fltIn
tFrac16	**MLIB_AddSat_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_AddSat_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	**MLIB_Add_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_Add_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In1 register tFrac32 f32In2
tFloat	**MLIB_Add_FLT**	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn1 register tFloat fltIn2
tFrac16	**MLIB_ConvertPU_F16F32**	register tFrac32 f32In
tFrac16	**MLIB_ConvertPU_F16FLT**	register tFloat fltIn
tFrac32	**MLIB_ConvertPU_F32F16**	register tFrac16 f16In
tFrac32	**MLIB_ConvertPU_F32FLT**	register tFloat fltIn
tFloat	**MLIB_ConvertPU_FLTF16**	register tFrac16 f16In
tFloat	**MLIB_ConvertPU_FLTF32**	register tFrac32 f32In
tFrac16	**MLIB_Convert_F16F32**	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	**MLIB_Convert_F16FLT**	register tFloat fltIn1 register tFloat fltIn2
tFrac32	**MLIB_Convert_F32F16**	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_Convert_F32FLT**	register tFloat fltIn1 register tFloat fltIn2
tFloat	**MLIB_Convert_FLTF16**	register tFrac16 f16In1 register tFrac16 f16In2

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFloat	**MLIB_Convert_FLTF32**	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	**MLIB_DivSat_F16**	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_DivSat_F32**	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	**MLIB_Div_F16**	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_Div_F32**	register tFrac32 f32In1 register tFrac32 f32In2
tFloat	**MLIB_Div_FLT**	register tFloat fltIn1 register tFloat fltIn2
tFrac16	**MLIB_MacSat_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac32	**MLIB_MacSat_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
tFrac32	**MLIB_MacSat_F32F16F16**	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac16	**MLIB_Mac_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3
tFrac32	**MLIB_Mac_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3 register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3
tFrac32	**MLIB_Mac_F32F16F16**	register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac32 f32In1 register tFrac16 f16In2 register tFrac16 f16In3
tFloat	**MLIB_Mac_FLT**	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3 register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3
tFrac16	**MLIB_MulSat_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFrac32	**MLIB_MulSat_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In1 register tFrac32 f32In2
tFrac32	**MLIB_MulSat_F32F16F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2
tFrac16	**MLIB_Mul_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_Mul_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In1 register tFrac32 f32In2
tFrac32	**MLIB_Mul_F32F16F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2
tFloat	**MLIB_Mul_FLT**	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn1 register tFloat fltIn2
tFrac16	**MLIB_NegSat_F16**	register tFrac16 f16In
tFrac32	**MLIB_NegSat_F32**	register tFrac32 f32In
tFrac16	**MLIB_Neg_F16**	register tFrac16 f16In
tFrac32	**MLIB_Neg_F32**	register tFrac32 f32In
tFloat	**MLIB_Neg_FLT**	register tFloat fltIn
tU16	**MLIB_Norm_F16**	register tFrac16 f16In
tU16	**MLIB_Norm_F32**	register tFrac32 f32In
tFrac16	**MLIB_Round_F16**	register tFrac16 f16In1 register tU16 u16In2
tFrac32	**MLIB_Round_F32**	register tFrac32 f32In1 register tU16 u16In2
tFrac16	**MLIB_ShBiSat_F16**	register tFrac16 f16In1 register tS16 s16In2
tFrac32	**MLIB_ShBiSat_F32**	register tFrac32 f32In1 register tS16 s16In2
tFrac16	**MLIB_ShBi_F16**	register tFrac16 f16In1 register tS16 s16In2
tFrac32	**MLIB_ShBi_F32**	register tFrac32 f32In1 register tS16 s16In2
tFrac16	**MLIB_ShLSat_F16**	register tFrac16 f16In1 register tU16 u16In2
tFrac32	**MLIB_ShLSat_F32**	register tFrac32 f32In1 register tU16 u16In2

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFrac16	**MLIB_ShL_F16**	register tFrac16 f16In1 register tU16 u16In2
tFrac32	**MLIB_ShL_F32**	register tFrac32 f32In1 register tU16 u16In2
tFrac16	**MLIB_ShR_F16**	register tFrac16 f16In1 register tU16 u16In2
tFrac32	**MLIB_ShR_F32**	register tFrac32 f32In1 register tU16 u16In2
tFrac16	**MLIB_SubSat_F16**	register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_SubSat_F32**	register tFrac32 f32In1 register tFrac32 f32In2
tFrac16	**MLIB_Sub_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In1 register tFrac16 f16In2
tFrac32	**MLIB_Sub_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In1 register tFrac32 f32In2
tFloat	**MLIB_Sub_FLT**	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn1 register tFloat fltIn2
tFrac16	**MLIB_VMac_F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4 register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4
tFrac32	**MLIB_VMac_F32**	register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3 register tFrac32 f32In4 register tFrac32 f32In1 register tFrac32 f32In2 register tFrac32 f32In3 register tFrac32 f32In4
tFrac32	**MLIB_VMac_F32F16F16**	register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4 register tFrac16 f16In1 register tFrac16 f16In2 register tFrac16 f16In3 register tFrac16 f16In4

Table continues on the next page...

Table 3-1. Quick function reference (continued)

Type	Name	Arguments
tFloat	**MLIB_VMac_FLT**	register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3 register tFloat fltIn4 register tFloat fltIn1 register tFloat fltIn2 register tFloat fltIn3 register tFloat fltIn4

Chapter 4
API References

This section describes in details the Application Interface for all functions available in Embedded Software and Motor Control Libraries for MPXR40xx.

4.1 Function GDFLIB_FilterFIRInit_F32

This function initializes the FIR filter buffers.

4.1.1 Declaration

```
void GDFLIB_FilterFIRInit_F32(const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F32 *const pState, tFrac32 *pInBuf);
```

4.1.2 Arguments

Table 4-1. GDFLIB_FilterFIRInit_F32 arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_F32 *const	pParam	**input**	Pointer to the parameters structure.
GDFLIB_FILTERFIR_STATE_T_F32 *const	pState	**input, output**	Pointer to the state structure.
tFrac32 *	pInBuf	**input, output**	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be a filter order + 1 long.

4.1.3 Return

void

4.1.4 Description

The function performs the initialization procedure for the GDFLIB_FilterFIR_F32 function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the GDFLIB_FilterFIR_F32 function.

Note

The input buffer pointer (State->pInBuf) must point to a Read/Write memory region, which must be at least the number of the filter taps long. The number of taps in a filter is equal to the filter order + 1. There is no restriction as to the location of the parameters structure as long as it is readable.

CAUTION

No check is performed for R/W capability and the length of the input buffer (pState->pInBuf).

4.1.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

4.1.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State;

tFrac32 f32InBuf[FIR_NUMTAPS_MAX];
```

```
tFrac32 f32CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac32 f32OutBuf[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N     = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f32CoefBuf[ii++] = 0xFFB10C14;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0xFFB10C14;

    Param.u32Order = 15;
    Param.pf32CoefBuf = &f32CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F32 (&Param, &State, &f32InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &f32InBuf[0], Define F32);

    // ##################################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##################################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &f32InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // f32Out contains step response of the filter
      f32OutBuf[ii] = GDFLIB_FilterFIR_F32 (FRAC32 (1.0), &Param, &State);

      // f32Out contains step response of the filter
      f32OutBuf[ii] = GDFLIB_FilterFIR (FRAC32 (1.0), &Param, &State, Define
F32);

      // ##################################################################
      // Available only if 32-bit fractional implementation selected
      // as default
```

```
            // ##############################################################

            // f32Out contains step response of the filter
            f32OutBuf[ii] = GDFLIB_FilterFIR (FRAC32 (1.0), &Param, &State);
        }
    }
```

4.2 Function GDFLIB_FilterFIR_F32

The function performs a single iteration of an FIR filter.

4.2.1 Declaration

```
tFrac32 GDFLIB_FilterFIR_F32(tFrac32 f32In, const GDFLIB_FILTERFIR_PARAM_T_F32 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F32 *const pState);
```

4.2.2 Arguments

Table 4-2. GDFLIB_FilterFIR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GDFLIB_FILTERFIR_PARAM_T_F32 *const	pParam	**input**	Pointer to the parameter structure.
GDFLIB_FILTERFIR_STATE_T_F32 *const	pState	**input, output**	Pointer to the filter state structure.

4.2.3 Return

The value of a filtered signal after processing by an FIR filter.

4.2.4 Description

The function performs the operation of an FIR filter on a sample-by-sample basis. At each new input to the FIR filter, the function should be called, which will return a new filtered value.

The FIR filter is defined by the following formula:

$$y[n] = h_0 x[n] + h_1 x[n-1] + \ldots + h_N x[n-N]$$

Equation **GDFLIB_FilterFIR_Eq1**

where: x[n] is the input signal, y[n] is the output signal, h_i are the filter coefficients, and N is the filter order. It should be noted, that the number of taps of the filter is N + 1 in this case.

The multiply and accumulate operations are performed with 64 accumulation, which means that no saturation is performed during computations. However, if the final value cannot fit in the return data type, saturation may occur. It should be noted, although rather theoretically, that no saturation is performed on the accumulation guard bits and an overflow over the accumulation guard bits may occur.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffffffff(hexadecimal) taps, which is equivalent to the order of 0xfffffffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member pState->pInBuf. The buffer index is stored in pState->u32Idx, which points to the buffer element where a new input signal sample will be stored.

The filter coefficients are stored in the parameter structure, in the structure member pParam->pCoefBuf.

The first call to the function must be preceded by an initialization, which can be made through the GDFLIB_FilterFIRInit_F32 function. The GDFLIB_FilterFIRInit_F32 and then the GDFLIB_FilterFIR_F32 functions should be called with the same parameters.

4.2.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

Note

From the performance point of view, the function is designed to work with filters with a larger number of taps (equal order + 1). As a rule of thumb, if the number of taps is lower than 5, a different algorithm should be considered.

4.2.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F32 Param;
GDFLIB_FILTERFIR_STATE_T_F32 State;

tFrac32 f32InBuf[FIR_NUMTAPS_MAX];
tFrac32 f32CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac32 f32OutBuf[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N    = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f32CoefBuf[ii++] = 0xFFB10C14;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x393ED867;
    f32CoefBuf[ii++] = 0x11950A3C;
    f32CoefBuf[ii++] = 0xF7183CC7;
    f32CoefBuf[ii++] = 0xFB245142;
    f32CoefBuf[ii++] = 0x028E6845;
    f32CoefBuf[ii++] = 0x01387DD7;
    f32CoefBuf[ii++] = 0xFF779D25;
    f32CoefBuf[ii++] = 0xFFB10C14;

    Param.u32Order = 15;
    Param.pf32CoefBuf = &f32CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F32 (&Param, &State, &f32InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &f32InBuf[0], Define F32);

    // ##################################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##################################################################

    // Initialize FIR filter
```

```
                    GDFLIB_FilterFIRInit (&Param, &State, &f32InBuf[0]);

                    // Compute step response of the filter
                    for(ii=0; ii < OUT_LEN; ii++)
                    {
                      // f32Out contains step response of the filter
                      f32OutBuf[ii] = GDFLIB_FilterFIR_F32 (FRAC32 (1.0), &Param, &State);

                      // f32Out contains step response of the filter
                      f32OutBuf[ii] = GDFLIB_FilterFIR (FRAC32 (1.0), &Param, &State, Define
F32);

                      // ##############################################################
                      // Available only if 32-bit fractional implementation selected
                      // as default
                      // ##############################################################

                      // f32Out contains step response of the filter
                      f32OutBuf[ii] = GDFLIB_FilterFIR (FRAC32 (1.0), &Param, &State);
                    }
                }
```

4.3 Function GDFLIB_FilterFIRInit_F16

This function initializes the FIR filter buffers.

4.3.1 Declaration

```
void GDFLIB_FilterFIRInit_F16(const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState, tFrac16 *pInBuf);
```

4.3.2 Arguments

Table 4-3. GDFLIB_FilterFIRInit_F16 arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_F16 *const	pParam	**input**	Pointer to the parameters structure.
GDFLIB_FILTERFIR_STATE_T_F16 *const	pState	**input, output**	Pointer to the state structure.
tFrac16 *	pInBuf	**input, output**	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be a filter order + 1 long.

4.3.3 Return

void

4.3.4 Description

The function performs the initialization procedure for the GDFLIB_FilterFIR_F16 function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the GDFLIB_FilterFIR_F16 function.

Note

The input buffer pointer (State->pInBuf) must point to a Read/Write memory region, which must be at least the number of the filter taps long. The number of taps in a filter is equal to the filter order + 1. There is no restriction as to the location of the parameters structure as long as it is readable.

CAUTION

No check is performed for R/W capability and the length of the input buffer (pState->pInBuf).

4.3.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

4.3.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State;

tFrac16 f16InBuf[FIR_NUMTAPS_MAX];
```

```
tFrac16 f16CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac16 f16OutBuf[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N     = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f16CoefBuf[ii++] = 0xFFB1;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0xFFB1;

    Param.u16Order = 15;
    Param.pf16CoefBuf = &f16CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F16 (&Param, &State, &f16InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &f16InBuf[0], Define F16);

    // ##################################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##################################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &f16InBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // f16Out contains step response of the filter
      f16OutBuf[ii] = GDFLIB_FilterFIR_F16 (FRAC16 (1.0), &Param, &State);

      // f16Out contains step response of the filter
      f16OutBuf[ii] = GDFLIB_FilterFIR (FRAC16 (1.0), &Param, &State, Define
F16);

      // ##################################################################
      // Available only if 16-bit fractional implementation selected
      // as default
```

```
            // ##############################################################

            // f16Out contains step response of the filter
            f16OutBuf[ii] = GDFLIB_FilterFIR (FRAC16 (1.0), &Param, &State);
        }
    }
```

4.4 Function GDFLIB_FilterFIR_F16

The function performs a single iteration of an FIR filter.

4.4.1 Declaration

```
tFrac16 GDFLIB_FilterFIR_F16(tFrac16 f16In, const GDFLIB_FILTERFIR_PARAM_T_F16 *const pParam,
GDFLIB_FILTERFIR_STATE_T_F16 *const pState);
```

4.4.2 Arguments

Table 4-4. GDFLIB_FilterFIR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GDFLIB_FILTERFIR_PARAM_T_F16 *const	pParam	**input**	Pointer to the parameter structure.
GDFLIB_FILTERFIR_STATE_T_F16 *const	pState	**input, output**	Pointer to the filter state structure.

4.4.3 Return

The value of a filtered signal after processing by an FIR filter.

4.4.4 Description

The function performs the operation of an FIR filter on a sample-by-sample basis. At each new input to the FIR filter, the function should be called, which will return a new filtered value.

The FIR filter is defined by the following formula:

$$y[n] = h_0 x[n] + h_1 x[n-1] + \ldots + h_N x[n-N]$$

Equation **GDFLIB_FilterFIR_Eq1**

where: x[n] is the input signal, y[n] is the output signal, h_i are the filter coefficients, and N is the filter order. It should be noted, that the number of taps of the filter is N + 1 in this case.

The multiply and accumulate operations are performed with 32 accumulation, which means that no saturation is performed during computations. However, if the final value cannot fit in the return data type, saturation may occur. It should be noted, although rather theoretically, that no saturation is performed on the accumulation guard bits and an overflow over the accumulation guard bits may occur.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffff(hexadecimal) taps, which is equivalent to the order of 0xfffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member pState->pInBuf. The buffer index is stored in pState->u16Idx, which points to the buffer element where a new input signal sample will be stored.

The filter coefficients are stored in the parameter structure, in the structure member pParam->pCoefBuf.

The first call to the function must be preceded by an initialization, which can be made through the GDFLIB_FilterFIRInit_F16 function. The GDFLIB_FilterFIRInit_F16 and then the GDFLIB_FilterFIR_F16 functions should be called with the same parameters.

4.4.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

Note

From the performance point of view, the function is designed to work with filters with a larger number of taps (equal order + 1). As a rule of thumb, if the number of taps is lower than 5, a different algorithm should be considered.

4.4.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_F16 Param;
GDFLIB_FILTERFIR_STATE_T_F16 State;

tFrac16 f16InBuf[FIR_NUMTAPS_MAX];
tFrac16 f16CoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFrac16 f16OutBuf[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N    = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    f16CoefBuf[ii++] = 0xFFB1;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x393E;
    f16CoefBuf[ii++] = 0x1195;
    f16CoefBuf[ii++] = 0xF718;
    f16CoefBuf[ii++] = 0xFB24;
    f16CoefBuf[ii++] = 0x028E;
    f16CoefBuf[ii++] = 0x0138;
    f16CoefBuf[ii++] = 0xFF77;
    f16CoefBuf[ii++] = 0xFFB1;

    Param.u16Order = 15;
    Param.pf16CoefBuf = &f16CoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_F16 (&Param, &State, &f16InBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &f16InBuf[0], Define F16);

    // ##################################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##################################################################

    // Initialize FIR filter
```

```
                    GDFLIB_FilterFIRInit (&Param, &State, &f16InBuf[0]);

                    // Compute step response of the filter
                    for(ii=0; ii < OUT_LEN; ii++)
                    {
                      // f16Out contains step response of the filter
                      f16OutBuf[ii] = GDFLIB_FilterFIR_F16 (FRAC16 (1.0), &Param, &State);

                      // f16Out contains step response of the filter
                      f16OutBuf[ii] = GDFLIB_FilterFIR (FRAC16 (1.0), &Param, &State, Define
F16);

                      // ##############################################################
                      // Available only if 16-bit fractional implementation selected
                      // as default
                      // ##############################################################

                      // f16Out contains step response of the filter
                      f16OutBuf[ii] = GDFLIB_FilterFIR (FRAC16 (1.0), &Param, &State);
                    }
                }
```

4.5 Function GDFLIB_FilterFIRInit_FLT

This function initializes the FIR filter buffers.

4.5.1 Declaration

```
void GDFLIB_FilterFIRInit_FLT(const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam,
GDFLIB_FILTERFIR_STATE_T_FLT *const pState, tFloat *pInBuf);
```

4.5.2 Arguments

Table 4-5. GDFLIB_FilterFIRInit_FLT arguments

Type	Name	Direction	Description
const GDFLIB_FILTERFIR_PARAM_T_FLT *const	pParam	**input**	Pointer to a parameters structure.
GDFLIB_FILTERFIR_STATE_T_FLT *const	pState	**input, output**	Pointer to a state structure.
tFloat *	pInBuf	**input, output**	Pointer to a buffer for storing filter input signal values, must point to a R/W memory region and must be the filter order + 1 long.

4.5.3 Return

void

4.5.4 Description

The function performs the initialization procedure for the GDFLIB_FilterFIR_FLT function. In particular, the function performs the following operations:

1. Resets the input buffer index to zero.
2. Initializes the input buffer pointer to the pointer provided as an argument.
3. Resets the input buffer.

After initialization, made by the function, the parameters and state structures should be provided as arguments to calls of the GDFLIB_FilterFIR_FLT function.

Note

The input buffer pointer (State->pInBuf) must point to a Read/Write memory region, which must be at least the number of the filter taps long. The number of taps in a filter is equal to the filter order + 1. There is no restriction as to the location of the parameters structure as long as it is readable.

CAUTION

No check is performed for R/W capability and the length of the input buffer (pState->pInBuf).

4.5.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

4.5.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)

GDFLIB_FILTERFIR_PARAM_T_FLT Param;
GDFLIB_FILTERFIR_STATE_T_FLT State;

tFloat fltInBuf[FIR_NUMTAPS_MAX];
```

```
tFloat fltCoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFloat fltOutBuf[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N     = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    fltCoefBuf[ii++] = -0.997590551617365;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] =  0.0199709259997918;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] = -0.930427167532233;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] = -0.30427167532233;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] =  0.199709259997918;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] = -0.997590551617365;

    Param.u32Order = 15;
    Param.pfltCoefBuf = &fltCoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_FLT (&Param, &State, &fltInBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &fltInBuf[0], Define FLT);

    // ##################################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##################################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &fltInBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // fltOut contains step response of the filter
      fltOutBuf[ii] = GDFLIB_FilterFIR_FLT ((tFloat)(1), &Param, &State);

      // fltOut contains step response of the filter
      fltOutBuf[ii] = GDFLIB_FilterFIR ((tFloat)(1), &Param, &State, Define
FLT);

      // ##################################################################
      // Available only if single precision floating point
      // implementation selected as default
```

```
        // ##############################################################

        // fltOut contains step response of the filter
        fltOutBuf[ii] = GDFLIB_FilterFIR ((tFloat)(1), &Param, &State);
      }
  }
```

4.6 Function GDFLIB_FilterFIR_FLT

The function performs a single iteration of an FIR filter.

4.6.1 Declaration

```
tFloat GDFLIB_FilterFIR_FLT(tFloat fltIn, const GDFLIB_FILTERFIR_PARAM_T_FLT *const pParam,
GDFLIB_FILTERFIR_STATE_T_FLT *const pState);
```

4.6.2 Arguments

Table 4-6. GDFLIB_FilterFIR_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GDFLIB_FILTERFIR_PARAM_T_FLT *const	pParam	**input**	Pointer to a parameter structure.
GDFLIB_FILTERFIR_STATE_T_FLT *const	pState	**input, output**	Pointer to a filter state structure.

4.6.3 Return

The value of a filtered signal after processing by an FIR filter.

4.6.4 Description

The function performs the operation of an FIR filter on a sample-by-sample basis. At each new input to the FIR filter, the function should be called, which will return a new filtered value.

The FIR filter is defined by the following formula:

$$y[n] = h_0 x[n] + h_1 x[n-1] + \ldots + h_N x[n-N]$$

Equation **GDFLIB_FilterFIR_Eq1**

where: x[n] is the input signal, y[n] is the output signal, h_i are the filter coefficients, and N is the filter order. It should be noted, that the number of taps of the filter is N + 1 in this case.

The function assumes that the filter order is at least one, which is equivalent to two taps. The filter also cannot contain more than 0xffffffff(hexadecimal) taps, which is equivalent to the order of 0xfffffffe (hexadecimal).

The input values are recorded by the function in the provided state structure in a circular buffer, pointed to by the state structure member pState->pInBuf. The buffer index is stored in pState->u32Idx, which points to the buffer element where a new input signal sample will be stored.

The filter coefficients are stored in the parameter structure, in the structure member pParam->pCoefBuf.

The first call to the function must be preceded by an initialization, which can be made through the GDFLIB_FilterFIRInit_FLT function. The GDFLIB_FilterFIRInit_FLT and then the GDFLIB_FilterFIR_FLT functions should be called with the same parameters.

4.6.5 Re-entrancy

The function is re-entrant only if the calling code is provided with a distinct instance of the structure pointed to by pState.

Note

From the performance point of view, the function is designed to work with filters with a larger number of taps (equal order + 1). As a rule of thumb, if the number of taps is lower than 5, a different algorithm should be considered.

4.6.6 Code Example

```
#include "gdflib.h"

#define FIR_NUMTAPS 16
#define FIR_NUMTAPS_MAX 64
#define FIR_ORDER (FIR_NUMTAPS - 1)
```

```
GDFLIB_FILTERFIR_PARAM_T_FLT Param;
GDFLIB_FILTERFIR_STATE_T_FLT State;

tFloat fltInBuf[FIR_NUMTAPS_MAX];
tFloat fltCoefBuf[FIR_NUMTAPS_MAX];

#define OUT_LEN 16

void main(void)
{
    int ii;
    tFloat fltOutBuf[OUT_LEN];

    // Define a simple low-pass filter
    // The filter coefficients were calculated by the following
    // Matlab function (coefficients are contained in Hd.Numerator):
    //
    //function Hd = fir_example
    //FIR_EXAMPLE Returns a discrete-time filter object.
    //N    = 15;
    //F6dB = 0.5;
    //
    //h = fdesign.lowpass('n,fc', N, F6dB);
    //
    //Hd = design(h, 'window');
    //return;
    ii = 0;
    fltCoefBuf[ii++] = -0.997590551617365;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] =  0.0199709259997918;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] = -0.930427167532233;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.447230387221663;
    fltCoefBuf[ii++] =  0.137360839237161;
    fltCoefBuf[ii++] = -0.30427167532233;
    fltCoefBuf[ii++] = -0.962045819946586;
    fltCoefBuf[ii++] =  0.199709259997918;
    fltCoefBuf[ii++] =  0.0095364856578114;
    fltCoefBuf[ii++] = -0.995837825348525;
    fltCoefBuf[ii++] = -0.997590551617365;

    Param.u32Order = 15;
    Param.pfltCoefBuf = &fltCoefBuf[0];

    // Initialize FIR filter
    GDFLIB_FilterFIRInit_FLT (&Param, &State, &fltInBuf[0]);

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &fltInBuf[0], Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Initialize FIR filter
    GDFLIB_FilterFIRInit (&Param, &State, &fltInBuf[0]);

    // Compute step response of the filter
    for(ii=0; ii < OUT_LEN; ii++)
    {
      // fltOut contains step response of the filter
      fltOutBuf[ii] = GDFLIB_FilterFIR_FLT ((tFloat)(1), &Param, &State);

      // fltOut contains step response of the filter
      fltOutBuf[ii] = GDFLIB_FilterFIR ((tFloat)(1), &Param, &State, Define
FLT);
```

```
            // ##############################################################
            // Available only if single precision floating point
            // implementation selected as default
            // ##############################################################

            // fltOut contains step response of the filter
            fltOutBuf[ii] = GDFLIB_FilterFIR ((tFloat)(1), &Param, &State);
        }
    }
```

4.7 Function GDFLIB_FilterIIR1Init_F32

This function initializes the First order IIR filter buffers.

4.7.1 Declaration

```
void GDFLIB_FilterIIR1Init_F32(GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```

4.7.2 Arguments

Table 4-7. GDFLIB_FilterIIR1Init_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	**input, output**	Pointer to filter structure with filter buffer and filter parameters.

4.7.3 Return

Function returns no value.

4.7.4 Description

This function clears the internal buffers of a first order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR1_F32 unless periodic clearing of filter buffers is required.

4.7.5 Re-entrancy

The function is re-entrant.

4.7.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_F32 (&f32trMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init (&f32trMyIIR1, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // function returns no value
    GDFLIB_FilterIIR1Init (&f32trMyIIR1);
}
```

4.8 Function GDFLIB_FilterIIR1_F32

This function implements a Direct Form I first order IIR filter.

4.8.1 Declaration

```
tFrac32 GDFLIB_FilterIIR1_F32(tFrac32 f32In, GDFLIB_FILTER_IIR1_T_F32 *const pParam);
```

4.8.2 Arguments

Table 4-8. GDFLIB_FilterIIR1_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Value of input signal to be filtered in step (k). The value is a 32-bit number in the 1.31 fractional format.
GDFLIB_FILTER_IIR1_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

4.8.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the filtered value of the input signal in step (k).

4.8.4 Description

This function calculates the first order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR1_Eq1**

where N denotes the filter order. The first order IIR filter in the Z-domain is therefore given from equation GDFLIB_FilterIIR1_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Equation **GDFLIB_FilterIIR1_Eq2**

In order to implement the first order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by equation GDFLIB_FilterIIR1_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Equation **GDFLIB_FilterIIR1_Eq3**

Equation GDFLIB_FilterIIR1_Eq3 represents a Direct Form I implementation of a first order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The

main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

Because there are two delay buffers necessary for both DF-I and DF-II implementation of the first order IIR filter, the DF-I implementation was chosen to be used in the GDFLIB_FilterIIR1_F32 function.

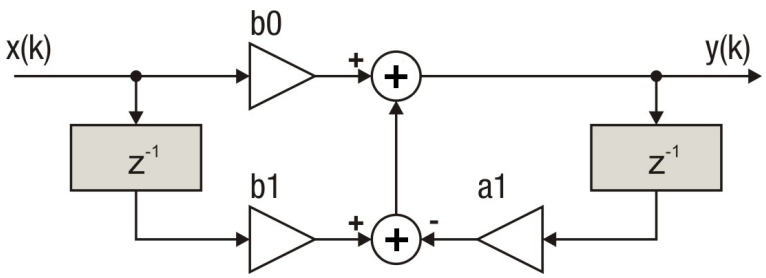

Figure 4-1. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in Figure 4-1 can be designed to meet the requirements for the first order Low (LPF) or High Pass (HPF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab *butter* function. In order to avoid overflow during the calculation of the GDFLIB_FilterIIR1_F32 function, filter coefficients must be divided by eight. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a first order filter. Therefore, the calculation of coefficients can be done using Matlab as follows:

```
freq_cut = 100;
               T_sampling  = 100e-6;

               [b,a]=butter(1,[freq_cut*T_sampling*2],'low');
               sys=tf(b,a,T_sampling);
               bode(sys)

               f32B0 = b(1)/8;
               f32B1 = b(2)/8;
               f32A1 = a(2)/8;
               disp('Coefficients for GDFLIB_FilterIIR1 function:');
               disp(['f32B0 = FRAC32 (' num2str(f32B0) ')']);
               disp(['f32B1 = FRAC32 (' num2str(f32B1) ')']);
               disp(['f32A1 = FRAC32 (' num2str(f32A1) ')']);
```

Note

The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR1_DEFAULT_F32

macro during instance declaration or by calling the GDFLIB_FilterIIR1Init_F32 function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR1_F32 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

4.8.5 Re-entrancy

The function is re-entrant.

4.8.6 Code Example

```
#include "gdflib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR1_T_F32 f32trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F32;

void main(void)
{
    // input value = 0.25
    f32In = FRAC32 (0.25);

    // filter coefficients (LPF 100Hz, Ts=100e-6)
    f32trMyIIR1.trFiltCoeff.f32B0 = FRAC32 (0.030468747091254/8);
    f32trMyIIR1.trFiltCoeff.f32B1 = FRAC32 (0.030468747091254/8);
    f32trMyIIR1.trFiltCoeff.f32A1 = FRAC32 (-0.939062505817492/8);

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init_F32 (&f32trMyIIR1);
    f32Out = GDFLIB_FilterIIR1_F32 (f32In, &f32trMyIIR1);

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init (&f32trMyIIR1, Define F32);
    f32Out = GDFLIB_FilterIIR1 (f32In, &f32trMyIIR1, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x00F99998 ~ FRAC32(0.007617)
    GDFLIB_FilterIIR1Init (&f32trMyIIR1);
    f32Out = GDFLIB_FilterIIR1 (f32In, &f32trMyIIR1);
}
```

4.9 Function GDFLIB_FilterIIR1Init_F16

This function initializes the First order IIR filter buffers.

4.9.1 Declaration

```
void GDFLIB_FilterIIR1Init_F16(GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

4.9.2 Arguments

Table 4-9. GDFLIB_FilterIIR1Init_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	**input, output**	Pointer to filter structure with filter buffer and filter parameters.

4.9.3 Return

Function returns no value.

4.9.4 Description

This function clears the internal buffers of a first order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR1_F16 unless periodic clearing of filter buffers is required.

4.9.5 Re-entrancy

The function is re-entrant.

4.9.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_F16 f16trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F16;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_F16 (&f16trMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init (&f16trMyIIR1, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR1Init (&f16trMyIIR1);
}
```

4.10 Function GDFLIB_FilterIIR1_F16

This function implements the first order IIR filter.

4.10.1 Declaration

```
tFrac16 GDFLIB_FilterIIR1_F16(tFrac16 f16In, GDFLIB_FILTER_IIR1_T_F16 *const pParam);
```

4.10.2 Arguments

Table 4-10. GDFLIB_FilterIIR1_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Value of input signal to be filtered in step (k). The value is a 16-bit number in the 1.15 fractional format.
GDFLIB_FILTER_IIR1_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

4.10.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the filtered value of the input signal in step (k).

4.10.4 Description

This function calculates the first order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR1_Eq1**

where N denotes the filter order. The first order IIR filter in the Z-domain is therefore given from equation GDFLIB_FilterIIR1_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Equation **GDFLIB_FilterIIR1_Eq2**

In order to implement the first order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by equation GDFLIB_FilterIIR1_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Equation **GDFLIB_FilterIIR1_Eq3**

Equation GDFLIB_FilterIIR1_Eq3 represents a Direct Form I implementation of a first order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the

signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

Because there are two delay buffers necessary for both DF-I and DF-II implementation of the first order IIR filter, the DF-I implementation was chosen to be used in the GDFLIB_FilterIIR1_F16 function.

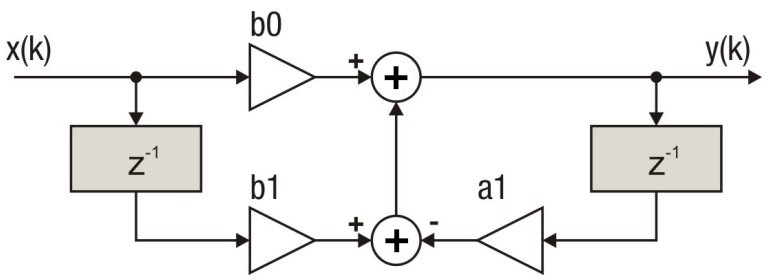

Figure 4-2. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in Figure 4-2 can be designed to meet the requirements for the first order Low (LPF) or High Pass (HPF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab *butter* function. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a first order filter. Therefore, the calculation of coefficients can be done using Matlab as follows:

```
freq_cut     = 100;
               T_sampling  = 100e-6;

               [b,a]=butter(1,[freq_cut*T_sampling*2],'low');
               sys=tf(b,a,T_sampling);
               bode(sys)

               f16B0 = b(1);
               f16B1 = b(2);
               f16A1 = a(2);
               disp('Coefficients for GDFLIB_FilterIIR1 function:');
               disp(['f16B0 = FRAC16 (' num2str(f16B0) ')']);
               disp(['f16B1 = FRAC16 (' num2str(f16B1) ')']);
               disp(['f16A1 = FRAC16 (' num2str(f16A1) ')']);
```

Note

The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR1_DEFAULT_F16 macro during instance declaration or by calling the GDFLIB_FilterIIR1Init_F16 function.

4.10.5 Re-entrancy

The function is re-entrant.

4.10.6 Code Example

```
#include "gdflib.h"

tFrac16 f16In;
tFrac16 f16Out;

GDFLIB_FILTER_IIR1_T_F16 f16trMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16 (0.25);

    // filter coefficients (LPF 100Hz, Ts=100e-6)
    f16trMyIIR1.trFiltCoeff.f16B0 = FRAC16 (0.030468747091254);
    f16trMyIIR1.trFiltCoeff.f16B1 = FRAC16 (0.030468747091254);
    f16trMyIIR1.trFiltCoeff.f16A1 = FRAC16 (-0.939062505817492);

    // output should be 0x07C8 ~ FRAC16(0.0608)
    GDFLIB_FilterIIR1Init_F16 (&f16trMyIIR1);
    f16Out = GDFLIB_FilterIIR1_F16 (f16In, &f16trMyIIR1);

    // output should be 0x07C8 ~ FRAC16(0.0608)
    GDFLIB_FilterIIR1Init (&f16trMyIIR1, Define F16);
    f16Out = GDFLIB_FilterIIR1 (f16In, &f16trMyIIR1, Define F16);

    // ##################################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##################################################################

    // output should be 0x07C8 ~ FRAC16(0.0608)
    GDFLIB_FilterIIR1Init (&f16trMyIIR1);
    f16Out = GDFLIB_FilterIIR1 (f16In, &f16trMyIIR1);
}
```

4.11 Function GDFLIB_FilterIIR1Init_FLT

This function initializes the First order IIR filter buffers.

4.11.1 Declaration

```
void GDFLIB_FilterIIR1Init_FLT(GDFLIB_FILTER_IIR1_T_FLT *const pParam);
```

4.11.2 Arguments

Table 4-11. GDFLIB_FilterIIR1Init_FLT arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR1_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

4.11.3 Return

Function returns no value.

4.11.4 Description

This function clears the internal buffers of a first order IIR filter. The function shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR1_FLT unless periodic clearing of filter buffers is required.

4.11.5 Re-entrancy

The function is re-entrant.

4.11.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR1_T_FLT flttrMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_FLT;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR1Init_FLT (&flttrMyIIR1);

    // function returns no value
    GDFLIB_FilterIIR1Init (&flttrMyIIR1, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
```

```
    // ##############################################################

    // function returns no value
    GDFLIB_FilterIIR1Init (&flttrMyIIR1);
}
```

4.12 Function GDFLIB_FilterIIR1_FLT

This function implements the first order IIR filter.

4.12.1 Declaration

```
tFloat GDFLIB_FilterIIR1_FLT(tFloat fltIn, GDFLIB_FILTER_IIR1_T_FLT *const pParam);
```

4.12.2 Arguments

Table 4-12. GDFLIB_FilterIIR1_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Value of the input signal to be filtered in step (k). Input is a 32-bit number that contains a single precision floating point value.
GDFLIB_FILTER_IIR1_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with a filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

4.12.3 Return

The function returns a 32-bit value in single precision floating point format, representing the filtered value of the input signal in step (k).

4.12.4 Description

This function calculates the first order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR1_Eq1**

where N denotes the filter order. The first order IIR filter in the Z-domain is therefore given from equation GDFLIB_FilterIIR1_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

Equation **GDFLIB_FilterIIR1_Eq2**

In order to implement the first order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by equation GDFLIB_FilterIIR1_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) - a_1 y(k-1)$$

Equation **GDFLIB_FilterIIR1_Eq3**

where: x[k] is the input signal, y[k] is the output signal, a_i and b_i are the filter coefficients. Equation GDFLIB_FilterIIR1_Eq3 represents a Direct Form I implementation of a first order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into lower order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

Because there are two delay buffers necessary for both DF-I and DF-II implementation of the first order IIR filter, the DF-I implementation was chosen to be used in the GDFLIB_FilterIIR1_FLT function.

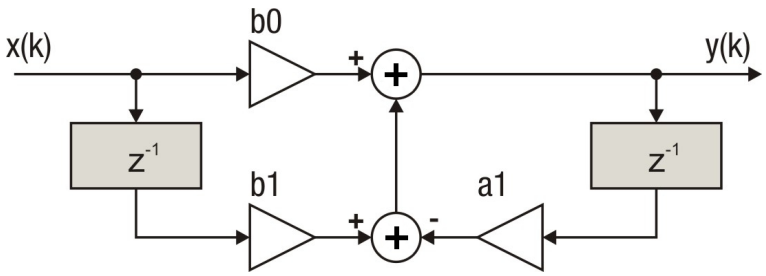

Figure 4-3. Direct Form 1 first order IIR filter

The coefficients of the filter depicted in Figure 4-3 can be designed to meet the requirements for the first order Low Pass (LPF) or High Pass (HPF) filters. Filter coefficients can be calculated using various tools, for example, the Matlab *butter* function.

Note

To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the Matlab precise reference model.

CAUTION

The filter delay line includes two delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR1_DEFAULT_FLT macro during instance declaration or by calling the GDFLIB_FilterIIR1Init_FLT function.

4.12.5 Re-entrancy

The function is re-entrant.

4.12.6 Code Example

```
#include "gdflib.h"

tFloat fltIn;
tFloat fltOut;

GDFLIB_FILTER_IIR1_T_FLT flttrMyIIR1 = GDFLIB_FILTER_IIR1_DEFAULT_FLT;

void main(void)
{
    // input value = 0.25
    fltIn = (tFloat)(0.25);

    // filter coefficients (LPF 100Hz)
```

```
    flttrMyIIR1.trFiltCoeff.fltB0 = (tFloat)(0.030468747091254);
    flttrMyIIR1.trFiltCoeff.fltB1 = (tFloat)(0.030468747091254);
    flttrMyIIR1.trFiltCoeff.fltA1 = (tFloat)(-0.939062505817492);

    // output should be 0.007617
    GDFLIB_FilterIIR1Init_FLT (&flttrMyIIR1);
    fltOut = GDFLIB_FilterIIR1_FLT (fltIn, &flttrMyIIR1);

    // output should be 0.007617
    GDFLIB_FilterIIR1Init (&flttrMyIIR1, Define FLT);
    fltOut = GDFLIB_FilterIIR1 (fltIn, &flttrMyIIR1, Define FLT);

    // ##################################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##################################################################

    // output should be 0.007617
    GDFLIB_FilterIIR1Init (&flttrMyIIR1);
    fltOut = GDFLIB_FilterIIR1 (fltIn, &flttrMyIIR1);
}
```

4.13 Function GDFLIB_FilterIIR2Init_F32

This function initializes the Second order IIR filter buffers.

4.13.1 Declaration

```
void GDFLIB_FilterIIR2Init_F32(GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

4.13.2 Arguments

Table 4-13. GDFLIB_FilterIIR2Init_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

4.13.3 Return

Function returns no value.

4.13.4 Description

This function clears the internal buffers of a second order IIR filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR2_F32 unless periodic clearing of filter buffers is required.

4.13.5 Re-entrancy

The function is re-entrant.

4.13.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR2_T_F32 f32trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F32;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_F32 (&f32trMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init (&f32trMyIIR2, Define F32);

    // ##################################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##################################################################

    // function returns no value
    GDFLIB_FilterIIR2Init (&f32trMyIIR2);
}
```

4.14 Function GDFLIB_FilterIIR2_F32

This function implements the second order IIR filter.

4.14.1 Declaration

```
tFrac32 GDFLIB_FilterIIR2_F32(tFrac32 f32In, GDFLIB_FILTER_IIR2_T_F32 *const pParam);
```

4.14.2 Arguments

Table 4-14. GDFLIB_FilterIIR2_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Value of input signal to be filtered in step (k). The value is a 32-bit number in the 1.31 fractional format.
GDFLIB_FILTER_IIR2_T_F32 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

4.14.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the filtered value of the input signal in step (k).

4.14.4 Description

This function calculates the second order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR2_Eq1**

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given from eq. GDFLIB_FilterIIR2_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation **GDFLIB_FilterIIR2_Eq2**

In order to implement the second order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by eq. GDFLIB_FilterIIR2_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Equation **GDFLIB_FilterIIR2_Eq3**

Equation GDFLIB_FilterIIR2_Eq3 represents a Direct Form I implementation of a second order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

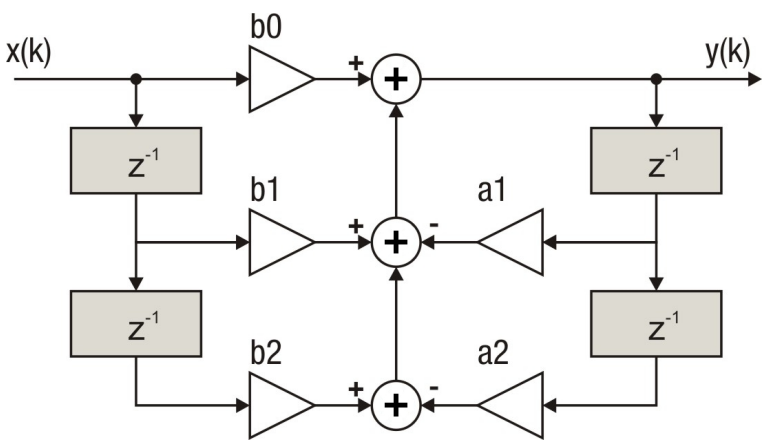

Figure 4-4. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in Figure 4-4 can be designed to meet the requirements for the second order Band Pass (BPF) or Band Stop (BSF) filters. Filter coefficients can be calculated using various tools, for example the Matlab *butter* function. In order to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F32 function, filter coefficients must be divided by eight. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a second order filter. Therefore, calculation of coefficients can be done using Matlab as follows:

```
freq_bot    = 400;
              freq_top    = 625;
              T_sampling  = 100e-6;

              [b,a]= butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
              sys =tf(b,a,T_sampling);
              bode(sys,[freq_bot:1:freq_top]*2*pi)

              f32B0 = b(1)/8;
              f32B1 = b(2)/8;
              f32B2 = b(3)/8;
              f32A1 = a(2)/8;
              f32A2 = a(3)/8;
              disp (' Coefficients for GDFLIB_FilterIIR2 function :');
              disp ([ 'f32B0 = FRAC32(' num2str( f32B0 ) ')']);
              disp ([ 'f32B1 = FRAC32(' num2str( f32B1 ) ')']);
              disp ([ 'f32B2 = FRAC32(' num2str( f32B2 ) ')']);
              disp ([ 'f32A1 = FRAC32(' num2str( f32A1 ) ')']);
              disp ([ 'f32A2 = FRAC32(' num2str( f32A2 ) ')']);
```

Note

The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR2_DEFAULT_F32 macro during instance declaration or by calling the GDFLIB_FilterIIR2Init_F32 function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F32 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

4.14.5 Re-entrancy

The function is re-entrant.

4.14.6 Code Example

```
#include "gdflib.h"

tFrac32 f32In;
tFrac32 f32Out;

GDFLIB_FILTER_IIR2_T_F32 f32trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F32;

void main(void)
{
    // input value = 0.25
    f32In = FRAC32 (0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
```

```
    f32trMyIIR2.trFiltCoeff.f32B0 = FRAC32 (0.066122101544579/8);
    f32trMyIIR2.trFiltCoeff.f32B1 = FRAC32 (0.0);
    f32trMyIIR2.trFiltCoeff.f32B2 = FRAC32 (-0.066122101544579/8);
    f32trMyIIR2.trFiltCoeff.f32A1 = FRAC32 (-1.776189018043779/8);
    f32trMyIIR2.trFiltCoeff.f32A2 = FRAC32 (0.867755796910841/8);

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init_F32 (&f32trMyIIR2);
    f32Out = GDFLIB_FilterIIR2_F32 (f32In, &f32trMyIIR2);

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init (&f32trMyIIR2, Define F32);
    f32Out = GDFLIB_FilterIIR2 (f32In, &f32trMyIIR2, Define F32);

    // ############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ############################################################

    // output should be 0x021DAC18 ~ FRAC32(0.0165305)
    GDFLIB_FilterIIR2Init (&f32trMyIIR2);
    f32Out = GDFLIB_FilterIIR2 (f32In, &f32trMyIIR2);
}
```

4.15 Function GDFLIB_FilterIIR2Init_F16

This function initializes the Second order IIR filter buffers.

4.15.1 Declaration

```
void GDFLIB_FilterIIR2Init_F16(GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

4.15.2 Arguments

Table 4-15. GDFLIB_FilterIIR2Init_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

4.15.3 Return

Function returns no value.

4.15.4 Description

This function clears the internal buffers of a second order IIR filter. The function shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR2_F16 unless periodic clearing of filter buffers is required.

4.15.5 Re-entrancy

The function is re-entrant.

4.15.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR2_T_F16 f16trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F16;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_F16 (&f16trMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init (&f16trMyIIR2, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // function returns no value
    GDFLIB_FilterIIR2Init (&f16trMyIIR2);
}
```

4.16 Function GDFLIB_FilterIIR2_F16

This function implements the second order IIR filter.

4.16.1 Declaration

```
tFrac16 GDFLIB_FilterIIR2_F16(tFrac16 f16In, GDFLIB_FILTER_IIR2_T_F16 *const pParam);
```

4.16.2 Arguments

Table 4-16. GDFLIB_FilterIIR2_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Value of input signal to be filtered in step (k). The value is a 16-bit number in the 1.15 fractional format.
GDFLIB_FILTER_IIR2_T_F16 *const	pParam	**input, output**	Pointer to the filter structure with a filter buffer and filter parameters.

4.16.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the filtered value of the input signal in step (k).

4.16.4 Description

This function calculates the second order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR2_Eq1**

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given from eq. GDFLIB_FilterIIR2_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation **GDFLIB_FilterIIR2_Eq2**

In order to implement the second order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by eq. GDFLIB_FilterIIR2_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Equation **GDFLIB_FilterIIR2_Eq3**

Equation GDFLIB_FilterIIR2_Eq3 represents a Direct Form I implementation of a second order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into low order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

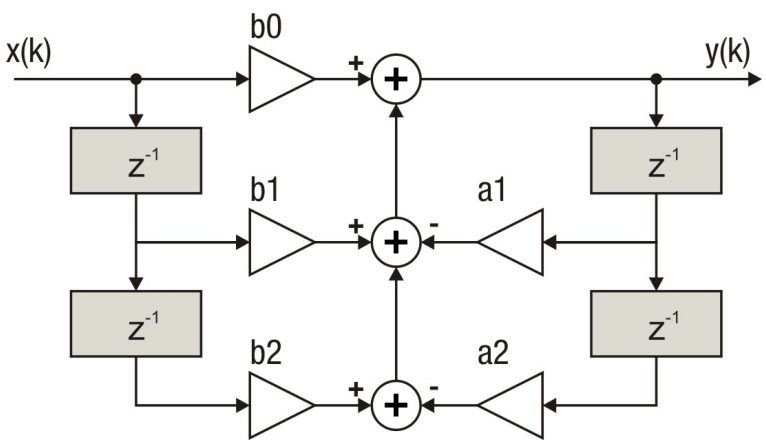

Figure 4-5. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in Figure 4-5 can be designed to meet the requirements for the second order Band Pass (BPF) or Band Stop (BSF) filters. Filter coefficients can be calculated using various tools, for example the Matlab *butter* function. In order to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F16 function, filter coefficients must be divided by eight. The coefficient quantization error due to finite precision arithmetic can be neglected in the case of a second order filter. Therefore, calculation of coefficients can be done using Matlab as follows:

```
freq_bot     = 400;
               freq_top     = 625;
               T_sampling   = 100e-6;

               [b,a]= butter(1,[freq_bot freq_top]*T_sampling *2, 'bandpass');
               sys =tf(b,a,T_sampling);
               bode(sys,[freq_bot:1:freq_top]*2*pi)

               f16B0 = b(1)/8;
               f16B1 = b(2)/8;
               f16B2 = b(3)/8;
               f16A1 = a(2)/8;
               f16A2 = a(3)/8;
               disp (' Coefficients for GDFLIB_FilterIIR2 function :');
               disp ([ 'f16B0 = FRAC16(' num2str( f16B0 ) ')']);
               disp ([ 'f16B1 = FRAC16(' num2str( f16B1 ) ')']);
               disp ([ 'f16B2 = FRAC16(' num2str( f16B2 ) ')']);
               disp ([ 'f16A1 = FRAC16(' num2str( f16A1 ) ')']);
               disp ([ 'f16A2 = FRAC16(' num2str( f16A2 ) ')']);
```

Note

The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR2_DEFAULT_F16 macro during instance declaration or by calling the GDFLIB_FilterIIR2Init_F16 function.

CAUTION

Because of fixed point implementation, and to avoid overflow during the calculation of the GDFLIB_FilterIIR2_F16 function, filter coefficients must be divided by eight. Function output is internally multiplied by eight to correct the coefficient scaling.

4.16.5 Re-entrancy

The function is re-entrant.

4.16.6 Code Example

```
#include "gdflib.h"

tFrac16 f16In;
tFrac16 f16Out;

GDFLIB_FILTER_IIR2_T_F16 f16trMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_F16;

void main(void)
{
    // input value = 0.25
    f16In = FRAC16 (0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
```

```
    f16trMyIIR2.trFiltCoeff.f16B0 = FRAC16 (0.066122101544579/8);
    f16trMyIIR2.trFiltCoeff.f16B1 = FRAC16 (0.0);
    f16trMyIIR2.trFiltCoeff.f16B2 = FRAC16 (-0.066122101544579/8);
    f16trMyIIR2.trFiltCoeff.f16A1 = FRAC16 (-1.776189018043779/8);
    f16trMyIIR2.trFiltCoeff.f16A2 = FRAC16 (0.867755796910841/8);

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init_F16 (&f16trMyIIR2);
    f16Out = GDFLIB_FilterIIR2_F16 (f16In, &f16trMyIIR2);

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init (&f16trMyIIR2, Define F16);
    f16Out = GDFLIB_FilterIIR2 (f16In, &f16trMyIIR2, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x021D ~ FRAC16(0.01651)
    GDFLIB_FilterIIR2Init (&f16trMyIIR2);
    f16Out = GDFLIB_FilterIIR2 (f16In, &f16trMyIIR2);
}
```

4.17 Function GDFLIB_FilterIIR2Init_FLT

This function initializes the Second order IIR filter buffers.

4.17.1 Declaration

```
void GDFLIB_FilterIIR2Init_FLT(GDFLIB_FILTER_IIR2_T_FLT *const pParam);
```

4.17.2 Arguments

Table 4-17. GDFLIB_FilterIIR2Init_FLT arguments

Type	Name	Direction	Description
GDFLIB_FILTER_IIR2_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

4.17.3 Return

Function returns no value.

4.17.4 Description

This function clears the internal buffers of a second order IIR filter. The function shall be called after filter parameter initialization and whenever the filter initialization is required.

Note

This function shall not be called together with GDFLIB_FilterIIR2_FLT unless periodic clearing of filter buffers is required.

4.17.5 Re-entrancy

The function is re-entrant.

4.17.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_IIR2_T_FLT fltrMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_FLT;

void main(void)
{
    // function returns no value
    GDFLIB_FilterIIR2Init_FLT (&fltrMyIIR2);

    // function returns no value
    GDFLIB_FilterIIR2Init (&fltrMyIIR2, Define FLT);

    // ##################################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##################################################################

    // function returns no value
    GDFLIB_FilterIIR2Init (&fltrMyIIR2);
}
```

4.18 Function GDFLIB_FilterIIR2_FLT

This function implements the second order IIR filter.

4.18.1 Declaration

```
tFloat GDFLIB_FilterIIR2_FLT(tFloat fltIn, GDFLIB_FILTER_IIR2_T_FLT *const pParam);
```

4.18.2 Arguments

Table 4-18. GDFLIB_FilterIIR2_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Value of the input signal to be filtered in step (k). Input is a 32-bit number that contains a single precision floating point value.
GDFLIB_FILTER_IIR2_T_FLT *const	pParam	**input, output**	Pointer to a filter structure with a filter buffer and filter parameters. Arguments of the structure contain single precision floating point values.

4.18.3 Return

The function returns a 32-bit value in single precision floating point format, representing the filtered value of the input signal in step (k).

4.18.4 Description

This function calculates the second order infinite impulse (IIR) filter. The IIR filters are also called recursive filters because both the input and the previously calculated output values are used for calculation of the filter equation in each step. This form of feedback enables transfer of the energy from the output to the input, which theoretically leads to an infinitely long impulse response (IIR).

A general form of the IIR filter expressed as a transfer function in the Z-domain is described as follows:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

Equation **GDFLIB_FilterIIR2_Eq1**

where N denotes the filter order. The second order IIR filter in the Z-domain is therefore given from eq. GDFLIB_FilterIIR2_Eq1 as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Equation **GDFLIB_FilterIIR2_Eq2**

In order to implement the second order IIR filter on a microcontroller, the discrete time domain representation of the filter, described by eq. GDFLIB_FilterIIR2_Eq2, must be transformed into a time difference equation as follows:

$$y(k) = b_0 x(k) + b_1 x(k-1) + b_2 x(k-2) - a_1 y(k-1) - a_2 y(k-2)$$

Equation **GDFLIB_FilterIIR2_Eq3**

where: x[k] is the input signal, y[k] is the output signal, a_i and b_i are the filter coefficients. Equation GDFLIB_FilterIIR2_Eq3 represents a Direct Form I implementation of a second order IIR filter. It is well known that Direct Form I (DF-I) and Direct Form II (DF-II) implementations of an IIR filter are generally sensitive to parameter quantization if a finite precision arithmetic is considered. This, however, can be neglected when the filter transfer function is broken down into lower order sections, i.e. first or second order. The main difference between DF-I and DF-II implementations of an IIR filter is in the number of delay buffers and in the number of guard bits required to handle the potential overflow. The DF-II implementation requires less delay buffers than DF-I, hence less data memory is utilized. On the other hand, since the poles come first in the DF-II realization, the signal entering the state delay-line typically requires a larger dynamic range than the output signal y(k). Therefore, overflow can occur at the delay-line input of the DF-II implementation, unlike in the DF-I implementation.

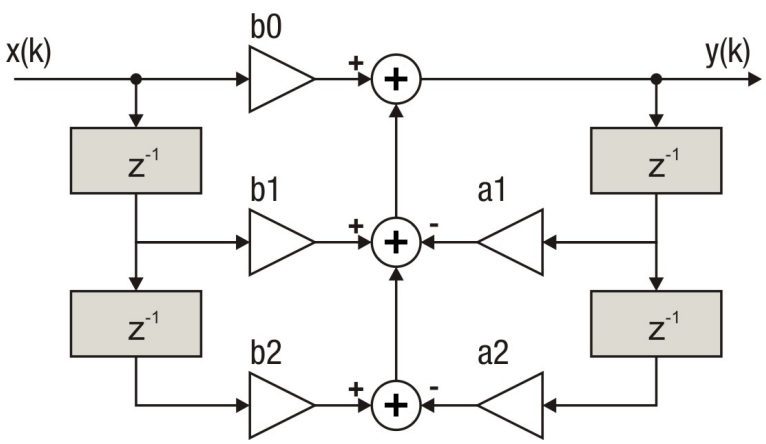

Figure 4-6. Direct Form 1 second order IIR filter

The coefficients of the filter depicted in Figure 4-6 can be designed to meet the requirements for the second order Band Pass (BPF) or Band Stop (BSF) filters. Filter coefficients can be calculated using various tools, for example the Matlab *butter* function.

Note

To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is

replaced by output from the previous calculation step of the Matlab precise reference model.

CAUTION

The filter delay line includes four delay buffers which should be reset after filter initialization. This can be done by assigning to the filter instance a GDFLIB_FILTER_IIR2_DEFAULT_FLT macro during instance declaration or by calling the GDFLIB_FilterIIR2Init_FLT function.

4.18.5 Re-entrancy

The function is re-entrant.

4.18.6 Code Example

```
#include "gdflib.h"

tFloat fltIn;
tFloat fltOut;

GDFLIB_FILTER_IIR2_T_FLT flttrMyIIR2 = GDFLIB_FILTER_IIR2_DEFAULT_FLT;

void main(void)
{
    // input value = 0.25
    fltIn = (tFloat)(0.25);

    // filter coefficients (BPF 400-625Hz, Ts=100e-6)
    flttrMyIIR2.trFiltCoeff.fltB0 = (tFloat)(0.066122101544579);
    flttrMyIIR2.trFiltCoeff.fltB1 = (tFloat)(0.0);
    flttrMyIIR2.trFiltCoeff.fltB2 = (tFloat)(-0.066122101544579);
    flttrMyIIR2.trFiltCoeff.fltA1 = (tFloat)(-1.776189018043779);
    flttrMyIIR2.trFiltCoeff.fltA2 = (tFloat)(0.867755796910841);

    // output should be 0.01651
    GDFLIB_FilterIIR2Init_FLT (&flttrMyIIR2);
    fltOut = GDFLIB_FilterIIR2_FLT (fltIn, &flttrMyIIR2);

    // output should be 0.01651
    GDFLIB_FilterIIR2Init (&flttrMyIIR2, Define FLT);
    fltOut = GDFLIB_FilterIIR2 (fltIn, &flttrMyIIR2, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.01651
    GDFLIB_FilterIIR2Init (&flttrMyIIR2);
    fltOut = GDFLIB_FilterIIR2 (fltIn, &flttrMyIIR2);
}
```

4.19 Function GDFLIB_FilterMAInit_F32

This function clears the internal filter accumulator.

4.19.1 Declaration

```
void GDFLIB_FilterMAInit_F32(GDFLIB_FILTER_MA_T_F32 *pParam);
```

4.19.2 Arguments

Table 4-19. GDFLIB_FilterMAInit_F32 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_F32 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and filter parameters.

4.19.3 Return

void

4.19.4 Description

This function clears the internal accumulator of a moving average filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \le u16NSamples \le 31$$

Equation **GDFLIB_FilterMA_Eq1**

Note

This function shall not be called together with GDFLIB_FilterMA_F32 unless periodic clearing of filter buffers is required.

4.19.5 Re-entrancy

The function is re-entrant.

4.19.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;

void main(void)
{
  GDFLIB_FilterMAInit_F32 (&f32trMyMA);

  GDFLIB_FilterMAInit (&f32trMyMA,Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  GDFLIB_FilterMAInit (&f32trMyMA);
}
```

4.20 Function GDFLIB_FilterMA_F32

This function implements a moving average recursive filter.

4.20.1 Declaration

```
tFrac32 GDFLIB_FilterMA_F32(tFrac32 f32In, GDFLIB_FILTER_MA_T_F32 *pParam);
```

4.20.2 Arguments

Table 4-20. GDFLIB_FilterMA_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Value of input signal to be filtered in step (k). The value is a 32-bit number in the Q1.31 format.
GDFLIB_FILTER_MA_T_F32 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and filter parameters.

4.20.3 Return

The function returns a 32-bit value in format Q1.31, representing the filtered value of the input signal in step (k).

4.20.4 Description

This function calculates a recursive form of an average filter. The filter calculation consists of the following equations:

$$acc(k) = acc(k-1) + x(k)$$

Equation **GDFLIB_FilterMA_Eq2**

$$y(k) = \frac{acc(k)}{n_p}$$

Equation **GDFLIB_FilterMA_Eq3**

$$acc(k) \leftarrow acc(k) - y(k)$$

Equation **GDFLIB_FilterMA_Eq4**

where x(k) is the actual value of the input signal, acc(k) is the internal filter accumulator, y(k) is the actual filter output and n_p is the number of points in the filtered window. The size of the filter window (number of filtered points) shall be defined prior to this function

call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \le u16NSamples \le 31$$

Equation **GDFLIB_FilterMA_Eq5**

Note

The size of the filter window (number of filtered points) must be defined prior to this function call and must be equal to or greater than 0, and equal to or smaller than 31 (0 < u16NSamples <31).

4.20.5 Re-entrancy

The function is re-entrant.

4.20.6 Code Example

```
#include "gdflib.h"

tFrac32 f32Input;
tFrac32 f32Output;

GDFLIB_FILTER_MA_T_F32 f32trMyMA = GDFLIB_FILTER_MA_DEFAULT_F32;

void main(void)
{
  // input value = 0.25
  f32Input  = FRAC32 (0.25);

  // filter window = 2^5 = 32 samples
  f32trMyMA.u16NSamples   = 5;
  GDFLIB_FilterMAInit_F32 (&f32trMyMA);

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA_F32 (f32Input,&f32trMyMA);

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA (f32Input,&f32trMyMA,Define F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x1000000 = FRAC32(0.0078125)
  f32Output = GDFLIB_FilterMA (f32Input,&f32trMyMA);
}
```

4.21 Function GDFLIB_FilterMAInit_F16

This function clears the internal filter accumulator.

4.21.1 Declaration

```
void GDFLIB_FilterMAInit_F16(GDFLIB_FILTER_MA_T_F16 *pParam);
```

4.21.2 Arguments

Table 4-21. GDFLIB_FilterMAInit_F16 arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_F16 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and filter parameters.

4.21.3 Return

void

4.21.4 Description

This function clears the internal accumulator of a moving average filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \le u16NSamples \le 15$$

Equation **GDFLIB_FilterMA_Eq1**

Note

This function shall not be called together with GDFLIB_FilterMA_F16 unless periodic clearing of filter buffers is required.

4.21.5 Re-entrancy

The function is re-entrant.

4.21.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
  GDFLIB_FilterMAInit_F16 (&f16trMyMA);

  GDFLIB_FilterMAInit (&f16trMyMA,Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  GDFLIB_FilterMAInit (&f16trMyMA);
}
```

4.22 Function GDFLIB_FilterMA_F16

This function implements a moving average recursive filter.

4.22.1 Declaration

```
tFrac16 GDFLIB_FilterMA_F16(tFrac16 f16In, GDFLIB_FILTER_MA_T_F16 *pParam);
```

4.22.2 Arguments

Table 4-22. GDFLIB_FilterMA_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Value of input signal to be filtered in step (k). The value is a 16-bit number in the Q1.15 format.
GDFLIB_FILTER_MA_T_F16 *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and filter parameters.

4.22.3 Return

The function returns a 16-bit value in format Q1.15, representing the filtered value of the input signal in step (k).

4.22.4 Description

This function calculates a recursive form of an average filter. The filter calculation consists of the following equations:

$$acc(k) = acc(k-1) + x(k)$$

Equation **GDFLIB_FilterMA_Eq2**

$$y(k) = \frac{acc(k)}{n_p}$$

Equation **GDFLIB_FilterMA_Eq3**

$$acc(k) \leftarrow acc(k) - y(k)$$

Equation **GDFLIB_FilterMA_Eq4**

where x(k) is the actual value of the input signal, acc(k) is the internal filter accumulator, y(k) is the actual filter output and n_p is the number of points in the filtered window. The size of the filter window (number of filtered points) shall be defined prior to this function

call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points as a power of 2 as follows:

$$n_p = 2^{u16NSamples} \quad 0 \le u16NSamples \le 8$$

Equation **GDFLIB_FilterMA_Eq5**

Note

The size of the filter window (number of filtered points) must be defined prior to this function call and must be equal to or greater than 0, and equal to or smaller than 8 (0 < u16NSamples <8). In case the filter window size is greater than 8 the output error is out of range.

4.22.5 Re-entrancy

The function is re-entrant.

4.22.6 Code Example

```
#include "gdflib.h"

tFrac16 f16Input;
tFrac16 f16Output;

GDFLIB_FILTER_MA_T_F16 f16trMyMA = GDFLIB_FILTER_MA_DEFAULT_F16;

void main(void)
{
  // input value = 0.25
  f16Input  = FRAC16 (0.25);

  // filter window = 2^3 = 8 samples
  f16trMyMA.u16NSamples   = 3;
  GDFLIB_FilterMAInit_F16 (&f16trMyMA);

  // output should be 0x0400 = FRAC16(0.03125)
  f16Output = GDFLIB_FilterMA_F16 (f16Input,&f16trMyMA);

  // output should be 0x0400 = FRAC16(0.03125)
  f16Output = GDFLIB_FilterMA (f16Input,&f16trMyMA,Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x0400 = FRAC16(0.03125)
  f16Output = GDFLIB_FilterMA (f16Input,&f16trMyMA);
}
```

4.23 Function GDFLIB_FilterMAInit_FLT

This function clears the internal filter accumulator.

4.23.1 Declaration

```
void GDFLIB_FilterMAInit_FLT(GDFLIB_FILTER_MA_T_FLT *pParam);
```

4.23.2 Arguments

Table 4-23. GDFLIB_FilterMAInit_FLT arguments

Type	Name	Direction	Description
GDFLIB_FILTER_MA_T_FLT *	pParam	**input, output**	Pointer to a filter structure with a filter accumulator and filter parameters.

4.23.3 Return

void

4.23.4 Description

This function clears the internal accumulator of a moving average filter. It shall be called after filter parameter initialization and whenever the filter initialization is required.

The size of the filter window (number of filtered points) shall be defined prior to this function call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points, as a power of 2, as follows:

$$n_p = 2^{u16NSamples} \quad 0 \le u16NSamples \le 31$$

Equation **GDFLIB_FilterMA_Eq1**

Note

This function shall not be called together with GDFLIB_FilterMA_FLT unless periodic clearing of filter buffers is required.

4.23.5 Re-entrancy

The function is re-entrant.

4.23.6 Code Example

```
#include "gdflib.h"

GDFLIB_FILTER_MA_T_FLT flttrMyMA = GDFLIB_FILTER_MA_DEFAULT_FLT;

void main(void)
{
  GDFLIB_FilterMAInit_FLT (&flttrMyMA);

  GDFLIB_FilterMAInit (&flttrMyMA,Define FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  GDFLIB_FilterMAInit (&flttrMyMA);
}
```

4.24 Function GDFLIB_FilterMA_FLT

This function implements a moving average recursive filter.

4.24.1 Declaration

```
tFloat GDFLIB_FilterMA_FLT(tFloat fltIn, GDFLIB_FILTER_MA_T_FLT *pParam);
```

4.24.2 Arguments

Table 4-24. GDFLIB_FilterMA_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Value of the input signal to be filtered in step (k). The value is a single precision floating point data type.
GDFLIB_FILTER_MA_T_FLT *	pParam	**input, output**	Pointer to the filter structure with a filter accumulator and filter parameters.

4.24.3 Return

The function returns a single precision floating point value, representing the filtered value of the input signal in step (k).

4.24.4 Description

This function calculates a recursive form of an average filter. The filter calculation consists of the following equations:

$$acc(k) = acc(k-1) + x(k)$$

Equation **GDFLIB_FilterMA_Eq2**

$$y(k) = \frac{acc(k)}{n_p}$$

Equation **GDFLIB_FilterMA_Eq3**

$$acc(k) \leftarrow acc(k) - y(k)$$

Equation **GDFLIB_FilterMA_Eq4**

where x(k) is the actual value of the input signal, acc(k) is the internal filter accumulator, y(k) is the actual filter output and n_p is the number of points in the filtered window. The size of the filter window (number of filtered points) shall be defined prior to this function

call. The number of the filtered points is defined by assigning a value to the u16NSamples variable stored within the filter structure. This number represents the number of filtered points, as a power of 2, as follows:

$$n_p = 2^{u16NSamples} \quad 0 \le u16NSamples \le 31$$

Equation **GDFLIB_FilterMA_Eq5**

Note

The size of the filter window (number of filtered points) must be defined prior to this function call and must be equal to or greater than 0, and equal to or smaller than 31 (0 < u16NSamples < 31).

4.24.5 Re-entrancy

The function is re-entrant.

4.24.6 Code Example

```
#include "gdflib.h"

tFloat fltInput;
tFloat fltOutput;

GDFLIB_FILTER_MA_T_FLT flttrMyMA = GDFLIB_FILTER_MA_DEFAULT_FLT;

void main(void)
{
  // input value = 0.25
  fltInput  = (tFloat)(0.25);

  // filter window = 2^5 = 32 samples
  flttrMyMA.u16NSamples   = 5;
  GDFLIB_FilterMAInit_FLT (&flttrMyMA);

  // output should be 0.0078125
  fltOutput = GDFLIB_FilterMA_FLT (fltInput,&flttrMyMA);

  // output should be 0.0078125
  fltOutput = GDFLIB_FilterMA (fltInput,&flttrMyMA,Define FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be 0.0078125
  fltOutput = GDFLIB_FilterMA (fltInput,&flttrMyMA);
}
```

4.25 Function GFLIB_Acos_F32

This function implements polynomial approximation of arccosine function.

4.25.1 Declaration

```
tFrac32 GFLIB_Acos_F32(tFrac32 f32In, const GFLIB_ACOS_T_F32 *const pParam);
```

4.25.2 Arguments

Table 4-25. GFLIB_Acos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains a value between [-1,1).
const GFLIB_ACOS_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.25.3 Return

The function returns arccos(f32In)/ π as a fixed point 32-bit number, normalized between [0,1).

4.25.4 Description

The GFLIB_Acos_F32 function provides a computational method for calculation of the standard inverse trigonometric *arccosine* function arccos(x), using the piece-wise polynomial approximation. Function arccos(x) takes the ratio of the length of the adjacent side to the length of the hypotenuse and returns the angle.

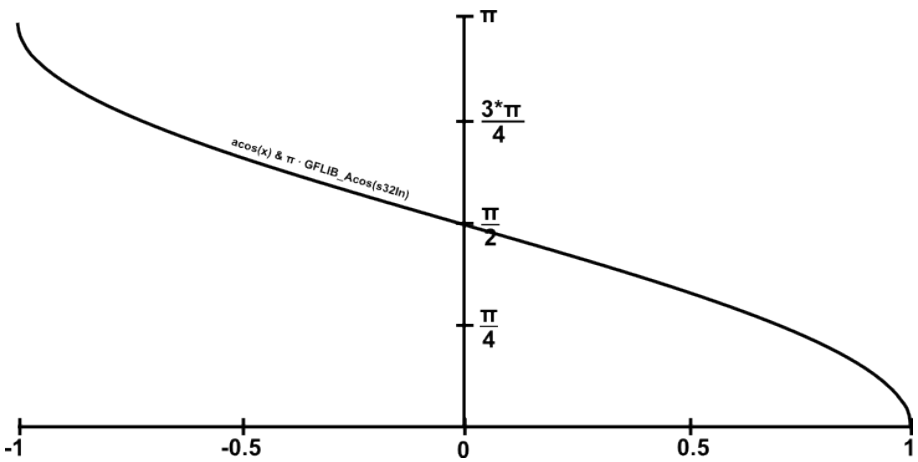

Figure 4-7. Course of the function GFLIB_Acos

The computational algorithm uses the symmetry of the arccos(x) function around the point (0, π/2), which allows for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- y[-1, 0) is the arccos(x) function value in the interval [-1, 0)
- y[0, 1) is the arccos(x) function value in the interval [0, 1)

Additionally, because the arccos(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\arccos\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arccos\left(\sqrt{x}\right)$$

Equation **GFLIB_Acos_Eq2**

In this way, the computation of the arccos(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

For the interval (0, 0.5], the algorithm uses a polynomial approximation as follows:

$$f32Dump = a_0 \bullet f32In^4 + a_1 \bullet f32In^3 + a_2 \bullet f32In^2 + a_3 \bullet f32In + a_4$$

Equation **GFLIB_Acos_Eq3**

$$\arccos\left(f32In\right) = \begin{cases} -f32Dump & \text{if} \quad -1 \le f32In \le 0 \\ f32Dump & \text{if} \quad 0 \le f32In < 1 \end{cases}$$

Equation **GFLIB_Acos_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-26.

Table 4-26. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0, 1/2)	91918582	66340080	9729967	682829947	12751
<1/2, 1)	-52453538	-36708911	-15136243	-964576326	1073652175

The implementation of the GFLIB_Acos_F32 is almost the same as in the function GFLIB_Asin_F32. However, the output of the GFLIB_Acos_F32 is corrected as follows:

$$s32Dump = FRAC32\left(0.5\right) - f32Dump$$

Equation **GFLIB_Acos_Eq5**

The polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of 5th order was used for the fitting of each respective sub-interval. The functions *arcsine* and *arccosine* are similar, therefore the GFLIB_Acos_F32 function uses the same polynomial coefficients as he GFLIB_Asin_F32 function.

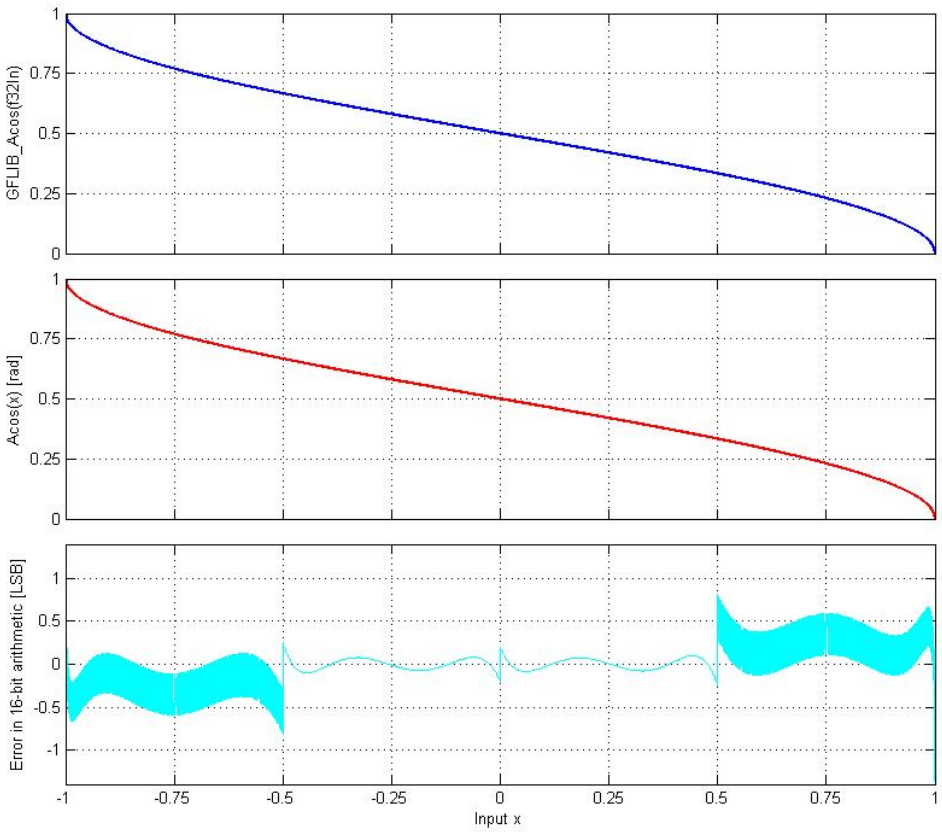

Figure 4-8. acos(x) vs. GFLIB_Acos(s32In)

Note

The output angle is normalized into the range [0,1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Acos_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Acos(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Acos(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default

GFLIB_ACOS_DEFAULT_F32 approximation coefficients are used.

4.25.5 Re-entrancy

The function is re-entrant.

4.25.6 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
  // input f32Input = 0
  f32Input  = FRAC32 (0);

  // output should be 0x400031EF = 0.5 => pi/2
  f32Angle = GFLIB_Acos_F32 (f32Input, GFLIB_ACOS_DEFAULT_F32);

  // output should be 0x400031EF = 0.5 => pi/2
  f32Angle = GFLIB_Acos (f32Input, GFLIB_ACOS_DEFAULT_F32, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x400031EF = 0.5 => pi/2
  f32Angle = GFLIB_Acos (f32Input);
}
```

4.26 Function GFLIB_Acos_F16

This function implements polynomial approximation of arccosine function.

4.26.1 Declaration

```
tFrac16 GFLIB_Acos_F16(tFrac16 f16In, const GFLIB_ACOS_T_F16 *const pParam);
```

4.26.2 Arguments

Table 4-27. GFLIB_Acos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains a value between [-1,1).
const GFLIB_ACOS_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.26.3 Return

The function returns arccos(f16In)/ π as a fixed point 16-bit number, normalized between [0,1).

4.26.4 Description

The GFLIB_Acos_F16 function provides a computational method for calculation of the standard inverse trigonometric *arccosine* function arccos(x), using the piece-wise polynomial approximation. Function arccos(x) takes the ratio of the length of the adjacent side to the length of the hypotenuse and returns the angle.

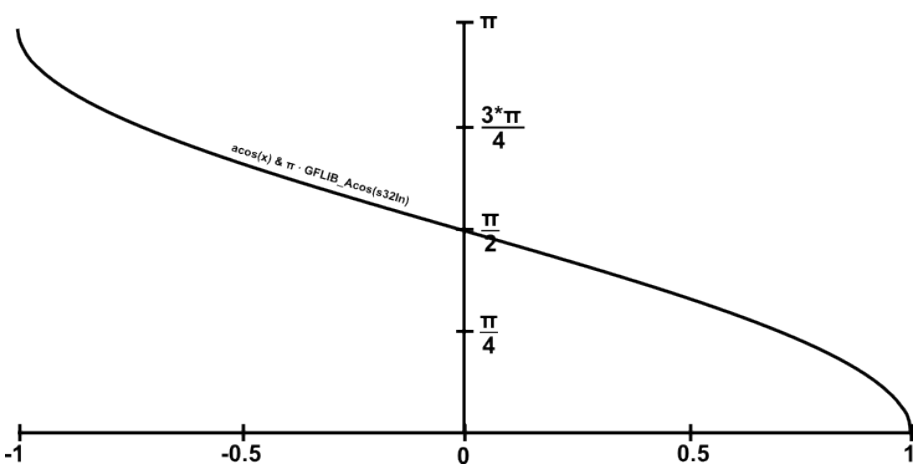

Figure 4-9. Course of the function GFLIB_Acos

The computational algorithm uses the symmetry of the arccos(x) function around the point (0, π/2), which allows for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- y[-1, 0) is the arccos(x) function value in the interval [-1, 0)
- y[0, 1) is the arccos(x) function value in the interval [0, 1)

Additionally, because the arccos(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\arccos\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arccos\left(\sqrt{x}\right)$$

Equation **GFLIB_Acos_Eq2**

In this way, the computation of the arccos(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

For the interval (0, 0.5], the algorithm uses a polynomial approximation as follows:

$$f16Dump = a_0 \bullet f16In^4 + a_1 \bullet f16In^3 + a_2 \bullet f16In^2 + a_3 \bullet f16In + a_4$$

Equation **GFLIB_Acos_Eq3**

$$\arccos\left(f16In\right) = \begin{cases} -f16Dump & \text{if} \quad -1 \le f16In \le 0 \\ f16Dump & \text{if} \quad 0 \le f16In < 1 \end{cases}$$

Equation **GFLIB_Acos_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-28.

Table 4-28. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0, 1/2)	1403	1012	148	10419	0

Table continues on the next page...

Table 4-28. Integer polynomial coefficients for each interval (continued)

<1/2, 1)	-800	-560	-231	-14718	16383

The implementation of the GFLIB_Acos_F16 is almost the same as in the function GFLIB_Asin_F16. However, the output of the GFLIB_Acos_F16 is corrected as follows:

$$s16Dump = \text{FRAC16}(0.5) - f16Dump$$

Equation **GFLIB_Acos_Eq5**

The polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of 5th order was used for the fitting of each respective sub-interval. The functions *arcsine* and *arccosine* are similar, therefore the GFLIB_Acos_F16 function uses the same polynomial coefficients as the GFLIB_Asin_F16 function.

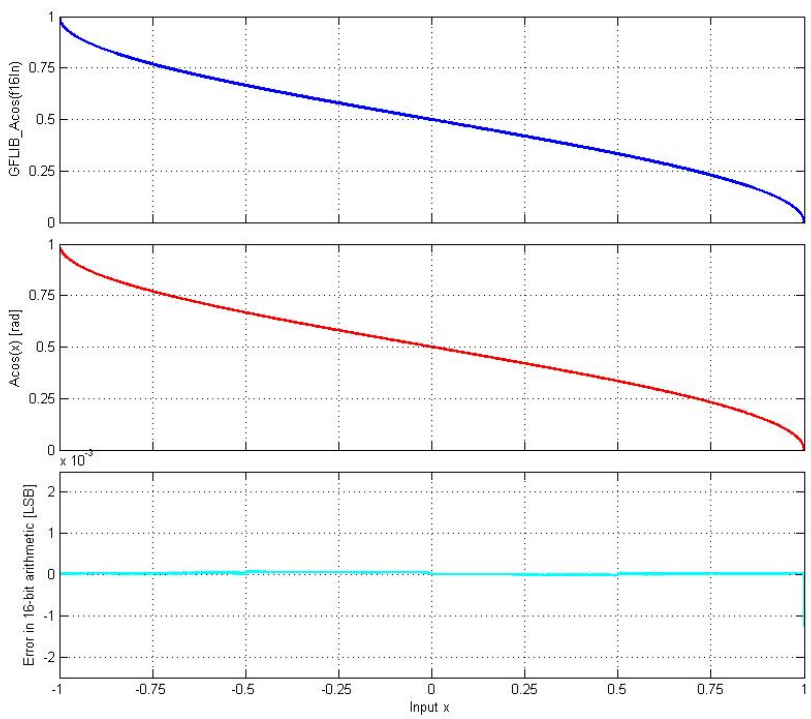

Figure 4-10. acos(x) vs. GFLIB_Acos(f16In)

Note

The output angle is normalized into the range [0,1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Acos_F16(f16In, &pParam)), where the &pParam is pointer to

approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Acos(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Acos(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ACOS_DEFAULT_F16 approximation coefficients are used.

4.26.5 Re-entrancy

The function is re-entrant.

4.26.6 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
  // input f16Input = 0
  f16Input  = FRAC16 (0);

  // output should be 0x4000 = 0.5 => pi/2
  f16Angle = GFLIB_Acos_F16 (f16Input, GFLIB_ACOS_DEFAULT_F16);

  // output should be 0x4000 = 0.5 => pi/2
  f16Angle = GFLIB_Acos (f16Input, GFLIB_ACOS_DEFAULT_F16, Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x4000 = 0.5 => pi/2
  f16Angle = GFLIB_Acos (f16Input);
}
```

4.27 Function GFLIB_Acos_FLT

This function implements square root based polynomial approximation of arccosine function.

4.27.1 Declaration

```
tFloat GFLIB_Acos_FLT(tFloat fltIn, const GFLIB_ACOS_T_FLT *const pParam);
```

4.27.2 Arguments

Table 4-29. GFLIB_Acos_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a 32-bit number that contains a single precision floating point value.
const GFLIB_ACOS_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients. The function alias GFLIB_Acos uses the default coefficients.

4.27.3 Return

The function returns arccos(fltIn) as a single precision floating point number.

4.27.4 Description

The GFLIB_Acos_FLT function provides a computational method for the calculation of the standard inverse trigonometric *arccosine* function arccos(x), using the square root based polynomial approximation. Function arccos(x) takes the ratio of the length of the adjacent side to the length of the hypotenuse and returns the angle.

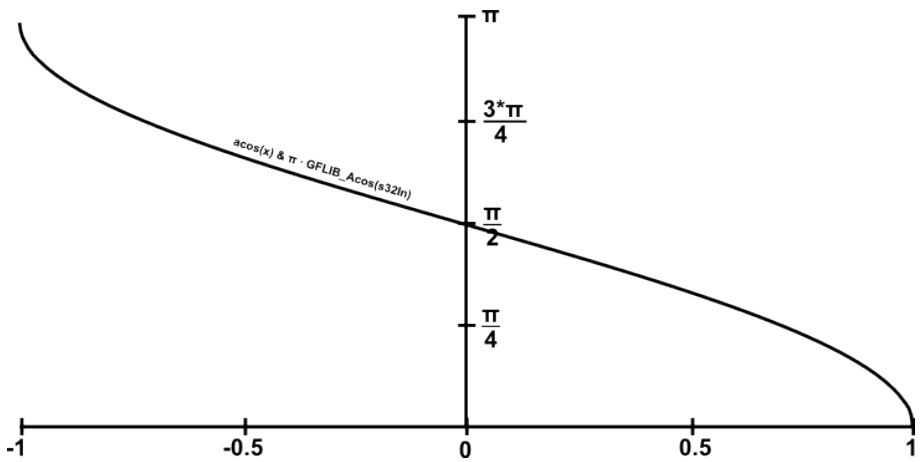

Figure 4-11. Course of the function GFLIB_Acos

The computation algorithm for arccos(x) uses the symmetry of the arcsin(x) function around the point (0, 0), which allows the calculation of the function values just in interval [0, 1), and to calculate the function values in the interval [-1, 0) use the simple formula:

$$y_{[-1,0)} = \frac{\pi}{2} + y_{[0,1)}$$

Equation **GFLIB_Acos_Eq1**

where:

- y[-1, 0) is the arccos(x) function value in the interval [-1, 0)
- y[0, 1) is the arccos(x) function value in the interval [0, 1)

For the interval [0, 1], the algorithm uses a polynomial approximation as follows:

$$\arcsin(\text{fltIn}) = a_4 - \sqrt{1 - \text{fltIn}} \bullet \left(a_3 + a_2 \bullet \text{fltIn} + a_1 \bullet \text{fltIn}^2 + a_0 \bullet \text{fltIn}^3\right)$$

Equation **GFLIB_Acos_Eq2**

To calculate the values of the arcsin(x) function in interval [-1, 0), the symmetry of the arcsin(x) function is used and the value is determined by the following formula:

$$\arcsin(\text{fltIn})_{[-1,0)} = -\arcsin(\text{fltIn})_{(0,1]}$$

Equation **GFLIB_Acos_Eq3**

The arccos(x) values are then calculated by the simple formula:

$$\arccos(x) = \frac{\pi}{2} - \arcsin(x)$$

Equation **GFLIB_Acos_Eq4**

Polynomial approximation coefficients used for GFLIB_Acos_FLT calculation are noted in Table 4-30.

Table 4-30. Approximation polynomial coefficients

a_0	a_1	a_2	a_3	a_4
-0.018725300	0.074261000	-0.212114380	1.570732421	1.570796326

The functions *arcsine* and *arccosine* are similar, therefore the GFLIB_Acos_FLT function uses the same polynomial coefficients as the GFLIB_Asin_FLT function.

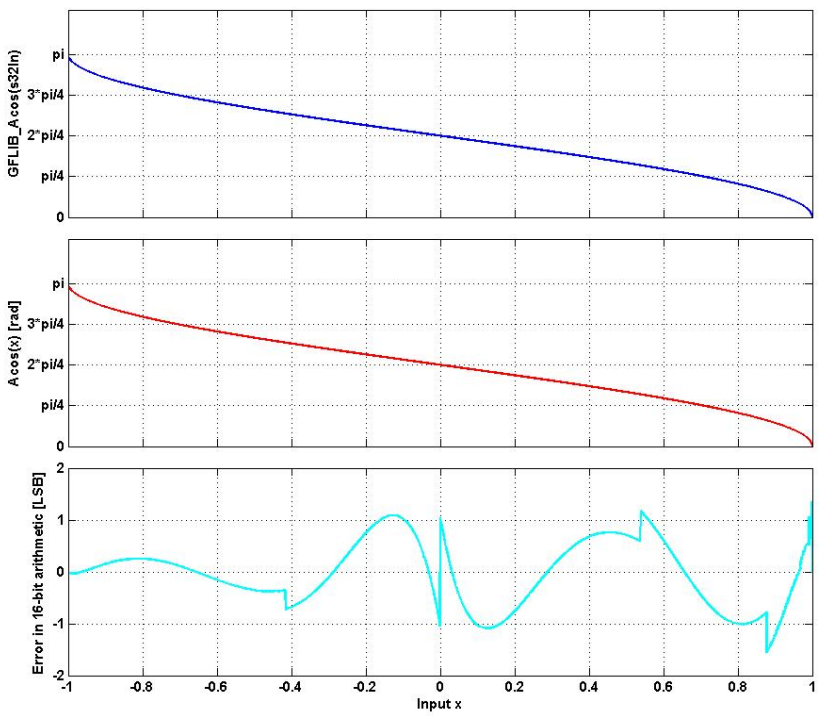

Figure 4-12. acos(x) vs. GFLIB_Acos(s32In)

Note

The input value is assumed to be in the interval [-1, 1]. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Acos_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Acos(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ACOS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Acos(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ACOS_DEFAULT_FLT approximation coefficients are used.

4.27.5 Re-entrancy

The function is re-entrant.

4.27.6 Code Example

```
#include "gflib.h"

tFloat fltInput;
tFloat fltAngle;

void main(void)
{
  // input fltInput = 0
  fltInput  = 0;

  // output should be 1.570796
  fltAngle = GFLIB_Acos_FLT (fltInput, GFLIB_ACOS_DEFAULT_FLT);

  // output should be 1.570796
  fltAngle = GFLIB_Acos (fltInput, GFLIB_ACOS_DEFAULT_FLT, Define FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################
```

```
    // output should be 1.570796
    fltAngle = GFLIB_Acos (fltInput);
}
```

4.28 Function GFLIB_Asin_F32

This function implements polynomial approximation of arcsine function.

4.28.1 Declaration

```
tFrac32 GFLIB_Asin_F32(tFrac32 f32In, const GFLIB_ASIN_T_F32 *const pParam);
```

4.28.2 Arguments

Table 4-31. GFLIB_Asin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains a value between [-1,1).
const GFLIB_ASIN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.28.3 Return

The function returns arcsin(f32In)/ π as a fixed point 32-bit number, normalized between [-1,1).

4.28.4 Description

The GFLIB_Asin_F32 function provides a computational method for calculation of a standard inverse trigonometric *arcsine* function arcsin(x), using the piece-wise polynomial approximation. Function arcsin(x) takes the ratio of the length of the opposite side to the length of the hypotenuse and returns the angle.

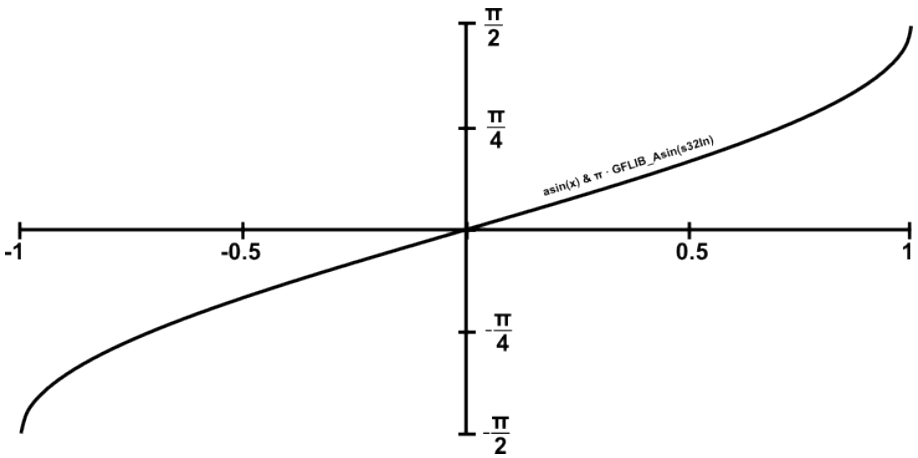

Figure 4-13. Course of the function GFLIB_Asin

The computational algorithm uses the symmetry of the arcsin(x) function around the point (0, π/2), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- y[-1, 0) is the arcsin(x) function value in the interval [-1, 0)
- y[1, 0) is the arcsin(x) function value in the interval [1, 0)

Additionally, because the arcsin(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5] :

$$\arcsin\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arcsin\left(\sqrt{x}\right)$$

Equation **GFLIB_Asin_Eq2**

In this way, the computation of the arcsin(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

For the interval (0, 0.5], the algorithm uses polynomial approximation as follows:

$$f32Dump = a_0 \bullet f32In^4 + a_1 \bullet f32In^3 + a_2 \bullet f32In^2 + a_3 \bullet f32In + a_4$$

Equation **GFLIB_Asin_Eq3**

$$\arcsin\left(f32In\right)=\begin{cases}-f32Dump & \text{if} \quad -1\le f32In\le 0\\ f32Dump & \text{if} \quad 0\le f32In<1\end{cases}$$

Equation **GFLIB_Asin_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-32.

Table 4-32. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	91918582	66340080	9729967	682829947	12751
<1/2, 1)	-52453538	-36708911	-15136243	-964576326	1073652175

Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 5th order was used for the fitting of each respective sub-interval. The Matlab was used as follows:

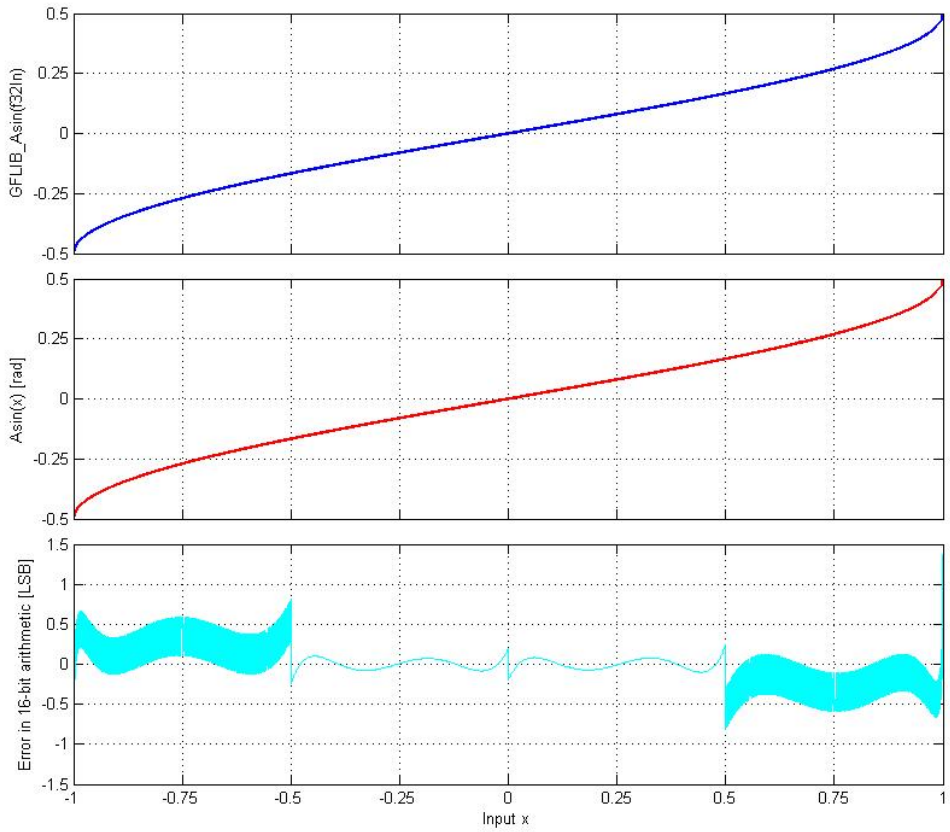

Figure 4-14. asin(x) vs. GFLIB_Asin(f32In)

Note

The output angle is normalized into the range [-0.5,0.5). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Asin_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Asin(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default

GFLIB_ASIN_DEFAULT_F32 approximation coefficients are used.

4.28.5 Re-entrancy

The function is re-entrant.

4.28.6 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
{
    // input f32Input = 1
    f32Input  = FRAC32 (1);

    // output should be 0x400031EF = 0.5 => pi/2
    f32Angle = GFLIB_Asin_F32 (f32Input, GFLIB_ASIN_DEFAULT_F32);

    // output should be 0x400031EF = 0.5 => pi/2
    f32Angle = GFLIB_Asin (f32Input, GFLIB_ASIN_DEFAULT_F32, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x400031EF = 0.5 => pi/2
    f32Angle = GFLIB_Asin (f32Input);
}
```

4.29 Function GFLIB_Asin_F16

This function implements polynomial approximation of arcsine function.

4.29.1 Declaration

```
tFrac16 GFLIB_Asin_F16(tFrac16 f16In, const GFLIB_ASIN_T_F16 *const pParam);
```

4.29.2 Arguments

Table 4-33. GFLIB_Asin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains a value between [-1,1).
const GFLIB_ASIN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.29.3 Return

The function returns arcsin(f16In)/ π as a fixed point 16-bit number, normalized between [-1,1).

4.29.4 Description

The GFLIB_Asin_F16 function provides a computational method for calculation of a standard inverse trigonometric *arcsine* function arcsin(x), using the piece-wise polynomial approximation. Function arcsin(x) takes the ratio of the length of the opposite side to the length of the hypotenuse and returns the angle.

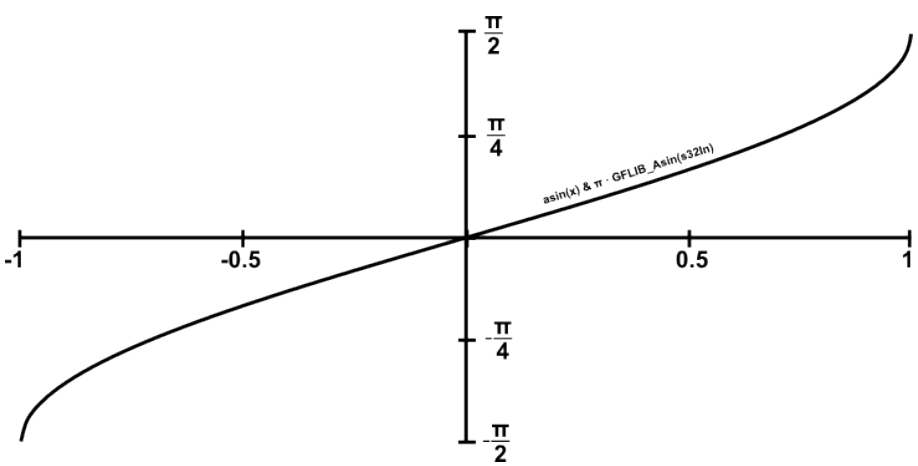

Figure 4-15. Course of the function GFLIB_Asin

The computational algorithm uses the symmetry of the arcsin(x) function around the point (0, π/2), which allows to for computing the function values just in the interval [0, 1) and to compute the function values in the interval [-1, 0) by the simple formula:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- y[-1, 0) is the arcsin(x) function value in the interval [-1, 0)
- y[1, 0) is the arcsin(x) function value in the interval [1, 0)

Additionally, because the arcsin(x) function is difficult for polynomial approximation for x approaching 1 (or -1 by symmetry), due to its derivatives approaching infinity, a special transformation is used to transform the range of x from [0.5, 1) to (0, 0.5]:

$$\arcsin\left(\sqrt{1-x}\right) = \frac{\pi}{2} - \arcsin\left(\sqrt{x}\right)$$

Equation **GFLIB_Asin_Eq2**

In this way, the computation of the arcsin(x) function in the range [0.5, 1) can be replaced by the computation in the range (0, 0.5], in which approximation is easier.

For the interval (0, 0.5], the algorithm uses polynomial approximation as follows:

$$f16Dump = a_0 \bullet f16In^4 + a_1 \bullet f16In^3 + a_2 \bullet f16In^2 + a_3 \bullet f16In + a_4$$

Equation **GFLIB_Asin_Eq3**

$$\arcsin\left(f16In\right) = \begin{cases} -f16Dump & \text{if} \quad -1 \le f16In \le 0 \\ f16Dump & \text{if} \quad 0 \le f16In < 1 \end{cases}$$

Equation **GFLIB_Asin_Eq4**

The division of the [0,1) interval into two sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-34.

Table 4-34. Integer polynomial coefficients for each interval

Interval	a_0	a_1	a_2	a_3	a_4
<0 , 1/2)	1403	1012	148	10419	0
<1/2, 1)	-800	-560	-231	-14718	16383

Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 5th order was used for the fitting of each respective sub-interval. The Matlab was used as follows:

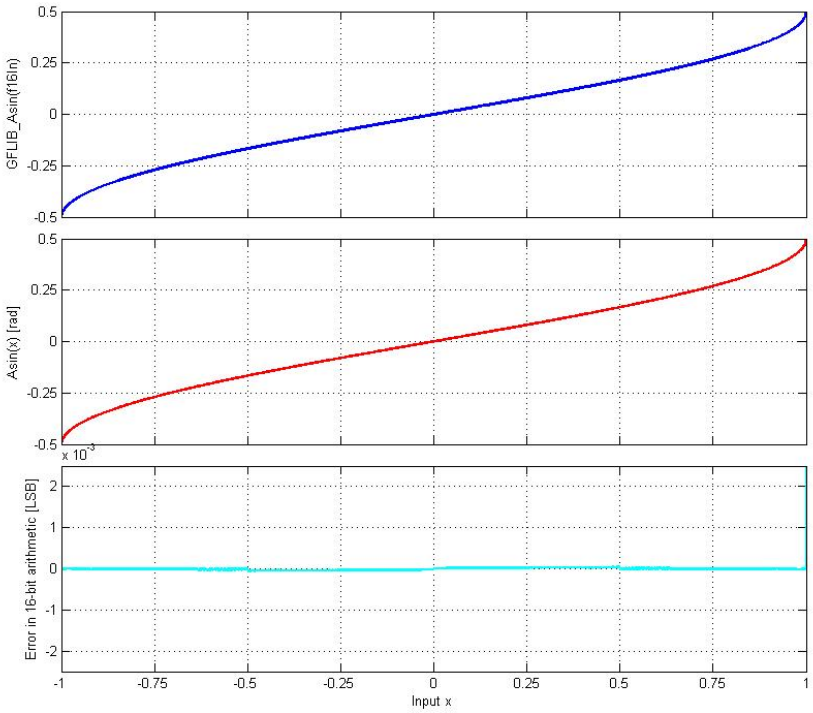

Figure 4-16. asin(x) vs. GFLIB_Asin(f16In)

Note

The output angle is normalized into the range [-0.5,0.5). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Asin_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Asin(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be

replaced with GFLIB_ASIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ASIN_DEFAULT_F16 approximation coefficients are used.

4.29.5 Re-entrancy

The function is re-entrant.

4.29.6 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input f16Input = 1
    f16Input  = FRAC16 (1);

    // output should be 0x4000 = 0.5 => pi/2
    f16Angle = GFLIB_Asin_F16 (f16Input, GFLIB_ASIN_DEFAULT_F16);

    // output should be 0x4000 = 0.5 => pi/2
    f16Angle = GFLIB_Asin (f16Input, GFLIB_ASIN_DEFAULT_F16, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x4000 = 0.5 => pi/2
    f16Angle = GFLIB_Asin (f16Input);
}
```

4.30 Function GFLIB_Asin_FLT

This function implements polynomial approximation of arcsine function.

4.30.1 Declaration

```
tFloat GFLIB_Asin_FLT(tFloat fltIn, const GFLIB_ASIN_T_FLT *const pParam);
```

4.30.2 Arguments

Table 4-35. GFLIB_Asin_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a 32-bit number that contains a single precision floating point value.
const GFLIB_ASIN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

4.30.3 Return

The function returns arcsin(fltIn) as a single precision floating point number.

4.30.4 Description

The GFLIB_Asin_FLT function provides a computational method for the calculation of the standard inverse trigonometric *arcsine* function arcsin(x), using the combination of a square root based polynomial approximation and a Minimax approximation. Function arcsin(x) takes the ratio of the length of the opposite side to the length of the hypotenuse and returns the angle.

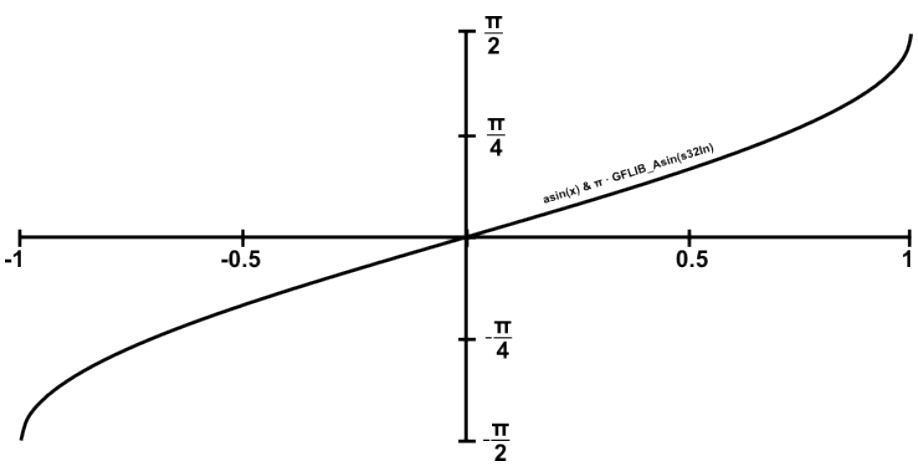

Figure 4-17. Course of the function GFLIB_Asin

The computation algorithm for arcsin(x) uses the symmetry of the arcsin(x) function around the point (0, 0), which allows the calculation of the function values just in interval [0, 1), and to calculation of the function values in the interval [-1, 0), use the simple formula:

$$y_{[-1,0)} = -y_{[0,1)}$$

Equation **GFLIB_Asin_Eq1**

where:

- y[-1, 0) is the arcsin(x) function value in the interval [-1, 0)
- y[0, 1] is the arcsin(x) function value in the interval [0, 1]

For the interval [0, 0.4), the algorithm uses a Minimax approximation as follows:

$$\arcsin(\text{fltIn}) = a_0 \bullet \text{fltIn} + a_1 \bullet \text{fltIn}^3 + a_2 \bullet \text{fltIn}^5$$

Equation **GFLIB_Asin_Eq2**

For the interval [0.4, 1), the algorithm uses a polynomial approximation as follows.

$$\arcsin(\text{fltIn}) = a_4 - \sqrt{1 - \text{fltIn}} \bullet \left(a_3 + a_2 \bullet \text{fltIn} + a_1 \bullet \text{fltIn}^2 + a_0 \bullet \text{fltIn}^3\right)$$

Equation **GFLIB_Asin_Eq3**

To calculate the values of the arcsin(x) function in interval [-1, 0), the symmetry of the arcsin(x) function is used and the value is determined by the following formula:

$$\arcsin(\text{fltIn})_{[-1,0)} = -\arcsin(\text{fltIn})_{(0,1]}$$

Equation **GFLIB_Asin_Eq4**

Minimax approximation coefficients used for GFLIB_Asin_FLT calculation in interval [0, 0.4), are noted in Table 4-36.

Table 4-36. Approximation polynomial coefficients

a_0	a_1	a_2
1.000001892183630	0.166350537976503	0.082733681152843

In Table 4-37 there are the approximation coefficients used for GFLIB_Asin_FLT calculation in interval [0.4, 1).

Table 4-37. Approximation polynomial coefficients

a_0	a_1	a_2	a_3	a_4
-0.018725300	0.074261000	-0.212114380	1.570732421	1.570796326

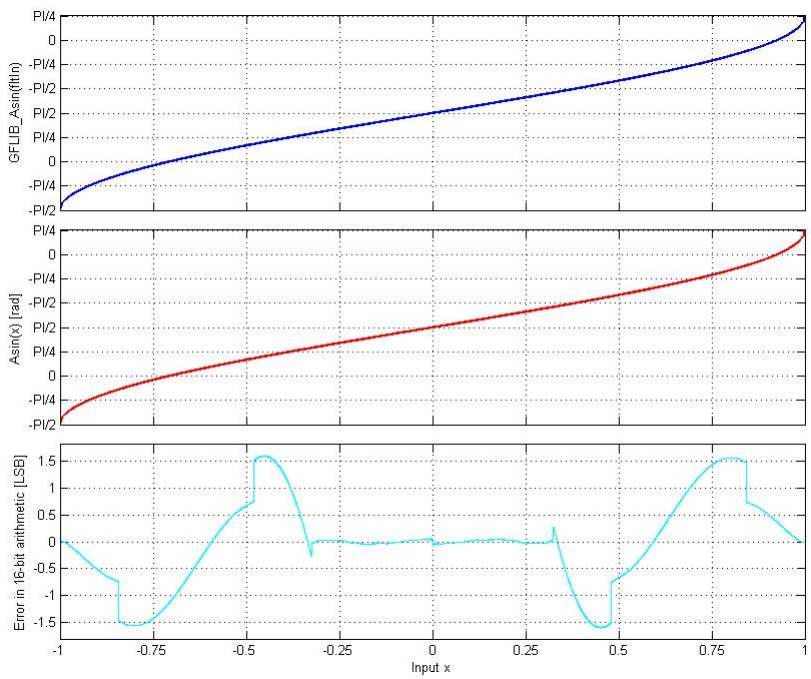

Figure 4-18. asin(x) vs. GFLIB_Asin(f32In)

Note

The input value is assumed to be in the interval [-1, 1] . The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Asin_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Asin(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ASIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Asin(fltIn, &pParam), where &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ASIN_DEFAULT_FLT approximation coefficients are used.

4.30.5 Re-entrancy

The function is re-entrant.

4.30.6 Code Example

```
#include "gflib.h"

tFloat fltInput;
tFloat fltAngle;

void main(void)
{
    // input fltInput = 1
    fltInput  = 1;

    // output should be 1.570796
    fltAngle = GFLIB_Asin_FLT (fltInput, GFLIB_ASIN_DEFAULT_FLT);

    // output should be 1.570796
    fltAngle = GFLIB_Asin (fltInput, GFLIB_ASIN_DEFAULT_FLT, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1.570796
    fltAngle = GFLIB_Asin (fltInput);
}
```

4.31 Function GFLIB_Atan_F32

This function implements Minimax polynomial approximation of arctangent function.

4.31.1 Declaration

```
tFrac32 GFLIB_Atan_F32(tFrac32 f32In, const GFLIB_ATAN_T_F32 *const pParam);
```

4.31.2 Arguments

Table 4-38. GFLIB_Atan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number between [-1,1).
const GFLIB_ATAN_T_F32 *const	pParam	**input**	Pointer to an array of minimax approximation coefficients.

4.31.3 Return

The function returns the atan of the input argument as a fixed point 32-bit number that contains the angle in radians between [- π/4, π/4), normalized between [-0.25,0.25).

4.31.4 Description

The GFLIB_Atan_F32 function provides a computational method for calculation of a standard trigonometric *arctangent* function arctan(x), using the piece-wise minimax polynomial approximation. Function arctan(x) takes a ratio and returns the angle of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The graph of arctan(x) is shown in Figure 4-19.

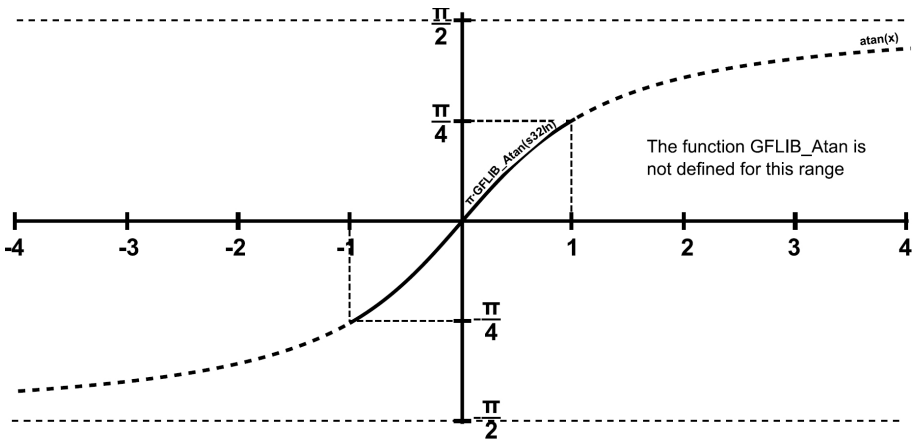

Figure 4-19. Course of the function GFLIB_Atan

The GFLIB_Atan_F32 function is implemented with consideration to fixed point fractional arithmetic. As can be further seen from Figure 4-19, the arctangent values are identical for the input ranges [-1, 0) and [0, 1).

Moreover, it can be observed from Figure 4-19 that the course of the arctan(x) function output for a ratio in interval 1. is identical, but with the opposite sign, to the output for a ratio in interval 2. Therefore, the approximation of the *arctangent* function over the entire defined range of input ratios can be simplified to the approximation for a ratio in range [0, 1), and then, depending on the input ratio, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval [0, 1) is further divided into eight equally spaced sub intervals, and minimax polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients. Moreover, it allows using a polynomial of only the 3rd order to achieve an accuracy of less than 0.5LSB (on the upper 16 bits of 32-bit results) across the entire range of input ratios.

The GFLIB_Atan_F32 function uses fixed point fractional arithmetic, so to cast the fractional value of the output angle [-0.25, 0.25) into the correct range [- π/4, π/4), the fixed point output angle can be multiplied by π for an angle in radians. Then, the fixed point fractional implementation of the minimax approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f32Dump = a_1 f32In^2 + a_2 f32In + a_3$$

Equation **GFLIB_Atan_Eq1**

$$\arctan\left(f32In\right) = \begin{cases} f32Dump & \text{if} \quad 0 \le f32In < 1 \\ -f32Dump & \text{if} \quad -1 \le f32In < 0 \end{cases}$$

Equation **GFLIB_Atan_Eq2**

The division of the [0, 1) interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-39. Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 3rd order was used for the fitting of each respective sub-interval.

Table 4-39. Integer minimax polynomial coefficients for each interval

Interval	a_1	a_2	a_3
<0, 1/8)	-164794	42515925	42667172
<1/8, 2/8)	-465182	41238272	126697014
<2/8, 3/8)	-690034	38899574	207041074
<3/8, 4/8)	-820713	35848645	281909001
<4/8, 5/8)	-865105	32453241	350251355
<5/8, 6/8)	-845462	29016149	411702516
<6/8, 7/8)	-786689	25743137	466407809
<7/8, 1)	-708969	22748418	514828039

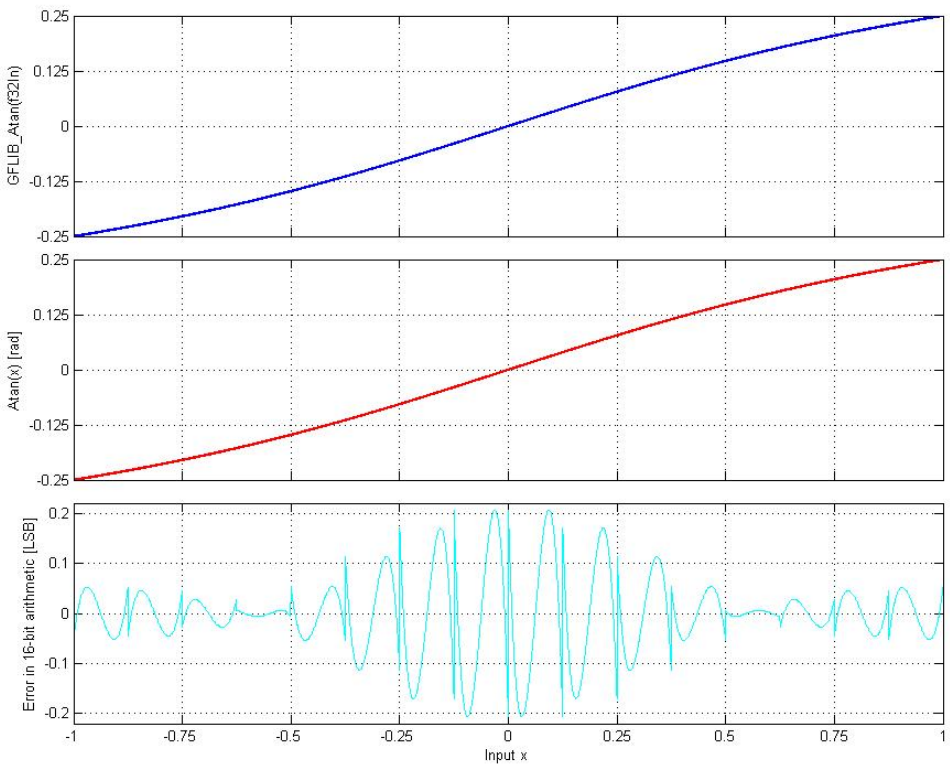

Figure 4-20. atan(x) vs. GFLIB_Atan(f32In)

Figure 4-20 depicts a floating point *arctangent* function generated from Matlab and the approximated value of the *arctangent* function obtained from GFLIB_Atan_F32, plus their difference. The course of calculation accuracy as a function of the input value can be observed from this figure. The achieved accuracy with consideration to the 3rd order piece-wise minimax polynomial approximation and described fixed point scaling is less than 0.5LSB on the upper 16 bits of the 32-bit result.

Note

The output angle is normalized into the range [-0.25, 0.25). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Atan_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Atan(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Atan(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ATAN_DEFAULT_F32 approximation coefficients are used.

4.31.5 Re-entrancy

The function is re-entrant.

4.31.6 Code Example

```
#include "gflib.h"

tFrac32 f32Input;
tFrac32 f32Angle;

void main(void)
```

```
{
    // input ratio = 0x7FFFFFFF => 1
    f32Input  = FRAC32 (1);

    // output angle should be 0x1FFFBD7F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan_F32 (f32Input, GFLIB_ATAN_DEFAULT_F32);

    // output angle should be 0x1FFFBD7F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan (f32Input, GFLIB_ATAN_DEFAULT_F32, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output angle should be 0x1FFFBD7F = 0.249999 => pi/4
    f32Angle = GFLIB_Atan (f32Input);
}
```

4.32 Function GFLIB_Atan_F16

This function implements Minimax polynomial approximation of arctangent function.

4.32.1 Declaration

```
tFrac16 GFLIB_Atan_F16(tFrac16 f16In, const GFLIB_ATAN_T_F16 *const pParam);
```

4.32.2 Arguments

Table 4-40. GFLIB_Atan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number between [-1, 1).
const GFLIB_ATAN_T_F16 *const	pParam	**input**	Pointer to an array of minimax approximation coefficients.

4.32.3 Return

The function returns the atan of the input argument as a fixed point 16-bit number that contains the angle in radians between [- π/4, π/4), normalized between [-0.25, 0.25).

4.32.4 Description

The GFLIB_Atan_F16 function provides a computational method for calculation of a standard trigonometric *arctangent* function arctan(x), using the piece-wise minimax polynomial approximation. Function arctan(x) takes a ratio and returns the angle of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The graph of arctan(x) is shown in Figure 4-21.

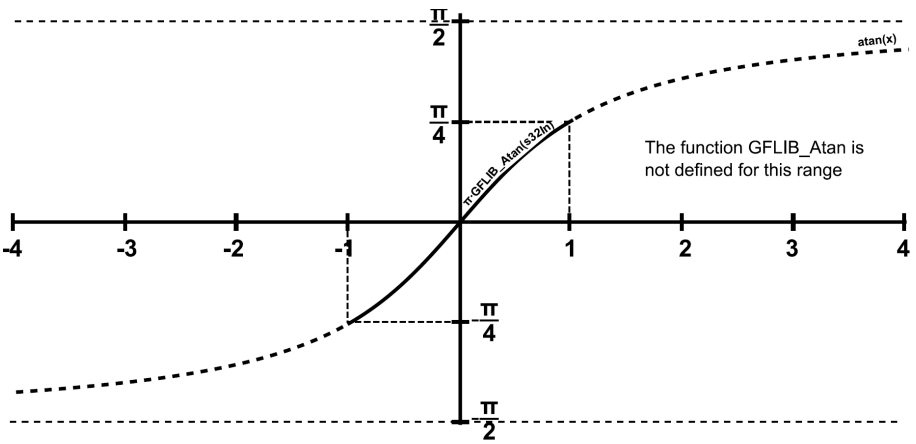

Figure 4-21. Course of the function GFLIB_Atan

The GFLIB_Atan_F16 function is implemented with consideration to fixed point fractional arithmetic. As can be further seen from Figure 4-21, the arctangent values are identical for the input ranges [-1, 0) and [0, 1).

Moreover, it can be observed from Figure 4-21 that the course of the arctan(x) function output for a ratio in interval 1. is identical, but with the opposite sign, to the output for a ratio in interval 2. Therefore, the approximation of the *arctangent* function over the entire defined range of input ratios can be simplified to the approximation for a ratio in range [0, 1), and then, depending on the input ratio, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval [0, 1) is further divided into eight equally spaced sub intervals, and minimax polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients. Moreover, it allows using a minimax polynomial of only the 3rd order to achieve an accuracy of less than 0.5LSB across the entire range of input ratios.

The GFLIB_Atan_F16 function uses fixed point fractional arithmetic, so to cast the fractional value of the output angle [-0.25, 0.25) into the correct range [- π/4, π/4), the fixed point output angle can be multiplied by π for an angle in radians. Then, the fixed point fractional implementation of the minimax approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f16Dump = a_1 f16In^2 + a_2 f16In + a_3$$

Equation **GFLIB_Atan_Eq1**

$$\arctan\left(f16In\right) = \begin{cases} f16Dump & \text{if} \quad 0 \le f16In < 1 \\ -f16Dump & \text{if} \quad -1 \le f16In < 0 \end{cases}$$

Equation **GFLIB_Atan_Eq2**

The division of the [0, 1) interval into eight sub-intervals, with minimax polynomial coefficients calculated for each sub-interval, is noted in Table 4-41. Minimax polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 3rd order was used for the fitting of each respective sub-interval.

Table 4-41. Integer polynomial coefficients for each interval

Interval	a_1	a_2	a_3
<0, 1/8)	-3	649	651
<1/8, 2/8)	-7	630	1933
<2/8, 3/8)	-11	594	3159
<3/8, 4/8)	-13	547	4302
<4/8, 5/8)	-13	495	5344
<5/8, 6/8)	-13	443	6282
<6/8, 7/8)	-12	393	7117
<7/8, 1)	-11	347	7856

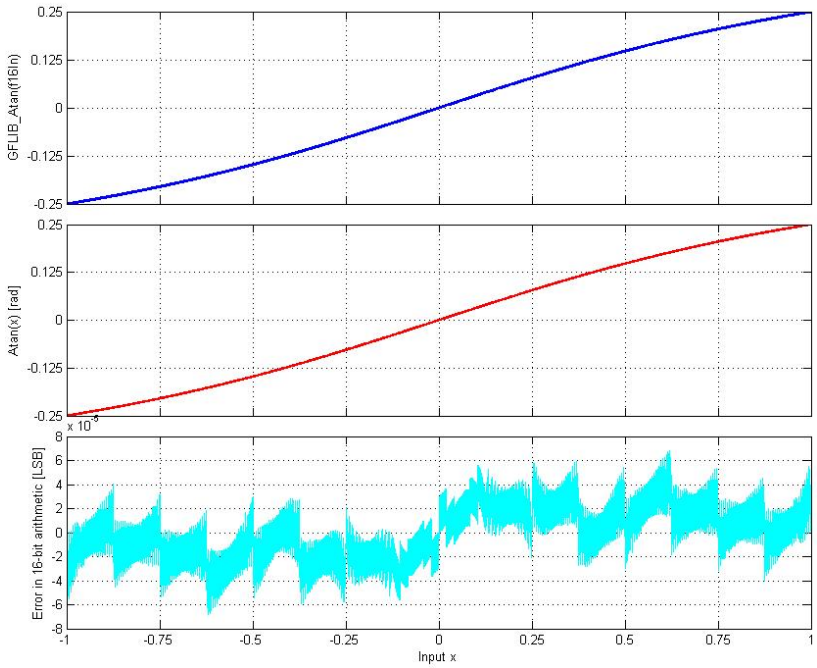

Figure 4-22. atan(x) vs. GFLIB_Atan(f16In)

Note

The output angle is normalized into the range [-0.25,0.25). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Atan_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Atan(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Atan(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ATAN_DEFAULT_F16 approximation coefficients are used.

4.32.5 Re-entrancy

The function is re-entrant.

4.32.6 Code Example

```
#include "gflib.h"

tFrac16 f16Input;
tFrac16 f16Angle;

void main(void)
{
    // input ratio = 0x7FFF => 1
    f16Input  = FRAC16 (1);

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan_F16 (f16Input, GFLIB_ATAN_DEFAULT_F16);

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan (f16Input, GFLIB_ATAN_DEFAULT_F16, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output angle should be 0x1FFF = 0.249999 => pi/4
    f16Angle = GFLIB_Atan (f16Input);
}
```

4.33 Function GFLIB_Atan_FLT

This function implements rational polynomial approximation of arctangent function.

4.33.1 Declaration

```
tFloat GFLIB_Atan_FLT(tFloat fltIn, const GFLIB_ATAN_T_FLT *const pParam);
```

4.33.2 Arguments

Table 4-42. GFLIB_Atan_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number between (-2^{128}, 2^{128}).

Table continues on the next page...

Table 4-42. GFLIB_Atan_FLT arguments (continued)

Type	Name	Direction	Description
const GFLIB_ATAN_T_FLT *const	pParam	**input**	Pointer to an array of rational polynomial coefficients.

4.33.3 Return

The function returns the atan of the input argument as a single precision floating point number that contains the angle in radians between (- π/2, π/2).

4.33.4 Description

The GFLIB_Atan_FLT function provides a computational method for the calculation of a standard trigonometric *arctangent* function arctan(x), using the rational polynomial approximation. Function arctan(x) takes a ratio and returns the angle of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The graph of arctan(x) is shown in Figure 4-23.

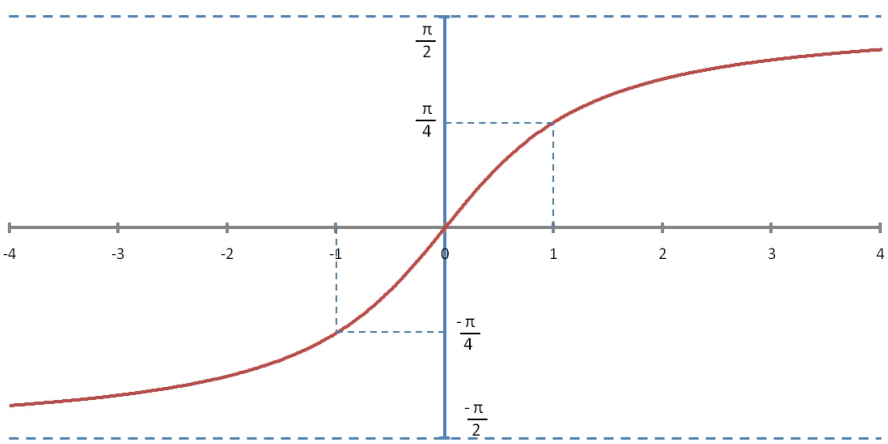

Figure 4-23. Course of the function GFLIB_Atan

As can be observed from Figure 4-23, the value of the arctan(x) function for interval (- ∞, 0) is identical, but with the opposite sign, to the output of the function for [0, ∞). Therefore, the approximation of the *arctangent* function over the entire defined range of input values can be simplified to the approximation for a ratio in the range [0, ∞), and then, depending on the input sign, the result will be negated.

In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval [0, ∞) is further divided into two intervals: [0, 1] and (1, ∞). For the first interval [0, 1] the rational polynomial approximation directly uses the input value, while for the second interval (1, ∞) the reciprocal input parameter is used as the input value for rational polynomial approximation, and the output value is subtracted from π/2.

The GFLIB_Atan_FLT function uses the rational polynomial approximation as defined in the following equation:

$$fltOut = \frac{fltIn \cdot (a_1 + fltIn^2 \cdot (a_2 + a_3 \cdot fltIn^2))}{a_4 + fltIn^2 \cdot (a_5 + fltIn^2 \cdot (a_6 + fltIn^2))}$$

Equation **GFLIB_Atan_Eq1**

The rational polynomial approximation coefficients used for GFLIB_Atan_FLT calculation, are noted in Table 4-43.

Table 4-43. Rational polynomial approximation coefficients

a_1	a_2	a_3	a_4	a_5	a_6
48.70107004	49.53262637	9.40604244	65.76631639	21.58787406	48.70107004

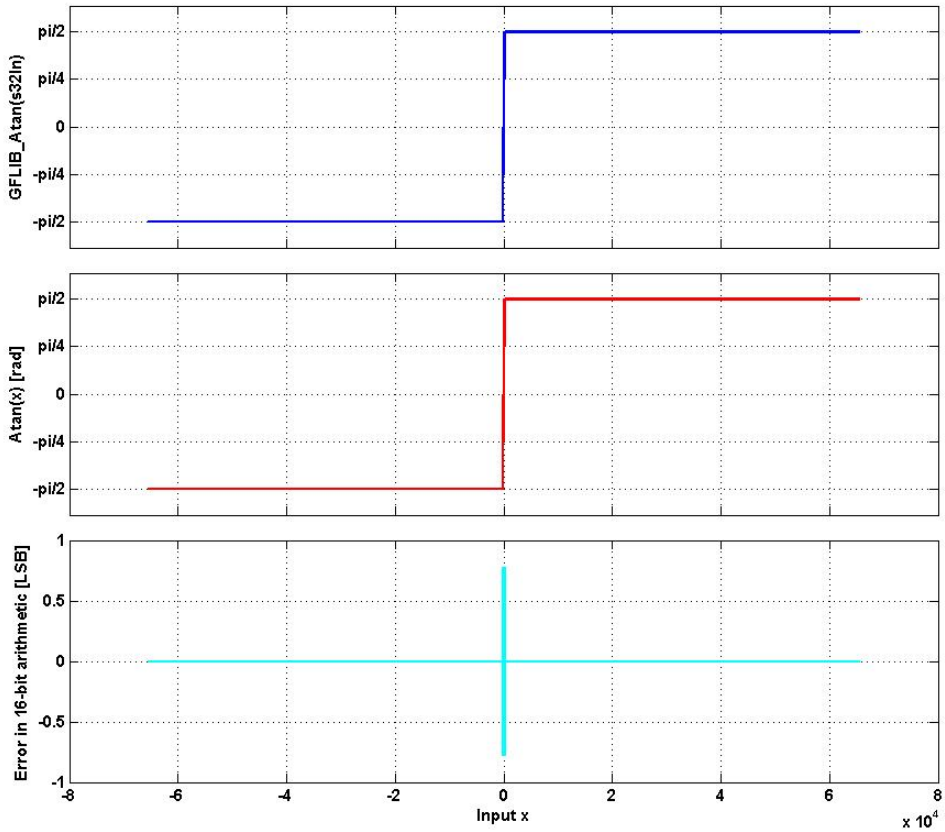

Figure 4-24. atan(x) vs. GFLIB_Atan_FLT(fltIn)

Note

The input value is assumed to be in the interval (-2 ^{128}, 2 ^{128}). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Atan_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Atan(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_ATAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Atan(fltIn, &pParam), where the &pParam is

pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_ATAN_DEFAULT_FLT approximation coefficients are used.

4.33.5 Re-entrancy

The function is re-entrant.

4.33.6 Code Example

```
#include "gflib.h"

tFloat fltInput;
tFloat fltAngle;

void main(void)
{
    // input ratio = 1
    fltInput  = 1;

    // output angle should be 0.7853981634 => pi/4
    fltAngle = GFLIB_Atan_FLT (fltInput, GFLIB_ATAN_DEFAULT_FLT);

    // output angle should be 0.7853981634 => pi/4
    fltAngle = GFLIB_Atan (fltInput, GFLIB_ATAN_DEFAULT_FLT, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output angle should be 0.7853981634 => pi/4
    fltAngle = GFLIB_Atan (fltInput);

}
```

4.34 Function GFLIB_AtanYX_F32

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

4.34.1 Declaration

```
tFrac32 GFLIB_AtanYX_F32(tFrac32 f32InY, tFrac32 f32InX);
```

4.34.2 Arguments

Table 4-44. GFLIB_AtanYX_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InY	**input**	The ordinate of the input vector (y coordinate).
tFrac32	f32InX	**input**	The abscissa of the input vector (x coordinate).

4.34.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

4.34.4 Description

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters. The first parameter, f32InY, is the ordinate (the y coordinate) and the second one, f32InX, is the abscissa (the x coordinate).

Both the input parameters are assumed to be in the fractional range of [-1, 1). The computed angle is limited by the fractional range of [-1, 1), which corresponds to the real range of [- π, π). The counter-clockwise direction is assumed to be positive and thus a positive angle will be computed if the provided ordinate (f32InY) is positive. Similarly, a negative angle will be computed for the negative ordinate.

The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle by dividing the angle into two parts: the integral multiple of 45 deg (half-quarter) and the remaining offset within the 45 deg range. Simple geometric properties of the Cartesian coordinate system are used to calculate the coordinates of the vector with the calculated angle offset.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the integral multiple of half-quarters and the angle offset within a single half-quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

In comparison to the GFLIB_Atan_F32 function, the GFLIB_AtanYX_F32 function correctly places the calculated angle within the whole fractional range of [-1, 1), which corresponds to the real angle range of [- π, π).

Note

The function calls the GFLIB_Atan_F32 function. The computed value is within the range of [-1, 1).

4.34.5 Re-entrancy

The function is re-entrant.

4.34.6 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    f32InY = FRAC32 (0.5);
    f32InX = FRAC32 (0.5);

    // output should be close to 0x20001000
    f32Ang = GFLIB_AtanYX_F32 (f32InY, f32InX);

    // output should be close to 0x20001000
    f32Ang = GFLIB_AtanYX (f32InY, f32InX, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be close to 0x20001000
    f32Ang = GFLIB_AtanYX (f32InY, f32InX);
}
```

4.35 Function GFLIB_AtanYX_F16

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

4.35.1 Declaration

```
tFrac16 GFLIB_AtanYX_F16(tFrac16 f16InY, tFrac16 f16InX);
```

4.35.2 Arguments

Table 4-45. GFLIB_AtanYX_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InY	**input**	The ordinate of the input vector (y coordinate).
tFrac16	f16InX	**input**	The abscissa of the input vector (x coordinate).

4.35.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

4.35.4 Description

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters. The first parameter, f16InY, is the ordinate (the y coordinate) and the second one, f16InX, is the abscissa (the x coordinate).

Both the input parameters are assumed to be in the fractional range of [-1, 1). The computed angle is limited by the fractional range of [-1, 1), which corresponds to the real range of [- π, π). The counter-clockwise direction is assumed to be positive and thus a positive angle will be computed if the provided ordinate (f16InY) is positive. Similarly, a negative angle will be computed for the negative ordinate.

The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle by dividing the angle into two parts: the integral multiple of 45 deg (half-quarter) and the remaining offset within the 45 deg range. Simple geometric properties of the Cartesian coordinate system are used to calculate the coordinates of the vector with the calculated angle offset.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the integral multiple of half-quarters and the angle offset within a single half-quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

In comparison to the GFLIB_Atan_F16 function, the GFLIB_AtanYX_F16 function correctly places the calculated angle within the whole fractional range of [-1, 1), which corresponds to the real angle range of [- π, π).

Note

The function calls the GFLIB_Atan_F16 function. The computed value is within the range of [-1, 1).

4.35.5 Re-entrancy

The function is re-entrant.

4.35.6 Code Example

```
#include "gflib.h"

tFrac16 f16InY;
tFrac16 f16InX;
tFrac16 f16Ang;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    f16InY = FRAC16 (0.5);
    f16InX = FRAC16 (0.5);

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX_F16 (f16InY, f16InX);

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX (f16InY, f16InX, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be close to 0x2000
    f16Ang = GFLIB_AtanYX (f16InY, f16InX);
}
```

4.36 Function GFLIB_AtanYX_FLT

The function calculates the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

4.36.1 Declaration

```
tFloat GFLIB_AtanYX_FLT(tFloat fltInY, tFloat fltInX);
```

4.36.2 Arguments

Table 4-46. GFLIB_AtanYX_FLT arguments

Type	Name	Direction	Description
tFloat	fltInY	**input**	The ordinate of the input vector (y coordinate).
tFloat	fltInX	**input**	The abscissa of the input vector (x coordinate).

4.36.3 Return

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters.

4.36.4 Description

The function returns the angle between the positive x-axis of a plane and the direction of the vector given by the x and y coordinates provided as parameters. The first parameter, fltInY, is the ordinate (the y coordinate) and the second one, fltInX, is the abscissa (the x coordinate).

Both the input parameters are assumed to be in the single precision floating point range of (-2^{128}, 2^{128}). The computed angle is in the range of [$-\pi$, π]. The counter-clockwise direction is assumed to be positive, and thus a positive angle will be computed if the provided ordinate (fltInY) is positive. Similarly, a negative angle will be computed for the negative ordinate. The calculations are performed in a few steps.

In the first step, the angle is positioned within the correct half-quarter of the circumference of a circle, and if necessary the final angle addition is prepared or the sign of the input value is corrected.

In the second step, the vector ordinate is divided by the vector abscissa (y/x) to obtain the tangent value of the angle offset. The angle offset is computed by applying the ordinary arctangent function.

The sum of the angle addition and the angle offset within a single quarter form the angle to be computed. The function will return 0 if both input arguments are 0.

Note

The function calls the GFLIB_Atan_FLT function. The computed value is within the range of [- π, π].

4.36.5 Re-entrancy

The function is re-entrant.

4.36.6 Code Example

```
#include "gflib.h"

tFloat fltInY;
tFloat fltInX;
tFloat fltAng;

void main(void)
{
    // Angle 45 deg = PI/4 rad
    fltInY = (tFloat)(0.5);
    fltInX = (tFloat)(0.5);

    // Output angle should be 45 deg = PI/4 rad = 0.7853981634
    fltAng = GFLIB_AtanYX_FLT (fltInY, fltInX);

    // Output angle should be 45 deg = PI/4 rad = 0.7853981634
    fltAng = GFLIB_AtanYX (fltInY, fltInX, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Output angle should be 45 deg = PI/4 rad = 0.7853981634
    fltAng = GFLIB_AtanYX (fltInY, fltInX);
}
```

4.37 Function GFLIB_AtanYXShifted_F32

This function calculates the angle of two sine waves shifted in phase to each other.

4.37.1 Declaration

```
tFrac32 GFLIB_AtanYXShifted_F32(tFrac32 f32InY, tFrac32 f32InX, GFLIB_ATANYXSHIFTED_T_F32
*pParam);
```

4.37.2 Arguments

Table 4-47. GFLIB_AtanYXShifted_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InY	**input**	The value of the first signal, assumed to be sin(θ).
tFrac32	f32InX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
GFLIB_ATANYXSHIFTED_T_F32 *	pParam	**input, output**	The parameters for the function.

4.37.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

4.37.4 Description

The function calculates the angle of two sinusoidal signals, one shifted in phase to the other. The phase shift between sinusoidal signals does not have to be π/2 and can be any value.

It is assumed that the arguments of the function are as follows:

$$y = \sin(\theta)$$
$$x = \sin(\theta + \Delta\theta)$$

Equation **GFLIB_AtanYXShifted_Eq1**

where:

- x, y are respectively, the f32InX, and f32InY arguments
- θ is the angle to be computed by the function
- Δθ is the phase difference between the x, y signals

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ. The angle offset is an additional parameter, which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ.

The GFLIB_AtanYXShifted_F32 function does not directly use the angle offset θ_{Offset} and the phase difference θ. The function's parameters, contained in the function parameters structure GFLIB_ATANYXSHIFTED_T_F32, need to be computed by means of the provided Matlab function (see below).

If $\Delta\theta = \pi/2$ or $\Delta\theta = -\pi/2$, then the function is similar to the GFLIB_AtanYX_F32 function, however, the GFLIB_AtanYX_F32 function in this case is more effective with regard to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define Δθ and θ_{Offset}, the Δθ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function
- convert the computed values into integer format and insert them into the C code (see also the C code example)

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)}\left(y + x\right)$$

$$y = \frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)}\left(x - y\right)$$

$$\theta = A\text{tanYX}\left(b, a\right) - \left(\frac{\Delta\theta}{2} - \theta_{\text{offset}}\right)$$

Equation **GFLIB_AtanYXShifted_Eq2**

where:

- x, y are respectively, the f32InX, and f32InY
- θ is the angle to be computed by the function, see the previous equation
- Δθ is the phase difference between the x, y signals, see the previous equation
- S is a scaling coefficient, S is almost 1, (S < 1), see also the explanation below
- a, b intermediate variables
- θ_{Offset} is the additional phase shift, the computed angle will be $\theta + \theta_{Offset}$

The scale coefficient S is used to prevent overflow and to assure symmetry around 0 for the entire fractional range. S shall be less than 1.0, but as large as possible. The algorithm implemented in this function uses the value of $1 - 2^{-15}$.

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan\left(\theta+\Delta\theta\right)=\frac{\left(y+x\right)\cos\frac{\Delta\theta}{2}}{\left(x-y\right)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanYXShifted_Eq3**

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)}=C_y=K_y2^{N_y}$$

$$\frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)}=C_x=K_x2^{N_x}$$

$$\theta_{\text{adj}}=\frac{\Delta\theta}{2}-\theta_{\text{offset}}$$

Equation **GFLIB_AtanYXShifted_Eq4**

where:

- C_y, C_x are the algorithm coefficients for y and x signals
- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->f32Ky
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->f32Kx
- N_y is scaling coefficient of the y signal, represented the by parameters structure member pParam->s32Ny
- N_x is scaling coefficient of the x signal, represented by the parameters structure member pParam->s32Nx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->f32ThetaAdj

The multiplication and scaling coefficients, and the adjusting angle, shall be defined in a parameters structure provided as the function input parameter.

The function initialization parameters can be calculated as shown in the following Matlab code:

While applying the function, some general guidelines should be considered as stated below.

```
function [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
      // ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
      //
      // [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
      //
      // dthdeg = phase shift (delta theta) between sine waves in degrees
```

```
// thoffsetdeg = angle offset (theta offset) in degrees
// NY - scaling coefficient of y signal
// NX - scaling coefficient of x signal
// KY - multiplication coefficient of y signal
// KX - multiplication coefficient of x signal
// THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

    if (dthdeg < -180) || (dthdeg >= 180)
        error('atanyxshiftedpar: dthdeg out of range');
    end
    if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
        error('atanyxshiftedpar: thoffsetdeg out of range');
    end

    dth2 = ((dthdeg/2)/180*pi);
    thoffset = (thoffsetdeg/180*pi);
    CY = (1 - 2^-15)/(2*cos(dth2));
    CX = (1 - 2^-15)/(2*sin(dth2));
    if(abs(CY) >= 1) NY = ceil(log2(abs(CY)));
    else NY = 0;
    end
    if(abs(CX) >= 1) NX = ceil(log2(abs(CX)));
    else NX = 0;
    end
    KY = CY/2^NY;
    KX = CX/2^NX;
    THETAADJ = dthdeg/2 - thoffsetdeg;

    if THETAADJ >= 180
        THETAADJ = THETAADJ - 360;
    elseif THETAADJ < -180
        THETAADJ = THETAADJ + 360;
    end

    THETAADJ = THETAADJ/180;

return;
```

At some values of the phase shift, and particularly at phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error where possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift $\Delta\theta$ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should be as close to 1.0 as much as possible.

The function has been tested to be reliable at a phase shift in the range of [-165, -15] and [15, 165] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note

The function calls the GFLIB_AtanYX_F32 function. The function may become numerically unstable for a phase shift approaching -180, 0 or 180 degrees. The function accuracy is guaranteed for a phase shift in the range of [-165, -15] and [15, 165] degrees at perfect input signals.

4.37.5 Re-entrancy

The function is re-entrant.

4.37.6 Code Example

```
#include "gflib.h"

tFrac32 f32InY;
tFrac32 f32InX;
tFrac32 f32Ang;
GFLIB_ATANYXSHIFTED_T_F32 Param;

void main(void)
{
    //dtheta = 69.33deg, thetaoffset = 10deg
    //CY = (1 - 2^-15)/(2*cos((69.33/2)/180*pi))= 0.60789036201452440
    //CX = (1 - 2^-15)/(2*sin((69.33/2)/180*pi))= 0.87905201358520957
    //NY = 0 (abs(CY) < 1)
    //NX = 0 (abs(CX) < 1)
    //KY = 0.60789/2^0 = 0.60789036201452440
    //KX = 0.87905/2^0 = 0.87905201358520957
    //THETAADJ = 10/180 = 0.05555555555

    Param.f32Ky = FRAC32 (0.60789036201452440);
    Param.f32Kx = FRAC32 (0.87905201358520957);
    Param.s32Ny = 0;
    Param.s32Nx = 0;
    Param.f32ThetaAdj = FRAC32 (0.05555555555);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    f32InY = FRAC32 (0.2588190);
    f32InX = FRAC32 (0.9951074);

    // f32Ang contains 0x11c6cdfc, the theoretical value is 0x1C34824B
    f32Ang = GFLIB_AtanYXShifted_F32 (f32InY, f32InX, &Param);

    // f32Ang contains 0x11c6cdfc, the theoretical value is 0x1C34824B
```

```
    f32Ang = GFLIB_AtanYXShifted (f32InY, f32InX, &Param, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // f32Ang contains 0x11c6cdfc, the theoretical value is 0x1C34824B
    f32Ang = GFLIB_AtanYXShifted (f32InY, f32InX, &Param);
}
```

4.38 Function GFLIB_AtanYXShifted_F16

This function calculates the angle of two sine waves shifted in phase to each other.

4.38.1 Declaration

```
tFrac16 GFLIB_AtanYXShifted_F16(tFrac16 f16InY, tFrac16 f16InX, GFLIB_ATANYXSHIFTED_T_F16
*pParam);
```

4.38.2 Arguments

Table 4-48. GFLIB_AtanYXShifted_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InY	**input**	The value of the first signal, assumed to be sin(θ).
tFrac16	f16InX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
GFLIB_ATANYXSHIFTED_T_F16 *	pParam	**input, output**	The parameters for the function.

4.38.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

4.38.4 Description

The function calculates the angle of two sinusoidal signals, one shifted in phase to the other. The phase shift between sinusoidal signals does not have to be π/2 and can be any value.

It is assumed that the arguments of the function are as follows:

$$y = \sin(\theta)$$
$$x = \sin(\theta + \Delta\theta)$$

Equation **GFLIB_AtanYXShifted_Eq1**

where:

- x, y are respectively, the f16InX, and f16InY arguments
- θ is the angle to be computed by the function
- Δθ is the phase difference between the x, y signals

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ. The angle offset is an additional parameter, which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ.

The GFLIB_AtanYXShifted_F16 function does not directly use the angle offset θ_{Offset} and the phase difference θ. The function's parameters, contained in the function parameters structure GFLIB_ATANYXSHIFTED_T_F16, need to be computed by means of the provided Matlab function (see below).

If Δθ= π/2 or Δθ=- π/2, then the function is similar to the GFLIB_AtanYX_F16 function, however, the GFLIB_AtanYX_F16 function in this case is more effective with regard to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define Δθ and θ_{Offset}, the Δθ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function
- convert the computed values into integer format and insert them into the C code (see also the C code example)

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)}\left(y + x\right)$$
$$y = \frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)}\left(x - y\right)$$
$$\theta = A\text{tanYX}\left(b, a\right) - \left(\frac{\Delta\theta}{2} - \theta_{\text{offset}}\right)$$

Equation **GFLIB_AtanYXShifted_Eq2**

where:

- x, y are respectively, the f16InX, and f16InY
- θ is the angle to be computed by the function, see the previous equation
- Δθ is the phase difference between the x, y signals, see the previous equation
- S is a scaling coefficient, S is almost 1, (S < 1), see also the explanation below
- a, b intermediate variables
- θ_{Offset} is the additional phase shift, the computed angle will be $\theta + \theta_{Offset}$

The scale coefficient S is used to prevent overflow and to assure symmetry around 0 for the entire fractional range. S shall be less than 1.0, but as large as possible. The algorithm implemented in this function uses the value of $1 - 2^{-15}$.

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan\left(\theta + \Delta\theta\right) = \frac{(y+x)\cos\frac{\Delta\theta}{2}}{(x-y)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanYXShifted_Eq3**

For the purposes of fractional arithmetic, the algorithm is implemented such that additional values are used as shown in the equation below:

$$\frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)} = C_y = K_y 2^{N_y}$$

$$\frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)} = C_x = K_x 2^{N_x}$$

$$\theta_{adj} = \frac{\Delta\theta}{2} - \theta_{offset}$$

Equation **GFLIB_AtanYXShifted_Eq4**

where:

- C_y, C_x are the algorithm coefficients for y and x signals
- K_y is multiplication coefficient of the y signal, represented by the parameters structure member pParam->f16Ky
- K_x is multiplication coefficient of the x signal, represented by the parameters structure member pParam->f16Kx
- N_y is scaling coefficient of the y signal, represented the by parameters structure member pParam->s16Ny
- N_x is scaling coefficient of the x signal, represented by the parameters structure member pParam->s16Nx
- θ_{adj} is an adjusting angle, represented by the parameters structure member pParam->f16ThetaAdj

The multiplication and scaling coefficients, and the adjusting angle, shall be defined in a parameters structure provided as the function input parameter.

The function initialization parameters can be calculated as shown in the following Matlab code:

While applying the function, some general guidelines should be considered as stated below.

```
function [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
      // ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
      //
      // [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
      //
      // dthdeg = phase shift (delta theta) between sine waves in degrees
      // thoffsetdeg = angle offset (theta offset) in degrees
      // NY - scaling coefficient of y signal
      // NX - scaling coefficient of x signal
      // KY - multiplication coefficient of y signal
      // KX - multiplication coefficient of x signal
      // THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

          if (dthdeg < -180) || (dthdeg >= 180)
              error('atanyxshiftedpar: dthdeg out of range');
          end
          if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
              error('atanyxshiftedpar: thoffsetdeg out of range');
          end

          dth2 = ((dthdeg/2)/180*pi);
          thoffset = (thoffsetdeg/180*pi);
          CY = (1 - 2^-15)/(2*cos(dth2));
          CX = (1 - 2^-15)/(2*sin(dth2));
          if(abs(CY) >= 1) NY = ceil(log2(abs(CY)));
          else NY = 0;
          end
          if(abs(CX) >= 1) NX = ceil(log2(abs(CX)));
          else NX = 0;
          end
          KY = CY/2^NY;
          KX = CX/2^NX;
          THETAADJ = dthdeg/2 - thoffsetdeg;

          if THETAADJ >= 180
              THETAADJ = THETAADJ - 360;
          elseif THETAADJ < -180
              THETAADJ = THETAADJ + 360;
          end

          THETAADJ = THETAADJ/180;

      return;
```

At some values of the phase shift, and particularly at phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error where possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion

- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift Δθ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should be as close to 1.0 as much as possible.

The function has been tested to be reliable at a phase shift in the range of [-165, -15] and [15, 165] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note

The function calls the GFLIB_AtanYX_F16 function. The function may become numerically unstable for a phase shift approaching -180, 0 or 180 degrees. The function accuracy is guaranteed for a phase shift in the range of [-165, -15] and [15, 165] degrees at perfect input signals.

4.38.5 Re-entrancy

The function is re-entrant.

4.38.6 Code Example

```
#include "gflib.h"

tFrac16 f16InY;
tFrac16 f16InX;
tFrac16 f16Ang;
GFLIB_ATANYXSHIFTED_T_F16 Param;

void main(void)
{
    //dtheta = 69.33deg, thetaoffset = 10deg
    //CY = (1 - 2^-15)/(2*cos((69.33/2)/180*pi))= 0.60789036201452440
    //CX = (1 - 2^-15)/(2*sin((69.33/2)/180*pi))= 0.87905201358520957
    //NY = 0 (abs(CY) < 1)
    //NX = 0 (abs(CX) < 1)
    //KY = 0.60789/2^0 = 0.60789036201452440
    //KX = 0.87905/2^0 = 0.87905201358520957
    //THETAADJ = 10/180 = 0.05555555555

    Param.f16Ky = FRAC16 (0.60789036201452440);
```

```
    Param.f16Kx = FRAC16 (0.87905201358520957);
    Param.s16Ny = 0;
    Param.s16Nx = 0;
    Param.f16ThetaAdj = FRAC16 (0.05555555555);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    f16InY = FRAC16 (0.2588190);
    f16InX = FRAC16 (0.9951074);

    // f16Ang contains 0x11c6c, the theoretical value is 0x1C34
    f16Ang = GFLIB_AtanYXShifted_F16 (f16InY, f16InX, &Param);

    // f16Ang contains 0x11c6, the theoretical value is 0x1C34
    f16Ang = GFLIB_AtanYXShifted (f16InY, f16InX, &Param, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // f16Ang contains 0x11c6, the theoretical value is 0x1C34
    f16Ang = GFLIB_AtanYXShifted (f16InY, f16InX, &Param);
}
```

4.39 Function GFLIB_AtanYXShifted_FLT

This function calculates the angle of two sine waves shifted in phase to each other.

4.39.1 Declaration

```
tFloat GFLIB_AtanYXShifted_FLT(tFloat fltInY, tFloat fltInX, GFLIB_ATANYXSHIFTED_T_FLT
*pParam);
```

4.39.2 Arguments

Table 4-49. GFLIB_AtanYXShifted_FLT arguments

Type	Name	Direction	Description
tFloat	fltInY	**input**	The value of the first signal, assumed to be sin(θ).
tFloat	fltInX	**input**	The value of the second signal, assumed to be sin(θ + Δθ).
GFLIB_ATANYXSHIFTED_T_FLT *	pParam	**input, output**	The parameters for the function.

4.39.3 Return

The function returns the angle of two sine waves shifted in phase to each other.

4.39.4 Description

The function calculates the angle of two sinusoidal signals, one shifted in phase to the other. The phase shift between sinusoidal signals does not have to be π/2 and can be any value.

It is assumed that the arguments of the function are as follows:

$$y = \sin(\theta)$$
$$x = \sin(\theta + \Delta\theta)$$

Equation **GFLIB_AtanYXShifted_Eq1**

where:

- x, y are respectively, the fltInX, and fltInY arguments
- θ is the angle to be computed by the function
- Δθ is the phase difference between the x, y signals

At the end of computations, an angle offset θ_{Offset} is added to the computed angle θ. The angle offset is an additional parameter which can be used to set the zero of the θ axis. If θ_{Offset} is zero, then the angle computed by the function will be exactly θ.

The GFLIB_AtanYXShifted_FLT function does not directly use the angle offset θ_{Offset} and the phase difference θ. The function's parameters, contained in the function parameters structure GFLIB_ATANYXSHIFTED_T_FLT, need to be computed by means of the provided Matlab function (see below).

If Δθ= π/2 or Δθ=- π/2, then the function is similar to the GFLIB_AtanYX_FLT function, however, the GFLIB_AtanYX_FLT function in this case is more effective with regards to execution time and accuracy.

In order to use the function, the following necessary steps need to be completed:

- define Δθ and θ_{Offset}, the Δθ shall be known from the input sinusoidal signals, the θ_{Offset} needs to be set arbitrarily
- compute values for the function parameters structure by means of the provided Matlab function

The function uses the following algorithm for computing the angle:

$$b = \frac{S}{2\cos\left(\frac{\Delta\theta}{2}\right)}\left(y + x\right)$$

$$y = \frac{S}{2\sin\left(\frac{\Delta\theta}{2}\right)}\left(x - y\right)$$

$$\theta = A\tan YX\left(b, a\right) - \left(\frac{\Delta\theta}{2} - \theta_{\text{offset}}\right)$$

Equation **GFLIB_AtanYXShifted_Eq2**

where:

- x, y are respectively, the fltInX, and fltInY
- θ is the angle to be computed by the function, see the previous equation
- Δθ is the phase difference between the x, y signals, see the previous equation
- a, b intermediate variables
- θ_{Offset} is the additional phase shift in radians, the computed angle will be $\theta + \theta_{Offset}$

The algorithm can be easily justified by proving the trigonometric identity:

$$\tan\left(\theta + \Delta\theta\right) = \frac{\left(y+x\right)\cos\frac{\Delta\theta}{2}}{\left(x-y\right)\sin\frac{\Delta\theta}{2}}$$

Equation **GFLIB_AtanYXShifted_Eq3**

The function initialization parameters can be calculated as shown in the following Matlab code:

```
function [KY, KX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
      // ATANYXSHIFTEDPAR calculation of parameters for atanyxshifted() function
      //
      // [KY, KX, NY, NX, THETAADJ] = atanyxshiftedpar(dthdeg, thoffsetdeg)
      //
      // dthdeg = phase shift (delta theta) between sine waves in degrees
      // thoffsetdeg = angle offset (theta offset) in degrees
      // KY - multiplication coefficient of y signal
      // KX - multiplication coefficient of x signal
      // THETAADJ - adjusting angle in radians, scaled from [-pi, pi) to [-1, 1)

          if (dthdeg < -180) || (dthdeg >= 180)
              error('atanyxshiftedpar: dthdeg out of range');
          end
          if (thoffsetdeg < -180) || (thoffsetdeg >= 180)
              error('atanyxshiftedpar: thoffsetdeg out of range');
          end

          dth2 = ((dthdeg/2)/180*pi);
          thoffset = (thoffsetdeg/180*pi);
          KY = 1/(2*cos(dth2));
          KX = 1/(2*sin(dth2));
          THETAADJ = dthdeg/2 - thoffsetdeg;

          if THETAADJ >= 180
              THETAADJ = THETAADJ - 360;
          elseif THETAADJ < -180
```

```
            THETAADJ = THETAADJ + 360;
        end

        THETAADJ = THETAADJ/180;

    return;
```

While applying the function, some general guidelines should be considered as stated below.

At some values of the phase shift, and particularly at a phase shift approaching -180, 0 or 180 degrees, the algorithm may become numerically unstable, causing any error, contributed by input signal imperfections or through finite precision arithmetic, to be magnified significantly. Therefore, some care should be taken to avoid error wherever possible. The detailed error analysis of the algorithm is beyond the scope of this documentation, however, general guidelines are provided.

There are several sources of error in the function:

- error of the supplied signal values due to the finite resolution of the AD conversion
- error contributed by higher order harmonics appearing in the input signals
- computational error of the multiplication due to the finite length of registers
- error of the phase shift $\Delta\theta$ representation in the finite precision arithmetic and in the values
- error due to differences in signal amplitudes

It should be noted that the function requires both signals to have the same amplitude. To minimize the output error, the amplitude of both signals should have the same amplitude.

The function has been tested to be reliable at a phase shift in the range of [-179, -1] and [1, 179] degrees for perfectly sinusoidal input signals. Beyond this range, the function operates correctly, however, the output error can be beyond the guaranteed value. In a real application, an error, contributed by an AD conversion and by higher order harmonics of the input signals, should be also taken into account.

Note

The function calls the GFLIB_AtanYX_FLT function. The function accuracy is guaranteed for a phase shift in the range of [-179, -1] and [1, 179] degrees at perfect input signals.

4.39.5 Re-entrancy

The function is re-entrant.

4.39.6 Code Example

```
#include "gflib.h"

tFloat fltInY;
tFloat fltInX;
tFloat fltAng;
GFLIB_ATANYXSHIFTED_T_FLT Param;

void main(void)
{
    //dtheta = 69.33deg, thetaoffset = 10deg
    //KY = 1/(2*cos((69.33/2)/180*pi))= 0.6079089139
    //KX = 1/(2*sin((69.33/2)/180*pi))= 0.8790788409

    //THETAADJ = 10*pi/180 = 0.1745329252

    Param.fltKy = (tFloat)(0.6079089139);
    Param.fltKx = (tFloat)(0.8790788409);
    Param.fltThetaAdj = (tFloat)(0.1745329252);

    // theta = 15 deg
    // Y = sin(theta) = 0.2588190
    // X = sin(theta + dtheta) = 0.9951074
    fltInY = (tFloat)(0.2588190);
    fltInX = (tFloat)(0.9951074);

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted_FLT (fltInY, fltInX, &Param);

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted (fltInY, fltInX, &Param, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // Output angle should be 0.69228911 rad
    fltAng = GFLIB_AtanYXShifted (fltInY, fltInX, &Param);
}
```

4.40 Function GFLIB_ControllerPIp_F32

This function calculates a parallel form of the Proportional-Integral controller, without integral anti-windup.

4.40.1 Declaration

```
tFrac32 GFLIB_ControllerPIp_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PI_P_T_F32 *const pParam);
```

4.40.2 Arguments

Table 4-50. GFLIB_ControllerPIp_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_P_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.40.3 Return

The function returns a 32-bit value in format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.40.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIp_F32 function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction.

An anti-windup strategy is not implemented in this function. Nevertheless, the accumulator overflow is prevented by correct saturation of the controller output at maximal values: [-1, 1) in fractional interpretation, or [-2^{31},2^{31}-1) in integer interpretation.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIp_Eq1**

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIp_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIp_Eq2**

The proportional part of equation GFLIB_ControllerPIp_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

Equation **GFLIB_ControllerPIp_Eq3**

Transforming the integral part of equation GFLIB_ControllerPIp_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIp_Eq4**

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIp_Eq3 results in:

$$u_{\mathrm{Pf}}(k) = e_f(k) \bullet K_{P\_SC} \quad \text{where} \quad K_{P\_SC} = K_P \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIp_Eq4 results in:

$$u_{\mathrm{If}}(k) = u_{\mathrm{If}}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_SC} = \frac{K_I T_S}{2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIp_Eq8**

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \bullet K_{P\_SC} + u_{\mathrm{If}}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

The problem is however, that either of the gain parameters K_{P_sc}, K_{I_sc} can be out of the [-1, 1) range, hence cannot be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f32\mathrm{PropGain} = K_{P\_sc} \bullet 2^{s16\mathrm{PropGainShift}}$$

Equation **GFLIB_ControllerPIp_Eq10**

and

$$f32\mathrm{IntegGain} = K_{I\_sc} \bullet 2^{s16\mathrm{IntegGainShift}}$$

Equation **GFLIB_ControllerPIp_Eq11**

where

- f32PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31,31]
- f32IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-31,31]

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_F32 macro.

4.40.5 Re-entrancy

The function is re-entrant.

4.40.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_P_T_F32 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32PropGain          = FRAC32 (0.01);
    trMyPI.f32IntegGain         = FRAC32 (0.02);
    trMyPI.s16PropGainShift     = 1;
    trMyPI.s16IntegGainShift    = 1;
    trMyPI.f32IntegPartK_1      = 0;

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIp_F32 (f32InErr, &trMyPI);

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIp (f32InErr, &trMyPI, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIp (f32InErr, &trMyPI);
}
```

4.41 Function GFLIB_ControllerPIp_F16

This function calculates a parallel form of the Proportional- Integral controller, without integral anti-windup.

4.41.1 Declaration

```
tFrac16 GFLIB_ControllerPIp_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PI_P_T_F16 *const pParam);
```

4.41.2 Arguments

Table 4-51. GFLIB_ControllerPIp_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_P_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.41.3 Return

The function returns a 16-bit value in format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.41.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIp_F16 function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction.

An anti-windup strategy is not implemented in this function. Nevertheless, the accumulator overflow is prevented by correct saturation of the controller output at maximal values: [-1, 1) in fractional interpretation, or [-2^{15},2^{15}-1) in integer interpretation.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIp_Eq1**

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIp_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIp_Eq2**

The proportional part of equation GFLIB_ControllerPIp_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

Equation **GFLIB_ControllerPIp_Eq3**

Transforming the integral part of equation GFLIB_ControllerPIp_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIp_Eq4**

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIp_Eq3 results in:

$$u_{Pf}(k) = e_f(k) \bullet K_{P\_SC} \quad \text{where} \quad K_{P\_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIp_Eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_SC} = \frac{K_I T_S}{2} \bullet \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIp_Eq8**

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \bullet K_{P\_SC} + u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

The problem is however, that either of the gain parameters K_{P_sc}, K_{I_sc} can be out of the [-1, 1) range, hence cannot be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f16PropGain = K_{P\_sc} \bullet 2^{s16PropGainShift}$$

Equation **GFLIB_ControllerPIp_Eq10**

and

$$f16IntegGain = K_{I\_sc} \bullet 2^{s16IntegGainShift}$$

Equation **GFLIB_ControllerPIp_Eq11**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-15, 15)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-15, 15)

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_F16 macro.

4.41.5 Re-entrancy

The function is re-entrant.

4.41.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PI_P_T_F16 trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16PropGain          = FRAC16 (0.01);
    trMyPI.f16IntegGain         = FRAC16 (0.02);
    trMyPI.s16PropGainShift     = 1;
    trMyPI.s16IntegGainShift    = 1;
    trMyPI.f32IntegPartK_1      = 0;

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp_F16 (f16InErr, &trMyPI);
```

```
    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp (f16InErr, &trMyPI, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIp (f16InErr, &trMyPI);
}
```

4.42 Function GFLIB_ControllerPIp_FLT

This function calculates a parallel form of the Proportional- Integral controller, without integral anti-windup.

4.42.1 Declaration

```
tFloat GFLIB_ControllerPIp_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PI_P_T_FLT *const pParam);
```

4.42.2 Arguments

Table 4-52. GFLIB_ControllerPIp_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller in single precision floating format.
GFLIB_CONTROLLER_PI_P_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.42.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.42.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIp_FLT function calculates the

Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction.

An anti-windup strategy is not implemented in this function.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIp_Eq1**

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIp_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIp_Eq2**

The proportional part of equation GFLIB_ControllerPIp_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

Equation **GFLIB_ControllerPIp_Eq3**

Transforming the integral part of equation GFLIB_ControllerPIp_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIp_Eq4**

where T_s [sec] is the sampling time.

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u_f(k) = e_f(k) \bullet K_{P\_SC} + u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIp_Eq9**

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_P_DEFAULT_FLT macro. As the PI controller also contains the integration part, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the integrator value from the previous step is not used for testing purposes and is replaced by the reference model integrator value in the previous step.

4.42.5 Re-entrancy

The function is re-entrant.

4.42.6 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PI_P_T_FLT trMyPI = GFLIB_CONTROLLER_PI_P_DEFAULT_FLT;

void main(void)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltPropGain        = (tFloat)(0.04);
    trMyPI.fltIntegGain       = (tFloat)(0.02);
    trMyPI.fltIntegPartK_1    = 0;

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIp_FLT (fltInErr, &trMyPI);

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIp (fltInErr, &trMyPI, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################
```

```
    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIp (fltInErr, &trMyPI);
}
```

4.43 Function GFLIB_ControllerPIpAW_F32

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

4.43.1 Declaration

```
tFrac32 GFLIB_ControllerPIpAW_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PIAW_P_T_F32 *const
pParam);
```

4.43.2 Arguments

Table 4-53. GFLIB_ControllerPIpAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_P_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.43.3 Return

The function returns a 32-bit value in format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.43.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIpAW function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction. The controller output is limited and the

limit values (f32UpperLimit and f32LowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PI controller parameters (GFLIB_CONTROLLER_PIAW_P_T_F32). If the PI controller output reaches the upper or lower limit then u16LimitFlag = 1, otherwise u16LimitFlag = 0 (integer values). An anti-windup strategy is implemented by limiting the integral portion. The integral state is limited by the controller limits, in the same way as the controller output.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIpAW_Eq1**

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIpAW_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIpAW_Eq2**

The proportional part of equation GFLIB_ControllerPIpAW_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

Equation **GFLIB_ControllerPIpAW_Eq3**

Transforming the integral part of equation GFLIB_ControllerPIpAW_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2} + e(k-1) \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIpAW_Eq4**

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIpAW_Eq3 results in:

$$u_{Pf}(k) = e_f(k) \bullet K_{P\_SC} \quad \text{where} \quad K_{P\_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIpAW_Eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_SC} = \frac{K_I T_S}{2} \bullet \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq8**

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller. The problem is however, that either of the gain parameters K_{P_sc}, K_{I_sc} can be out of the [-1, 1) range, hence can not be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f32PropGain = K_{P\_sc} \bullet 2^{s16PropGainShift}$$

Equation **GFLIB_ControllerPIpAW_Eq9**

$$f32IntegGain = K_{I\_sc} \bullet 2^{s16IntegGainShift}$$

Equation **GFLIB_ControllerPIpAW_Eq10**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-31, 31)

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f \bullet K_{P\_sc} + u_{1f}(k-1) + K_{I\_sc} \bullet e_f(k) + K_{I\_sc} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIpAW_Eq11**

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values f32UpperLimit, f32LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f32UpperLimit & \Rightarrow \quad u_f(k) \geq f32UpperLimit \\ u_f(k) & \Rightarrow \quad f32LowerLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & \Rightarrow \quad u_f(k) \leq f32LowerLimit \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_Eq12. When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 macro.

4.43.5 Re-entrancy

The function is re-entrant.

4.43.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_P_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32PropGain        = FRAC32 (0.01);
    trMyPI.f32IntegGain       = FRAC32 (0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f32IntegPartK_1    = FRAC32 (0);
    trMyPI.f32UpperLimit      = FRAC32 (1.0);
    trMyPI.f32LowerLimit      = FRAC32 (-1.0);

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIpAW_F32 (f32InErr, &trMyPI);

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIpAW (f32InErr, &trMyPI, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x01EB851E
    f32Output = GFLIB_ControllerPIpAW (f32InErr, &trMyPI);
```

4.44 Function GFLIB_ControllerPIpAW_F16

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

4.44.1 Declaration

```
tFrac16 GFLIB_ControllerPIpAW_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PIAW_P_T_F16 *const
pParam);
```

4.44.2 Arguments

Table 4-54. GFLIB_ControllerPIpAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_P_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.44.3 Return

The function returns a 16-bit value in format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.44.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIpAW function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction. The controller output is limited and the limit values (f16UpperLimit and f16LowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PI controller parameters

(GFLIB_CONTROLLER_PIAW_P_T_F16). If the PI controller output reaches the upper or lower limit then u16LimitFlag = 1, otherwise u16LimitFlag = 0 (integer values). An anti- windup strategy is implemented by limiting the integral portion. The integral state is limited by the controller limits, in the same way as the controller output.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIpAW_Eq1**

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIpAW_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIpAW_Eq2**

The proportional part of equation GFLIB_ControllerPIpAW_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

Equation **GFLIB_ControllerPIpAW_Eq3**

Transforming the integral part of equation GFLIB_ControllerPIpAW_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2} + e(k-1) \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIpAW_Eq4**

where T_s [sec] is the sampling time.

In order to implement the discrete equation of the controller on the fixed point arithmetic platform, the maximal values (scales) of input and output signals have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values:

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

The fractional representation of both input and output signals, normalized between [-1, 1), is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq5**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

Applying such scaling (normalization) on the proportional term of equation GFLIB_ControllerPIpAW_Eq3 results in:

$$u_{Pf}(k) = e_f(k) \bullet K_{P\_SC} \quad \text{where} \quad K_{P\_SC} = K_P \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq7**

where K_{P_sc} is the proportional gain parameter considering input/output scaling.

Analogically, scaling the integral term of equation GFLIB_ControllerPIpAW_Eq4 results in:

$$u_{If}(k) = u_{If}(k-1) + K_{I\_SC} \bullet e_f(k) + K_{I\_SC} \bullet e_f(k-1) \quad \text{where} \quad K_{I\_SC} = \frac{K_I T_S}{2} \bullet \frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIpAW_Eq8**

where K_{I_sc} is the integral gain parameter considering input/output scaling.

The sum of the scaled proportional and integral terms gives a complete equation of the controller. The problem is however, that either of the gain parameters K_{P_sc}, K_{I_sc} can be out of the [-1, 1) range, hence can not be directly interpreted as fractional values. To overcome this, it is necessary to scale these gain parameters using the shift values as follows:

$$f16PropGain = K_{P\_sc} \bullet 2^{s16PropGainShift}$$

Equation **GFLIB_ControllerPIpAW_Eq9**

$$f16IntegGain = K_{I\_sc} \bullet 2^{s16IntegGainShift}$$

Equation **GFLIB_ControllerPIpAW_Eq10**

where

- f16PropGain - is the scaled value of proportional gain [-1, 1)
- s16PropGainShift - is the scaling shift for proportional gain [-31, 31)
- f16IntegGain - is the scaled value of integral gain [-1, 1)
- s16IntegGainShift - is the scaling shift for integral gain [-31, 31)

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f \bullet K_{P\_sc} + u_{If}(k-1) + K_{I\_sc} \bullet e_f(k) + K_{I\_sc} \bullet e_f(k-1)$$

Equation **GFLIB_ControllerPIpAW_Eq11**

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values f16UpperLimit, f16LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} f16UpperLimit & \Rightarrow \quad u_f(k) \geq f16UpperLimit \\ u_f(k) & \Rightarrow \quad f16LowerLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & \Rightarrow \quad u_f(k) \leq f16LowerLimit \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_Eq12. When the bounds are exceeded the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the integral part accumulator (limitation during the calculation) and on the overall controller output. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 macro. To effectively reach the target precision the internal calculation is done in 32-bit fractional arithmetic and internal accumulator is 32-bit wide.

4.44.5 Re-entrancy

The function is re-entrant.

4.44.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_P_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16PropGain        = FRAC16 (0.01);
    trMyPI.f16IntegGain       = FRAC16 (0.02);
    trMyPI.s16PropGainShift   = 1;
    trMyPI.s16IntegGainShift  = 1;
    trMyPI.f32IntegPartK_1    = FRAC32 (0);
    trMyPI.f16UpperLimit      = FRAC16 (1.0);
    trMyPI.f16LowerLimit      = FRAC16 (-1.0);

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIpAW_F16 (f16InErr, &trMyPI);

    // output should be 0x01EB
    f16Output = GFLIB_ControllerPIpAW (f16InErr, &trMyPI, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
```

```
// as default
// ###############################################################

// output should be 0x01EB
f16Output = GFLIB_ControllerPIpAW (f16InErr, &trMyPI);
```

4.45 Function GFLIB_ControllerPIpAW_FLT

The function calculates the parallel form of the Proportional-Integral (PI) controller with implemented integral anti-windup functionality.

4.45.1 Declaration

```
tFloat GFLIB_ControllerPIpAW_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PIAW_P_T_FLT *const
pParam);
```

4.45.2 Arguments

Table 4-55. GFLIB_ControllerPIpAW_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller is a single precision floating point data type.
GFLIB_CONTROLLER_PIAW_P_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.45.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.45.4 Description

A PI controller attempts to correct the error between a measured process variable and a desired set-point by calculating and then outputting a corrective action that can adjust the process accordingly. The GFLIB_ControllerPIpAW function calculates the Proportional-Integral (PI) algorithm according to the equations below. The PI algorithm is

implemented in the parallel (non-interacting) form, allowing the user to define the P and I parameters independently without interaction. The controller output is limited, and the limit values (fltUpperLimit and fltLowerLimit) are defined by the user. The PI controller algorithm also returns a limitation flag. This flag (u16LimitFlag) is a member of the structure of the PI controller parameters (GFLIB_CONTROLLER_PIAW_P_T_FLT). If the PI controller output reaches the upper or lower limit then u16LimitFlag = 1, otherwise u16LimitFlag = 0 (integer values). An anti-windup strategy is implemented by limiting the controller output.

The PI algorithm in the continuous time domain can be described as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIpAW_Eq1**

where

- e(t) - input error in the continuous time domain
- u(t) - controller output in the continuous time domain
- K_P - proportional gain
- K_I - integral gain

Equation GFLIB_ControllerPIpAW_Eq1 can be described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = K_P + K_I \frac{1}{s}$$

Equation **GFLIB_ControllerPIpAW_Eq2**

The proportional part of equation GFLIB_ControllerPIpAW_Eq2 is transformed into the discrete time domain simply as:

$$u_P(k) = K_P \bullet e(k)$$

Equation **GFLIB_ControllerPIpAW_Eq3**

Transforming the integral part of equation GFLIB_ControllerPIpAW_Eq2 into a discrete time domain using the Bilinear method, also known as trapezoidal approximation, leads to the following equation:

$$u_I(k) = u_I(k-1) + e(k) \bullet \frac{K_I T_s}{2} + e(k-1) \frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIpAW_Eq4**

where T_s [sec] is the sampling time.

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values fltUpperLimit, fltLowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u(k) = \begin{cases} \text{fltUpperLimit} & \Rightarrow \quad u_f(k) \geq \text{fltUpperLimit} \\ u_f(k) & \Rightarrow \quad \text{fltLowerLimit} < u_f(k) < \text{fltUpperLimit} \\ \text{fltLowerLimit} & \Rightarrow \quad u_f(k) \leq \text{fltLowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIpAW_Eq12**

The sum of the scaled proportional and integral terms gives a complete equation of the controller:

$$u(k) = e_f(k) \bullet K_P + u_I(k-1) + e(k)\frac{K_I T_s}{2} + e(k-1)\frac{K_I T_s}{2}$$

Equation **GFLIB_ControllerPIpAW_Eq6**

The bounds are described by a limitation element equation GFLIB_ControllerPIpAW_Eq6. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the overall controller output. Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT macro. As the PI controller contains also the integration part, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the integrator value from the previous step is not used for testing purposes and is replaced by the reference model integrator value in the previous step.

4.45.5 Re-entrancy

The function is re-entrant.

4.45.6 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_P_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT;

void main(void)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltPropGain        = (tFloat)(0.04);
    trMyPI.fltIntegGain       = (tFloat)(0.02);
    trMyPI.fltIntegPartK_1    = 0;
    trMyPI.fltUpperLimit      = (tFloat)(1.0);
    trMyPI.fltLowerLimit      = (tFloat)(-1.0);

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIpAW_FLT (fltInErr, &trMyPI);

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIpAW (fltInErr, &trMyPI, Define FLT);

    // ##############################################################
    // Available only if single precision floating point implementation
    // selected as default
    // ##############################################################

    // output should be 1.5e-2
    fltOutput = GFLIB_ControllerPIpAW (fltInErr, &trMyPI);

}
```

4.46 Function GFLIB_ControllerPIr_F32

This function calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

4.46.1 Declaration

```
tFrac32 GFLIB_ControllerPIr_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PI_R_T_F32 *const pParam);
```

4.46.2 Arguments

Table 4-56. GFLIB_ControllerPIr_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_R_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.46.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.46.4 Description

The function GFLIB_ControllerPIr_F32 calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = c(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIr_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \bullet K_I}{s}$$

Equation **GFLIB_ControllerPIr_Eq2**

Transforming equation GFLIB_ControllerPIr_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet \mathrm{CC1} + e(k-1) \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIr_Eq3**

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are controller coefficients calculated depending on the discretization method used, as shown in Table 4-57.

Table 4-57. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p+K_I T_S/2$	$K_p+K_I T_S$	K_p
CC2=	$-K_p+K_I T_S/2$	$-K_p$	$-K_p+K_I T_S$

In order to implement the discrete equation of the controller GFLIB_ControllerPIr_Eq3 on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq4**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{MAX} = u_f(k-1) \bullet U^{MAX} + e_f(k) \bullet E^{MAX} \bullet CC1 + e_f(k-1) \bullet E^{MAX} \bullet CC2$$

Equation **GFLIB_ControllerPIr_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \bullet CC1_f + e_f(k-1) \bullet CC2_f$$

Equation **GFLIB_ControllerPIr_Eq7**

where

$$CC1_f = CC1\frac{E^{MAX}}{U^{MAX}} \quad CC2_f = CC2\frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both CC1 $_f$ and CC2 $_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E ^{MAX}, U ^{MAX}, the calculation of CC1 $_f$ and CC2 $_f$ may result in values outside this fractional range. Therefore, a scaling of CC1 $_f$, CC2 $_f$ is introduced as follows:

$$f32CC1sc = CC1_f \bullet 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq9**

$$f32CC2sc = CC2_f \bullet 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f32CC1sc, f32CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16Nshift = \max\left(\text{ceil}\left(\frac{\log(\text{abs}(CC1_f))}{\log(2)}\right), \text{ceil}\left(\frac{\log(\text{abs}(CC2_f))}{\log(2)}\right)\right)$$

Equation **GFLIB_ControllerPIr_Eq11**

The final, scaled, fractional equation of a recurrent PI controller on a 32-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16NShift}\right) = u_f(k-1) \bullet \left(2^{-u16NShift}\right) + e_f(k) \bullet f32CC1sc + e_f(k-1) \bullet f32CC2sc$$

Equation **GFLIB_ControllerPIr_Eq12**

where:

- $u_f(k)$ - fractional representation [-1, 1) of the controller output
- $e_f(k)$ - fractional representation [-1, 1) of the controller input (error)
- f32CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f32CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,31] - is chosen such that both coefficients f32CC1sc and f32CC2sc are in the range [-1, 1)

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_F32 macro.

4.46.5 Re-entrancy

The function is re-entrant.

4.46.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PI_R_T_F32 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32CC1sc  = FRAC32 (0.01);
    trMyPI.f32CC2sc  = FRAC32 (0.02);
    trMyPI.u16NShift = 1;

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr_F32 (f32InErr,&trMyPI);

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr (f32InErr,&trMyPI, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################
```

```
    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIr (f32InErr,&trMyPI);
}
```

4.47 Function GFLIB_ControllerPIr_F16

This function calculates a standard recurrent form of the Proportional- Integral controller, without integral anti-windup.

4.47.1 Declaration

```
tFrac16 GFLIB_ControllerPIr_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PI_R_T_F16 *const pParam);
```

4.47.2 Arguments

Table 4-58. GFLIB_ControllerPIr_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PI_R_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.47.3 Return

The function returns a 16-bit value in fractional format 1.15, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.47.4 Description

The function GFLIB_ControllerPIr_F16 calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = c(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{d}t$$

Equation **GFLIB_ControllerPIr_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \bullet K_I}{s}$$

Equation **GFLIB_ControllerPIr_Eq2**

Transforming equation GFLIB_ControllerPIr_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet CC1 + e(k-1) \bullet CC2$$

Equation **GFLIB_ControllerPIr_Eq3**

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are controller coefficients calculated depending on the discretization method used, as shown in Table 4-59.

Table 4-59. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p+K_I T_S/2$	$K_p+K_I T_S$	K_p
CC2=	$-K_p+K_I T_S/2$	$-K_p$	$-K_p+K_I T_S$

In order to implement the discrete equation of the controller GFLIB_ControllerPIr_Eq3 on the fixed point arithmetic platform, the maximal values (scales) of the input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq4**

$$u_f(k) = \frac{u(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{MAX} = u_f(k-1) \bullet U^{MAX} + e_f(k) \bullet E^{MAX} \bullet CC1 + e_f(k-1) \bullet E^{MAX} \bullet CC2$$

Equation **GFLIB_ControllerPIr_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k) \bullet CC1_f + e_f(k-1) \bullet CC2_f$$

Equation **GFLIB_ControllerPIr_Eq7**

where

$$CC1_f = CC1\frac{E^{MAX}}{U^{MAX}} \quad CC2_f = CC2\frac{E^{MAX}}{U^{MAX}}$$

Equation **GFLIB_ControllerPIr_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E^{MAX}, U^{MAX}, the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f16CC1sc = CC1_f \bullet 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq9**

$$f16CC2sc = CC2_f \bullet 2^{-u16Nshift}$$

Equation **GFLIB_ControllerPIr_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f16CC1sc, f16CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16\mathrm{Nshift} = \max\left(\mathrm{ceil}\left(\frac{\log(\mathrm{abs}(CC1_f))}{\log(2)}\right), \mathrm{ceil}\left(\frac{\log(\mathrm{abs}(CC2_f))}{\log(2)}\right)\right)$$

Equation **GFLIB_ControllerPIr_Eq11**

The final, scaled, fractional equation of a recurrent PI controller on a 16-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16\mathrm{NShift}}\right) = u_f(k-1) \bullet \left(2^{-u16\mathrm{NShift}}\right) + e_f(k) \bullet f16\mathrm{CC1sc} + e_f(k-1) \bullet f16\mathrm{CC2sc}$$

Equation **GFLIB_ControllerPIr_Eq12**

where:

- u $_f$(k) - fractional representation [-1, 1) of the controller output
- e $_f$(k) - fractional representation [-1, 1) of the controller input (error)
- f16CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f16CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,15] - is chosen such that both coefficients f16CC1sc and f16CC2sc are in the range [-1, 1)

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_F16 macro.

4.47.5 Re-entrancy

The function is re-entrant.

4.47.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;
```

```
GFLIB_CONTROLLER_PI_R_T_F16 trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16CC1sc  = FRAC16 (0.01);
    trMyPI.f16CC2sc  = FRAC16 (0.02);
    trMyPI.u16NShift = 1;

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr_F16 (f16InErr,&trMyPI);

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr (f16InErr,&trMyPI, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIr (f16InErr,&trMyPI);
}
```

4.48 Function GFLIB_ControllerPIr_FLT

This function calculates a standard recurrent form of the Proportional- Integral controller, without integral anti-windup.

4.48.1 Declaration

```
tFloat GFLIB_ControllerPIr_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PI_R_T_FLT *const pParam);
```

4.48.2 Arguments

Table 4-60. GFLIB_ControllerPIr_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller as a single precision floating point value.
GFLIB_CONTROLLER_PI_R_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.48.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.48.4 Description

The function GFLIB_ControllerPIr_FLT calculates a standard recurrent form of the Proportional-Integral controller, without integral anti-windup. The continuous time domain representation of the PI controller is defined as:

$$u(t) = c(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIr_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \bullet K_I}{s}$$

Equation **GFLIB_ControllerPIr_Eq2**

Transforming equation GFLIB_ControllerPIr_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet \mathrm{CC1} + e(k-1) \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIr_Eq3**

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are controller coefficients calculated depending on the discretization method used, as shown in Table 4-61.

Table 4-61. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p+K_I T_S/2$	$K_p+K_I T_S$	K_p
CC2=	$-K_p+K_I T_S/2$	$-K_p$	$-K_p+K_I T_S$

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PI_R_DEFAULT_FLT macro. As the PI controller also contains the integration part, the output result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

4.48.5 Re-entrancy

The function is re-entrant.

4.48.6 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PI_R_T_FLT trMyPI = GFLIB_CONTROLLER_PI_R_DEFAULT_FLT;

void main(void)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltCC1sc           = (tFloat)(0.01);
    trMyPI.fltCC2sc           = (tFloat)(0.01);

    // output should be 5e-3
    fltOutput = GFLIB_ControllerPIr_FLT (fltInErr,&trMyPI);

    // output should be 5e-3
    fltOutput = GFLIB_ControllerPIr (fltInErr,&trMyPI, Define FLT);

    // ###############################################################
    // Available only if single precision floating point implementation
    // selected as default
    // ###############################################################

    // output should be 5e-3
    fltOutput = GFLIB_ControllerPIr (fltInErr,&trMyPI);
}
```

4.49 Function GFLIB_ControllerPIrAW_F32

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

4.49.1 Declaration

```
tFrac32 GFLIB_ControllerPIrAW_F32(tFrac32 f32InErr, GFLIB_CONTROLLER_PIAW_R_T_F32 *const
pParam);
```

4.49.2 Arguments

Table 4-62. GFLIB_ControllerPIrAW_F32 arguments

Type	Name	Direction	Description
tFrac32	f32InErr	**input**	Input error signal to the controller is a 32-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_R_T_F32 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.49.3 Return

The function returns a 32-bit value in fractional format 1.31, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.49.4 Description

The function GFLIB_ControllerPIrAW calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIrAW_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \bullet K_I}{s}$$

Equation **GFLIB_ControllerPIrAW_Eq2**

Transforming equation GFLIB_ControllerPIrAW_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet CC1 + e(k-1) \bullet CC2$$

Equation **GFLIB_ControllerPIrAW_Eq3**

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are the controller coefficients calculated depending on the discretization method used, as shown in Table 4-63.

Table 4-63. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p + K_i T_s/2$	$K_p + K_i T_s$	K_p
CC2=	$-K_p + K_i T_s/2$	$-K_p$	$-K_p + K_i T_s$

In order to implement the discrete equation of the controller GFLIB_ControllerPIrAW_Eq3 on the fixed point arithmetic platform, the maximal values (scales) of input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIrAW_Eq4**

$$u_f(k) = \frac{e(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIrAW_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{\mathrm{MAX}} = u_f(k-1) \bullet U^{\mathrm{MAX}} + e_f \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC1} + e_f(k-1) \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIrAW_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k)\mathrm{CC1}_f + e_f(k-1) \bullet \mathrm{CC2}_f$$

Equation **GFLIB_ControllerPIrAW_Eq7**

where

$$\mathrm{CC1}_f = \mathrm{CC1} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}} \quad \mathrm{CC2}_f = \mathrm{CC2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both CC1 $_f$ and CC2 $_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E ^{MAX}, U ^{MAX}, the calculation of CC1 $_f$ and CC2 $_f$ may result in values outside this fractional range. Therefore, a scaling of CC1 $_f$, CC2 $_f$ is introduced as follows:

$$f32\mathrm{CC1}_{\mathrm{SC}} = \mathrm{CC1}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq9**

$$f32\mathrm{CC2}_{\mathrm{SC}} = \mathrm{CC2}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f32CC1sc,f32CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation. Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16NShift = \max\left(\text{ceil}\left(\frac{\log\left(\text{abs}\left(CC1_f\right)\right)}{\log(2)}\right) \ , \ \text{ceil}\frac{\log\left(\text{abs}\left(CC2_f\right)\right)}{\log(2)}\right)$$

Equation **GFLIB_ControllerPIrAW_Eq11**

The final, scaled, fractional equation of the recurrent PI controller on a 32-bit fixed point platform is therefore implemented as follows:

$$u_f(k) \bullet \left(2^{-u16NShift}\right) = u_f(k-1) \bullet \left(2^{-u16NShift}\right) + e_f(k) \bullet f32CC1_{SC} + e_f(k-1) \bullet f32CC2_{SC}$$

Equation **GFLIB_ControllerPIrAW_Eq12**

where:

- u $_f$(k) - fractional representation [-1, 1) of the controller output
- e $_f$(k) - fractional representation [-1, 1) of the controller input (error)
- f32CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f32CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,31] - is chosen such that both coefficients f32CC1sc and f32CC2sc are in the range [-1, 1)

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u_f\left(k\right) = \begin{cases} f32UpperLimit & \Rightarrow \quad u_f(k) \geq f32UpperLimit \\ u_f(k) & \Rightarrow \quad f32LowerLimit < u_f(k) < f32UpperLimit \\ f32LowerLimit & \Rightarrow \quad u_f(k) \leq f32LowerLimit \end{cases}$$

Equation **GFLIB_ControllerPIrAW_Eq13**

The bounds are described by a limitation element equation GFLIB_ControllerPIrAW_Eq13. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the internal controller accumulator (limitation

during the calculation). Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 macro.

4.49.5 Re-entrancy

The function is re-entrant.

4.49.6 Code Example

```
#include "gflib.h"

tFrac32 f32InErr;
tFrac32 f32Output;

GFLIB_CONTROLLER_PIAW_R_T_F32 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32;

void main(void)
{
    // input error = 0.25
    f32InErr  = FRAC32 (0.25);

    // controller parameters
    trMyPI.f32CC1sc           = FRAC32 (0.01);
    trMyPI.f32CC2sc           = FRAC32 (0.02);
    trMyPI.u16NShift          = 1;
    trMyPI.f32UpperLimit      = FRAC32 (1.0);
    trMyPI.f32LowerLimit      = FRAC32 (-1.0);

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW_F32 (f32InErr, &trMyPI);

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW (f32InErr, &trMyPI, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x00A3D70A
    f32Output = GFLIB_ControllerPIrAW (f32InErr, &trMyPI);
}
```

4.50 Function GFLIB_ControllerPIrAW_F16

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

4.50.1 Declaration

```
tFrac16 GFLIB_ControllerPIrAW_F16(tFrac16 f16InErr, GFLIB_CONTROLLER_PIAW_R_T_F16 *const
pParam);
```

4.50.2 Arguments

Table 4-64. GFLIB_ControllerPIrAW_F16 arguments

Type	Name	Direction	Description
tFrac16	f16InErr	**input**	Input error signal to the controller is a 16-bit number normalized between [-1, 1).
GFLIB_CONTROLLER_PIAW_R_T_F16 *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.50.3 Return

The function returns a 16-bit value in fractional format 1.16, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.50.4 Description

The function GFLIB_ControllerPIrAW calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIrAW_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \bullet K_I}{s}$$

Equation **GFLIB_ControllerPIrAW_Eq2**

Transforming equation GFLIB_ControllerPIrAW_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet CC1 + e(k-1) \bullet CC2$$

Equation **GFLIB_ControllerPIrAW_Eq3**

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are the controller coefficients calculated depending on the discretization method used, as shown in Table 4-65.

Table 4-65. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p + K_i T_s/2$	$K_p + K_i T_s$	K_p
CC2=	$-K_p + K_i T_s/2$	$-K_p$	$-K_p + K_i T_s$

In order to implement the discrete equation of the controller GFLIB_ControllerPIrAW_Eq3 on the fixed point arithmetic platform, the maximal values (scales) of input and output signals

- E^{MAX} - maximal value of the controller input error signal
- U^{MAX} - maximal value of the controller output signal

have to be known a priori. This is essential for correct casting of the physical signal values into fixed point values [-1, 1).

Then the fractional representation [-1, 1) of both input and output signals is obtained as follows:

$$e_f(k) = \frac{e(k)}{E^{MAX}}$$

Equation **GFLIB_ControllerPIrAW_Eq4**

$$u_f(k) = \frac{e(k)}{U^{MAX}}$$

Equation **GFLIB_ControllerPIrAW_Eq5**

The resulting controller discrete time domain equation in fixed point fractional representation is therefore given as:

$$u_f(k) \bullet U^{\mathrm{MAX}} = u_f(k-1) \bullet U^{\mathrm{MAX}} + e_f \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC1} + e_f(k-1) \bullet E^{\mathrm{MAX}} \bullet \mathrm{CC2}$$

Equation **GFLIB_ControllerPIrAW_Eq6**

which can be rearranged into the following form:

$$u_f(k) = u_f(k-1) + e_f(k)\mathrm{CC1}_f + e_f(k-1) \bullet \mathrm{CC2}_f$$

Equation **GFLIB_ControllerPIrAW_Eq7**

where

$$\mathrm{CC1}_f = \mathrm{CC1} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}} \quad \mathrm{CC2}_f = \mathrm{CC2} \bullet \frac{E^{\mathrm{MAX}}}{U^{\mathrm{MAX}}}$$

Equation **GFLIB_ControllerPIrAW_Eq8**

are the controller coefficients adapted according to the input and output scale values. In order to implement both coefficients as fractional numbers, both $CC1_f$ and $CC2_f$ must reside in the fractional range [-1, 1). However, depending on values CC1, CC2, E^{MAX}, U^{MAX}, the calculation of $CC1_f$ and $CC2_f$ may result in values outside this fractional range. Therefore, a scaling of $CC1_f$, $CC2_f$ is introduced as follows:

$$f16\mathrm{CC1}_{\mathrm{SC}} = \mathrm{CC1}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq9**

$$f16\mathrm{CC2}_{\mathrm{SC}} = \mathrm{CC2}_f \bullet 2^{-u16\mathrm{NShift}}$$

Equation **GFLIB_ControllerPIrAW_Eq10**

The introduced scaling shift u16NShift is chosen such that both coefficients f16CC1sc,f16CC2sc reside in the range [-1, 1). To simplify the implementation, this scaling shift is chosen to be a power of 2, so the final scaling is a simple shift operation.

Moreover, the scaling shift cannot be a negative number, so the operation of scaling is always to scale numbers with an absolute value larger than 1 down to fit in the range [-1, 1).

$$u16NShift = \max\left(\mathrm{ceil}\left(\frac{\log\left(\mathrm{abs}\left(CC1_f\right)\right)}{\log(2)}\right),\ \mathrm{ceil}\frac{\log\left(\mathrm{abs}\left(CC2_f\right)\right)}{\log(2)}\right)$$

Equation **GFLIB_ControllerPIrAW_Eq11**

The final, scaled, fractional equation of the recurrent PI controller on a 16-bit fixed point platform is therefore implemented as follows:

$$u_f(k)\bullet\left(2^{-u16NShift}\right) = u_f(k-1)\bullet\left(2^{-u16NShift}\right) + e_f(k)\bullet f16CC1_{SC} + e_f(k-1)\bullet f16CC2_{SC}$$

Equation **GFLIB_ControllerPIrAW_Eq12**

where:

- u $_f$(k) - fractional representation [-1, 1) of the controller output
- e $_f$(k) - fractional representation [-1, 1) of the controller input (error)
- f16CC1sc - fractional representation [-1, 1) of the 1st controller coefficient
- f16CC2sc - fractional representation [-1, 1) of the 2nd controller coefficient
- u16NShift - in range [0,15] - is chosen such that both coefficients f16CC1sc and f16CC2sc are in the range [-1, 1)

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

$$u_f(k) = \begin{cases} f16UpperLimit & \Rightarrow \quad u_f(k) \geq f16UpperLimit \\ u_f(k) & \Rightarrow \quad f16LowerLimit < u_f(k) < f16UpperLimit \\ f16LowerLimit & \Rightarrow \quad u_f(k) \leq f16LowerLimit \end{cases}$$

Equation **GFLIB_ControllerPIrAW_Eq13**

The bounds are described by a limitation element equation GFLIB_ControllerPIrAW_Eq13. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented on the internal controller accumulator (limitation during the calculation). Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 macro.

4.50.5 Re-entrancy

The function is re-entrant.

4.50.6 Code Example

```
#include "gflib.h"

tFrac16 f16InErr;
tFrac16 f16Output;

GFLIB_CONTROLLER_PIAW_R_T_F16 trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16;

void main(void)
{
    // input error = 0.25
    f16InErr  = FRAC16 (0.25);

    // controller parameters
    trMyPI.f16CC1sc           = FRAC16 (0.01);
    trMyPI.f16CC2sc           = FRAC16 (0.02);
    trMyPI.u16NShift          = 1;
    trMyPI.f16UpperLimit      = FRAC16 (1.0);
    trMyPI.f16LowerLimit      = FRAC16 (-1.0);

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW_F16 (f16InErr, &trMyPI);

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW (f16InErr, &trMyPI, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x00A3
    f16Output = GFLIB_ControllerPIrAW (f16InErr, &trMyPI);
}
```

4.51 Function GFLIB_ControllerPIrAW_FLT

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

4.51.1 Declaration

```
tFloat GFLIB_ControllerPIrAW_FLT(tFloat fltInErr, GFLIB_CONTROLLER_PIAW_R_T_FLT *const
pParam);
```

4.51.2 Arguments

Table 4-66. GFLIB_ControllerPIrAW_FLT arguments

Type	Name	Direction	Description
tFloat	fltInErr	**input**	Input error signal to the controller in single precision floating point data format.
GFLIB_CONTROLLER_PIAW_R_T_FLT *const	pParam	**input, output**	Pointer to the controller parameters structure.

4.51.3 Return

The function returns a single precision floating point value, representing the signal to be applied to the controlled system so that the input error is forced to zero.

4.51.4 Description

The function GFLIB_ControllerPIrAW calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

The continuous time domain representation of the PI controller is defined as:

$$u(t) = e(t) \bullet K_P + K_I \int_0^t e(t) \mathrm{dt}$$

Equation **GFLIB_ControllerPIrAW_Eq1**

The transfer function for this kind of PI controller, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{K_P + s \bullet K_I}{s}$$

Equation **GFLIB_ControllerPIrAW_Eq2**

Transforming equation GFLIB_ControllerPIrAW_Eq2 into a discrete time domain leads to the following equation:

$$u(k) = u(k-1) + e(k) \bullet CC1 + e(k-1) \bullet CC2$$

Equation **GFLIB_ControllerPIrAW_Eq3**

where Kp is proportional gain, Ki is integral gain, Ts is the sampling period, u(k) is the controller output, e(k) is the controller input error signal, CC1 and CC2 are the controller coefficients calculated depending on the discretization method used, as shown in Table 4-67.

The output signal limitation is implemented in this controller. The actual output u(k) is bounded not to exceed the given limit values UpperLimit, LowerLimit. This is due to either the bounded power of the actuator or to the physical constraints of the plant.

Table 4-67. Calculation of coefficients CC1 and CC2 using various discretization methods

	Trapezoidal	Backward Rect.	Forward Rect.
CC1=	$K_p+K_iT_s/2$	$K_p+K_iT_s$	K_p
CC2=	$-K_p+K_iT_s/2$	$-K_p$	$-K_p+K_iT_s$

$$u_f(k) = \begin{cases} \text{fltUpperLimit} & \Rightarrow \quad u_f(k) \geq \text{fltUpperLimit} \\ u_f(k) & \Rightarrow \quad \text{fltLowerLimit} < u_f(k) < \text{fltUpperLimit} \\ \text{fltLowerLimit} & \Rightarrow \quad u_f(k) \leq \text{fltLowerLimit} \end{cases}$$

Equation **GFLIB_ControllerPIrAW_Eq13**

The bounds are described by a limitation element equation GFLIB_ControllerPIrAW_Eq4. When the bounds are exceeded, the non-linear saturation characteristic will take effect and influence the dynamic behavior. The described limitation is implemented in the internal controller accumulator (limitation during the calculation). Therefore, if the limitation occurs, the controller output is clipped to its bounds and the wind-up occurrence of the accumulator portion is avoided by saturating the actual sum.

Note

All controller parameters and states can be reset during declaration using the GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT macro. As the PI controller also contains the integration part, the output

result is saddled by cumulative error. To enumerate the computation error in one calculation cycle, the internal accumulator is not used for testing purposes and is replaced by output from the previous calculation step of the reference model.

4.51.5 Re-entrancy

The function is re-entrant.

4.51.6 Code Example

```
#include "gflib.h"

tFloat fltInErr;
tFloat fltOutput;

GFLIB_CONTROLLER_PIAW_R_T_FLT trMyPI = GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT;

void main(void)
{
    // input error = 0.25
    fltInErr  = (tFloat)(0.25);

    // controller parameters
    trMyPI.fltCC1sc           = (tFloat)(0.01);
    trMyPI.fltCC2sc           = (tFloat)(0.01);
    trMyPI.fltUpperLimit      = (tFloat)(1.0);
    trMyPI.fltLowerLimit      = (tFloat)(-1.0);

    // output should be 5e-3
    fltOutput = GFLIB_ControllerPIrAW_FLT (fltInErr, &trMyPI);

    // output should be 5e-3
    fltOutput = GFLIB_ControllerPIrAW (fltInErr, &trMyPI, Define FLT);

    // ##############################################################
    // Available only if single precision floating point implementation
    // selected as default
    // ##############################################################

    // output should be 5e-3
    fltOutput = GFLIB_ControllerPIrAW (fltInErr, &trMyPI);
}
```

4.52 Function GFLIB_Cos_F32

This function implements polynomial approximation of cosine function.

4.52.1 Declaration

```
tFrac32 GFLIB_Cos_F32(tFrac32 f32In, const GFLIB_COS_T_F32 *const pParam);
```

4.52.2 Arguments

Table 4-68. GFLIB_Cos_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_COS_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.52.3 Return

The function returns the cos of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

4.52.4 Description

The GFLIB_Cos_F32 function provides a computational method for calculation of a standard trigonometric *cosine* function cos(x), using the 9th order Taylor polynomial approximation of the *sine* function. The following two equations describe the chosen approach of calculating the *cosine* function:

$$\cos(f32In) = \sin\left(\frac{\pi}{2} + f32In\right) = \sin(x)$$

Equation **GFLIB_Cos_Eq1**

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \_ = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Cos_Eq2**

The 9th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation. Accuracy criterion is to have the output error within 3LSB on the upper 16 bits of the 32-bit result.

Because the GFLIB_Cos_F32 function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f32In [-1, 1) into the correct range [- π, π), the input f32In must be multiplied by π. So, the fixed point fractional implementation of the GFLIB_Sin_F32 function, using 9th order Taylor approximation, is given as follows:

$$\sin(\pi \bullet f32In) = (\pi \bullet f32In) - \frac{(\pi \bullet f32In)^3}{3!} + \frac{(\pi \bullet f32In)^5}{5!} - \frac{(\pi \bullet f32In)^7}{7!} + \frac{(\pi \bullet f32In)^9}{9!}$$

Equation **GFLIB_Cos_Eq3**

The 9th order polynomial approximation of the sine function has a very good accuracy in the range [- π/2, π/2) of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function sin(x) = sin(π - x) is utilized. Therefore, the input argument is transferred to be always in the range [- π/2, π/2) and the Taylor polynomial is calculated only in the range of the argument [- π/2, π/2).

To make calculations more precise, the given argument value f32In (that is to be transferred into the range [-0.5, 0.5) due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of f32In ^{2}, used in the calculations, is in the range [-1, 1) instead of [-0.25, 0.25]. Shifting the input value by 1 bit to the left will increase the accuracy of the calculated sin(π * f32In) function. Implementing such a scale on the approximation function described by equation GFLIB_Cos_Eq2, results in the following:

$$\sin\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right) = -\left(\frac{(f32In \bullet 2 \bullet \frac{\pi}{2})}{2} - \frac{(f32In \bullet 2 \bullet \frac{\pi}{2})^3}{3!2} + \frac{(f32In \bullet 2 \bullet \frac{\pi}{2})^5}{5!2} - \frac{(f32In \bullet 2 \bullet \frac{\pi}{2})^7}{7!2} + \frac{(f32In \bullet 2 \bullet \frac{\pi}{2})^9}{9!2}\right) \bullet 2$$

Equation **GFLIB_Cos_Eq4**

Equation GFLIB_Cos_Eq3 can be further rewritten into the following form:

$$\begin{aligned}\sin(f32In \bullet \pi) = {} & (f32In \bullet 2)(a_1 + \\ & + (f32In \bullet 2)^2(a_2 + \\ & + (f32In \bullet 2)^2(a_3 + \\ & + (f32In \bullet 2)^2(a_4 + \\ & + (f32In \bullet 2)^2(a_5)))))) \bullet 2\end{aligned}$$

Equation **GFLIB_Cos_Eq5**

where a $_1$...a $_5$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 32-bit signed fractional numbers):

$$a_1 = \frac{\frac{\pi}{2}}{2} = 0.785398163397448 \Rightarrow \frac{\left(\frac{\pi}{2}\right)}{2} \bullet 2^{31} = 0x6487ED51$$

$$a_2 = -\frac{\left(\frac{\pi}{2}\right)^3}{3!2} = -0.322982048753123 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^3}{3!2} \bullet 2^{31} = 0xD6A88634$$

$$a_3 = \frac{\left(\frac{\pi}{2}\right)^5}{5!2} = 0.03988463131230835 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^5}{5!2} \bullet 2^{31} = 0x0519AF1A$$

$$a_4 = -\frac{\left(\frac{\pi}{2}\right)^7}{7!2} = -0.00234087706765934 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^7}{7!2} \bullet 2^{31} = 0xFFB34B4D$$

$$a_5 = \frac{\left(\frac{\pi}{2}\right)^9}{9!2} = 8.02205923936799e^{-005} \Rightarrow \frac{\left(\frac{\pi}{2}\right)^9}{9!2} \bullet 2^{31} = 0x0002A0F0$$

Equation **GFLIB_Cos_Eq6**

Therefore, the resulting equation has the following form:

$$\begin{aligned}\sin(f32In \bullet \pi) = {} & (f32In \bullet 2)(0x6487ED51 + \\ & + (f32In \bullet 2)^2(0xD6A88634 + \\ & + (f32In \bullet 2)^2(0x0519AF1A + \\ & + (f32In \bullet 2)^2(0xFFB34B4D + \\ & + (f32In \bullet 2)^2(0x0002A0F0)))))) \bullet 2\end{aligned}$$

Equation **GFLIB_Cos_Eq7**

Figure 4-25 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_F32, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

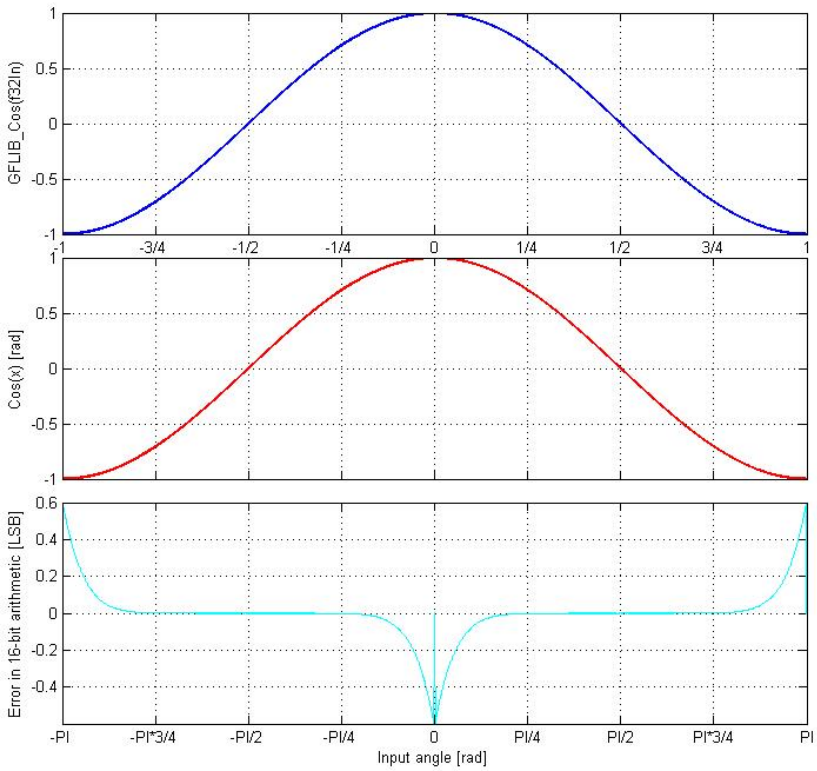

Figure 4-25. cos(x) vs. GFLIB_Cos(f32In)

Note

The input angle (f32In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Cos_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Cos(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(f32In, &pParam), where &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default

GFLIB_COS_DEFAULT_F32 approximation coefficients are used.

4.52.5 Re-entrancy

The function is re-entrant.

4.52.6 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
  // input angle = 0.25 => pi/4
  f32Angle  = FRAC32 (0.25);

  // output should be 0x5A824000
  f32Output = GFLIB_Cos_F32 (f32Angle, GFLIB_COS_DEFAULT_F32);

  // output should be 0x5A824000
  f32Output = GFLIB_Cos (f32Angle, GFLIB_COS_DEFAULT_F32, Define F32);

  // ##################################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##################################################################

  // output should be 0x5A824000
  f32Output = GFLIB_Cos (f32Angle);
}
```

4.53 Function GFLIB_Cos_F16

This function implements polynomial approximation of cosine function.

4.53.1 Declaration

```
tFrac16 GFLIB_Cos_F16(tFrac16 f16In, const GFLIB_COS_T_F16 *const pParam);
```

4.53.2 Arguments

Table 4-69. GFLIB_Cos_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_COS_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.53.3 Return

The function returns the cos of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

4.53.4 Description

The GFLIB_Cos_F16 function provides a computational method for calculation of a standard trigonometric *cosine* function cos(x), using the 7th order Taylor polynomial approximation of the *sine* function. The following two equations describe the chosen approach of calculating the *cosine* function:

$$\cos(f16In) = \sin\left(\frac{\pi}{2} + f16In\right) = \sin(x)$$

Equation **GFLIB_Cos_Eq1**

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \_ = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Cos_Eq2**

The 7th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation.

Because the GFLIB_Cos_F16 function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f16In [-1, 1) into the correct range

[- π, π), the input f16In must be multiplied by π. So, the fixed point fractional implementation of the GFLIB_Sin_F16 function, using 9th order Taylor approximation, is given as follows:

$$\sin\left(\pi \bullet f16In\right) = \left(\pi \bullet f16In\right) - \frac{\left(\pi \bullet f16In\right)^3}{3!} + \frac{\left(\pi \bullet f16In\right)^5}{5!} - \frac{\left(\pi \bullet f16In\right)^7}{7!}$$

Equation **GFLIB_Cos_Eq3**

The 7th order polynomial approximation of the sine function has a good accuracy in the range [- π/2, π/2) of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function sin(x) = sin(π - x) is utilized. Therefore, the input argument is transferred to be always in the range [- π/2, π/2) and the Taylor polynomial is calculated only in the range of the argument [- π/2, π/2).

To make calculations more precise, the given argument value f16In (that is to be transferred into the range [-0.5,0.5) due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of f16In ^{2}, used in the calculations, is in the range [-1, 1) instead of [-0.25, 0.25]. Shifting the input value by 1 bit to the left will increase the accuracy of the calculated sin(π * f16In) function. Implementing such a scale on the approximation function described by equation GFLIB_Cos_Eq2, results in the following:

$$\sin\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right) = \_ = \left(\frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)}{2} - \frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)^3}{3 \bullet 2} + \frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)^5}{5 \bullet 2} - \frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)^7}{7 \bullet 2}\right) \bullet 2$$

Equation **GFLIB_Cos_Eq4**

Equation GFLIB_Cos_Eq3 can be further rewritten into the following form:

$$\begin{aligned}\sin\left(f16In \bullet \pi\right) \;=\; & \left(f16In \bullet 2\right)\left(a_1 + \right. \\ & + \left(f16In \bullet 2\right)^2\left(a_2 + \right. \\ & + \left(f16In \bullet 2\right)^2\left(a_3 + \right. \\ & + \left(f16In \bullet 2\right)^2\left(a_4\right)\left.\right)\left.\right)\left.\right) \bullet 2\end{aligned}$$

Equation **GFLIB_Cos_Eq5**

where a $_1$...a $_5$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 16-bit signed fractional numbers):

$$a_1 = \frac{\frac{\pi}{2}}{2} = 0.785398163397448 \Rightarrow \frac{\left(\frac{\pi}{2}\right)}{2} \bullet 2^{15} = 0x6487$$
$$a_2 = -\frac{\left(\frac{\pi}{2}\right)^3}{3!2} = -0.322982048753123 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^3}{3!2} \bullet 2^{15} = 0xD6A9$$
$$a_3 = \frac{\left(\frac{\pi}{2}\right)^5}{5!2} = 0.03988463131230835 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^5}{5!2} \bullet 2^{15} = 0x051A$$
$$a_4 = -\frac{\left(\frac{\pi}{2}\right)^7}{7!2} = -0.00234087706765934 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^7}{7!2} \bullet 2^{15} = 0xFFB3$$

Equation **GFLIB_Cos_Eq6**

Therefore, the resulting equation has the following form:

$$\begin{aligned}\sin(f16In \bullet \pi) = {} & (f16In \bullet 2)(0x6488 + \\ & + (f16In \bullet 2)^2(0xD6A9 + \\ & + (f16In \bullet 2)^2(0x051A + \\ & + (f16In \bullet 2)^2(0xFFB3)))))) \bullet 2\end{aligned}$$

Equation **GFLIB_Cos_Eq7**

Figure 4-26 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_F16, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

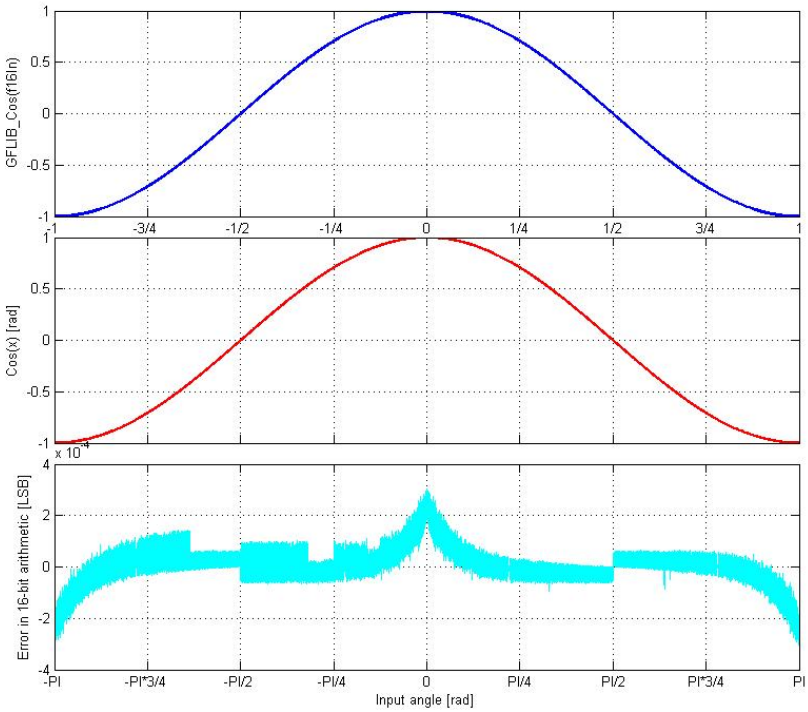

Figure 4-26. cos(x) vs. GFLIB_Cos(f16In)

Note

The input angle (f16In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Cos_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Cos(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_COS_DEFAULT_F16 approximation coefficients are used.

4.53.5 Re-entrancy

The function is re-entrant.

4.53.6 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
  // input angle = 0.25 => pi/4
  f16Angle  = FRAC16 (0.25);

  // output should be 0x5A82
  f16Output = GFLIB_Cos_F16 (f16Angle, GFLIB_COS_DEFAULT_F16);

  // output should be 0x5A82
  f16Output = GFLIB_Cos (f16Angle, GFLIB_COS_DEFAULT_F16, Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x5A82
  f16Output = GFLIB_Cos (f16Angle);
}
```

4.54 Function GFLIB_Cos_FLT

This function implements polynomial approximation of cosine function.

4.54.1 Declaration

```
tFloat GFLIB_Cos_FLT(tFloat fltIn, const GFLIB_COS_T_FLT *const pParam);
```

4.54.2 Arguments

Table 4-70. GFLIB_Cos_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians between [- π, π].
const GFLIB_COS_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

4.54.3 Return

The function returns the cosine of the input argument as a single precision floating point number.

4.54.4 Description

The GFLIB_Cos_FLT function provides a computational method for the calculation of a standard trigonometric *cosine* function cos(x), using the floating point optimized minimax polynomial approximation of the *sine* function. The following two equations describe the chosen approach of calculating the *sine* function:

$$\sin(\text{fltIn}) = \text{fltIn} + \text{fltIn}^3 \bullet a_2 + \text{fltIn}^5 \bullet a_1 + \text{fltIn}^7 \bullet a_0$$

Equation **GFLIB_Cos_Eq1**

$$\cos(\text{fltIn}) = \sin(\text{fltIn} + \tfrac{\pi}{2})$$

Equation **GFLIB_Cos_Eq2**

where a $_0$ ... a $_2$ are coefficients of the approximation polynomial.

Equations GFLIB_Cos_Eq1 and GFLIB_Cos_Eq2 can be further rewritten into the following form:

$$\cos(\text{fltIn}) = (\text{fltIn} + \tfrac{\pi}{2}) + (\text{fltIn} + \tfrac{\pi}{2})^3 \bullet (a_2 + (\text{fltIn} + \tfrac{\pi}{2})^2 \bullet (a_1 + (\text{fltIn} + \tfrac{\pi}{2})^2 \bullet a_0))$$

Equation **GFLIB_Cos_Eq3**

The floating point optimized minimax approximation is chosen as sufficient in order to achieve the best ratio between calculation accuracy and execution speed. Optimized floating point minimax approximation of the *sine* function reaches the best precision in the [- π/2, π/2] range. To calculate the values in the interval [- π, - π/2) and (π/2, π] the following equation can be used:

$$\sin(\text{fltIn}) = \sin(\pi - \text{fltIn})$$

Equation **GFLIB_Cos_Eq4**

for interval (π/2, π]

$$\sin(\text{fltIn}) = -\sin(\pi + \text{fltIn})$$

Equation **GFLIB_Cos_Eq5**

for interval [- π, - π/2)

$$\cos(-\text{fltIn}) = \cos(\text{fltIn})$$

Equation **GFLIB_Cos_Eq6**

Considering GFLIB_Cos_Eq2 and GFLIB_Cos_Eq6, the previously defined interval [- π/2, π/2] is transformed to the interval [- π,0], and intervals (π/2, π] and [- π, - π/2) are transformed to the interval (0, π].

Equations GFLIB_Cos_Eq2, GFLIB_Cos_Eq4 and GFLIB_Cos_Eq5 are finally transformed to the following formula:

$$\cos\left(\text{fltIn}\right) = \begin{cases} \cos\left(\text{fltIn} + \frac{\pi}{2}\right) & \text{if} \quad -1 \le \text{fltIn} < 0 \\ \cos\left(\frac{\pi}{2} - \text{fltIn}\right) & \text{if} \quad 0 \le \text{fltIn} < 1 \end{cases}$$

Equation **GFLIB_Cos_Eq7**

The floating point optimized approximation coefficients used for GFLIB_Cos_FLT calculation are noted in Table 4-71.

Table 4-71. Approximation polynomial coefficients

a_0	a_1	a_2
-0.000184881402886	0.008311899801389	-0.166655540927577

Figure 4-27 depicts a floating point *cosine* function generated from Matlab and the approximated value of the *cosine* function obtained from GFLIB_Cos_FLT, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

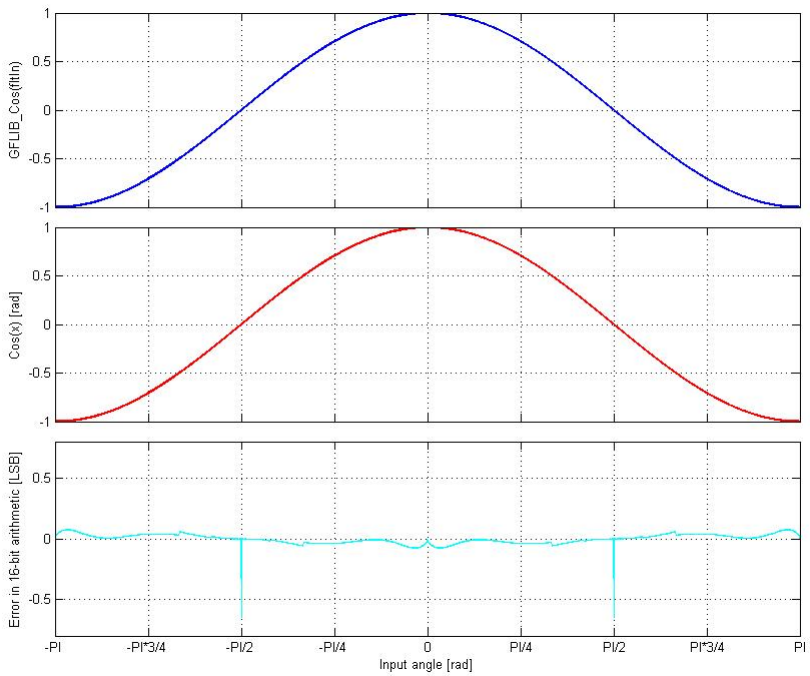

Figure 4-27. cos(x) vs. GFLIB_Cos(fltIn)

Note

The input angle (fltIn) is in single precision floating point format considering the input angle directly in radians. As during the computation the irrational value of π/2 is used for subtraction, the correction constant is used to increase the calculation precision in boundary intervals. This correction constant is equal to the difference of π/2 computed in double and in single precision. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Cos_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default

approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Cos(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_COS_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Cos(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_COS_DEFAULT_FLT approximation coefficients are used.

4.54.5 Re-entrancy

The function is re-entrant.

4.54.6 Code Example

```
#include "gflib.h"


tFloat fltAngle;
tFloat fltOutput;

void main(void)
{
  // input angle = 0.785398163 = pi/4
  fltAngle  = (tFloat)(0.785398163);

  // output should be 0.70710678
  fltOutput = GFLIB_Cos_FLT (fltAngle, GFLIB_COS_DEFAULT_FLT);

  // output should be 0.70710678
  fltOutput = GFLIB_Cos (fltAngle, GFLIB_COS_DEFAULT_FLT, Define FLT);

  // ##################################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##################################################################

  // output should be 0.70710678
  fltOutput = GFLIB_Cos (fltAngle);
}
```

4.55 Function GFLIB_Hyst_F32

This function implements the hysteresis functionality.

4.55.1 Declaration

```
tFrac32 GFLIB_Hyst_F32(tFrac32 f32In, GFLIB_HYST_T_F32 *const pParam);
```

4.55.2 Arguments

Table 4-72. GFLIB_Hyst_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input signal in the form of a 32-bit fixed point number, normalized between [-1, 1).
GFLIB_HYST_T_F32 *const	pParam	**input, output**	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain fixed point 32-bit values, normalized between [-1, 1).

4.55.3 Return

The function returns the value of the hysteresis output, which is equal to either f32OutValOn or f32OutValOff depending on the value of the input and the state of the function output in the previous calculation step. The output value is interpreted as a fixed point 32-bit number, normalized between [-1, 1).

4.55.4 Description

The GFLIB_Hyst function provides a computational method for the calculation of a hysteresis (relay) function. The function switches the output between the two predefined values stored in the f32OutValOn and f32OutValOff members of structure GFLIB_HYST_T_F32. When the value of the input is higher than the upper threshold f32HystOn, then the output value is equal to f32OutValOn. On the other hand, when the input value is lower than the lower threshold f32HystOff, then the output value is equal to f32OutValOff. When the input value is between these two threshold values then the output retains its value (the previous state).

$$f32OutState(k) = \begin{cases} f32OutValOn & \text{if} \quad f32In \geq f32HystOn \\ f32OutValOff & \text{if} \quad f32In \leq f32HystOff \\ f32OUtState(k-1) & \text{otherwise} \end{cases}$$

Equation **GFLIB_Hyst_Eq1**

A graphical description of GFLIB_Hyst functionality is shown in Figure 4-28.

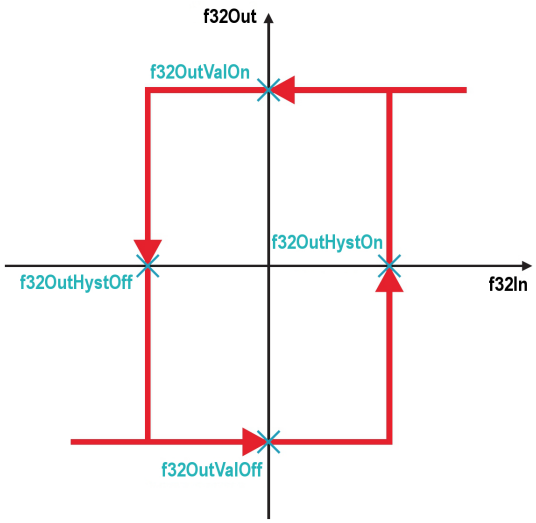

Figure 4-28. Hysteresis function

CAUTION

For correct functionality, the threshold f32HystOn value must be greater than the f32HystOff value.

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_HYST_DEFAULT_F32 macro.

4.55.5 Re-entrancy

The function is re-entrant.

4.55.6 Code Example

```
#include "gflib.h"
```

```
tFrac32 f32In;
tFrac32 f32Out;
GFLIB_HYST_T_F32 f32trMyHyst = GFLIB_HYST_DEFAULT_F32;

void main(void)
{
    // Setting parameters for hysteresis
    f32trMyHyst.f32HystOn = FRAC32 (0.1289);
    f32trMyHyst.f32HystOff = FRAC32 (-0.3634);
    f32trMyHyst.f32OutValOn = FRAC32 (0.589);
    f32trMyHyst.f32OutValOff = FRAC32 (-0.123);
    f32trMyHyst.f32OutState = FRAC32 (-0.3333);
    // input value = -0.41115
    f32In = FRAC32 (-0.41115);

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32Out = GFLIB_Hyst_F32 (f32In,&f32trMyHyst);

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32trMyHyst.f32OutState = FRAC32 (0);
    f32Out = GFLIB_Hyst (f32In, &f32trMyHyst, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x8FBE76C8 ~ FRAC32(-0.123)
    f32trMyHyst.f32OutState = FRAC32 (0);
    f32Out = GFLIB_Hyst (f32In, &f32trMyHyst);
}
```

4.56 Function GFLIB_Hyst_F16

This function implements the hysteresis functionality.

4.56.1 Declaration

```
tFrac16 GFLIB_Hyst_F16(tFrac16 f16In, GFLIB_HYST_T_F16 *const pParam);
```

4.56.2 Arguments

Table 4-73. GFLIB_Hyst_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input signal in the form of a 16-bit fixed point number, normalized between [-1, 1).
GFLIB_HYST_T_F16 *const	pParam	**input, output**	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain fixed point 16-bit values, normalized between [-1, 1).

4.56.3 Return

The function returns the value of the hysteresis output, which is equal to either f16OutValOn or f16OutValOff depending on the value of the input and the state of the function output in the previous calculation step. The output value is interpreted as a fixed point 16-bit number, normalized between [-1, 1).

4.56.4 Description

The GFLIB_Hyst function provides a computational method for the calculation of a hysteresis (relay) function. The function switches the output between the two predefined values stored in the f16OutValOn and f16OutValOff members of structure GFLIB_HYST_T_F16. When the value of the input is higher than the upper threshold f16HystOn, then the output value is equal to f16OutValOn. On the other hand, when the input value is lower than the lower threshold f16HystOff, then the output value is equal to f16OutValOff. When the input value is between these two threshold values then the output retains its value (the previous state).

$$f16OutState(k) = \begin{cases} f16OutValOn & \text{if} \quad f16In \geq f16HystOn \\ f16OutValOff & \text{if} \quad f16In \leq f16HystOff \\ f16OUtState(k-1) & \text{otherwise} \end{cases}$$

Equation **GFLIB_Hyst_Eq1**

A graphical description of GFLIB_Hyst functionality is shown in Figure 4-29.

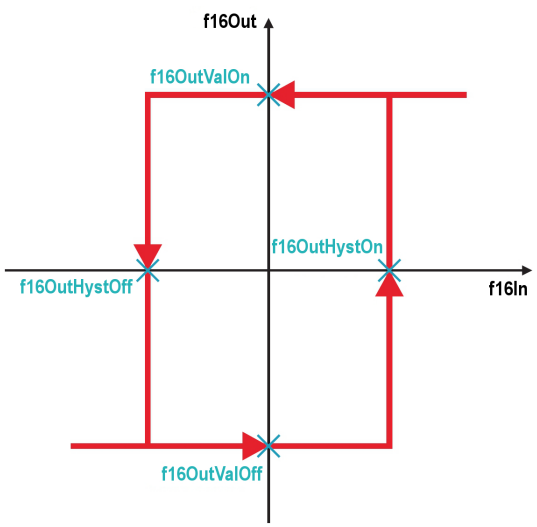

Figure 4-29. Hysteresis function

CAUTION

For correct functionality, the threshold f16HystOn value must be greater than the f16HystOff value.

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_HYST_DEFAULT_F16 macro.

4.56.5 Re-entrancy

The function is re-entrant.

4.56.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_HYST_T_F16 f16trMyHyst = GFLIB_HYST_DEFAULT_F16;

void main(void)
{
    // Setting parameters for hysteresis
    f16trMyHyst.f16HystOn = FRAC16 (0.1289);
    f16trMyHyst.f16HystOff = FRAC16 (-0.3634);
    f16trMyHyst.f16OutValOn = FRAC16 (0.589);
    f16trMyHyst.f16OutValOff = FRAC16 (-0.123);
    f16trMyHyst.f16OutState = FRAC16 (-0.3333);
    // input value = -0.41115
    f16In = FRAC16 (-0.41115);

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16Out = GFLIB_Hyst_F16 (f16In,&f16trMyHyst);

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16trMyHyst.f16OutState = FRAC16 (0);
    f16Out = GFLIB_Hyst (f16In, &f16trMyHyst, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x8FBE ~ FRAC16(-0.123)
    f16trMyHyst.f16OutState = FRAC16 (0);
    f16Out = GFLIB_Hyst (f16In, &f16trMyHyst);
}
```

4.57 Function GFLIB_Hyst_FLT

This function implements the hysteresis functionality.

4.57.1 Declaration

```
tFloat GFLIB_Hyst_FLT(tFloat fltIn, GFLIB_HYST_T_FLT *const pParam);
```

4.57.2 Arguments

Table 4-74. GFLIB_Hyst_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value, in single precision floating point data format.
GFLIB_HYST_T_FLT *const	pParam	**input, output**	Pointer to the structure with parameters and states of the hysteresis function. Arguments of the structure contain a single precision floating point values.

4.57.3 Return

The function returns the value of the hysteresis output, which is equal to either fltOutValOn or fltOutValOff depending on the value of the input and the state of the function output in the previous calculation step. The output value is in single precision floating point format.

4.57.4 Description

The GFLIB_Hyst function provides a computational method for the calculation of a hysteresis (relay) function. The function switches the output between the two predefined values stored in the fltOutValOn and fltOutValOff members of structure GFLIB_HYST_T_FLT. When the value of the input is higher than the upper threshold fltHystOn, then the output value is equal to fltOutValOn. On the other hand, when the input value is lower than the lower threshold fltHystOff, then the output value is equal to fltOutValOff. When the input value is between these two threshold values then the output retains its value (the previous state).

$$\mathrm{fltOutState}\left(k\right)=\begin{cases}\mathrm{fltOutValOn} & \text{if} \quad \mathrm{fltIn}\geq\mathrm{fltHystOn}\\ \mathrm{fltOutValOff} & \text{if} \quad \mathrm{fltIn}\leq\mathrm{fltHystOff}\\ \mathrm{fltOUtState}\left(k-1\right) & \text{otherwise}\end{cases}$$

Equation **GFLIB_Hyst_Eq1**

A graphical description of GFLIB_Hyst functionality is shown in Figure 4-30.

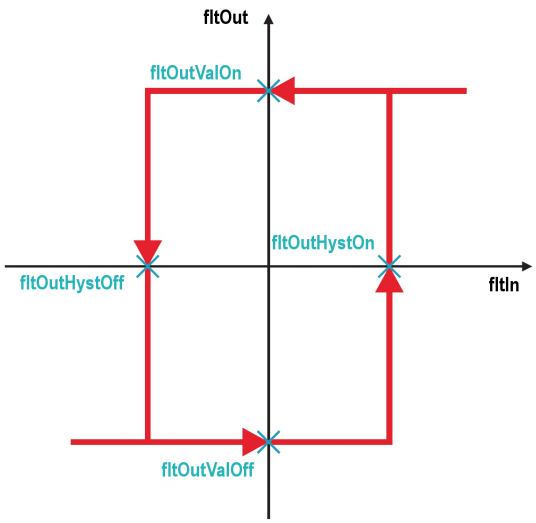

Figure 4-30. Hysteresis function

CAUTION

For correct functionality, the threshold fltHystOn value must be greater than the fltHystOff value.

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_HYST_DEFAULT_FLT macro.

4.57.5 Re-entrancy

The function is re-entrant.

4.57.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_HYST_T_FLT flttrMyHyst = GFLIB_HYST_DEFAULT_FLT;

void main(void)
{
    // Setting parameters for hysteresis
    flttrMyHyst.fltHystOn = (tFloat)(0.1289);
```

```
    flttrMyHyst.fltHystOff = (tFloat)(-0.3634);
    flttrMyHyst.fltOutValOn = (tFloat)(0.589);
    flttrMyHyst.fltOutValOff = (tFloat)(-0.123);
    flttrMyHyst.fltOutState = (tFloat)(-0.3333);
    // input value = -0.41115
    fltIn = (tFloat)(-0.41115);

    // output should be -0.123
    fltOut = GFLIB_Hyst_FLT (fltIn,&flttrMyHyst);

    // output should be -0.123
    flttrMyHyst.fltOutState = (tFloat)(0);
    fltOut = GFLIB_Hyst (fltIn, &flttrMyHyst, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be -0.123
    flttrMyHyst.fltOutState = (tFloat)(0);
    fltOut = GFLIB_Hyst (fltIn, &flttrMyHyst);
}
```

4.58 Function GFLIB_IntegratorTR_F32

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

4.58.1 Declaration

```
tFrac32 GFLIB_IntegratorTR_F32(tFrac32 f32In, GFLIB_INTEGRATOR_TR_T_F32 *const pParam);
```

4.58.2 Arguments

Table 4-75. GFLIB_IntegratorTR_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_F32 *const	pParam	**input, output**	Pointer to the integrator parameters structure.

4.58.3 Return

The function returns a 32-bit value in format Q1.31, which represents the actual integrated value of the input signal.

4.58.4 Description

The function GFLIB_IntegratorTR_F32 implements a discrete integrator using trapezoidal (Bilinear) transformation.

The continuous time domain representation of the integrator is defined as:

$$u(t) = \int_0^t e(t)\mathrm{dt}$$

Equation **GFLIB_IntegratorTR_Eq1**

The transfer function for this integrator, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation **GFLIB_IntegratorTR_Eq2**

Transforming equation GFLIB_IntegratorTR_Eq2 into a digital time domain using Bilinear transformation, leads to the following transfer function:

$$\mathbb{Z}\ \{\mathrm{H(s)}\} = \frac{U(z)}{E(z)} \frac{T_s + T_s z^{-1}}{2 - 2z^{-1}}$$

Equation **GFLIB_IntegratorTR_Eq3**

where T_s is the sampling period of the system. The discrete implementation of the digital transfer function GFLIB_IntegratorTR_Eq3 is as follows:

$$u(k) = u(k-1) + e(k)\frac{T_s}{2} + e(k-1)\frac{T_s}{2}$$

Equation **GFLIB_IntegratorTR_Eq4**

Considering fractional maths implementation, the integrator input and output maximal values (scales) must be known. Then the discrete implementation is given as follows:

$$u(k) = u(k-1) + e(k) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}} + e(k-1) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}}$$

Equation **GFLIB_IntegratorTR_Eq5**

where E_{MAX} is the input scale and U_{MAX} is the output scale. Then integrator constant C1 is defined as:

$$C1_f = \frac{T_s}{2} \bullet \frac{E_{MAX}}{U_{MAX}}$$

Equation **GFLIB_IntegratorTR_Eq6**

In order to implement the discrete form integrator as in GFLIB_IntegratorTR_Eq5 on a fixed point platform, the value of $C1_f$ coefficient must reside in a the fractional range [-1,1). Therefore, scaling must be introduced as follows:

$$f32C1 = C1_f \bullet 2^{-u16NShift}$$

Equation **GFLIB_IntegratorTR_Eq7**

The introduced scaling is chosen such that coefficient f32C1 fits into fractional range [-1,1). To simplify the implementation, this scaling is chosen to be a power of 2, so the final scaling is a simple shift operation using the u16NShift variable. Hence, the shift is calculated as:

$$u16NShift = ceil\left(\frac{\log(C1_f)}{\log(2)}\right)$$

Equation **GFLIB_IntegratorTR_Eq8**

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_INTEGRATOR_TR_DEFAULT_F32 macro.

4.58.5 Re-entrancy

The function is re-entrant.

4.58.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F32 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F32;
```

```
void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f32C1         = FRAC32 (100e-4/2);
    trMyIntegrator.u16NShift     = 0;

    // input value = 0.5
    f32In  = FRAC32 (0.5);

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR_F32 (f32In, &trMyIntegrator);

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR (f32In, &trMyIntegrator, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x0051EB85
    f32Out = GFLIB_IntegratorTR (f32In, &trMyIntegrator);
}
```

4.59 Function GFLIB_IntegratorTR_F16

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

4.59.1 Declaration

```
tFrac16 GFLIB_IntegratorTR_F16(tFrac16 f16In, GFLIB_INTEGRATOR_TR_T_F16 *const pParam);
```

4.59.2 Arguments

Table 4-76. GFLIB_IntegratorTR_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_F16 *const	pParam	**input, output**	Pointer to the integrator parameters structure.

4.59.3 Return

The function returns a 16-bit value in format Q1.15, which represents the actual integrated value of the input signal.

4.59.4 Description

The function GFLIB_IntegratorTR_F16 implements a discrete integrator using trapezoidal (Bilinear) transformation.

The continuous time domain representation of the integrator is defined as:

$$u(t) = \int_0^t e(t)\mathrm{dt}$$

Equation **GFLIB_IntegratorTR_Eq1**

The transfer function for this integrator, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation **GFLIB_IntegratorTR_Eq2**

Transforming equation GFLIB_IntegratorTR_Eq2 into a digital time domain using Bilinear transformation, leads to the following transfer function:

$$\mathbb{Z}\ \{\mathrm{H(s)}\} = \frac{U(z)}{E(z)} \frac{T_s + T_s z^{-1}}{2 - 2z^{-1}}$$

Equation **GFLIB_IntegratorTR_Eq3**

where T_s is the sampling period of the system. The discrete implementation of the digital transfer function GFLIB_IntegratorTR_Eq3 is as follows:

$$u(k) = u(k-1) + e(k)\frac{T_s}{2} + e(k-1)\frac{T_s}{2}$$

Equation **GFLIB_IntegratorTR_Eq4**

Considering fractional maths implementation, the integrator input and output maximal values (scales) must be known. Then the discrete implementation is given as follows:

$$u(k) = u(k-1) + e(k) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}} + e(k-1) \bullet \frac{T_s}{2} \bullet \frac{E_{\mathrm{MAX}}}{U_{\mathrm{MAX}}}$$

Equation **GFLIB_IntegratorTR_Eq5**

where E $_{MAX}$ is the input scale and U $_{MAX}$ is the output scale. Then integrator constant C1 is defined as:

$$C1_f = \frac{T_s}{2} \bullet \frac{E_{MAX}}{U_{MAX}}$$

Equation **GFLIB_IntegratorTR_Eq6**

In order to implement the discrete form integrator as in GFLIB_IntegratorTR_Eq5 on a fixed point platform, the value of C1 $_f$ coefficient must reside in a the fractional range [-1,1). Therefore, scaling must be introduced as follows:

$$f16C1 = C1_f \bullet 2^{-u16NShift}$$

Equation **GFLIB_IntegratorTR_Eq7**

The introduced scaling is chosen such that coefficient f16C1 fits into fractional range [-1,1). To simplify the implementation, this scaling is chosen to be a power of 2, so the final scaling is a simple shift operation using the u16NShift variable. Hence, the shift is calculated as:

$$u16NShift = ceil\left(\frac{\log(C1_f)}{\log(2)}\right)$$

Equation **GFLIB_IntegratorTR_Eq8**

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_INTEGRATOR_TR_DEFAULT_F16 macro.

4.59.5 Re-entrancy

The function is re-entrant.

4.59.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_F16 trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_F16;
```

```
void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4, E_MAX=U_MAX=1
    trMyIntegrator.f16C1         = FRAC16 (100e-4/2);
    trMyIntegrator.u16NShift     = 0;

    // input value = 0.5
    f16In  = FRAC16 (0.5);

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR_F16 (f16In, &trMyIntegrator);

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR (f16In, &trMyIntegrator, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x0051
    f16Out = GFLIB_IntegratorTR (f16In, &trMyIntegrator);
}
```

4.60 Function GFLIB_IntegratorTR_FLT

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

4.60.1 Declaration

```
tFloat GFLIB_IntegratorTR_FLT(tFloat fltIn, GFLIB_INTEGRATOR_TR_T_FLT *const pParam);
```

4.60.2 Arguments

Table 4-77. GFLIB_IntegratorTR_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument to be integrated.
GFLIB_INTEGRATOR_TR_T_FLT *const	pParam	**input, output**	Pointer to the integrator parameters structure.

4.60.3 Return

The function returns a single precision floating point value, which represents the actual integrated value of the input signal.

4.60.4 Description

The function GFLIB_IntegratorTR_FLT implements a discrete integrator using trapezoidal (Bilinear) transformation.

The continuous time domain representation of the integrator is defined as:

$$u(t) = \int_0^t e(t)\mathrm{dt}$$

Equation **GFLIB_IntegratorTR_Eq1**

The transfer function for this integrator, in a continuous time domain, is described using the Laplace transformation as follows:

$$H(s) = \frac{U(s)}{E(s)} = \frac{1}{s}$$

Equation **GFLIB_IntegratorTR_Eq2**

Transforming equation GFLIB_IntegratorTR_Eq2 into a digital time domain using the Bilinear transformation, leads to the following transfer function:

$$\mathbb{Z}\ \{\mathrm{H(s)}\} = \frac{U(z)}{E(z)}\frac{T_s + T_s z^{-1}}{2 - 2z^{-1}}$$

Equation **GFLIB_IntegratorTR_Eq3**

where T_s is the sampling period of the system. The discrete implementation of the digital transfer function GFLIB_IntegratorTR_Eq3 is as follows:

$$u(k) = u(k-1) + e(k)\frac{T_s}{2} + e(k-1)\frac{T_s}{2}$$

Equation **GFLIB_IntegratorTR_Eq4**

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_INTEGRATOR_TR_DEFAULT_FLT macro.

4.60.5 Re-entrancy

The function is re-entrant.

4.60.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

// Definition of one integrator instance
GFLIB_INTEGRATOR_TR_T_FLT trMyIntegrator = GFLIB_INTEGRATOR_TR_DEFAULT_FLT;

void main(void)
{
    // Setting parameters for integrator, Ts = 100e-4,
    trMyIntegrator.fltC1 = (tFloat)(100e-4/2);

    // input value = 0.5
    fltIn  = 0.5;

    // output should be 2.5e-3
    fltOut = GFLIB_IntegratorTR_FLT (fltIn, &trMyIntegrator);

    // output should be 2.5e-3
    fltOut = GFLIB_IntegratorTR (fltIn, &trMyIntegrator, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 2.5e-3
    fltOut = GFLIB_IntegratorTR (fltIn, &trMyIntegrator);
```

4.61 Function GFLIB_Limit_F32

This function tests whether the input value is within the upper and lower limits.

4.61.1 Declaration

```
tFrac32 GFLIB_Limit_F32(tFrac32 f32In, const GFLIB_LIMIT_T_F32 *const pParam);
```

4.61.2 Arguments

Table 4-78. GFLIB_Limit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GFLIB_LIMIT_T_F32 *const	pParam	**input**	Pointer to the limits structure.

4.61.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

4.61.4 Description

The GFLIB_Limit function tests whether the input value is within the upper and lower limits. If so, the input value will be returned. If the input value is above the upper limit, the upper limit will be returned. Similarly, if the input value is below the lower limit, the lower limit will be returned.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer pParam.

Note

The function assumes that the upper limit f32UpperLimit is greater than the lower limit f32LowerLimit. Otherwise, the function returns an undefined value.

4.61.5 Re-entrancy

The function is re-entrant.

4.61.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
```

```
GFLIB_LIMIT_T_F32 f32trMyLimit = GFLIB_LIMIT_DEFAULT_F32;

void main(void)
{
    // upper/lower limits
    f32trMyLimit.f32UpperLimit = FRAC32 (0.5);
    f32trMyLimit.f32LowerLimit = FRAC32 (-0.5);

    // input value = 0.75
    f32In = FRAC32 (0.75);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit_F32 (f32In, &f32trMyLimit);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit (f32In, &f32trMyLimit, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_Limit (f32In, &f32trMyLimit);
}
```

4.62 Function GFLIB_Limit_F16

This function tests whether the input value is within the upper and lower limits.

4.62.1 Declaration

```
tFrac16 GFLIB_Limit_F16(tFrac16 f16In, const GFLIB_LIMIT_T_F16 *const pParam);
```

4.62.2 Arguments

Table 4-79. GFLIB_Limit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GFLIB_LIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

4.62.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

4.62.4 Description

The GFLIB_Limit function tests whether the input value is within the upper and lower limits. If so, the input value will be returned. If the input value is above the upper limit, the upper limit will be returned. Similarly, if the input value is below the lower limit, the lower limit will be returned.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer pParam.

Note

The function assumes that the upper limit f16UpperLimit is greater than the lower limit f16LowerLimit. Otherwise, the function returns an undefined value.

4.62.5 Re-entrancy

The function is re-entrant.

4.62.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LIMIT_T_F16 f16trMyLimit = GFLIB_LIMIT_DEFAULT_F16;

void main(void)
{
    // upper/lower limits
    f16trMyLimit.f16UpperLimit = FRAC16 (0.5);
    f16trMyLimit.f16LowerLimit = FRAC16 (-0.5);

    // input value = 0.75
    f16In = FRAC16 (0.75);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit_F16 (f16In, &f16trMyLimit);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit (f16In, &f16trMyLimit, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_Limit (f16In, &f16trMyLimit);
}
```

4.63 Function GFLIB_Limit_FLT

This function tests whether the input value is within the upper and lower limits.

4.63.1 Declaration

```
tFloat GFLIB_Limit_FLT(tFloat fltIn, const GFLIB_LIMIT_T_FLT *const pParam);
```

4.63.2 Arguments

Table 4-80. GFLIB_Limit_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GFLIB_LIMIT_T_FLT *const	pParam	**input**	Pointer to the limits structure.

4.63.3 Return

The input value in case the input value is below the limits, or the upper or lower limit if the input value is above these limits.

4.63.4 Description

The GFLIB_Limit function tests whether the input value is within the upper and lower limits. If so, the input value will be returned. If the input value is above the upper limit, the upper limit will be returned. Similarly, if the input value is below the lower limit, the lower limit will be returned.

The upper and lower limits can be found in the limits structure, supplied to the function as a pointer pParam.

Note

The function assumes that the upper limit fltUpperLimit is greater than the lower limit fltLowerLimit. Otherwise, the function returns an undefined value.

4.63.5 Re-entrancy

The function is re-entrant.

4.63.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_LIMIT_T_FLT flttrMyLimit = GFLIB_LIMIT_DEFAULT_FLT;

void main(void)
{
    // upper/lower limits
    flttrMyLimit.fltUpperLimit = 0.5;
    flttrMyLimit.fltLowerLimit = -0.5;

    // input value = 0.75
    fltIn = 0.75;

    // output should be 0.5
    fltOut = GFLIB_Limit_FLT (fltIn, &flttrMyLimit);

    // output should be 0.5
    fltOut = GFLIB_Limit (fltIn, &flttrMyLimit, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.5
    fltOut = GFLIB_Limit (fltIn, &flttrMyLimit);
}
```

4.64 Function GFLIB_LowerLimit_F32

This function tests whether the input value is above the lower limit.

4.64.1 Declaration

```
tFrac32 GFLIB_LowerLimit_F32(tFrac32 f32In, const GFLIB_LOWERLIMIT_T_F32 *const pParam);
```

4.64.2 Arguments

Table 4-81. GFLIB_LowerLimit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GFLIB_LOWERLIMIT_T_F32 *const	pParam	**input**	Pointer to the limits structure.

4.64.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

4.64.4 Description

The function tests whether the input value is above the lower limit. If so, the input value will be returned. Otherwise, if the input value is below the lower limit, the lower limit will be returned.

The lower limit f32LowerLimit can be found in the limits structure, supplied to the function as a pointer pParam.

4.64.5 Re-entrancy

The function is re-entrant.

4.64.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LOWERLIMIT_T_F32 f32trMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_F32;

void main(void)
{
    // lower limit
    f32trMyLowerLimit.f32LowerLimit = FRAC32 (0.5);

    // input value = 0.75
    f32In = FRAC32 (0.75);
```

```
    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit_F32 (f32In,&f32trMyLowerLimit);

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit (f32In,&f32trMyLowerLimit,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = GFLIB_LowerLimit (f32In,&f32trMyLowerLimit);
}
```

4.65 Function GFLIB_LowerLimit_F16

This function tests whether the input value is above the lower limit.

4.65.1 Declaration

```
tFrac16 GFLIB_LowerLimit_F16(tFrac16 f16In, const GFLIB_LOWERLIMIT_T_F16 *const pParam);
```

4.65.2 Arguments

Table 4-82. GFLIB_LowerLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GFLIB_LOWERLIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

4.65.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

4.65.4 Description

The function tests whether the input value is above the lower limit. If so, the input value will be returned. Otherwise, if the input value is below the lower limit, the lower limit will be returned.

The lower limit f32LowerLimit can be found in the limits structure, supplied to the function as a pointer pParam.

4.65.5 Re-entrancy

The function is re-entrant.

4.65.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LOWERLIMIT_T_F16 f16trMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_F16;

void main(void)
{
    // lower limit
    f16trMyLowerLimit.f16LowerLimit = FRAC16 (0.5);

    // input value = 0.75
    f16In = FRAC16 (0.75);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit_F16 (f16In,&f16trMyLowerLimit);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit (f16In,&f16trMyLowerLimit,Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = GFLIB_LowerLimit (f16In,&f16trMyLowerLimit);
}
```

4.66 Function GFLIB_LowerLimit_FLT

This function tests whether the input value is above the lower limit.

4.66.1 Declaration

```
tFloat GFLIB_LowerLimit_FLT(tFloat fltIn, const GFLIB_LOWERLIMIT_T_FLT *const pParam);
```

4.66.2 Arguments

Table 4-83. GFLIB_LowerLimit_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GFLIB_LOWERLIMIT_T_FLT *const	pParam	**input**	Pointer to the limits structure.

4.66.3 Return

The input value in case the input value is above the limit, or the lower limit if the input value is below the limit.

4.66.4 Description

The function tests whether the input value is above the lower limit. If so, the input value will be returned. Otherwise, if the input value is below the lower limit, the lower limit will be returned.

The lower limit f32LowerLimit can be found in the limits structure, supplied to the function as a pointer pParam.

4.66.5 Re-entrancy

The function is re-entrant.

4.66.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
```

```
GFLIB_LOWERLIMIT_T_FLT flttrMyLowerLimit = GFLIB_LOWERLIMIT_DEFAULT_FLT;

void main(void)
{
    // lower limit
    flttrMyLowerLimit.fltLowerLimit = 0.5;

    // input value = 0.75
    fltIn = 0.75;

    // output should be 0.75
    fltOut = GFLIB_LowerLimit_FLT (fltIn,&flttrMyLowerLimit);

    // output should be 0.75
    fltOut = GFLIB_LowerLimit (fltIn,&flttrMyLowerLimit,Define FLT);

    // ################################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ################################################################

    // output should be 0.75
    fltOut = GFLIB_LowerLimit (fltIn,&flttrMyLowerLimit);
}
```

4.67 Function GFLIB_Lut1D_F32

This function implements the one-dimensional look-up table.

4.67.1 Declaration

```
tFrac32 GFLIB_Lut1D_F32(tFrac32 f32In, const GFLIB_LUT1D_T_F32 *const pParam);
```

4.67.2 Arguments

Table 4-84. GFLIB_Lut1D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_F32 *const	pParam	**input**	Pointer to the parameters structure with parameters of the look-up table function.

4.67.3 Return

The interpolated value from the look-up table with 16-bit accuracy.

4.67.4 Description

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}\left(x - x_1\right)$$

Equation **GFLIB_Lut1D_Eq1**

where:

- y is the interpolated value
- y_1 and y_2 are the ordinate values at, respectively, the beginning and the end of the interpolating interval
- x_1 and x_2 are the abscissa values at, respectively, the beginning and the end of the interpolating interval
- the x is the input value provided to the function in the f32In argument

The interpolating intervals are defined in the table provided by the pf32Table member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index, while the interpolating index zero is the table element pointed to by the pf32Table parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, let's consider the following interpolating table:

Table 4-85. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1*(2^{-1})$
pf32Table 0.0	0	$0*(2^{-1})$
0.25	1	$1*(2^{-1})$
0.5	N/A	$2*(2^{-1})$

The Table 4-85 contains 4 interpolating points (note four rows). The interpolating interval length in this example is equal to 2^{-1} . The pf32Table parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column.

It should be noted that the pf32Table pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is 32 bits long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the linear interpolation will be performed:

$$I = x >> s_{\text{Interval}}$$

Equation **GFLIB_Lut1D_Eq2**

where I is the interval index and the s_{Interval} the shift amount provided in the parameters structure as the member s32ShamIntvl. The operator >> represents the binary arithmetic right shift.

2. Compute the abscissa offset within an interpolating interval:

$$\Delta x = x << s_{\text{Offset}} \,\&\, 0x7fffffff$$

Equation **GFLIB_Lut1D_Eq3**

where $\Delta\{x\}$ is the abscissa offset within an interval and the s_{Offset} is the shift amount provided in the parameters structure. The operators << and & represent, respectively, the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents the extraction of some least significant bits of the x with the sign bit cleared.

3.

$$y_1 = \left(32-\text{bit data at address pTable}\right) + 4 \bullet I$$

$$y_2 = \left(32-\text{bit data at address pTable}\right) + 4 \bullet \left(I+1\right)$$

$$y = y_1 + \left(y_2 - y_1\right) \bullet \Delta x$$

Equation **GFLIB_Lut1D_Eq4**

where y, y_1 and y_2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The pTable is the address provided in the parameters structure pParam->f32Table. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

It should be noted that the computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->s32ShamOffset, pParam->s32ShamIntvl). The address of the table with the data, the pTable, shall be defined by the parameter structure member pParam->pf32Table.

The shift amounts, the $s_{Interval}$ and s_{Offset}, can be computed with the following formulas:

$$s_{Interval} = 31 - |n|$$

$$s_{Offset} = |n|$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 32-bit word, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assuming that all interpolating interval lengths equal 2^{-n}.

It should be noted that the input abscissa value can be positive or negative. If it is, positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure (pParam->pf32Table). However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pf32Table pointer.

Note

The function performs a linear interpolation.

CAUTION

The function does not check whether the input abscissa value is within the range allowed by the interpolating data table pParam->pf32Table. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data.

4.67.5 Re-entrancy

The function is re-entrant.

4.67.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_LUT1D_T_F32 trf32MyLut1D = GFLIB_LUT1D_DEFAULT_F32;
tFrac32 pf32Table1D[9] = {FRAC32 (-1),FRAC32 (0),FRAC32 (-0.2),FRAC32
(0),FRAC32 (0.2),
                          FRAC32 (0),FRAC32 (0.8),FRAC32 (1),FRAC32 (1)};

void main(void)
{
    // setting parameters for Lut1D function
    trf32MyLut1D.pf32Table = &(pf32Table1D[4]);
    trf32MyLut1D.s32ShamOffset = 2;
    trf32MyLut1D.s32ShamIntvl = 31 - 2;
    // input vector = 0.5
    f32In = FRAC32 (0.5);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D_F32 (f32In,&trf32MyLut1D);

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D,Define F32);

    // ##################################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##################################################################

    // output should be 0x66666666 ~ FRAC32(0.8)
    f32Out = GFLIB_Lut1D (f32In,&trf32MyLut1D);
}
```

4.68 Function GFLIB_Lut1D_F16

This function implements the one-dimensional look-up table.

4.68.1 Declaration

```
tFrac16 GFLIB_Lut1D_F16(tFrac16 f16In, const GFLIB_LUT1D_T_F16 *const pParam);
```

4.68.2 Arguments

Table 4-86. GFLIB_Lut1D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	The abscissa for which 1D interpolation is performed.

Table continues on the next page...

Table 4-86. GFLIB_Lut1D_F16 arguments (continued)

Type	Name	Direction	Description
const GFLIB_LUT1D_T_F16 *const	pParam	**input**	Pointer to the parameters structure with parameters of the look-up table function.

4.68.3 Return

The interpolated value from the look-up table.

4.68.4 Description

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}\left(x - x_1\right)$$

Equation **GFLIB_Lut1D_Eq1**

where:

- y is the interpolated value
- y_1 and y_2 are the ordinate values at, respectively, the beginning and the end of the interpolating interval
- x_1 and x_2 are the abscissa values at, respectively, the beginning and the end of the interpolating interval
- the x is the input value provided to the function in the f16In argument

The interpolating intervals are defined in the table provided by the pf16Table member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index, while the interpolating index zero is the table element pointed to by the pf16Table parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, let's consider the following interpolating table:

Table 4-87. GFLIB_Lut1D example table

ordinate (y)	interpolating index	abscissa (x)
-0.5	-1	$-1*(2^{-1})$
pf16Table 0.0	0	$0*(2^{-1})$
0.25	1	$1*(2^{-1})$

Table continues on the next page...

Table 4-87. GFLIB_Lut1D example table (continued)

0.5	N/A	$2*(2^{-1})$

The Table 4-87 contains 4 interpolating points (note four rows). The interpolating interval length in this example is equal to 2^{-1}. The pf16Table parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column.

It should be noted that the pf16Table pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- the values of the interpolated function are 16 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 13
- the provided abscissa for interpolation is 16 bits long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the linear interpolation will be performed:

$$I = x >> s_{\text{Interval}}$$

Equation **GFLIB_Lut1D_Eq2**

where I is the interval index and the s_{Interval} the shift amount provided in the parameters structure as the member s16ShamIntvl. The operator >> represents the binary arithmetic right shift.

2. Compute the abscissa offset within an interpolating interval:

$$\Delta x = x << s_{\text{Offset}} \,\&\, 0x7fff$$

Equation **GFLIB_Lut1D_Eq3**

where $\Delta\{x\}$ is the abscissa offset within an interval and the s_{Offset} is the shift amount provided in the parameters structure. The operators << and & represent, respectively, the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents the extraction of some least significant bits of the x with the sign bit cleared.

3.

$$y_1 = \left(16-\text{bit data at address pTable}\right) + 4 \bullet I$$
$$y_2 = \left(16-\text{bit data at address pTable}\right) + 4 \bullet \left(I+1\right)$$
$$y = y_1 + \left(y_2 - y_1\right) \bullet \Delta x$$

Equation **GFLIB_Lut1D_Eq4**

where y, y_1 and y_2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The pTable is the address provided in the parameters structure pParam->f16Table. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

It should be noted that the computations are performed with a 16-bit accuracy. In particular, the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->s16ShamOffset, pParam->s16ShamIntvl). The address of the table with the data, the pTable, shall be defined by the parameter structure member pParam->pf16Table.

The shift amounts, the $s_{Interval}$ and s_{Offset}, can be computed with the following formulas:

$$s_{\text{Interval}} = 15 - |n|$$
$$s_{\text{Offset}} = |n|$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -15.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 16-bit halfword, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assuming that all interpolating interval lengths equal 2^{-n}.

It should be noted that the input abscissa value can be positive or negative. If it is, positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure (pParam->pf16Table). However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pf16Table pointer.

Note

The function performs a linear interpolation.

CAUTION

The function does not check whether the input abscissa value is within the range allowed by the interpolating data table pParam->pf16Table. If the computed interval index points to data outside the provided data table, then the interpolation will be computed with invalid data.

4.68.5 Re-entrancy

The function is re-entrant.

4.68.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_LUT1D_T_F16 trf16MyLut1D = GFLIB_LUT1D_DEFAULT_F16;
tFrac16 pf16Table1D[9] = {FRAC16 (-1),FRAC16 (0),FRAC16 (-0.2),FRAC16
(0),FRAC16 (0.2),
                          FRAC16 (0),FRAC16 (0.8),FRAC16 (1),FRAC16 (1)};

void main(void)
{
    // setting parameters for Lut1D function
    trf16MyLut1D.pf16Table = &(pf16Table1D[4]);
    trf16MyLut1D.s16ShamOffset = 2;
    trf16MyLut1D.s16ShamIntvl = 15 - 2;
    // input vector = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D_F16 (f16In,&trf16MyLut1D);

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D,Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x6666 ~ FRAC16(0.8)
    f16Out = GFLIB_Lut1D (f16In,&trf16MyLut1D);
}
```

4.69 Function GFLIB_Lut1D_FLT

This function implements the one-dimensional look-up table.

4.69.1 Declaration

```
tFloat GFLIB_Lut1D_FLT(tFloat fltIn, const GFLIB_LUT1D_T_FLT *const pParam);
```

4.69.2 Arguments

Table 4-88. GFLIB_Lut1D_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	The abscissa for which 1D interpolation is performed.
const GFLIB_LUT1D_T_FLT *const	pParam	**input**	Pointer to the parameters structure with parameters of the look-up table function.

4.69.3 Return

The interpolated value from the look-up table.

4.69.4 Description

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}\left(x - x_1\right)$$

Equation **GFLIB_Lut1D_Eq1**

where:

- y is the interpolated value
- y_1 and y_2 are the function values at the beginning and the end of the researched interpolation interval
- x_1 and x_2 are the values on the x-axis which determine the beginning and the end of the interpolation interval
- the x is the input value provided to the function in the fltIn argument

The interpolating intervals are defined in the table provided by the pfltTable member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, a table contains 4 interpolating points. The interpolating interval length in this example is equal to $2^{-1}=0.5$.

It should be noted that the pfltTable pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions, as provided below:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the abscissa provided for interpolation is 32 bits long

The algorithm performs the following steps:

1. Count the equidistant interpolating interval, in which the linear interpolation will be performed:

$$fSegments = (tFloat)(1 << s32ShamOffset + 1))$$

Equation **GFLIB_Lut1D_Eq2**

where fSegments is the interval length and the pParam->s32ShamOffset is the shift amount provided in the parameters structure as the member s32ShamOffset. Operator << represents the binary arithmetic left shift.

2. $$s32Intvl = (tS32)((fsegments) \bullet (fltIn))$$

Equation **GFLIB_Lut1D_Eq3**

where s32Intvl is the position of the first ordinate value at the beginning of the interpolating interval, the fSegments is the equidistant interpolating interval and the fltIn is the input value provided to the function as an argument. The casting to tS32 data type is a final operation of the equation which cuts the fractional part of the value and then indicates the position of the first ordinate value in the table.

3.

$$y_1 = \left(32-\text{bit data at address pTable}\right) + s32\text{Intvl}$$

$$y_2 = \left(32-\text{bit data at address pTable}\right) + \left(s32\text{Intvl} + 1\right)$$

$$\Delta x = \left(\text{tFloat}\right)\left(x \bullet 2^n\right) - \left(\text{tU32}\right)\left(x \bullet 2^n\right)$$

$$y = y_1 + \left(y_2 - y_1\right) \bullet \Delta x$$

Equation **GFLIB_Lut1D_Eq4**

where y, y1 and y2 are, respectively, the interpolated value, the ordinate at the start of the interpolating interval, the ordinate at the end of the interpolating interval. The pTable is the address provided in the parameters structure pParam->fltTable and the x is the input value provided to the function as an fltIn argument. It should be noted that due to assumption of equidistant data points, division by the interval length is avoided.

The address of the table with the data, the pTable, shall be defined by the parameter structure member pParam->pfltTable.

The offset amount can be provided by the following formula:

$$s32\text{ShamOffset} = |n|$$

Equation **GFLIB_Lut1D_Eq5**

where n is the integer defining the length of the interpolating interval in the range of 1, 2, ... 29.

This simple way to calculate the interpolating abscissa offset is a consequence of assuming that all interpolating interval lengths equal 2^{-n}.

It should be noted that the input abscissa value can be positive or negative. If it is positive then the ordinate values are read as in the ordinary data array, that is, at or after the data pointer provided in the parameters structure pParam->pfltTable. However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pfltTable pointer.

Note

The function performs a linear interpolation.

CAUTION

The function does not check whether the input abscissa value is within the range allowed by the interpolating data table pParam->pfltTable. If the computed interval index points to

data outside the provided data table, then the interpolation will be computed with invalid data.

4.69.5 Re-entrancy

The function is re-entrant.

4.69.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_LUT1D_T_FLT trfltMyLut1D = GFLIB_LUT1D_DEFAULT_FLT;
tFloat pfltTable1D[9] = {-1,0,-0.2,0,0.2,0,0.8,1,1};

void main(void)
{
    // setting parameters for Lut1D function
    trfltMyLut1D.pfltTable = &(pfltTable1D[4]);
    trfltMyLut1D.s32ShamOffset = 2;
    // input vector = 0.5
    fltIn = 0.5;

    // output should be 0.8
    fltOut = GFLIB_Lut1D_FLT (fltIn,&trfltMyLut1D);

    // output should be 0.8
    fltOut = GFLIB_Lut1D (fltIn,&trfltMyLut1D,Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.8
    fltOut = GFLIB_Lut1D (fltIn,&trfltMyLut1D);
}
```

4.70 Function GFLIB_Lut2D_F32

This function implements the two-dimensional look-up table.

4.70.1 Declaration

```
tFrac32 GFLIB_Lut2D_F32(tFrac32 f32In1, tFrac32 f32In2, const GFLIB_LUT2D_T_F32 *const
pParam);
```

4.70.2 Arguments

Table 4-89. GFLIB_Lut2D_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In1	**input**	First input variable for which 2D interpolation is performed.
tFrac32	f32In2	**input**	Second input variable for which 2D interpolation is performed.
const GFLIB_LUT2D_T_F32 *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

4.70.3 Return

The interpolated value from the look-up table with 16-bit accuracy.

4.70.4 Description

The GFLIB_Lut2D_F32 function performs two dimensional linear interpolation over a 2D table of data.

The following interpolation formulas are used:

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq1**

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq2**

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{y_2 - y_1} \bullet (y - y_1)$$

Equation **GFLIB_Lut2D_Eq3**

where:

- f[x,y] is the interpolated value
- f[x,y $_1$] and f[x,y $_2$] are the ordinate values at, respectively, the beginning and the end of the final interpolating interval
- x $_1$, x $_2$, y $_1$ and y $_2$ are the area values, respectively, the interpolated area
- the x, y are the input values provided to the function in the f32In1 and f32In2 arguments

The interpolating intervals are defined in the table provided by the pf32Table member of the parameters structure. The table contains ordinate values consecutively over the whole interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index while the interpolating index zero is the table element pointed to by the pf32Table parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example let's consider the following interpolating table:

Table 4-90. GFLIB_Lut2D example table

ordinate (f[x,y])	interpolating index	abscissa (x)	abscissa (y)
-0.5	-1	-1*(2 $^{-1}$)	-1*(2 $^{-1}$)
pf32Table 0.0	0	0*(2 $^{-1}$)	0*(2 $^{-1}$)
0.5	1	1*(2 $^{-1}$)	1*(2 $^{-1}$)
1.0	N/A	2*(2 $^{-1}$)	2*(2 $^{-1}$)

The Table 4-90 contains 4 interpolating points (note four rows). Interpolating interval length in this example is equal to 2 $^{-1}$. The pf32Table parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column. It should be noticed that the pf32Table pointer does not have to point to the start of the memory area with ordinate values. Therefore the interpolating index can be positive or negative or, even, does not have to have zero in its range.

Special algorithm is used to make the computation efficient, however under some additional assumptions as provided below:

- the values of the interpolated function are 32-bit long
- the length of each interpolating interval is equal to 2 $^{-n}$, where n is an integer in the range of 1, 2, ... 29
- the provided abscissa for interpolation is 32-bit long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the bilinear interpolation will be performed in each axis:

$$I_x = x >> s_{\text{IntegralX}}$$

Equation **GFLIB_Lut2D_Eq4**

$$I_y = y >> s_{\text{IntegralY}}$$

Equation **GFLIB_Lut2D_Eq5**

where I_x and I_y is the interval index and the $s_{\text{IntervalX}}$ and $s_{\text{IntervalY}}$ are the shift amounts provided in the parameters structure as a member s32ShamIntvl1 or s32ShamIntvl2. The operator >> represents the binary arithmetic right shift.

2. Compute the abscissas offset for both axis within an interpolating intervals:

$$\Delta x = x << s_{\text{Offset1}} \,\&\, 0x7FFFFFFF$$

Equation **GFLIB_Lut2D_Eq6**

$$\Delta y = y << s_{\text{Offset2}} \,\&\, 0x7FFFFFFF$$

Equation **GFLIB_Lut2D_Eq7**

where $\Delta\{x\}$ and $\Delta\{y\}$ are the abscissas offset within an Xinterval and Yinterval. The s_{Offset1} and s_{Offset2} are the shift amounts provided in the parameters structure. The operators << and & represent the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents extraction of some least significant bits of the x and y with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation in X-axis by y_1 value between the ordinates x

$$f[x_1, y_1] = 32\text{-bit data at address pTable for } x1, y1$$

Equation **GFLIB_Lut2D_Eq8**

$$f[x_2, y_1] = 32\text{-bit data at address pTable for } x2, y1$$

Equation **GFLIB_Lut2D_Eq9**

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{\Delta x} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq10**

where f[x,y $_1$] is interpolated value, f[x $_1$,y $_1$] and f[x $_2$,y $_1$] are, the ordinate at the start of the interpolating interval and the ordinate at the end of the interpolating interval. The pTable} is the address provided in the parameters structure pParam->f32Table.

4. The same computation as shown above in X-axis by y $_2$ value. Interpolation formulas are the same as paragraph 3:

$$f[x_1, y_2] = 32 - \text{bit data at address pTable for } x1, y2$$

Equation **GFLIB_Lut2D_Eq11**

$$f[x_2, y_2] = 32 - \text{bit data at address pTable for } x2, y2$$

Equation **GFLIB_Lut2D_Eq12**

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{\Delta x} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq13**

5. Final linear interpolation in Y-axis way Interpolation formulas are the same as paragraph 3 or 4:

$$f[x, y_1] = \text{result coputed in paragraph 3}$$

Equation **GFLIB_Lut2D_Eq14**

$$f[x, y_2] = \text{result coputed in paragraph 4}$$

Equation **GFLIB_Lut2D_Eq15**

$$f[x, y] = f[x, y_1] + \frac{f[x,y_2] - f[x,y_1]}{\Delta y} \bullet (y - y_1)$$

Equation **GFLIB_Lut2D_Eq16**

It should be noted that due to assumption of the equidistant data points, the division by the interval length is avoided.

It should be noted that the computations are performed with the 16-bit accuracy. In particular the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->s32ShamOffset1, pParam->s32ShamIntvl1, pParam->s32ShamOffset2, andpParam->s32ShamIntvl2). The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pf32Table.

The shift amounts, the $s_{Interval1,2}$ and $s_{Offset1,2}$, can be computed with the following formulas:

$$s_{Interval1,2} = 31 - |n|$$
$$s_{Offset1,2} = |n|$$

Equation **GFLIB_Lut2D_Eq17**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 32-bit word, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assumption of all interpolating interval lengths equal 2^{-n}.

It should be noted that the input abscissa value can be positive or negative. If it is positive then the ordinate values are read as in the ordinary data array, that is at or after the data pointer provided in the parameters structure (pParam->pf32Table). However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pf32Table pointer.

Note

The function performs the bilinear interpolation with 16-bit accuracy.

CAUTION

The function does not check whether the input values are within a range allowed by the interpolating data table pParam->pf32Table. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data.

4.70.5 Re-entrancy

The function is re-entrant.

4.70.6 Code Example

```
#include "gflib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;
GFLIB_LUT2D_T_F32 tr32tMyLut2D = GFLIB_LUT2D_DEFAULT_F32;
tFrac32 pf32Table2D[81] = {FRAC32 (0.0), FRAC32 (0.01), FRAC32 (0.02), FRAC32
(0.03), FRAC32 (0.04), FRAC32 (0.05), FRAC32 (0.06), FRAC32 (0.07), FRAC32 (0.08),
                                      FRAC32 (0.1), FRAC32 (0.11), FRAC32 (0.12), FRAC32
(0.13), FRAC32 (0.14), FRAC32 (0.15), FRAC32 (0.16), FRAC32 (0.17), FRAC32 (0.18),
                                      FRAC32 (0.2), FRAC32 (0.21), FRAC32 (0.22), FRAC32
(0.23), FRAC32 (0.24), FRAC32 (0.25), FRAC32 (0.26), FRAC32 (0.27), FRAC32 (0.28),
                                      FRAC32 (0.3), FRAC32 (0.31), FRAC32 (0.32), FRAC32
(0.33), FRAC32 (0.34), FRAC32 (0.35), FRAC32 (0.36), FRAC32 (0.37), FRAC32 (0.38),
                                      FRAC32 (0.4), FRAC32 (0.41), FRAC32 (0.42), FRAC32
(0.43), FRAC32 (0.44), FRAC32 (0.45), FRAC32 (0.46), FRAC32 (0.47), FRAC32 (0.48),
                                      FRAC32 (0.5), FRAC32 (0.51), FRAC32 (0.52), FRAC32
(0.53), FRAC32 (0.54), FRAC32 (0.55), FRAC32 (0.56), FRAC32 (0.57), FRAC32 (0.58),
                                      FRAC32 (0.6), FRAC32 (0.61), FRAC32 (0.62), FRAC32
(0.63), FRAC32 (0.64), FRAC32 (0.65), FRAC32 (0.66), FRAC32 (0.67), FRAC32 (0.68),
                                      FRAC32 (0.7), FRAC32 (0.71), FRAC32 (0.72), FRAC32
(0.73), FRAC32 (0.74), FRAC32 (0.75), FRAC32 (0.76), FRAC32 (0.77), FRAC32 (0.78),
                                      FRAC32 (0.8), FRAC32 (0.81), FRAC32 (0.82), FRAC32
(0.83), FRAC32 (0.84), FRAC32 (0.85), FRAC32 (0.86), FRAC32 (0.87), FRAC32 (0.88)};

void main(void)
{
    // setting parameters for Lut2D function
    tr32tMyLut2D.pf32Table = &(pf32Table2D[40]);
    tr32tMyLut2D.s32ShamOffset1 = 2;
    tr32tMyLut2D.s32ShamOffset2 = 2;
    // input vector
    f32In1 = FRAC32 (0.5);
    f32In2 = FRAC32 (0.5);

    // output should be 0x7A03D70A ~ FRAC32(0.88)
    f32Out = GFLIB_Lut2D_F32 (f32In1,f32In2,&tr32tMyLut2D);
```

```
    // output should be 0x7A03D70A ~ FRAC32(0.88)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D,Define F32);

    // ############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ############################################################

    // output should be 0x7A03D70A ~ FRAC32(0.88)
    f32Out = GFLIB_Lut2D (f32In1,f32In2,&tr32tMyLut2D);
}
```

4.71 Function GFLIB_Lut2D_F16

This function implements the two-dimensional look-up table.

4.71.1 Declaration

```
tFrac16 GFLIB_Lut2D_F16(tFrac16 f16In1, tFrac16 f16In2, const GFLIB_LUT2D_T_F16 *const
pParam);
```

4.71.2 Arguments

Table 4-91. GFLIB_Lut2D_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In1	**input**	First input variable for which 2D interpolation is performed.
tFrac16	f16In2	**input**	Second input variable for which 2D interpolation is performed.
const GFLIB_LUT2D_T_F16 *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

4.71.3 Return

The interpolated value from the look-up table.

4.71.4 Description

The GFLIB_Lut2D_F16 function performs two dimensional linear interpolation over a 2D table of data.

The following interpolation formulas are used:

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq1**

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq2**

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{y_2 - y_1} \bullet (y - y_1)$$

Equation **GFLIB_Lut2D_Eq3**

where:

- f[x,y] is the interpolated value
- f[x,y $_1$] and f[x,y $_2$] are the ordinate values at, respectively, the beginning and the end of the final interpolating interval
- x $_1$, x $_2$, y $_1$ and y $_2$ are the area values, respectively, the interpolated area
- the x, y are the input values provided to the function in the f16In1 and f16In2 arguments

The interpolating intervals are defined in the table provided by the pf16Table member of the parameters structure. The table contains ordinate values consecutively over the whole interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length and a table index while the interpolating index zero is the table element pointed to by the pf16Table parameter. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example let's consider the following interpolating table:

Table 4-92. GFLIB_Lut2D example table

ordinate (f[x,y])	interpolating index	abscissa (x)	abscissa (y)
-0.5	-1	-1*(2 $^{-1}$)	-1*(2 $^{-1}$)
pf16Table 0.0	0	0*(2 $^{-1}$)	0*(2 $^{-1}$)
0.5	1	1*(2 $^{-1}$)	1*(2 $^{-1}$)
1.0	N/A	2*(2 $^{-1}$)	2*(2 $^{-1}$)

The Table 4-92 contains 4 interpolating points (note four rows). Interpolating interval length in this example is equal to 2^{-1} . The pf16Table parameter points to the second row, defining also the interpolating index 0. The x-coordinates of the interpolating points are calculated in the right column. It should be noticed that the pf16Table pointer does not have to point to the start of the memory area with ordinate values. Therefore the interpolating index can be positive or negative or, even, does not have to have zero in its range.

Special algorithm is used to make the computation efficient, however under some additional assumptions as provided below:

- the values of the interpolated function are 16-bit long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 13
- the provided abscissa for interpolation is 16-bit long

The algorithm performs the following steps:

1. Compute the index representing the interval, in which the bilinear interpolation will be performed in each axis:

$$I_x = x >> s_{\text{IntegralX}}$$

Equation **GFLIB_Lut2D_Eq4**

$$I_y = y >> s_{\text{IntegralY}}$$

Equation **GFLIB_Lut2D_Eq5**

   where I_x and I_y is the interval index and the $s_{\text{IntervalX}}$ and $s_{\text{IntervalY}}$ are the shift amounts provided in the parameters structure as a member s16ShamIntvl1 or s16ShamIntvl2. The operator >> represents the binary arithmetic right shift.
2. Compute the abscissas offset for both axis within an interpolating intervals:

$$\Delta x = x << s_{\text{Offset1}} \,\&\, 0x7FFF$$

Equation **GFLIB_Lut2D_Eq6**

$$\Delta y = y << s_{\text{Offset2}} \,\&\, 0x7FFF$$

Equation **GFLIB_Lut2D_Eq7**

where Δ{x} and Δ{y} are the abscissas offset within an Xinterval and Yinterval. The $s_{Offset1}$ and $s_{Offset2}$ are the shift amounts provided in the parameters structure. The operators << and & represent the binary left shift and the bitwise logical conjunction. It should be noted that the computation represents extraction of some least significant bits of the x and y with the sign bit cleared.

3. Compute the interpolated value by the linear interpolation in X-axis by y_1 value between the ordinates x

$$f\left[x_1, y_1\right] = 16-\text{bit data at address pTable for } x1, y1$$

Equation **GFLIB_Lut2D_Eq8**

$$f\left[x_2, y_1\right] = 16-\text{bit data at address pTable for } x2, y1$$

Equation **GFLIB_Lut2D_Eq9**

$$f\left[x, y_1\right] = f\left[x_1, y_1\right] + \frac{f\left[x_2, y_1\right] - f\left[x_1, y_1\right]}{\Delta x} \bullet \left(x - x_1\right)$$

Equation **GFLIB_Lut2D_Eq10**

where $f[x,y_1]$ is interpolated value, $f[x_1,y_1]$ and $f[x_2,y_1]$ are, the ordinate at the start of the interpolating interval and the ordinate at the end of the interpolating interval. The pTable} is the address provided in the parameters structure pParam->f16Table.

4. The same computation as shown above in X-axis by y_2 value. Interpolation formulas are the same as paragraph 3:

$$f\left[x_1, y_2\right] = 16-\text{bit data at address pTable for } x1, y2$$

Equation **GFLIB_Lut2D_Eq11**

$$f\left[x_2, y_2\right] = 16-\text{bit data at address pTable for } x2, y2$$

Equation **GFLIB_Lut2D_Eq12**

$$f\left[x, y_2\right] = f\left[x_1, y_2\right] + \frac{f\left[x_2, y_2\right] - f\left[x_1, y_2\right]}{\Delta x} \bullet \left(x - x_1\right)$$

Equation **GFLIB_Lut2D_Eq13**

5. Final linear interpolation in Y-axis way Interpolation formulas are the same as paragraph 3 or 4:

$$f\left[x, y_1\right] = \text{result coputed in paragraph 3}$$

Equation **GFLIB_Lut2D_Eq14**

$$f\left[x, y_2\right] = \text{result coputed in paragraph 4}$$

Equation **GFLIB_Lut2D_Eq15**

$$f\left[x, y\right] = f\left[x, y_1\right] + \frac{f\left[x, y_2\right] - f\left[x, y_1\right]}{\Delta y} \bullet \left(y - y_1\right)$$

Equation **GFLIB_Lut2D_Eq16**

It should be noted that due to assumption of the equidistant data points, the division by the interval length is avoided.

It should be noted that the computations are performed with the 16-bit accuracy. In particular the 16 least significant bits are ignored in all multiplications.

The shift amounts shall be provided in the parameters structure (pParam->s16ShamOffset1, pParam->s16ShamIntvl1, pParam->s16ShamOffset2, andpParam->s16ShamIntvl2). The address of the table with the data, the pTable}, shall be defined by the parameter structure member pParam->pf16Table.

The shift amounts, the $s_{Interval1,2}$ and $s_{Offset1,2}$, can be computed with the following formulas:

$$s_{\text{Interval1,2}} = 15 - |n|$$
$$s_{\text{Offset1,2}} = |n|$$

Equation **GFLIB_Lut2D_Eq17**

where n is the integer defining the length of the interpolating interval in the range of -1, -2, ... -29.

The computation of the abscissa offset and the interval index can be viewed also in the following way. The input abscissa value can be divided into two parts. The first n most significant bits of the 16-bit halfword, after the sign bit, compose the interval index, in which interpolation is performed. The rest of the bits form the abscissa offset within the interpolating interval. This simple way to calculate the interpolating interval index and the abscissa offset is the consequence of assumption of all interpolating interval lengths equal 2^{-n}.

It should be noted that the input abscissa value can be positive or negative. If it is positive then the ordinate values are read as in the ordinary data array, that is at or after the data pointer provided in the parameters structure (pParam->pf16Table). However, if it is negative, then the ordinate values are read from the memory, which is located behind the pParam->pf16Table pointer.

Note

The function performs the bilinear interpolation.

CAUTION

The function does not check whether the input values are within a range allowed by the interpolating data table pParam->pf16Table. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data.

4.71.5 Re-entrancy

The function is re-entrant.

4.71.6 Code Example

```
#include "gflib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
GFLIB_LUT2D_T_F16 tr16tMyLut2D = GFLIB_LUT2D_DEFAULT_F16;
tFrac16 pf16Table2D[81] = {FRAC16 (0.0), FRAC16 (0.01), FRAC16 (0.02), FRAC16
(0.03), FRAC16 (0.04), FRAC16 (0.05), FRAC16 (0.06), FRAC16 (0.07), FRAC16 (0.08),
                                      FRAC16 (0.1), FRAC16 (0.11), FRAC16 (0.12), FRAC16
(0.13), FRAC16 (0.14), FRAC16 (0.15), FRAC16 (0.16), FRAC16 (0.17), FRAC16 (0.18),
```

```
                                        FRAC16 (0.2), FRAC16 (0.21), FRAC16 (0.22), FRAC16
(0.23), FRAC16 (0.24), FRAC16 (0.25), FRAC16 (0.26), FRAC16 (0.27), FRAC16 (0.28),
                                        FRAC16 (0.3), FRAC16 (0.31), FRAC16 (0.32), FRAC16
(0.33), FRAC16 (0.34), FRAC16 (0.35), FRAC16 (0.36), FRAC16 (0.37), FRAC16 (0.38),
                                        FRAC16 (0.4), FRAC16 (0.41), FRAC16 (0.42), FRAC16
(0.43), FRAC16 (0.44), FRAC16 (0.45), FRAC16 (0.46), FRAC16 (0.47), FRAC16 (0.48),
                                        FRAC16 (0.5), FRAC16 (0.51), FRAC16 (0.52), FRAC16
(0.53), FRAC16 (0.54), FRAC16 (0.55), FRAC16 (0.56), FRAC16 (0.57), FRAC16 (0.58),
                                        FRAC16 (0.6), FRAC16 (0.61), FRAC16 (0.62), FRAC16
(0.63), FRAC16 (0.64), FRAC16 (0.65), FRAC16 (0.66), FRAC16 (0.67), FRAC16 (0.68),
                                        FRAC16 (0.7), FRAC16 (0.71), FRAC16 (0.72), FRAC16
(0.73), FRAC16 (0.74), FRAC16 (0.75), FRAC16 (0.76), FRAC16 (0.77), FRAC16 (0.78),
                                        FRAC16 (0.8), FRAC16 (0.81), FRAC16 (0.82), FRAC16
(0.83), FRAC16 (0.84), FRAC16 (0.85), FRAC16 (0.86), FRAC16 (0.87), FRAC16 (0.88)};

            void main(void)
            {
                // setting parameters for Lut2D function
                tr16tMyLut2D.pf16Table = &(pf16Table2D[40]);
                tr16tMyLut2D.s16ShamOffset1 = 3;
                tr16tMyLut2D.s16ShamOffset2 = 3;
                // input vector
                f16In1 = FRAC16 (0.5);
                f16In2 = FRAC16 (0.5);

                // output should be 0x7A03 ~ FRAC16(0.88)
                f16Out = GFLIB_Lut2D_F16 (f16In1),f16In2,&tr16tMyLut2D);

                // output should be 0x7A03 ~ FRAC16(0.88)
                f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D,Define F16);

                // ##############################################################
                // Available only if 16-bit fractional implementation selected
                // as default
                // ##############################################################

                // output should be 0x7A03 ~ FRAC16(0.88)
                f16Out = GFLIB_Lut2D (f16In1,f16In2,&tr16tMyLut2D);
            }
```

4.72 Function GFLIB_Lut2D_FLT

This function implements the two-dimensional look-up table.

4.72.1 Declaration

```
tFloat GFLIB_Lut2D_FLT(tFloat fltIn1, tFloat fltIn2, const GFLIB_LUT2D_T_FLT *const pParam);
```

4.72.2 Arguments

Table 4-93. GFLIB_Lut2D_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn1	**input**	First input variable for which 2D interpolation is performed. Input value is in single precision floating data format.
tFloat	fltIn2	**input**	Second input variable for which 2D interpolation is performed. Input value is in single precision floating data format.
const GFLIB_LUT2D_T_FLT *const	pParam	**input**	Pointer to the parameters structure with parameters of the two dimensional look-up table function.

4.72.3 Return

The function returns the interpolated value. The output value is in single precision floating point format.

4.72.4 Description

The GFLIB_Lut2D_FLT function performs a two dimensional linear interpolation over a 2D table of data.

The following interpolation formulas are used:

$$f[x, y_1] = f[x_1, y_1] + \frac{f[x_2, y_1] - f[x_1, y_1]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq1**

$$f[x, y_2] = f[x_1, y_2] + \frac{f[x_2, y_2] - f[x_1, y_2]}{x_2 - x_1} \bullet (x - x_1)$$

Equation **GFLIB_Lut2D_Eq2**

$$f[x, y] = f[x, y_1] + \frac{f[x, y_2] - f[x, y_1]}{y_2 - y_1} \bullet (y - y_1)$$

Equation **GFLIB_Lut2D_Eq3**

where:

- f[x,y] is the interpolated value
- f[x,y $_1$] and f[x,y $_2$] are the ordinate values at, respectively, the beginning and the end of the final interpolating interval
- x $_1$, x $_2$, y $_1$ and y $_2$ are the area values, respectively, the interpolated area
- the x, y are the input values provided to the function in the fltIn1 and fltIn2 arguments

The interpolating intervals are defined in the table provided by the pfltTable member of the parameters structure. The table contains ordinate values consecutively over the entire interpolating range. The abscissa values are assumed to be defined implicitly by a single interpolating interval length. The abscissa value is equal to the multiplication of the interpolating index and the interpolating interval length. For example, a table contains 4 interpolating points. The interpolating interval length in this example is equal to $2^{-1}=0.5$.

It should be noted that the pfltTable pointer does not have to point to the start of the memory area of ordinate values. Therefore, the interpolating index can be positive or negative or, even, does not have to have zero in its range.

A special algorithm is used to make the computation efficient, however, under some additional assumptions as provided below:

- the values of the interpolated function are 32 bits long
- the length of each interpolating interval is equal to 2^{-n}, where n is an integer in the range of 1, 2, ... 29
- the abscissa provided for interpolation is 32 bits long

The bilinear interpolation performs the following steps:

1. Count the equidistant interpolating intervals, in which the bilinear interpolation will be performed:

$$\mathrm{fSegmentsX} = (\mathrm{tFloat})(1 << s32\mathrm{ShamOffset1} + 1))$$
$$\mathrm{fSegmentsY} = (\mathrm{tFloat})(1 << s32\mathrm{ShamOffset2} + 1))$$

Equation **GFLIB_Lut2D_Eq4**

where fSegmentsX and fSegmentsY are the interval lengths, the pParam->s32ShamOffset1pParam->s32ShamOffset2 are the shift amounts provided in the parameters structure as the members s32ShamOffset1, s32ShamOffset2. The operator "<<" represents the binary arithmetic left shift. The explicit casting to tFloat

data type is required because parameters s32ShamOffset1,2 are integer numbers which have to be stored as float variables, and it then indicates the position of the first interpolated value in the X-axis and the Y-axis in the table.

2. $$s32IntvlX=(tS32)((fSegmentsX) \bullet (fltIn1))$$
$$s32IntvlY=(tS32)((fSegmentsY) \bullet (fltIn2))$$

Equation **GFLIB_Lut2D_Eq5**

where s32IntvlX and s32IntvlY are the coordinates of the first ordinate value at the beginning of the interpolating interval, the fSegmentsX and fSegmentsY are the equidistant interpolating intervals in the both axes and the fltIn1, fltIn2 are the input values provided to the function as arguments. The casting to tS32 data type is a final operation of the equation which cuts the fractional part of the value and then indicates the position of the first interpolated value in the X-axis and the Y-axis in the table.

3. Count the number of the interpolation samples in the X-axis:

$$s32Xrange = ((1 < < pParam \rightarrow s32ShamOffset + 1) + 1)$$

Equation **GFLIB_Lut2D_Eq6**

4. $$psfIntvl = (32 - bit \quad data \quad at \quad address \quad pTable) + s32IntvlX + (s32Xrange \bullet fltIntvlY)$$

Equation **GFLIB_Lut2D_Eq7**

where the pTable} is the address provided in the parameters structure pParam->fltTable.

5. Compute the interpolated value by the linear interpolation in the X-axis of the y_1 value between the ordinates x_1 and x

$$z_1 = z_{11} + \frac{(x - x_1) \bullet (z_{21} - z_{11})}{x_2 - x_1}$$

Equation **GFLIB_Lut2D_Eq8**

reduced to (division by the interval length is avoided):

$$z_1 = z_{11} + (z_{21} - z_{11}) \bullet ((tFloat)(x \bullet 2^n) - (tU32)(x \bullet 2^n))$$

Equation **GFLIB_Lut2D_Eq9**

interpretation in C language is following:

$$fDX = fltIn1 * fSegmentsX - s32IntvlX$$

$$fltZ1 = (*psfIntvl) + (((*(psfIntvl + 1)) - (*psfIntvl)) * fDX))$$

Equation **GFLIB_Lut2D_Eq10**

6. The same computation as shown above in the X-axis by the y $_2$ value:

$$z_2 = z_{12} + \frac{(x - x_1) \bullet (z_{22} - z_{12})}{x_2 - x_1}$$

Equation **GFLIB_Lut2D_Eq11**

reduced to (division by the interval length is avoided):

$$z_2 = z_{12} + (z_{22} - z_{12}) \bullet ((tFloat)(x \bullet 2^n) - (tU32)(x \bullet 2^n))$$

Equation **GFLIB_Lut2D_Eq12**

interpretation in C language is the following:

```
fltZ2=(*(psfIntvl+s32Xrange))+(((*(psfIntvl+s32Xrange+1))-(*(psfIntvl+s32Xrange)))*(fDX))
```

Equation **GFLIB_Lut2D_Eq13**

7. Final linear interpolation in the Y-axis:

$$z = z_1 + \frac{(y - y_1) \bullet (z_2 - z_1)}{y_2 - y_1}$$

Equation **GFLIB_Lut2D_Eq14**

reduced to (division by the interval length is avoided):

$$z = z_1 + (z_2 - z_1) \bullet ((tFloat)(y \bullet 2^n) - (tU32)(y \bullet 2^n))$$

Equation **GFLIB_Lut2D_Eq15**

interpretation in C language is the following:

$$return(fltZ1 + ((fltZ2 - fltZ1) * (fltIn2 * fSegmentsY - s32IntvlY)))$$

Equation **GFLIB_Lut2D_Eq16**

The shift amounts shall be provided in the parameters structure pParam->s32ShamOffset1 and pParam->s32ShamOffset2. The address of the table with the

data, the pTable}, shall be defined by the parameter structure member pParam->pfltTable.

The offset amounts s32ShamOffset_{12} can be provided by the following formulas:

$$s32ShamOffset1 = |n_1|$$
$$s32ShamOffset2 = |n_2|$$

Equation **GFLIB_Lut2D_Eq17**

where n_1 and n_2 are the integers defining the length of the interpolating interval in the range of 1, 2, ... 29.

This simple way to calculate the interpolating abscissa offsets is the consequence of assuming that all interpolating interval lengths equal 2^{-n1} and 2^{-n2}.

It should be noted that the input abscissa values can be positive or negative. If they are positive then the ordinate values are read as in the ordinary data array, that are at or after the data pointer provided in the parameters structure pParam->pfltTable. However, if they are negative, then the ordinate values are read from the memory, which is located behind the pParam->pfltTable pointer.

Note

The function performs a bilinear interpolation.

CAUTION

The function does not check whether the input values are within the range allowed by the interpolating data table pParam->pfltTable. If the computed interval index points to data outside the provided data table then the interpolation will be computed with invalid data.

4.72.5 Re-entrancy

The function is re-entrant.

4.72.6 Code Example

```
#include "gflib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;
GFLIB_LUT2D_T_FLT trfltMyLut2D = GFLIB_LUT2D_DEFAULT_FLT;
```

```
tFloat pfltTable2D[81] = {0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08,
                          0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18,
                          0.2, 0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28,
                          0.3, 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38,
                          0.4, 0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48,
                          0.5, 0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58,
                          0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68,
                          0.7, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78,
                          0.8, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87,
0.88};

void main(void)
{
    // setting parameters for Lut2D function
    trfltMyLut2D.pfltTable = &(pfltTable2D[40]);
    trfltMyLut2D.s32ShamOffset1 = 3;
    trfltMyLut2D.s32ShamOffset2 = 3;
    // input vector = 0.5
    fltIn1 = 0.5;
    fltIn2 = 0.5;

    // output should be 0.88
    fltOut = GFLIB_Lut2D_FLT (fltIn1,fltIn2,&trfltMyLut2D);

    // output should be 0.88
    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D,Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.88
    fltOut = GFLIB_Lut2D (fltIn1,fltIn2,&trfltMyLut2D);
}
```

4.73 Function GFLIB_Ramp_F32

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

4.73.1 Declaration

```
tFrac32 GFLIB_Ramp_F32(tFrac32 f32In, GFLIB_RAMP_T_F32 *const pParam);
```

4.73.2 Arguments

Table 4-94. GFLIB_Ramp_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument representing the desired output value.
GFLIB_RAMP_T_F32 *const	pParam	**input, output**	Pointer to the ramp parameters structure.

4.73.3 Return

The function returns a 32-bit value in format Q1.31, which represents the actual ramp output value. This, in time, is approaching the desired (input) value by step increments defined in the pParam structure.

4.73.4 Description

The GFLIB_Ramp function, denoting ANSI-C compatible source code implementation, can be called via the function alias GFLIB_Ramp.

If the desired (input) value is greater than the ramp output value, the function adds the f32RampUp value to the actual output value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the f32RampDown value from the actual value. The output cannot be lower than the desired value.

Functionality of the implemented ramp algorithm can be explained with use of Figure 4-31

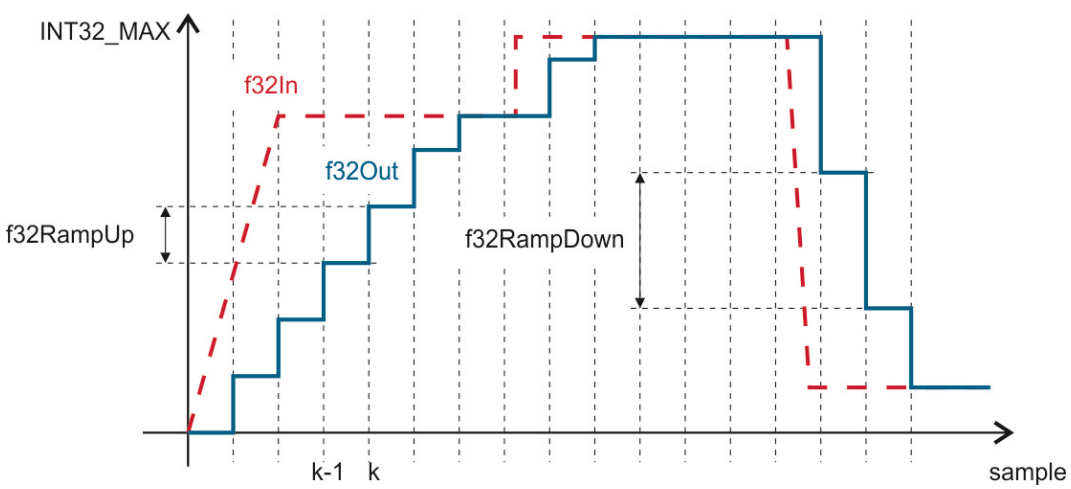

Figure 4-31. GFLIB_Ramp functionality

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_RAMP_DEFAULT macro.

4.73.5 Re-entrancy

The function is re-entrant.

4.73.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_RAMP_T_F32 f32trMyRamp = GFLIB_RAMP_DEFAULT_F32;

void main(void)
{
    // increment/decrement coefficients
    f32trMyRamp.f32RampUp = FRAC32 (0.1);
    f32trMyRamp.f32RampDown = FRAC32 (0.03333333);
    // input value = 0.5
    f32In  = FRAC32 (0.5);

    // output should be 0x0CCCCCCC ~ FRAC32(0.1)
    f32Out = GFLIB_Ramp_F32 (f32In, &f32trMyRamp);

    // output should be 0x0CCCCCCC ~ FRAC32(0.1)
    f32trMyRamp.f32State = 0;
    f32Out = GFLIB_Ramp (f32In, &f32trMyRamp, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x0CCCCCCC ~ FRAC32(0.1)
    f32trMyRamp.f32State = 0;
    f32Out = GFLIB_Ramp (f32In, &f32trMyRamp);
}
```

4.74 Function GFLIB_Ramp_F16

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

4.74.1 Declaration

```
tFrac16 GFLIB_Ramp_F16(tFrac16 f16In, GFLIB_RAMP_T_F16 *const pParam);
```

4.74.2 Arguments

Table 4-95. GFLIB_Ramp_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument representing the desired output value.
GFLIB_RAMP_T_F16 *const	pParam	**input, output**	Pointer to the ramp parameters structure.

4.74.3 Return

The function returns a 16-bit value in format Q1.15, which represents the actual ramp output value. This, in time, is approaching the desired (input) value by step increments defined in the pParam structure.

4.74.4 Description

The GFLIB_Ramp function, denoting ANSI-C compatible source code implementation, can be called via the function alias GFLIB_Ramp.

If the desired (input) value is greater than the ramp output value, the function adds the f16RampUp value to the actual output value. The output cannot be greater than the desired value.

If the desired value is lower than the actual value, the function subtracts the f16RampDown value from the actual value. The output cannot be lower than the desired value.

Functionality of the implemented ramp algorithm can be explained with use of Figure 4-32

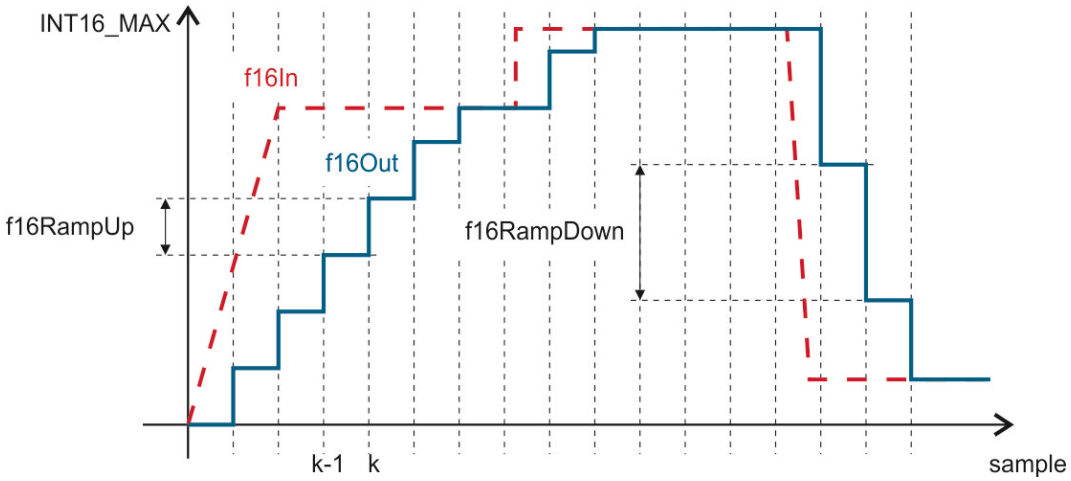

Figure 4-32. GFLIB_Ramp functionality

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_RAMP_DEFAULT macro.

4.74.5 Re-entrancy

The function is re-entrant.

4.74.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_RAMP_T_F16 f16trMyRamp = GFLIB_RAMP_DEFAULT_F16;

void main(void)
{
    // increment/decrement coefficients
    f16trMyRamp.f16RampUp = FRAC16 (0.1);
    f16trMyRamp.f16RampDown = FRAC16 (0.03333333);
    // input value = 0.5
    f16In  = FRAC16 (0.5);

    // output should be 0x0CCC ~ FRAC16(0.1)
    f16Out = GFLIB_Ramp_F16 (f16In, &f16trMyRamp);

    // output should be 0x0CCC ~ FRAC16(0.1)
    f16trMyRamp.f16State = 0;
    f16Out = GFLIB_Ramp (f16In, &f16trMyRamp, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################
```

```
    // output should be 0x0CCC ~ FRAC16(0.1)
    f16trMyRamp.f16State = 0;
    f16Out = GFLIB_Ramp (f16In, &f16trMyRamp);
}
```

4.75 Function GFLIB_Ramp_FLT

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

4.75.1 Declaration

```
tFloat GFLIB_Ramp_FLT(tFloat fltIn, GFLIB_RAMP_T_FLT *const pParam);
```

4.75.2 Arguments

Table 4-96. GFLIB_Ramp_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument representing the desired output value. Input value is in single precision floating data format.
GFLIB_RAMP_T_FLT *const	pParam	**input, output**	Pointer to the ramp parameters structure. Arguments of the structure contain single precision floating point values.

4.75.3 Return

The function returns a value in single precision floating point format, which represents the actual ramp output. This value can be also described as a Ramp state value increased/ decreased by a slope in order to achieve the desired input value.

4.75.4 Description

The GFLIB_Ramp function, denoting ANSI-C compatible source code implementation, can be called via the function alias GFLIB_Ramp.

If the desired (input) value is greater than the ramp output value, the function adds the fltRampUp value to the actual output value. The output cannot be greater than the desired value. If the desired value is lower than the actual value, the function subtracts the fltRampDown value from the actual value. The output cannot be lower than the desired value. The ramp function is performed as follows:

$$fltOut = \begin{cases} (fltState + fltRampUp) & \text{if} \quad (fltIn \geq fltState) \\ (fltState - fltRampDown) & \text{if} \quad (fltIn < fltState) \\ (fltIn) & \text{if} \quad ((fltState + fltRampUp) \geq fltIn) \\ (fltIn) & \text{if} \quad ((fltState - fltRampDown) \leq fltIn) \end{cases}$$

Equation **GFLIB_Ramp_Eq1**

Functionality of the implemented ramp algorithm can be explained with the use of Figure 4-33

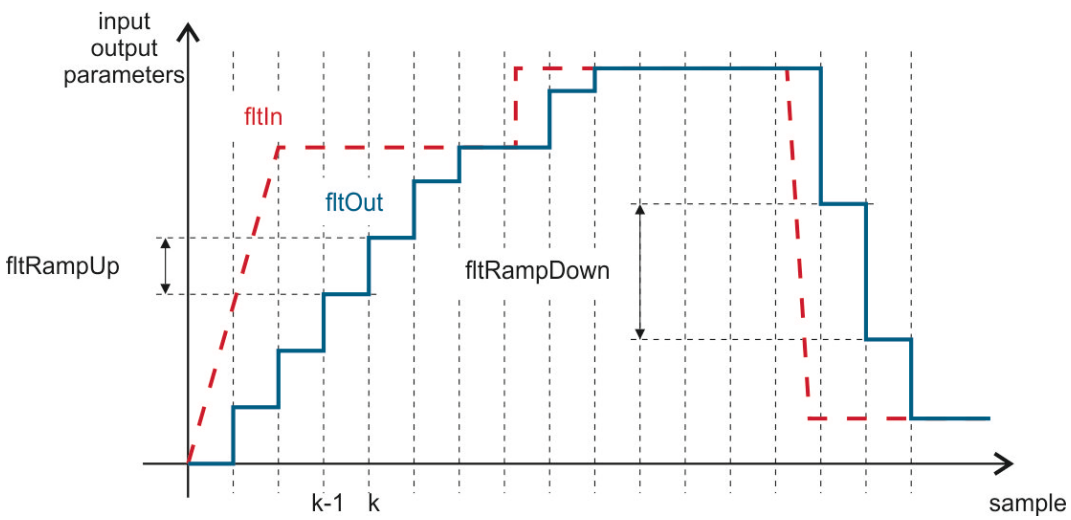

Figure 4-33. GFLIB_Ramp functionality

Note

All parameters and states used by the function can be reset during declaration using the GFLIB_RAMP_DEFAULT macro.

4.75.5 Re-entrancy

The function is re-entrant.

4.75.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_RAMP_T_FLT flttrMyRamp = GFLIB_RAMP_DEFAULT_FLT;

void main(void)
{
    // increment/decrement coefficients
    flttrMyRamp.fltRampUp = (tFloat)(0.1);
    flttrMyRamp.fltRampDown = (tFloat)(0.03333333);
    // input value = 0.5
    fltIn  = (tFloat)(0.5);

    // output should be 0.1
    fltOut = GFLIB_Ramp_FLT (fltIn, &flttrMyRamp);

    // output should be 0.1
    flttrMyRamp.fltState = 0;
    fltOut = GFLIB_Ramp (fltIn, &flttrMyRamp, Define FLT);

    // ##################################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##################################################################

    // output should be 0.1
    flttrMyRamp.fltState = 0;
    fltOut = GFLIB_Ramp (fltIn, &flttrMyRamp);
}
```

4.76 Function GFLIB_Sign_F32

This function returns the signum of input value.

4.76.1 Declaration

```
tFrac32 GFLIB_Sign_F32(tFrac32 f32In);
```

4.76.2 Arguments

Table 4-97. GFLIB_Sign_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument.

4.76.3 Return

The function returns the sign of the input argument.

4.76.4 Description

The GFLIB_Sign function calculates the sign of the input argument. The function return value is computed as follows. If the input value is negative, then the return value will be set to "-1" (0x80000000 hex), if the input value is zero, then the function returns "0" (0x0 hex), otherwise if the input value is greater than zero, the return value will be "1" (0x7fffffff hex).

In mathematical terms, the function works according to the equation below:

$$y_{out} = \begin{cases} 1 & \text{if } x_{in} > 0 \\ 0 & \text{if } x_{in} = 0 \\ -1 & \text{if } x_{in} < 0 \end{cases}$$

Equation **GFLIB_Sign_Eq1**

where:

- y_{out} is the return value
- x_{in} is the input value provided as the f32In parameter

Note

The input and the output values are in the 32-bit fixed point fractional data format.

4.76.5 Re-entrancy

The function is re-entrant.

4.76.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.5
```

```
    f32In  = FRAC32 (0.5);

    // output should be 0x7FFFFFFF ~ FRAC32(1)
    f32Out = GFLIB_Sign_F32 (f32In);

    // output should be 0x7FFFFFFF ~ FRAC32(1)
    f32Out = GFLIB_Sign (f32In,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x7FFFFFFF ~ FRAC32(1)
    f32Out = GFLIB_Sign (f32In);
}
```

4.77 Function GFLIB_Sign_F16

This function returns the signum of input value.

4.77.1 Declaration

```
tFrac16 GFLIB_Sign_F16(tFrac16 f16In);
```

4.77.2 Arguments

Table 4-98. GFLIB_Sign_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument.

4.77.3 Return

The function returns the sign of the input argument.

4.77.4 Description

The GFLIB_Sign function calculates the sign of the input argument. The function return value is computed as follows. If the input value is negative, then the return value will be set to "-1" (0x8000 hex), if the input value is zero, then the function returns "0" (0x0 hex), otherwise if the input value is greater than zero, the return value will be "1" (0x7fff hex).

In mathematical terms, the function works according to the equation below:

$$y_{out} = \begin{cases} 1 & \text{if } x_{in} > 0 \\ 0 & \text{if } x_{in} = 0 \\ -1 & \text{if } x_{in} < 0 \end{cases}$$

Equation **GFLIB_Sign_Eq1**

where:

- y_{out} is the return value
- x_{in} is the input value provided as the f16In parameter

Note

The input and the output values are in the 16-bit fixed point fractional data format.

4.77.5 Re-entrancy

The function is re-entrant.

4.77.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.5
    f16In  = FRAC16 (0.5);

    // output should be 0x7FFF ~ FRAC16(1)
    f16Out = GFLIB_Sign_F16 (f16In);

    // output should be 0x7FFF ~ FRAC16(1)
    f16Out = GFLIB_Sign (f16In,Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFF ~ FRAC16(1)
    f16Out = GFLIB_Sign (f16In);
}
```

4.78 Function GFLIB_Sign_FLT

This function returns the signum of input value.

4.78.1 Declaration

```
tFloat GFLIB_Sign_FLT(tFloat fltIn);
```

4.78.2 Arguments

Table 4-99. GFLIB_Sign_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument.

4.78.3 Return

The function returns the sign of the input argument.

4.78.4 Description

The GFLIB_Sign function calculates the sign of the input argument. The function return value is computed as follows. If the input value is negative, then the return value will be set to "-1", if the input value is zero, then the function returns "0", otherwise if the input value is greater than zero, the return value will be "1".

In mathematical terms, the function works according to the equation below:

$$y_{out} = \begin{cases} 1 & \text{if } x_{in} > 0 \\ 0 & \text{if } x_{in} = 0 \\ -1 & \text{if } x_{in} < 0 \end{cases}$$

Equation **GFLIB_Sign_Eq1**

where:

- y_{out} is the return value
- x_{in} is the input value provided as the fltIn parameter

Note

The input and the output are in single precision floating point data format.

4.78.5 Re-entrancy

The function is re-entrant.

4.78.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = 0.5
    fltIn  = (tFloat)(0.5);

    // output should be 1
    fltOut = GFLIB_Sign_FLT (fltIn);

    // output should be 1
    fltOut = GFLIB_Sign (fltIn,Define FLT);

    // ###############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ###############################################################

    // output should be 1
    fltOut = GFLIB_Sign (fltIn);
}
```

4.79 Function GFLIB_Sin_F32

This function implements polynomial approximation of sine function.

4.79.1 Declaration

```
tFrac32 GFLIB_Sin_F32(tFrac32 f32In, const GFLIB_SIN_T_F32 *const pParam);
```

4.79.2 Arguments

Table 4-100. GFLIB_Sin_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_SIN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.79.3 Return

The function returns the sin of the input argument as a fixed point 32-bit number, normalized between [-1, 1).

4.79.4 Description

The GFLIB_Sin_F32 function provides a computational method for calculation of a standard trigonometric *sine* function sin(x), using the 9th order Taylor polynomial approximation. The Taylor polynomial approximation of a *sine* function is described as follows:

$$\sin\left(x\right) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \_ = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Sin_Eq1**

where x is the input angle.

The 9th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation. Because the GFLIB_Sin_F32 function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f32In [-1, 1) into the correct range [- π, π), the input f32In must be multiplied by π. So the fixed point fractional implementation of the GFLIB_Sin_F32 function, using 9th order Taylor approximation, is given as follows:

$$\sin\left(\pi \bullet f32\mathrm{In}\right) = \left(\pi \bullet f32\mathrm{In}\right) - \frac{(\pi \bullet f32\mathrm{In})^3}{3!} + \frac{(\pi \bullet f32\mathrm{In})^5}{5!} - \frac{(\pi \bullet f32\mathrm{In})^7}{7!} + \frac{(\pi \bullet f32\mathrm{In})^9}{9!}$$

Equation **GFLIB_Sin_Eq2**

The 9th order polynomial approximation of the sine function has a very good accuracy in the range [- π/2, π/2) of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function sin(x) = sin(π - x) is utilized. Therefore, the input argument is transferred to be always in the range [- π/2, π/2) and the Taylor polynomial is calculated only in the range of the argument [- π/2, π/2).

To make calculations more precise, the given argument value f32In (that is to be transferred into the range [-0.5, 0.5) due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of f32In ^{2}, used in the calculations, is in the range [-1, 1) instead of [-0.25, 0.25). Shifting the input value by 1 bit to the left will increase the accuracy of the calculated sin(π * f32In) function. Implementing such a scale on the approximation function described by equation GFLIB_Sin_Eq2, results in the following:

$$\sin\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right) = \_ = \left(\frac{\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right)}{2} - \frac{\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right)^3}{3 \bullet 2} + \frac{\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right)^5}{5 \bullet 2} - \frac{\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right)^7}{7 \bullet 2} + \frac{\left(f32In \bullet 2 \bullet \frac{\pi}{2}\right)^9}{9 \bullet 2}\right) \bullet 2$$

Equation **GFLIB_Sin_Eq3**

Equation GFLIB_Sin_Eq3 can be further rewritten into the following form:

$$\begin{aligned}\sin\left(f32In \bullet \pi\right) = & \left(f32In \bullet 2\right)\left(a_1 + \right. \\ & + \left(f32In \bullet 2\right)^2\left(a_2 + \right. \\ & + \left(f32In \bullet 2\right)^2\left(a_3 + \right. \\ & + \left(f32In \bullet 2\right)^2\left(a_4 + \right. \\ & + \left(f32In \bullet 2\right)^2\left(a_5\right)\right)\right)\right)\right) \bullet 2\end{aligned}$$

Equation **GFLIB_Sin_Eq4**

where a $_1$ ... a $_5$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 32-bit signed fractional numbers):

$$a_1 = \frac{\frac{\pi}{2}}{2} = 0.785398163397448 \Rightarrow \frac{\left(\frac{\pi}{2}\right)}{2} \bullet 2^{31} = 0x6487ED51$$

$$a_2 = -\frac{\left(\frac{\pi}{2}\right)^3}{3!2} = -0.322982048753123 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^3}{3!2} \bullet 2^{31} = 0xD6A88634$$

$$a_3 = \frac{\left(\frac{\pi}{2}\right)^5}{5!2} = 0.03988463131230835 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^5}{5!2} \bullet 2^{31} = 0x0519AF1A$$

$$a_4 = -\frac{\left(\frac{\pi}{2}\right)^7}{7!2} = -0.00234087706765934 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^7}{7!2} \bullet 2^{31} = 0xFFB34B4D$$

$$a_5 = \frac{\left(\frac{\pi}{2}\right)^9}{9!2} = 8.02205923936799e^{-005} \Rightarrow \frac{\left(\frac{\pi}{2}\right)^9}{9!2} \bullet 2^{31} = 0x0002A0F0$$

Equation **GFLIB_Sin_Eq5**

Therefore, the resulting equation has the following form:

$$\begin{aligned}\sin(f32In \bullet \pi) = {} & (f32In \bullet 2)(0x6487ED51 + \\ & + (f32In \bullet 2)^2(0xD6A88634 + \\ & + (f32In \bullet 2)^2(0x0519AF1A + \\ & + (f32In \bullet 2)^2(0xFFB34B4D + \\ & + (f32In \bullet 2)^2(0x0002A0F0)))))) \bullet 2\end{aligned}$$

Equation **GFLIB_Sin_Eq6**

Figure 4-34 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from GFLIB_Sin_F32, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

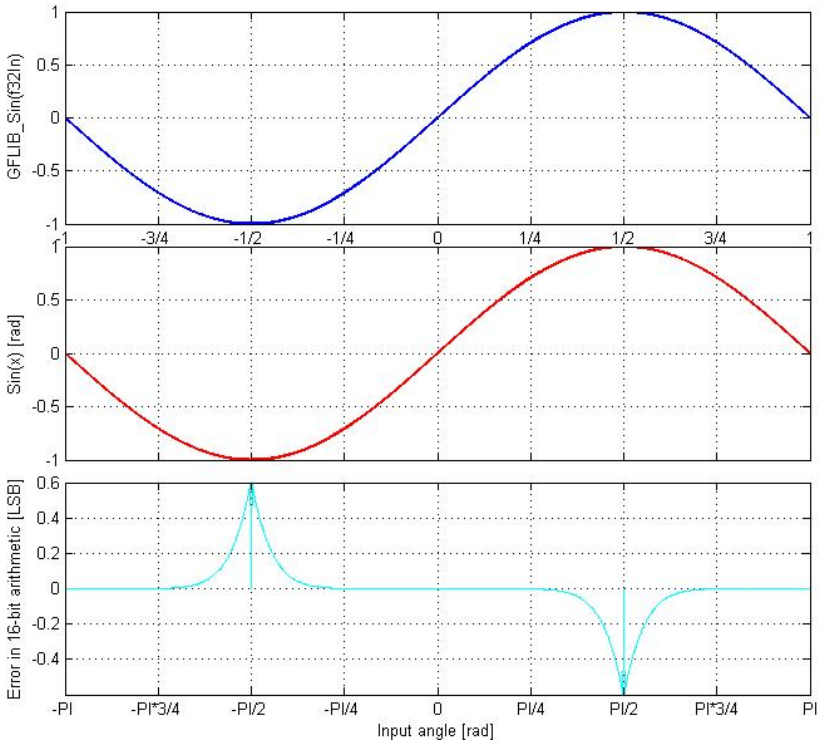

Figure 4-34. sin(x) vs. GFLIB_Sin_F32(f32In)

Note

The input angle (f32In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Sin_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Sin(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SIN_DEFAULT_F32 approximation coefficients are used.

4.79.5 Re-entrancy

The function is re-entrant.

4.79.6 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f32Angle  = FRAC32 (0.5);

    // output should be 0x7FFF8000
    f32Output = GFLIB_Sin_F32 (f32Angle, GFLIB_SIN_DEFAULT_F32);

    // output should be 0x7FFF8000
    f32Output = GFLIB_Sin (f32Angle, GFLIB_SIN_DEFAULT_F32, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x7FFF8000
    f32Output = GFLIB_Sin (f32Angle);
}
```

4.80 Function GFLIB_Sin_F16

This function implements polynomial approximation of sine function.

4.80.1 Declaration

```
tFrac16 GFLIB_Sin_F16(tFrac16 f16In, const GFLIB_SIN_T_F16 *const pParam);
```

4.80.2 Arguments

Table 4-101. GFLIB_Sin_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_SIN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.80.3 Return

The function returns the sin of the input argument as a fixed point 16-bit number, normalized between [-1, 1).

4.80.4 Description

The GFLIB_Sin_F16 function provides a computational method for calculation of a standard trigonometric *sine* function sin(x), using the 7th order Taylor polynomial approximation. The Taylor polynomial approximation of a *sine* function is described as follows:

$$\sin\left(x\right) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \_ = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

Equation **GFLIB_Sin_Eq1**

where x is the input angle.

The 9th order polynomial approximation is chosen as sufficient an order to achieve the best ratio between calculation accuracy and speed of calculation. Because the GFLIB_Sin_F16 function is implemented with consideration to fixed point fractional arithmetic, all variables are normalized to fit into the [-1, 1) range. Therefore, in order to cast the fractional value of the input angle f16In [-1, 1) into the correct range [- π, π), the input f16In must be multiplied by π. So the fixed point fractional implementation of the GFLIB_Sin_F16 function, using 7th order Taylor approximation, is given as follows:

$$\sin\left(\pi \bullet f16\mathrm{In}\right) = \left(\pi \bullet f16\mathrm{In}\right) - \frac{(\pi \bullet f16\mathrm{In})^3}{3!} + \frac{(\pi \bullet f16\mathrm{In})^5}{5!} - \frac{(\pi \bullet f16\mathrm{In})^7}{7!}$$

Equation **GFLIB_Sin_Eq2**

The 7th order polynomial approximation of the sine function has a very good accuracy in the range [- π/2, π/2) of the argument, but in wider ranges the calculation error quickly increases. To minimize the error without having to use a higher order polynomial, the symmetry of the sine function sin(x) = sin(π - x) is utilized. Therefore, the input argument is transferred to be always in the range [- π/2, π/2) and the Taylor polynomial is calculated only in the range of the argument [- π/2, π/2).

To make calculations more precise, the given argument value f16In (that is to be transferred into the range [-0.5, 0.5) due to the *sine* function symmetry) is shifted by 1 bit to the left (multiplied by 2). Then, the value of f16In ^{2}, used in the calculations, is in the range [-1, 1) instead of [-0.25, 0.25). Shifting the input value by 1 bit to the left will increase the accuracy of the calculated sin(π * f16In) function. Implementing such a scale on the approximation function described by equation GFLIB_Sin_Eq2, results in the following:

$$\sin\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right) = \_ = \left(\frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)}{2} - \frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)^3}{3 \bullet 2} + \frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)^5}{5 \bullet 2} - \frac{\left(f16In \bullet 2 \bullet \frac{\pi}{2}\right)^7}{7 \bullet 2}\right) \bullet 2$$

Equation **GFLIB_Sin_Eq3**

Equation GFLIB_Sin_Eq3 can be further rewritten into the following form:

$$\begin{aligned}\sin(f16In \bullet \pi) \;=\; & (f16In \bullet 2)(a_1 + \\ & + (f16In \bullet 2)^2(a_2 + \\ & + (f16In \bullet 2)^2(a_3 + \\ & + (f16In \bullet 2)^2(a_4)))))) \bullet 2\end{aligned}$$

Equation **GFLIB_Sin_Eq4**

where a $_1$ ... a $_4$ are coefficients of the approximation polynomial, which are calculated as follows (represented as 16-bit signed fractional numbers):

$$\begin{aligned}
a_1 &= \frac{\frac{\pi}{2}}{} = 0.785398163397448 \Rightarrow \frac{\left(\frac{\pi}{2}\right)}{2} \bullet 2^{15} = 0x6487 \\
a_2 &= -\frac{\left(\frac{\pi}{2}\right)^3}{3 \bullet 2} = -0.322982048753123 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^3}{3 \bullet 2} \bullet 2^{15} = 0xD6A9 \\
a_3 &= \frac{\left(\frac{\pi}{2}\right)^5}{5 \bullet 2} = 0.03988463131230835 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^5}{5 \bullet 2} \bullet 2^{15} = 0x051A \\
a_4 &= -\frac{\left(\frac{\pi}{2}\right)^7}{7 \bullet 2} = -0.00234087706765934 \Rightarrow \frac{\left(\frac{\pi}{2}\right)^7}{7 \bullet 2} \bullet 2^{15} = 0xFFB3
\end{aligned}$$

Equation **GFLIB_Sin_Eq5**

Therefore, the resulting equation has the following form:

$$\begin{aligned}\sin(f16In \bullet \pi) &= (f16In \bullet 2)(0x6488 + \\ &+ (f16In \bullet 2)^2(0xD6A9 + \\ &+ (f16In \bullet 2)^2(0x051A + \\ &+ (f16In \bullet 2)^2(0xFFB3)))))) \bullet 2\end{aligned}$$

Equation **GFLIB_Sin_Eq6**

Figure 4-35 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from GFLIB_Sin_F16, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

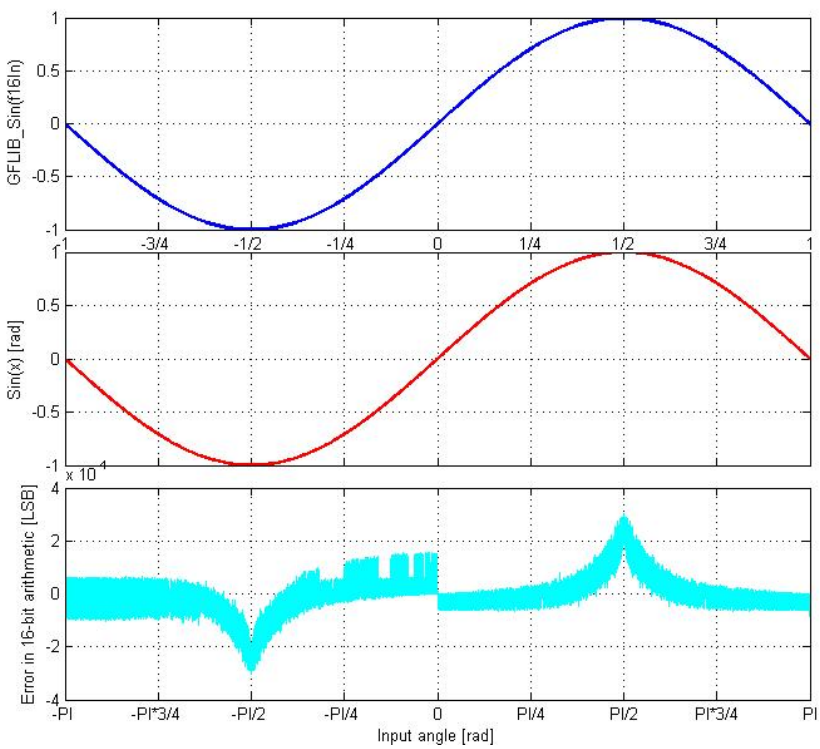

Figure 4-35. sin(x) vs. GFLIB_Sin_F16(f16In)

Note

The input angle (f16In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Sin_F16(f16In, &pParam)), where the &pParam is pointer to

approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.

- With additional implementation parameter (i.e. GFLIB_Sin(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(f16In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SIN_DEFAULT_F16 approximation coefficients are used.

4.80.5 Re-entrancy

The function is re-entrant.

4.80.6 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.5 => pi/2
    f16Angle  = FRAC16 (0.5);

    // output should be 0x7FFF
    f16Output = GFLIB_Sin_F16 (f16Angle, GFLIB_SIN_DEFAULT_F16);

    // output should be 0x7FFF
    f16Output = GFLIB_Sin (f16Angle, GFLIB_SIN_DEFAULT_F16, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x7FFF
    f16Output = GFLIB_Sin (f16Angle);
}
```

4.81 Function GFLIB_Sin_FLT

This function implements polynomial approximation of sine function.

4.81.1 Declaration

```
tFloat GFLIB_Sin_FLT(tFloat fltIn, const GFLIB_SIN_T_FLT *const pParam);
```

4.81.2 Arguments

Table 4-102. GFLIB_Sin_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians between [- π, π].
const GFLIB_SIN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

4.81.3 Return

The function returns the sine of the input argument as a single precision floating point number.

4.81.4 Description

The GFLIB_Sin_FLT function provides a computational method for the calculation of a standard trigonometric *sine* function sin(x), using the floating point optimized minimax polynomial approximation. The following equation describes the chosen approach of calculating the *sine* function:

$$\sin(\mathrm{fltIn}) = \mathrm{fltIn} + \mathrm{fltIn}^3 \bullet a_2 + \mathrm{fltIn}^5 \bullet a_1 + \mathrm{fltIn}^7 \bullet a_0$$

Equation **GFLIB_Sin_Eq1**

where fltIn is the input angle.

Equation GFLIB_Sin_Eq1 can be rewritten into the following form:

$$\sin(\text{fltIn}) = \text{fltIn} + \text{fltIn}^3 \bullet (a_2 + \text{fltIn}^2 \bullet (a_1 + \text{fltIn}^2 \bullet a_0))$$

Equation **GFLIB_Sin_Eq2**

The floating point optimized minimax approximation is chosen as sufficient in order to achieve the best ratio between calculation accuracy and execution speed. Optimized floating point minimax approximation of *sine* function reaches the best precision in the [- π/2, π/2] range. To calculate the values in the interval [- π, - π/2) and (π/2, π] the following equation can be used:

$$\sin(\text{fltIn}) = \sin(\pi - \text{fltIn})$$

Equation **GFLIB_Sin_Eq3**

for interval (π/2, π]

$$\sin(\text{fltIn}) = -\sin(\pi + \text{fltIn})$$

Equation **GFLIB_Sin_Eq4**

for interval [- π,-{ π/2)

Equations GFLIB_Sin_Eq3 and GFLIB_Sin_Eq4 can be transformed to the following formula:

$$\sin(\text{fltIn}) = \begin{cases} \sin(\text{fltIn} + \pi) & \text{if} \quad -\pi \le \text{fltIn} < -\frac{\pi}{2} \\ \sin(\text{fltIn}) & \text{if} \quad -\frac{\pi}{2} \le \text{fltIn} \le \frac{\pi}{2} \\ \sin(\pi - \text{fltIn}) & \text{if} \quad \frac{\pi}{2} < \text{fltIn} < \pi \end{cases}$$

Equation **GFLIB_Sin_Eq5**

The floating point optimized approximation coefficients used for GFLIB_Sin_FLT calculation are noted in Table 4-103.

Table 4-103. Approximation polynomial coefficients

a_0	a_1	a_2
-0.000184881402886	0.008311899801389	-0.166655540927577

Figure 4-36 depicts a floating point *sine* function generated from Matlab and the approximated value of the *sine* function obtained from GFLIB_Sin_FLT, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

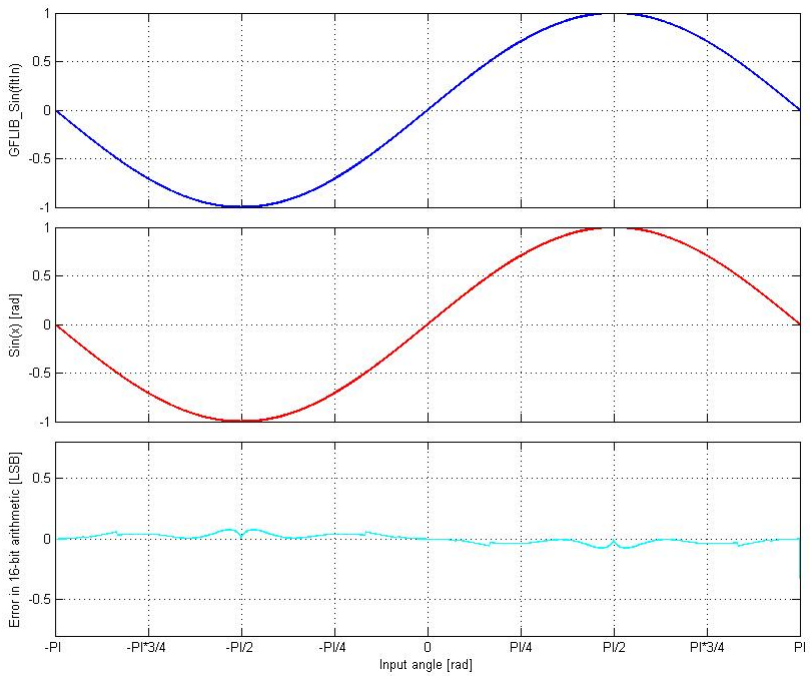

Figure 4-36. sin(x) vs. GFLIB_Sin_FLT(fltIn)

Note

The input angle (fltIn) is in single precision floating point format considering the input angle directly in radians. As during the computation the irrational value of π is used for subtraction, the correction constant is used to increase the calculation precision in boundary intervals. This correction constant is equal to the difference of π computed in double and in single precision. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Sin_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Sin(fltIn, &pParam, FLT), where the &pParam is

pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_SIN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Sin(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_SIN_DEFAULT_FLT approximation coefficients are used.

4.81.5 Re-entrancy

The function is re-entrant.

4.81.6 Code Example

```
#include "gflib.h"

tFloat fltAngle;
tFloat fltOutput;

void main(void)
{
    // input angle = 1.5707963 => pi/2
    fltAngle  = (tFloat)(1.5707963);

    // output should be 1
    fltOutput = GFLIB_Sin_FLT (fltAngle, GFLIB_SIN_DEFAULT_FLT);

    // output should be 1
    fltOutput = GFLIB_Sin (fltAngle, GFLIB_SIN_DEFAULT_FLT, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1
    fltOutput = GFLIB_Sin (fltAngle);
}
```

4.82 Function GFLIB_Sqrt_F32

This function returns the square root of input value.

4.82.1 Declaration

```
tFrac32 GFLIB_Sqrt_F32(tFrac32 f32In);
```

4.82.2 Arguments

Table 4-104. GFLIB_Sqrt_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	The input value.

4.82.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

4.82.4 Description

The GFLIB_Sqrt_F32 function computes the square root of the input value.

The computations are made by a simple iterative testing of each bit starting from the most significant one. In total, 15 iterations are made, each performing the following steps:

1. Add to the tentative square root value a single testing bit.
2. Square the tentative square root value and test whether it is greater or lower than the input value.
3. If greater, then clear the bit in the tentative square root value.
4. Shift the single testing bit right

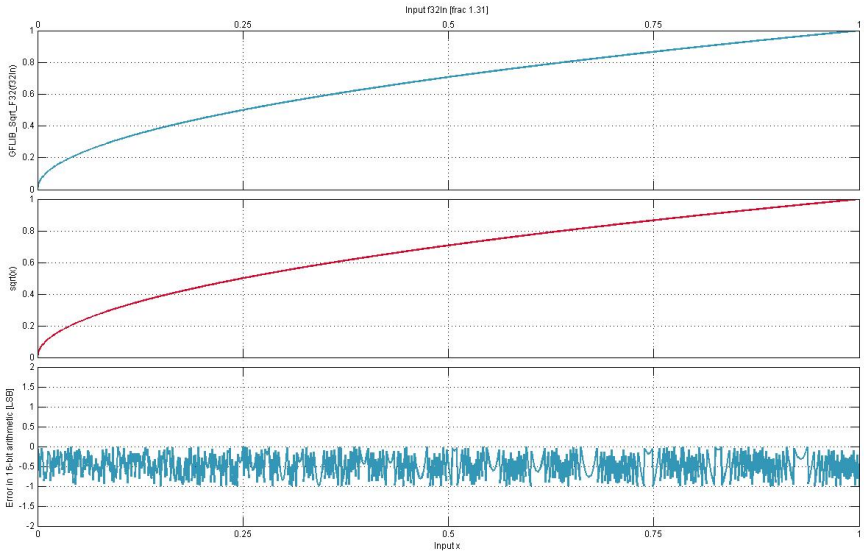

Figure 4-37. real sqrt(x) vs. GFLIB_Sqrt(f32In)

Note

The input value is limited to the range [0, 1), if not within this range the computed value is undefined.

4.82.5 Re-entrancy

The function is re-entrant.

4.82.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.5
    f32In = FRAC32 (0.5);

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt_F32 (f32In);

    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt (f32In,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################
```

```
    // output should be 0x5A820000 ~ FRAC32(0.70710678)
    f32Out = GFLIB_Sqrt (f32In);
}
```

4.83 Function GFLIB_Sqrt_F16

This function returns the square root of input value.

4.83.1 Declaration

```
tFrac16 GFLIB_Sqrt_F16(tFrac16 f16In);
```

4.83.2 Arguments

Table 4-105. GFLIB_Sqrt_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	The input value.

4.83.3 Return

The function returns the square root of the input value. The return value is within the [0, 1) fraction range.

4.83.4 Description

The GFLIB_Sqrt_F16 function computes the square root of the input value.

The computations are made by a simple iterative testing of each bit starting from the most significant one. In total, 15 iterations are made, each performing the following steps:

1. Add to the tentative square root value a single testing bit.
2. Square the tentative square root value and test whether it is greater or lower than the input value.
3. If greater, then clear the bit in the tentative square root value.
4. Shift the single testing bit right

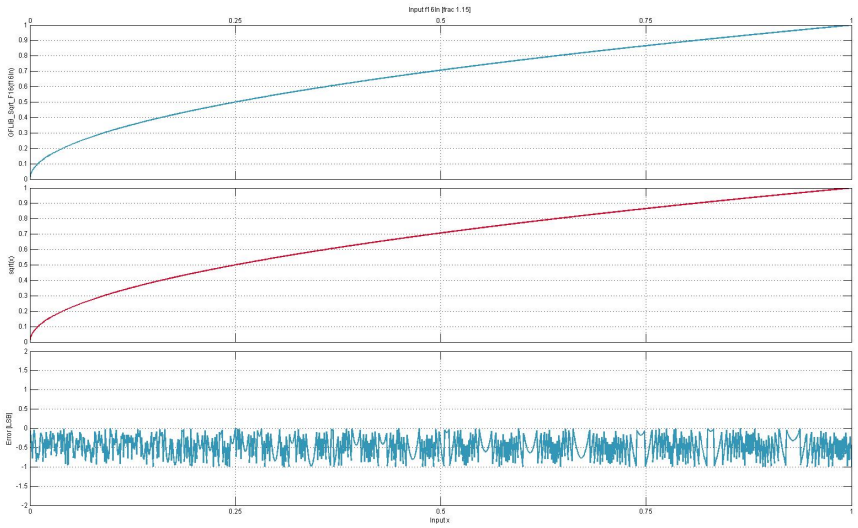

Figure 4-38. real sqrt(x) vs. GFLIB_Sqrt(f16In)

Note

The input value is limited to the range [0, 1), if not within this range the computed value is undefined.

4.83.5 Re-entrancy

The function is re-entrant.

4.83.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.5
    f16In = FRAC16 (0.5);

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt_F16 (f16In);

    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt (f16In,Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################
```

```
    // output should be 0x5A82 ~ FRAC16(0.70710678)
    f16Out = GFLIB_Sqrt (f16In);
}
```

4.84 Function GFLIB_Sqrt_FLT

This function returns the square root of input value.

4.84.1 Declaration

```
tFloat GFLIB_Sqrt_FLT(tFloat fltIn);
```

4.84.2 Arguments

Table 4-106. GFLIB_Sqrt_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	The input value.

4.84.3 Return

The function returns the square root of the input value. The return value is in single precision floating point format.

4.84.4 Description

The GFLIB_Sqrt_FLT function computes the square root of the input value.

The computations are made by an approximation with two additional "Babylonian" steps for elimination quadratic errors. This technique is based on the fact that the IEEE floating point format approximates base-2 logarithm. For example, you can get the approximate logarithm of 32-bit single precision floating point number by translating its binary representation as an integer, scaling it by 2^{-23} and removing a bias of 127 as shown in the following formula:

$$x_{\text{integer}} \bullet 2^{-23} - 127 = \log_2\left(x_{\text{float}}\right)$$

Equation **GFLIB_Sqrt_Eq1**

In the above, the operations to remove last exponent bit and add the bias (according to standard IEEE754) can be combined into a single operation. The formula can be rewritten into C language as:

$$\text{tmp} = (1 << 29) + (x_{\text{integer}} >> 1) - (1 << 22);$$

Equation **GFLIB_Sqrt_Eq2**

For error elimination are the two Babylonian steps added to give a quadratic improvement:

$$x_{\text{float}} = x_{\text{float}} + \frac{x_{\text{integer}}}{x_{\text{float}}}$$

Equation **GFLIB_Sqrt_Eq3**

$$x_{\text{float}} = 0.25f \bullet x_{\text{float}} + \frac{x_{\text{integer}}}{x_{\text{float}}}$$

Equation **GFLIB_Sqrt_Eq4**

Figure 4-39 depicts a floating point *sqrt(x)* function generated from Matlab and the calculated value of the *sqrt* function obtained from GFLIB_Sqrt_FLT, plus their difference. The course of calculation accuracy as a function of the input value can be observed from this figure. The computed value is equal to or is 1LSB less than the true square root value. In order to obtain a value with a 0.5LSB (16bit) accuracy, additional iterations are required.

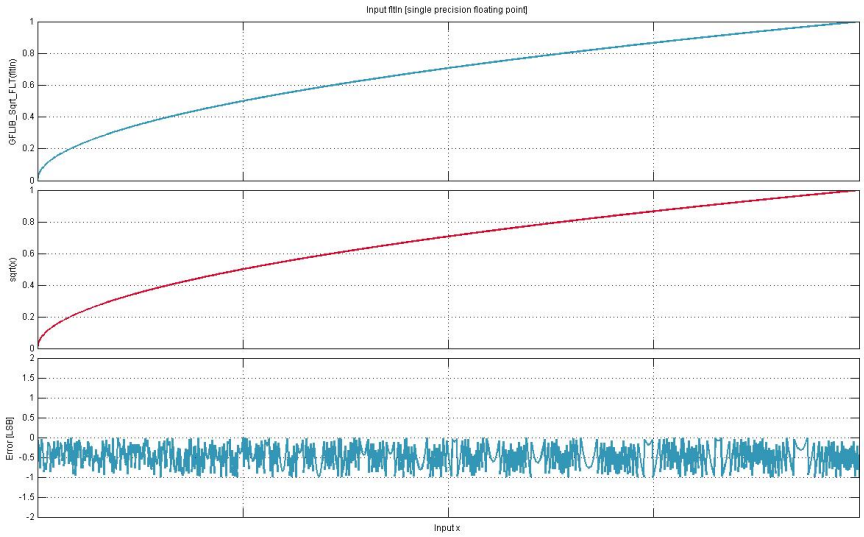

Figure 4-39. real sqrt(x) vs. GFLIB_Sqrt(fltIn)

Note

Input argument is a single precision floating point number limited to the range [0, 2^{128}), if not within this range the computed value is undefined.

4.84.5 Re-entrancy

The function is re-entrant.

4.84.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = 0.5
    fltIn = 0.5;

    // output should be 0.70710678
    fltOut = GFLIB_Sqrt_FLT (fltIn);

    // output should be 0.70710678
    fltOut = GFLIB_Sqrt (fltIn,Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################
```

```
    // output should be 0.70710678
    fltOut = GFLIB_Sqrt (fltIn);
}
```

4.85 Function GFLIB_Tan_F32

This function implements polynomial approximation of tangent function.

4.85.1 Declaration

```
tFrac32 GFLIB_Tan_F32(tFrac32 f32In, const GFLIB_TAN_T_F32 *const pParam);
```

4.85.2 Arguments

Table 4-107. GFLIB_Tan_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input argument is a 32-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_TAN_T_F32 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.85.3 Return

The function returns tan(π * f32In) as a fixed point 32-bit number, normalized between [-1, 1).

4.85.4 Description

The GFLIB_Tan_F32 function provides a computational method for calculation of a standard trigonometric *tangent* function tan(x), using the piece-wise polynomial approximation. Function tan(x) takes an angle and returns the ratio of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. Therefore, the tangent function is defined by:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Equation **GFLIB_Tan_Eq1**

Because both sin(x) and cos(x) are defined on interval [- π, π), function tan(x) is equal to zero when sin(x)=0 and is equal to infinity when cos(x)=0. Therefore, the *tangent* function has asymptotes at n* π/2 for n= , , ... The graph of tan(x) is shown in Figure 4-40.

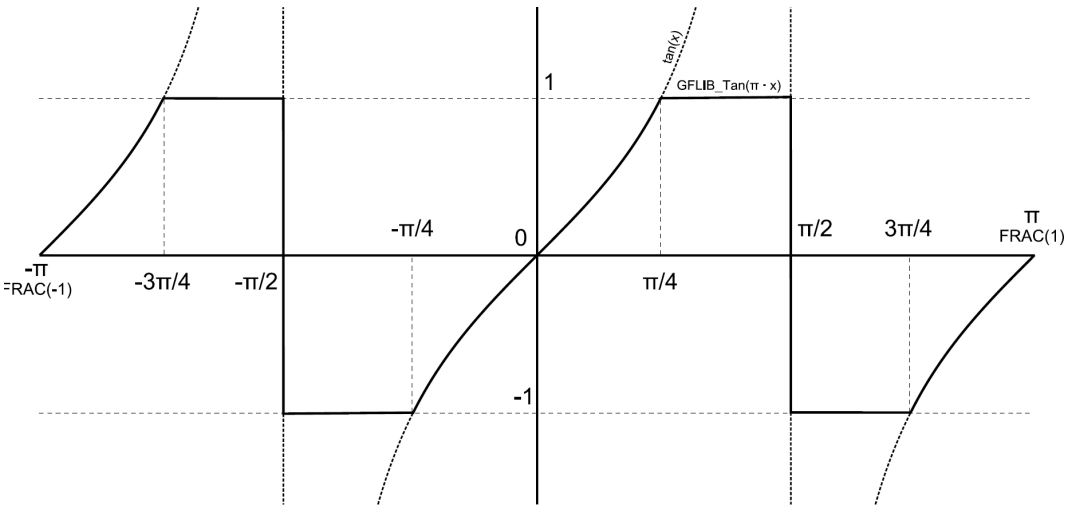

Figure 4-40. Course of the function GFLIB_Tan

The GFLIB_Tan_F32 function is implemented with consideration to fixed point fractional arithmetic, hence all tangent values falling beyond [-1, 1) are truncated to -1 and 1 respectively. This truncation is applied for angles in the ranges [-3 π/4, - π/4) and [π/4, 3 π/4). As can be further seen from Figure 4-40, tangent values are identical for angles in the ranges:

1. [- π/4, 0) and [3 π/4, π)
2. [- π, -3 π/4) and [0, π/4)

Moreover, it can be observed from Figure 4-40 that the course of the tan(x) function output for angles in interval 1. is identical, but with the opposite sign, to output for angles in interval 2. Therefore, the approximation of the *tangent* function over the entire defined range of input angles can be simplified to an approximation for angles in the range [0, π/4), and then, depending on the input angle, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval [0, π/4) is further divided into eight equally spaced sub intervals, and polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients. Moreover, it allows using a polynomial of only the 4th order to achieve an accuracy of less than 0.5LSB (on the upper 16 bits of 32-bit results) across the full range of input angles.

The GFLIB_Tan_F32 function uses fixed point fractional arithmetic, so to cast the fractional value of the input angle f32In [-1, 1) into the correct range [- π, π), the fixed point input angle f32In must be multiplied by π. Then the fixed point fractional implementation of the approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f32Dump = a_1 \bullet f32In^3 + a_2 \bullet f32In^2 + a_3 \bullet f32In + a_4$$

Equation **GFLIB_Tan_Eq2**

$$\tan\left(\pi \bullet f32In\right) = \begin{cases} f32Dump & \text{if } -1 \le f32In < -0.5 \text{ or } 0 \le f32In < 0.5 \\ -f32Dump & \text{if } -0.5 \le f32In < 0 \text{ or } 0.5 \le f32In < 1 \end{cases}$$

Equation **GFLIB_Tan_Eq3**

The division of the [0, π/4) interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-108. Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 4th order was used for the fitting of each respective sub-interval.

Table 4-108. Integer polynomial coefficients for each interval

Interval	a_1	a_2	a_3	a_4
<0, π/32)	688168	1024000	211337216	105498624
< π/32, 2 π/32)	737280	3145728	107732992	318550016
<2 π/32, 3 π/32)	851968	5521408	224055296	537917440
<3 π/32, 4 π/32)	1064960	8364032	237821952	768380928
<4 π/32, 5 π/32)	1392640	12001280	257990656	1015683072
<5 π/32, 6 π/32)	1916928	16900096	286568448	1287151616
<6 π/32, 7 π/32)	2785280	23855104	326795264	1592680448
<7 π/32, 8 π/32)	4292608	34275328	384016384	1946363904

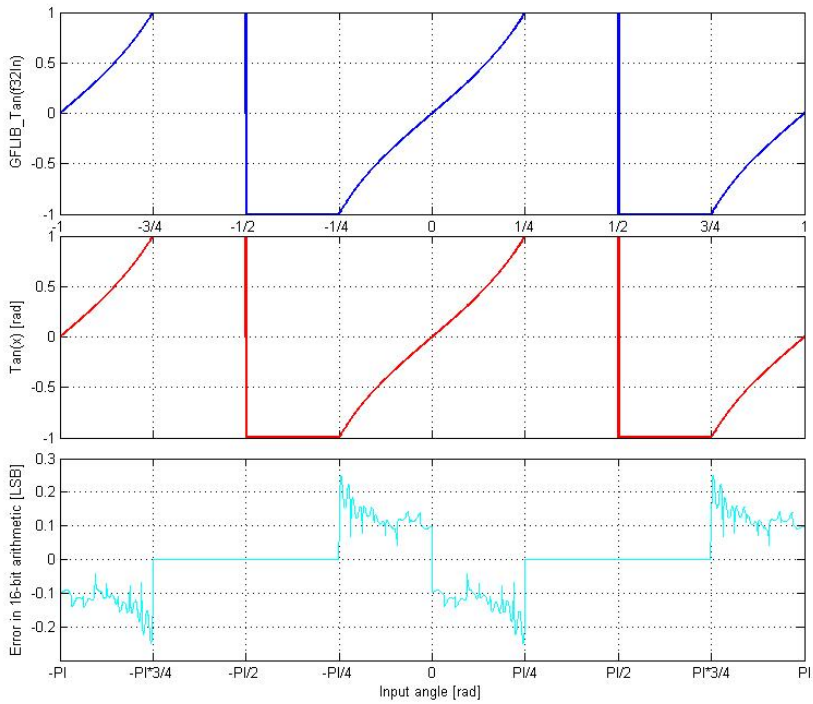

Figure 4-41. tan(x) vs. GFLIB_Tan(f32In)

Figure 4-41 depicts a floating point *tangent* function generated from Matlab and the approximated value of *the* tangent function obtained from GFLIB_Tan_F32, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure. The achieved accuracy with consideration to the 4th order piece-wise polynomial approximation and described fixed point scaling is less than 0.5LSB on the upper 16 bits of the 32-bit result.

Note

The input angle (f32In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Tan_F32(f32In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Tan(f32In, &pParam, F32), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be

replaced with GFLIB_TAN_DEFAULT_F32 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Tan(f32In, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_TAN_DEFAULT_F32 approximation coefficients are used.

4.85.5 Re-entrancy

The function is re-entrant.

4.85.6 Code Example

```
#include "gflib.h"

tFrac32 f32Angle;
tFrac32 f32Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f32Angle  = FRAC32 (0.25);

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan_F32 (f32Angle, GFLIB_TAN_DEFAULT_F32);

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan (f32Angle, GFLIB_TAN_DEFAULT_F32, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x7FFFFFFF = 1
    f32Output = GFLIB_Tan (f32Angle);
}
```

4.86 Function GFLIB_Tan_F16

This function implements polynomial approximation of tangent function.

4.86.1 Declaration

```
tFrac16 GFLIB_Tan_F16(tFrac16 f16In, const GFLIB_TAN_T_F16 *const pParam);
```

4.86.2 Arguments

Table 4-109. GFLIB_Tan_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input argument is a 16-bit number that contains an angle in radians between [- π, π) normalized between [-1, 1).
const GFLIB_TAN_T_F16 *const	pParam	**input**	Pointer to an array of Taylor coefficients.

4.86.3 Return

The function returns tan(π * f16In) as a fixed point 16-bit number, normalized between [-1, 1).

4.86.4 Description

The GFLIB_Tan_F16 function provides a computational method for calculation of a standard trigonometric *tangent* function tan(x), using the piece-wise polynomial approximation. Function tan(x) takes an angle and returns the ratio of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. Therefore, the tangent function is defined by:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Equation **GFLIB_Tan_Eq1**

Because both sin(x) and cos(x) are defined on interval [- π, π), function tan(x) is equal to zero when sin(x)=0 and is equal to infinity when cos(x)=0. Therefore, the *tangent* function has asymptotes at n* π/2 for n= , , ... The graph of tan(x) is shown in Figure 4-42.

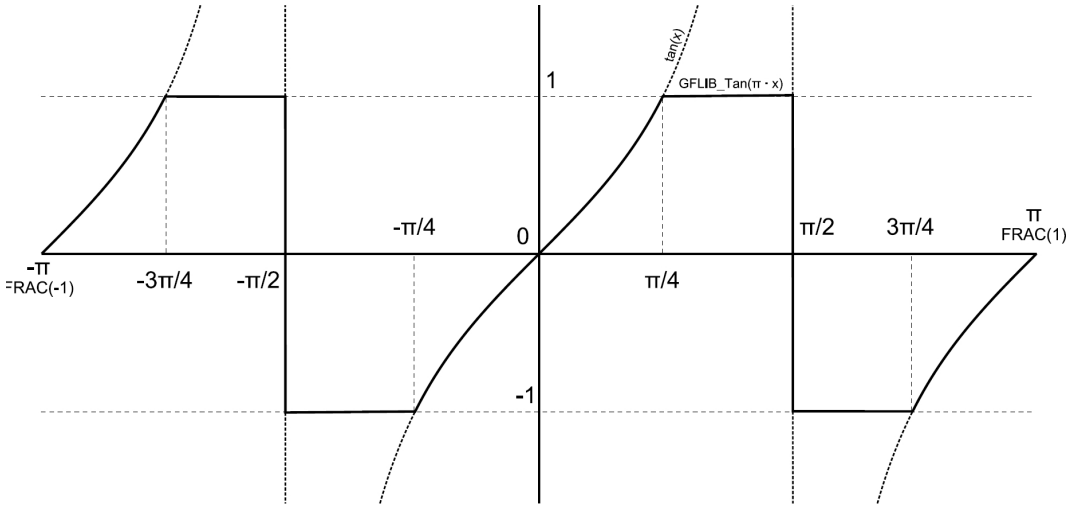

Figure 4-42. Course of the function GFLIB_Tan

The GFLIB_Tan_F16 function is implemented with consideration to fixed point fractional arithmetic, hence all tangent values falling beyond [-1, 1) are truncated to -1 and 1 respectively. This truncation is applied for angles in the ranges [-3 π/4, - π/4) and [π/4, 3 π/4). As can be further seen from Figure 4-42, tangent values are identical for angles in the ranges:

1. [- π/4, 0) and [3 π/4, π)
2. [- π,- 3 π/4) and [0, π/4)

Moreover, it can be observed from Figure 4-42 that the course of the tan(x) function output for angles in interval 1. is identical, but with the opposite sign, to output for angles in interval 2. Therefore, the approximation of the *tangent* function over the entire defined range of input angles can be simplified to an approximation for angles in the range [0, π/4), and then, depending on the input angle, the result will be negated. In order to increase the accuracy of approximation without the need for a higher order polynomial, the interval [0, π/4) is further divided into eight equally spaced sub intervals, and polynomial approximation is done for each interval respectively. Such a division results in eight sets of polynomial coefficients.

The GFLIB_Tan_F16 function uses fixed point fractional arithmetic, so to cast the fractional value of the input angle f16In [-1, 1) into the correct range [- π, π), the fixed point input angle f16In must be multiplied by π. Then the fixed point fractional implementation of the approximation polynomial, used for calculation of each sub sector, is defined as follows:

$$f16Dump = a_1 \bullet f16In^3 + a_2 \bullet f16In^2 + a_3 \bullet f16In + a_4$$

Equation **GFLIB_Tan_Eq2**

$$\tan\left(\pi \bullet f16In\right) = \begin{cases} f16Dump & \text{if } -1 \le f16In < -0.5 \text{ or } 0 \le f16In < 0.5 \\ -f16Dump & \text{if } -0.5 \le f16In < 0 \text{ or } 0.5 \le f16In < 1 \end{cases}$$

Equation **GFLIB_Tan_Eq3**

The division of the [0, π/4) interval into eight sub-intervals, with polynomial coefficients calculated for each sub-interval, is noted in Table 4-110. Polynomial coefficients were obtained using the Matlab fitting function, where a polynomial of the 4th order was used for the fitting of each respective sub-interval.

Table 4-110. Integer polynomial coefficients for each interval

Interval	a_1	a_2	a_3	a_4
<0, π/32)	11	102416000	3225	1610
< π/32, 2 π/32)	11	48	3288	4861
<2 π/32, 3 π/32)	13	84	3419	8208
<3 π/32, 4 π/32)	16	128	3629	11725
<4 π/32, 5 π/32)	21	183	3937	15498
<5 π/32, 6 π/32)	29	258	4373	19640
<6 π/32, 7 π/32)	43	364	4987	24302
<7 π/32, 8 π/32)	66	523	5860	29699

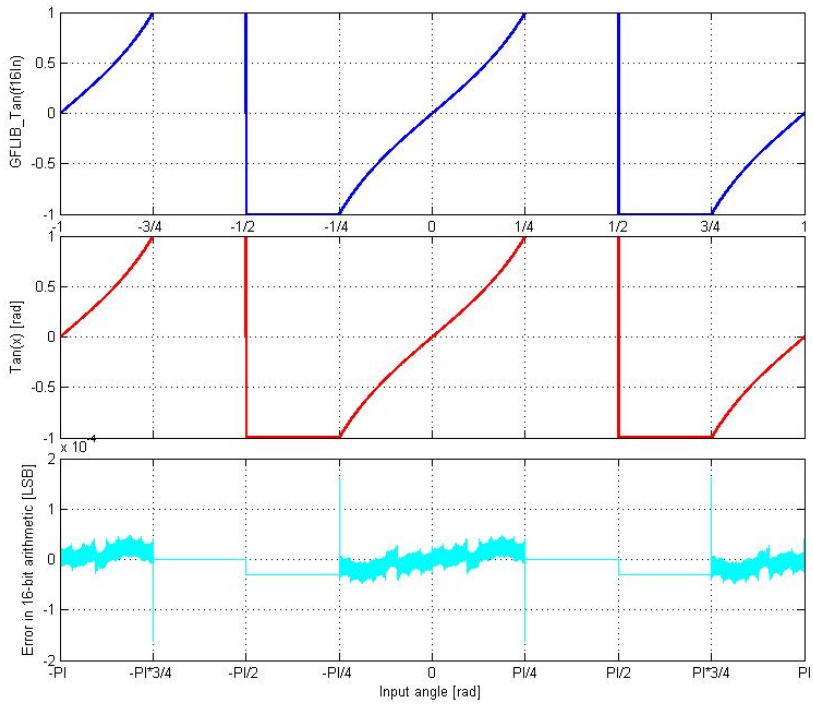

Figure 4-43. tan(x) vs. GFLIB_Tan(f16In)

Figure 4-43 depicts a floating point *tangent* function generated from Matlab and the approximated value of *the* tangent function obtained from GFLIB_Tan_F16, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

Note

The input angle (f16In) is normalized into the range [-1, 1). The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Tan_F16(f16In, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F16 symbol. The &pParam parameter is , mandatory.
- With additional implementation parameter (i.e. GFLIB_Tan(f16In, &pParam, F16), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_F16 symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Tan(f16In, &pParam), where the &pParam is

pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_TAN_DEFAULT_F16 approximation coefficients are used.

4.86.5 Re-entrancy

The function is re-entrant.

4.86.6 Code Example

```
#include "gflib.h"

tFrac16 f16Angle;
tFrac16 f16Output;

void main(void)
{
    // input angle = 0.25 => pi/4
    f16Angle  = FRAC16 (0.25);

    // output should be 0x7FFF = 1
    f16Output = GFLIB_Tan_F16 (f16Angle, GFLIB_TAN_DEFAULT_F16);

    // output should be 0x7FFF = 1
    f16Output = GFLIB_Tan (f16Angle, GFLIB_TAN_DEFAULT_F16, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x7FFF = 1
    f16Output = GFLIB_Tan (f16Angle);
}
```

4.87 Function GFLIB_Tan_FLT

This function implements polynomial approximation of tangent function.

4.87.1 Declaration

```
tFloat GFLIB_Tan_FLT(tFloat fltIn, const GFLIB_TAN_T_FLT *const pParam);
```

4.87.2 Arguments

Table 4-111. GFLIB_Tan_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input argument is a single precision floating point number that contains an angle in radians between (- π, π).
const GFLIB_TAN_T_FLT *const	pParam	**input**	Pointer to an array of approximation coefficients.

4.87.3 Return

The function returns tan(fltIn) as a single precision floating point number.

4.87.4 Description

The GFLIB_Tan_FLT function provides a computational method for the calculation of a standard trigonometric *tangent* function tan(x), using the rational polynomial approximation. Function tan(x) takes an angle and returns the ratio of two sides of a right-angled triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. Therefore, the tangent function is defined by:

$$\tan(x) = \frac{\sin(x)}{\cos(x)}$$

Equation **GFLIB_Tan_Eq1**

Because both sin(x) and cos(x) are defined in interval [- π, π), function tan(x) is equal to zero when sin(x)=0 and is equal to infinity when cos(x)=0. Therefore, the *tangent* function has asymptotes at n* π/2 for n= , , ... The graph of tan(x) is shown in Figure 4-44.

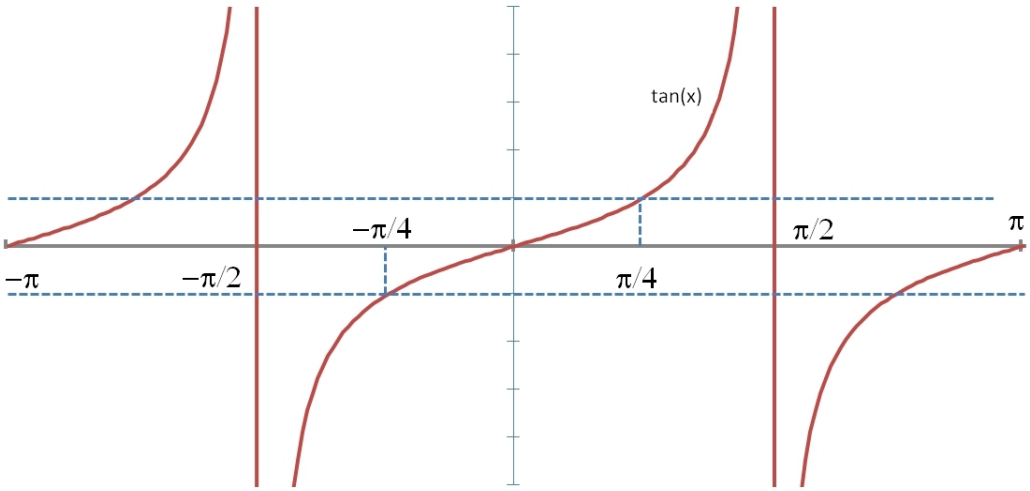

Figure 4-44. Course of the function GFLIB_Tan

As can be seen from Figure 4-44, tangent values are identical for angles in the ranges:

1. [- π/2, 0) and [π/2, π)
2. (- π, - π/2) and [0, π/2)

$$\tan(x) = \frac{1}{\tan\left(\frac{\pi}{2} - x\right)} \quad \text{for} \quad \frac{\pi}{4} < x < \frac{\pi}{2}$$

Equation **GFLIB_Tan_Eq2**

As the approximation reaches the best precision in range [- π/4, π/4), the range reduction is applied to whole interval dividing it into four sub-intervals also using the GFLIB_Tan_Eq3 equation:

$$\tan(-x) = -\tan(x)$$

Equation **GFLIB_Tan_Eq3**

$$\tan(\text{fltIn}) = \begin{cases} \tan(\text{fltIn}) & \text{if} \quad 0 \le \text{fltIn} < \frac{\pi}{4} \\ \frac{1}{\tan\left(\frac{\pi}{2} - \text{fltIn}\right)} & \text{if} \quad \frac{\pi}{4} \le \text{fltIn} < \frac{\pi}{2} \\ -\frac{1}{\tan\left(\text{fltIn} - \frac{\pi}{2}\right)} & \text{if} \quad \frac{\pi}{2} \le \text{fltIn} < \frac{3\pi}{4} \\ -\tan(\pi - \text{fltIn}) & \text{if} \quad \frac{3\pi}{4} \le \text{fltIn} \le \pi \end{cases}$$

Equation **GFLIB_Tan_Eq4**

Finally due to central symmetry of the tan function around the 3 π/4 point, the GFLIB_Tan_Eq4 is transformed to GFLIB_Tan_Eq5 equation:

$$\tan\left(\text{fltIn}\right) = \begin{cases} \tan\left(\text{fltIn}\right) & \text{if} \quad 0 \le \text{fltIn} < \frac{\pi}{4} \\ \frac{1}{\tan\left(\text{fltIn} - \frac{\pi}{2}\right)} & \text{if} \quad \frac{\pi}{4} \le \text{fltIn} < \frac{3\pi}{4} \\ -\tan\left(\pi - \text{fltIn}\right) & \text{if} \quad \frac{3\pi}{4} \le \text{fltIn} \le \pi \end{cases}$$

Equation **GFLIB_Tan_Eq5**

The rational polynomial approximation used actually approximates the tan(x * π/4), thus the input value has to be multiplied by the 4/ π coefficient.

Finally, the sign is added to the computed value in interval (- π, 0). To keep the approximation error within range, the output value is limited to 1000 (defined by the GFLIB_TAN_LIMIT_FLT constant) and dividing by zero is handled by a special case. The floating point optimized approximation coefficients used for GFLIB_Tan_FLT calculation are noted in Table 4-112.

Table 4-112. Approximation polynomial coefficients

a_0	a_1	a_2	a_3
211.8493696641210	-12.5288887278448	269.7350131214120	-71.4145309347748

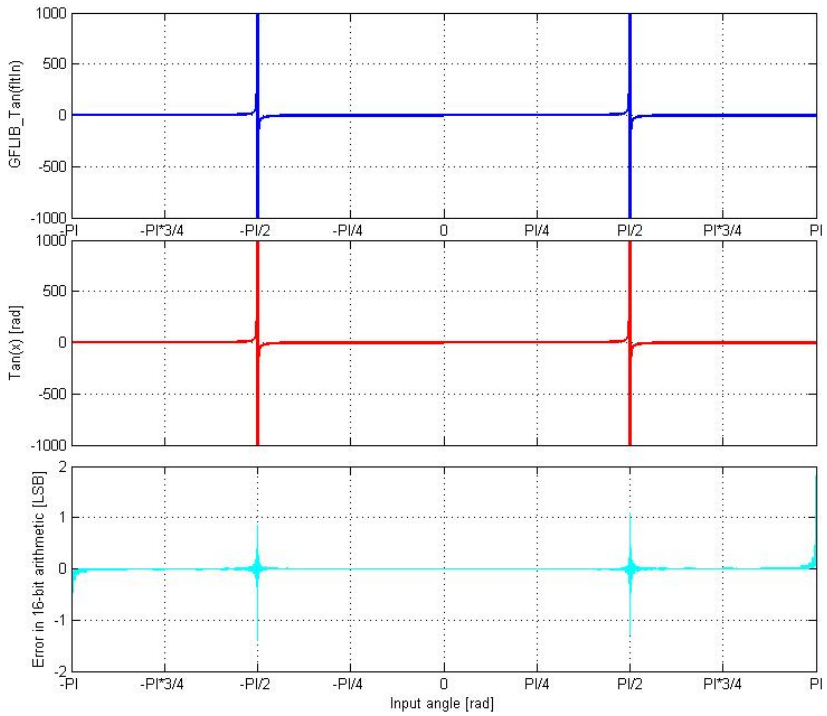

Figure 4-45. tan(x) vs. GFLIB_Tan(fltIn)

Figure 4-45 depicts a floating point *tangent* function generated from Matlab and the approximated value of the *tangent* function obtained from GFLIB_Tan_FLT, plus their difference. The course of calculation accuracy as a function of the input angle can be observed from this figure.

Note

The input angle (fltIn) is in single precision floating point format considering the input angle directly in radians. The approximation precision is guaranteed for interval -(π- π/1024) < fltIn < (π- π/1024). Outside this interval, the function returns the tangent of the input angle, however, the approximation error might be outside the range. The function call is slightly different from common approach in the library set. The function can be called in four different ways:

- With implementation postfix (i.e. GFLIB_Tan_FLT(fltIn, &pParam)), where the &pParam is pointer to approximation coefficients. In case the default approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With additional implementation parameter (i.e. GFLIB_Tan(fltIn, &pParam, FLT), where the &pParam is pointer to approximation coefficients. In case the default

approximation coefficients are used, the &pParam must be replaced with GFLIB_TAN_DEFAULT_FLT symbol. The &pParam parameter is mandatory.
- With preselected default implementation (i.e. GFLIB_Tan(fltIn, &pParam), where the &pParam is pointer to approximation coefficients. The &pParam parameter is optional and in case it is not used, the default GFLIB_TAN_DEFAULT_FLT approximation coefficients are used.

4.87.5 Re-entrancy

The function is re-entrant.

4.87.6 Code Example

```
#include "gflib.h"

tFloat fltAngle;
tFloat fltOutput;

void main(void)
{
    // input angle = pi/4
    fltAngle  = 0.78539816;

    // output should be 1
    fltOutput = GFLIB_Tan_FLT (fltAngle, GFLIB_TAN_DEFAULT_FLT);

    // output should be 1
    fltOutput = GFLIB_Tan (fltAngle, GFLIB_TAN_DEFAULT_FLT, Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 1
    fltOutput = GFLIB_Tan (fltAngle);
}
```

4.88 Function GFLIB_UpperLimit_F32

This function tests whether the input value is below the upper limit.

4.88.1 Declaration

```
tFrac32 GFLIB_UpperLimit_F32(tFrac32 f32In, const GFLIB_UPPERLIMIT_T_F32 *const pParam);
```

4.88.2 Arguments

Table 4-113. GFLIB_UpperLimit_F32 arguments

Type	Name	Direction	Description
tFrac32	f32In	**input**	Input value.
const GFLIB_UPPERLIMIT_T_F32 *const	pParam	**input**	Pointer to the limits structure.

4.88.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

4.88.4 Description

The GFLIB_UpperLimit function tests whether the input value is below the upper limit. If so, the input value will be returned. Otherwise, if the input value is above the upper limit, the upper limit will be returned.

The upper limit f32UpperLimit can be found in the parameters structure, supplied to the function as a pointer pParam.

4.88.5 Re-entrancy

The function is re-entrant.

4.88.6 Code Example

```
#include "gflib.h"

tFrac32 f32In;
tFrac32 f32Out;
GFLIB_UPPERLIMIT_T_F32 f32trMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_F32;
```

```
void main(void)
{
    // upper limit
    f32trMyUpperLimit.f32UpperLimit = FRAC32 (0.5);
    // input value = 0.75
    f32In = FRAC32 (0.75);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit_F32 (f32In,&f32trMyUpperLimit);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit (f32In,&f32trMyUpperLimit,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = GFLIB_UpperLimit (f32In,&f32trMyUpperLimit);
}
```

4.89 Function GFLIB_UpperLimit_F16

This function tests whether the input value is below the upper limit.

4.89.1 Declaration

```
tFrac16 GFLIB_UpperLimit_F16(tFrac16 f16In, const GFLIB_UPPERLIMIT_T_F16 *const pParam);
```

4.89.2 Arguments

Table 4-114. GFLIB_UpperLimit_F16 arguments

Type	Name	Direction	Description
tFrac16	f16In	**input**	Input value.
const GFLIB_UPPERLIMIT_T_F16 *const	pParam	**input**	Pointer to the limits structure.

4.89.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

4.89.4 Description

The GFLIB_UpperLimit function tests whether the input value is below the upper limit. If so, the input value will be returned. Otherwise, if the input value is above the upper limit, the upper limit will be returned.

The upper limit f16UpperLimit can be found in the parameters structure, supplied to the function as a pointer pParam.

4.89.5 Re-entrancy

The function is re-entrant.

4.89.6 Code Example

```
#include "gflib.h"

tFrac16 f16In;
tFrac16 f16Out;
GFLIB_UPPERLIMIT_T_F16 f16trMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_F16;

void main(void)
{
    // upper limit
    f16trMyUpperLimit.f16UpperLimit = FRAC16 (0.5);
    // input value = 0.75
    f16In = FRAC16 (0.75);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit_F16 (f16In,&f16trMyUpperLimit);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit (f16In,&f16trMyUpperLimit,Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = GFLIB_UpperLimit (f16In,&f16trMyUpperLimit);
}
```

4.90 Function GFLIB_UpperLimit_FLT

This function tests whether the input value is below the upper limit.

4.90.1 Declaration

```
tFloat GFLIB_UpperLimit_FLT(tFloat fltIn, const GFLIB_UPPERLIMIT_T_FLT *const pParam);
```

4.90.2 Arguments

Table 4-115. GFLIB_UpperLimit_FLT arguments

Type	Name	Direction	Description
tFloat	fltIn	**input**	Input value.
const GFLIB_UPPERLIMIT_T_FLT *const	pParam	**input**	Pointer to the limits structure.

4.90.3 Return

The input value in case the input value is below the limit, or the upper limit if the input value is above the limit.

4.90.4 Description

The GFLIB_UpperLimit function tests whether the input value is below the upper limit. If so, the input value will be returned. Otherwise, if the input value is above the upper limit, the upper limit will be returned.

The upper limit fltUpperLimit can be found in the parameters structure, supplied to the function as a pointer pParam.

4.90.5 Re-entrancy

The function is re-entrant.

4.90.6 Code Example

```
#include "gflib.h"

tFloat fltIn;
tFloat fltOut;
GFLIB_UPPERLIMIT_T_FLT flttrMyUpperLimit = GFLIB_UPPERLIMIT_DEFAULT_FLT;
```

```
void main(void)
{
    // upper limit
    flttrMyUpperLimit.fltUpperLimit = 0.5;
    // input value = 0.75
    fltIn = 0.75;

    // output should be 0.5
    fltOut = GFLIB_UpperLimit_FLT (fltIn,&flttrMyUpperLimit);

    // output should be 0.5
    fltOut = GFLIB_UpperLimit (fltIn,&flttrMyUpperLimit,Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be 0.5
    fltOut = GFLIB_UpperLimit (fltIn,&flttrMyUpperLimit);
}
```

4.91 Function GFLIB_VectorLimit_F32

This function limits the magnitude of the input vector.

4.91.1 Declaration

```
tBool GFLIB_VectorLimit_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn,
const GFLIB_VECTORLIMIT_T_F32 *const pParam);
```

4.91.2 Arguments

Table 4-116. GFLIB_VectorLimit_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure of the input vector.
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT_T_F32 *const	pParam	**input**	Pointer to the parameters structure.

4.91.3 Return

The function will return true "TRUE" if the input vector is being limited, or false "FALSE" otherwise.

4.91.4 Description

The GFLIB_VectorLimit function limits the magnitude of the input vector, keeping its direction unchanged. Limitation is performed as follows:

$$y_{out} = \begin{cases} \frac{y_{in}}{\sqrt{x_{in}^2 + y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2 + y_{in}^2} > L \\ y_{in} & \text{if } \sqrt{x_{in}^2 + y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq1**

$$x_{out} = \begin{cases} \frac{x_{in}}{\sqrt{x_{in}^2 + y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2 + y_{in}^2} > L \\ x_{in} & \text{if } \sqrt{x_{in}^2 + y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq2**

Where:

- x_{in}, y_{in} and x_{out}, y_{out} are the co-ordinates of the input and output vector, respectively
- L is the maximum magnitude of the vector

The input vector co-ordinates are defined by the structure pointed to by the pIn parameter, and the output vector co-ordinates be found in the structure pointed by the pOut parameter. The maximum vector magnitude is defined in the parameters structure pointed to by the pParam function parameter.

A graphical interpretation of the function can be seen in the figure below.

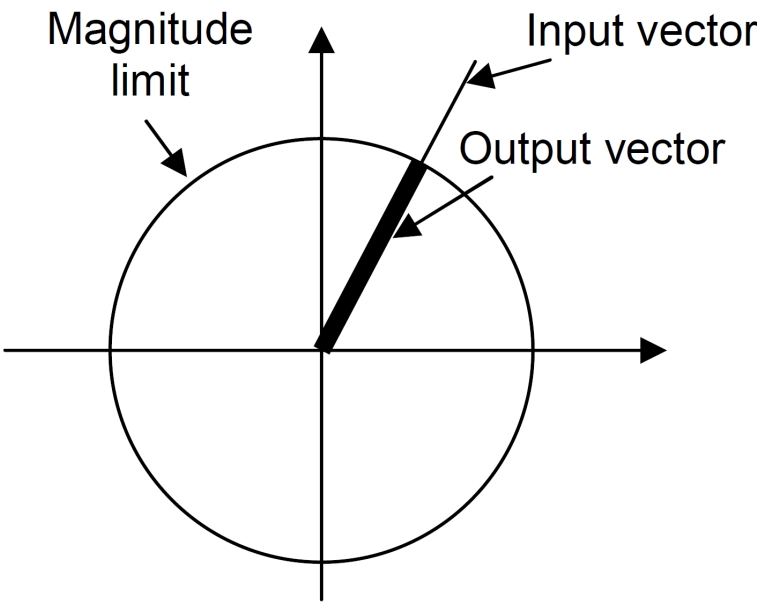

Figure 4-46. Graphical interpretation of the GFLIB_VectorLimit function.

If an actual limitation occurs, the function will return the logical true (TRUE), otherwise the logical false will be returned (FALSE).

For computational reasons, the output vector will be computed as zero if the input vector magnitude is lower than 2^{-15}, regardless of the set maximum magnitude of the input vector. The function returns the logical true (TRUE) in this case.

Also, the 16 least significant bits of the maximum vector magnitude in the parameters structure, the pParam->f32Limit, are ignored. This means that the defined magnitude must be equal to or greater than 2^{-15}, otherwise the result is undefined.

Note

The function calls the MCLIB square root routine GFLIB_Sqrt.

CAUTION

The maximum vector magnitude in the parameters structure, the pParam->f32Limit, must be positive and equal to or greater than 2^{-15}, otherwise the result is undefined. The function does not check for the valid range of pParam->f32Limit.

4.91.5 Re-entrancy

The function is re-entrant.

4.91.6 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F32 f32pIn;
SWLIBS_2Syst_F32 f32pOut;
GFLIB_VECTORLIMIT_T_F32 f32trMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_F32;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    f32trMyVectorLimit.f32Limit = FRAC32 (0.25);
    // input vector
    f32pIn.f32Arg1 = FRAC32 (0.25);
    f32pIn.f32Arg2 = FRAC32 (0.25);

    // output should be: bLim = TRUE;
    //                   f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    //                   f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit_F32 (&f32pOut,&fp32In,&f32trMyVectorLimit);

    // output should be: bLim = TRUE;
    //                   f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    //                   f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit (&f32pOut,&f32pIn,&f32trMyVectorLimit,Define
F32);

    // ##################################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##################################################################

    // output should be: bLim = TRUE;
    //                   f32pOut.f32Arg1 = 0x16A08000 ~ FRAC32(0.17677)
    //                   f32pOut.f32Arg2 = 0x16A08000 ~ FRAC32(0.17677)
    bLim = GFLIB_VectorLimit (&f32pOut,&f32pIn,&f32trMyVectorLimit);
}
```

4.92 Function GFLIB_VectorLimit_F16

This function limits the magnitude of the input vector.

4.92.1 Declaration

```
tBool GFLIB_VectorLimit_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn,
const GFLIB_VECTORLIMIT_T_F16 *const pParam);
```

4.92.2 Arguments

Table 4-117. GFLIB_VectorLimit_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure of the input vector.
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT_T_F16 *const	pParam	**input**	Pointer to the parameters structure.

4.92.3 Return

The function will return true "TRUE" if the input vector is being limited, or false "FALSE" otherwise.

4.92.4 Description

The GFLIB_VectorLimit function limits the magnitude of the input vector, keeping its direction unchanged. Limitation is performed as follows:

$$y_{out} = \begin{cases} \frac{y_{in}}{\sqrt{x_{in}^2+y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2+y_{in}^2} > L \\ y_{in} & \text{if } \sqrt{x_{in}^2+y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq1**

$$x_{out} = \begin{cases} \frac{x_{in}}{\sqrt{x_{in}^2+y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2+y_{in}^2} > L \\ x_{in} & \text{if } \sqrt{x_{in}^2+y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq2**

Where:

- x_{in}, y_{in} and x_{out}, y_{out} are the co-ordinates of the input and output vector, respectively
- L is the maximum magnitude of the vector

The input vector co-ordinates are defined by the structure pointed to by the pIn parameter, and the output vector co-ordinates be found in the structure pointed by the pOut parameter. The maximum vector magnitude is defined in the parameters structure pointed to by the pParam function parameter.

A graphical interpretation of the function can be seen in the figure below.

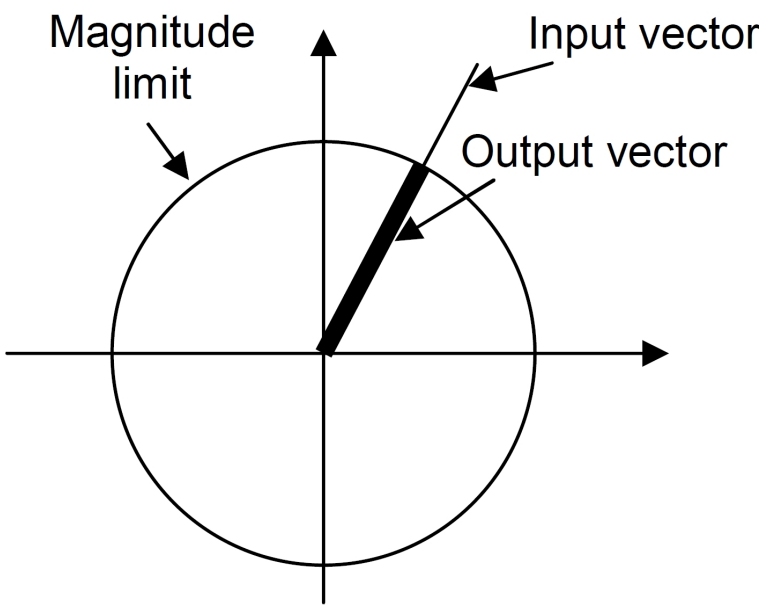

Figure 4-47. Graphical interpretation of the GFLIB_VectorLimit function.

If an actual limitation occurs, the function will return the logical true (TRUE), otherwise the logical false will be returned (FALSE).

Note

The function calls the MCLIB square root routine GFLIB_Sqrt.

CAUTION

The maximum vector magnitude in the parameters structure, the pParam->f16Limit, must be positive and equal to or greater than "0", otherwise the result is undefined. The function does not check for the valid range of the parameter pParam->f16Limit.

4.92.5 Re-entrancy

The function is re-entrant.

4.92.6 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_F16 f16pIn;
SWLIBS_2Syst_F16 f16pOut;
GFLIB_VECTORLIMIT_T_F16 f16trMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_F16;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    f16trMyVectorLimit.f16Limit = FRAC16 (0.25);
    // input vector
    f16pIn.f16Arg1 = FRAC16 (0.25);
    f16pIn.f16Arg2 = FRAC16 (0.25);

    // output should be: bLim = TRUE;
    //                   f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    //                   f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit_F16 (&f16pOut,&fp16In,&f16trMyVectorLimit);

    // output should be: bLim = TRUE;
    //                   f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    //                   f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit (&f16pOut,&f16pIn,&f16trMyVectorLimit,Define
F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be: bLim = TRUE;
    //                   f16pOut.f16Arg1 = 0x16A0 ~ FRAC16(0.17677)
    //                   f16pOut.f16Arg2 = 0x16A0 ~ FRAC16(0.17677)
    bLim = GFLIB_VectorLimit (&f16pOut,&f16pIn,&f16trMyVectorLimit);
}
```

4.93 Function GFLIB_VectorLimit_FLT

This function limits the magnitude of the input vector.

4.93.1 Declaration

```
tBool GFLIB_VectorLimit_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pIn,
const GFLIB_VECTORLIMIT_T_FLT *const pParam);
```

4.93.2 Arguments

Table 4-118. GFLIB_VectorLimit_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure of the input vector.
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure of the limited output vector.
const GFLIB_VECTORLIMIT_T_FLT *const	pParam	**input**	Pointer to the parameters structure.

4.93.3 Return

The function will return true "TRUE" if the input vector is being limited, or false "FALSE" otherwise.

4.93.4 Description

The GFLIB_VectorLimit function limits the magnitude of the input vector, keeping its direction unchanged. The maximum magnitude of the input vector in the following equation is defined by the letter "L". Limitation is performed as follows:

$$y_{out} = \begin{cases} \frac{y_{in}}{\sqrt{x_{in}^2+y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2+y_{in}^2} > L \\ y_{in} & \text{if } \sqrt{x_{in}^2+y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq1**

$$x_{out} = \begin{cases} \frac{x_{in}}{\sqrt{x_{in}^2+y_{in}^2}} \bullet L & \text{if } \sqrt{x_{in}^2+y_{in}^2} > L \\ x_{in} & \text{if } \sqrt{x_{in}^2+y_{in}^2} \le L \end{cases}$$

Equation **GFLIB_VectorLimit_Eq2**

Where:

- x_{in}, y_{in} and x_{out}, y_{out} are the co-ordinates of the input and output vector, respectively
- L is the maximum magnitude of the vector

The input vector co-ordinates are defined by the structure pointed to by the pIn parameter, and the output vector co-ordinates shall be found in the structure pointed to by the pOut parameter. The maximum vector magnitude fltLimit is defined in the parameters structure pointed to by the pParam function parameter.

A graphical interpretation of the function can be seen in the figure below.

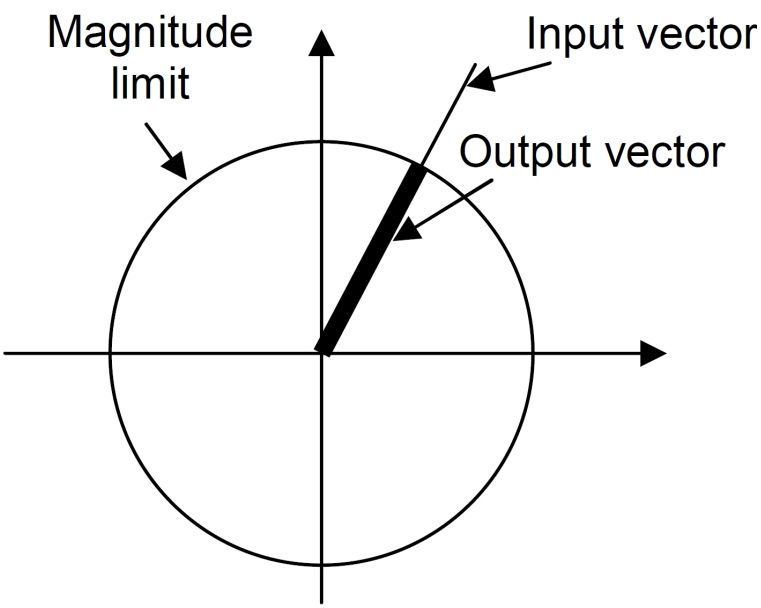

Figure 4-48. Graphical interpretation of the GFLIB_VectorLimit function.

If an actual limitation occurs, the function will return the logical true "TRUE", otherwise the logical false "FALSE" will be returned.

Note

The function calls the MCLIB square root routine GFLIB_Sqrt.

CAUTION

The maximum vector magnitude in the parameters structure, the pParam->fltLimit, must be positive and equal to or greater than "0", otherwise the result is undefined. The function does not check for the valid range of the parameter pParam->fltLimit.

4.93.5 Re-entrancy

The function is re-entrant.

4.93.6 Code Example

```
#include "gflib.h"

SWLIBS_2Syst_FLT fltpIn;
SWLIBS_2Syst_FLT fltpOut;
GFLIB_VECTORLIMIT_T_FLT flttrMyVectorLimit = GFLIB_VECTORLIMIT_DEFAULT_FLT;
tBool bLim;

void main(void)
{
    // desired magnitude of the input vector
    flttrMyVectorLimit.fltLimit = 0.25;
    // input vector
    fltpIn.fltArg1 = 0.25;
    fltpIn.fltArg2 = 0.25;

    // output should be: bLim = TRUE;
    //                   fltpOut.fltArg1 = 0.17677
    //                   fltpOut.fltArg2 = 0.17677
    bLim = GFLIB_VectorLimit_FLT (&fltpOut,&fpltIn,&flttrMyVectorLimit);

    // output should be: bLim = TRUE;
    //                   fltpOut.fltArg1 = 0.17677
    //                   fltpOut.fltArg2 = 0.17677
    bLim = GFLIB_VectorLimit (&fltpOut,&fltpIn,&flttrMyVectorLimit,Define
FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be: bLim = TRUE;
    //                   fltpOut.fltArg1 = 0.17677
    //                   fltpOut.fltArg2 = 0.17677
    bLim = GFLIB_VectorLimit (&fltpOut,&fltpIn,&flttrMyVectorLimit);
}
```

4.94 Function GMCLIB_Clark_F32

The function implements the Clarke transformation.

4.94.1 Declaration

```
void GMCLIB_Clark_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_3Syst_F32 *const pIn);
```

4.94.2 Arguments

Table 4-119. GMCLIB_Clark_F32 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β). Arguments of the structure contain fixed point 32-bit values.

4.94.3 Return

Function returns no value.

4.94.4 Description

The Clarke Transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α- β) orthogonal coordinate system, according to the following equations:

$$i_{\alpha} = i_{f32A}$$

Equation **GMCLIB_Clark_Eq1**

$$i_{\beta} = \left(i_{f32B} - i_{f32C}\right) \bullet \frac{1}{\sqrt{3}}$$

Equation **GMCLIB_Clark_Eq2**

where it is assumed that the axis f32A (axis of the first phase) and the axis α are in the same direction.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.94.5 Re-entrancy

The function is re-entrant.

4.94.6 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F32 f32trAbc;
SWLIBS_2Syst_F32 f32trAlBe;

void main(void)
{
    // input value
    f32trAbc.f32Arg1 = FRAC32 (0.707106781);  // input phase A ~ sin(45) ~
0.707106781
    f32trAbc.f32Arg2 = FRAC32 (0.258819045);  // input phase B ~ sin(45 +
120) ~ 0.258819045
    f32trAbc.f32Arg3 = FRAC32 (-0.965925826); // input phase C ~ sin(45 -
120) ~ -0.965925826

    // output should be f32trAlBe.f32Arg1 ~ alpha = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 ~ beta  = 0x5A827999 ~
FRAC32(0.707106781)
    GMCLIB_Clark_F32 (&f32trAlBe,&f32trAbc);

    // output should be f32trAlBe.f32Arg1 ~ alpha = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 ~ beta  = 0x5A827999 ~
FRAC32(0.707106781)
    GMCLIB_Clark (&f32trAlBe,&f32trAbc,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be f32trAlBe.f32Arg1 ~ alpha = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAlBe.f32Arg2 ~ beta  = 0x5A827999 ~
FRAC32(0.707106781)
    GMCLIB_Clark (&f32trAlBe,&f32trAbc);
}
```

4.95 Function GMCLIB_Clark_F16

The function implements the Clarke transformation.

4.95.1 Declaration

```
void GMCLIB_Clark_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_3Syst_F16 *const pIn);
```

4.95.2 Arguments

Table 4-120. GMCLIB_Clark_F16 arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β). Arguments of the structure contain fixed point 16-bit values.

4.95.3 Return

Function returns no value.

4.95.4 Description

The Clarke Transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α- β) orthogonal coordinate system, according to the following equations:

$$i_{\alpha} = i_{f16A}$$

Equation **GMCLIB_Clark_Eq1**

$$i_{\beta} = \left(i_{f16B} - i_{f16C}\right) \bullet \frac{1}{\sqrt{3}}$$

Equation **GMCLIB_Clark_Eq2**

where it is assumed that the axis f16A (axis of the first phase) and the axis α are in the same direction.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.95.5 Re-entrancy

The function is re-entrant.

4.95.6 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_F16 f16trAbc;
SWLIBS_2Syst_F16 f16trAlBe;

void main(void)
{
    // input value
    f16trAbc.f16Arg1 = FRAC16 (0.707106781);  // input phase A ~ sin(45) ~
0.707106781
    f16trAbc.f16Arg2 = FRAC16 (0.258819045);  // input phase B ~ sin(45 +
120) ~ 0.258819045
    f16trAbc.f16Arg3 = FRAC16 (-0.965925826); // input phase C ~ sin(45 -
120) ~ -0.965925826

    // output should be f16trAlBe.f16Arg1 ~ alpha = 0x5A82 ~
FRAC16(0.707106781)
    // output should be f16trAlBe.f16Arg2 ~ beta  = 0x5A82 ~
FRAC16(0.707106781)
    GMCLIB_Clark_F16 (&f16trAlBe,&f16trAbc);

    // output should be f16trAlBe.f16Arg1 ~ alpha = 0x5A82 ~
FRAC16(0.707106781)
    // output should be f16trAlBe.f16Arg2 ~ beta  = 0x5A82 ~
FRAC16(0.707106781)
    GMCLIB_Clark (&f16trAlBe,&f16trAbc,Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be f16trAlBe.f16Arg1 ~ alpha = 0x5A82 ~
FRAC16(0.707106781)
    // output should be f16trAlBe.f16Arg2 ~ beta  = 0x5A82 ~
FRAC16(0.707106781)
    GMCLIB_Clark (&f16trAlBe,&f16trAbc);
}
```

4.96 Function GMCLIB_Clark_FLT

The function implements the Clarke transformation.

4.96.1 Declaration

```
void GMCLIB_Clark_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_3Syst_FLT *const pIn);
```

4.96.2 Arguments

Table 4-121. GMCLIB_Clark_FLT arguments

Type	Name	Direction	Description
const SWLIBS_3Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the three-phase stationary system (fltA-fltB-fltC). Arguments of the structure contain single precision floating point values.
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β). Arguments of the structure contain single precision floating point values.

4.96.3 Return

Function returns no value.

4.96.4 Description

The Clarke Transformation is used to transform values from the three-phase (A-B-C) coordinate system to the two-phase (α- β) orthogonal coordinate system, according to the following equations:

$$i_{\alpha} = i_{\mathrm{fltA}}$$

Equation **GMCLIB_Clark_Eq1**

$$i_{\beta} = \left(i_{\mathrm{fltB}} - i_{\mathrm{fltC}}\right) \bullet \frac{1}{\sqrt{3}}$$

Equation **GMCLIB_Clark_Eq2**

where it is assumed that the axis (axis of the first phase) and the axis α are in the same direction.

Note

The inputs and the outputs are in full range single precision floating point format.

4.96.5 Re-entrancy

The function is re-entrant.

4.96.6 Code Example

```
#include "gmclib.h"

SWLIBS_3Syst_FLT fltrAbc;
SWLIBS_2Syst_FLT fltrAlBe;

void main(void)
{
    // input value
    fltrAbc.fltArg1 = 0.707106781;  // input phase A ~ sin(45) ~ 0.707106781
    fltrAbc.fltArg2 = 0.258819045;  // input phase B ~ sin(45 + 120) ~
0.258819045
    fltrAbc.fltArg3 = -0.965925826; // input phase C ~ sin(45 - 120) ~
-0.965925826

    // output should be fltrAlBe.fltArg1 ~ alpha = 0.707106781
    // output should be fltrAlBe.fltArg2 ~ beta  = 0.707106781
    GMCLIB_Clark_FLT (&fltrAlBe,&fltrAbc);

    // output should be fltrAlBe.fltArg1 ~ alpha = 0.707106781
    // output should be fltrAlBe.fltArg2 ~ beta  = 0.707106781
    GMCLIB_Clark (&fltrAlBe,&fltrAbc,Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be fltrAlBe.fltArg1 ~ alpha = 0.707106781
    // output should be fltrAlBe.fltArg2 ~ beta  = 0.707106781
    GMCLIB_Clark (&fltrAlBe,&fltrAbc);
}
```

4.97 Function GMCLIB_ClarkInv_F32

The function implements the inverse Clarke transformation.

4.97.1 Declaration

```
void GMCLIB_ClarkInv_F32(SWLIBS_3Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn);
```

4.97.2 Arguments

Table 4-122. GMCLIB_ClarkInv_F32 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β). Arguments of the structure contain fixed point 32-bit values.
SWLIBS_3Syst_F32 *const	pOut	**output**	Pointer to the structure containing data of the three-phase stationary system (f32A-f32B-f32C). Arguments of the structure contain fixed point 32-bit values.

4.97.3 Return

Function returns no value.

4.97.4 Description

The GMCLIB_ClarkInv function calculates the Inverse Clarke transformation, which is used to transform values from the two-phase (α- β) orthogonal coordinate system to the three-phase (f32A-f32B-f32C) coordinate system, according to these equations:

$$i_{f32A} = i_{\alpha}$$

Equation **GMCLIB_ClarkInv_Eq1**

$$i_{f32B} = -\frac{1}{2} \bullet i_{\alpha} + \frac{\sqrt{3}}{2} \bullet i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq2**

$$i_{f32C} = -\frac{1}{2} \bullet i_{\alpha} - \frac{\sqrt{3}}{2} \bullet i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq3**

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.97.5 Re-entrancy

The function is re-entrant.

4.97.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 f32trAlBe;
SWLIBS_3Syst_F32 f32trAbc;

void main(void)
{
    // input value
    f32trAlBe.f32Arg1 = FRAC32 (0.707106781); // input phase alpha ~ sin(45)
~ 0.707106781
    f32trAlBe.f32Arg2 = FRAC32 (0.707106781); // input phase beta ~ cos(45) ~
0.707106781

    // output should be f32trAbc.f32Arg1 ~ phA = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAbc.f32Arg2 ~ phB = 0x2120FB83 ~
FRAC32(0.258819045)
    // output should be f32trAbc.f32Arg3 ~ phC = 0x845C8AE5 ~
FRAC32(-0.965925826)
    GMCLIB_ClarkInv_F32 (&f32trAbc,&f32trAlBe);

    // output should be f32trAbc.f32Arg1 ~ phA = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAbc.f32Arg2 ~ phB = 0x2120FB83 ~
FRAC32(0.258819045)
    // output should be f32trAbc.f32Arg3 ~ phC = 0x845C8AE5 ~
FRAC32(-0.965925826)
    GMCLIB_ClarkInv (&f32trAbc,&f32trAlBe,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be f32trAbc.f32Arg1 ~ phA = 0x5A827999 ~
FRAC32(0.707106781)
    // output should be f32trAbc.f32Arg2 ~ phB = 0x2120FB83 ~
FRAC32(0.258819045)
    // output should be f32trAbc.f32Arg3 ~ phC = 0x845C8AE5 ~
FRAC32(-0.965925826)
    GMCLIB_ClarkInv (&f32trAbc,&f32trAlBe);
}
```

4.98 Function GMCLIB_ClarkInv_F16

The function implements the inverse Clarke transformation.

4.98.1 Declaration

```
void GMCLIB_ClarkInv_F16(SWLIBS_3Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn);
```

4.98.2 Arguments

Table 4-123. GMCLIB_ClarkInv_F16 arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β). Arguments of the structure contain fixed point 16-bit values.
SWLIBS_3Syst_F16 *const	pOut	**output**	Pointer to the structure containing data of the three-phase stationary system (f16A-f16B-f16C). Arguments of the structure contain fixed point 16-bit values.

4.98.3 Return

Function returns no value.

4.98.4 Description

The GMCLIB_ClarkInv function calculates the Inverse Clarke transformation, which is used to transform values from the two-phase (α- β) orthogonal coordinate system to the three-phase (f16A-f16B-f16C) coordinate system, according to these equations:

$$i_{f16A} = i_{\alpha}$$

Equation **GMCLIB_ClarkInv_Eq1**

$$i_{f16B} = -\frac{1}{2} \bullet i_{\alpha} + \frac{\sqrt{3}}{2} \bullet i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq2**

$$i_{f16C} = -\frac{1}{2} \bullet i_{\alpha} - \frac{\sqrt{3}}{2} \bullet i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq3**

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.98.5 Re-entrancy

The function is re-entrant.

4.98.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 f16trAlBe;
SWLIBS_3Syst_F16 f16trAbc;

void main(void)
{
    // input value
    f16trAlBe.f16Arg1 = FRAC16 (0.707106781); // input phase alpha ~ sin(45)
~ 0.707106781
    f16trAlBe.f16Arg2 = FRAC16 (0.707106781); // input phase beta ~ cos(45) ~
0.707106781

    // output should be f16trAbc.f16Arg1 ~ phA = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 ~ phB = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 ~ phC = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv_F16 (&f16trAbc,&f16trAlBe);

    // output should be f16trAbc.f16Arg1 ~ phA = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 ~ phB = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 ~ phC = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv (&f16trAbc,&f16trAlBe,Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be f16trAbc.f16Arg1 ~ phA = 0x5A82 ~ FRAC16(0.707106781)
    // output should be f16trAbc.f16Arg2 ~ phB = 0x2120 ~ FRAC16(0.258819045)
    // output should be f16trAbc.f16Arg3 ~ phC = 0x845C ~ FRAC16(-0.965925826)
    GMCLIB_ClarkInv (&f16trAbc,&f16trAlBe);
}
```

4.99 Function GMCLIB_ClarkInv_FLT

The function implements the inverse Clarke transformation.

4.99.1 Declaration

```
void GMCLIB_ClarkInv_FLT(SWLIBS_3Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pIn);
```

4.99.2 Arguments

Table 4-124. GMCLIB_ClarkInv_FLT arguments

Type	Name	Direction	Description
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β). Arguments of the structure contain single precision floating point values.
SWLIBS_3Syst_FLT *const	pOut	**output**	Pointer to the structure containing data of the three-phase stationary system (fltA-fltB-fltC). Arguments of the structure contain single precision floating point values.

4.99.3 Return

Function returns no value.

4.99.4 Description

The GMCLIB_ClarkInv function calculates the Inverse Clarke transformation, which is used to transform values from the two-phase (α- β) orthogonal coordinate system to the three-phase (fltA-fltB-fltC) coordinate system, according to these equations:

$$i_{\mathrm{fltA}} = i_{\alpha}$$

Equation **GMCLIB_ClarkInv_Eq1**

$$i_{\mathrm{fltB}} = -\frac{1}{2} \bullet i_{\alpha} + \frac{\sqrt{3}}{2} \bullet i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq2**

$$i_{f16C} = -\frac{1}{2} \bullet i_{\alpha} - \frac{\sqrt{3}}{2} \bullet i_{\beta}$$

Equation **GMCLIB_ClarkInv_Eq3**

Note

The inputs and the outputs are in full range single precision floating point format.

4.99.5 Re-entrancy

The function is re-entrant.

4.99.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT flttrAlBe;
SWLIBS_3Syst_FLT flttrAbc;

void main(void)
{
    // input value
    flttrAlBe.fltArg1 = 0.707106781; // input phase alpha ~ sin(45) ~
0.707106781
    flttrAlBe.fltArg2 = 0.707106781; // input phase beta ~ cos(45) ~
0.707106781

    // output should be flttrAbc.fltArg1 ~ phA = 0.707106781
    // output should be flttrAbc.fltArg2 ~ phB = 0.258819045
    // output should be flttrAbc.fltArg3 ~ phC = -0.965925826
    GMCLIB_ClarkInv_FLT (&flttrAbc,&flttrAlBe);

    // output should be flttrAbc.fltArg1 ~ phA = 0.707106781
    // output should be flttrAbc.fltArg2 ~ phB = 0.258819045
    // output should be flttrAbc.fltArg3 ~ phC = -0.965925826
    GMCLIB_ClarkInv (&flttrAbc,&flttrAlBe,Define FLT);

    // ##############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ##############################################################

    // output should be flttrAbc.fltArg1 ~ phA = 0.707106781
    // output should be flttrAbc.fltArg2 ~ phB = 0.258819045
    // output should be flttrAbc.fltArg3 ~ phC = -0.965925826
    GMCLIB_ClarkInv (&flttrAbc,&flttrAlBe);
}
```

4.100 Function GMCLIB_DecouplingPMSM_F32

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing on-linearity of the field oriented control.

4.100.1 Declaration

```
void GMCLIB_DecouplingPMSM_F32(SWLIBS_2Syst_F32 *const pUdqDec, const SWLIBS_2Syst_F32 *const
pUdq, const SWLIBS_2Syst_F32 *const pIdq, tFrac32 f32AngularVel, const
GMCLIB_DECOUPLINGPMSM_T_F32 *const pParam);
```

4.100.2 Arguments

Table 4-125. GMCLIB_DecouplingPMSM_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pUdqDec	**output**	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_F32 *const	pUdq	**input**	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_F32 *const	pIdq	**input**	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFrac32	f32AngularVel	**input**	Rotor angular velocity in rad/sec, referred to as ($_{ef}$) in the detailed section of the documentation.
const GMCLIB_DECOUPLINGPMSM_T_F32 *const	pParam	**input**	Pointer to the structure containing k_{df} and k_{qf} coefficients (see the detailed section of the documentation) and scale parameters (k_{d_shift}) and (k_{q_shift}).

4.100.3 Return

Function returns no value.

4.100.4 Description

The quadrature phase model of a PMSM motor, in a synchronous reference frame, is very popular for field oriented control structures because both controllable quantities, current and voltage, are DC values. This allows employing only simple controllers to force the machine currents into the defined states.

The voltage equations of this model can be obtained by transforming the motor three phase voltage equations into a quadrature phase rotational frame, which is aligned and rotates synchronously with the rotor. Such a transformation, after some mathematical corrections, yields the following set of equations, describing the quadrature phase model of a PMSM motor, in a synchronous reference frame:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_S \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q \\ L_d \end{bmatrix} \begin{bmatrix} i_q \\ i_d \end{bmatrix} + \omega_e \psi_{pm} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation **GMCLIB_DecouplingPMSM_Eq1**

It can be seen that GMCLIB_DecouplingPMSM_Eq1 represents a non-linear cross dependent system. The linear voltage components cover the model of the phase winding, which is simplified to a resistance in series with inductance (R-L circuit). The cross-coupling components represent the mutual coupling between the two phases of the quadrature phase model, and the back-EMF component (visible only in q-axis voltage) represents the generated back EMF voltage caused by rotor rotation.

In order to achieve dynamic torque, speed and positional control, the non-linear and back-EMF components from GMCLIB_DecouplingPMSM_Eq1 must be compensated for. This will result in a fully decoupled flux and torque control of the machine and simplifies the PMSM motor model into two independent R-L circuit models as follows:

$$u_d = R_s i_d + L_d \frac{di_d}{dt}$$

$$u_q = R_s i_q + L_q \frac{di_q}{dt}$$

Equation **GMCLIB_DecouplingPMSM_Eq2**

Such a simplification of the PMSM model also greatly simplifies the design of both the d-q current controllers.

Therefore, it is advantageous to compensate for the cross-coupling terms in GMCLIB_DecouplingPMSM_Eq1, using the feed-forward voltages u_dq_comp given from GMCLIB_DecouplingPMSM_Eq1 as follows:

$$u_{dcomp} = -\omega_e L_q i_q$$

$$u_{qcomp} = \omega_e L_d i_d$$

Equation **GMCLIB_DecouplingPMSM_Eq3**

The feed-forward voltages u_dq_comp are added to the voltages generated by the current controllers u_dq, which cover the R-L model. The resulting voltages represent the direct u_{q_dec} and quadrature u_{q_decq} components of the decoupled voltage vector that is to be applied on the motor terminals (using a pulse width modulator). The back EMF voltage component is already considered to be compensated by an external function.

The function GMCLIB_DecouplingPMSM_F32 calculates the cross-coupling voltages u_dq_comp and adds these to the input u_dq voltage vector. Because the back EMF voltage component is considered compensated, this component is equal to zero. Therefore, calculations performed by GMCLIB_DecouplingPMSM_F32 are derived from these two equations:

$$u_{d_{\mathrm{dec}}} = u_d + u_{d_{\mathrm{comnp}}}$$
$$u_{q_{\mathrm{dec}}} = u_q + u_{q_{\mathrm{comp}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq4**

where u_dq is the voltage vector calculated by the controllers (with the already compensated back EMF component), u_dq_comp is the feed-forward compensating voltage vector described in GMCLIB_DecouplingPMSM_Eq3, and u_dq_dec is the resulting decoupled voltage vector to be applied on the motor terminals. Substituting GMCLIB_DecouplingPMSM_Eq3 into GMCLIB_DecouplingPMSM_Eq4, and normalizing GMCLIB_DecouplingPMSM_Eq4, results in the following set of equations:

$$u\mathrm{df}_{\mathrm{dec}} \bullet U_{\max} = u_{\mathrm{df}} \bullet U_{\max} - \omega_{\mathrm{ef}} \bullet \Omega_{\max} \bullet L_q \bullet i_{\mathrm{qf}} \bullet I_{\max}$$
$$u\mathrm{qf}_{\mathrm{dec}} \bullet U_{\max} = u_{\mathrm{qf}} \bullet U_{\max} + \omega_{\mathrm{ef}} \bullet \Omega_{\max} \bullet L_d \bullet i_{\mathrm{df}} \bullet I_{\max}$$

Equation **GMCLIB_DecouplingPMSM_Eq5**

where subscript f denotes the fractional representation of the respective quantity, and U_{max}, I_{max}, Ω_{max} are the maximal values (scale values) for the voltage, current and angular velocity respectively.

Real quantities are converted to the fractional range [-1, 1) using the following equations:

$$u_{df_{dec}} = \frac{u_{d_{dec}}}{U_{max}} \qquad u_{qf_{dec}} = \frac{u_{q_{dec}}}{U_{max}}$$
$$u_{df} = \frac{u_d}{U_{max}} \qquad u_{qf} = \frac{u_q}{U_{max}}$$
$$i_{df} = \frac{i_d}{I_{max}} \qquad i_{qf} = \frac{i_q}{I_{max}}$$
$$\omega_{ef} = \frac{\omega_e}{\Omega_{max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq6**

Further, rearranging GMCLIB_DecouplingPMSM_Eq5 results in:

$$u_{df_{dec}} = u_{df} - \omega_{ef} \bullet i_{qf} \frac{L_q \bullet \Omega_{max} \bullet I_{max}}{U_{max}} = u_{df} - \omega_{ef} \bullet i_{qf} \bullet k_d$$
$$uqf_{dec} = u_{qf} - \omega_{ef} \bullet i_{df} \frac{L_d \bullet \Omega_{max} \bullet I_{max}}{U_{max}} = u_{qf} - \omega_{ef} \bullet i_{df} \bullet k_q$$

Equation **GMCLIB_DecouplingPMSM_Eq7**

where k_d and k_q are coefficients calculated as:

$$k_d = L_q \bullet \Omega_{max} \bullet \frac{I_{max}}{U_{max}}$$
$$k_q = L_d \bullet \Omega_{max} \bullet \frac{I_{max}}{U_{max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq8**

Because function GMCLIB_DecouplingPMSM_F32 is implemented using the fractional arithmetic, both the k_d and k_q coefficients also have to be scaled to fit into the fractional range [-1, 1). For that purpose, two additional scaling coefficients are defined as:

$$k_{d\_shift} = \text{ceil}\left(\frac{\log(k_d)}{\log(2)}\right)$$
$$k_{q\_shift} = \text{ceil}\left(\frac{\log(k_q)}{\log(2)}\right)$$

Equation **GMCLIB_DecouplingPMSM_Eq9**

Using scaling coefficients GMCLIB_DecouplingPMSM_Eq9, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_Eq8 are derived as follows:

$$k_{\mathrm{df}} = k_d \bullet 2^{-k_{d\_\mathrm{shift}}}$$
$$k_{qf} = k_q \bullet 2^{-k_{q\_\mathrm{shift}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq10**

Substituting GMCLIB_DecouplingPMSM_Eq8 - GMCLIB_DecouplingPMSM_Eq10 into GMCLIB_DecouplingPMSM_Eq7 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F32 function:

$$u_{df_{\mathrm{dec}}} = u_{\mathrm{df}} - \omega_{\mathrm{ef}} \bullet i_{\mathrm{qf}} \bullet k_{\mathrm{df}} \bullet 2^{k_{d\_\mathrm{shift}}}$$
$$u_{qf_{\mathrm{dec}}} = u_{\mathrm{qf}} + \omega_{\mathrm{ef}} \bullet i_{\mathrm{df}} \bullet k_{\mathrm{qf}} \bullet 2^{k_{q\_\mathrm{shift}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq11**

Scaling of both equations into the fractional range is done using a multiplication by $2^{k\_d\_shift}$, $2^{k\_q\_shift}$, respectively. Therefore, it is implemented as a simple left shift with overflow protection.

Note

All parameters can be reset during declaration using the GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 macro.

4.100.5 Re-entrancy

The function is re-entrant.

4.100.6 Code Example

```
#include "gmclib.h"

#define L_D     (50.0e-3)    // Ld inductance = 50mH
#define L_Q     (100.0e-3)   // Lq inductance = 100mH
#define U_MAX   (50.0)       // scale for voltage = 50V
#define I_MAX   (10.0)       // scale for current = 10A
#define W_MAX   (2000.0)     // scale for angular velocity = 2000rad/sec

GMCLIB_DECOUPLINGPMSM_T_F32 f32trDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_F32;
SWLIBS_2Syst_F32 f32trUDQ;
SWLIBS_2Syst_F32 f32trIDQ;
SWLIBS_2Syst_F32 f32trUDecDQ;
tFrac32 f32We;
```

```
                void main(void)
                {
                  // input values - scaling coefficients of given decoupling algorithm
                  f32trDec.f32Kd = FRAC32 (0.625);
                  f32trDec.s16KdShift = 6;
                  f32trDec.f32Kq = FRAC32 (0.625);
                  f32trDec.s16KqShift = 5;
                  f32trUDQ.f32Arg1 = FRAC32 (5.0/U_MAX);  // d quantity of input voltage
vector 5[V]
                  f32trUDQ.f32Arg2 = FRAC32 (10.0/U_MAX); // q quantity of input voltage
vector 10[V]
                  f32trIDQ.f32Arg1 = FRAC32 (6.0/I_MAX);  // d quantity of measured current
vector 6[A]
                  f32trIDQ.f32Arg2 = FRAC32 (4.0/I_MAX);  // q quantity of measured current
vector 4[A]
                  f32We = FRAC32 (100.0/W_MAX); // rotor angular velocity

                  //output should be f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V
~=-35[V]
                  //output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
~=40[V]
                  GMCLIB_DecouplingPMSM_F32
(&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec);

                  //output should be f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V
~=-35[V]
                  //output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
~=40[V]
                  GMCLIB_DecouplingPMSM
(&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec,Define F32);

                  // ###############################################################
                  // Available only if 32-bit fractional implementation selected
                  // as default
                  // ###############################################################

                  //output should be f32trUDecDQ.f32Arg1 ~ 0xA6666666 ~ FRAC32(-0.7)*50V
~=-35[V]
                  //output should be f32trUDecDQ.f32Arg2 ~ 0x66666666 ~ FRAC32(0.8)*50V
~=40[V]
                  GMCLIB_DecouplingPMSM (&f32trUDecDQ,&f32trUDQ,&f32trIDQ,f32We,&f32trDec);
                }
```

4.101 Function GMCLIB_DecouplingPMSM_F16

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the field oriented control.

4.101.1 Declaration

```
void GMCLIB_DecouplingPMSM_F16(SWLIBS_2Syst_F16 *const pUdqDec, const SWLIBS_2Syst_F16 *const
pUdq, const SWLIBS_2Syst_F16 *const pIdq, tFrac16 f16AngularVel, const
GMCLIB_DECOUPLINGPMSM_T_F16 *const pParam);
```

4.101.2 Arguments

Table 4-126. GMCLIB_DecouplingPMSM_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pUdqDec	**output**	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_F16 *const	pUdq	**input**	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_F16 *const	pIdq	**input**	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFrac16	f16AngularVel	**input**	Rotor angular velocity in rad/sec, referred to as ($_{ef}$) in the detailed section of the documentation.
const GMCLIB_DECOUPLINGPMSM_T_F16 *const	pParam	**input**	Pointer to the structure containing k_{df} and k_{qf} coefficients (see the detailed section of the documentation) and scale parameters (k_{d_shift}) and (k_{q_shift}).

4.101.3 Return

Function returns no value.

4.101.4 Description

The quadrature phase model of a PMSM motor, in a synchronous reference frame, is very popular for field oriented control structures because both controllable quantities, current and voltage, are DC values. This allows employing only simple controllers to force the machine currents into the defined states.

The voltage equations of this model can be obtained by transforming the motor three phase voltage equations into a quadrature phase rotational frame, which is aligned and rotates synchronously with the rotor. Such a transformation, after some mathematical corrections, yields the following set of equations, describing the quadrature phase model of a PMSM motor, in a synchronous reference frame:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_S \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q \\ L_d \end{bmatrix} \begin{bmatrix} i_q \\ i_d \end{bmatrix} + \omega_e \psi_{pm} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation **GMCLIB_DecouplingPMSM_Eq1**

It can be seen that GMCLIB_DecouplingPMSM_Eq1 represents a non-linear cross dependent system. The linear voltage components cover the model of the phase winding, which is simplified to a resistance in series with inductance (R-L circuit). The cross-coupling components represent the mutual coupling between the two phases of the quadrature phase model, and the back-EMF component (visible only in q-axis voltage) represents the generated back EMF voltage caused by rotor rotation.

In order to achieve dynamic torque, speed and positional control, the non-linear and back-EMF components from GMCLIB_DecouplingPMSM_Eq1 must be compensated for. This will result in a fully decoupled flux and torque control of the machine and simplifies the PMSM motor model into two independent R-L circuit models as follows:

$$u_d = R_s i_d + L_d \frac{di_d}{dt}$$
$$u_q = R_s i_q + L_q \frac{di_q}{dt}$$

Equation **GMCLIB_DecouplingPMSM_Eq2**

Such a simplification of the PMSM model also greatly simplifies the design of both the d-q current controllers.

Therefore, it is advantageous to compensate for the cross-coupling terms in GMCLIB_DecouplingPMSM_Eq1, using the feed-forward voltages $\overline{u\_dq\_comp}$ given from GMCLIB_DecouplingPMSM_Eq1 as follows:

$$u_{dcomp} = -\omega_e L_q i_q$$
$$u_{qcomp} = \omega_e L_d i_d$$

Equation **GMCLIB_DecouplingPMSM_Eq3**

The feed-forward voltages $\overline{u\_dq\_comp}$ are added to the voltages generated by the current controllers $\overline{u\_dq}$, which cover the R-L model. The resulting voltages represent the direct u_{q_dec} and quadrature u_{q_decq} components of the decoupled voltage vector that is to be applied on the motor terminals (using a pulse width modulator). The back EMF voltage component is already considered to be compensated by an external function.

The function GMCLIB_DecouplingPMSM_F16 calculates the cross-coupling voltages $\overline{u\_dq\_comp}$ and adds these to the input $\overline{u\_dq}$ voltage vector. Because the back EMF voltage component is considered compensated, this component is equal to zero. Therefore, calculations performed by GMCLIB_DecouplingPMSM_F16 are derived from these two equations:

$$u_{d_{dec}} = u_d + u_{d_{comnp}}$$
$$u_{q_{dec}} = u_q + u_{q_{comp}}$$

Equation **GMCLIB_DecouplingPMSM_Eq4**

where $\overline{u\_dq}$ is the voltage vector calculated by the controllers (with the already compensated back EMF component), $\overline{u\_dq\_comp}$ is the feed-forward compensating voltage vector described in GMCLIB_DecouplingPMSM_Eq3, and $\overline{u\_dq\_dec}$ is the resulting decoupled voltage vector to be applied on the motor terminals. Substituting GMCLIB_DecouplingPMSM_Eq3 into GMCLIB_DecouplingPMSM_Eq4, and normalizing GMCLIB_DecouplingPMSM_Eq4, results in the following set of equations:

$$u_{df_{dec}} \bullet U_{max} = u_{df} \bullet U_{max} - \omega_{ef} \bullet \Omega_{max} \bullet L_q \bullet i_{qf} \bullet I_{max}$$
$$u_{qf_{dec}} \bullet U_{max} = u_{qf} \bullet U_{max} + \omega_{ef} \bullet \Omega_{max} \bullet L_d \bullet i_{df} \bullet I_{max}$$

Equation **GMCLIB_DecouplingPMSM_Eq5**

where subscript f denotes the fractional representation of the respective quantity, and U_{max}, I_{max}, Ω_{max} are the maximal values (scale values) for the voltage, current and angular velocity respectively.

Real quantities are converted to the fractional range [-1, 1) using the following equations:

$$u_{df_{dec}} = \frac{u_{d_{dec}}}{U_{max}} \qquad u_{qf_{dec}} = \frac{u_{q_{dec}}}{U_{max}}$$
$$u_{df} = \frac{u_d}{U_{max}} \qquad u_{qf} = \frac{u_q}{U_{max}}$$
$$i_{df} = \frac{i_d}{I_{max}} \qquad i_{qf} = \frac{i_q}{I_{max}}$$
$$\omega_{ef} = \frac{\omega_e}{\Omega_{max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq6**

Further, rearranging GMCLIB_DecouplingPMSM_Eq5 results in:

$$u_{df_{dec}} = u_{df} - \omega_{ef} \bullet i_{qf} \frac{L_q \bullet \Omega_{max} \bullet I_{max}}{U_{max}} = u_{df} - \omega_{ef} \bullet i_{qf} \bullet k_d$$
$$uqf_{dec} = u_{qf} - \omega_{ef} \bullet i_{df} \frac{L_d \bullet \Omega_{max} \bullet I_{max}}{U_{max}} = u_{qf} - \omega_{ef} \bullet i_{df} \bullet k_q$$

Equation **GMCLIB_DecouplingPMSM_Eq7**

where k_d and k_q are coefficients calculated as:

$$k_d = L_q \bullet \Omega_{\max} \bullet \frac{I_{\max}}{U_{\max}}$$
$$k_q = L_d \bullet \Omega_{\max} \bullet \frac{I_{\max}}{U_{\max}}$$

Equation **GMCLIB_DecouplingPMSM_Eq8**

Because function GMCLIB_DecouplingPMSM_F16 is implemented using the fractional arithmetic, both the k_d and k_q coefficients also have to be scaled to fit into the fractional range [-1, 1). For that purpose, two additional scaling coefficients are defined as:

$$k_{d\_\text{shift}} = \text{ceil}\left(\frac{\log(k_d)}{\log(2)}\right)$$
$$k_{q\_\text{shift}} = \text{ceil}\left(\frac{\log(k_q)}{\log(2)}\right)$$

Equation **GMCLIB_DecouplingPMSM_Eq9**

Using scaling coefficients GMCLIB_DecouplingPMSM_Eq9, the fractional representation of coefficients k_d and k_q from GMCLIB_DecouplingPMSM_Eq8 are derived as follows:

$$k_{\text{df}} = k_d \bullet 2^{-k_{d\_\text{shift}}}$$
$$k_{qf} = k_q \bullet 2^{-k_{q\_\text{shift}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq10**

Substituting GMCLIB_DecouplingPMSM_Eq8 - GMCLIB_DecouplingPMSM_Eq10 into GMCLIB_DecouplingPMSM_Eq7 results in the final form of the equation set, actually implemented in the GMCLIB_DecouplingPMSM_F16 function:

$$ud_{f_{\text{dec}}} = u_{\text{df}} - \omega_{\text{ef}} \bullet i_{\text{qf}} \bullet k_{\text{df}} \bullet 2^{k_{d\_\text{shift}}}$$
$$uq_{f_{\text{dec}}} = u_{\text{qf}} + \omega_{\text{ef}} \bullet i_{\text{df}} \bullet k_{\text{qf}} \bullet 2^{k_{q\_\text{shift}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq11**

Scaling of both equations into the fractional range is done using a multiplication by $2^{k\_d\_shift}$, $2^{k\_q\_shift}$, respectively. Therefore, it is implemented as a simple left shift with overflow protection.

Note

All parameters can be reset during declaration using the GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 macro.

4.101.5 Re-entrancy

The function is re-entrant.

4.101.6 Code Example

```
#include "gmclib.h"

#define L_D     (50.0e-3)   // Ld inductance = 50mH
#define L_Q     (100.0e-3)  // Lq inductance = 100mH
#define U_MAX   (50.0)      // scale for voltage = 50V
#define I_MAX   (10.0)      // scale for current = 10A
#define W_MAX   (2000.0)    // scale for angular velocity = 2000rad/sec

GMCLIB_DECOUPLINGPMSM_T_F16 f16trDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_F16;
SWLIBS_2Syst_F16 f16trUDQ;
SWLIBS_2Syst_F16 f16trIDQ;
SWLIBS_2Syst_F16 f16trUDecDQ;
tFrac16 f16We;

void main(void)
{
  // input values - scaling coefficients of given decoupling algorithm
  f16trDec.f16Kd = FRAC16 (0.625);
  f16trDec.s16KdShift = 6;
  f16trDec.f16Kq = FRAC16 (0.625);
  f16trDec.s16KqShift = 5;
  f16trUDQ.f16Arg1 = FRAC16 (5.0/U_MAX);  // d quantity of input voltage
vector 5[V]
  f16trUDQ.f16Arg2 = FRAC16 (10.0/U_MAX); // q quantity of input voltage
vector 10[V]
  f16trIDQ.f16Arg1 = FRAC16 (6.0/I_MAX);  // d quantity of measured current
vector 6[A]
  f16trIDQ.f16Arg2 = FRAC16 (4.0/I_MAX);  // q quantity of measured current
vector 4[A]
  f16We = FRAC16 (100.0/W_MAX); // rotor angular velocity

  //output should be f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
  //output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM_F16
(&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec);

  //output should be f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
  //output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
  GMCLIB_DecouplingPMSM
(&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec,Define F16);

  // ###########################################################
  // Available only if 16-bit fractional implementation selected
```

```
    // as default
    // ###############################################################

    //output should be f16trUDecDQ.f16Arg1 ~ 0xA666 ~ FRAC16(-0.7)*50V ~=-35[V]
    //output should be f16trUDecDQ.f16Arg2 ~ 0x6666 ~ FRAC16(0.8)*50V ~=40[V]
    GMCLIB_DecouplingPMSM (&f16trUDecDQ,&f16trUDQ,&f16trIDQ,f16We,&f16trDec);
}
```

4.102 Function GMCLIB_DecouplingPMSM_FLT

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the field oriented control.

4.102.1 Declaration

```
void GMCLIB_DecouplingPMSM_FLT(SWLIBS_2Syst_FLT *const pUdqDec, const SWLIBS_2Syst_FLT *const
pUdq, const SWLIBS_2Syst_FLT *const pIdq, tFloat fltAngularVel, const
GMCLIB_DECOUPLINGPMSM_T_FLT *const pParam);
```

4.102.2 Arguments

Table 4-127. GMCLIB_DecouplingPMSM_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pUdqDec	**output**	Pointer to the structure containing direct (u_{df_dec}) and quadrature (u_{qf_dec}) components of the decoupled stator voltage vector to be applied on the motor terminals.
const SWLIBS_2Syst_FLT *const	pUdq	**input**	Pointer to the structure containing direct (u_{df}) and quadrature (u_{qf}) components of the stator voltage vector generated by the current controllers.
const SWLIBS_2Syst_FLT *const	pIdq	**input**	Pointer to the structure containing direct (i_{df}) and quadrature (i_{qf}) components of the stator current vector measured on the motor terminals.
tFloat	fltAngularVel	**input**	Rotor angular velocity in rad/sec, referred to as ($_{ef}$) in the detailed section of the documentation.
const GMCLIB_DECOUPLINGPMSM_T_FLT *const	pParam	**input**	Pointer to the structure containing L_D and L_Q coefficients (see the detailed section of the documentation).

4.102.3 Return

Function returns no value.

4.102.4 Description

The quadrature phase model of a PMSM motor, in a synchronous reference frame, is very popular for field oriented control structures because both controllable quantities, current and voltage, are DC values. This allows employing only simple controllers to force the machine currents into the defined states.

The voltage equations of this model can be obtained by transforming the motor three phase voltage equations into a quadrature phase rotational frame, which is aligned and rotates synchronously with the rotor. Such a transformation, after some mathematical corrections, yields the following set of equations, describing the quadrature phase model of a PMSM motor, in a synchronous reference frame:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_S \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} \frac{d}{dt} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q \\ L_d \end{bmatrix} \begin{bmatrix} i_q \\ i_d \end{bmatrix} + \omega_e \psi_{pm} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Equation **GMCLIB_DecouplingPMSM_Eq1**

It can be seen that GMCLIB_DecouplingPMSM_Eq1 represents a non-linear cross dependent system. The linear voltage components cover the model of the phase winding, which is simplified to a resistance in series with inductance (R-L circuit). The cross-coupling components represent the mutual coupling between the two phases of the quadrature phase model, and the back-EMF component (visible only in q-axis voltage) represents the generated back EMF voltage caused by rotor rotation.

In order to achieve dynamic torque, speed and positional control, the non-linear and back-EMF components from GMCLIB_DecouplingPMSM_Eq1 must be compensated for. This will result in a fully decoupled flux and torque control of the machine and simplifies the PMSM motor model into two independent R-L circuit models as follows:

$$u_d = R_s i_d + L_d \frac{di_d}{dt}$$

$$u_q = R_s i_q + L_q \frac{di_q}{dt}$$

Equation **GMCLIB_DecouplingPMSM_Eq2**

Such a simplification of the PMSM model also greatly simplifies the design of both the d-q current controllers.

Therefore, it is advantageous to compensate for the cross-coupling terms in GMCLIB_DecouplingPMSM_Eq1, using the feed-forward voltages u_dq_comp given from GMCLIB_DecouplingPMSM_Eq1 as follows:

$$u_{\text{dcomp}} = -\omega_e L_q i_q$$
$$u_{\text{qcomp}} = \omega_e L_d i_d$$

Equation **GMCLIB_DecouplingPMSM_Eq3**

The feed-forward voltages u_dq_comp are added to the voltages generated by the current controllers u_dq, which cover the R-L model. The resulting voltages represent the direct u $_{\text{q\_dec}}$ and quadrature u $_{\text{q\_decq}}$ components of the decoupled voltage vector that is to be applied on the motor terminals (using a pulse width modulator). The back EMF voltage component is already considered to be compensated for by an external function.

The function GMCLIB_DecouplingPMSM_FLT calculates the cross-coupling voltages u_dq_comp and adds these to the input u_dq voltage vector. Because the back EMF voltage component is considered compensated, this component is equal to zero. Therefore, calculations performed by GMCLIB_DecouplingPMSM_FLT are derived from these two equations:

$$u_{d_{\text{dec}}} = u_d + u_{d_{\text{comnp}}}$$
$$u_{q_{\text{dec}}} = u_q + u_{q_{\text{comp}}}$$

Equation **GMCLIB_DecouplingPMSM_Eq4**

where u_dq is the voltage vector calculated by the controllers (with the already compensated for back EMF component), u_dq_comp is the feed-forward compensating voltage vector described in GMCLIB_DecouplingPMSM_Eq3, and u_dq_dec is the resulting decoupled voltage vector to be applied on the motor terminals. Substituting GMCLIB_DecouplingPMSM_Eq3 into GMCLIB_DecouplingPMSM_Eq4, and normalizing GMCLIB_DecouplingPMSM_Eq4, results in the following set of equations:

$$u_{\text{df}_{\text{dec}}} \bullet U_{\max} = u_{\text{df}} \bullet U_{\max} - \omega_{\text{ef}} \bullet \Omega_{\max} \bullet L_q \bullet i_{\text{qf}} \bullet I_{\max}$$
$$u_{\text{qf}_{\text{dec}}} \bullet U_{\max} = u_{\text{qf}} \bullet U_{\max} + \omega_{\text{ef}} \bullet \Omega_{\max} \bullet L_d \bullet i_{\text{df}} \bullet I_{\max}$$

Equation **GMCLIB_DecouplingPMSM_Eq5**

Note

All parameters can be reset during declaration using the GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT macro. All inputs and parameters contain single precision floating point data type values.

4.102.5 Re-entrancy

The function is re-entrant.

4.102.6 Code Example

```
#include "gmclib.h"

GMCLIB_DECOUPLINGPMSM_T_FLT flttrDec = GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT;
SWLIBS_2Syst_FLT flttrUDQ;
SWLIBS_2Syst_FLT flttrIDQ;
SWLIBS_2Syst_FLT flttrUDecDQ;
tFloat fltWe;

void main(void)
{
  // input values
  flttrDec.fltLD = 50e-3; // LD inductance = 50mH
  flttrDec.fltLQ = 100e-3; // LQ inductance = 100mH
  flttrUDQ.fltArg1 = 5.0; // D quantity of input voltage vector 5[V]
  flttrUDQ.fltArg2 = 10.0; // Q quantity of input voltage vector 10[V]
  flttrIDQ.fltArg1 = 6.0; // D quantity of measured current vector 6[A]
  flttrIDQ.fltArg2 = 4.0; // Q quantity of measured current vector 4[A]
  fltWe 100.0; // rotor angular velocity

  // output should be flttrUDecDQ.fltArg1 ~= -35[V]
  // output should be flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM_FLT
(&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec);

  // output should be flttrUDecDQ.fltArg1 ~= -35[V]
  // output should be flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM
(&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec,Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be flttrUDecDQ.fltArg1 ~= -35[V]
  // output should be flttrUDecDQ.fltArg2 ~= 40[V]
  GMCLIB_DecouplingPMSM (&flttrUDecDQ,&flttrUDQ,&flttrIDQ,fltWe,&flttrDec);
}
```

4.103 Function GMCLIB_ElimDcBusRip_F32

This function implements the DC Bus voltage ripple elimination.

4.103.1 Declaration

```
void GMCLIB_ElimDcBusRip_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_F32 *const pParam);
```

4.103.2 Arguments

Table 4-128. GMCLIB_ElimDcBusRip_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	**output**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUSRIP_T_F32 *const	pParam	**input**	Pointer to the parameters structure.

4.103.3 Return

Function returns no value.

4.103.4 Description

The GMCLIB_ElimDcBusRip function provides a computational method for the recalculation of the direct (α) and quadrature (β) components of the required stator voltage vector, so as to compensate for voltage ripples on the DC bus of the power stage.

Considering a cascaded type structure of the control system in a standard motor control application, the required voltage vector to be applied on motor terminals is generated by a set of controllers (usually P, PI or PID) only with knowledge of the maximal value of the DC bus voltage. The amplitude and phase of the required voltage vector are then used by the pulse width modulator (PWM) for generation of appropriate duty-cycles for the power inverter switches. Obviously, the amplitude of the generated phase voltage (averaged across one switching period) does not only depend on the actual on/off times of the given phase switches and the maximal value of the DC bus voltage. The actual amplitude of the phase voltage is also directly affected by the actual value of the

available DC bus voltage. Therefore, any variations in amplitude of the actual DC bus voltage must be accounted for by modifying the amplitude of the required voltage so that the output phase voltage remains unaffected.

For a better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC\_BUS\_MAX}$=100[V]
- actual amplitude of the DC bus voltage $_{DC\_BUS\_ACTUAL}$[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req\_new} = \frac{U_{req} \bullet U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_Eq1**

Example 2:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC\_BUS\_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC\_BUS\_ACTUAL}$=90[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req\_new} = \frac{U_{req} \bullet U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_Eq2**

The imperfections of the DC bus voltage are compensated for by the modification of amplitudes of the direct- α and the quadrature- β components of the stator reference voltage vector. The following formulas are used:

- for the α-component:

$$u_\alpha^* = \begin{cases} \frac{f32ModIndex \bullet u_\alpha}{f32ArgDcBusMsr/2} & \text{if } abs\left(f32ModIndex \bullet u_\alpha\right) < \frac{f32ArgDcBusMsr}{2} \\ sign\left(u_\alpha\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β-component:

$$u_\beta^* = \begin{cases} \dfrac{f32ModIndex \bullet u_\beta}{f32ArgDcBusMsr/2} & \text{if} \quad abs\left(f32ModIndex \bullet u_\beta\right) < \dfrac{f32ArgDcBusMsr}{2} \\ sign\left(u_\beta\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq4**

where: f32ModIndex is the inverse modulation index, f32ArgDcBusMsr is the measured DC bus voltage, the u_α and u_β are the input voltages, and the u_α^* and u_β^* are the output duty-cycle ratios.

The f32ModIndex and f32ArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_α, u_β correspond respectively to the f32Arg1 and f32Arg2 members of the input structure, and the u_α^* and u_β^* respectively to the f32Arg1 and f32Arg2 members of the output structure.

It should be noted that although the modulation index (see the parameters structure, the f32ModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

In order to correctly handle the discontinuity at f32ArgDcBusMsr approaching 0, and for efficiency reasons, the function will assign 0 to the output duty cycle ratios if the f32ArgDcBusMsr is below the threshold of 2^{-15}. In other words, the 16 least significant bits of the f32DcBusMsr are ignored. Also, the computed output of the and u_β^* components may have an inaccuracy in the 16 least significant bits.

Note

Both the inverse modulation index pIn->f32ModIndex and the measured DC bus voltage pIn->f32DcBusMsr must be equal to or greater than 0, otherwise the results are undefined.

4.103.5 Re-entrancy

The function is re-entrant.

4.103.6 Code Example

```
#include "gmclib.h"

#define U_MAX   (36.0) // voltage scale
SWLIBS_2Syst_F32 f32AB;
```

```
                SWLIBS_2Syst_F32 f32OutAB;
                GMCLIB_ELIMDCBUSRIP_T_F32 f32trMyElimDcBusRip =
GMCLIB_ELIMDCBUSRIP_DEFAULT_F32;

                void main(void)
                {
                  // inverse modulation coefficient for standard space vector modulation
                  f32trMyElimDcBusRip.f32ModIndex = FRAC32 (0.866025403784439);
                  // Input voltage vector 15V @ angle 30deg
                  // alpha component of input voltage vector = 12.99[V]
                  f32AB.f32Arg1 = FRAC32 (12.99/U_MAX);
                  // beta component of input voltage vector = 7.5[V]
                  f32AB.f32Arg2 = FRAC32 (7.5/U_MAX);
                  // value of the measured DC bus voltage 17[V]
                  f32trMyElimDcBusRip.f32ArgDcBusMsr = FRAC32 (17.0/U_MAX);

                  // output alpha component of the output vector should be
                  // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC32(1.0)
~ 0x7FFFFFFF
                  // output beta component of the output vector should be
                  // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
FRAC32(0.7641) ~ 0x61CF8000
                  GMCLIB_ElimDcBusRip_F32 (&f32OutAB,&f32AB,&f32trMyElimDcBusRip);

                  // output alpha component of the output vector should be
                  // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC32(1.0)
~ 0x7FFFFFFF
                  // output beta component of the output vector should be
                  // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
FRAC32(0.7641) ~ 0x61CF8000
                  GMCLIB_ElimDcBusRip (&f32OutAB,&f32AB,&f32trMyElimDcBusRip,Define F32);

                  // ###############################################################
                  // Available only if 32-bit fractional implementation selected
                  // as default
                  // ###############################################################

                  // output alpha component of the output vector should be
                  // f32OutAB.f32Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC32(1.0)
~ 0x7FFFFFFF
                  // output beta component of the output vector should be
                  // f32OutAB.f32Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
FRAC32(0.7641) ~ 0x61CF8000
                  GMCLIB_ElimDcBusRip (&f32OutAB,&f32AB,&f32trMyElimDcBusRip);
                }
```

4.104 Function GMCLIB_ElimDcBusRip_F16

This function implements the DC Bus voltage ripple elimination.

4.104.1 Declaration

```
void GMCLIB_ElimDcBusRip_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_F16 *const pParam);
```

4.104.2 Arguments

Table 4-129. GMCLIB_ElimDcBusRip_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	**output**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUSRIP_T_F16 *const	pParam	**input**	Pointer to the parameters structure.

4.104.3 Return

Function returns no value.

4.104.4 Description

The GMCLIB_ElimDcBusRip function provides a computational method for the recalculation of the direct (α) and quadrature (β) components of the required stator voltage vector, so as to compensate for voltage ripples on the DC bus of the power stage.

Considering a cascaded type structure of the control system in a standard motor control application, the required voltage vector to be applied on motor terminals is generated by a set of controllers (usually P, PI or PID) only with knowledge of the maximal value of the DC bus voltage. The amplitude and phase of the required voltage vector are then used by the pulse width modulator (PWM) for generation of appropriate duty-cycles for the power inverter switches. Obviously, the amplitude of the generated phase voltage (averaged across one switching period) does not only depend on the actual on/off times of the given phase switches and the maximal value of the DC bus voltage. The actual amplitude of the phase voltage is also directly affected by the actual value of the available DC bus voltage. Therefore, any variations in amplitude of the actual DC bus voltage must be accounted for by modifying the amplitude of the required voltage so that the output phase voltage remains unaffected.

For a better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage U_{reg}=50[V]

- maximal amplitude of the DC bus voltage $U_{DC\_BUS\_MAX}$=100[V]
- actual amplitude of the DC bus voltage $_{DC\_BUS\_ACTUAL}$[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req\_new} = \frac{U_{req} \bullet U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_Eq1**

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC\_BUS\_MAX}$=100[V]
- actual amplitude of the DC bus voltage $U_{DC\_BUS\_ACTUAL}$=90[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req\_new} = \frac{U_{req} \bullet U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_Eq2**

- for the α-component:

$$u_{\alpha}^{*} = \begin{cases} \frac{f16ModIndex \bullet u_{\alpha}}{f16ArgDcBusMsr/2} & \text{if } abs\left(f16ModIndex \bullet u_{\alpha}\right) < \frac{f16ArgDcBusMsr}{2} \\ sign\left(u_{\alpha}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β-component:

$$u_{\beta}^{*} = \begin{cases} \frac{f32ModIndex \bullet u_{\beta}}{f32ArgDcBusMsr/2} & \text{if } abs\left(f32ModIndex \bullet u_{\beta}\right) < \frac{f32ArgDcBusMsr}{2} \\ sign\left(u_{\beta}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq4**

where: f16ModIndex is the inverse modulation index, f16ArgDcBusMsr is the measured DC bus voltage, the u_{α} and u_{β} are the input voltages, and the u_{α}^{*} and u_{β}^{*} are the output duty-cycle ratios.

The f16ModIndex and f16ArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_α, u_β correspond respectively to the f16Arg1 and f16Arg2 members of the input structure, and the u_α* and u_β* respectively to the f16Arg1 and f16Arg2 members of the output structure.

It should be noted that although the modulation index (see the parameters structure, the f16ModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

In order to correctly handle the discontinuity at f16ArgDcBusMsr approaching 0, and for efficiency reasons, the function will assign 0 to the output duty cycle ratios if the f16ArgDcBusMsr is below the threshold of 2^{-15}. In other words, the 16 least significant bits of the f16DcBusMsr are ignored. Also, the computed output of the and u_β* components may have an inaccuracy in the 16 least significant bits.

Note

Both the inverse modulation index pIn->f16ModIndex and the measured DC bus voltage pIn->f16DcBusMsr must be equal to or greater than 0, otherwise the results are undefined.

4.104.5 Re-entrancy

The function is re-entrant.

4.104.6 Code Example

```
#include "gmclib.h"

#define U_MAX   (36.0) // voltage scale
SWLIBS_2Syst_F16 f16AB;
SWLIBS_2Syst_F16 f16OutAB;
GMCLIB_ELIMDCBUSRIP_T_F16 f16trMyElimDcBusRip =
GMCLIB_ELIMDCBUSRIP_DEFAULT_F16;

void main(void)
{
  // inverse modulation coefficient for standard space vector modulation
  f16trMyElimDcBusRip.f16ModIndex = FRAC16 (0.866025403784439);
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  f16AB.f16Arg1 = FRAC16 (12.99/U_MAX);
  // beta component of input voltage vector = 7.5[V]
  f16AB.f16Arg2 = FRAC16 (7.5/U_MAX);
  // value of the measured DC bus voltage 17[V]
  f16trMyElimDcBusRip.f16ArgDcBusMsr = FRAC16 (17.0/U_MAX);

  // output alpha component of the output vector should be
  // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC16(1.0)
~ 0x7FFF
```

```
                // output beta component of the output vector should be
                // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
FRAC16(0.7641) ~ 0x61CF
                GMCLIB_ElimDcBusRip_F16 (&f16OutAB,&f16AB,&f16trMyElimDcBusRip);

                // output alpha component of the output vector should be
                // f16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC16(1.0)
~ 0x7FFF
                // output beta component of the output vector should be
                // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
FRAC16(0.7641) ~ 0x61CF
                GMCLIB_ElimDcBusRip (&f16OutAB,&f16AB,&f16trMyElimDcBusRip,Define F16);

                // ###############################################################
                // Available only if 16-bit fractional implementation selected
                // as default
                // ###############################################################

                // output alpha component of the output vector should be
                // 16OutAB.f16Arg1 = (12.99/36)*0.8660/(17.0/36/2) = 1.3235 -> FRAC16(1.0)
~ 0x7FFF
                // output beta component of the output vector should be
                // f16OutAB.f16Arg2 = (7.5/36)*0.8660/(17.0/36/2) = 0.7641 ->
FRAC16(0.7641) ~ 0x61CF
                GMCLIB_ElimDcBusRip (&f16OutAB,&f16AB,&f16trMyElimDcBusRip);
              }
```

4.105 Function GMCLIB_ElimDcBusRip_FLT

This function implements the DC Bus voltage ripple elimination.

4.105.1 Declaration

```
void GMCLIB_ElimDcBusRip_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pIn,
const GMCLIB_ELIMDCBUSRIP_T_FLT *const pParam);
```

4.105.2 Arguments

Table 4-130. GMCLIB_ElimDcBusRip_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pOut	**output**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector re-calculated so as to compensate for voltage ripples on the DC bus.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure with direct (α) and quadrature (β) components of the required stator voltage vector before compensation of voltage ripples on the DC bus.
const GMCLIB_ELIMDCBUSRIP_T_FLT *const	pParam	**input**	Pointer to the parameters structure.

4.105.3 Return

Function returns no value.

4.105.4 Description

The GMCLIB_ElimDcBusRip function provides a computational method for the recalculation of the direct (α) and quadrature (β) components of the required stator voltage vector, so as to compensate for voltage ripples on the DC bus of the power stage.

Considering a cascaded type structure of the control system in a standard motor control application, the required voltage vector to be applied on motor terminals is generated by a set of controllers (usually P, PI or PID) with only the knowledge of the maximal value of the DC bus voltage. The amplitude and phase of the required voltage vector are then used by the pulse width modulator (PWM) for generation of appropriate duty-cycles for the power inverter switches. Obviously, the amplitude of the generated phase voltage (averaged across one switching period) does not only depend on the actual on/off times of the given phase switches and the maximal value of the DC bus voltage. The actual amplitude of the phase voltage is also directly affected by the actual value of the available DC bus voltage. Therefore, any variations in amplitude of the actual DC bus voltage must be accounted for by modifying the amplitude of the required voltage so that the output phase voltage remains unaffected.

For a better understanding, let's consider the following two simple examples:

Example 1:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC\_BUS\_MAX}$=100[V]
- actual amplitude of the DC bus voltage $_{DC\_BUS\_ACTUAL}$[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req\_new} = \frac{U_{req} \bullet U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 50V$$

Equation **GMCLIB_ElimDcBusRip_Eq1**

Example 2:

- amplitude of the required phase voltage U_{reg}=50[V]
- maximal amplitude of the DC bus voltage $U_{DC\_BUS\_MAX}$=100[V]

- actual amplitude of the DC bus voltage $U_{DC\_BUS\_ACTUAL}$=90[V]
- voltage to be applied to the PWM modulator to generate U_{reg}=50[V] on the inverter phase output:

$$U_{req\_new} = \frac{U_{req} \bullet U_{DC\_BUS\_MAX}}{U_{DC\_BUS\_ACTUAL}} = 55.5V$$

Equation **GMCLIB_ElimDcBusRip_Eq2**

The imperfections of the DC bus voltage are compensated for by the modification of amplitudes of the direct- α and the quadrature- β components of the stator reference voltage vector. The following formulas are used:

- for the α-component:

$$u_{\alpha}^{*} = \begin{cases} \frac{\text{fltModIndex} \bullet u_{\alpha}}{\text{fltArgDcBusMsr}/2} & \text{if} \quad \text{abs}\left(\text{fltModIndex} \bullet u_{\alpha}\right) < \frac{\text{fltArgDcBusMsr}}{2} \\ \text{sign}\left(u_{\alpha}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq3**

- for the β-component:

$$u_{\beta}^{*} = \begin{cases} \frac{\text{fltModIndex} \bullet u_{\beta}}{\text{fltArgDcBusMsr}/2} & \text{if} \quad \text{abs}\left(\text{fltModIndex} \bullet u_{\beta}\right) < \frac{\text{fltArgDcBusMsr}}{2} \\ \text{sign}\left(u_{\beta}\right) & \text{otherwise} \end{cases}$$

Equation **GMCLIB_ElimDcBusRip_Eq4**

where fltModIndex is the inverse modulation index, fltArgDcBusMsr is the measured DC bus voltage, the u_{α} and u_{β} are the input voltages, and the u_{α}^{*} and u_{β}^{*} are the output duty-cycle ratios.

The fltModIndex and fltArgDcBusMsr are supplied to the function within the parameters structure through its members. The u_{α}, u_{β} correspond respectively to the fltArg1 and fltArg2 members of the input structure pIn, and the u_{α}^{*} and u_{β}^{*} respectively to the fltArg1 and fltArg2 members of the output structure pOut.

It should be noted that although the modulation index (see the parameters structure pParam, the fltModIndex member) is assumed to be equal to or greater than zero, the possible values are restricted to those values resulting from the use of Space Vector Modulation techniques.

Note

Both the inverse modulation index pIn->fltModIndex and the measured DC bus voltage pIn->fltDcBusMsr must be equal to or greater than 0, otherwise the results are undefined.

4.105.5 Re-entrancy

The function is re-entrant.

4.105.6 Code Example

```
                #include "gmclib.h"

                SWLIBS_2Syst_FLT fltAB;
                SWLIBS_2Syst_FLT fltOutAB;
                GMCLIB_ELIMDCBUSRIP_T_FLT flttrMyElimDcBusRip =
GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT;

                void main(void)
                {
                  // inverse modulation coefficient for standard space vector modulation
                  trMyElimDcBusRip.fltModIndex = 0.866025404;
                  // Input voltage vector 15V @ angle 30deg
                  // alpha component of input voltage vector = 12.99[V]
                  fltAB.fltArg1 = 12.99038106;
                  // beta component of input voltage vector = 7.5[V]
                  fltAB.fltArg2 = 7.5;
                  // value of the measured DC bus voltage 17V
                  flttrMyElimDcBusRip.fltArgDcBusMsr = 17;

                  // output alpha component of the output vector should be fltOutAB.fltArg1 =
1
                  // output beta component of the output vector should be fltOutAB.fltArg2 =
0.764140062
                  GMCLIB_ElimDcBusRip_FLT (&fltOutAB,&fltAB,&flttrMyElimDcBusRip);

                  // output alpha component of the output vector should be fltOutAB.fltArg1 =
1
                  // output beta component of the output vector should be fltOutAB.fltArg2 =
0.764140062
                  GMCLIB_ElimDcBusRip (&fltOutAB,&fltAB,&flttrMyElimDcBusRip,Define FLT);

                  // ##############################################################
                  // Available only if single precision floating point
                  // implementation selected as default
                  // ##############################################################

                  // output alpha component of the output vector should be fltOutAB.fltArg1 =
1
                  // output beta component of the output vector should be fltOutAB.fltArg2 =
0.764140062
                  GMCLIB_ElimDcBusRip (&fltOutAB,&fltAB,&flttrMyElimDcBusRip);
                }
```

4.106 Function GMCLIB_Park_F32

This function implements the calculation of Park transformation.

4.106.1 Declaration

```
void GMCLIB_Park_F32(SWLIBS_2Syst_F32 *pOut, const SWLIBS_2Syst_F32 *const pInAngle, const
SWLIBS_2Syst_F32 *const pIn);
```

4.106.2 Arguments

Table 4-131. GMCLIB_Park_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *	pOut	**input, output**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_F32 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β).

4.106.3 Return

void

4.106.4 Description

The GMCLIB_Park_F32 function calculates the Park Transformation, which transforms values (flux, voltage, current) from the two-phase (α- β) stationary orthogonal coordinate system to the two-phase (d-q) rotational orthogonal coordinate system, according to these equations:

$$d = \cos(\theta_e) \bullet \alpha + \sin(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq1**

$$q = -\sin(\theta_e) \bullet \alpha + \cos(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq2**

where θ_e represents the electrical position of the rotor flux.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.106.5 Re-entrancy

The function is re-entrant.

4.106.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 tr32Angle;
SWLIBS_2Syst_F32 tr32AlBe;
SWLIBS_2Syst_F32 tr32Dq;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr32Angle.f32Arg1  = FRAC32 (0.866025403);
  tr32Angle.f32Arg2  = FRAC32 (0.5);

  // input alpha = 0.123
  // input beta = 0.654
  tr32AlBe.f32Arg1  = FRAC32 (0.123);
  tr32AlBe.f32Arg2  = FRAC32 (0.654);

  // output should be
  // tr32Dq.f32Arg1 ~ d = 0x505E6455
  // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
  GMCLIB_Park_F32 (&tr32Dq,&tr32Angle,&tr32AlBe);

  // output should be
  // tr32Dq.f32Arg1 ~ d = 0x505E6455
  // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
  GMCLIB_Park (&tr32Dq,&tr32Angle,&tr32AlBe,Define F32);

  // ##################################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##################################################################

  // output should be
  // tr32Dq.f32Arg1 ~ d = 0x505E6455
  // tr32Dq.f32Arg2 ~ q = 0x1C38ABDC
  GMCLIB_Park (&tr32Dq,&tr32Angle,&tr32AlBe);
}
```

4.107 Function GMCLIB_Park_F16

This function implements the calculation of Park transformation.

4.107.1 Declaration

```
void GMCLIB_Park_F16(SWLIBS_2Syst_F16 *pOut, const SWLIBS_2Syst_F16 *const pInAngle, const
SWLIBS_2Syst_F16 *const pIn);
```

4.107.2 Arguments

Table 4-132. GMCLIB_Park_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *	pOut	**input, output**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_F16 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β).

4.107.3 Return

void

4.107.4 Description

The GMCLIB_Park_F16 function calculates the Park Transformation, which transforms values (flux, voltage, current) from the two-phase (α- β) stationary orthogonal coordinate system to the two-phase (d-q) rotational orthogonal coordinate system, according to these equations:

$$d = \cos(\theta_e) \bullet \alpha + \sin(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq1**

$$q = -\sin(\theta_e) \bullet \alpha + \cos(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq2**

where θ_e represents the electrical position of the rotor flux.

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.107.5 Re-entrancy

The function is re-entrant.

4.107.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 tr16Angle;
SWLIBS_2Syst_F16 tr16AlBe;
SWLIBS_2Syst_F16 tr16Dq;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr16Angle.f16Arg1  = FRAC16 (0.866025403);
  tr16Angle.f16Arg2  = FRAC16 (0.5);

  // input alpha = 0.123
  // input beta = 0.654
  tr16AlBe.f16Arg1  = FRAC16 (0.123);
  tr16AlBe.f16Arg2  = FRAC16 (0.654);

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park_F16 (&tr16Dq,&tr16Angle,&tr16AlBe);

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park (&tr16Dq,&tr16Angle,&tr16AlBe,Define F16);

  // ##################################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##################################################################

  // output should be
  // tr16Dq.f16Arg1 ~ d = 0x505E
  // tr16Dq.f16Arg2 ~ q = 0x1C38
  GMCLIB_Park (&tr16Dq,&tr16Angle,&tr16AlBe);
}
```

4.108 Function GMCLIB_Park_FLT

This function implements the calculation of Park transformation.

4.108.1 Declaration

```
void GMCLIB_Park_FLT(SWLIBS_2Syst_FLT *pOut, const SWLIBS_2Syst_FLT *const pInAngle, const
SWLIBS_2Syst_FLT *const pIn);
```

4.108.2 Arguments

Table 4-133. GMCLIB_Park_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *	pOut	**input, output**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).
const SWLIBS_2Syst_FLT *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β).

4.108.3 Return

void

4.108.4 Description

The GMCLIB_Park_FLT function calculates the Park Transformation, which transforms values (flux, voltage, current) from the two-phase (α- β) stationary orthogonal coordinate system to the two-phase (d-q) rotational orthogonal coordinate system, according to these equations:

$$d = \cos(\theta_e) \bullet \alpha + \sin(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq1**

$$q = -\sin(\theta_e) \bullet \alpha + \cos(\theta_e) \bullet \beta$$

Equation **GMCLIB_Park_Eq2**

where θ_e represents the electrical position of the rotor flux.

Note

The inputs and the outputs are in single precision floating point data format.

4.108.5 Re-entrancy

The function is re-entrant.

4.108.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT trfltAngle;
SWLIBS_2Syst_FLT trfltAlBe;
SWLIBS_2Syst_FLT trfltDq;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  trfltAngle.fltArg1  = 0.866025403;
  trfltAngle.fltArg2  = 0.5;

  // input alpha = 0.123
  // input beta = 0.654
  trfltAlBe.fltArg1  = 0.123;
  trfltAlBe.fltArg2  = 0.654;

  // output should be:
  // trfltDq.fltArg1 ~ d = 0.627880613
  // trfltDq.fltArg2 ~ q = 0.220479472
  GMCLIB_Park_FLT (&trfltDq,&trfltAngle,&trfltAlBe);

  // output should be:
  // trfltDq.fltArg1 ~ d = 0.627880613
  // trfltDq.fltArg2 ~ q = 0.220479472
  GMCLIB_Park (&trfltDq,&trfltAngle,&trfltAlBe,Define FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be:
  // trfltDq.fltArg1 ~ d = 0.627880613
  // trfltDq.fltArg2 ~ q = 0.220479472
  GMCLIB_Park (&trfltDq,&trfltAngle,&trfltAlBe);
}
```

4.109 Function GMCLIB_ParkInv_F32

This function implements the inverse Park transformation.

4.109.1 Declaration

```
void GMCLIB_ParkInv_F32(SWLIBS_2Syst_F32 *const pOut, const SWLIBS_2Syst_F32 *const pInAngle,
const SWLIBS_2Syst_F32 *const pIn);
```

4.109.2 Arguments

Table 4-134. GMCLIB_ParkInv_F32 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F32 *const	pOut	**input, output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β).
const SWLIBS_2Syst_F32 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

4.109.3 Return

void

4.109.4 Description

The GMCLIB_ParkInv_F32 function calculates the Inverse Park Transformation, which transforms quantities (flux, voltage, current) from the two-phase (d-q) rotational orthogonal coordinate system to the two-phase (α- β) stationary orthogonal coordinate system, according to these equations:

$$\alpha = \cos(\theta_e) \bullet d - \sin(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq1**

$$\beta = \sin(\theta_e) \bullet d + \cos(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq2**

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.109.5 Re-entrancy

The function is re-entrant.

4.109.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F32 tr32Angle;
SWLIBS_2Syst_F32 tr32Dq;
SWLIBS_2Syst_F32 tr32AlBe;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr32Angle.f32Arg1  = FRAC32 (0.866025403);
  tr32Angle.f32Arg2  = FRAC32 (0.5);

  // input d  = 0.123
  // input q  = 0.654
  tr32Dq.f32Arg1  = FRAC32 (0.123);
  tr32Dq.f32Arg2  = FRAC32 (0.654);

  // output should be
  // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
  // tr32AlBe.f32Arg2 ~ beta  = 0x377D9EE4
  GMCLIB_ParkInv_F32 (&tr32AlBe,&tr32Angle,&tr32Dq);

  // output should be
  // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
  // tr32AlBe.f32Arg2 ~ beta  = 0x377D9EE4
  GMCLIB_ParkInv (&tr32AlBe,&tr32Angle,&tr32Dq,Define F32);

  // ##################################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##################################################################

  // output should be
  // tr32AlBe.f32Arg1 ~ alpha = 0xBF601273
  // tr32AlBe.f32Arg2 ~ beta  = 0x377D9EE4
  GMCLIB_ParkInv (&tr32AlBe,&tr32Angle,&tr32Dq);
}
```

4.110 Function GMCLIB_ParkInv_F16

This function implements the inverse Park transformation.

4.110.1 Declaration

```
void GMCLIB_ParkInv_F16(SWLIBS_2Syst_F16 *const pOut, const SWLIBS_2Syst_F16 *const pInAngle,
const SWLIBS_2Syst_F16 *const pIn);
```

4.110.2 Arguments

Table 4-135. GMCLIB_ParkInv_F16 arguments

Type	Name	Direction	Description
SWLIBS_2Syst_F16 *const	pOut	**input, output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β).
const SWLIBS_2Syst_F16 *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

4.110.3 Return

void

4.110.4 Description

The GMCLIB_ParkInv_F16 function calculates the Inverse Park Transformation, which transforms quantities (flux, voltage, current) from the two-phase (d-q) rotational orthogonal coordinate system to the two-phase (α- β) stationary orthogonal coordinate system, according to these equations:

$$\alpha = \cos(\theta_e) \bullet d - \sin(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq1**

$$\beta = \sin(\theta_e) \bullet d + \cos(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq2**

Note

The inputs and the outputs are normalized to fit in the range [-1, 1).

4.110.5 Re-entrancy

The function is re-entrant.

4.110.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_F16 tr16Angle;
SWLIBS_2Syst_F16 tr16Dq;
SWLIBS_2Syst_F16 tr16AlBe;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  tr16Angle.f16Arg1  = FRAC16 (0.866025403);
  tr16Angle.f16Arg2  = FRAC16 (0.5);

  // input d  = 0.123
  // input q  = 0.654
  tr16Dq.f16Arg1  = FRAC16 (0.123);
  tr16Dq.f16Arg2  = FRAC16 (0.654);

  // output should be
  // tr16AlBe.f16Arg1 ~ alpha = 0xBF60
  // tr16AlBe.f16Arg2 ~ beta  = 0x377D
  GMCLIB_ParkInv_F16 (&tr16AlBe,&tr16Angle,&tr16Dq);

  // output should be
  // tr16AlBe.f16Arg1 ~ alpha = 0xBF60
  // tr16AlBe.f16Arg2 ~ beta  = 0x377D
  GMCLIB_ParkInv (&tr16AlBe,&tr16Angle,&tr16Dq,Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be
  // tr16AlBe.f16Arg1 ~ alpha = 0xBF60
  // tr16AlBe.f16Arg2 ~ beta  = 0x377D
  GMCLIB_ParkInv (&tr16AlBe,&tr16Angle,&tr16Dq);
}
```

4.111 Function GMCLIB_ParkInv_FLT

This function implements the inverse Park transformation.

4.111.1 Declaration

```
void GMCLIB_ParkInv_FLT(SWLIBS_2Syst_FLT *const pOut, const SWLIBS_2Syst_FLT *const pInAngle,
const SWLIBS_2Syst_FLT *const pIn);
```

4.111.2 Arguments

Table 4-136. GMCLIB_ParkInv_FLT arguments

Type	Name	Direction	Description
SWLIBS_2Syst_FLT *const	pOut	**input, output**	Pointer to the structure containing data of the two-phase stationary orthogonal system (α− β).
const SWLIBS_2Syst_FLT *const	pInAngle	**input**	Pointer to the structure where the values of the sine and cosine of the rotor position are stored.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing data of the two-phase rotational orthogonal system (d-q).

4.111.3 Return

void

4.111.4 Description

The GMCLIB_ParkInv_FLT function calculates the Inverse Park Transformation, which transforms quantities (flux, voltage, current) from the two-phase (d-q) rotational orthogonal coordinate system to the two-phase (α- β) stationary orthogonal coordinate system, according to these equations:

$$\alpha = \cos(\theta_e) \bullet d - \sin(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq1**

$$\beta = \sin(\theta_e) \bullet d + \cos(\theta_e) \bullet q$$

Equation **GMCLIB_ParkInv_Eq2**

Note

The inputs and the outputs are in single precision floating point data format.

4.111.5 Re-entrancy

The function is re-entrant.

4.111.6 Code Example

```
#include "gmclib.h"

SWLIBS_2Syst_FLT trfltAngle;
SWLIBS_2Syst_FLT trfltDq;
SWLIBS_2Syst_FLT trfltAlBe;

void main(void)
{
  // input angle sin(60) = 0.866025403
  // input angle cos(60) = 0.5
  trfltAngle.fltArg1  = 0.866025403;
  trfltAngle.fltArg2  = 0.5;

  // input d  = 0.123
  // input q  = 0.654
  trfltDq.fltArg1  = 0.123;
  trfltDq.fltArg2  = 0.654;

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv_FLT (&trfltAlBe,&trfltAngle,&trfltDq);

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv (&trfltAlBe,&trfltAngle,&trfltDq,Define FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output should be:
  // trfltAlBe.fltArg1 ~ alpha = -0.495119387
  // trfltAlBe.fltArg2 ~ beta  = 0.433521139
  GMCLIB_ParkInv (&trfltAlBe,&trfltAngle,&trfltDq);
}
```

4.112 Function GMCLIB_SvmStd_F32

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

4.112.1 Declaration

```
tU32 GMCLIB_SvmStd_F32(SWLIBS_3Syst_F32 *pOut, const SWLIBS_2Syst_F32 *const pIn);
```

4.112.2 Arguments

Table 4-137. GMCLIB_SvmStd_F32 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F32 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F32 *const	pIn	**input**	Pointer to the structure containing direct U_{α} and quadrature U_{β} components of the stator voltage vector.

4.112.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

4.112.4 Description

The GMCLIB_SvmStd_F32 function for calculating duty-cycle ratios is widely-used in the modern electric drive. This, function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Standard Space Vector Modulation. The basic principle of the Standard Space Vector Modulation Technique can be explained with the help of the power stage diagram in Figure 4-49.

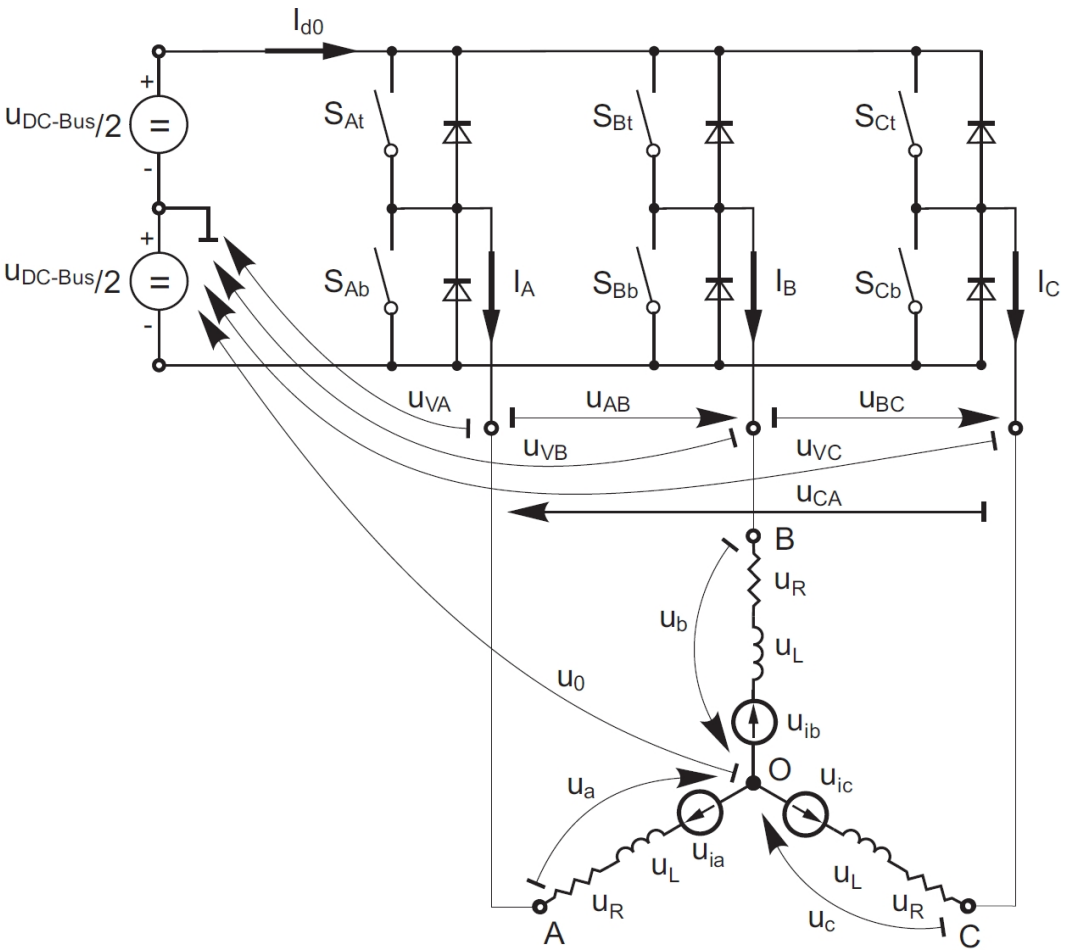

Figure 4-49. Power stage schematic diagram

Top and bottom switches work in a complementary mode; i.e., if the top switch, S_{At}, is ON, then the corresponding bottom switch, S_{Ab}, is OFF, and vice versa. Considering that value 1 is assigned to the ON state of the top switch, and value 0 is assigned to the ON state of the bottom switch, the switching vector, $[a, b, c]^T$ can be defined. Creating such a vector allows a numerical definition of all possible switching states. In a three-phase power stage configuration (as shown in Figure 4-49), eight possible switching states (detailed in Figure 4-50) are feasible.

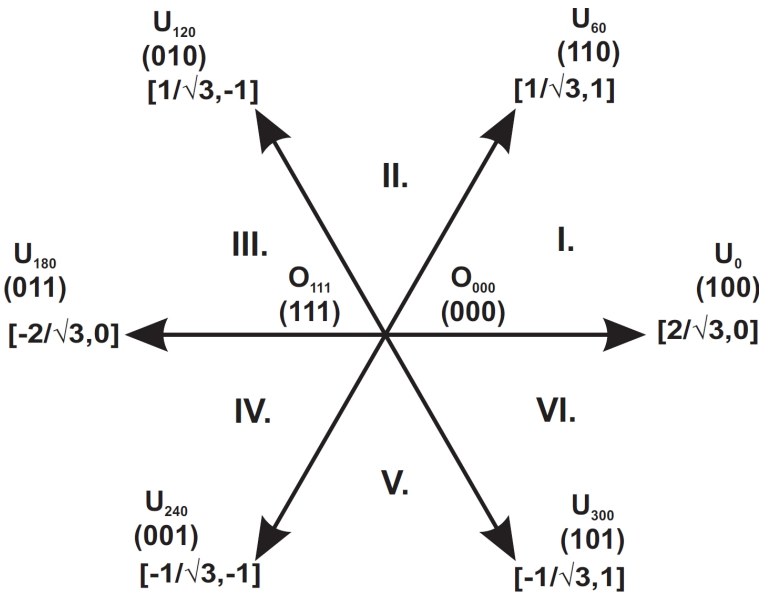

Figure 4-50. Basic space vectors

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in Table 4-138.

Table 4-138. Switching patterns

a	b	c	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	2/3*U_{DCBus}	-1/3*U_{DCBus}	-1/3*U_{DCBus}	U_{DCBus}	0	-U_{DCBus}	U_0
1	1	0	1/3*U_{DCBus}	1/3*U_{DCBus}	-2/3*U_{DCBus}	0	U_{DCBus}	-U_{DCBus}	U_{60}
0	1	0	-1/3*U_{DCBus}	2/3*U_{DCBus}	-1/3*U_{DCBus}	-U_{DCBus}	U_{DCBus}	0	U_{120}
0	1	1	-2/3*U_{DCBus}	1/3*U_{DCBus}	1/3*U_{DCBus}	-U_{DCBus}	0	U_{DCBus}	U_{240}
0	0	1	-1/3*U_{DCBus}	-1/3*U_{DCBus}	2/3*U_{DCBus}	0	-U_{DCBus}	U_{DCBus}	U_{300}
1	0	1	1/3*U_{DCBus}	-2/3*U_{DCBus}	1/3*U_{DCBus}	U_{DCBus}	-U_{DCBus}	0	U_{360}
1	1	1	0	0	0	0	0	0	O_{111}

The quantities of the direct-u_α and the quadrature-u_β components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke Transformation.

$$U_\alpha = \frac{2}{3}\left(U_a - \frac{U_b}{2} - \frac{U_c}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq1**

$$U_\beta = \frac{2}{3}\left(0 + \frac{\sqrt{3}U_b}{2} - \frac{\sqrt{3}U_c}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq2**

The three-phase stator voltages, U_a, U_b, and U_c, are transformed using the Clarke Transformation into the U_α and the U_β components of the two-phase orthogonal coordinate system. The transformation results are listed in Table 4-139.

Table 4-139. Switching patterns and space vectors

a	b	c	u_α	u_β	Vector
0	0	0	0	0	O_{000}
1	0	0	2/3*U_{DCBus}	0	U_0
1	1	0	1/3*U_{DCBus}	1/ √(3)*U_{DCBus}	U_{60}
0	1	0	-1/3*U_{DCBus}	1/ √(3)*U_{DCBus}	U_{120}
0	1	1	-2/3*U_{DCBus}	0	U_{240}
0	0	1	-1/3*U_{DCBus}	-1/ √(3)*U_{DCBus}	U_{300}
1	0	1	1/3*U_{DCBus}	-1/ √(3)*U_{DCBus}	U_{360}
1	1	1	0	0	O_{111}

Figure 4-50 graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors U_0, U_{60}, U_{120}, U_{240}, U_{300}, and two zero vectors O_{111}, O_{000}, usable for switching. Therefore, the principle of the Standard Space Vector Modulation resides in applying appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

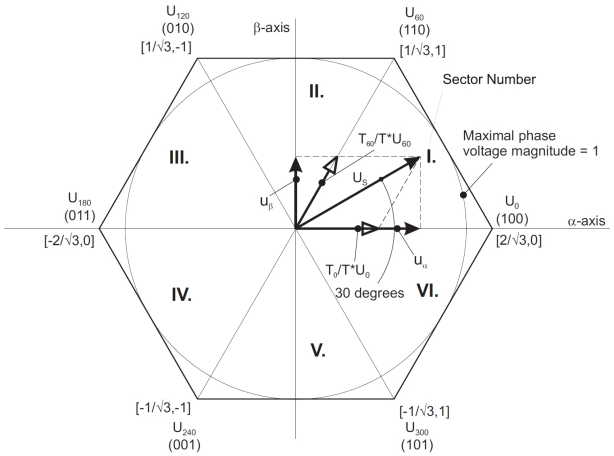

Figure 4-51. Projection of reference voltage vector in sector I

Referring to that principle, an objective of the Standard Space Vector Modulation is an approximation of the reference stator voltage vector U_S with an appropriate combination of the switching patterns composed of basic space vectors. The graphical explanation of this objective is shown in Figure 4-51 and Figure 4-52.

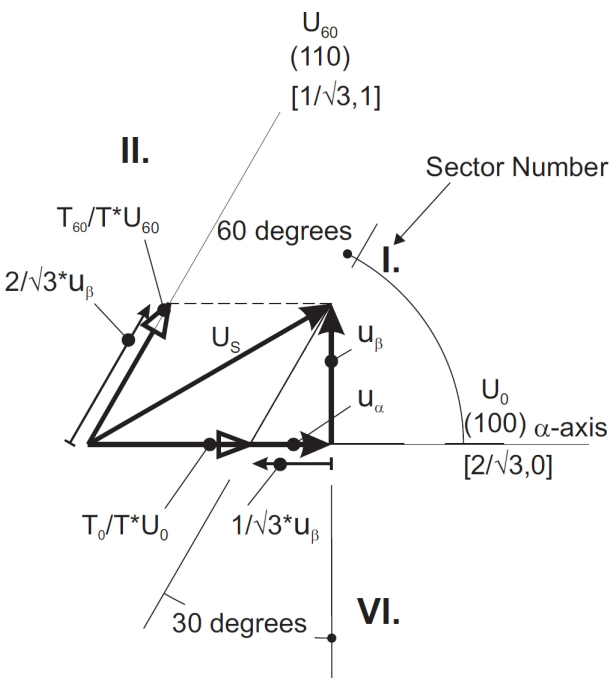

Figure 4-52. Detail of the voltage vector projection in sector I

The stator reference voltage vector U_S is phase-advanced by 30 ° from the axis- α and thus might be generated with an appropriate combination of the adjacent basic switching states U_0 and $_{60}$. These figures also indicate the resultant U_α and U_β components for space vectors U_0 and U_{60}

In this case, the reference stator voltage vector U_S is located in Sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states U_{60} and U_0. The principal equations concerning this vector location are:

$$T = T_{60} + T_0 + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq3**

$$U_S = \frac{T_{60}}{T} U_{60} + \frac{T_0}{T} U_0$$

Equation **GMCLIB_SvmStd_Eq4**

where T_{60} and T_0 are the respective duty-cycle ratios for which the basic space vectors U_{60} and U_0 should be applied within the time period T. T_{null} is the course of time for which the null vectors O_{000} and O_{111} are applied. Those duty-cycle ratios can be calculated using equations:

$$u_\beta = \frac{T_{60}}{\text{T}} |U_{60}| \sin\ 60^\circ$$

Equation **GMCLIB_SvmStd_Eq5**

$$u_\alpha = \frac{T_0}{T} |U_0| + \frac{u_\beta}{\tan 60^\circ}$$

Equation **GMCLIB_SvmStd_Eq6**

Considering that the normalized magnitudes of the basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{(3)}$ and by substitution of the trigonometric expressions sin(60 °) and tan(60 °) by their quantities $2/\sqrt{(3)}$ and $\sqrt{(3)}$, respectively, equation GMCLIB_SvmStd_Eq5 and equation GMCLIB_SvmStd_Eq6 can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T:

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

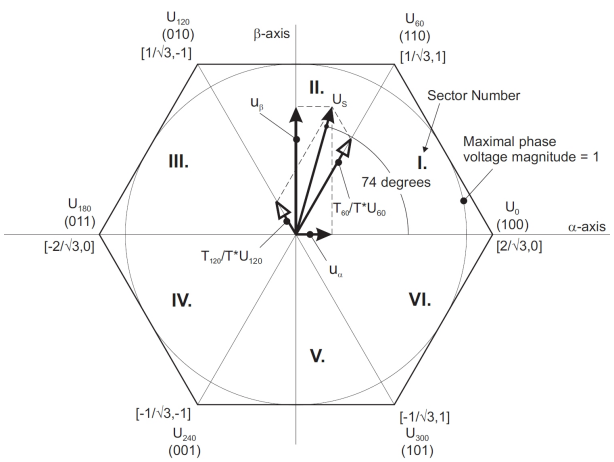

Figure 4-53. Projection of the reference voltage vector in sector II

Sector II is depicted in Figure 4-53. In this particular case, the reference stator voltage vector U $_S$ is generated by the appropriate duty-cycle ratios of the basic switching states U $_{60}$ and $_{120}$. The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq9**

$$U_S = \frac{T_{120}}{T} U_{120} + \frac{T_{60}}{T} U_{60}$$

Equation **GMCLIB_SvmStd_Eq10**

where T $_{120}$ and T $_{60}$ are the respective duty-cycle ratios for which the basic space vectors U $_{120}$ and U $_{60}$ should be applied within the time period T. These resultant duty-cycle ratios are formed from the auxiliary components termed A and B. The graphical representation of the auxiliary components is shown in Figure 4-54.

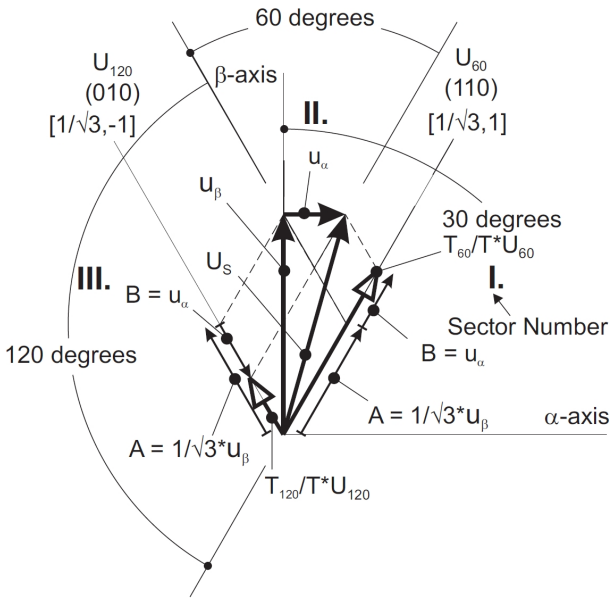

Figure 4-54. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30^\circ}{\sin 120^\circ} = \frac{A}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq11**

$$\frac{\sin 60^\circ}{\sin 120^\circ} = \frac{B}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq12**

Equation GMCLIB_SvmStd_Eq11 and equation GMCLIB_SvmStd_Eq12 have been formed using the sine rule. These equations can be rearranged for the calculation of the auxiliary time-duration components A and B. This is done simply by substitution of the trigonometric terms sin(30 °), sin(120 °) and sin(60 °) by their numerical representations 1/2, √(3)/2 and 1/ √(3), respectively.

$$A = \frac{1}{\sqrt{3}} u_\beta$$

Equation **GMCLIB_SvmStd_Eq13**

$$B = u_\alpha$$

Equation **GMCLIB_SvmStd_Eq14**

The resultant duty-cycle ratios, T_{120}/T and T_{60}/T, are then expressed in terms of the auxiliary time-duration components defined by equation GMCLIB_SvmStd_Eq13 and equation GMCLIB_SvmStd_Eq14, as follows:

$$\frac{T_{120}}{T} |U_{120}| = A - B$$

Equation **GMCLIB_SvmStd_Eq15**

$$\frac{T_{60}}{T} |U_{60}| = A + B$$

Equation **GMCLIB_SvmStd_Eq16**

With the help of these equations, and also considering the normalized magnitudes of the basic space vectors to be $|U_{120}| = |U_{60}| = 2/(3)$, the equations expressed for the unknown duty-cycle ratios of basic space vectors T_{120}/T and T_{60}/T can be written:

$$\frac{T_{120}}{T} = \frac{1}{2}\left(u_\beta - \sqrt{3}\, u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq17**

$$\frac{T_{60}}{T} = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq18**

The duty-cycle ratios in remaining sectors can be derived using the same approach. The resulting equations will be similar to those derived for Sector I and Sector II.

To depict duty-cycle ratios of the basic space vectors for all sectors, we define:

- Three auxiliary variables:

$$X = u_\beta$$

Equation **GMCLIB_SvmStd_Eq19**

$$Y = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq20**

$$Z = \frac{1}{2}\left(u_\beta - \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq21**

Two expressions t_1 and t_2 generally represent duty-cycle ratios of the basic space vectors in the respective sector; e.g., for the first sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{60} and U_{0}; for the second sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{120} and U_{60}, etc.

For each sector, the expressions t_1 and t_2, in terms of auxiliary variables X, Y and Z, are listed in Table 4-140.

Table 4-140. Determination of t_1 and t_2 expressions

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
t_1	X	Y	-Y	Z	-Z	-X
t_2	-Z	Z	X	-X	-Y	Y

For the determination of auxiliary variables X equation GMCLIB_SvmStd_Eq19, Y equation GMCLIB_SvmStd_Eq20 and Z equation GMCLIB_SvmStd_Eq21, the sector number is required. This information can be obtained by several approaches. One approach discussed here requires the use of a modified Inverse Clark Transformation to transform the direct- α and quadrature- components into a balanced three-phase quantity u_{ref1}, u_{ref2} and u_{ref3}, used for a straightforward calculation of the sector number, to be shown later.

$$u_{ref1} = u_{\beta}$$

Equation **GMCLIB_SvmStd_Eq22**

$$u_{ref2} = \frac{1}{2}\left(-u_{\beta} + \sqrt{3}\,u_{\alpha}\right)$$

Equation **GMCLIB_SvmStd_Eq23**

$$u_{ref3} = \frac{1}{2}\left(-u_{\beta} - \sqrt{3}\,u_{\alpha}\right)$$

Equation **GMCLIB_SvmStd_Eq24**

The modified Inverse Clark Transformation projects the quadrature-u_{β} component into u_{ref1}, as shown in Figure 4-55 and Figure 4-56, whereas voltages generated by the conventional Inverse Clark Transformation project the u_{α} component into u_{ref1}.

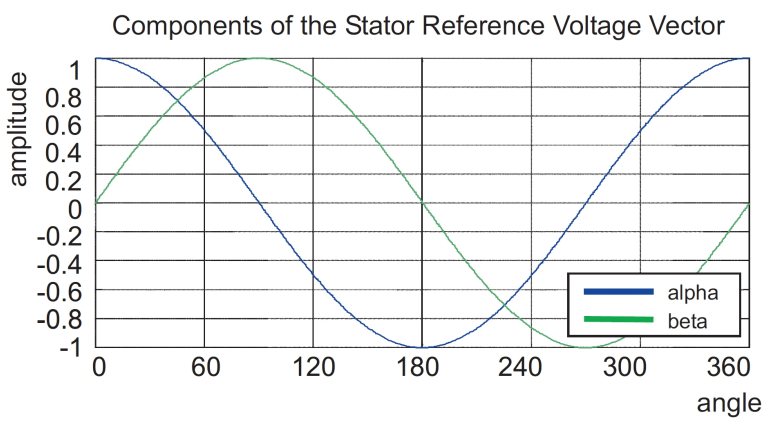

Figure 4-55. Direct-ua and quadrature-ub components of stator reference voltage

Figure 4-55 depicts the u_{α} and u_{β} components of the stator reference voltage vector U_S that were calculated by the equations $u_{\alpha} = \cos(\theta)$ and $u_{\beta} = \sin(\theta)$, respectively.

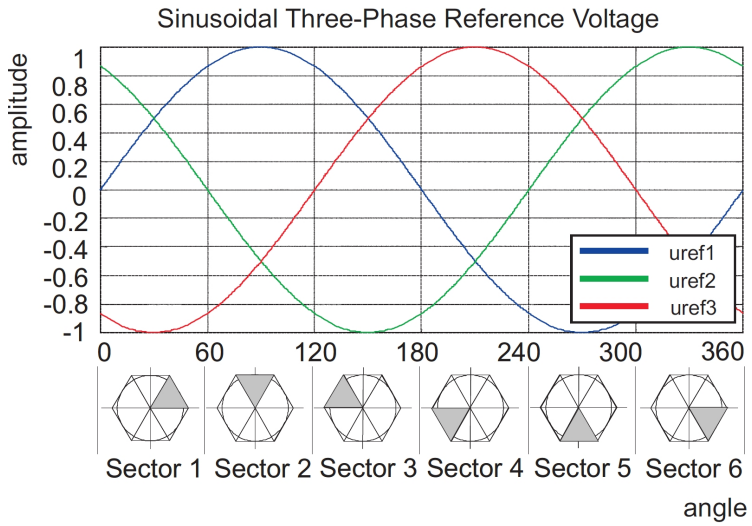

Figure 4-56. Reference Voltages u_ref1, u_ref2 and u_ref3

The Sector Identification Tree, shown in Figure 4-57, can be a numerical solution of the approach shown in Figure 4-56.

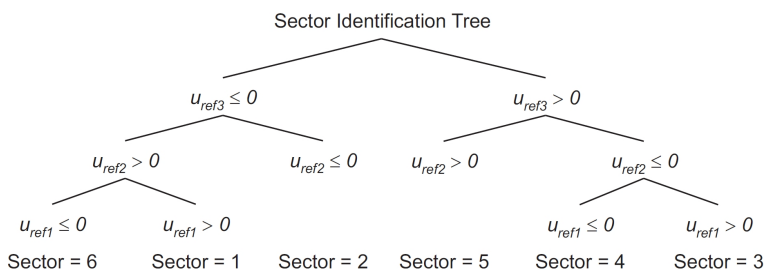

Figure 4-57. Identification of the sector number

It should be pointed out that, in the worst case, three simple comparisons are required to precisely identify the sector of the stator reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 4-51, the stator reference voltage vector is phase-advanced by 30 ° from the α-axis, which results in the positive quantities of u_{ref1} and u_{ref2} and the negative quantity of u_{ref3}; refer to Figure 4-56. If these quantities are used as the inputs to the Sector Identification Tree, the product of those comparisons will be Sector I. Using the same approach identifies Sector II, if the stator reference voltage vector is located according to the one shown in Figure 4-54. The variables t_1, t_2 and t_3, representing the switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

$$t_1 = \frac{T - t\_1 - t\_2}{2}$$

Equation **GMCLIB_SvmStd_Eq25**

$$t_2 = t_1 + t\_1$$

Equation **GMCLIB_SvmStd_Eq26**

$$t_3 = t_2 + t\_2$$

Equation **GMCLIB_SvmStd_Eq27**

where T is the switching period, t_1 and t_2 are the duty-cycle ratios (see Table 4-140) of the basic space vectors, given for the respective sector. Equation GMCLIB_SvmStd_Eq25, equation GMCLIB_SvmStd_Eq26 and equation GMCLIB_SvmStd_Eq27 are specific solely to the Standard Space Vector Modulation technique; consequently, other Space Vector Modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, t_1, t_2 and t_3, to the respective motor phases. This is a simple task, accomplished in view of the position of the stator reference voltage vector as shown in Table 4-141.

Table 4-141. Assignment of the duty-cycle ratios to motor phases

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
pwm_a	t_3	t_2	t_1	t_1	t_2	t_3
pwm_b	t_2	t_3	t_3	t_2	t_1	t_1
pwm_c	t_1	t_1	t_2	t_3	t_3	t_2

The principle of the Space Vector Modulation technique consists in applying the basic voltage vectors U_{XXX} and O_{XXX} for the certain time in such a way that the mean vector, generated by the Pulse Width Modulation approach for the period T, is equal to the original stator reference voltage vector U_S. This provides a great variability of the arrangement of the basic vectors during the PWM period T. Those vectors might be arranged either to lower switching losses or to achieve diverse results, such as centre-aligned PWM, edge-aligned PWM or a minimal number of switching states. A brief discussion of the widely-used centre-aligned PWM follows. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels,

pwm $_a$, pwm $_b$ and pwm $_c$ with a free-running up-down counter. The timer counts to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 4-58

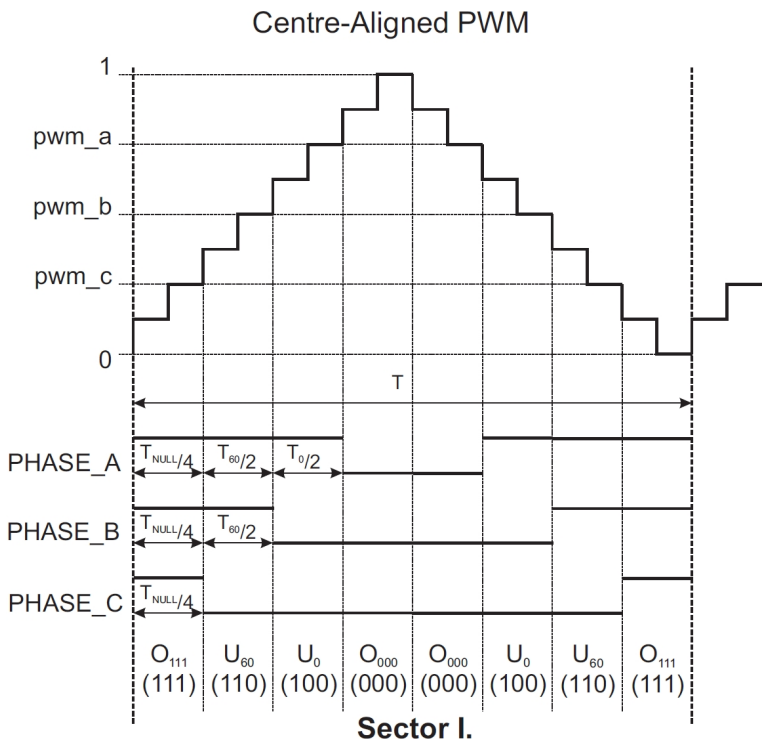

Figure 4-58. Standard space vector modulation technique - centre-aligned PWM

4.112.5 Re-entrancy

The function is re-entrant.

4.112.6 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F32 tr32InVoltage;
SWLIBS_3Syst_F32 tr32PwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr32InVoltage.f32Arg1       = FRAC32 (12.99/U_MAX);
  tr32InVoltage.f32Arg2       = FRAC32 (7.5/U_MAX);
```

```
  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle    = 0x7FFF A2C9 = FRAC32(0.9999888... )
  // pwmb dutycycle    = 0x4000 5D35 = FRAC32(0.5000111... )
  // pwmc dutycycle    = 0x0000 5D35 = FRAC32(0.0000111... )
  // svmSector         = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd_F32 (&tr32PwmABC,&tr32InVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle    = 0x7FFF A2C9 = FRAC32(0.9999888... )
  // pwmb dutycycle    = 0x4000 5D35 = FRAC32(0.5000111... )
  // pwmc dutycycle    = 0x0000 5D35 = FRAC32(0.0000111... )
  // svmSector         = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd (&tr32PwmABC,&tr32InVoltage,Define F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle    = 0x7FFF A2C9 = FRAC32(0.9999888... )
  // pwmb dutycycle    = 0x4000 5D35 = FRAC32(0.5000111... )
  // pwmc dutycycle    = 0x0000 5D35 = FRAC32(0.0000111... )
  // svmSector         = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd (&tr32PwmABC,&tr32InVoltage);
}
```

4.113 Function GMCLIB_SvmStd_F16

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

4.113.1 Declaration

```
tU16 GMCLIB_SvmStd_F16(SWLIBS_3Syst_F16 *pOut, const SWLIBS_2Syst_F16 *const pIn);
```

4.113.2 Arguments

Table 4-142. GMCLIB_SvmStd_F16 arguments

Type	Name	Direction	Description
SWLIBS_3Syst_F16 *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_F16 *const	pIn	**input**	Pointer to the structure containing direct U_{α} and quadrature U_{β} components of the stator voltage vector.

4.113.3 Return

The function returns a 16-bit value in format INT, representing the actual space sector which contains the stator reference vector U $_s$.

4.113.4 Description

The GMCLIB_SvmStd_F16 function for calculating duty-cycle ratios is widely-used in the modern electric drive. This, function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Standard Space Vector Modulation. The basic principle of the Standard Space Vector Modulation Technique can be explained with the help of the power stage diagram in Figure 4-59.

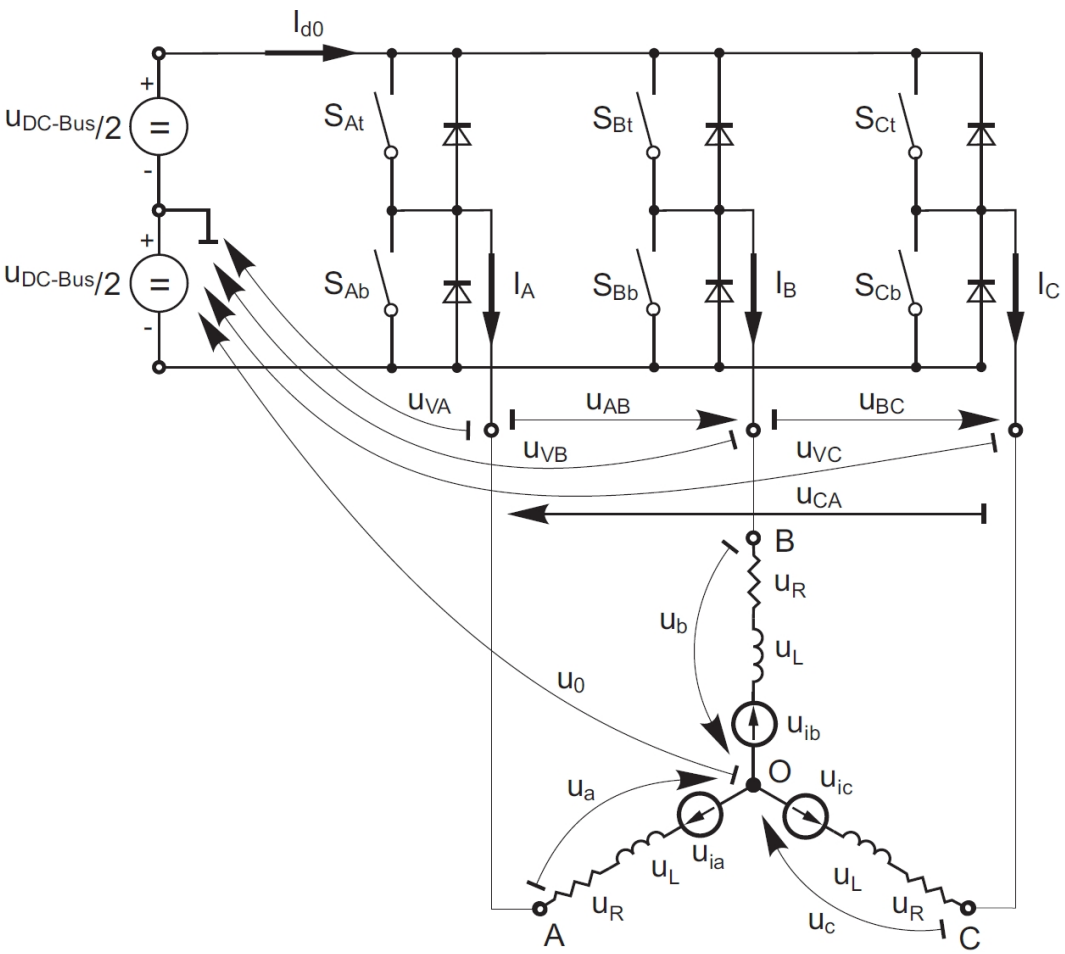

Figure 4-59. Power stage schematic diagram

Top and bottom switches work in a complementary mode; i.e., if the top switch, S_{At}, is ON, then the corresponding bottom switch, S_{Ab}, is OFF, and vice versa. Considering that value 1 is assigned to the ON state of the top switch, and value 0 is assigned to the ON state of the bottom switch, the switching vector, $[a, b, c]^T$ can be defined. Creating such a vector allows a numerical definition of all possible switching states. In a three-phase power stage configuration (as shown in Figure 4-59), eight possible switching states (detailed in Figure 4-60) are feasible.

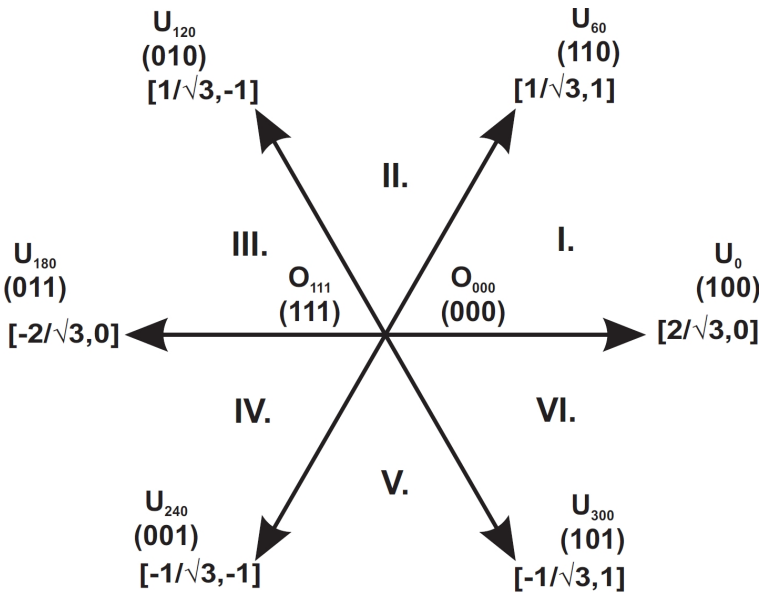

Figure 4-60. Basic space vectors

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in Table 4-143.

Table 4-143. Switching patterns

a	b	c	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	2/3*U_{DCBus}	-1/3*U_{DCBus}	-1/3*U_{DCBus}	U_{DCBus}	0	-U_{DCBus}	U_0
1	1	0	1/3*U_{DCBus}	1/3*U_{DCBus}	-2/3*U_{DCBus}	0	U_{DCBus}	-U_{DCBus}	U_{60}
0	1	0	-1/3*U_{DCBus}	2/3*U_{DCBus}	-1/3*U_{DCBus}	-U_{DCBus}	U_{DCBus}	0	U_{120}
0	1	1	-2/3*U_{DCBus}	1/3*U_{DCBus}	1/3*U_{DCBus}	-U_{DCBus}	0	U_{DCBus}	U_{240}
0	0	1	-1/3*U_{DCBus}	-1/3*U_{DCBus}	2/3*U_{DCBus}	0	-U_{DCBus}	U_{DCBus}	U_{300}
1	0	1	1/3*U_{DCBus}	-2/3*U_{DCBus}	1/3*U_{DCBus}	U_{DCBus}	-U_{DCBus}	0	U_{360}
1	1	1	0	0	0	0	0	0	O_{111}

The quantities of the direct-u $_{\alpha}$ and the quadrature-u $_{\beta}$ components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke Transformation.

$$U_{\alpha} = \frac{2}{3}\left(U_a - \frac{U_b}{2} - \frac{U_c}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq1**

$$U_{\beta} = \frac{2}{3}\left(0 + \frac{\sqrt{3}U_b}{2} - \frac{\sqrt{3}U_C}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq2**

The three-phase stator voltages,U $_{a}$, U $_{b}$, and U $_{c}$, are transformed using the Clarke Transformation into the U $_{\alpha}$ and the U $_{\beta}$ components of the two-phase orthogonal coordinate system. The transformation results are listed in Table 4-144.

Table 4-144. Switching patterns and space vectors

a	b	c	u $_{\alpha}$	u $_{\beta}$	Vector
0	0	0	0	0	O $_{000}$
1	0	0	2/3*U $_{DCBus}$	0	U $_{0}$
1	1	0	1/3*U $_{DCBus}$	1/ √(3)*U $_{DCBus}$	U $_{60}$
0	1	0	-1/3*U $_{DCBus}$	1/ √(3)*U $_{DCBus}$	U $_{120}$
0	1	1	-2/3*U $_{DCBus}$	0	U $_{240}$
0	0	1	-1/3*U $_{DCBus}$	-1/ √(3)*U $_{DCBus}$	U $_{300}$
1	0	1	1/3*U $_{DCBus}$	-1/ √(3)*U $_{DCBus}$	U $_{360}$
1	1	1	0	0	O $_{111}$

Figure 4-60 graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors U $_{0}$, U $_{60}$, U $_{120}$, U $_{240}$. U $_{300}$, and two zero vectors O $_{111}$, O $_{000}$, usable for switching. Therefore, the principle of the Standard Space Vector Modulation resides in applying appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

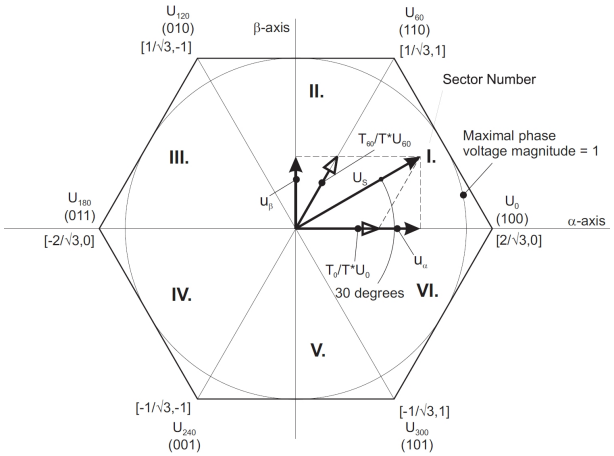

Figure 4-61. Projection of reference voltage vector in sector I

Referring to that principle, an objective of the Standard Space Vector Modulation is an approximation of the reference stator voltage vector U_S with an appropriate combination of the switching patterns composed of basic space vectors. The graphical explanation of this objective is shown in Figure 4-61 and Figure 4-62.

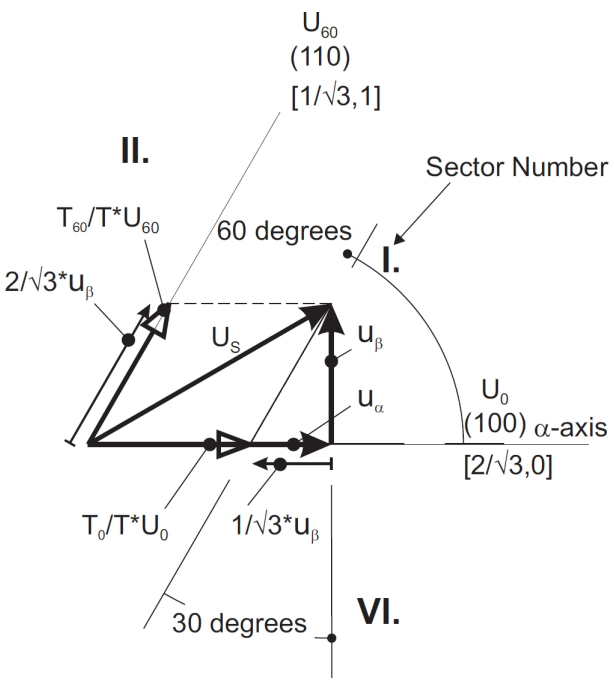

Figure 4-62. Detail of the voltage vector projection in sector I

The stator reference voltage vector U_S is phase-advanced by 30 ° from the axis- α and thus might be generated with an appropriate combination of the adjacent basic switching states U_0 and $_{60}$. These figures also indicate the resultant U_α and U_β components for space vectors U_0 and U_{60}

In this case, the reference stator voltage vector U_S is located in Sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states U_{60} and U_0. The principal equations concerning this vector location are:

$$T = T_{60} + T_0 + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq3**

$$U_S = \frac{T_{60}}{T} U_{60} + \frac{T_0}{T} U_0$$

Equation **GMCLIB_SvmStd_Eq4**

where T_{60} and T_0 are the respective duty-cycle ratios for which the basic space vectors U_{60} and U_0 should be applied within the time period T. T_{null} is the course of time for which the null vectors O_{000} and O_{111} are applied. Those duty-cycle ratios can be calculated using equations:

$$u_\beta = \frac{T_{60}}{\text{T}} |U_{60}| \sin\ 60^\circ$$

Equation **GMCLIB_SvmStd_Eq5**

$$u_\alpha = \frac{T_0}{T} |U_0| + \frac{u_\beta}{\tan 60^\circ}$$

Equation **GMCLIB_SvmStd_Eq6**

Considering that the normalized magnitudes of the basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{(3)}$ and by substitution of the trigonometric expressions sin(60 °) and tan(60 °) by their quantities $2/\sqrt{(3)}$ and $\sqrt{(3)}$, respectively, equation GMCLIB_SvmStd_Eq5 and equation GMCLIB_SvmStd_Eq6 can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T:

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

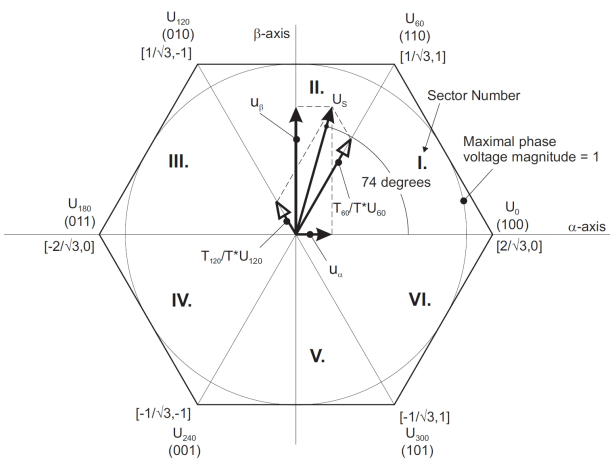

Figure 4-63. Projection of the reference voltage vector in sector II

Sector II is depicted in Figure 4-63. In this particular case, the reference stator voltage vector U_S is generated by the appropriate duty-cycle ratios of the basic switching states U_{60} and U_{120}. The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq9**

$$U_S = \frac{T_{120}}{T} U_{120} + \frac{T_{60}}{T} U_{60}$$

Equation **GMCLIB_SvmStd_Eq10**

where T $_{120}$ and T $_{60}$ are the respective duty-cycle ratios for which the basic space vectors U $_{120}$ and U $_{60}$ should be applied within the time period T. These resultant duty-cycle ratios are formed from the auxiliary components termed A and B. The graphical representation of the auxiliary components is shown in Figure 4-64.

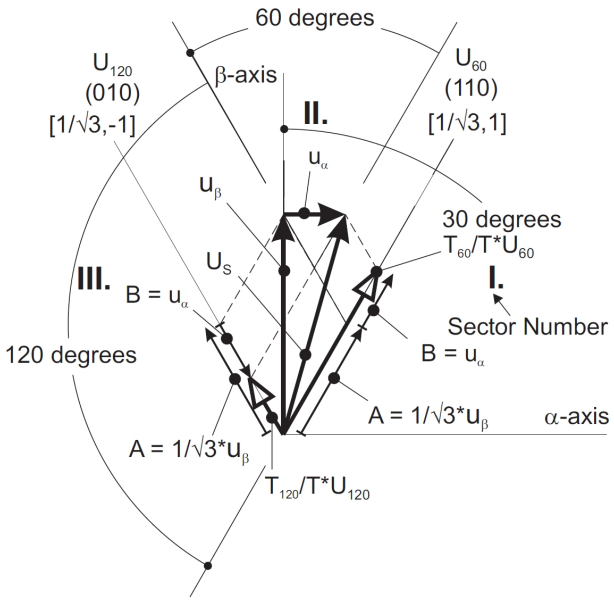

Figure 4-64. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30^\circ}{\sin 120^\circ} = \frac{A}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq11**

$$\frac{\sin 60^\circ}{\sin 120^\circ} = \frac{B}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq12**

Equation GMCLIB_SvmStd_Eq11 and equation GMCLIB_SvmStd_Eq12 have been formed using the sine rule. These equations can be rearranged for the calculation of the auxiliary time-duration components A and B. This is done simply by substitution of the trigonometric terms sin(30 °), sin(120 °) and sin(60 °) by their numerical representations 1/2, √(3)/2 and 1/ √(3), respectively.

$$A = \frac{1}{\sqrt{3}} u_\beta$$

Equation **GMCLIB_SvmStd_Eq13**

$$B = u_\alpha$$

Equation **GMCLIB_SvmStd_Eq14**

The resultant duty-cycle ratios, T_{120}/T and T_{60}/T, are then expressed in terms of the auxiliary time-duration components defined by equation GMCLIB_SvmStd_Eq13 and equation GMCLIB_SvmStd_Eq14, as follows:

$$\frac{T_{120}}{T} |U_{120}| = A - B$$

Equation **GMCLIB_SvmStd_Eq15**

$$\frac{T_{60}}{T} |U_{60}| = A + B$$

Equation **GMCLIB_SvmStd_Eq16**

With the help of these equations, and also considering the normalized magnitudes of the basic space vectors to be $|U_{120}| = |U_{60}| = 2/(3)$, the equations expressed for the unknown duty-cycle ratios of basic space vectors T_{120}/T and T_{60}/T can be written:

$$\frac{T_{120}}{T} = \frac{1}{2}\left(u_\beta - \sqrt{3}\, u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq17**

$$\frac{T_{60}}{T} = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq18**

The duty-cycle ratios in remaining sectors can be derived using the same approach. The resulting equations will be similar to those derived for Sector I and Sector II.

To depict duty-cycle ratios of the basic space vectors for all sectors, we define:

- Three auxiliary variables:

$$X = u_\beta$$

Equation **GMCLIB_SvmStd_Eq19**

$$Y = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq20**

$$Z = \frac{1}{2}\left(u_\beta - \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq21**

Two expressions t_1 and t_2 generally represent duty-cycle ratios of the basic space vectors in the respective sector; e.g., for the first sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{60} and U_{0}; for the second sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{120} and U_{60}, etc.

For each sector, the expressions t_1 and t_2, in terms of auxiliary variables X, Y and Z, are listed in Table 4-145.

Table 4-145. Determination of t_1 and t_2 expressions

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
t_1	X	Y	-Y	Z	-Z	-X
t_2	-Z	Z	X	-X	-Y	Y

For the determination of auxiliary variables X equation GMCLIB_SvmStd_Eq19, Y equation GMCLIB_SvmStd_Eq20 and Z equation GMCLIB_SvmStd_Eq21, the sector number is required. This information can be obtained by several approaches. One approach discussed here requires the use of a modified Inverse Clark Transformation to transform the direct- α and quadrature- components into a balanced three-phase quantity u_{ref1}, u_{ref2} and u_{ref3}, used for a straightforward calculation of the sector number, to be shown later.

$$u_{ref1} = u_{\beta}$$

Equation **GMCLIB_SvmStd_Eq22**

$$u_{ref2} = \frac{1}{2}\left(-u_{\beta} + \sqrt{3}\,u_{\alpha}\right)$$

Equation **GMCLIB_SvmStd_Eq23**

$$u_{ref3} = \frac{1}{2}\left(-u_{\beta} - \sqrt{3}\,u_{\alpha}\right)$$

Equation **GMCLIB_SvmStd_Eq24**

The modified Inverse Clark Transformation projects the quadrature-u_{β} component into u_{ref1}, as shown in Figure 4-65 and Figure 4-66, whereas voltages generated by the conventional Inverse Clark Transformation project the u_{α} component into u_{ref1}.

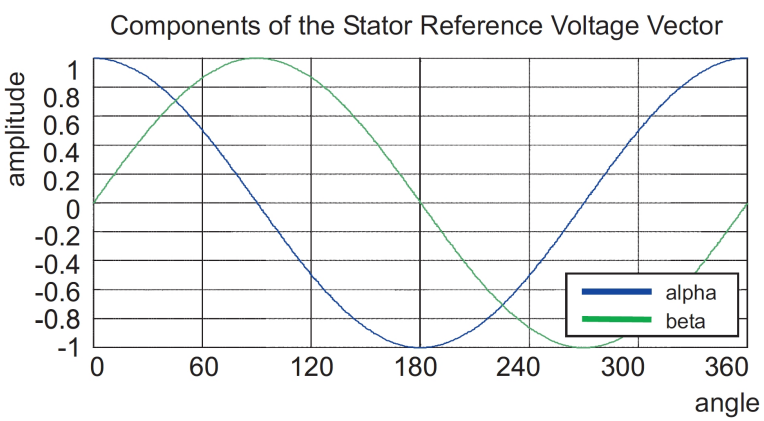

Figure 4-65. Direct-ua and quadrature-ub components of stator reference voltage

Figure 4-65 depicts the u_{α} and u_{β} components of the stator reference voltage vector U_S that were calculated by the equations $u_{\alpha} = \cos(\theta)$ and $u_{\beta} = \sin(\theta)$, respectively.

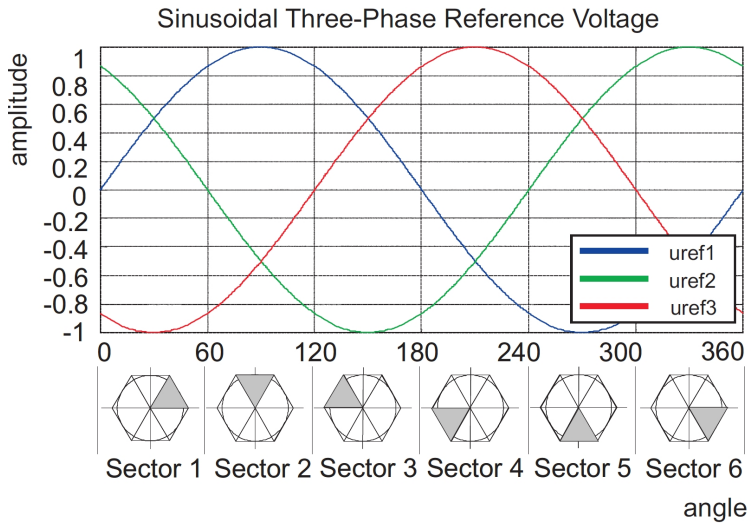

Figure 4-66. Reference voltages u_ref1, u_ref2 and u_ref3

The Sector Identification Tree, shown in Figure 4-67, can be a numerical solution of the approach shown in Figure 4-66.

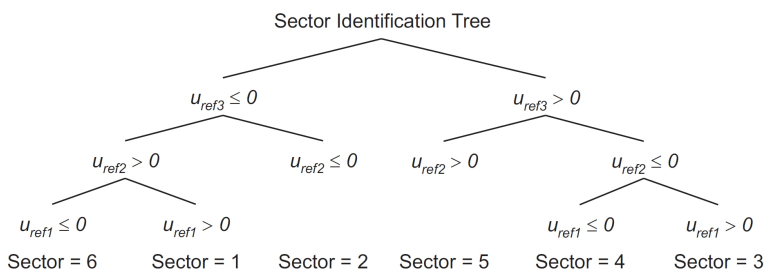

Figure 4-67. Identification of the sector number

It should be pointed out that, in the worst case, three simple comparisons are required to precisely identify the sector of the stator reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 4-61, the stator reference voltage vector is phase-advanced by 30 ° from the α-axis, which results in the positive quantities of u_{ref1} and u_{ref2} and the negative quantity of u_{ref3}; refer to Figure 4-66. If these quantities are used as the inputs to the Sector Identification Tree, the product of those comparisons will be Sector I. Using the same approach identifies Sector II, if the stator reference voltage vector is located according to the one shown in Figure 4-64. The variables t_1, t_2 and t_3, representing the switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

$$t_1 = \frac{T - t\_1 - t\_2}{2}$$

Equation **GMCLIB_SvmStd_Eq25**

$$t_2 = t_1 + t\_1$$

Equation **GMCLIB_SvmStd_Eq26**

$$t_3 = t_2 + t\_2$$

Equation **GMCLIB_SvmStd_Eq27**

where T is the switching period, t_1 and t_2 are the duty-cycle ratios (see Table 4-145) of the basic space vectors, given for the respective sector. Equation GMCLIB_SvmStd_Eq25, equation GMCLIB_SvmStd_Eq26 and equation GMCLIB_SvmStd_Eq27 are specific solely to the Standard Space Vector Modulation technique; consequently, other Space Vector Modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, t_1, t_2 and t_3, to the respective motor phases. This is a simple task, accomplished in view of the position of the stator reference voltage vector as shown in Table 4-146.

Table 4-146. Assignment of the duty-cycle ratios to motor phases

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
pwm_a	t_3	t_2	t_1	t_1	t_2	t_3
pwm_b	t_2	t_3	t_3	t_2	t_1	t_1
pwm_c	t_1	t_1	t_2	t_3	t_3	t_2

The principle of the Space Vector Modulation technique consists in applying the basic voltage vectors U_{XXX} and O_{XXX} for the certain time in such a way that the mean vector, generated by the Pulse Width Modulation approach for the period T, is equal to the original stator reference voltage vector U_S. This provides a great variability of the arrangement of the basic vectors during the PWM period T. Those vectors might be arranged either to lower switching losses or to achieve diverse results, such as centre-aligned PWM, edge-aligned PWM or a minimal number of switching states. A brief discussion of the widely-used centre-aligned PWM follows. Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels,

pwm $_a$, pwm $_b$ and pwm $_c$ with a free-running up-down counter. The timer counts to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 4-68

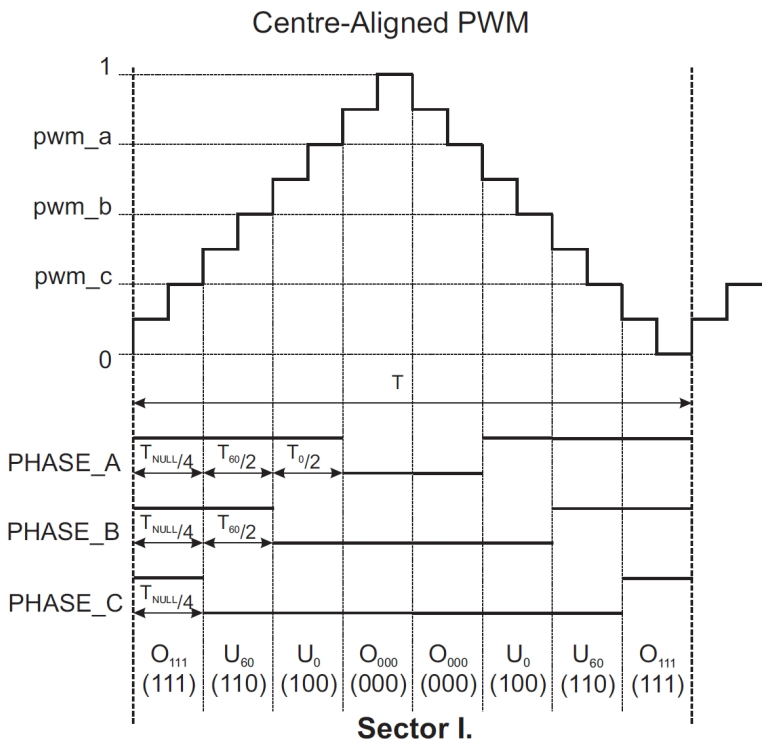

Figure 4-68. Standard space vector modulation technique - centre-aligned PWM

4.113.5 Re-entrancy

The function is re-entrant.

4.113.6 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_F16 tr16InVoltage;
SWLIBS_3Syst_F16 tr16PwmABC;
tU16             u16SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tr16InVoltage.f16Arg1       = FRAC16 (12.99/U_MAX);
  tr16InVoltage.f16Arg2       = FRAC16 (7.5/U_MAX);
```

```
  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle    = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle    = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle    = 0x0000 = FRAC16(0.0000... )
  // svmSector         = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmStd_F16 (&tr16PwmABC,&tr16InVoltage);

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle    = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle    = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle    = 0x0000 = FRAC16(0.0000... )
  // svmSector         = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmStd (&tr16PwmABC,&tr16InVoltage,Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr16PwmABC
  // pwmA dutycycle    = 0x7FFF = FRAC16(0.9999... )
  // pwmb dutycycle    = 0x4000 = FRAC16(0.5000... )
  // pwmc dutycycle    = 0x0000 = FRAC16(0.0000... )
  // svmSector         = 0x1 [sector]
  u16SvmSector = GMCLIB_SvmStd (&tr16PwmABC,&tr16InVoltage);
}
```

4.114 Function GMCLIB_SvmStd_FLT

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

4.114.1 Declaration

```
tU32 GMCLIB_SvmStd_FLT(SWLIBS_3Syst_FLT *pOut, const SWLIBS_2Syst_FLT *const pIn);
```

4.114.2 Arguments

Table 4-147. GMCLIB_SvmStd_FLT arguments

Type	Name	Direction	Description
SWLIBS_3Syst_FLT *	pOut	**input, output**	Pointer to the structure containing calculated duty-cycle ratios of the 3-Phase system.
const SWLIBS_2Syst_FLT *const	pIn	**input**	Pointer to the structure containing direct U_α and quadrature U_β components of the stator voltage vector.

4.114.3 Return

The function returns a 32-bit value in format INT, representing the actual space sector which contains the stator reference vector U_s.

4.114.4 Description

The GMCLIB_SvmStd_FLT function for calculating duty-cycle ratios is widely-used in the modern electric drive. This function calculates appropriate duty-cycle ratios, which are needed for generating the given stator reference voltage vector using a special Space Vector Modulation technique, termed Standard Space Vector Modulation.

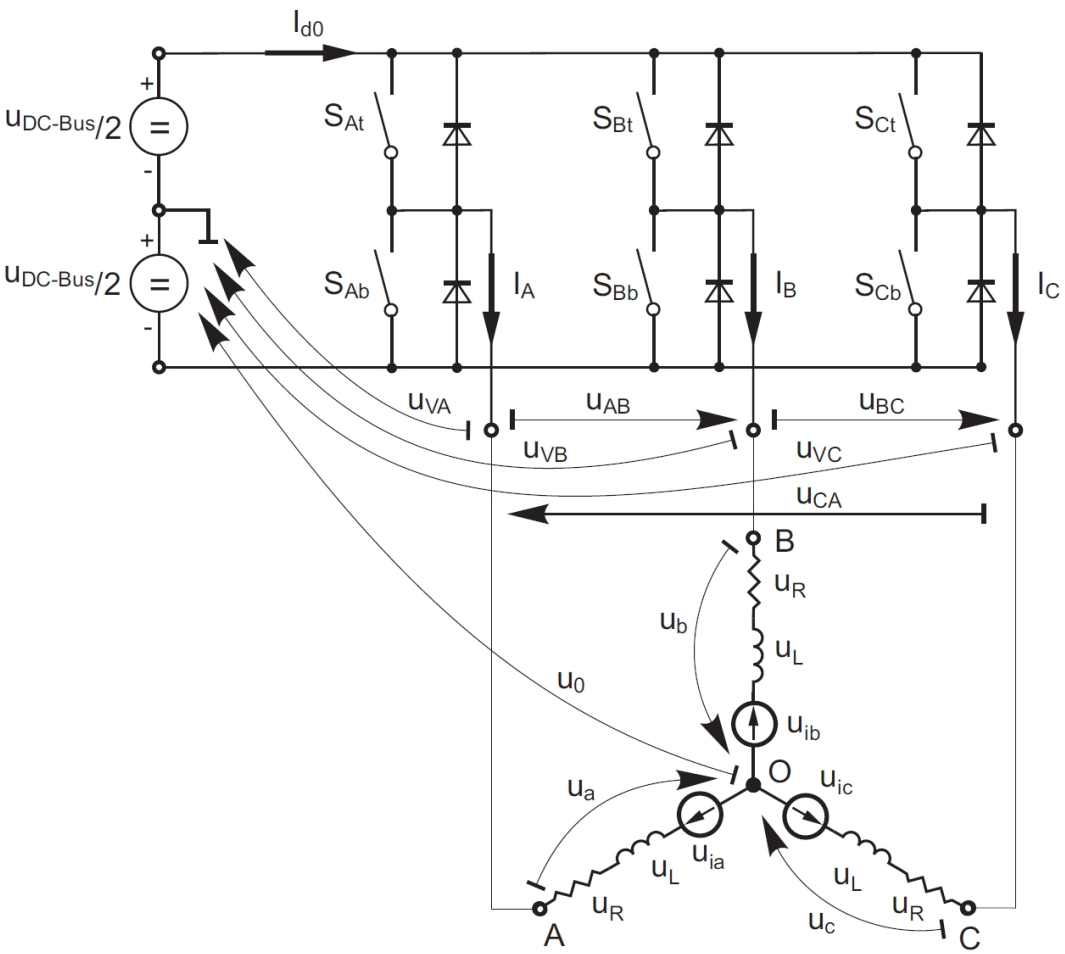

Figure 4-69. Power stage schematic diagram

Top and bottom switches work in a complementary mode; i.e., if the top switch (S_{At}) is ON, then the corresponding bottom switch (S_{Ab}) is OFF, and vice versa. Considering that value 1 is assigned to the ON state of the top switch, and value 0 is assigned to the ON state of the bottom switch, the switching vector, $[a, b, c]^T$ can be defined. Creating

such a vector allows a numerical definition of all possible switching states. In a three-phase power stage configuration (as shown in Figure 4-69), eight possible switching states (detailed in Figure 4-70) are feasible.

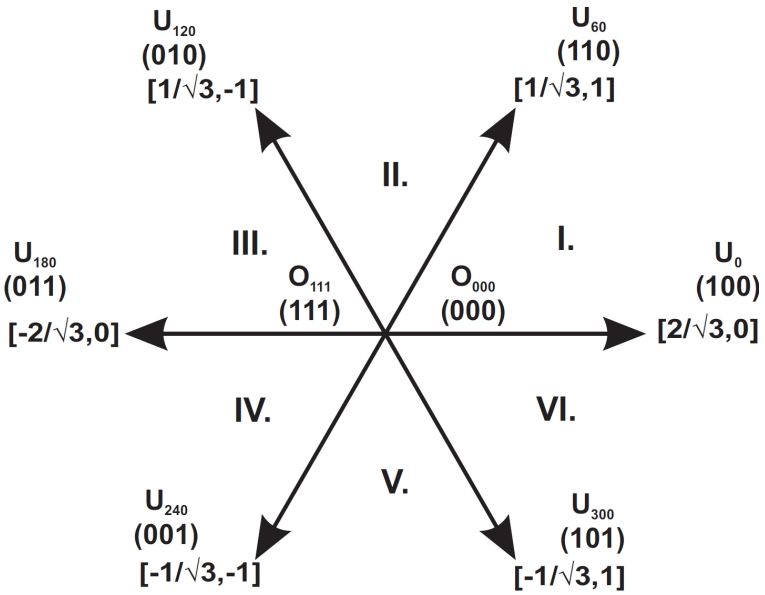

Figure 4-70. Basic space vectors

These states, together with the resulting instantaneous output line-to-line and phase voltages, are listed in Table 4-148.

Table 4-148. Switching patterns

a	b	c	U_a	U_b	U_c	U_{AB}	U_{BC}	U_{CA}	Vector
0	0	0	0	0	0	0	0	0	O_{000}
1	0	0	2/3*U_{DCBus}	-1/3*U_{DCBus}	-1/3*U_{DCBus}	U_{DCBus}	0	-U_{DCBus}	U_0
1	1	0	1/3*U_{DCBus}	1/3*U_{DCBus}	-2/3*U_{DCBus}	0	U_{DCBus}	-U_{DCBus}	U_{60}
0	1	0	-1/3*U_{DCBus}	2/3*U_{DCBus}	-1/3*U_{DCBus}	-U_{DCBus}	U_{DCBus}	0	U_{120}
0	1	1	-2/3*U_{DCBus}	1/3*U_{DCBus}	1/3*U_{DCBus}	-U_{DCBus}	0	U_{DCBus}	U_{240}
0	0	1	-1/3*U_{DCBus}	-1/3*U_{DCBus}	2/3*U_{DCBus}	0	-U_{DCBus}	U_{DCBus}	U_{300}
1	0	1	1/3*U_{DCBus}	-2/3*U_{DCBus}	1/3*U_{DCBus}	U_{DCBus}	-U_{DCBus}	0	U_{360}
1	1	1	0	0	0	0	0	0	O_{111}

The quantities of the direct-u_α and the quadrature-u_β components of the two-phase orthogonal coordinate system, describing the three-phase stator voltages, are expressed by the Clarke Transformation.

$$U_\alpha = \frac{2}{3}\left(U_a - \frac{U_b}{2} - \frac{U_c}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq1**

$$U_\beta = \frac{2}{3}\left(0 + \frac{\sqrt{3}U_b}{2} - \frac{\sqrt{3}U_C}{2}\right)$$

Equation **GMCLIB_SvmStd_Eq2**

The three-phase stator voltages,U_a, U_b, and U_c, are transformed using the Clarke Transformation into the U_α and the U_β components of the two-phase orthogonal coordinate system. The transformation results are listed in Table 4-149.

Table 4-149. Switching patterns and space vectors

a	b	c	u_α	u_β	Vector
0	0	0	0	0	O_{000}
1	0	0	$2/3*U_{DCBus}$	0	U_0
1	1	0	$1/3*U_{DCBus}$	$1/\sqrt{(3)}*U_{DCBus}$	U_{60}
0	1	0	$-1/3*U_{DCBus}$	$1/\sqrt{(3)}*U_{DCBus}$	U_{120}
0	1	1	$-2/3*U_{DCBus}$	0	U_{240}
0	0	1	$-1/3*U_{DCBus}$	$-1/\sqrt{(3)}*U_{DCBus}$	U_{300}
1	0	1	$1/3*U_{DCBus}$	$-1/\sqrt{(3)}*U_{DCBus}$	U_{360}
1	1	1	0	0	O_{111}

Figure 4-70 graphically depicts some feasible basic switching states (vectors). It is clear that there are six non-zero vectors U_0, U_{60}, U_{120}, U_{240}. U_{300}, and two zero vectors O_{111}, O_{000}, usable for switching. Therefore, the principle of the Standard Space Vector Modulation resides in applying appropriate switching states for a certain time and thus generating a voltage vector identical to the reference one.

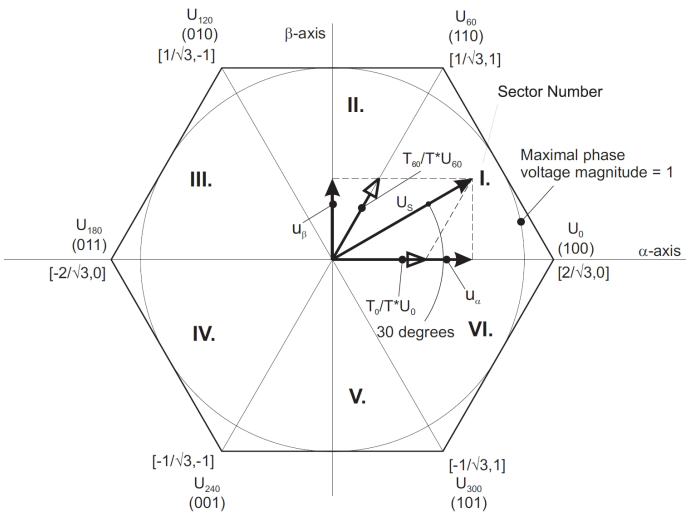

Figure 4-71. Projection of reference voltage vector in sector I

Referring to that principle, an objective of the Standard Space Vector Modulation is the approximation of the reference stator voltage vector U_S with an appropriate combination of the switching patterns composed of basic space vectors. A graphical explanation of this objective is shown in Figure 4-71 and Figure 4-72.

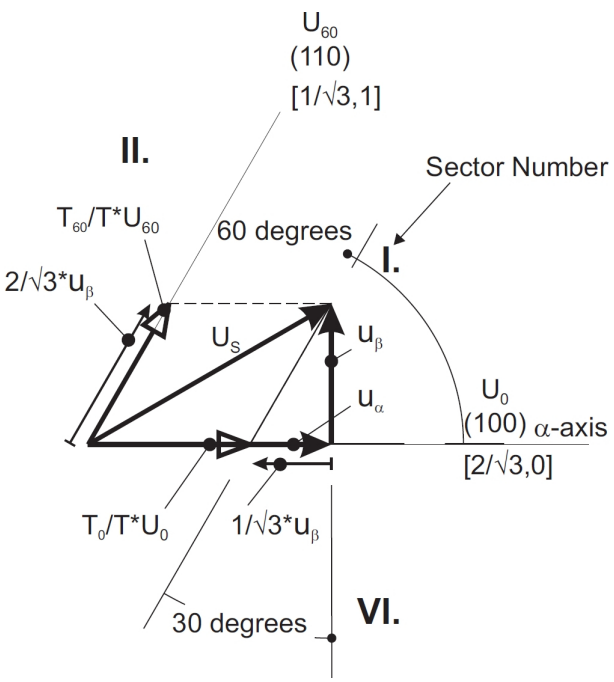

Figure 4-72. Detail of the voltage vector projection in sector I

The stator reference voltage vector U_S is phase-advanced by 30 ° from the axis- α and thus might be generated with an appropriate combination of the adjacent basic switching states U_0 and $_{60}$. These figures also indicate the resultant U_α and U_β components for space vectors U_0 and U_{60}.

In this case, the reference stator voltage vector U_S is located in Sector I and, as previously mentioned, can be generated with the appropriate duty-cycle ratios of the basic switching states U_{60} and U_0. The principal equations concerning this vector location are:

$$T = T_{60} + T_0 + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq3**

$$U_S = \frac{T_{60}}{T} U_{60} + \frac{T_0}{T} U_0$$

Equation **GMCLIB_SvmStd_Eq4**

where T_{60} and T_0 are the respective duty-cycle ratios for which the basic space vectors U_{60} and U_0 should be applied within the time period T. T_{null} is the course of time for which the null vectors O_{000} and O_{111} are applied. Those duty-cycle ratios can be calculated using equations:

$$u_\beta = \frac{T_{60}}{\text{T}} |U_{60}| \sin\ 60^\circ$$

Equation **GMCLIB_SvmStd_Eq5**

$$u_\alpha = \frac{T_0}{T} |U_0| + \frac{u_\beta}{\tan 60^\circ}$$

Equation **GMCLIB_SvmStd_Eq6**

Considering that the normalized magnitudes of the basic space vectors are $|U_{60}| = |U_0| = 2/\sqrt{(3)}$ and by substitution of the trigonometric expressions sin(60 °) and tan(60 °) by their quantities $2/\sqrt{(3)}$ and $\sqrt{(3)}$, respectively, equation GMCLIB_SvmStd_Eq5 and equation GMCLIB_SvmStd_Eq6 can be rearranged for the unknown duty-cycle ratios T_{60}/T and T_0/T:

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

$$\frac{T_{60}}{T} = u_\beta$$

Equation **GMCLIB_SvmStd_Eq7**

$$\frac{T_0}{T} = \frac{1}{2}\left(\sqrt{3}\,u_\alpha - u_\beta\right)$$

Equation **GMCLIB_SvmStd_Eq8**

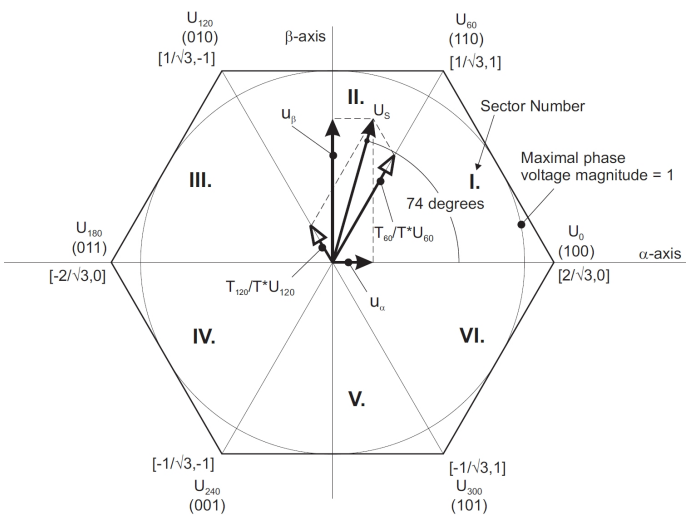

Figure 4-73. Projection of the reference voltage vector in sector II

Sector II is depicted in Figure 4-73. In this particular case, the reference stator voltage vector U_S is generated by the appropriate duty-cycle ratios of the basic switching states U_{60} and U_{120}. The basic equations describing this sector are:

$$T = T_{120} + T_{60} + T_{\text{null}}$$

Equation **GMCLIB_SvmStd_Eq9**

$$U_S = \frac{T_{120}}{T} U_{120} + \frac{T_{60}}{T} U_{60}$$

Equation **GMCLIB_SvmStd_Eq10**

where T $_{120}$ and T $_{60}$ are the respective duty-cycle ratios for which the basic space vectors U $_{120}$ and U $_{60}$ should be applied within the time period T. These resultant duty-cycle ratios are formed from the auxiliary components termed A and B. A graphical representation of the auxiliary components is shown in Figure 4-74.

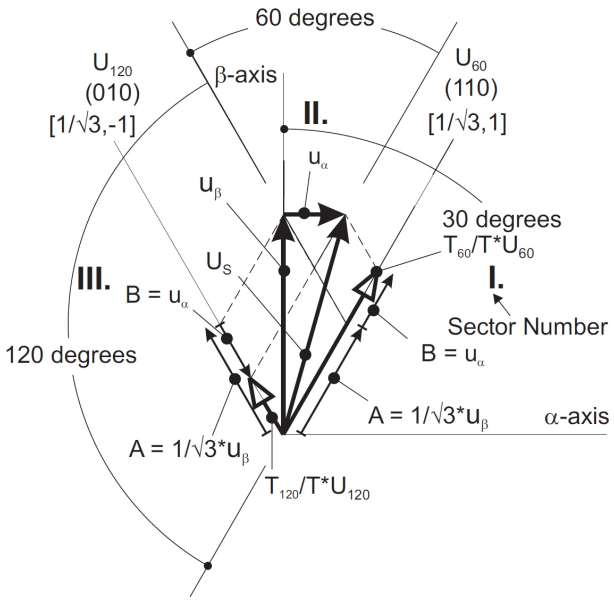

Figure 4-74. Detail of the voltage vector projection in sector II

The equations describing those auxiliary time-duration components are:

$$\frac{\sin 30^\circ}{\sin 120^\circ} = \frac{A}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq11**

$$\frac{\sin 60^\circ}{\sin 120^\circ} = \frac{B}{u_\alpha}$$

Equation **GMCLIB_SvmStd_Eq12**

Equation GMCLIB_SvmStd_Eq11 and equation GMCLIB_SvmStd_Eq12 have been formed using the sine rule. These equations can be rearranged for the calculation of the auxiliary time-duration components A and B. This is done simply by substitution of the trigonometric terms sin(30 °), sin(120 °) and sin(60 °) by their numerical representations 1/2, √(3)/2 and 1/ √(3), respectively.

$$A = \frac{1}{\sqrt{3}} u_\beta$$

Equation **GMCLIB_SvmStd_Eq13**

$$B = u_\alpha$$

Equation **GMCLIB_SvmStd_Eq14**

The resultant duty-cycle ratios, T_{120}/T and T_{60}/T, are then expressed in terms of the auxiliary time-duration components defined by equation GMCLIB_SvmStd_Eq14 and equation GMCLIB_SvmStd_Eq15, as follows:

$$\frac{T_{120}}{T} |U_{120}| = A - B$$

Equation **GMCLIB_SvmStd_Eq15**

$$\frac{T_{60}}{T} |U_{60}| = A + B$$

Equation **GMCLIB_SvmStd_Eq16**

With the help of these equations, and also considering the normalized magnitudes of the basic space vectors to be $|U_{120}| = |U_{60}| = 2/(3)$, the equations expressed for the unknown duty-cycle ratios of basic space vectors T_{120}/T and T_{60}/T can be written:

$$\frac{T_{120}}{T} = \frac{1}{2}\left(u_\beta - \sqrt{3}\, u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq17**

$$\frac{T_{60}}{T} = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq18**

The duty-cycle ratios in remaining sectors can be derived using the same approach. The resulting equations will be similar to those derived for Sector I and Sector II.

To depict duty-cycle ratios of the basic space vectors for all sectors, we define three auxiliary variables:

$$X = u_\beta$$

Equation **GMCLIB_SvmStd_Eq19**

$$Y = \frac{1}{2}\left(u_\beta + \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq20**

$$Z = \frac{1}{2}\left(u_\beta - \sqrt{3}\,u_\alpha\right)$$

Equation **GMCLIB_SvmStd_Eq21**

To simplify the calculation, two expressions, t_1 and t_2, are introduced. These expressions represent duty-cycle ratios of the basic space vectors in the respective sector; e.g., for the first sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{60} and U_{0}; for the second sector, t_1 and t_2 represent duty-cycle ratios of the basic space vectors U_{120} and U_{60}, etc.

For each sector, the relation between the t_1 and t_2 and variables X (GMCLIB_SvmStd_Eq19), Y (GMCLIB_SvmStd_Eq20) and Z (GMCLIB_SvmStd_Eq21), are listed in Table 4-150.

Table 4-150. Determination of t_1 and t_2 expressions

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0

Table continues on the next page...

Table 4-150. Determination of t_1 and t_2 expressions (continued)

t_1	X	Y	-Y	Z	-Z	-X
t_2	-Z	Z	X	-X	-Y	Y

For the determination of auxiliary variables X equation GMCLIB_SvmStd_Eq19, Y equation GMCLIB_SvmStd_Eq20 and Z equation GMCLIB_SvmStd_Eq21, the sector number is required. This information can be obtained by several approaches. One approach discussed here requires the use of a modified Inverse Clark Transformation to transform the direct (α) and quadrature ( ) components into a balanced three-phase quantity u $_{ref1}$, u $_{ref2}$ and u $_{ref3}$, used for a straightforward calculation of the sector number, to be shown later.

$$u_{\mathrm{ref1}} = u_{\beta}$$

Equation **GMCLIB_SvmStd_Eq22**

$$u_{\mathrm{ref2}} = \frac{1}{2}\left(- u_{\beta} + \sqrt{3}\, u_{\alpha} \right)$$

Equation **GMCLIB_SvmStd_Eq23**

$$u_{\mathrm{ref3}} = \frac{1}{2}\left(- u_{\beta} - \sqrt{3}\, u_{\alpha} \right)$$

Equation **GMCLIB_SvmStd_Eq24**

The modified Inverse Clark Transformation projects the quadrature (u $_{β}$) component into u $_{ref1}$, as shown in Figure 4-75 and Figure 4-76, whereas voltages generated by the conventional Inverse Clark Transformation project the u $_{α}$ component into u $_{ref1}$.

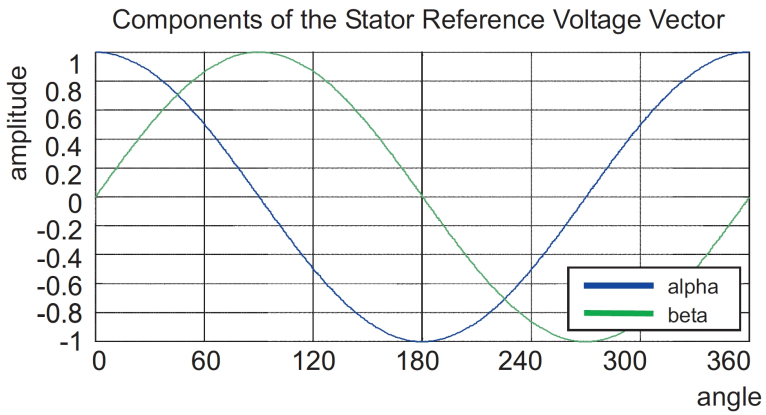

Figure 4-75. Direct (u_a) and quadrature (u_b) components of stator reference voltage

Figure 4-75 depicts the u_α and u_β components of the stator reference voltage vector U_S that were calculated by the equations $u_\alpha = \cos(\theta)$ and $u_\beta = \sin(\theta)$, respectively.

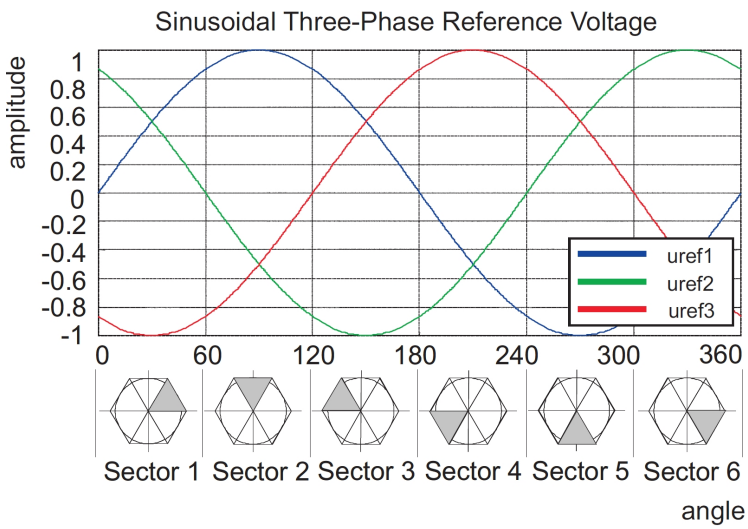

Figure 4-76. Reference voltages u_ref1, u_ref2 and u_ref3

The Sector Identification Tree, shown in Figure 4-77, can be a numerical solution of the approach shown in Figure 4-76.

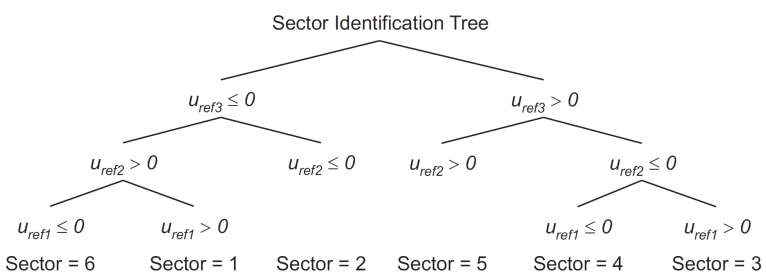

Figure 4-77. Identification of the sector number

It should be pointed out that, in the worst case, three simple comparisons are required to precisely identify the sector of the stator reference voltage vector. For example, if the stator reference voltage vector resides according to the one shown in Figure 4-71, the stator reference voltage vector is phase-advanced by 30 ° from the α-axis, which results in the positive quantities of u_{ref1} and u_{ref2} and the negative quantity of u_{ref3}; refer to Figure 4-76. If these quantities are used as the inputs to the Sector Identification Tree, the product of those comparisons will be Sector I. Using the same approach identifies Sector II, if the stator reference voltage vector is located according to the one shown in Figure 4-74. The variables t_1, t_2 and t_3, representing the switching duty-cycle ratios of the respective three-phase system, are given by the following equations:

$$t_1 = \frac{T - t\_1 - t\_2}{2}$$

Equation **GMCLIB_SvmStd_Eq25**

$$t_2 = t_1 + t\_1$$

Equation **GMCLIB_SvmStd_Eq26**

$$t_3 = t_2 + t\_2$$

Equation **GMCLIB_SvmStd_Eq27**

where T is the switching period, t_1 and t_2 are the duty-cycle ratios (see Table 4-150) of the basic space vectors, given for the respective sector. Equation GMCLIB_SvmStd_Eq25, equation GMCLIB_SvmStd_Eq26 and equation GMCLIB_SvmStd_Eq27 are specific solely to the Standard Space Vector Modulation technique; consequently, other Space Vector Modulation techniques discussed later will require deriving different equations.

The next step is to assign the correct duty-cycle ratios, t_1, t_2 and t_3, to the respective motor phases. This is a simple task, accomplished in view of the position of the stator reference voltage vector as shown in Table 4-151.

Table 4-151. Assignment of the duty-cycle ratios to motor phases

Sector	U_0, U_{60}	U_{60}, U_{120}	U_{120}, U_{180}	U_{180}, U_{240}	U_{240}, U_{300}	U_{300}, U_0
pwm_a	t_3	t_2	t_1	t_1	t_2	t_3

Table continues on the next page...

Table 4-151. Assignment of the duty-cycle ratios to motor phases (continued)

pwm $_b$	t_2	t_3	t_3	t_2	t_1	t_1
pwm $_c$	t_1	t_1	t_2	t_3	t_3	t_2

The principle of the Space Vector Modulation technique resides in applying the basic voltage vectors U $_{XXX}$ and O $_{XXX}$ for a certain time in such a way that the main vector, generated by the Pulse Width Modulation approach for the period T, is equal to the original stator reference voltage vector U $_S$. This provides a great variability of the arrangement of the basic vectors during the PWM period T. Those vectors might be arranged either to lower switching losses or to achieve diverse results, such as centre-aligned PWM, edge-aligned PWM or a minimal number of switching states. A brief discussion of the widely-used centre-aligned PWM follows.

Generating the centre-aligned PWM pattern is accomplished practically by comparing the threshold levels, pwm $_a$, pwm $_b$ and pwm $_c$ with a free-running up-down counter. The timer counts to 1 (0x7FFF) and then down to 0 (0x0000). It is supposed that when a threshold level is larger than the timer value, the respective PWM output is active. Otherwise, it is inactive; see Figure 4-78

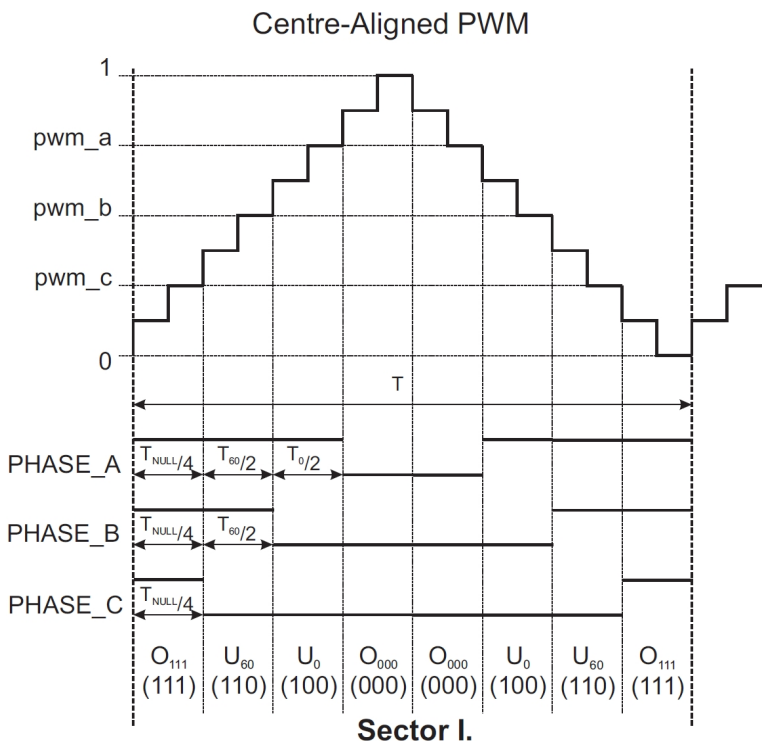

Figure 4-78. Standard space vector modulation technique - centre-aligned PWM

Note

As during the computation the irrational value of √(3)/4 is used for calculation, the correction constant is used to increase the

calculation precision in boundary intervals. This correction constant is equal to the difference of √(3)/4 computed in double and in single precision.

4.114.5 Re-entrancy

The function is re-entrant.

4.114.6 Code Example

```
#include "gmclib.h"
#define U_MAX 15

SWLIBS_2Syst_FLT tInVoltage;
SWLIBS_3Syst_FLT tPwmABC;
tU32             u32SvmSector;

void main(void)
{
  // Input voltage vector 15V @ angle 30deg
  // alpha component of input voltage vector = 12.99[V]
  // beta component of input voltage vector  = 7.5[V]
  tInVoltage.fltArg1       = (tFloat)(12.99/U_MAX);
  tInVoltage.fltArg2       = (tFloat)(7.5/U_MAX);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle    = (tFloat)(0.9999888)
  // pwmb dutycycle    = (tFloat)(0.5000111)
  // pwmc dutycycle    = (tFloat)(0.0000111)
  // u32SvmSector         = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd_FLT (&tPwmABC,&tInVoltage);

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle    = (tFloat)(0.9999888)
  // pwmb dutycycle    = (tFloat)(0.5000111)
  // pwmc dutycycle    = (tFloat)(0.0000111)
  // u32SvmSector         = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd (&tPwmABC,&tInVoltage,Define FLT);

  // ##############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ##############################################################

  // output pwm dutycycles stored in structure referenced by tr32PwmABC
  // pwmA dutycycle    = (tFloat)(0.9999888)
  // pwmb dutycycle    = (tFloat)(0.5000111)
  // pwmc dutycycle    = (tFloat)(0.0000111)
  // u32SvmSector         = 0x1 [sector]
  u32SvmSector = GMCLIB_SvmStd (&tPwmABC,&tInVoltage);
}
```

4.115 Function MLIB_Abs_F32

This function returns absolute value of input parameter.

4.115.1 Declaration

```
tFrac32 MLIB_Abs_F32(register tFrac32 f32In);
```

4.115.2 Arguments

Table 4-152. MLIB_Abs_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.
register tFrac32	f32In	**input**	Input value

4.115.3 Return

Absolute value of input parameter.Absolute value of input parameter

4.115.4 Description

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} f32In & \text{if } f32In \geq 0 \\ (-f32In) & \text{if } f32In < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.115.5 Re-entrancy

The function is re-entrant.

4.115.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32 (-0.25);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs_F32(f32In);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs (f32In, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs (f32In);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} f32In & \text{if } f32In \geq 0 \\ (-f32In) & \text{if } f32In < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.115.7 Re-entrancy

The function is re-entrant.

4.115.8 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32 (-0.25);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs_F32(f32In);

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs (f32In, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC32(0.25)
    f32Out = MLIB_Abs (f32In);
}
```

4.116 Function MLIB_Abs_F16

This function returns absolute value of input parameter.

4.116.1 Declaration

```
tFrac16 MLIB_Abs_F16(register tFrac16 f16In);
```

4.116.2 Arguments

Table 4-153. MLIB_Abs_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value.
register tFrac16	f16In	**input**	Input value

4.116.3 Return

Absolute value of input parameter.Absolute value of input parameter

4.116.4 Description

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} f16In & \text{if } f16In \geq 0 \\ (-f16In) & \text{if } f16In < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.116.5 Re-entrancy

The function is re-entrant.

4.116.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16 (-0.25);
```

```
    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs_F16(f16In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs (f16In, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs (f16In);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} f16In & \text{if } f16In \geq 0 \\ (-f16In) & \text{if } f16In < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.116.7 Re-entrancy

The function is re-entrant.

4.116.8 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16 (-0.25);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs_F16(f16In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs (f16In, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################
```

```
    // output should be FRAC16(0.25)
    f16Out = MLIB_Abs (f16In);
}
```

4.117 Function MLIB_Abs_FLT

This function returns absolute value of input parameter.

4.117.1 Declaration

```
tFloat MLIB_Abs_FLT(register tFloat fltIn);
```

4.117.2 Arguments

Table 4-154. MLIB_Abs_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value.
register tFloat	fltIn	**input**	Input value

4.117.3 Return

Absolute value of input parameter.Absolute value of input parameter

4.117.4 Description

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$\mathrm{fltOut} = \begin{cases} \mathrm{fltIn} & \text{if} \quad \mathrm{fltIn} \geq 0 \\ (-\mathrm{fltIn}) & \text{if} \quad \mathrm{fltIn} < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

This inline function returns the absolute value of input parameter. The input value as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the absolute value of input parameter is outside the (-2^{128}, 2^{128}) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.117.5 Re-entrancy

The function is re-entrant.

4.117.6 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = -0.25
    fltIn = -0.25;

    // output should be 0.25
    fltOut = MLIB_Abs_FLT(fltIn);

    // output should be 0.25
    fltOut = MLIB_Abs (fltIn, Define FLT);

    // ###############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ###############################################################

    // output should be 0.25
    fltOut = MLIB_Abs (fltIn);
}
```

The output of the function is defined by the following simple equation:

$$\mathrm{fltOut} = \begin{cases} \mathrm{fltIn} & \text{if} \quad \mathrm{fltIn} \geq 0 \\ (-\mathrm{fltIn}) & \text{if} \quad \mathrm{fltIn} < 0 \end{cases}$$

Equation **MLIB_Abs_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.117.7 Re-entrancy

The function is re-entrant.

4.117.8 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
    // input value = -0.25
    fltIn = -0.25;

    // output should be 0.25
    fltOut = MLIB_Abs_FLT(fltIn);

    // output should be 0.25
    fltOut = MLIB_Abs (fltIn, Define FLT);

    // ###############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ###############################################################

    // output should be 0.25
    fltOut = MLIB_Abs (fltIn);
}
```

4.118 Function MLIB_AbsSat_F32

This function returns absolute value of input parameter and saturate if necessary.

4.118.1 Declaration

```
tFrac32 MLIB_AbsSat_F32(register tFrac32 f32In);
```

4.118.2 Arguments

Table 4-155. MLIB_AbsSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value.

4.118.3 Return

Absolute value of input parameter, saturated if necessary.

4.118.4 Description

This inline function returns the absolute value of input parameter. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad |f32In| < FRAC32\_MIN \\ |f32In| & \text{if} \quad FRAC32\_MIN \le |f32In| \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad |f32In| > FRAC32\_MAX \end{cases}$$

Equation **MLIB_AbsSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.118.5 Re-entrancy

The function is re-entrant.

4.118.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
```

```
tFrac32 f32Out;

void main(void)
{
    // input value = -0.25
    f32In = FRAC32 (-0.25);

    // output should be FRAC32(0.25)
    f32Out = MLIB_AbsSat_F32(f32In);

    // output should be FRAC32(0.25)
    f32Out = MLIB_AbsSat (f32In, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC32(0.25)
    f32Out = MLIB_AbsSat (f32In);
}
```

4.119 Function MLIB_AbsSat_F16

This function returns absolute value of input parameter and saturate if necessary.

4.119.1 Declaration

```
tFrac16 MLIB_AbsSat_F16(register tFrac16 f16In);
```

4.119.2 Arguments

Table 4-156. MLIB_AbsSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value.

4.119.3 Return

Absolute value of input parameter, saturated if necessary.

4.119.4 Description

This inline function returns the absolute value of input parameter. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the absolute value of input parameter is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad |f16In| < FRAC16\_MIN \\ |f16In| & \text{if} \quad FRAC16\_MIN \le |f16In| \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad |f16In| > FRAC16\_MAX \end{cases}$$

Equation **MLIB_AbsSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.119.5 Re-entrancy

The function is re-entrant.

4.119.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
    // input value = -0.25
    f16In = FRAC16 (-0.25);

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat_F16(f32In);

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat (f16In, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC16(0.25)
    f16Out = MLIB_AbsSat (f16In);
}
```

4.120 Function MLIB_Add_F32

This function returns sum of two input parameters.

4.120.1 Declaration

```
tFrac32 MLIB_Add_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.120.2 Arguments

Table 4-157. MLIB_Add_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be add.
register tFrac32	f32In2	**input**	Second value to be add.
register tFrac32	f32In1	**input**	First value to be add
register tFrac32	f32In2	**input**	Second value to be add

4.120.3 Return

Sum of two input values.Sum of two input values

4.120.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 + f32In2\right)$$

Equation **MLIB_Add_Eq1**

This inline function returns the sum of two input values. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.120.5 Re-entrancy:

The function is re-entrant.

4.120.6 Code Example:

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1  = FRAC32 (0.25);
  // input value 2 = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add (f32In1, f32In2, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add (f32In1, f32In2);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 + f32In2\right)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is not detected in this function.

4.120.7 Re-entrancy:

The function is re-entrant.

4.120.8 Code Example:

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1  = FRAC32 (0.25);
  // input value 2 = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add (f32In1, f32In2, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_Add (f32In1, f32In2);
}
```

4.121 Function MLIB_Add_F16

This function returns sum of two input parameters.

4.121.1 Declaration

```
tFrac16 MLIB_Add_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.121.2 Arguments

Table 4-158. MLIB_Add_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be add.
register tFrac16	f16In2	**input**	Second value to be add.
register tFrac16	f16In1	**input**	First value to be add
register tFrac16	f16In2	**input**	Second value to be add

4.121.3 Return

Sum of two input values.Sum of two input values

4.121.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = \left(f16In1 + f16In2\right)$$

Equation **MLIB_Add_Eq1**

This inline function returns the sum of two input values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.121.5 Re-entrancy

The function is re-entrant.

4.121.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1  = FRAC16 (0.25);
  // input value 2 = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add (f16In1, f16In2, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add (f16In1, f16In2);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = \left(f16In1 + f16In2\right)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is written in S12Z-core assembly thus is not ANSI-C compliant. The overflow is not detected in this function.

4.121.7 Re-entrancy

The function is re-entrant.

4.121.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;
```

```
void main(void)
{
  // input value 1 = 0.25
  f16In1  = FRAC16 (0.25);
  // input value 2 = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add (f16In1, f16In2, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Add (f16In1, f16In2);
}
```

4.122 Function MLIB_Add_FLT

This function returns sum of two input parameters.

4.122.1 Declaration

```
tFloat MLIB_Add_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

4.122.2 Arguments

Table 4-159. MLIB_Add_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	First value to be add.
register tFloat	fltIn2	**input**	Second value to be add.
register tFloat	fltIn1	**input**	First value to be add
register tFloat	fltIn2	**input**	Second value to be add

4.122.3 Return

Sum of two input values.Sum of two input values

4.122.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 + fltIn2)$$

Equation **MLIB_Add_Eq1**

This inline function returns the sum of two input values. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the sum of input values is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.122.5 Re-entrancy

The function is re-entrant.

4.122.6 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFloat fltOut;

void main(void)
{
  // input value 1 = 0.25
  fltIn1  = 0.25;
  // input value 2 = 0.25
  fltIn2  = 0.25;

  // output should be 0.5
  fltOut = MLIB_Add_FLT(fltIn1, fltIn2);

  // output should be 0.5
  fltOut = MLIB_Add (fltIn1, fltIn2, Define FLT);

  // ###############################################################
  // Available only if single precision floating point
```

```
  // implementation selected as default
  // ###############################################################

  // output should be 0.5
  fltOut = MLIB_Add (fltIn1, fltIn2);
}
```

The output of the function is defined by the following simple equation:

$$\mathrm{fltOut} = \left(\mathrm{fltIn1} + \mathrm{fltIn2}\right)$$

Equation **MLIB_Add_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.122.7 Re-entrancy

The function is re-entrant.

4.122.8 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFloat fltOut;

void main(void)
{
  // input value 1 = 0.25
  fltIn1  = 0.25;
  // input value 2 = 0.25
  fltIn2  = 0.25;

  // output should be 0.5
  fltOut = MLIB_Add_FLT(fltIn1, fltIn2);

  // output should be 0.5
  fltOut = MLIB_Add (fltIn1, fltIn2, Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 0.5
  fltOut = MLIB_Add (fltIn1, fltIn2);
}
```

4.123 Function MLIB_AddSat_F32

This function returns sum of two input parameters and saturate if necessary.

4.123.1 Declaration

```
tFrac32 MLIB_AddSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.123.2 Arguments

Table 4-160. MLIB_AddSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be add.
register tFrac32	f32In2	**input**	Second value to be add.
register tFrac32	f32In1	**input**	First value to be add
register tFrac32	f32In2	**input**	Second value to be add

4.123.3 Return

Sum of two input values, saturated if necessary.Sum of two input values, saturated if necessary

4.123.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 + f32In2) < FRAC32\_MIN \\ (f32In1 + f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 + f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 + f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_AddSat_Eq1**

This inline function returns the sum of two input values. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.123.5 Re-entrancy

The function is re-entrant.

4.123.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1  = FRAC32 (0.25);
  // input value 2 = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat (f32In1, f32In2, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat (f32In1, f32In2);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 + f32In2) < FRAC32\_MIN \\ (f32In1 + f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 + f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 + f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_AddSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.123.7 Re-entrancy

The function is re-entrant.

4.123.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1  = FRAC32 (0.25);
  // input value 2 = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat (f32In1, f32In2, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_AddSat (f32In1, f32In2);
}
```

4.124 Function MLIB_AddSat_F16

This function returns sum of two input parameters and saturate if necessary.

4.124.1 Declaration

```
tFrac16 MLIB_AddSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.124.2 Arguments

Table 4-161. MLIB_AddSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be add.
register tFrac16	f16In2	**input**	Second value to be add.
register tFrac16	f16In1	**input**	First value to be add
register tFrac16	f16In2	**input**	Second value to be add

4.124.3 Return

Sum of two input values, saturated if necessary.Sum of two input values, saturated if necessary

4.124.4 Description

This inline function returns the sum of two input values. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 + f16In2) < FRAC16\_MIN \\ (f16In1 + f16In2) & \text{if} \quad FRAC16\_MIN \le (f16In1 + f16In2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 + f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_AddSat_Eq1**

This inline function returns the sum of two input values. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.124.5 Re-entrancy

The function is re-entrant.

4.124.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1  = FRAC16 (0.25);
  // input value 2 = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat (f16In1, f16In2, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat (f16In1, f16In2);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 + f16In2) < FRAC16\_MIN \\ (f16In1 + f16In2) & \text{if} \quad FRAC16\_MIN \le (f16In1 + f16In2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 + f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_AddSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.124.7 Re-entrancy

The function is re-entrant.

4.124.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1  = FRAC16 (0.25);
  // input value 2 = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat (f16In1, f16In2, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_AddSat (f16In1, f16In2);
}
```

4.125 Function MLIB_Convert_F32F16

This function converts the input value to different representation with scale.

4.125.1 Declaration

```
tFrac32 MLIB_Convert_F32F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.125.2 Arguments

Table 4-162. MLIB_Convert_F32F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value in 16-bit fractional format to be converted.
register tFrac16	f16In2	**input**	Scale factor in 16-bit fractional format.

4.125.3 Return

Converted input value in 32-bit fractional format.

4.125.4 Description

This inline function returns converted input value. The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type thus both values represent the values in unity model. The second value represents the scale factor and is considered as 16-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} (tFrac32)\frac{f16In1}{-f16In2} & \text{if } (f16In2) < (tFrac16)0 \\ (tFrac32)(f16In1 \bullet f16In2) & \text{if } (f16In2) \le (tFrac16)0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.125.5 Re-entrancy

The function is re-entrant.

4.125.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac32 f32Out;

void main(void)
{
```

```
    // input value = 0.25 = 0x2000
    f16In1  = FRAC16 (0.25);

    // scale value = 0.5 = 0x4000
    f16In2 =  FRAC16 (0.5);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert_F32F16(f16In1, f16In2);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert (f16In1, f16In2, F32F16);

    // scale value = -0.5 = 0xC000
    f16In2 =  FRAC16 (-0.5);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert_F32F16(f16In1, f16In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert (f16In1, f16In2, F32F16);
}
```

4.126 Function MLIB_Convert_F32FLT

This function converts the input value to different representation with scale.

4.126.1 Declaration

```
tFrac32 MLIB_Convert_F32FLT(register tFloat fltIn1, register tFloat fltIn2);
```

4.126.2 Arguments

Table 4-163. MLIB_Convert_F32FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value in single precision floating point format to be converted.
register tFloat	fltIn2	**input**	Scale factor in single precision floating point format.

4.126.3 Return

Converted input value in 32-bit fractional format.

4.126.4 Description

This inline function returns converted input value. The input value is considered as single precision floating point data type and output value is considered as 32-bit fractional data type. The second value represents the scale factor and is considered as single precision floating point data type. The output value represents the values in unity model. The output saturation is implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (fltIn1 \bullet fltIn2) < FRAC32\_MIN \\ \frac{fltIn1 \bullet fltIn2}{INT32\_MAX} & \text{if} \quad FRAC32\_MIN \le (fltIn1 \bullet fltIn2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (fltIn1 \bullet fltIn2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.126.5 Re-entrancy

The function is re-entrant.

4.126.6 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25
    fltIn1  = 0.25;

    // scale value = 0.5
    fltIn2 =  0.5;

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert_F32FLT(fltIn1, fltIn2);

    // output should be FRAC32(0.125) = 0x10000000
    f32Out = MLIB_Convert (fltIn1, fltIn2, F32FLT);

    // scale value = 2
```

```
    fltIn2 =  2;

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert_F32FLT(fltIn1, fltIn2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Convert (fltIn1, fltIn2, F32FLT);
}
```

4.127 Function MLIB_Convert_F16F32

This function converts the input value to different representation with scale.

4.127.1 Declaration

```
tFrac16 MLIB_Convert_F16F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.127.2 Arguments

Table 4-164. MLIB_Convert_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value in 32-bit fractional format to be converted.
register tFrac32	f32In2	**input**	Scale factor in 32-bit fractional format.

4.127.3 Return

Converted input value in 16-bit fractional format.

4.127.4 Description

This inline function returns converted input value. The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type thus both values represent the values in unity model. The second value represents the scale factor and is considered as 32-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value

and converted to the output format. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} (tFrac16)\frac{f32In1}{-f32In2} & \text{if } (f32In2) < (tFrac32)0 \\ (tFrac16)(f32In1 \bullet f32In2) & \text{if } (f32In2) \le (tFrac32)0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.127.5 Re-entrancy

The function is re-entrant.

4.127.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25 = 0x20000000
    f32In1  = FRAC32 (0.25);

    // scale value = 0.5 = 0x40000000
    f32In2 =  FRAC32 (0.5);

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert_F16F32(f32In1, f32In2);

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert (f32In1, f32In2, F16F32);

    // scale value = -0.5 = 0xC0000000
    f32In2 =  FRAC32 (-0.5);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert_F16F32(f32In1, f32In2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert (f32In1, f32In2, F16F32);
}
```

4.128 Function MLIB_Convert_F16FLT

This function converts the input value to different representation with scale.

4.128.1 Declaration

```
tFrac16 MLIB_Convert_F16FLT(register tFloat fltIn1, register tFloat fltIn2);
```

4.128.2 Arguments

Table 4-165. MLIB_Convert_F16FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value in single precision floating point format to be converted.
register tFloat	fltIn2	**input**	Scale factor in single precision floating point format.

4.128.3 Return

Converted input value in 16-bit fractional format.

4.128.4 Description

This inline function returns converted input value. The input value is considered as single precision floating point data type and output value is considered as 16-bit fractional data type. The second value represents the scale factor and is considered as single precision floating point data type. The output value represents the values in unity model. The output saturation is implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (fltIn1 \bullet fltIn2) < FRAC16\_MIN \\ \frac{fltIn1 \bullet fltIn2}{INT16\_MAX} & \text{if} \quad FRAC16\_MIN \le (fltIn1 \bullet fltIn2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (fltIn1 \bullet fltIn2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.128.5 Re-entrancy

The function is re-entrant.

4.128.6 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25
    fltIn1  = 0.25;

    // scale value = 0.5
    fltIn2 =  0.5;

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert_F16FLT(fltIn1, fltIn2);

    // output should be FRAC16(0.125) = 0x1000
    f16Out = MLIB_Convert (fltIn1, fltIn2, F16FLT);

    // scale value = 2
    fltIn2 =  2;

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert_F16FLT(fltIn1, fltIn2);

    // output should be FRAC16(0.5) = 0x4000
    f16Out = MLIB_Convert (fltIn1, fltIn2, F16FLT);
}
```

4.129 Function MLIB_Convert_FLTF16

This function converts the input value to different representation.

4.129.1 Declaration

```
tFloat MLIB_Convert_FLTF16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.129.2 Arguments

Table 4-166. MLIB_Convert_FLTF16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value in 16-bit fractional format to be converted.
register tFrac16	f16In2	**input**	Scale factor in 16-bit fractional format.

4.129.3 Return

Converted input value in single precision floating point format.

4.129.4 Description

This inline function returns converted input value. The input value is considered as 16-bit fractional data type and output value is considered as single precision floating point data type. The input value represents the values in unity model. The second value represents the scale factor and is considered as 16-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$\text{fltOut} = \begin{cases} \dfrac{(\text{INT\_16}) f16\text{In1}}{-f16\text{In2} \bullet (\text{INT16\_MAX}+1)} & \text{if } (f16\text{In2}) < (\text{tFrac16})0 \\ \dfrac{(\text{INT16})(f16\text{In1} \bullet f16\text{In2})}{\text{INT16\_MAX}+1} & \text{if } (f16\text{In2}) \leq (\text{tFrac16})0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.129.5 Re-entrancy

The function is re-entrant.

4.129.6 Code Example

```
#include "mlib.h"

tF16 f16In1, f16In2;
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In1  = FRAC16 (0.25);

    // scale value = 0.5 = 0x4000
    f16In2  = FRAC16 (0.5);

    // output should be 0.125
    fltOut = MLIB_Convert_FLTF16(f16In1, f16In2);

    // output should be 0.125
    fltOut = MLIB_Convert (f16In1, f16In2, FLTF16);

    // scale value = -0.5 = 0xC000
    f16In2  = FRAC16 (-0.5);

    // output should be 0.5
    fltOut = MLIB_Convert_FLTF16(f16In1, f16In2);

    // output should be 0.5
    fltOut = MLIB_Convert (f16In1, f16In2, FLTF16);
}
```

4.130 Function MLIB_Convert_FLTF32

This function converts the input value to different representation.

4.130.1 Declaration

```
tFloat MLIB_Convert_FLTF32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.130.2 Arguments

Table 4-167. MLIB_Convert_FLTF32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value in 32-bit fractional format to be converted.
register tFrac32	f32In2	**input**	Scale factor in 32-bit fractional format.

4.130.3 Return

Converted input value in single precision floating point format.

4.130.4 Description

This inline function returns converted input value. The input value is considered as 32-bit fractional data type and output value is considered as single precision floating point data type. The input value represents the values in unity model. The second value represents the scale factor and is considered as 32-bit fractional data type. The sign of the second value represents the scale mechanism. In case the second value is positive the first input value is multiplied with the second one and converted to the output format. In case the second value is negative, the first input value is divided by absolute value of second input value and converted to the output format. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$\text{fltOut} = \begin{cases} \dfrac{(\text{INT\_32})f32\text{In1}}{-f32\text{In2}\bullet(\text{INT32\_MAX})} & \text{if } (f32\text{In2}) < (\text{tFrac32})0 \\ \dfrac{(\text{INT32})(f32\text{In1}\bullet f32\text{In2})}{\text{INT32\_MAX}} & \text{if } (f32\text{In2}) \le (\text{tFrac32})0 \end{cases}$$

Equation **MLIB_Convert_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.130.5 Re-entrancy

The function is re-entrant.

4.130.6 Code Example

```
#include "mlib.h"

tF32 f32In1, f32In2;
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000
    f32In1  = FRAC32 (0.25);

    // scale value = 0.5 = 0x4000
    f32In2  = FRAC32 (0.5);

    // output should be 0.125
    fltOut = MLIB_Convert_FLTF32(f32In1, f32In2);

    // output should be 0.125
    fltOut = MLIB_Convert (f32In1, f32In2, FLTF32);

    // scale value = -0.5 = 0xC000
    f32In2  = FRAC32 (-0.5);

    // output should be 0.5
    fltOut = MLIB_Convert_FLTF32(f32In1, f32In2);

    // output should be 0.5
    fltOut = MLIB_Convert (f32In1, f32In2, FLTF32);
}
```

4.131 Function MLIB_ConvertPU_F32F16

This function converts the input value to different representation without scale.

4.131.1 Declaration

```
tFrac32 MLIB_ConvertPU_F32F16(register tFrac16 f16In);
```

4.131.2 Arguments

Table 4-168. MLIB_ConvertPU_F32F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value in 16-bit fractional format to be converted.

4.131.3 Return

Converted input value in 32-bit fractional format.

4.131.4 Description

This inline function returns converted input value. The input value is considered as 16-bit fractional data type and output value is considered as 32-bit fractional data type thus both values represent the values in unity model. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (tFrac32) f16In$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.131.5 Re-entrancy

The function is re-entrant.

4.131.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In  = FRAC16 (0.25);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU_F32F16(f16In);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU (f16In, F32F16);
}
```

4.132 Function MLIB_ConvertPU_F32FLT

This function converts the input value to different representation without scale.

4.132.1 Declaration

```
tFrac32 MLIB_ConvertPU_F32FLT(register tFloat fltIn);
```

4.132.2 Arguments

Table 4-169. MLIB_ConvertPU_F32FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value in single precision floating point format to be converted.

4.132.3 Return

Converted input value in 32-bit fractional format.

4.132.4 Description

This inline function returns converted input value. The input value is considered as single precision floating point data type and output value is considered as 32-bit fractional data type. The output value represents the values in unity model. The output saturation is implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (fltIn) < FRAC32\_MIN \\ \frac{fltIn}{INT32\_MAX} & \text{if} \quad FRAC32\_MIN \le fltIn \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (fltIn) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.132.5 Re-entrancy

The function is re-entrant.

4.132.6 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFrac32 f32Out;

void main(void)
{
    // input value = 0.25
    fltIn  = 0.25;

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU_F32FLT(fltIn);

    // output should be FRAC32(0.25) = 0x20000000
    f32Out = MLIB_ConvertPU (fltIn, F32FLT);
}
```

4.133 Function MLIB_ConvertPU_F16F32

This function converts the input value to different representation without scale.

4.133.1 Declaration

```
tFrac16 MLIB_ConvertPU_F16F32(register tFrac32 f32In);
```

4.133.2 Arguments

Table 4-170. MLIB_ConvertPU_F16F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value in 32-bit fractional format to be converted.

4.133.3 Return

Converted input value in 16-bit fractional format.

4.133.4 Description

This inline function returns converted input value. The input value is considered as 32-bit fractional data type and output value is considered as 16-bit fractional data type thus both values represent the values in unity model. The output saturation is not implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (tFrac16) f32In$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.133.5 Re-entrancy

The function is re-entrant.

4.133.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25 = 0x2000 0000
    f32In  = FRAC32 (0.25);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU_F16F32(f32In);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU (f32In, F16F32);
}
```

4.134 Function MLIB_ConvertPU_F16FLT

This function converts the input value to different representation without scale.

4.134.1 Declaration

```
tFrac16 MLIB_ConvertPU_F16FLT(register tFloat fltIn);
```

4.134.2 Arguments

Table 4-171. MLIB_ConvertPU_F16FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value in single precision floating point format to be converted.

4.134.3 Return

Converted input value in 16-bit fractional format.

4.134.4 Description

This inline function returns converted input value. The input value is considered as single precision floating point data type and output value is considered as 16-bit fractional data type. The output value represents the values in unity model. The output saturation is implemented in this function, thus in case the input value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (fltIn) < FRAC16\_MIN \\ \frac{fltIn}{INT16\_MAX} & \text{if} \quad FRAC16\_MIN \le fltIn \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (fltIn) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.134.5 Re-entrancy

The function is re-entrant.

4.134.6 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFrac16 f16Out;

void main(void)
{
    // input value = 0.25
    fltIn  = 0.25;

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU_F16FLT(fltIn);

    // output should be FRAC16(0.25) = 0x2000
    f16Out = MLIB_ConvertPU (fltIn, F16FLT);
}
```

4.135 Function MLIB_ConvertPU_FLTF16

This function converts the input value to different representation without scale.

4.135.1 Declaration

```
tFloat MLIB_ConvertPU_FLTF16(register tFrac16 f16In);
```

4.135.2 Arguments

Table 4-172. MLIB_ConvertPU_FLTF16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value in 16-bit fractional format to be converted.

4.135.3 Return

Converted input value in single precision floating point format.

4.135.4 Description

This inline function returns converted input value. The input value is considered as 16-bit fractional data type and output value is considered as single precision floating point data type. The input value represents the values in unity model. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$\text{fltOut} = \frac{(\text{INT\_16})f16\text{In}}{\text{INT16\_MAX+1}}$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.135.5 Re-entrancy

The function is re-entrant.

4.135.6 Code Example

```
#include "mlib.h"

tF16 f16In;
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000
    f16In  = FRAC16 (0.25);

    // output should be 0.25
    fltOut = MLIB_ConvertPU_FLTF16(f16In);

    // output should be 0.25
    fltOut = MLIB_ConvertPU (f16In, FLTF16);
}
```

4.136 Function MLIB_ConvertPU_FLTF32

This function converts the input value to different representation without scale.

4.136.1 Declaration

```
tFloat MLIB_ConvertPU_FLTF32(register tFrac32 f32In);
```

4.136.2 Arguments

Table 4-173. MLIB_ConvertPU_FLTF32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value in 32-bit fractional format to be converted.

4.136.3 Return

Converted input value in single precision floating point format.

4.136.4 Description

This inline function returns converted input value. The input value is considered as 32-bit fractional data type and output value is considered as single precision floating point data type. The input value represents the values in unity model. The output saturation is not implemented in this function.

The output of the function is defined by the following simple equation:

$$\text{fltOut} = \frac{(\text{INT\_32})f32\text{In}}{\text{INT32\_MAX}}$$

Equation **MLIB_ConvertPU_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.136.5 Re-entrancy

The function is re-entrant.

4.136.6 Code Example

```
#include "mlib.h"

tF32 f32In;
tFloat fltOut;

void main(void)
{
    // input value = 0.25 = 0x2000 0000
    f32In  = FRAC32 (0.25);

    // output should be 0.25
    fltOut = MLIB_ConvertPU_FLTF32(f32In);

    // output should be 0.25
    fltOut = MLIB_ConvertPU (f32In, FLTF32);
}
```

4.137 Function MLIB_Div_F32

This function divides the first parameter by the second one.

4.137.1 Declaration

```
tFrac32 MLIB_Div_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.137.2 Arguments

Table 4-174. MLIB_Div_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

4.137.3 Return

Division of two input values.

4.137.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the numerator is greater or equal to denominator, the output value is undefined and will overflow without any detection. As the division by zero can be handled differently on each platform and potentially can cause the core exception, the division by zero is handled separately.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} \text{FRAC32\_MIN} & \text{if } \left(f32In2 = 0\right) \& \left(f32In1 \le 0\right) \\ \frac{f32In1}{f32In2} & \text{if } f32In2 \ne 0 \\ \text{FRAC32\_MAX} & \text{if } \left(f32In2 = 0\right) \& \left(f32In1 > 0\right) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

CAUTION

Due to effectivity reason the division is held in 16-bit precision.

4.137.5 Re-entrancy

The function is re-entrant.

4.137.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1  = FRAC32 (0.25);
  // input value 2 = 0.5
  f32In2  = FRAC32 (0.5);
```

```
    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Div_F32(f32In1, f32In2);

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Div (f32In1, f32In2, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be FRAC32(0.5) = 0x40000000
    f32Out = MLIB_Div (f32In1, f32In2);
}
```

4.138 Function MLIB_Div_F16

This function divides the first parameter by the second one.

4.138.1 Declaration

```
tFrac16 MLIB_Div_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.138.2 Arguments

Table 4-175. MLIB_Div_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Numerator of division.
register tFrac16	f16In2	**input**	Denominator of division.

4.138.3 Return

Division of two input values.

4.138.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the numerator is greater or equal to

denominator, the output value is undefined and will overflow without any detection. As the division by zero can be handled differently on each platform and potentially can cause the core exception, the division by zero is handled separately.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} \text{FRAC32\_MIN} & \text{if } (f32In2 = 0) \,\&\, (f32In1 \le 0) \\ \frac{f32In1}{f32In2} & \text{if } f32In2 \ne 0 \\ \text{FRAC32\_MAX} & \text{if } (f32In2 = 0) \,\&\, (f32In1 > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.138.5 Re-entrancy

The function is re-entrant.

4.138.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1  = FRAC16 (0.25);
  // input value 2 = 0.5
  f16In2  = FRAC16 (0.5);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Div_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Div (f16In1, f16In2, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_Div (f16In1, f16In2);
}
```

4.139 Function MLIB_Div_FLT

This function divides the first parameter by the second one.

4.139.1 Declaration

```
tFloat MLIB_Div_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

4.139.2 Arguments

Table 4-176. MLIB_Div_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Numerator of division.
register tFloat	fltIn2	**input**	Denominator of division.

4.139.3 Return

Division of two input values.

4.139.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the division of input values is outside the (-2^{128}, 2^{128}) interval, the output value will overflow without any detection. As the division by zero can be handled differently on each platform and potentially can cause the core exception, the division by zero is handled separately.

The output of the function is defined by the following simple equation:

$$fltOut = \begin{cases} FLOAT\_MIN & \text{if } (fltIn2 = 0) \,\&\, (fltIn1 \le 0) \\ \frac{fltIn1}{fltIn2} & \text{if } fltIn2 \ne 0 \\ FLOAT\_MAX & \text{if } (fltIn2 = 0) \,\&\, (fltIn1 > 0) \end{cases}$$

Equation **MLIB_Div_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.139.5 Re-entrancy

The function is re-entrant.

4.139.6 Code Example

```
#include "mlib.h"

tFloat fltIn1, fltIn2;
tFloat fltOut;

void main(void)
{
  // input value 1 = 0.25
  fltIn1  = 0.25;
  // input value 2 = 0.5
  fltIn2  = 0.5;

  // output should be 0.5
  fltOut = MLIB_Div_FLT(fltIn1, fltIn2);

  // output should be 0.5
  fltOut = MLIB_Div (fltIn1, fltIn2, Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

// output should be 0.5
  fltOut = MLIB_Div (fltIn1, fltIn2);
}
```

4.140 Function MLIB_DivSat_F32

This function divides the first parameter by the second one as saturate.

4.140.1 Declaration

```
tFrac32 MLIB_DivSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.140.2 Arguments

Table 4-177. MLIB_DivSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Numerator of division.
register tFrac32	f32In2	**input**	Denominator of division.

4.140.3 Return

Division of two input values, saturated if necessary.

4.140.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad \frac{f32In1}{f32In2} < FRAC32\_MIN \\ \frac{f32In1}{f32In2} & \text{if} \quad FRAC32\_MIN \le \frac{f32In1}{f32In2} \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad \frac{f32In1}{f32In2} > FRAC32\_MAX \end{cases}$$

Equation **MLIB_DivSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.140.5 Re-entrancy

The function is re-entrant.

4.140.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1, f32In2;
tFrac32 f32Out;

void main(void)
{
  // input value 1 = 0.25
  f32In1  = FRAC32 (0.25);
  // input value 2 = 0.5
  f32In2  = FRAC32 (0.5);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSat_F32(f32In1, f32In2);

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSat (f32In1, f32In2, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.5) = 0x40000000
  f32Out = MLIB_DivSat (f32In1, f32In2);
}
```

4.141 Function MLIB_DivSat_F16

This function divides the first parameter by the second one as saturate.

4.141.1 Declaration

```
tFrac16 MLIB_DivSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.141.2 Arguments

Table 4-178. MLIB_DivSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Numerator of division.
register tFrac16	f16In2	**input**	Denominator of division.

4.141.3 Return

Division of two input values, saturated if necessary.

4.141.4 Description

This inline function returns the division of two input values. The first input value is numerator and the second input value is denominator. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the sum of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad \frac{f16In1}{f16In2} < FRAC16\_MIN \\ \frac{f16In1}{f16In2} & \text{if} \quad FRAC16\_MIN \le \frac{f16In1}{f16In2} \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad \frac{f16In1}{f16In2} > FRAC16\_MAX \end{cases}$$

Equation **MLIB_DivSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.141.5 Re-entrancy

The function is re-entrant.

4.141.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1, f16In2;
tFrac16 f16Out;

void main(void)
{
  // input value 1 = 0.25
  f16In1  = FRAC16 (0.25);
  // input value 2 = 0.5
  f16In2  = FRAC16 (0.5);
```

```
  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_DivSat_F16(f16In1, f16In2);

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_DivSat (f16In1, f16In2, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.5) = 0x4000
  f16Out = MLIB_DivSat (f16In1, f16In2);
}
```

4.142 Function MLIB_Mac_F32

This function implements the multiply accumulate function.

4.142.1 Declaration

```
tFrac32 MLIB_Mac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register tFrac32
f32In3);
```

4.142.2 Arguments

Table 4-179. MLIB_Mac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac32	f32In2	**input**	First value to be multiplied.
register tFrac32	f32In3	**input**	Second value to be multiplied.
register tFrac32	f32In1	**input**	Input value to be add
register tFrac32	f32In2	**input**	First value to be multiplied
register tFrac32	f32In3	**input**	Second value to be multiplied

4.142.3 Return

Multiplied second and third input value with adding of first input value.Multiplied second and third input value with adding of first input value

4.142.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.142.5 Re-entrancy

The function is re-entrant.

4.142.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f32In2  = FRAC32 (0.15);

  // input3 value = 0.35
  f32In3  = FRAC32 (0.35);
```

```
  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac (f32In1, f32In2, f32In3, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac (f32In1, f32In2, f32In3);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.142.7 Re-entrancy

The function is re-entrant.

4.142.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f32In2  = FRAC32 (0.15);

  // input3 value = 0.35
  f32In3  = FRAC32 (0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac (f32In1, f32In2, f32In3, Define F32);
```

```
    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_Mac (f32In1, f32In2, f32In3);
}
```

4.143 Function MLIB_Mac_F32F16F16

This function implements the multiply accumulate function.

4.143.1 Declaration

```
tFrac32 MLIB_Mac_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register tFrac16
f16In3);
```

4.143.2 Arguments

Table 4-180. MLIB_Mac_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.
register tFrac32	f32In1	**input**	Input value to be add
register tFrac16	f16In2	**input**	First value to be multiplied
register tFrac16	f16In3	**input**	Second value to be multiplied

4.143.3 Return

Multiplied second and third input value with adding of first input value.Multiplied second and third input value with adding of first input value

4.143.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The first input value as well as output value is considered as 32-bit fractional values. The second and third input values are considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

This implementation is available if 32-bit fractional implementations are enabled. However it is not possible to use the default implementation based function call, thus the implementation post-fix or additional parameter function call shall be used.

This inline function returns the multiplied second and third input value with adding of first input value. The first input value as well as output value is considered as 32-bit fractional values. The second and third input values are considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.143.5 Re-entrancy

The function is re-entrant.

4.143.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);
```

```
  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac_F32F16F16(f32In1, f16In2, f16In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_Mac (f32In1, f32In2, f32In3, F32F16F16);

}
```

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

This implementation is available if 32-bit fractional implementations are enabled. However it is not possible to use the default implementation based function call, thus the implementation post-fix or additional parameter function call shall be used.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.143.7 Re-entrancy

The function is re-entrant.

4.143.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
```

```
    f32Out = MLIB_Mac_F32F16F16(f32In1, f16In2, f16In3);

    // output should be FRAC32(0.3025) = 0x26B851EB
    f32Out = MLIB_Mac (f32In1, f32In2, f32In3, F32F16F16);

}
```

4.144 Function MLIB_Mac_F16

This function implements the multiply accumulate function.

4.144.1 Declaration

```
tFrac16 MLIB_Mac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register tFrac16
f16In3);
```

4.144.2 Arguments

Table 4-181. MLIB_Mac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.
register tFrac16	f16In1	**input**	Input value to be add
register tFrac16	f16In2	**input**	First value to be multiplied
register tFrac16	f16In3	**input**	Second value to be multiplied

4.144.3 Return

Multiplied second and third input value with adding of first input value.Multiplied second and third input value with adding of first input value

4.144.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.144.5 Re-entrancy

The function is re-entrant.

4.144.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);
```

```
  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac (f16In1, f16In2, f16In3, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac (f16In1, f16In2, f16In3);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right)$$

Equation **MLIB_Mac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.144.7 Re-entrancy

The function is re-entrant.

4.144.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_Mac (f16In1, f16In2, f16In3, Define F16);
```

```
    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC16(0.3025) = 0x26B8
    f16Out = MLIB_Mac (f16In1, f16In2, f16In3);
}
```

4.145 Function MLIB_Mac_FLT

This function implements the multiply accumulate function.

4.145.1 Declaration

```
tFloat MLIB_Mac_FLT(register tFloat fltIn1, register tFloat fltIn2, register tFloat fltIn3);
```

4.145.2 Arguments

Table 4-182. MLIB_Mac_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Input value to be add.
register tFloat	fltIn2	**input**	First value to be multiplied.
register tFloat	fltIn3	**input**	Second value to be multiplied.
register tFloat	fltIn1	**input**	Input value to be add
register tFloat	fltIn2	**input**	First value to be multiplied
register tFloat	fltIn3	**input**	Second value to be multiplied

4.145.3 Return

Multiplied second and third input value with adding of first input value.Multiplied second and third input value with adding of first input value

4.145.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the output value is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 + (fltIn2 \bullet fltIn3))$$

Equation **MLIB_Mac_Eq1**

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the output value is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.145.5 Re-entrancy

The function is re-entrant.

4.145.6 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltOut;

void main(void)
{
  // input1 value = 0.25
  fltIn1  = (0.25);

  // input2 value = 0.15
  fltIn2  = (0.15);

  // input3 value = 0.35
  fltIn3  = (0.35);
```

```
  // output should be 0.3025
  fltOut = MLIB_Mac_FLT(fltIn1, fltIn2, fltIn3);

  // output should be 0.3025
  fltOut = MLIB_Mac (fltIn1, fltIn2, fltIn3, Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 0.3025
  fltOut = MLIB_Mac (fltIn1, fltIn2, fltIn3);
}
```

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 + (fltIn2 \bullet fltIn3))$$

Equation **MLIB_Mac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.145.7 Re-entrancy

The function is re-entrant.

4.145.8 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltOut;

void main(void)
{
  // input1 value = 0.25
  fltIn1  = (0.25);

  // input2 value = 0.15
  fltIn2  = (0.15);

  // input3 value = 0.35
  fltIn3  = (0.35);

  // output should be 0.3025
  fltOut = MLIB_Mac_FLT(fltIn1, fltIn2, fltIn3);

  // output should be 0.3025
  fltOut = MLIB_Mac (fltIn1, fltIn2, fltIn3, Define FLT);
```

```
    // ###############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ###############################################################

    // output should be 0.3025
    fltOut = MLIB_Mac (fltIn1, fltIn2, fltIn3);
}
```

4.146 Function MLIB_MacSat_F32

This function implements the multiply accumulate function saturated if necessary.

4.146.1 Declaration

```
tFrac32 MLIB_MacSat_F32(register tFrac32 f32In1, register tFrac32 f32In2, register tFrac32
f32In3);
```

4.146.2 Arguments

Table 4-183. MLIB_MacSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac32	f32In2	**input**	First value to be multiplied.
register tFrac32	f32In3	**input**	Second value to be multiplied.

4.146.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

4.146.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 32-bit fractional values. The output saturation is implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if } \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right) < FRAC32\_MIN \\ \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right) & \text{if } FRAC32\_MIN \le \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if } \left(f32In1 + \left(f32In2 \bullet f32In3\right)\right) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.146.5 Re-entrancy

The function is re-entrant.

4.146.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f32In2  = FRAC32 (0.15);

  // input3 value = 0.35
  f32In3  = FRAC32 (0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat_F32(f32In1, f32In2, f32In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat (f32In1, f32In2, f32In3, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat (f32In1, f32In2, f32In3);
}
```

4.147 Function MLIB_MacSat_F32F16F16

This function implements the multiply accumulate function saturated if necessary.

4.147.1 Declaration

```
tFrac32 MLIB_MacSat_F32F16F16(register tFrac32 f32In1, register tFrac16 f16In2, register
tFrac16 f16In3);
```

4.147.2 Arguments

Table 4-184. MLIB_MacSat_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

4.147.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

4.147.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The first input values as well as output value is considered as 32-bit fractional values, second and third input values are considered as 16-bit fractional values. The output saturation is implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad \left(f32In1 + \left(f16In2 \bullet f16In3\right)\right) < FRAC32\_MIN \\ \left(f32In1 + \left(f16In2 \bullet f16In3\right)\right) & \text{if} \quad FRAC32\_MIN \le \left(f32In1 + \left(f16In2 \bullet f16In3\right)\right) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad \left(f32In1 + \left(f16In2 \bullet f16In3\right)\right) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.147.5 Re-entrancy

The function is re-entrant.

4.147.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat_F32F16F16(f32In1, f16In2, f16In3);

  // output should be FRAC32(0.3025) = 0x26B851EB
  f32Out = MLIB_MacSat (f32In1, f16In2, f16In3, F32F16F16);
}
```

4.148 Function MLIB_MacSat_F16

This function implements the multiply accumulate function saturated if necessary.

4.148.1 Declaration

```
tFrac16 MLIB_MacSat_F16(register tFrac16 f16In1, register tFrac16 f16In2, register tFrac16
f16In3);
```

4.148.2 Arguments

Table 4-185. MLIB_MacSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Input value to be add.
register tFrac16	f16In2	**input**	First value to be multiplied.
register tFrac16	f16In3	**input**	Second value to be multiplied.

4.148.3 Return

Multiplied second and third input value with adding of first input value. The output value is saturated if necessary.

4.148.4 Description

This inline function returns the multiplied second and third input value with adding of first input value. The input values as well as output value is considered as 16-bit fractional values. The output saturation is implemented in this function, thus in case the output value is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if } \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right) < FRAC16\_MIN \\ \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right) & \text{if } FRAC16\_MIN \le \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if } \left(f16In1 + \left(f16In2 \bullet f16In3\right)\right) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_MacSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.148.5 Re-entrancy

The function is re-entrant.

4.148.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_MacSat_F16(f16In1, f16In2, f16In3);

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_MacSat (f16In1, f16In2, f16In3, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(0.3025) = 0x26B8
  f16Out = MLIB_MacSat (f16In1, f16In2, f16In3);
}
```

4.149 Function MLIB_Mul_F32

This function multiplies two input parameters.

4.149.1 Declaration

```
tFrac32 MLIB_Mul_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.149.2 Arguments

Table 4-186. MLIB_Mul_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1).

Table continues on the next page...

Table 4-186. MLIB_Mul_F32 arguments (continued)

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1)
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1)

4.149.3 Return

Fractional multiplication of the input arguments.Fractional multiplication of the input arguments

4.149.4 Description

Fractional multiplication of two fractional 32-bit values. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 \bullet f32In2$$

Equation **MLIB_Mul_Eq1**

Fractional multiplication of two fractional 32-bit values. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.149.5 Re-entrancy

The function is re-entrant.

4.149.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1  = FRAC32 (0.5);

  // second input = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul_F32(f32In1,f32In2);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul (f32In1,f32In2,Define F32);


  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul (f32In1,f32In2);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = f32In1 \bullet f32In2$$

Equation **MLIB_Mul_Eq1**

Note

Overflow is not detected.

4.149.7 Re-entrancy

The function is re-entrant.

4.149.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;
```

```
void main(void)
{
  // first input = 0.5
  f32In1  = FRAC32 (0.5);

  // second input = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul_F32(f32In1,f32In2);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul (f32In1,f32In2,Define F32);


  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul (f32In1,f32In2);
}
```

4.150 Function MLIB_Mul_F32F16F16

This function multiplies two input parameters.

4.150.1 Declaration

```
tFrac32 MLIB_Mul_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.150.2 Arguments

Table 4-187. MLIB_Mul_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1)
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1)

4.150.3 Return

Fractional multiplication of the input arguments.Fractional multiplication of the input arguments

4.150.4 Description

Fractional multiplication of two fractional 16-bit values. The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = f16In1 \bullet f16In2$$

Equation **MLIB_Mul_Eq1**

Fractional multiplication of two fractional 16-bit values. The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.150.5 Re-entrancy

The function is re-entrant.

4.150.6 Code Example

```
#include "mlib.h"

tFrac32 f16In1;
tFrac32 f16In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul_F32F16F16(f16In1, f16In2);
```

```
  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul (f16In1, f16In2, F32F16F16);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = f16In1 \bullet f16In2$$

Equation **MLIB_Mul_Eq1**

Note

Overflow is not detected.

4.150.7 Re-entrancy

The function is re-entrant.

4.150.8 Code Example

```
#include "mlib.h"

tFrac32 f16In1;
tFrac32 f16In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul_F32F16F16(f16In1, f16In2);

  // output should be 0x10000000 = FRAC32(0.125)
  f32Out = MLIB_Mul (f16In1, f16In2, F32F16F16);
}
```

4.151 Function MLIB_Mul_F16

This function multiplies two input parameters.

4.151.1 Declaration

```
tFrac16 MLIB_Mul_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.151.2 Arguments

Table 4-188. MLIB_Mul_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1)
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1)

4.151.3 Return

Fractional multiplication of the input arguments.Fractional multiplication of the input arguments

4.151.4 Description

Fractional multiplication of two fractional 16-bit values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 \bullet f16In2$$

Equation **MLIB_Mul_Eq1**

Fractional multiplication of two fractional 16-bit values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.151.5 Re-entrancy

The function is re-entrant.

4.151.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul_F16(f16In1,f16In2);

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul (f16In1,f16In2,Define F16);


  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul (f16In1,f16In2);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = f16In1 \bullet f16In2$$

Equation **MLIB_Mul_Eq1**

Note

Overflow is not detected.

4.151.7 Re-entrancy

The function is re-entrant.

4.151.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
```

```
void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul_F16(f16In1,f16In2);

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul (f16In1,f16In2,Define F16);


  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x1000 = FRAC16(0.125)
  f16Out = MLIB_Mul (f16In1,f16In2);
}
```

4.152 Function MLIB_Mul_FLT

This function multiplies two input parameters.

4.152.1 Declaration

```
tFloat MLIB_Mul_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

4.152.2 Arguments

Table 4-189. MLIB_Mul_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Operand is a single precision floating point number.
register tFloat	fltIn2	**input**	Operand is a single precision floating point number.
register tFloat	fltIn1	**input**	operand is a single precision floating point number
register tFloat	fltIn2	**input**	operand is a single precision floating point number

4.152.3 Return

Floating point multiplication of the input arguments.Floating point multiplication of the input arguments

4.152.4 Description

Single precision floating point multiplication of two input values. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$fltOut = fltIn1 \bullet fltIn2$$

Equation **MLIB_Mul_Eq1**

Single precision floating point multiplication of two input values. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the multiplication of input values is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.152.5 Re-entrancy

The function is re-entrant.

4.152.6 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;

void main(void)
{
  // first input = 50.5
  fltIn1  = 50.5;

  // second input = 25.25
  fltIn2  = 25.25;

  // output should be 1275.125
  fltOut = MLIB_Mul_FLT(fltIn1,fltIn2);

  // output should be 1275.125
```

```
  fltOut = MLIB_Mul (fltIn1,fltIn2,Define FLT);


  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 1275.125
  fltOut = MLIB_Mul (fltIn1,fltIn2);
}
```

The output of the function is defined by the following simple equation:

$$fltOut = fltIn1 \bullet fltIn2$$

Equation **MLIB_Mul_Eq1**

Note

Overflow is not detected.

4.152.7 Re-entrancy

The function is re-entrant.

4.152.8 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;

void main(void)
{
  // first input = 50.5
  fltIn1  = 50.5;

  // second input = 25.25
  fltIn2  = 25.25;

  // output should be 1275.125
  fltOut = MLIB_Mul_FLT(fltIn1,fltIn2);

  // output should be 1275.125
  fltOut = MLIB_Mul (fltIn1,fltIn2,Define FLT);


  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 1275.125
  fltOut = MLIB_Mul (fltIn1,fltIn2);
}
```

4.153 Function MLIB_MulSat_F32

This function multiplies two input parameters and saturate if necessary.

4.153.1 Declaration

```
tFrac32 MLIB_MulSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.153.2 Arguments

Table 4-190. MLIB_MulSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1)
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1)

4.153.3 Return

Fractional multiplication of the input arguments.Fractional multiplication of the input arguments.

4.153.4 Description

Fractional multiplication of two fractional 32-bit values. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 \bullet f32In2) < FRAC32\_MIN \\ (f32In1 \bullet f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 \bullet f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 \bullet f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Fractional multiplication of two fractional 32-bit values. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

Note

Overflow is detected. The functions saturates the return value if it cannot fit into the return type.

4.153.5 Re-entrancy

The function is re-entrant.

4.153.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.8
  f32In1  = FRAC32 (0.8);

  // second input = 0.75
  f32In2  = FRAC32 (0.75);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32(f32In1,f32In2);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2,Define F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 \bullet f32In2) < FRAC32\_MIN \\ (f32In1 \bullet f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 \bullet f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 \bullet f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Overflow is detected. The functions saturates the return value if it cannot fit into the return type.

4.153.7 Re-entrancy

The function is re-entrant.

4.153.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.8
  f32In1  = FRAC32 (0.8);

  // second input = 0.75
  f32In2  = FRAC32 (0.75);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32(f32In1,f32In2);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2,Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2);
}
```

4.154 Function MLIB_MulSat_F32F16F16

This function multiplies two input parameters and saturate if necessary.

4.154.1 Declaration

```
tFrac32 MLIB_MulSat_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.154.2 Arguments

Table 4-191. MLIB_MulSat_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1)
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1)

4.154.3 Return

Fractional multiplication of the input arguments.Fractional multiplication of the input arguments.

4.154.4 Description

Fractional multiplication of two fractional 16-bit values. The input values are considered as 16-bit fractional data type and the output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f16In1 \bullet f16In2) < FRAC32\_MIN \\ (f16In1 \bullet f16In2) & \text{if} \quad FRAC32\_MIN \le (f16In1 \bullet f16In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f16In1 \bullet f16In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Fractional multiplication of two fractional 16-bit values. The input values are considered as 16-bit fractional data type and the output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

Note

Overflow is detected. The functions saturates the return value if it cannot fit into the return type.

4.154.5 Re-entrancy

The function is re-entrant.

4.154.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.8
  f16In1  = FRAC16 (0.8);

  // second input = 0.75
  f16In2  = FRAC16 (0.75);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32F16F16(f16In1,f16In2);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2,F32F16f16);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f16In1 \bullet f16In2) < FRAC32\_MIN \\ (f16In1 \bullet f16In2) & \text{if} \quad FRAC32\_MIN \le (f16In1 \bullet f16In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f16In1 \bullet f16In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Overflow is detected. The functions saturates the return value if it cannot fit into the return type.

4.154.7 Re-entrancy

The function is re-entrant.

4.154.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.8
  f16In1  = FRAC16 (0.8);

  // second input = 0.75
  f16In2  = FRAC16 (0.75);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat_F32F16F16(f16In1,f16In2);

  // output should be 0x4ccccccc = FRAC32(0.6)
  f32Out = MLIB_MulSat (f32In1,f32In2,F32F16f16);
}
```

4.155 Function MLIB_MulSat_F16

This function multiplies two input parameters and saturate if necessary.

4.155.1 Declaration

```
tFrac16 MLIB_MulSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.155.2 Arguments

Table 4-192. MLIB_MulSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1)
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1)

4.155.3 Return

Fractional multiplication of the input arguments.Fractional multiplication of the input arguments.

4.155.4 Description

Fractional multiplication of two fractional 16-bit values. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if } (f16In1 \bullet f16In2) < FRAC16\_MIN \\ (f16In1 \bullet f16In2) & \text{if } FRAC16\_MIN \le (f16In1 \bullet f16In2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if } (f16In1 \bullet f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Fractional multiplication of two fractional 16-bit values. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the multiplication of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

Note

Overflow is detected. The functions saturates the return value if it cannot fit into the return type.

4.155.5 Re-entrancy

The function is re-entrant.

4.155.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;
```

```
void main(void)
{
  // first input = 0.8
  f16In1  = FRAC16 (0.8);

  // second input = 0.75
  f16In2  = FRAC16 (0.75);

  // output should be 0x4ccc = FRAC16(0.6)
  f16Out = MLIB_MulSat_F16(f16In1,f16In2);

  // output should be 0x4ccc = FRAC16(0.6)
  f16Out = MLIB_MulSat (f16In1,f16In2,Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x4ccc = FRAC32(0.6)
  f16Out = MLIB_MulSat (f16In1,f16In2);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 \bullet f16In2) < FRAC16\_MIN \\ (f16In1 \bullet f16In2) & \text{if} \quad FRAC16\_MIN \le (f16In1 \bullet f16In2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 \bullet f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_MulSat_Eq1**

Note

Overflow is detected. The functions saturates the return value if it cannot fit into the return type.

4.155.7 Re-entrancy

The function is re-entrant.

4.155.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.8
  f16In1  = FRAC16 (0.8);

  // second input = 0.75
```

```
    f16In2  = FRAC16 (0.75);

    // output should be 0x4ccc = FRAC16(0.6)
    f16Out = MLIB_MulSat_F16(f16In1,f16In2);

    // output should be 0x4ccc = FRAC16(0.6)
    f16Out = MLIB_MulSat (f16In1,f16In2,Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x4ccc = FRAC32(0.6)
    f16Out = MLIB_MulSat (f16In1,f16In2);
}
```

4.156 Function MLIB_Neg_F32

This function returns negative value of input parameter.

4.156.1 Declaration

```
tFrac32 MLIB_Neg_F32(register tFrac32 f32In);
```

4.156.2 Arguments

Table 4-193. MLIB_Neg_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value which negative value should be returned.

4.156.3 Return

Negative value of input parameter.

4.156.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = -(f32In)$$

Equation **MLIB_Neg_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.156.5 Re-entrancy

The function is re-entrant.

4.156.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
  // input value = 0.25
  f32In  = FRAC32 (0.25);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_Neg_F32(f32In);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_Neg (f32In, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_Neg (f32In);
}
```

4.157 Function MLIB_Neg_F16

This function returns negative value of input parameter.

4.157.1 Declaration

```
tFrac16 MLIB_Neg_F16(register tFrac16 f16In);
```

4.157.2 Arguments

Table 4-194. MLIB_Neg_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value which negative value should be returned.

4.157.3 Return

Negative value of input parameter.

4.157.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = -(f16In)$$

Equation **MLIB_Neg_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.157.5 Re-entrancy

The function is re-entrant.

4.157.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
  // input value = 0.25
  f16In  = FRAC16 (0.25);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg_F16(f16In);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg (f16In, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_Neg (f16In);
}
```

4.158 Function MLIB_Neg_FLT

This function returns negative value of input parameter.

4.158.1 Declaration

```
tFloat MLIB_Neg_FLT(register tFloat fltIn);
```

4.158.2 Arguments

Table 4-195. MLIB_Neg_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn	**input**	Input value which negative value should be returned.

4.158.3 Return

Negative value of input parameter.

4.158.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the negation of input values is outside the (-2 ^{128}, 2 ^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$fltOut = -(fltIn)$$

Equation **MLIB_Neg_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.158.5 Re-entrancy

The function is re-entrant.

4.158.6 Code Example

```
#include "mlib.h"

tFloat fltIn;
tFloat fltOut;

void main(void)
{
  // input value = 0.25
  fltIn  = (0.25);

  // output should be (-0.25)
  fltOut = MLIB_Neg_FLT(fltIn);

  // output should be (-0.25)
  fltOut = MLIB_Neg (fltIn, Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be (-0.25)
  fltOut = MLIB_Neg (fltIn);
}
```

4.159 Function MLIB_NegSat_F32

This function returns negative value of input parameter and saturate if necessary.

4.159.1 Declaration

```
tFrac32 MLIB_NegSat_F32(register tFrac32 f32In);
```

4.159.2 Arguments

Table 4-196. MLIB_NegSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	Input value which negative value should be returned.

4.159.3 Return

Negative value of input parameter.

4.159.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad -(f32In) < FRAC32\_MIN \\ -(f32In) & \text{if} \quad FRAC32\_MIN \le -(f32In) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad -(f32In) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_NegSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.159.5 Re-entrancy

The function is re-entrant.

4.159.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tFrac32 f32Out;

void main(void)
{
  // input value = 0.25
  f32In  = FRAC32 (0.25);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_NegSat_F32(f32In);

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_NegSat (f32In, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(-0.25) = 0xA0000000
  f32Out = MLIB_NegSat (f32In);
}
```

4.160 Function MLIB_NegSat_F16

This function returns negative value of input parameter and saturate if necessary.

4.160.1 Declaration

```
tFrac16 MLIB_NegSat_F16(register tFrac16 f16In);
```

4.160.2 Arguments

Table 4-197. MLIB_NegSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	Input value which negative value should be returned.

4.160.3 Return

Negative value of input parameter.

4.160.4 Description

This inline function returns the negative value of input parameter. The input values as well as output value is considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the negation of input values is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad -(f16In) < FRAC16\_MIN \\ -(f16In) & \text{if} \quad FRAC16\_MIN \le -(f16In) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad -(f16In) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_NegSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.160.5 Re-entrancy

The function is re-entrant.

4.160.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tFrac16 f16Out;

void main(void)
{
  // input value = 0.25
  f16In  = FRAC16 (0.25);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_NegSat_F16(f16In);

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_NegSat (f16In, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC16(-0.25) = 0xA000
  f16Out = MLIB_NegSat (f16In);
}
```

4.161 Function MLIB_Norm_F32

This function returns the number of left shifts needed to normalize the input parameter.

4.161.1 Declaration

```
tU16 MLIB_Norm_F32(register tFrac32 f32In);
```

4.161.2 Arguments

Table 4-198. MLIB_Norm_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In	**input**	The first value to be normalized.

4.161.3 Return

The number of left shift needed to normalize the argument. For the input "0" returns "0".

4.161.4 Description

Depending on the sign of the input value the function counts and returns the number of the left shift needed to get an equality between input value and the maximum fractional values "1" or "-1".

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.161.5 Re-entrancy

The function is re-entrant.

4.161.6 Code Example

```
#include "mlib.h"

tFrac32 f32In;
tU16 u16Out;

void main(void)
{
    // first input = 0.00005
    f32In = FRAC32 (0.00005);

    // output should be 14
    u16Out = MLIB_Norm_F32(f32In);

    // output should be 14
    u16Out = MLIB_Norm (f32In,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 14
    u16Out = MLIB_Norm (f32In);
}
```

4.162 Function MLIB_Norm_F16

This function returns the number of left shifts needed to normalize the input parameter.

4.162.1 Declaration

```
tU16 MLIB_Norm_F16(register tFrac16 f16In);
```

4.162.2 Arguments

Table 4-199. MLIB_Norm_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In	**input**	The first value to be normalized.

4.162.3 Return

The number of left shift needed to normalize the argument. For the input "0" returns "0".

4.162.4 Description

Depending on the sign of the input value the function counts and returns the number of the left shift needed to get an equality between input value and the maximum fractional values "1" or "-1".

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.162.5 Re-entrancy

The function is re-entrant.

4.162.6 Code Example

```
#include "mlib.h"

tFrac16 f16In;
tU16 u16Out;

void main(void)
{
    // first input = 0.00005
    f16In = FRAC16 (0.00005);
```

```
    // output should be 14
    u16Out = MLIB_Norm_F16(f16In);

    // output should be 14
    u16Out = MLIB_Norm (f16In,Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 14
    u16Out = MLIB_Norm (f16In);
}
```

4.163 Function MLIB_Round_F32

The function rounds the first input value for number of digits defined by second parameter and saturate automatically.

4.163.1 Declaration

```
tFrac32 MLIB_Round_F32(register tFrac32 f32In1, register tU16 u16In2);
```

4.163.2 Arguments

Table 4-200. MLIB_Round_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	The first value to be rounded.
register tU16	u16In2	**input**	The round digits amount.

4.163.3 Return

32-bit fractional value rounded to the nearest n-bit fractional value where "n" is defined by the second input value. The bits beyond the 16-bit boundary are discarded.

4.163.4 Description

This function rounds the first argument to nearest value defined by the number of bits defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The round amount cannot exceed in magnitude the bit-width of the rounded value, that means must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.163.5 Re-entrancy

The function is re-entrant.

4.163.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.5
    f32In1 = FRAC32 (0.5);
    // second input = 29
    u16In2 = 29;

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = MLIB_Round_F32(f32In1,u16In2);

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = MLIB_Round (f32In1,u16In2,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x60000000 ~ FRAC32(0.75)
    f32Out = MLIB_Round (f32In1,u16In2);
}
```

4.164 Function MLIB_Round_F16

The function rounds the first input value for number of digits defined by second parameter and saturate automatically.

4.164.1 Declaration

```
tFrac16 MLIB_Round_F16(register tFrac16 f16In1, register tU16 u16In2);
```

4.164.2 Arguments

Table 4-201. MLIB_Round_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	The first value to be rounded.
register tU16	u16In2	**input**	The round digits amount.

4.164.3 Return

16-bit fractional value rounded to the nearest n-bit fractional value where "n" is defined by the second input value. The bits beyond the 16-bit boundary are discarded.

4.164.4 Description

This function rounds the first argument to nearest value defined by the number of bits defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The round amount cannot exceed in magnitude the bit-width of the rounded value, that means must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.164.5 Re-entrancy

The function is re-entrant.

4.164.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.5
    f16In1 = FRAC16 (0.5);
    // second input = 13
    u16In2 = 13;

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = MLIB_Round_F16(f16In1,u16In2);

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = MLIB_Round (f16In1,u16In2,Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x6000 ~ FRAC16(0.75)
    f16Out = MLIB_Round (f16In1,u16In2);
}
```

4.165 Function MLIB_ShBi_F32

This function shifts the first argument to left or right by number defined by second argument.

4.165.1 Declaration

```
tFrac32 MLIB_ShBi_F32(register tFrac32 f32In1, register tS16 s16In2);
```

4.165.2 Arguments

Table 4-202. MLIB_ShBi_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be shift.
register tS16	s16In2	**input**	The shift amount value.

4.165.3 Return

32-bit fractional value shifted to left or right by the shift amount. The bits beyond the 32-bit boundary are discarded.

4.165.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -31...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.165.5 Re-entrancy

The function is re-entrant.

4.165.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBi_F32(f32In1, s16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBi (f32In1, s16In2, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################
```

```
    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBi (f32In1, s16In2);
}
```

4.166 Function MLIB_ShBi_F16

This function shifts the first argument to left or right by number defined by second argument.

4.166.1 Declaration

```
tFrac16 MLIB_ShBi_F16(register tFrac16 f16In1, register tS16 s16In2);
```

4.166.2 Arguments

Table 4-203. MLIB_ShBi_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tS16	s16In2	**input**	The shift amount value.

4.166.3 Return

16-bit fractional value shifted to left or right by the shift amount. The bits beyond the 16-bit boundary are discarded.

4.166.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -15...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.166.5 Re-entrancy

The function is re-entrant.

4.166.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi_F16(f16In1, s16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi (f16In1, s16In2, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBi (f16In1, s16In2);
}
```

4.167 Function MLIB_ShBiSat_F32

This function shifts the first argument to left or right by number defined by second argument and saturate if necessary.

4.167.1 Declaration

```
tFrac32 MLIB_ShBiSat_F32(register tFrac32 f32In1, register tS16 s16In2);
```

4.167.2 Arguments

Table 4-204. MLIB_ShBiSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be shift.
register tS16	s16In2	**input**	The shift amount value.

4.167.3 Return

32-bit fractional value shifted to left or right by the shift amount. The bits beyond the 32-bit boundary are discarded.

4.167.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if an overflow is detected. If the sign of second parameter is negative, shift to right.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -31...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.167.5 Re-entrancy

The function is re-entrant.

4.167.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
```

```
    f32In1 = FRAC32 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat_F32(f32In1, s16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat (f32In1, s16In2, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShBiSat (f32In1, s16In2);
}
```

4.168 Function MLIB_ShBiSat_F16

This function shifts the first argument to left or right by number defined by second argument and saturate if necessary.

4.168.1 Declaration

```
tFrac16 MLIB_ShBiSat_F16(register tFrac16 f16In1, register tS16 s16In2);
```

4.168.2 Arguments

Table 4-205. MLIB_ShBiSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tS16	s16In2	**input**	The shift amount value.

4.168.3 Return

16-bit fractional value shifted to left or right by the shift amount. The bits beyond the 16-bit boundary are discarded.

4.168.4 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if an overflow is detected. If the sign of second parameter is negative, shift to right.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range -15...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.168.5 Re-entrancy

The function is re-entrant.

4.168.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tS16 s16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = -1
    s16In2 = -1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat_F16(f16In1, s16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat (f16In1, s16In2, Define F16);

    // ##############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShBiSat (f16In1, s16In2);
}
```

4.169 Function MLIB_ShL_F32

This function shifts the first parameter to left by number defined by second parameter.

4.169.1 Declaration

```
tFrac32 MLIB_ShL_F32(register tFrac32 f32In1, register tU16 u16In2);
```

4.169.2 Arguments

Table 4-206. MLIB_ShL_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

4.169.3 Return

32-bit fractional value shifted to left by the shift amount. The bits beyond the 32-bit boundary are discarded.

4.169.4 Description

This function shifts the first argument to left by number defined by second argument. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.169.5 Re-entrancy

The function is re-entrant.

4.169.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL_F32(f32In1, u16In2);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL (f32In1, u16In2, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShL (f32In1, u16In2);
}
```

4.170 Function MLIB_ShL_F16

This function shifts the first parameter to left by number defined by second parameter.

4.170.1 Declaration

```
tFrac16 MLIB_ShL_F16(register tFrac16 f16In1, register tU16 u16In2);
```

4.170.2 Arguments

Table 4-207. MLIB_ShL_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

4.170.3 Return

16-bit fractional value shifted to left by the shift amount. The bits beyond the 16-bit boundary are discarded.

4.170.4 Description

This function shifts the first argument to left by number defined by second argument. Overflow is not detected.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.170.5 Re-entrancy

The function is re-entrant.

4.170.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShL_F16(f16In1, u16In2);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShL (f16In1, u16In2, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################
```

```
    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShL (f16In1, u16In2);
}
```

4.171 Function MLIB_ShLSat_F32

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

4.171.1 Declaration

```
tFrac32 MLIB_ShLSat_F32(register tFrac32 f32In1, register tU16 u16In2);
```

4.171.2 Arguments

Table 4-208. MLIB_ShLSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

4.171.3 Return

32-bit fractional value shifted to left by the shift amount. The bits beyond the 32-bit boundary are discarded.

4.171.4 Description

This function shifts the first argument to left by number defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.171.5 Re-entrancy

The function is re-entrant.

4.171.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat_F32(f32In1, u16In2);

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat (f32In1, u16In2, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x40000000 ~ FRAC32(0.5)
    f32Out = MLIB_ShLSat (f32In1, u16In2);
}
```

4.172 Function MLIB_ShLSat_F16

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

4.172.1 Declaration

```
tFrac16 MLIB_ShLSat_F16(register tFrac16 f16In1, register tU16 u16In2);
```

4.172.2 Arguments

Table 4-209. MLIB_ShLSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be left shift.
register tU16	u16In2	**input**	The shift amount value.

4.172.3 Return

16-bit fractional value shifted to left by the shift amount. The bits beyond the 16-bit boundary are discarded.

4.172.4 Description

This function shifts the first argument to left by number defined by second argument and saturate if an overflow is detected. The function returns a saturated fractional value if the return value cannot fit into the return type.

Note

The shift amount cannot exceed in magnitude the bit-width of the shift value, that means must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.172.5 Re-entrancy

The function is re-entrant.

4.172.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
```

```
    f16In1 = FRAC16 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat_F16(f16In1, u16In2);

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat (f16In1, u16In2, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x4000 ~ FRAC16(0.5)
    f16Out = MLIB_ShLSat (f16In1, u16In2);
}
```

4.173 Function MLIB_ShR_F32

This function shifts the first parameter to right by number defined by second parameter.

4.173.1 Declaration

```
tFrac32 MLIB_ShR_F32(register tFrac32 f32In1, register tU16 u16In2);
```

4.173.2 Arguments

Table 4-210. MLIB_ShR_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First value to be right shift.
register tU16	u16In2	**input**	The shift amount value.

4.173.3 Return

32-bit fractional value shifted right by the shift amount. The bits beyond the 32-bit boundary of the result are discarded.

4.173.4 Description

This function shifts the first argument to right by number defined by second argument.

Note

The shift amount cannot exceed in magnitude the bit-width of the shifted value, that means it must be within the range 0...31. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.173.5 Re-entrancy

The function is re-entrant.

4.173.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f32In1 = FRAC32 (0.25);
    // second input = 1
    u16In2 = 1;

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShR_F32(f32In1, u16In2);

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShR (f32In1, u16In2, Define F32);

    // ##############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ##############################################################

    // output should be 0x10000000 ~ FRAC32(0.125)
    f32Out = MLIB_ShR (f32In1, u16In2);
}
```

4.174 Function MLIB_ShR_F16

This function shifts the first parameter to right by number defined by second parameter.

4.174.1 Declaration

```
tFrac16 MLIB_ShR_F16(register tFrac16 f16In1, register tU16 u16In2);
```

4.174.2 Arguments

Table 4-211. MLIB_ShR_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First value to be right shift.
register tU16	u16In2	**input**	The shift amount value.

4.174.3 Return

16-bit fractional value shifted right by the shift amount. The bits beyond the 16-bit boundary of the result are discarded.

4.174.4 Description

This function shifts the first argument to right by number defined by second argument.

Note

The shift amount cannot exceed in magnitude the bit-width of the shifted value, that means it must be within the range 0...15. Otherwise the result of the function is undefined. Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.174.5 Re-entrancy

The function is re-entrant.

4.174.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16Out;
tU16 u16In2;

void main(void)
{
    // first input = 0.25
    f16In1 = FRAC16 (0.25);
    // second input = 1
```

```
    u16In2 = 1;

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR_F16(f16In1, u16In2);

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR (f16In1, u16In2, Define F16);

    // ###############################################################
    // Available only if 16-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x1000 ~ FRAC16(0.125)
    f16Out = MLIB_ShR (f16In1, u16In2);
}
```

4.175 Function MLIB_Sub_F32

This function subtracts the second parameter from the first one.

4.175.1 Declaration

```
tFrac32 MLIB_Sub_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.175.2 Arguments

Table 4-212. MLIB_Sub_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between[-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between[-1,1).
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between[-1,1)
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between[-1,1)

4.175.3 Return

The subtraction of the second argument from the first argument.The subtraction of the second argument from the first argument.

4.175.4 Description

Subtraction of two fractional 32-bit values. The second argument is subtracted from the first one. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 - f32In2)$$

Equation **MLIB_Sub_Eq1**

Subtraction of two fractional 32-bit values. The second argument is subtracted from the first one. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.175.5 Re-entrancy

The function is re-entrant.

4.175.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1  = FRAC32 (0.5);

  // second input = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be 0x20000000
  f32Out = MLIB_Sub_F32(f32In1,f32In2);

  // output should be 0x20000000
  f32Out = MLIB_Sub (f32In1,f32In2,Define F32);
```

```
  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x20000000
  f32Out = MLIB_Sub (f32In1,f32In2);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \left(f32In1 - f32In2\right)$$

Equation **MLIB_Sub_Eq1**

Note

Overflow is not detected.

4.175.7 Re-entrancy

The function is re-entrant.

4.175.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1  = FRAC32 (0.5);

  // second input = 0.25
  f32In2  = FRAC32 (0.25);

  // output should be 0x20000000
  f32Out = MLIB_Sub_F32(f32In1,f32In2);

  // output should be 0x20000000
  f32Out = MLIB_Sub (f32In1,f32In2,Define F32);

  // ##############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ##############################################################

  // output should be 0x20000000
  f32Out = MLIB_Sub (f32In1,f32In2);
}
```

4.176 Function MLIB_Sub_F16

This function subtracts the second parameter from the first one.

4.176.1 Declaration

```
tFrac16 MLIB_Sub_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.176.2 Arguments

Table 4-213. MLIB_Sub_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In1	**input**	operand is a 16-bit number normalized between [-1,1)
register tFrac16	f16In2	**input**	operand is a 16-bit number normalized between [-1,1)

4.176.3 Return

The subtraction of the second argument from the first argument.The subtraction of the second argument from the first argument.

4.176.4 Description

Subtraction of two fractional 16-bit values. The second argument is subtracted from the first one. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = \left(f16In1 - f16In2\right)$$

Equation **MLIB_Sub_Eq1**

Subtraction of two fractional 16-bit values. The second argument is subtracted from the first one. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.176.5 Re-entrancy

The function is re-entrant.

4.176.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x2000
  f16Out = MLIB_Sub_F16(f16In1,f16In2);

  // output should be 0x2000
  f16Out = MLIB_Sub (f16In1,f16In2,Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x20000000
  f16Out = MLIB_Sub (f16In1,f16In2);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 - f16In2)$$

Equation **MLIB_Sub_Eq1**

Note

Overflow is not detected.

4.176.7 Re-entrancy

The function is re-entrant.

4.176.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x2000
  f16Out = MLIB_Sub_F16(f16In1,f16In2);

  // output should be 0x2000
  f16Out = MLIB_Sub (f16In1,f16In2,Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be 0x20000000
  f16Out = MLIB_Sub (f16In1,f16In2);
}
```

4.177 Function MLIB_Sub_FLT

This function subtracts the second parameter from the first one.

4.177.1 Declaration

```
tFloat MLIB_Sub_FLT(register tFloat fltIn1, register tFloat fltIn2);
```

4.177.2 Arguments

Table 4-214. MLIB_Sub_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	Operand is a single precision floating point number.
register tFloat	fltIn2	**input**	Operand is a single precision floating point number.
register tFloat	fltIn1	**input**	Operand is a single precision floating point number
register tFloat	fltIn2	**input**	Operand is a single precision floating point number

4.177.3 Return

The subtraction of the second argument from the first argument.The subtraction of the second argument from the first argument

4.177.4 Description

Subtraction of two single precision floating point values. The second argument is subtracted from the first one. The input value as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-2^{128}, 2^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 - fltIn2)$$

Equation **MLIB_Sub_Eq1**

Subtraction of two single precision floating point values. The second argument is subtracted from the first one. The input value as well as output value is considered as single precision floating point data type. The output saturation is not implemented in this function, thus in case the subtraction of input parameters is outside the (-2^{128}, 2^{128}) interval, the output value will overflow without any detection.

Note

Overflow is not detected.

4.177.5 Re-entrancy

The function is re-entrant.

4.177.6 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;

void main(void)
{
  // first input = 50.5
  fltIn1  = 50.5

  // second input = 25.25
  fltIn2  = 25.25;

  // output should be 25.25
  fltOut = MLIB_Sub_FLT(fltIn1,fltIn2);

  // output should be 25.25
  fltOut = MLIB_Sub (fltIn1,fltIn2,Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 25.25
  fltOut = MLIB_Sub (fltIn1,fltIn2);
}
```

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 - fltIn2)$$

Equation **MLIB_Sub_Eq1**

Note

Overflow is not detected.

4.177.7 Re-entrancy

The function is re-entrant.

4.177.8 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltOut;

void main(void)
{
  // first input = 50.5
  fltIn1  = 50.5

  // second input = 25.25
  fltIn2  = 25.25;

  // output should be 25.25
  fltOut = MLIB_Sub_FLT(fltIn1,fltIn2);

  // output should be 25.25
  fltOut = MLIB_Sub (fltIn1,fltIn2,Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 25.25
  fltOut = MLIB_Sub (fltIn1,fltIn2);
}
```

4.178 Function MLIB_SubSat_F32

This function subtracts the second parameter from the first one and saturate if necessary.

4.178.1 Declaration

```
tFrac32 MLIB_SubSat_F32(register tFrac32 f32In1, register tFrac32 f32In2);
```

4.178.2 Arguments

Table 4-215. MLIB_SubSat_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	Operand is a 32-bit number normalized between [-1,1).
register tFrac32	f32In2	**input**	Operand is a 32-bit number normalized between [-1,1).

4.178.3 Return

The subtraction of the second argument from the first argument.

4.178.4 Description

Subtraction with overflow control of two fractional 32-bit numbers. The second argument is subtracted from the first one. The input values as well as output value are considered as 32-bit fractional data type. The output saturation is implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f32Out = \begin{cases} FRAC32\_MIN & \text{if} \quad (f32In1 - f32In2) < FRAC32\_MIN \\ (f32In1 - f32In2) & \text{if} \quad FRAC32\_MIN \le (f32In1 - f32In2) \le FRAC32\_MAX \\ FRAC32\_MAX & \text{if} \quad (f32In1 - f32In2) > FRAC32\_MAX \end{cases}$$

Equation **MLIB_SubSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.178.5 Re-entrancy

The function is re-entrant.

4.178.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32Out;

void main(void)
{
  // first input = 0.5
  f32In1  = FRAC32 (0.5);

  // second input = 0.25
  f32In2  = FRAC32 (0.25);
```

```
    // output should be 0x20000000
    f32Out = MLIB_SubSat_F32(f32In1,f32In2);

    // output should be 0x20000000
    f32Out = MLIB_SubSat (f32In1,f32In2,Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be 0x20000000
    f32Out = MLIB_SubSat (f32In1,f32In2);
}
```

4.179 Function MLIB_SubSat_F16

This function subtracts the second parameter from the first one and saturate if necessary.

4.179.1 Declaration

```
tFrac16 MLIB_SubSat_F16(register tFrac16 f16In1, register tFrac16 f16In2);
```

4.179.2 Arguments

Table 4-216. MLIB_SubSat_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	Operand is a 16-bit number normalized between [-1,1).
register tFrac16	f16In2	**input**	Operand is a 16-bit number normalized between [-1,1).

4.179.3 Return

The subtraction of the second argument from the first argument.

4.179.4 Description

Subtraction with overflow control of two fractional 16-bit numbers. The second argument is subtracted from the first one. The input values as well as output value are considered as 16-bit fractional data type. The output saturation is implemented in this function, thus in case the subtraction of input parameters is outside the (-1, 1) interval, the output value is limited to the boundary value.

The output of the function is defined by the following simple equation:

$$f16Out = \begin{cases} FRAC16\_MIN & \text{if} \quad (f16In1 - f16In2) < FRAC16\_MIN \\ (f16In1 - f16In2) & \text{if} \quad FRAC16\_MIN \le (f16In1 - f16In2) \le FRAC16\_MAX \\ FRAC16\_MAX & \text{if} \quad (f16In1 - f16In2) > FRAC16\_MAX \end{cases}$$

Equation **MLIB_SubSat_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.179.5 Re-entrancy

The function is re-entrant.

4.179.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16Out;

void main(void)
{
  // first input = 0.5
  f16In1  = FRAC16 (0.5);

  // second input = 0.25
  f16In2  = FRAC16 (0.25);

  // output should be 0x2000
  f16Out = MLIB_SubSat_F16(f16In1,f16In2);

  // output should be 0x2000
  f16Out = MLIB_SubSat (f16In1,f16In2,Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
```

```
    // as default
    // ###############################################################

    // output should be 0x2000
    f16Out = MLIB_SubSat (f16In1,f16In2);
}
```

4.180 Function MLIB_VMac_F32

This function implements the vector multiply accumulate function.

4.180.1 Declaration

```
tFrac32 MLIB_VMac_F32(register tFrac32 f32In1, register tFrac32 f32In2, register tFrac32
f32In3, register tFrac32 f32In4);
```

4.180.2 Arguments

Table 4-217. MLIB_VMac_F32 arguments

Type	Name	Direction	Description
register tFrac32	f32In1	**input**	First input value to first multiplication.
register tFrac32	f32In2	**input**	Second input value to first multiplication.
register tFrac32	f32In3	**input**	First input value to second multiplication.
register tFrac32	f32In4	**input**	Second input value to second multiplication.
register tFrac32	f32In1	**input**	First input value to first multiplication
register tFrac32	f32In2	**input**	Second input value to first multiplication
register tFrac32	f32In3	**input**	First input value to second multiplication
register tFrac32	f32In4	**input**	Second input value to second multiplication

4.180.3 Return

Vector multiplied input values with addition.Vector multiplied input values with addition

4.180.4 Description

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 \bullet f32In2) + (f32In3 \bullet f32In4)$$

Equation **MLIB_VMac_Eq1**

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as 32-bit fractional values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.180.5 Re-entrancy

The function is re-entrant.

4.180.6 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32In4;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f32In2  = FRAC32 (0.15);

  // input3 value = 0.35
```

```
  f32In3  = FRAC32 (0.35);

  // input4 value = 0.45
  f32In4  = FRAC32 (0.45);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac_F32(f32In1, f32In2, f32In3, f32In4);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac (f32In1, f32In2, f32In3, f32In4, Define F32);

  // ###############################################################
  // Available only if 32-bit fractional implementation selected
  // as default
  // ###############################################################

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac (f32In1, f32In2, f32In3, f32In4);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = (f32In1 \bullet f32In2) + (f32In3 \bullet f32In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.180.7 Re-entrancy

The function is re-entrant.

4.180.8 Code Example

```
#include "mlib.h"

tFrac32 f32In1;
tFrac32 f32In2;
tFrac32 f32In3;
tFrac32 f32In4;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f32In1  = FRAC32 (0.25);

  // input2 value = 0.15
  f32In2  = FRAC32 (0.15);

  // input3 value = 0.35
  f32In3  = FRAC32 (0.35);
```

```
    // input4 value = 0.45
    f32In4  = FRAC32 (0.45);

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac_F32(f32In1, f32In2, f32In3, f32In4);

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac (f32In1, f32In2, f32In3, f32In4, Define F32);

    // ###############################################################
    // Available only if 32-bit fractional implementation selected
    // as default
    // ###############################################################

    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac (f32In1, f32In2, f32In3, f32In4);
}
```

4.181 Function MLIB_VMac_F32F16F16

This function implements the vector multiply accumulate function.

4.181.1 Declaration

```
tFrac32 MLIB_VMac_F32F16F16(register tFrac16 f16In1, register tFrac16 f16In2, register
tFrac16 f16In3, register tFrac16 f16In4);
```

4.181.2 Arguments

Table 4-218. MLIB_VMac_F32F16F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First input value to first multiplication.
register tFrac16	f16In2	**input**	Second input value to first multiplication.
register tFrac16	f16In3	**input**	First input value to second multiplication.
register tFrac16	f16In4	**input**	Second input value to second multiplication.
register tFrac16	f16In1	**input**	First input value to first multiplication
register tFrac16	f16In2	**input**	Second input value to first multiplication
register tFrac16	f16In3	**input**	First input value to second multiplication
register tFrac16	f16In4	**input**	Second input value to second multiplication

4.181.3 Return

Vector multiplied input values with addition.Vector multiplied input values with addition

4.181.4 Description

This inline function returns the vector multiply accumulate of input values. The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f32Out = \left(f16In1 \bullet f16In2\right) + \left(f16In3 \bullet f16In4\right)$$

Equation **MLIB_VMac_Eq1**

This inline function returns the vector multiply accumulate of input values. The input values are considered as 16-bit fractional values and the output value is considered as 32-bit fractional value. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.181.5 Re-entrancy

The function is re-entrant.

4.181.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);
```

```
  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // input4 value = 0.45
  f16In4  = FRAC16 (0.45);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac_F32F16F16(f16In1, f16In2, f16In3, f16In4);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4, F32F16F16);
}
```

The output of the function is defined by the following simple equation:

$$f32Out = \left(f16In1 \bullet f16In2\right) + \left(f16In3 \bullet f16In4\right)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.181.7 Re-entrancy

The function is re-entrant.

4.181.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac32 f32Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // input4 value = 0.45
  f16In4  = FRAC16 (0.45);

  // output should be FRAC32(0.195) = 0x18F5C28F
  f32Out = MLIB_VMac_F32F16F16(f16In1, f16In2, f16In3, f16In4);
```

```
    // output should be FRAC32(0.195) = 0x18F5C28F
    f32Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4, F32F16F16);
}
```

4.182 Function MLIB_VMac_F16

This function implements the vector multiply accumulate function.

4.182.1 Declaration

```
tFrac16 MLIB_VMac_F16(register tFrac16 f16In1, register tFrac16 f16In2, register tFrac16
f16In3, register tFrac16 f16In4);
```

4.182.2 Arguments

Table 4-219. MLIB_VMac_F16 arguments

Type	Name	Direction	Description
register tFrac16	f16In1	**input**	First input value to first multiplication.
register tFrac16	f16In2	**input**	Second input value to first multiplication.
register tFrac16	f16In3	**input**	First input value to second multiplication.
register tFrac16	f16In4	**input**	Second input value to second multiplication.
register tFrac16	f16In1	**input**	First input value to first multiplication
register tFrac16	f16In2	**input**	Second input value to first multiplication
register tFrac16	f16In3	**input**	First input value to second multiplication
register tFrac16	f16In4	**input**	Second input value to second multiplication

4.182.3 Return

Vector multiplied input values with addition.Vector multiplied input values with addition

4.182.4 Description

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 \bullet f16In2) + (f16In3 \bullet f16In4)$$

Equation **MLIB_VMac_Eq1**

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as 16-bit fractional values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-1, 1) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.182.5 Re-entrancy

The function is re-entrant.

4.182.6 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // input4 value = 0.45
  f16In4  = FRAC16 (0.45);

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac_F16(f16In1, f16In2, f16In3, f16In4);

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4, Define F16);

  // ###############################################################
  // Available only if 16-bit fractional implementation selected
```

```
  // as default
  // ##############################################################

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4);
}
```

The output of the function is defined by the following simple equation:

$$f16Out = (f16In1 \bullet f16In2) + (f16In3 \bullet f16In4)$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.182.7 Re-entrancy

The function is re-entrant.

4.182.8 Code Example

```
#include "mlib.h"

tFrac16 f16In1;
tFrac16 f16In2;
tFrac16 f16In3;
tFrac16 f16In4;
tFrac16 f16Out;

void main(void)
{
  // input1 value = 0.25
  f16In1  = FRAC16 (0.25);

  // input2 value = 0.15
  f16In2  = FRAC16 (0.15);

  // input3 value = 0.35
  f16In3  = FRAC16 (0.35);

  // input4 value = 0.45
  f16In4  = FRAC16 (0.45);

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac_F16(f16In1, f16In2, f16In3, f16In4);

  // output should be FRAC16(0.195) = 0x18F5
  f16Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4, Define F16);

  // ##############################################################
  // Available only if 16-bit fractional implementation selected
  // as default
  // ##############################################################
```

```
    // output should be FRAC16(0.195) = 0x18F5
    f16Out = MLIB_VMac (f16In1, f16In2, f16In3, f16In4);
}
```

4.183 Function MLIB_VMac_FLT

This function implements the vector multiply accumulate function.

4.183.1 Declaration

```
tFloat MLIB_VMac_FLT(register tFloat fltIn1, register tFloat fltIn2, register tFloat fltIn3,
register tFloat fltIn4);
```

4.183.2 Arguments

Table 4-220. MLIB_VMac_FLT arguments

Type	Name	Direction	Description
register tFloat	fltIn1	**input**	First input value to first multiplication.
register tFloat	fltIn2	**input**	Second input value to first multiplication.
register tFloat	fltIn3	**input**	First input value to second multiplication.
register tFloat	fltIn4	**input**	Second input value to second multiplication.
register tFloat	fltIn1	**input**	First input value to first multiplication
register tFloat	fltIn2	**input**	Second input value to first multiplication
register tFloat	fltIn3	**input**	First input value to second multiplication
register tFloat	fltIn4	**input**	Second input value to second multiplication

4.183.3 Return

Vector multiplied input values with addition.Vector multiplied input values with addition

4.183.4 Description

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as single precision floating point values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-2^{128}, 2^{128}) interval, the output value will overflow without any detection.

The output of the function is defined by the following simple equation:

$$fltOut = (fltIn1 \bullet fltIn2) + (fltIn3 \bullet flt6In4)$$

Equation **MLIB_VMac_Eq1**

This inline function returns the vector multiply accumulate of input values. The input values as well as output value is considered as single precision floating point values. The output saturation is not implemented in this function, thus in case the vector multiply-add of input values is outside the (-2^{128}, 2^{128}) interval, the output value will overflow without any detection.

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.183.5 Re-entrancy

The function is re-entrant.

4.183.6 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltIn4;
tFloat fltOut;

void main(void)
{
  // input1 value = 0.25
  fltIn1  = 0.25;

  // input2 value = 0.15
  fltIn2  = 0.15;

  // input3 value = 0.35
```

```
  fltIn3  = 0.35;

  // input4 value = 0.45
  fltIn4  = 0.45;

  // output should be 0.195
  fltOut = MLIB_VMac_FLT(fltIn1, fltIn2, fltIn3, fltIn4);

  // output should be 0.195
  fltOut = MLIB_VMac (fltIn1, fltIn2, fltIn3, fltIn4, Define FLT);

  // ###############################################################
  // Available only if single precision floating point
  // implementation selected as default
  // ###############################################################

  // output should be 0.195
  fltOut = MLIB_VMac (fltIn1, fltIn2, fltIn3, fltIn4);
}
```

The output of the function is defined by the following simple equation:

$$\text{fltOut} = (\text{fltIn1} \bullet \text{fltIn2}) + (\text{fltIn3} \bullet \text{flt6In4})$$

Equation **MLIB_VMac_Eq1**

Note

Due to effectivity reason this function is implemented as inline and thus is not ANSI-C compliant.

4.183.7 Re-entrancy

The function is re-entrant.

4.183.8 Code Example

```
#include "mlib.h"

tFloat fltIn1;
tFloat fltIn2;
tFloat fltIn3;
tFloat fltIn4;
tFloat fltOut;

void main(void)
{
  // input1 value = 0.25
  fltIn1  = 0.25;

  // input2 value = 0.15
  fltIn2  = 0.15;

  // input3 value = 0.35
  fltIn3  = 0.35;
```

```
    // input4 value = 0.45
    fltIn4  = 0.45;

    // output should be 0.195
    fltOut = MLIB_VMac_FLT(fltIn1, fltIn2, fltIn3, fltIn4);

    // output should be 0.195
    fltOut = MLIB_VMac (fltIn1, fltIn2, fltIn3, fltIn4, Define FLT);

    // ###############################################################
    // Available only if single precision floating point
    // implementation selected as default
    // ###############################################################

    // output should be 0.195
    fltOut = MLIB_VMac (fltIn1, fltIn2, fltIn3, fltIn4);
}
```

4.184 Function MCLIB_GetVersion

This function returns the information about MCLIB version.

4.184.1 Declaration

```
const MCLIB_VERSION_T * MCLIB_GetVersion();
```

4.184.2 Return

The function returns the information about the version of Motor Control Library Set.

4.184.3 Description

The function returns the information about the version of Motor Control Library Set. The information are structured as follows:

- Motor Control Library Set identification code
- Motor Control Library Set version code
- Motor Control Library Set supported implementation code
- Motor Control Library Set default implementation code

4.184.4 Reentrancy

The function is reentrant.

Chapter 5

5.1 Typedefs Index

Table 5-1. Quick typedefs reference

Type	Name	Description
typedef unsigned char	tBool	basic boolean type
typedef float	tFloat	single precision float type
typedef tS16	tFrac16	16-bit signed fractional Q1.15 type
typedef tS32	tFrac32	32-bit Q1.31 type
typedef signed short	tS16	signed 16-bit integer type
typedef signed long	tS32	signed 32-bit integer type
typedef signed long long	tS64	signed 64-bit integer type
typedef unsigned short	tU16	unsigned 16-bit integer type
typedef unsigned long	tU32	unsigned 32-bit integer type

Chapter 6
Compound Data Types

Table 6-1. Compound data types overview

Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_F16	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_COEFF_T_F32	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_COEFF_T_FLT	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR1_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR1_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR1_T_FLT	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_F16	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_F32	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_COEFF_T_FLT	Sub-structure containing filter coefficients.
GDFLIB_FILTER_IIR2_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_IIR2_T_FLT	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_F16	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_F32	Structure containing filter buffer and coefficients.
GDFLIB_FILTER_MA_T_FLT	Structure containing filter buffer and coefficients.
GDFLIB_FILTERFIR_PARAM_T_F16	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_PARAM_T_F32	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_PARAM_T_FLT	Structure containing parameters of the filter.
GDFLIB_FILTERFIR_STATE_T_F16	Structure containing the current state of the filter.
GDFLIB_FILTERFIR_STATE_T_F32	Structure containing the current state of the filter.
GDFLIB_FILTERFIR_STATE_T_FLT	Structure containing the current state of the filter.
GFLIB_ACOS_T_F16	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_T_F32	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_T_FLT	Default approximation coefficients datatype for arccosine approximation.
GFLIB_ACOS_TAYLOR_COEF_T_F16	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ACOS_TAYLOR_COEF_T_F32	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ASIN_T_F16	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_T_F32	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_T_FLT	Default approximation coefficients datatype for arcsine approximation.
GFLIB_ASIN_TAYLOR_COEF_T_F16	Array of approximation coefficients for piece-wise polynomial.

Table continues on the next page...

Table 6-1. Compound data types overview (continued)

Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F32	Array of approximation coefficients for piece-wise polynomial.
GFLIB_ATAN_T_F16	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.
GFLIB_ATAN_T_F32	Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.
GFLIB_ATAN_T_FLT	Structure containing the approximation coefficients.
GFLIB_ATAN_TAYLOR_COEF_T_F16	Array of polynomial approximation coefficients for one sub-interval.
GFLIB_ATAN_TAYLOR_COEF_T_F32	Array of minimax polynomial approximation coefficients for one sub-interval.
GFLIB_ATANYXSHIFTED_T_F16	Structure containing the parameter for the AtanYXShifted function.
GFLIB_ATANYXSHIFTED_T_F32	Structure containing the parameter for the AtanYXShifted function.
GFLIB_ATANYXSHIFTED_T_FLT	Structure containing the parameter for the AtanYXShifted function.
GFLIB_CONTROLLER_PI_P_T_F16	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_P_T_F32	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_P_T_FLT	Structure containing parameters and states of the parallel form PI controller.
GFLIB_CONTROLLER_PI_R_T_F16	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PI_R_T_F32	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PI_R_T_FLT	Structure containing parameters and states of the recurrent form PI controller.
GFLIB_CONTROLLER_PIAW_P_T_F16	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_P_T_F32	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_P_T_FLT	Structure containing parameters and states of the parallel form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_F16	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_F32	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_CONTROLLER_PIAW_R_T_FLT	Structure containing parameters and states of the recurrent form PI controller with anti-windup.
GFLIB_COS_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_COS_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_COS_T_FLT	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.
GFLIB_HYST_T_F16	Structure containing parameters and states for the hysteresis function.
GFLIB_HYST_T_F32	Structure containing parameters and states for the hysteresis function.
GFLIB_HYST_T_FLT	Structure containing parameters and states for the hysteresis function.
GFLIB_INTEGRATOR_TR_T_F16	Structure containing integrator parameters and coefficients.
GFLIB_INTEGRATOR_TR_T_F32	Structure containing integrator parameters and coefficients.
GFLIB_INTEGRATOR_TR_T_FLT	Structure containing integrator parameters and coefficients.

Table continues on the next page...

Table 6-1. Compound data types overview (continued)

Name	Description
GFLIB_LIMIT_T_F16	Structure containing the limits.
GFLIB_LIMIT_T_F32	Structure containing the limits.
GFLIB_LIMIT_T_FLT	Structure containing the limits.
GFLIB_LOWERLIMIT_T_F16	Structure containing the lower limit.
GFLIB_LOWERLIMIT_T_F32	Structure containing the lower limit.
GFLIB_LOWERLIMIT_T_FLT	Structure containing the lower limit.
GFLIB_LUT1D_T_F16	Structure containing 1D look-up table parameters.
GFLIB_LUT1D_T_F32	Structure containing 1D look-up table parameters.
GFLIB_LUT1D_T_FLT	Structure containing 1D look-up table parameters.
GFLIB_LUT2D_T_F16	Structure containing 2D look-up table parameters.
GFLIB_LUT2D_T_F32	Structure containing 2D look-up table parameters.
GFLIB_LUT2D_T_FLT	Structure containing 2D look-up table parameters.
GFLIB_RAMP_T_F16	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_RAMP_T_F32	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_RAMP_T_FLT	Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.
GFLIB_SIN_T_F16	Array of four 16-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SIN_T_F32	Array of five 32-bit elements for storing coefficients of the Taylor polynomial.
GFLIB_SIN_T_FLT	Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.
GFLIB_TAN_T_F16	Output of tan(Π * f16In) for interval [0, Π/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.
GFLIB_TAN_T_F32	Output of tan(Π * f32In) for interval [0, Π/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.
GFLIB_TAN_T_FLT	Polynomial coefficient for fractional approximation in single precision floating point format.
GFLIB_TAN_TAYLOR_COEF_T_F16	Structure containing four polynomial coefficients for one sub-interval.
GFLIB_TAN_TAYLOR_COEF_T_F32	Structure containing four polynomial coefficients for one sub-interval.
GFLIB_UPPERLIMIT_T_F16	Structure containing the upper limit.
GFLIB_UPPERLIMIT_T_F32	Structure containing the upper limit.
GFLIB_UPPERLIMIT_T_FLT	Structure containing the upper limit.
GFLIB_VECTORLIMIT_T_F16	Structure containing the limit.
GFLIB_VECTORLIMIT_T_F32	Structure containing the limit.
GFLIB_VECTORLIMIT_T_FLT	Structure containing the limit.

Table continues on the next page...

Table 6-1. Compound data types overview (continued)

Name	Description
GMCLIB_DECOUPLINGPMSM_T_F16	Structure containing coefficients for calculation of the decoupling.
GMCLIB_DECOUPLINGPMSM_T_F32	Structure containing coefficients for calculation of the decoupling.
GMCLIB_DECOUPLINGPMSM_T_FLT	Structure containing coefficients for calculation of the decoupling.
GMCLIB_ELIMDCBUSRIP_T_F16	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
GMCLIB_ELIMDCBUSRIP_T_F32	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
GMCLIB_ELIMDCBUSRIP_T_FLT	Structure containing the PWM modulation index and the measured value of the DC bus voltage.
MCLIB_VERSION_T	Motor Control Library Set identification structure.
SWLIBS_2Syst_F16	Array of two standard 16-bit fractional arguments.
SWLIBS_2Syst_F32	Array of two standard 32-bit fractional arguments.
SWLIBS_2Syst_FLT	Array of two standard single precision floating point arguments.
SWLIBS_3Syst_F16	Array of three standard 16-bit fractional arguments.
SWLIBS_3Syst_F32	Array of three standard 32-bit fractional arguments.
SWLIBS_3Syst_FLT	Array of three standard single precision floating point arguments.

6.1 GDFLIB_FILTER_IIR1_COEFF_T_F16

```
#include <GDFLIB_FilterIIR1.h>
```

6.1.1 Description

Sub-structure containing filter coefficients.

6.1.2 Compound Type Members

Table 6-2. GDFLIB_FILTER_IIR1_COEFF_T_F16 members description

Type	Name	Description
tFrac16	f16B0	
tFrac16	f16B1	
tFrac16	f16A1	

6.2 GDFLIB_FILTER_IIR1_COEFF_T_F32

```
#include <GDFLIB_FilterIIR1.h>
```

6.2.1 Description

Sub-structure containing filter coefficients.

6.2.2 Compound Type Members

Table 6-3. GDFLIB_FILTER_IIR1_COEFF_T_F32 members description

Type	Name	Description
tFrac32	f32B0	
tFrac32	f32B1	
tFrac32	f32A1	

6.3 GDFLIB_FILTER_IIR1_COEFF_T_FLT

```
#include <GDFLIB_FilterIIR1.h>
```

6.3.1 Description

Sub-structure containing filter coefficients.

6.3.2 Compound Type Members

Table 6-4. GDFLIB_FILTER_IIR1_COEFF_T_FLT members description

Type	Name	Description
tFloat	fltB0	
tFloat	fltB1	
tFloat	fltA1	

6.4 GDFLIB_FILTER_IIR1_T_F16

```
#include <GDFLIB_FilterIIR1.h>
```

6.4.1 Description

Structure containing filter buffer and coefficients.

6.4.2 Compound Type Members

Table 6-5. GDFLIB_FILTER_IIR1_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_F16	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac16	f16FiltBufferX	Input buffer of an IIR1 filter, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16FiltBufferY	

6.5 GDFLIB_FILTER_IIR1_T_F32

```
#include <GDFLIB_FilterIIR1.h>
```

6.5.1 Description

Structure containing filter buffer and coefficients.

6.5.2 Compound Type Members

Table 6-6. GDFLIB_FILTER_IIR1_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_F32	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac32	f32FiltBufferX	Input buffer of an IIR1 filter, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32FiltBufferY	

6.6 GDFLIB_FILTER_IIR1_T_FLT

```
#include <GDFLIB_FilterIIR1.h>
```

6.6.1 Description

Structure containing filter buffer and coefficients.

6.6.2 Compound Type Members

Table 6-7. GDFLIB_FILTER_IIR1_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_IIR1_COEFF_T_FLT	trFiltCoeff	Sub-structure containing filter coefficients.
tFloat	fltFiltBufferX	Input buffer of an IIR1 filter. The input values are in full range single precision floating point format.
tFloat	fltFiltBufferY	

6.7 GDFLIB_FILTER_IIR2_COEFF_T_F16

```
#include <GDFLIB_FilterIIR2.h>
```

6.7.1 Description

Sub-structure containing filter coefficients.

6.7.2 Compound Type Members

Table 6-8. GDFLIB_FILTER_IIR2_COEFF_T_F16 members description

Type	Name	Description
tFrac16	f16B0	

Table continues on the next page...

Table 6-8. GDFLIB_FILTER_IIR2_COEFF_T_F16 members description (continued)

Type	Name	Description
tFrac16	f16B1	
tFrac16	f16B2	
tFrac16	f16A1	
tFrac16	f16A2	

6.8 GDFLIB_FILTER_IIR2_COEFF_T_F32

```
#include <GDFLIB_FilterIIR2.h>
```

6.8.1 Description

Sub-structure containing filter coefficients.

6.8.2 Compound Type Members

Table 6-9. GDFLIB_FILTER_IIR2_COEFF_T_F32 members description

Type	Name	Description
tFrac32	f32B0	
tFrac32	f32B1	
tFrac32	f32B2	
tFrac32	f32A1	
tFrac32	f32A2	

6.9 GDFLIB_FILTER_IIR2_COEFF_T_FLT

```
#include <GDFLIB_FilterIIR2.h>
```

6.9.1 Description

Sub-structure containing filter coefficients.

6.9.2 Compound Type Members

Table 6-10. GDFLIB_FILTER_IIR2_COEFF_T_FLT members description

Type	Name	Description
tFloat	fltB0	
tFloat	fltB1	
tFloat	fltB2	
tFloat	fltA1	
tFloat	fltA2	

6.10 GDFLIB_FILTER_IIR2_T_F16

```
#include <GDFLIB_FilterIIR2.h>
```

6.10.1 Description

Structure containing filter buffer and coefficients.

6.10.2 Compound Type Members

Table 6-11. GDFLIB_FILTER_IIR2_T_F16 members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_F16	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac16	f16FiltBufferX	Input buffer of an IIR2 filter, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16FiltBufferY	

6.11 GDFLIB_FILTER_IIR2_T_F32

```
#include <GDFLIB_FilterIIR2.h>
```

6.11.1 Description

Structure containing filter buffer and coefficients.

6.11.2 Compound Type Members

Table 6-12. GDFLIB_FILTER_IIR2_T_F32 members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_F32	trFiltCoeff	Sub-structure containing filter coefficients.
tFrac32	f32FiltBufferX	Input buffer of an IIR2 filter, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32FiltBufferY	

6.12 GDFLIB_FILTER_IIR2_T_FLT

```
#include <GDFLIB_FilterIIR2.h>
```

6.12.1 Description

Structure containing filter buffer and coefficients.

6.12.2 Compound Type Members

Table 6-13. GDFLIB_FILTER_IIR2_T_FLT members description

Type	Name	Description
GDFLIB_FILTER_IIR2_COEFF_T_FLT	trFiltCoeff	Sub-structure containing filter coefficients.
tFloat	fltFiltBufferX	Input buffer of an IIR2 filter. The input values are in full range single precision floating point format.
tFloat	fltFiltBufferY	

6.13 GDFLIB_FILTER_MA_T_F16

```
#include <GDFLIB_FilterMA.h>
```

6.13.1 Description

Structure containing filter buffer and coefficients.

6.13.2 Compound Type Members

Table 6-14. GDFLIB_FILTER_MA_T_F16 members description

Type	Name	Description
tFrac32	f32Acc	Filter accumulator.
tU16	u16NSamples	Number of samples for averaging, filter sample window [0,15].

6.14 GDFLIB_FILTER_MA_T_F32

```
#include <GDFLIB_FilterMA.h>
```

6.14.1 Description

Structure containing filter buffer and coefficients.

6.14.2 Compound Type Members

Table 6-15. GDFLIB_FILTER_MA_T_F32 members description

Type	Name	Description
tFrac32	f32Acc	Filter accumulator.
tU16	u16NSamples	Number of samples for averaging, filter sample window [0,31].

6.15 GDFLIB_FILTER_MA_T_FLT

```
#include <GDFLIB_FilterMA.h>
```

6.15.1 Description

Structure containing filter buffer and coefficients.

6.15.2 Compound Type Members

Table 6-16. GDFLIB_FILTER_MA_T_FLT members description

Type	Name	Description
tFloat	fltAcc	Filter accumulator.
tU16	u16NSamples	Number of samples for averaging, filter sample window [0,31].

6.16 GDFLIB_FILTERFIR_PARAM_T_F16

```
#include <GDFLIB_FilterFIR.h>
```

6.16.1 Description

Structure containing parameters of the filter.

6.16.2 Compound Type Members

Table 6-17. GDFLIB_FILTERFIR_PARAM_T_F16 members description

Type	Name	Description
tU16	u16Order	FIR filter order, must be 1 or more.
const tFrac16 *	pCoefBuf	FIR filter coefficients buffer.

6.17 GDFLIB_FILTERFIR_PARAM_T_F32

```
#include <GDFLIB_FilterFIR.h>
```

6.17.1 Description

Structure containing parameters of the filter.

6.17.2 Compound Type Members

Table 6-18. GDFLIB_FILTERFIR_PARAM_T_F32 members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be 1 or more.
const tFrac32 *	pCoefBuf	FIR filter coefficients buffer.

6.18 GDFLIB_FILTERFIR_PARAM_T_FLT

```
#include <GDFLIB_FilterFIR.h>
```

6.18.1 Description

Structure containing parameters of the filter.

6.18.2 Compound Type Members

Table 6-19. GDFLIB_FILTERFIR_PARAM_T_FLT members description

Type	Name	Description
tU32	u32Order	FIR filter order, must be 1 or more.
const tFloat *	pCoefBuf	FIR filter coefficients buffer.

6.19 GDFLIB_FILTERFIR_STATE_T_F16

```
#include <GDFLIB_FilterFIR.h>
```

6.19.1 Description

Structure containing the current state of the filter.

6.19.2 Compound Type Members

Table 6-20. GDFLIB_FILTERFIR_STATE_T_F16 members description

Type	Name	Description
tU16	u16Idx	Input buffer index.
tFrac16 *	pInBuf	Pointer to the input buffer.

6.20 GDFLIB_FILTERFIR_STATE_T_F32

```
#include <GDFLIB_FilterFIR.h>
```

6.20.1 Description

Structure containing the current state of the filter.

6.20.2 Compound Type Members

Table 6-21. GDFLIB_FILTERFIR_STATE_T_F32 members description

Type	Name	Description
tU32	u32Idx	Input buffer index.
tFrac32 *	pInBuf	Pointer to the input buffer.

6.21 GDFLIB_FILTERFIR_STATE_T_FLT

```
#include <GDFLIB_FilterFIR.h>
```

6.21.1 Description

Structure containing the current state of the filter.

6.21.2 Compound Type Members

Table 6-22. GDFLIB_FILTERFIR_STATE_T_FLT members description

Type	Name	Description
tU32	u32Idx	Input buffer index.
tFloat *	pInBuf	Pointer to the input buffer.

6.22 GFLIB_ACOS_T_F16

```
#include <GFLIB_Acos.h>
```

6.22.1 Description

Default approximation coefficients datatype for arccosine approximation.

6.22.2 Compound Type Members

Table 6-23. GFLIB_ACOS_T_F16 members description

Type	Name	Description
GFLIB_ACOS_TAYLOR_COEF_T_F16	GFLIB_ACOS_SECTOR	Array of two elements for storing two sub-arrays (each sub-array contains five 16-bit coefficients) for all sub-intervals.

6.23 GFLIB_ACOS_T_F32

```
#include <GFLIB_Acos.h>
```

6.23.1 Description

Default approximation coefficients datatype for arccosine approximation.

6.23.2 Compound Type Members

Table 6-24. GFLIB_ACOS_T_F32 members description

Type	Name	Description
GFLIB_ACOS_TAYLOR_COEF_T_F32	GFLIB_ACOS_SECTOR	Array of two elements for storing two sub-arrays (each sub-array contains five 32-bit coefficients) for all sub-intervals.

6.24 GFLIB_ACOS_T_FLT

```
#include <GFLIB_Acos.h>
```

6.24.1 Description

Default approximation coefficients datatype for arccosine approximation.

6.24.2 Compound Type Members

Table 6-25. GFLIB_ACOS_T_FLT members description

Type	Name	Description
const tFloat	fltA	Array of approximation coefficients.

6.25 GFLIB_ACOS_TAYLOR_COEF_T_F16

```
#include <GFLIB_Acos.h>
```

6.25.1 Description

Array of approximation coefficients for piece-wise polynomial.

6.25.2 Compound Type Members

Table 6-26. GFLIB_ACOS_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	Array of five 16-bit elements for storing coefficients of the piece-wise polynomial.

6.26 GFLIB_ACOS_TAYLOR_COEF_T_F32

```
#include <GFLIB_Acos.h>
```

6.26.1 Description

Array of approximation coefficients for piece-wise polynomial.

6.26.2 Compound Type Members

Table 6-27. GFLIB_ACOS_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the piece-wise polynomial.

6.27 GFLIB_ASIN_T_F16

```
#include <GFLIB_Asin.h>
```

6.27.1 Description

Default approximation coefficients datatype for arcsine approximation.

6.27.2 Compound Type Members

Table 6-28. GFLIB_ASIN_T_F16 members description

Type	Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F16	GFLIB_ASIN_SECTOR	

6.28 GFLIB_ASIN_T_F32

```
#include <GFLIB_Asin.h>
```

6.28.1 Description

Default approximation coefficients datatype for arcsine approximation.

6.28.2 Compound Type Members

Table 6-29. GFLIB_ASIN_T_F32 members description

Type	Name	Description
GFLIB_ASIN_TAYLOR_COEF_T_F32	GFLIB_ASIN_SECTOR	

6.29 GFLIB_ASIN_T_FLT

```
#include <GFLIB_Asin.h>
```

6.29.1 Description

Default approximation coefficients datatype for arcsine approximation.

6.29.2 Compound Type Members

Table 6-30. GFLIB_ASIN_T_FLT members description

Type	Name	Description
const tFloat	fltA	

6.30 GFLIB_ASIN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Asin.h>
```

6.30.1 Description

Array of approximation coefficients for piece-wise polynomial.

6.30.2 Compound Type Members

Table 6-31. GFLIB_ASIN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	

6.31 GFLIB_ASIN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Asin.h>
```

6.31.1 Description

Array of approximation coefficients for piece-wise polynomial.

6.31.2 Compound Type Members

Table 6-32. GFLIB_ASIN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	Array of five 32-bit elements for storing coefficients of the piece-wise polynomial.

6.32 GFLIB_ATAN_T_F16

```
#include <GFLIB_Atan.h>
```

6.32.1 Description

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

6.32.2 Compound Type Members

Table 6-33. GFLIB_ATAN_T_F16 members description

Type	Name	Description
const GFLIB_ATAN_TAYLOR_COEF_T_F16	GFLIB_ATAN_SECTOR	

6.33 GFLIB_ATAN_T_F32

```
#include <GFLIB_Atan.h>
```

6.33.1 Description

Structure containing eight sub-structures with polynomial coefficients to cover all sub-intervals.

6.33.2 Compound Type Members

Table 6-34. GFLIB_ATAN_T_F32 members description

Type	Name	Description
const GFLIB_ATAN_TAYLOR_COEF_T_F32	GFLIB_ATAN_SECTOR	

6.34 GFLIB_ATAN_T_FLT

```
#include <GFLIB_Atan.h>
```

6.34.1 Description

Structure containing the approximation coefficients.

6.34.2 Compound Type Members

Table 6-35. GFLIB_ATAN_T_FLT members description

Type	Name	Description
const tFloat	fltA	

6.35 GFLIB_ATAN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Atan.h>
```

6.35.1 Description

Array of polynomial approximation coefficients for one sub-interval.

6.35.2 Compound Type Members

Table 6-36. GFLIB_ATAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	

6.36 GFLIB_ATAN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Atan.h>
```

6.36.1 Description

Array of minimax polynomial approximation coefficients for one sub-interval.

6.36.2 Compound Type Members

Table 6-37. GFLIB_ATAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	

6.37 GFLIB_ATANYXSHIFTED_T_F16

```
#include <GFLIB_AtanYXShifted.h>
```

6.37.1 Description

Structure containing the parameter for the AtanYXShifted function.

6.37.2 Compound Type Members

Table 6-38. GFLIB_ATANYXSHIFTED_T_F16 members description

Type	Name	Description
tFrac16	f16Ky	Multiplication coefficient for the y-signal.
tFrac16	f16Kx	Multiplication coefficient for the x-signal.
tS16	s16Ny	Scaling coefficient for the y-signal.
tS16	s16Nx	Scaling coefficient for the x-signal.
tFrac16	f16ThetaAdj	Adjusting angle.

6.38 GFLIB_ATANYXSHIFTED_T_F32

```
#include <GFLIB_AtanYXShifted.h>
```

6.38.1 Description

Structure containing the parameter for the AtanYXShifted function.

6.38.2 Compound Type Members

Table 6-39. GFLIB_ATANYXSHIFTED_T_F32 members description

Type	Name	Description
tFrac32	f32Ky	Multiplication coefficient for the y-signal.
tFrac32	f32Kx	Multiplication coefficient for the x-signal.
tS32	s32Ny	Scaling coefficient for the y-signal.
tS32	s32Nx	Scaling coefficient for the x-signal.
tFrac32	f32ThetaAdj	Adjusting angle.

6.39 GFLIB_ATANYXSHIFTED_T_FLT

```
#include <GFLIB_AtanYXShifted.h>
```

6.39.1 Description

Structure containing the parameter for the AtanYXShifted function.

6.39.2 Compound Type Members

Table 6-40. GFLIB_ATANYXSHIFTED_T_FLT members description

Type	Name	Description
tFloat	fltKy	Multiplication coefficient for the y-signal.
tFloat	fltKx	Multiplication coefficient for the x-signal.
tFloat	fltThetaAdj	Adjusting angle.

6.40 GFLIB_CONTROLLER_PI_P_T_F16

```
#include <GFLIB_ControllerPIp.h>
```

6.40.1 Description

Structure containing parameters and states of the parallel form PI controller.

6.40.2 Compound Type Members

Table 6-41. GFLIB_CONTROLLER_PI_P_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-15, 15].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-15, 15].
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16InK_1	State variable input error at step k-1.

6.41 GFLIB_CONTROLLER_PI_P_T_F32

```
#include <GFLIB_ControllerPIp.h>
```

6.41.1 Description

Structure containing parameters and states of the parallel form PI controller.

6.41.2 Compound Type Members

Table 6-42. GFLIB_CONTROLLER_PI_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-31, 31].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-31, 31].
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.

6.42 GFLIB_CONTROLLER_PI_P_T_FLT

```
#include <GFLIB_ControllerPIp.h>
```

6.42.1 Description

Structure containing parameters and states of the parallel form PI controller.

6.42.2 Compound Type Members

Table 6-43. GFLIB_CONTROLLER_PI_P_T_FLT members description

Type	Name	Description
tFloat	fltPropGain	Proportional Gain, single precision floating point format.
tFloat	fltIntegGain	Integral Gain, single precision floating point format.
tFloat	fltIntegPartK_1	State variable integral part at step k-1, single precision floating point format.
tFloat	fltInK_1	State variable input error at step k-1, single precision floating point format.

6.43 GFLIB_CONTROLLER_PI_R_T_F16

```
#include <GFLIB_ControllerPIr.h>
```

6.43.1 Description

Structure containing parameters and states of the recurrent form PI controller.

6.43.2 Compound Type Members

Table 6-44. GFLIB_CONTROLLER_PI_R_T_F16 members description

Type	Name	Description
tFrac16	f16CC1sc	CC1 coefficient, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16CC2sc	CC2 coefficient, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac32	f32Acc	Internal controller accumulator.
tFrac16	f16InErrK1	Controller input from the previous calculation step.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 15].

6.44 GFLIB_CONTROLLER_PI_R_T_F32

```
#include <GFLIB_ControllerPIr.h>
```

6.44.1 Description

Structure containing parameters and states of the recurrent form PI controller.

6.44.2 Compound Type Members

Table 6-45. GFLIB_CONTROLLER_PI_R_T_F32 members description

Type	Name	Description
tFrac32	f32CC1sc	CC1 coefficient, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32CC2sc	CC2 coefficient, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32Acc	Internal controller accumulator.
tFrac32	f32InErrK1	Controller input from the previous calculation step.
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 31].

6.45 GFLIB_CONTROLLER_PI_R_T_FLT

```
#include <GFLIB_ControllerPIr.h>
```

6.45.1 Description

Structure containing parameters and states of the recurrent form PI controller.

6.45.2 Compound Type Members

Table 6-46. GFLIB_CONTROLLER_PI_R_T_FLT members description

Type	Name	Description
tFloat	fltCC1sc	CC1 coefficient, single precision floating point format.
tFloat	fltCC2sc	CC2 coefficient, single precision floating point format.
tFloat	fltAcc	Internal controller accumulator, single precision floating point format.
tFloat	fltInErrK1	Controller input from the previous calculation step, single precision floating point format.

6.46 GFLIB_CONTROLLER_PIAW_P_T_F16

```
#include <GFLIB_ControllerPIpAW.h>
```

6.46.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

6.46.2 Compound Type Members

Table 6-47. GFLIB_CONTROLLER_PIAW_P_T_F16 members description

Type	Name	Description
tFrac16	f16PropGain	Proportional Gain, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16IntegGain	Integral Gain, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-15, 15].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-15, 15].
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).

Table continues on the next page...

Table 6-47. GFLIB_CONTROLLER_PIAW_P_T_F16 members description (continued)

Type	Name	Description
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac16	f16InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

6.47 GFLIB_CONTROLLER_PIAW_P_T_F32

```
#include <GFLIB_ControllerPIpAW.h>
```

6.47.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

6.47.2 Compound Type Members

Table 6-48. GFLIB_CONTROLLER_PIAW_P_T_F32 members description

Type	Name	Description
tFrac32	f32PropGain	Proportional Gain, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32IntegGain	Integral Gain, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tS16	s16PropGainShift	Proportional Gain Shift, integer format [-31, 31].
tS16	s16IntegGainShift	Integral Gain Shift, integer format [-31, 31].
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32IntegPartK_1	State variable integral part at step k-1.
tFrac32	f32InK_1	State variable input error at step k-1.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

6.48 GFLIB_CONTROLLER_PIAW_P_T_FLT

```
#include <GFLIB_ControllerPIpAW.h>
```

6.48.1 Description

Structure containing parameters and states of the parallel form PI controller with anti-windup.

6.48.2 Compound Type Members

Table 6-49. GFLIB_CONTROLLER_PIAW_P_T_FLT members description

Type	Name	Description
tFloat	fltPropGain	Proportional Gain, single precision floating point format.
tFloat	fltIntegGain	Integral Gain, single precision floating point format.
tFloat	fltLowerLimit	Lower Limit of the controller, single precision floating point format.
tFloat	fltUpperLimit	Upper Limit of the controller, single precision floating point format.
tFloat	fltIntegPartK_1	State variable integral part at step k-1, single precision floating point format.
tFloat	fltInK_1	State variable input error at step k-1, single precision floating point format.
tU16	u16LimitFlag	Limitation flag, if set to 1, the controller output has reached either the UpperLimit or LowerLimit.

6.49 GFLIB_CONTROLLER_PIAW_R_T_F16

```
#include <GFLIB_ControllerPIrAW.h>
```

6.49.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

6.49.2 Compound Type Members

Table 6-50. GFLIB_CONTROLLER_PIAW_R_T_F16 members description

Type	Name	Description
tFrac16	f16CC1sc	CC1 coefficient, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16CC2sc	CC2 coefficient, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac32	f32Acc	Internal controller accumulator.
tFrac16	f16InErrK1	Controller input from the previous calculation step.
tFrac16	f16UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tFrac16	f16LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (-2 ^{15}, 2 ^{15}-1).
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 15].

6.50 GFLIB_CONTROLLER_PIAW_R_T_F32

```
#include <GFLIB_ControllerPIrAW.h>
```

6.50.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

6.50.2 Compound Type Members

Table 6-51. GFLIB_CONTROLLER_PIAW_R_T_F32 members description

Type	Name	Description
tFrac32	f32CC1sc	CC1 coefficient, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32CC2sc	CC2 coefficient, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32Acc	Internal controller accumulator.
tFrac32	f32InErrK1	Controller input from the previous calculation step.
tFrac32	f32UpperLimit	Upper Limit of the controller, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tFrac32	f32LowerLimit	Lower Limit of the controller, fractional format normalized to fit into (-2 ^{31}, 2 ^{31}-1).
tU16	u16NShift	Scaling factor for the controller coefficients, integer format [0, 31].

6.51 GFLIB_CONTROLLER_PIAW_R_T_FLT

```
#include <GFLIB_ControllerPIrAW.h>
```

6.51.1 Description

Structure containing parameters and states of the recurrent form PI controller with anti-windup.

6.51.2 Compound Type Members

Table 6-52. GFLIB_CONTROLLER_PIAW_R_T_FLT members description

Type	Name	Description
tFloat	fltCC1sc	CC1 coefficient, single precision floating point format.
tFloat	fltCC2sc	CC2 coefficient, single precision floating point format.
tFloat	fltAcc	Internal controller accumulator, single precision floating point format.

Table continues on the next page...

Table 6-52. GFLIB_CONTROLLER_PIAW_R_T_FLT members description (continued)

Type	Name	Description
tFloat	fltInErrK1	Controller input from the previous calculation step, single precision floating point format.
tFloat	fltUpperLimit	Upper Limit of the controller, single precision floating point format.
tFloat	fltLowerLimit	Lower Limit of the controller, single precision floating point format.

6.52 GFLIB_COS_T_F16

```
#include <GFLIB_Cos.h>
```

6.52.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

6.52.2 Compound Type Members

Table 6-53. GFLIB_COS_T_F16 members description

Type	Name	Description
tFrac16	f16A	

6.53 GFLIB_COS_T_F32

```
#include <GFLIB_Cos.h>
```

6.53.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

6.53.2 Compound Type Members

Table 6-54. GFLIB_COS_T_F32 members description

Type	Name	Description
tFrac32	f32A	

6.54 GFLIB_COS_T_FLT

```
#include <GFLIB_Cos.h>
```

6.54.1 Description

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

6.54.2 Compound Type Members

Table 6-55. GFLIB_COS_T_FLT members description

Type	Name	Description
tFloat	fltA	

6.55 GFLIB_HYST_T_F16

```
#include <GFLIB_Hyst.h>
```

6.55.1 Description

Structure containing parameters and states for the hysteresis function.

6.55.2 Compound Type Members

Table 6-56. GFLIB_HYST_T_F16 members description

Type	Name	Description
tFrac16	f16HystOn	Value determining the upper threshold.
tFrac16	f16HystOff	Value determining the lower threshold.
tFrac16	f16OutValOn	
tFrac16	f16OutValOff	
tFrac16	f16OutState	Actual state of the output.

6.56 GFLIB_HYST_T_F32

```
#include <GFLIB_Hyst.h>
```

6.56.1 Description

Structure containing parameters and states for the hysteresis function.

6.56.2 Compound Type Members

Table 6-57. GFLIB_HYST_T_F32 members description

Type	Name	Description
tFrac32	f32HystOn	Value determining the upper threshold.
tFrac32	f32HystOff	Value determining the lower threshold.
tFrac32	f32OutValOn	
tFrac32	f32OutValOff	
tFrac32	f32OutState	Actual state of the output.

6.57 GFLIB_HYST_T_FLT

```
#include <GFLIB_Hyst.h>
```

6.57.1 Description

Structure containing parameters and states for the hysteresis function.

6.57.2 Compound Type Members

Table 6-58. GFLIB_HYST_T_FLT members description

Type	Name	Description
tFloat	fltHystOn	Value determining the upper threshold.
tFloat	fltHystOff	Value determining the lower threshold.
tFloat	fltOutValOn	
tFloat	fltOutValOff	
tFloat	fltOutState	Actual state of the output.

6.58 GFLIB_INTEGRATOR_TR_T_F16

```
#include <GFLIB_IntegratorTR.h>
```

6.58.1 Description

Structure containing integrator parameters and coefficients.

6.58.2 Compound Type Members

Table 6-59. GFLIB_INTEGRATOR_TR_T_F16 members description

Type	Name	Description
tFrac32	f32State	Integrator state value.
tFrac16	f16InK1	Input value in step k-1.
tFrac16	f16C1	Integrator coefficient = $(E_{MAX}/T_s)(U_{MAX}*2)*(2^{-u16NShift})$.
tU16	u16NShift	Scaling factor for the integrator coefficient f16C1, integer format [0, 15].

6.59 GFLIB_INTEGRATOR_TR_T_F32

```
#include <GFLIB_IntegratorTR.h>
```

6.59.1 Description

Structure containing integrator parameters and coefficients.

6.59.2 Compound Type Members

Table 6-60. GFLIB_INTEGRATOR_TR_T_F32 members description

Type	Name	Description
tFrac32	f32State	Integrator state value.
tFrac32	f32InK1	Input value in step k-1.
tFrac32	f32C1	Integrator coefficient = $(E_{MAX}/T_s)(U_{MAX}*2)*(2^{-u16NShift})$.
tU16	u16NShift	Scaling factor for the integrator coefficient f32C1, integer format [0, 31].

6.60 GFLIB_INTEGRATOR_TR_T_FLT

```
#include <GFLIB_IntegratorTR.h>
```

6.60.1 Description

Structure containing integrator parameters and coefficients.

6.60.2 Compound Type Members

Table 6-61. GFLIB_INTEGRATOR_TR_T_FLT members description

Type	Name	Description
tFloat	fltState	Integrator state value, single precision floating point format.
tFloat	fltInK1	Input value in step k-1, single precision floating point format.
tFloat	fltC1	Integrator coefficient, single precision floating point format.

6.61 GFLIB_LIMIT_T_F16

```
#include <GFLIB_Limit.h>
```

6.61.1 Description

Structure containing the limits.

6.61.2 Compound Type Members

Table 6-62. GFLIB_LIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	
tFrac16	f16UpperLimit	

6.62 GFLIB_LIMIT_T_F32

```
#include <GFLIB_Limit.h>
```

6.62.1 Description

Structure containing the limits.

6.62.2 Compound Type Members

Table 6-63. GFLIB_LIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	
tFrac32	f32UpperLimit	

6.63 GFLIB_LIMIT_T_FLT

```
#include <GFLIB_Limit.h>
```

6.63.1 Description

Structure containing the limits.

6.63.2 Compound Type Members

Table 6-64. GFLIB_LIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLowerLimit	
tFloat	fltUpperLimit	

6.64 GFLIB_LOWERLIMIT_T_F16

```
#include <GFLIB_LowerLimit.h>
```

6.64.1 Description

Structure containing the lower limit.

6.64.2 Compound Type Members

Table 6-65. GFLIB_LOWERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16LowerLimit	

6.65 GFLIB_LOWERLIMIT_T_F32

```
#include <GFLIB_LowerLimit.h>
```

6.65.1 Description

Structure containing the lower limit.

6.65.2 Compound Type Members

Table 6-66. GFLIB_LOWERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32LowerLimit	

6.66 GFLIB_LOWERLIMIT_T_FLT

```
#include <GFLIB_LowerLimit.h>
```

6.66.1 Description

Structure containing the lower limit.

6.66.2 Compound Type Members

Table 6-67. GFLIB_LOWERLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLowerLimit	

6.67 GFLIB_LUT1D_T_F16

```
#include <GFLIB_Lut1D.h>
```

6.67.1 Description

Structure containing 1D look-up table parameters.

6.67.2 Compound Type Members

Table 6-68. GFLIB_LUT1D_T_F16 members description

Type	Name	Description
tS16	s16ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
tS16	s16ShamIntvl	Shift amount for extracting the interval index of an interpolated interval.
const tFrac16 *	pf16Table	

6.68 GFLIB_LUT1D_T_F32

```
#include <GFLIB_Lut1D.h>
```

6.68.1 Description

Structure containing 1D look-up table parameters.

6.68.2 Compound Type Members

Table 6-69. GFLIB_LUT1D_T_F32 members description

Type	Name	Description
tS32	s32ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval.
tS32	s32ShamIntvl	Shift amount for extracting the interval index of an interpolated interval.
const tFrac32 *	pf32Table	

6.69 GFLIB_LUT1D_T_FLT

```
#include <GFLIB_Lut1D.h>
```

6.69.1 Description

Structure containing 1D look-up table parameters.

6.69.2 Compound Type Members

Table 6-70. GFLIB_LUT1D_T_FLT members description

Type	Name	Description
tS32	s32ShamOffset	Shift amount for extracting the fractional offset within an interpolated interval. The value is a 32-bit signed integer number.
const tFloat *	pfltTable	

6.70 GFLIB_LUT2D_T_F16

```
#include <GFLIB_Lut2D.h>
```

6.70.1 Description

Structure containing 2D look-up table parameters.

6.70.2 Compound Type Members

Table 6-71. GFLIB_LUT2D_T_F16 members description

Type	Name	Description
tS16	s16ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tS16	s16ShamIntvl1	Shift amount for extracting the interval index of an interpolated interval.
tS16	s16ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
tS16	s16ShamIntvl2	Shift amount for extracting the interval index of an interpolated interval.
const tFrac16 *	pf16Table	

6.71 GFLIB_LUT2D_T_F32

```
#include <GFLIB_Lut2D.h>
```

6.71.1 Description

Structure containing 2D look-up table parameters.

6.71.2 Compound Type Members

Table 6-72. GFLIB_LUT2D_T_F32 members description

Type	Name	Description
tS32	s32ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval.
tS32	s32ShamIntvl1	Shift amount for extracting the interval index of an interpolated interval.
tS32	s32ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval.
tS32	s32ShamIntvl2	Shift amount for extracting the interval index of an interpolated interval.
const tFrac32 *	pf32Table	

6.72 GFLIB_LUT2D_T_FLT

```
#include <GFLIB_Lut2D.h>
```

6.72.1 Description

Structure containing 2D look-up table parameters.

6.72.2 Compound Type Members

Table 6-73. GFLIB_LUT2D_T_FLT members description

Type	Name	Description
tS32	s32ShamOffset1	Shift amount for extracting the fractional offset within an interpolated interval. The value is a 32-bit signed integer number.

Table continues on the next page...

Table 6-73. GFLIB_LUT2D_T_FLT members description (continued)

Type	Name	Description
tS32	s32ShamOffset2	Shift amount for extracting the fractional offset within an interpolated interval. The value is a 32-bit signed integer number.
const tFloat *	pfltTable	

6.73 GFLIB_RAMP_T_F16

```
#include <GFLIB_Ramp.h>
```

6.73.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

6.73.2 Compound Type Members

Table 6-74. GFLIB_RAMP_T_F16 members description

Type	Name	Description
tFrac16	f16State	
tFrac16	f16RampUp	
tFrac16	f16RampDown	

6.74 GFLIB_RAMP_T_F32

```
#include <GFLIB_Ramp.h>
```

6.74.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

6.74.2 Compound Type Members

Table 6-75. GFLIB_RAMP_T_F32 members description

Type	Name	Description
tFrac32	f32State	
tFrac32	f32RampUp	
tFrac32	f32RampDown	

6.75 GFLIB_RAMP_T_FLT

```
#include <GFLIB_Ramp.h>
```

6.75.1 Description

Structure containing increment/decrement coefficients and state value for the ramp function implemented in GFLIB_Ramp.

6.75.2 Compound Type Members

Table 6-76. GFLIB_RAMP_T_FLT members description

Type	Name	Description
tFloat	fltState	
tFloat	fltRampUp	
tFloat	fltRampDown	

6.76 GFLIB_SIN_T_F16

```
#include <GFLIB_Sin.h>
```

6.76.1 Description

Array of four 16-bit elements for storing coefficients of the Taylor polynomial.

6.76.2 Compound Type Members

Table 6-77. GFLIB_SIN_T_F16 members description

Type	Name	Description
tFrac16	f16A	

6.77 GFLIB_SIN_T_F32

```
#include <GFLIB_Sin.h>
```

6.77.1 Description

Array of five 32-bit elements for storing coefficients of the Taylor polynomial.

6.77.2 Compound Type Members

Table 6-78. GFLIB_SIN_T_F32 members description

Type	Name	Description
tFrac32	f32A	

6.78 GFLIB_SIN_T_FLT

```
#include <GFLIB_Sin.h>
```

6.78.1 Description

Array of three single precision floating point elements for storing coefficients of the floating point optimized minimax approximation polynomial.

6.78.2 Compound Type Members

Table 6-79. GFLIB_SIN_T_FLT members description

Type	Name	Description
tFloat	fltA	

6.79 GFLIB_TAN_T_F16

```
#include <GFLIB_Tan.h>
```

6.79.1 Description

Output of tan(π * f16In) for interval [0, π/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F16) structure.

6.79.2 Compound Type Members

Table 6-80. GFLIB_TAN_T_F16 members description

Type	Name	Description
GFLIB_TAN_TAYLOR_COEF_T_F16	GFLIB_TAN_SECTOR	

6.80 GFLIB_TAN_T_F32

```
#include <GFLIB_Tan.h>
```

6.80.1 Description

Output of tan(π * f32In) for interval [0, π/4) of the input angles is divided into eight sub-sectors. Polynomial approximation is done using a 4th order polynomial, for each sub-sector respectively. Eight arrays, each including four polynomial coefficients for each sub-interval, are stored in this (GFLIB_TAN_T_F32) structure.

6.80.2 Compound Type Members

Table 6-81. GFLIB_TAN_T_F32 members description

Type	Name	Description
GFLIB_TAN_TAYLOR_COEF_T_F32	GFLIB_TAN_SECTOR	

6.81 GFLIB_TAN_T_FLT

```
#include <GFLIB_Tan.h>
```

6.81.1 Description

Polynomial coefficient for fractional approximation in single precision floating point format.

6.81.2 Compound Type Members

Table 6-82. GFLIB_TAN_T_FLT members description

Type	Name	Description
tFloat	fltA	

6.82 GFLIB_TAN_TAYLOR_COEF_T_F16

```
#include <GFLIB_Tan.h>
```

6.82.1 Description

Structure containing four polynomial coefficients for one sub-interval.

6.82.2 Compound Type Members

Table 6-83. GFLIB_TAN_TAYLOR_COEF_T_F16 members description

Type	Name	Description
const tFrac16	f16A	

6.83 GFLIB_TAN_TAYLOR_COEF_T_F32

```
#include <GFLIB_Tan.h>
```

6.83.1 Description

Structure containing four polynomial coefficients for one sub-interval.

6.83.2 Compound Type Members

Table 6-84. GFLIB_TAN_TAYLOR_COEF_T_F32 members description

Type	Name	Description
const tFrac32	f32A	

6.84 GFLIB_UPPERLIMIT_T_F16

```
#include <GFLIB_UpperLimit.h>
```

6.84.1 Description

Structure containing the upper limit.

6.84.2 Compound Type Members

Table 6-85. GFLIB_UPPERLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16UpperLimit	

6.85 GFLIB_UPPERLIMIT_T_F32

```
#include <GFLIB_UpperLimit.h>
```

6.85.1 Description

Structure containing the upper limit.

6.85.2 Compound Type Members

Table 6-86. GFLIB_UPPERLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32UpperLimit	

6.86 GFLIB_UPPERLIMIT_T_FLT

```
#include <GFLIB_UpperLimit.h>
```

6.86.1 Description

Structure containing the upper limit.

6.86.2 Compound Type Members

Table 6-87. GFLIB_UPPERLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltUpperLimit	

6.87 GFLIB_VECTORLIMIT_T_F16

```
#include <GFLIB_VectorLimit.h>
```

6.87.1 Description

Structure containing the limit.

6.87.2 Compound Type Members

Table 6-88. GFLIB_VECTORLIMIT_T_F16 members description

Type	Name	Description
tFrac16	f16Limit	

6.88 GFLIB_VECTORLIMIT_T_F32

```
#include <GFLIB_VectorLimit.h>
```

6.88.1 Description

Structure containing the limit.

6.88.2 Compound Type Members

Table 6-89. GFLIB_VECTORLIMIT_T_F32 members description

Type	Name	Description
tFrac32	f32Limit	

6.89 GFLIB_VECTORLIMIT_T_FLT

```
#include <GFLIB_VectorLimit.h>
```

6.89.1 Description

Structure containing the limit.

6.89.2 Compound Type Members

Table 6-90. GFLIB_VECTORLIMIT_T_FLT members description

Type	Name	Description
tFloat	fltLimit	

6.90 GMCLIB_DECOUPLINGPMSM_T_F16

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.90.1 Description

Structure containing coefficients for calculation of the decoupling.

6.90.2 Compound Type Members

Table 6-91. GMCLIB_DECOUPLINGPMSM_T_F16 members description

Type	Name	Description
tFrac16	f16Kd	
tS16	s16KdShift	Scaling coefficient k_{d_shift}.
tFrac16	f16Kq	
tS16	s16KqShift	Scaling coefficient k_{q_shift}.

6.91 GMCLIB_DECOUPLINGPMSM_T_F32

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.91.1 Description

Structure containing coefficients for calculation of the decoupling.

6.91.2 Compound Type Members

Table 6-92. GMCLIB_DECOUPLINGPMSM_T_F32 members description

Type	Name	Description
tFrac32	f32Kd	
tS16	s16KdShift	Scaling coefficient k_{d_shift}.
tFrac32	f32Kq	
tS16	s16KqShift	Scaling coefficient k_{q_shift}.

6.92 GMCLIB_DECOUPLINGPMSM_T_FLT

```
#include <GMCLIB_DecouplingPMSM.h>
```

6.92.1 Description

Structure containing coefficients for calculation of the decoupling.

6.92.2 Compound Type Members

Table 6-93. GMCLIB_DECOUPLINGPMSM_T_FLT members description

Type	Name	Description
tFloat	fltLD	
tFloat	fltLQ	

6.93 GMCLIB_ELIMDCBUSRIP_T_F16

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.93.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

6.93.2 Compound Type Members

Table 6-94. GMCLIB_ELIMDCBUSRIP_T_F16 members description

Type	Name	Description
tFrac16	f16ModIndex	
tFrac16	f16ArgDcBusMsr	

6.94 GMCLIB_ELIMDCBUSRIP_T_F32

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.94.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

6.94.2 Compound Type Members

Table 6-95. GMCLIB_ELIMDCBUSRIP_T_F32 members description

Type	Name	Description
tFrac32	f32ModIndex	
tFrac32	f32ArgDcBusMsr	

6.95 GMCLIB_ELIMDCBUSRIP_T_FLT

```
#include <GMCLIB_ElimDcBusRip.h>
```

6.95.1 Description

Structure containing the PWM modulation index and the measured value of the DC bus voltage.

6.95.2 Compound Type Members

Table 6-96. GMCLIB_ELIMDCBUSRIP_T_FLT members description

Type	Name	Description
tFloat	fltModIndex	
tFloat	fltArgDcBusMsr	

6.96 MCLIB_VERSION_T

```
#include <SWLIBS_Version.h>
```

6.96.1 Description

Motor Control Library Set identification structure.

6.96.2 Compound Type Members

Table 6-97. MCLIB_VERSION_T members description

Type	Name	Description
tU16	mcId	MCLIB identification code
tU16	mcVersion	MCLIB version code
tU16	mcImpl	MCLIB supported implementation code
tU16	mcDefaultImpl	MCLIB default implementation code

6.97 SWLIBS_2Syst_F16

```
#include <SWLIBS_Typedefs.h>
```

6.97.1 Description

Array of two standard 16-bit fractional arguments.

6.97.2 Compound Type Members

Table 6-98. SWLIBS_2Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument

6.98 SWLIBS_2Syst_F32

```
#include <SWLIBS_Typedefs.h>
```

6.98.1 Description

Array of two standard 32-bit fractional arguments.

6.98.2 Compound Type Members

Table 6-99. SWLIBS_2Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument

6.99 SWLIBS_2Syst_FLT

```
#include <SWLIBS_Typedefs.h>
```

6.99.1 Description

Array of two standard single precision floating point arguments.

6.99.2 Compound Type Members

Table 6-100. SWLIBS_2Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument
tFloat	fltArg2	Second argument

6.100 SWLIBS_3Syst_F16

```
#include <SWLIBS_Typedefs.h>
```

6.100.1 Description

Array of three standard 16-bit fractional arguments.

6.100.2 Compound Type Members

Table 6-101. SWLIBS_3Syst_F16 members description

Type	Name	Description
tFrac16	f16Arg1	First argument
tFrac16	f16Arg2	Second argument
tFrac16	f16Arg3	Third argument

6.101 SWLIBS_3Syst_F32

```
#include <SWLIBS_Typedefs.h>
```

6.101.1 Description

Array of three standard 32-bit fractional arguments.

6.101.2 Compound Type Members

Table 6-102. SWLIBS_3Syst_F32 members description

Type	Name	Description
tFrac32	f32Arg1	First argument
tFrac32	f32Arg2	Second argument
tFrac32	f32Arg3	Third argument

6.102 SWLIBS_3Syst_FLT

```
#include <SWLIBS_Typedefs.h>
```

6.102.1 Description

Array of three standard single precision floating point arguments.

6.102.2 Compound Type Members

Table 6-103. SWLIBS_3Syst_FLT members description

Type	Name	Description
tFloat	fltArg1	First argument
tFloat	fltArg2	Second argument
tFloat	fltArg3	Third argument

Chapter 7

7.1 Macro Definitions

7.1.1 Macro Definitions Overview

```
#define F16

#define F16TOINT16

#define F16TOINT32

#define F16TOINT64

#define F16_1_DIVBY_SQRT3

#define F16_SQRT2_DIVBY_2

#define F16_SQRT3_DIVBY_2

#define F32

#define F32TOINT16

#define F32TOINT32

#define F32TOINT64

#define F32_1_DIVBY_SQRT3

#define F32_SQRT2_DIVBY_2

#define F32_SQRT3_DIVBY_2

#define F64TOINT16

#define F64TOINT32

#define F64TOINT64
```

```
#define FLOAT_0_5

#define FLOAT_2_PI

#define FLOAT_4_DIVBY_PI

#define FLOAT_DIVBY_SQRT3

#define FLOAT_MAX

#define FLOAT_MIN

#define FLOAT_MINUS_1

#define FLOAT_PI

#define FLOAT_PI_CORRECTION

#define FLOAT_PI_DIVBY_2

#define FLOAT_PI_DIVBY_4

#define FLOAT_PI_SINGLE_CORRECTION

#define FLOAT_PLUS_1

#define FLOAT_SQRT3_DIVBY_2

#define FLOAT_SQRT3_DIVBY_4

#define FLOAT_SQRT3_DIVBY_4_CORRECTION

#define FLT

#define FRAC16

#define FRAC16_0_25

#define FRAC16_0_5

#define FRAC32

#define FRAC32_0_25

#define FRAC32_0_5

#define FRACT_MAX

#define FRACT_MIN

#define GDFLIB_FILTERFIR_PARAM_T

#define GDFLIB_FILTERFIR_PARAM_T
```

```
#define GDFLIB_FILTERFIR_PARAM_T

#define GDFLIB_FILTERFIR_STATE_T

#define GDFLIB_FILTERFIR_STATE_T

#define GDFLIB_FILTERFIR_STATE_T

#define GDFLIB_FILTER_IIR1_DEFAULT

#define GDFLIB_FILTER_IIR1_DEFAULT

#define GDFLIB_FILTER_IIR1_DEFAULT

#define GDFLIB_FILTER_IIR1_DEFAULT_F16

#define GDFLIB_FILTER_IIR1_DEFAULT_F32

#define GDFLIB_FILTER_IIR1_DEFAULT_FLT

#define GDFLIB_FILTER_IIR1_T

#define GDFLIB_FILTER_IIR1_T

#define GDFLIB_FILTER_IIR1_T

#define GDFLIB_FILTER_IIR2_DEFAULT

#define GDFLIB_FILTER_IIR2_DEFAULT

#define GDFLIB_FILTER_IIR2_DEFAULT

#define GDFLIB_FILTER_IIR2_DEFAULT_F16

#define GDFLIB_FILTER_IIR2_DEFAULT_F32

#define GDFLIB_FILTER_IIR2_DEFAULT_FLT

#define GDFLIB_FILTER_IIR2_T

#define GDFLIB_FILTER_IIR2_T

#define GDFLIB_FILTER_IIR2_T

#define GDFLIB_FILTER_MA_DEFAULT

#define GDFLIB_FILTER_MA_DEFAULT

#define GDFLIB_FILTER_MA_DEFAULT

#define GDFLIB_FILTER_MA_DEFAULT_F16

#define GDFLIB_FILTER_MA_DEFAULT_F32
```

#define GDFLIB_FILTER_MA_DEFAULT_FLT

#define GDFLIB_FILTER_MA_T

#define GDFLIB_FILTER_MA_T

#define GDFLIB_FILTER_MA_T

#define GDFLIB_FilterFIR

#define GDFLIB_FilterFIRInit

#define GDFLIB_FilterIIR1

#define GDFLIB_FilterIIR1Init

#define GDFLIB_FilterIIR2

#define GDFLIB_FilterIIR2Init

#define GDFLIB_FilterMA

#define GDFLIB_FilterMAInit

#define GFLIB_ACOS_DEFAULT

#define GFLIB_ACOS_DEFAULT

#define GFLIB_ACOS_DEFAULT

#define GFLIB_ACOS_DEFAULT_F16

#define GFLIB_ACOS_DEFAULT_F32

#define GFLIB_ACOS_DEFAULT_FLT

#define GFLIB_ACOS_T

#define GFLIB_ACOS_T

#define GFLIB_ACOS_T

#define GFLIB_ASIN_DEFAULT

#define GFLIB_ASIN_DEFAULT

#define GFLIB_ASIN_DEFAULT

#define GFLIB_ASIN_DEFAULT_F16

#define GFLIB_ASIN_DEFAULT_F32

#define GFLIB_ASIN_DEFAULT_FLT

```
#define GFLIB_ASIN_T

#define GFLIB_ASIN_T

#define GFLIB_ASIN_T

#define GFLIB_ATANYXSHIFTED_T

#define GFLIB_ATANYXSHIFTED_T

#define GFLIB_ATANYXSHIFTED_T

#define GFLIB_ATAN_DEFAULT

#define GFLIB_ATAN_DEFAULT

#define GFLIB_ATAN_DEFAULT

#define GFLIB_ATAN_DEFAULT_F16

#define GFLIB_ATAN_DEFAULT_F32

#define GFLIB_ATAN_DEFAULT_FLT

#define GFLIB_ATAN_T

#define GFLIB_ATAN_T

#define GFLIB_ATAN_T

#define GFLIB_Acos

#define GFLIB_Asin

#define GFLIB_Atan

#define GFLIB_AtanYX

#define GFLIB_AtanYXShifted

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT

#define GFLIB_CONTROLLER_PIAW_P_T
```

```
#define GFLIB_CONTROLLER_PIAW_P_T

#define GFLIB_CONTROLLER_PIAW_P_T

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT

#define GFLIB_CONTROLLER_PIAW_R_T

#define GFLIB_CONTROLLER_PIAW_R_T

#define GFLIB_CONTROLLER_PIAW_R_T

#define GFLIB_CONTROLLER_PI_P_DEFAULT

#define GFLIB_CONTROLLER_PI_P_DEFAULT

#define GFLIB_CONTROLLER_PI_P_DEFAULT

#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16

#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32

#define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT

#define GFLIB_CONTROLLER_PI_P_T

#define GFLIB_CONTROLLER_PI_P_T

#define GFLIB_CONTROLLER_PI_P_T

#define GFLIB_CONTROLLER_PI_R_DEFAULT

#define GFLIB_CONTROLLER_PI_R_DEFAULT

#define GFLIB_CONTROLLER_PI_R_DEFAULT

#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16

#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32

#define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT

#define GFLIB_CONTROLLER_PI_R_T
```

```
#define GFLIB_CONTROLLER_PI_R_T

#define GFLIB_CONTROLLER_PI_R_T

#define GFLIB_COS_DEFAULT

#define GFLIB_COS_DEFAULT

#define GFLIB_COS_DEFAULT

#define GFLIB_COS_DEFAULT_F16

#define GFLIB_COS_DEFAULT_F32

#define GFLIB_COS_DEFAULT_FLT

#define GFLIB_COS_T

#define GFLIB_COS_T

#define GFLIB_COS_T

#define GFLIB_ControllerPIp

#define GFLIB_ControllerPIpAW

#define GFLIB_ControllerPIr

#define GFLIB_ControllerPIrAW

#define GFLIB_Cos

#define GFLIB_HYST_DEFAULT

#define GFLIB_HYST_DEFAULT

#define GFLIB_HYST_DEFAULT

#define GFLIB_HYST_DEFAULT_F16

#define GFLIB_HYST_DEFAULT_F32

#define GFLIB_HYST_DEFAULT_FLT

#define GFLIB_HYST_T

#define GFLIB_HYST_T

#define GFLIB_HYST_T

#define GFLIB_Hyst

#define GFLIB_INTEGRATOR_TR_DEFAULT
```

#define GFLIB_INTEGRATOR_TR_DEFAULT

#define GFLIB_INTEGRATOR_TR_DEFAULT

#define GFLIB_INTEGRATOR_TR_DEFAULT_F16

#define GFLIB_INTEGRATOR_TR_DEFAULT_F32

#define GFLIB_INTEGRATOR_TR_DEFAULT_FLT

#define GFLIB_INTEGRATOR_TR_T

#define GFLIB_INTEGRATOR_TR_T

#define GFLIB_INTEGRATOR_TR_T

#define GFLIB_IntegratorTR

#define GFLIB_LIMIT_DEFAULT

#define GFLIB_LIMIT_DEFAULT

#define GFLIB_LIMIT_DEFAULT

#define GFLIB_LIMIT_DEFAULT_F16

#define GFLIB_LIMIT_DEFAULT_F32

#define GFLIB_LIMIT_DEFAULT_FLT

#define GFLIB_LIMIT_T

#define GFLIB_LIMIT_T

#define GFLIB_LIMIT_T

#define GFLIB_LOWERLIMIT_DEFAULT

#define GFLIB_LOWERLIMIT_DEFAULT

#define GFLIB_LOWERLIMIT_DEFAULT

#define GFLIB_LOWERLIMIT_DEFAULT_F16

#define GFLIB_LOWERLIMIT_DEFAULT_F32

#define GFLIB_LOWERLIMIT_DEFAULT_FLT

#define GFLIB_LOWERLIMIT_T

#define GFLIB_LOWERLIMIT_T

#define GFLIB_LOWERLIMIT_T

```
#define GFLIB_LUT1D_DEFAULT

#define GFLIB_LUT1D_DEFAULT

#define GFLIB_LUT1D_DEFAULT

#define GFLIB_LUT1D_DEFAULT_F16

#define GFLIB_LUT1D_DEFAULT_F32

#define GFLIB_LUT1D_DEFAULT_FLT

#define GFLIB_LUT1D_T

#define GFLIB_LUT1D_T

#define GFLIB_LUT1D_T

#define GFLIB_LUT2D_DEFAULT

#define GFLIB_LUT2D_DEFAULT

#define GFLIB_LUT2D_DEFAULT

#define GFLIB_LUT2D_DEFAULT_F16

#define GFLIB_LUT2D_DEFAULT_F32

#define GFLIB_LUT2D_DEFAULT_FLT

#define GFLIB_LUT2D_T

#define GFLIB_LUT2D_T

#define GFLIB_LUT2D_T

#define GFLIB_Limit

#define GFLIB_LowerLimit

#define GFLIB_Lut1D

#define GFLIB_Lut2D

#define GFLIB_RAMP_DEFAULT

#define GFLIB_RAMP_DEFAULT

#define GFLIB_RAMP_DEFAULT

#define GFLIB_RAMP_DEFAULT_F16

#define GFLIB_RAMP_DEFAULT_F32
```

```
#define GFLIB_RAMP_DEFAULT_FLT

#define GFLIB_RAMP_T

#define GFLIB_RAMP_T

#define GFLIB_RAMP_T

#define GFLIB_Ramp

#define GFLIB_SIN_DEFAULT

#define GFLIB_SIN_DEFAULT

#define GFLIB_SIN_DEFAULT

#define GFLIB_SIN_DEFAULT_F16

#define GFLIB_SIN_DEFAULT_F32

#define GFLIB_SIN_DEFAULT_FLT

#define GFLIB_SIN_T

#define GFLIB_SIN_T

#define GFLIB_SIN_T

#define GFLIB_Sign

#define GFLIB_Sin

#define GFLIB_Sqrt

#define GFLIB_TAN_DEFAULT

#define GFLIB_TAN_DEFAULT

#define GFLIB_TAN_DEFAULT

#define GFLIB_TAN_DEFAULT_F16

#define GFLIB_TAN_DEFAULT_F32

#define GFLIB_TAN_DEFAULT_FLT

#define GFLIB_TAN_LIMIT_FLT

#define GFLIB_TAN_T

#define GFLIB_TAN_T

#define GFLIB_TAN_T
```

```
#define GFLIB_Tan

#define GFLIB_UPPERLIMIT_DEFAULT

#define GFLIB_UPPERLIMIT_DEFAULT

#define GFLIB_UPPERLIMIT_DEFAULT

#define GFLIB_UPPERLIMIT_DEFAULT_F16

#define GFLIB_UPPERLIMIT_DEFAULT_F32

#define GFLIB_UPPERLIMIT_DEFAULT_FLT

#define GFLIB_UPPERLIMIT_T

#define GFLIB_UPPERLIMIT_T

#define GFLIB_UPPERLIMIT_T

#define GFLIB_UpperLimit

#define GFLIB_VECTORLIMIT_DEFAULT

#define GFLIB_VECTORLIMIT_DEFAULT

#define GFLIB_VECTORLIMIT_DEFAULT

#define GFLIB_VECTORLIMIT_DEFAULT_F16

#define GFLIB_VECTORLIMIT_DEFAULT_F32

#define GFLIB_VECTORLIMIT_DEFAULT_FLT

#define GFLIB_VECTORLIMIT_T

#define GFLIB_VECTORLIMIT_T

#define GFLIB_VECTORLIMIT_T

#define GFLIB_VectorLimit

#define GMCLIB_Clark

#define GMCLIB_ClarkInv

#define GMCLIB_DECOUPLINGPMSM_DEFAULT

#define GMCLIB_DECOUPLINGPMSM_DEFAULT

#define GMCLIB_DECOUPLINGPMSM_DEFAULT

#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16
```

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

#define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT

#define GMCLIB_DECOUPLINGPMSM_T

#define GMCLIB_DECOUPLINGPMSM_T

#define GMCLIB_DECOUPLINGPMSM_T

#define GMCLIB_DecouplingPMSM

#define GMCLIB_ELIMDCBUSRIP_DEFAULT

#define GMCLIB_ELIMDCBUSRIP_DEFAULT

#define GMCLIB_ELIMDCBUSRIP_DEFAULT

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

#define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT

#define GMCLIB_ELIMDCBUSRIP_T

#define GMCLIB_ELIMDCBUSRIP_T

#define GMCLIB_ELIMDCBUSRIP_T

#define GMCLIB_ElimDcBusRip

#define GMCLIB_Park

#define GMCLIB_ParkInv

#define GMCLIB_SvmStd

#define INT16TOF16

#define INT16TOF32

#define INT16TOINT32

#define INT16_MAX

#define INT16_MIN

#define INT32TOF16

#define INT32TOF32

#define INT32TOINT16
```

```
#define INT32TOINT64

#define INT32_MAX

#define INT32_MIN

#define INT64TOF16

#define INT64TOF32

#define INT64TOINT32

#define MCLIB_DEFAULT_IMPLEMENTATION

#define MCLIB_DEFAULT_IMPLEMENTATION_F16

#define MCLIB_DEFAULT_IMPLEMENTATION_F32

#define MCLIB_DEFAULT_IMPLEMENTATION_FLT

#define MCLIB_ID

#define MCLIB_MCID_SIZE

#define MCLIB_MCIMPLEMENTATION_SIZE

#define MCLIB_MCVERSION_SIZE

#define MCLIB_STD_OFF

#define MCLIB_STD_ON

#define MCLIB_SUPPORTED_IMPLEMENTATION

#define MCLIB_SUPPORT_F16

#define MCLIB_SUPPORT_F32

#define MCLIB_SUPPORT_FLT

#define MCLIB_VERSION

#define MCLIB_VERSION_DEFAULT

#define MLIB_Abs

#define MLIB_AbsSat

#define MLIB_Add

#define MLIB_AddSat

#define MLIB_Convert
```

```
#define MLIB_ConvertPU

#define MLIB_Div

#define MLIB_DivSat

#define MLIB_Mac

#define MLIB_MacSat

#define MLIB_Mul

#define MLIB_MulSat

#define MLIB_Neg

#define MLIB_NegSat

#define MLIB_Norm

#define MLIB_Round

#define MLIB_ShBi

#define MLIB_ShBiSat

#define MLIB_ShL

#define MLIB_ShLSat

#define MLIB_ShR

#define MLIB_Sub

#define MLIB_SubSat

#define MLIB_VMac

#define SFRACT_MAX

#define SFRACT_MIN

#define inline
```

Chapter 8
Macro References

This section describes in details the macro definitions available in Embedded Software and Motor Control Libraries for MPXR40xx.

8.1 Define GDFLIB_FilterFIRInit

```
#include <GDFLIB_FilterFIR.h>
```

8.1.1 Macro Definition

```
#define GDFLIB_FilterFIRInit macro_dispatcher(GDFLIB_FilterFIRInit, __VA_ARGS__)(__VA_ARGS__)
```

8.1.2 Description

This function initializes the FIR filter buffers.

8.2 Define GDFLIB_FilterFIR

```
#include <GDFLIB_FilterFIR.h>
```

8.2.1 Macro Definition

```
#define GDFLIB_FilterFIR macro_dispatcher(GDFLIB_FilterFIR, __VA_ARGS__)(__VA_ARGS__)
```

8.2.2 Description

The function performs a single iteration of an FIR filter.

8.3 Define GDFLIB_FILTERFIR_PARAM_T

```
#include <GDFLIB_FilterFIR.h>
```

8.3.1 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_FLT
```

8.3.2 Description

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_FLT datatype in case the single precision floating point implementation is selected.

8.4 Define GDFLIB_FILTERFIR_PARAM_T

```
#include <GDFLIB_FilterFIR.h>
```

8.4.1 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_FLT
```

8.4.2 Description

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_FLT datatype in case the single precision floating point implementation is selected.

8.5 Define GDFLIB_FILTERFIR_PARAM_T

```
#include <GDFLIB_FilterFIR.h>
```

8.5.1 Macro Definition

```
#define GDFLIB_FILTERFIR_PARAM_T GDFLIB_FILTERFIR_PARAM_T_FLT
```

8.5.2 Description

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_PARAM_T_FLT datatype in case the single precision floating point implementation is selected.

8.6 Define GDFLIB_FILTERFIR_STATE_T

```
#include <GDFLIB_FilterFIR.h>
```

8.6.1 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_FLT
```

8.6.2 Description

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_FLT datatype in case the single precision floating point implementation is selected.

8.7 Define GDFLIB_FILTERFIR_STATE_T

```
#include <GDFLIB_FilterFIR.h>
```

8.7.1 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_FLT
```

8.7.2 Description

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_FLT datatype in case the single precision floating point implementation is selected.

8.8 Define GDFLIB_FILTERFIR_STATE_T

```
#include <GDFLIB_FilterFIR.h>
```

8.8.1 Macro Definition

```
#define GDFLIB_FILTERFIR_STATE_T GDFLIB_FILTERFIR_STATE_T_FLT
```

8.8.2 Description

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of alias for GDFLIB_FILTERFIR_STATE_T_FLT datatype in case the single precision floating point implementation is selected.

8.9 Define GDFLIB_FilterIIR1Init

```
#include <GDFLIB_FilterIIR1.h>
```

8.9.1 Macro Definition

```
#define GDFLIB_FilterIIR1Init macro_dispatcher(GDFLIB_FilterIIR1Init, __VA_ARGS__)
(__VA_ARGS__)
```

8.9.2 Description

This function initializes the first order IIR filter buffers.

8.10 Define GDFLIB_FilterIIR1

```
#include <GDFLIB_FilterIIR1.h>
```

8.10.1 Macro Definition

```
#define GDFLIB_FilterIIR1 macro_dispatcher(GDFLIB_FilterIIR1, __VA_ARGS__)(__VA_ARGS__)
```

8.10.2 Description

This function implements the first order IIR filter.

8.11 Define GDFLIB_FILTER_IIR1_T

```
#include <GDFLIB_FilterIIR1.h>
```

8.11.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_FLT
```

8.11.2 Description

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_FLT datatype in case the single precision floating point implementation is selected.

8.12 Define GDFLIB_FILTER_IIR1_T

```
#include <GDFLIB_FilterIIR1.h>
```

8.12.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_FLT
```

8.12.2 Description

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_FLT datatype in case the single precision floating point implementation is selected.

8.13 Define GDFLIB_FILTER_IIR1_T

```
#include <GDFLIB_FilterIIR1.h>
```

8.13.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_T GDFLIB_FILTER_IIR1_T_FLT
```

8.13.2 Description

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_T as alias for GDFLIB_FILTER_IIR1_T_FLT datatype in case the single precision floating point implementation is selected.

8.14 Define GDFLIB_FILTER_IIR1_DEFAULT

```
#include <GDFLIB_FilterIIR1.h>
```

8.14.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_FLT
```

8.14.2 Description

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.15 Define GDFLIB_FILTER_IIR1_DEFAULT

```
#include <GDFLIB_FilterIIR1.h>
```

8.15.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_FLT
```

8.15.2 Description

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.16 Define GDFLIB_FILTER_IIR1_DEFAULT

```
#include <GDFLIB_FilterIIR1.h>
```

8.16.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT GDFLIB_FILTER_IIR1_DEFAULT_FLT
```

8.16.2 Description

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR1_DEFAULT as alias for GDFLIB_FILTER_IIR1_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.17 Define GDFLIB_FILTER_IIR1_DEFAULT_F32

```
#include <GDFLIB_FilterIIR1.h>
```

8.17.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F32 {{0,0,0},{0},{0}}
```

8.17.2 Description

Default value for GDFLIB_FILTER_IIR1_T_F32.

8.18 Define GDFLIB_FILTER_IIR1_DEFAULT_F16

```
#include <GDFLIB_FilterIIR1.h>
```

8.18.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_F16 {{0,0,0},{0},{0}}
```

8.18.2 Description

Default value for GDFLIB_FILTER_IIR1_T_F16.

8.19 Define GDFLIB_FILTER_IIR1_DEFAULT_FLT

```
#include <GDFLIB_FilterIIR1.h>
```

8.19.1 Macro Definition

```
#define GDFLIB_FILTER_IIR1_DEFAULT_FLT {{0,0,0},{0},{0}}
```

8.19.2 Description

Default value for GDFLIB_FILTER_IIR1_T_FLT.

8.20 Define GDFLIB_FilterIIR2Init

```
#include <GDFLIB_FilterIIR2.h>
```

8.20.1 Macro Definition

```
#define GDFLIB_FilterIIR2Init macro_dispatcher(GDFLIB_FilterIIR2Init, __VA_ARGS__)
(__VA_ARGS__)
```

8.20.2 Description

This function initializes the second order IIR filter buffers.

8.21 Define GDFLIB_FilterIIR2

```
#include <GDFLIB_FilterIIR2.h>
```

8.21.1 Macro Definition

```
#define GDFLIB_FilterIIR2 macro_dispatcher(GDFLIB_FilterIIR2, __VA_ARGS__)(__VA_ARGS__)
```

8.21.2 Description

This function implements the second order IIR filter.

8.22 Define GDFLIB_FILTER_IIR2_T

```
#include <GDFLIB_FilterIIR2.h>
```

8.22.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_FLT
```

8.22.2 Description

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_FLT datatype in case the single precision floating point implementation is selected.

8.23 Define GDFLIB_FILTER_IIR2_T

```
#include <GDFLIB_FilterIIR2.h>
```

8.23.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_FLT
```

8.23.2 Description

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_FLT datatype in case the single precision floating point implementation is selected.

8.24 Define GDFLIB_FILTER_IIR2_T

```
#include <GDFLIB_FilterIIR2.h>
```

8.24.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_T GDFLIB_FILTER_IIR2_T_FLT
```

8.24.2 Description

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_T as alias for GDFLIB_FILTER_IIR2_T_FLT datatype in case the single precision floating point implementation is selected.

8.25 Define GDFLIB_FILTER_IIR2_DEFAULT

```
#include <GDFLIB_FilterIIR2.h>
```

8.25.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT
```

8.25.2 Description

Definition of GDFLIB_FILTER_IIR2_DEFAULT as alias for GDFLIB_FILTER_IIR2_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.26 Define GDFLIB_FILTER_IIR2_DEFAULT

```
#include <GDFLIB_FilterIIR2.h>
```

8.26.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT
```

8.26.2 Description

Definition of GDFLIB_FILTER_IIR2_DEFAULT as alias for GDFLIB_FILTER_IIR2_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.27 Define GDFLIB_FILTER_IIR2_DEFAULT

```
#include <GDFLIB_FilterIIR2.h>
```

8.27.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT
```

8.27.2 Description

Definition of GDFLIB_FILTER_IIR2_DEFAULT as alias for GDFLIB_FILTER_IIR2_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_IIR2_DEFAULT GDFLIB_FILTER_IIR2_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.28 Define GDFLIB_FILTER_IIR2_DEFAULT_F32

```
#include <GDFLIB_FilterIIR2.h>
```

8.28.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_F32 {{0,0,0,0,0},{0,0},{0,0}}
```

8.28.2 Description

Default value for GDFLIB_FILTER_IIR2_T_F32.

8.29 Define GDFLIB_FILTER_IIR2_DEFAULT_F16

```
#include <GDFLIB_FilterIIR2.h>
```

8.29.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_F16 {{0,0,0,0,0},{0,0},{0,0}}
```

8.29.2 Description

Default value for GDFLIB_FILTER_IIR2_T_F16.

8.30 Define GDFLIB_FILTER_IIR2_DEFAULT_FLT

```
#include <GDFLIB_FilterIIR2.h>
```

8.30.1 Macro Definition

```
#define GDFLIB_FILTER_IIR2_DEFAULT_FLT {{0,0,0,0,0},{0,0},{0,0}}
```

8.30.2 Description

Default value for GDFLIB_FILTER_IIR2_T_FLT.

8.31 Define GDFLIB_FilterMAInit

```
#include <GDFLIB_FilterMA.h>
```

8.31.1 Macro Definition

```
#define GDFLIB_FilterMAInit macro_dispatcher(GDFLIB_FilterMAInit, __VA_ARGS__)(__VA_ARGS__)
```

8.31.2 Description

This function clears the internal filter accumulator.

8.32 Define GDFLIB_FilterMA

```
#include <GDFLIB_FilterMA.h>
```

8.32.1 Macro Definition

```
#define GDFLIB_FilterMA macro_dispatcher(GDFLIB_FilterMA, __VA_ARGS__)(__VA_ARGS__)
```

8.32.2 Description

This function implements a moving average recursive filter.

8.33 Define GDFLIB_FILTER_MA_T

```
#include <GDFLIB_FilterMA.h>
```

8.33.1 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_FLT
```

8.33.2 Description

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_FLT datatype in case the single precision floating point implementation is selected.

8.34 Define GDFLIB_FILTER_MA_T

```
#include <GDFLIB_FilterMA.h>
```

8.34.1 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_FLT
```

8.34.2 Description

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_FLT datatype in case the single precision floating point implementation is selected.

8.35 Define GDFLIB_FILTER_MA_T

```
#include <GDFLIB_FilterMA.h>
```

8.35.1 Macro Definition

```
#define GDFLIB_FILTER_MA_T GDFLIB_FILTER_MA_T_FLT
```

8.35.2 Description

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_T as alias for GDFLIB_FILTER_MA_T_FLT datatype in case the single precision floating point implementation is selected.

8.36 Define GDFLIB_FILTER_MA_DEFAULT

```
#include <GDFLIB_FilterMA.h>
```

8.36.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_FLT
```

8.36.2 Description

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.37 Define GDFLIB_FILTER_MA_DEFAULT

```
#include <GDFLIB_FilterMA.h>
```

8.37.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_FLT
```

8.37.2 Description

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.38 Define GDFLIB_FILTER_MA_DEFAULT

```
#include <GDFLIB_FilterMA.h>
```

8.38.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT GDFLIB_FILTER_MA_DEFAULT_FLT
```

8.38.2 Description

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GDFLIB_FILTER_MA_DEFAULT as alias for GDFLIB_FILTER_MA_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.39 Define GDFLIB_FILTER_MA_DEFAULT_F32

```
#include <GDFLIB_FilterMA.h>
```

8.39.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F32 {0,0}
```

8.39.2 Description

Default value for GDFLIB_FILTER_MA_T_F32.

8.40 Define GDFLIB_FILTER_MA_DEFAULT_F16

```
#include <GDFLIB_FilterMA.h>
```

8.40.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_F16 {0,0}
```

8.40.2 Description

Default value for GDFLIB_FILTER_MA_T_F16.

8.41 Define GDFLIB_FILTER_MA_DEFAULT_FLT

```
#include <GDFLIB_FilterMA.h>
```

8.41.1 Macro Definition

```
#define GDFLIB_FILTER_MA_DEFAULT_FLT {0,0}
```

8.41.2 Description

Default value for GDFLIB_FILTER_MA_T_FLT.

8.42 Define GFLIB_Acos

```
#include <GFLIB_Acos.h>
```

8.42.1 Macro Definition

```
#define GFLIB_Acos macro_dispatcher(GFLIB_Acos, __VA_ARGS__)(__VA_ARGS__)
```

8.42.2 Description

This function implements polynomial approximation of arccosine function.

8.43 Define GFLIB_ACOS_T

```
#include <GFLIB_Acos.h>
```

8.43.1 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_FLT
```

8.43.2 Description

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_FLT datatype in case the Single precision floating point implementation is selected.

8.44 Define GFLIB_ACOS_T

```
#include <GFLIB_Acos.h>
```

8.44.1 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_FLT
```

8.44.2 Description

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_FLT datatype in case the Single precision floating point implementation is selected.

8.45 Define GFLIB_ACOS_T

```
#include <GFLIB_Acos.h>
```

8.45.1 Macro Definition

```
#define GFLIB_ACOS_T GFLIB_ACOS_T_FLT
```

8.45.2 Description

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_T as alias for GFLIB_ACOS_T_FLT datatype in case the Single precision floating point implementation is selected.

8.46 Define GFLIB_ACOS_DEFAULT

```
#include <GFLIB_Acos.h>
```

8.46.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_FLT
```

8.46.2 Description

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.47 Define GFLIB_ACOS_DEFAULT

```
#include <GFLIB_Acos.h>
```

8.47.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_FLT
```

8.47.2 Description

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.48 Define GFLIB_ACOS_DEFAULT

```
#include <GFLIB_Acos.h>
```

8.48.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT GFLIB_ACOS_DEFAULT_FLT
```

8.48.2 Description

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ACOS_DEFAULT as alias for GFLIB_ACOS_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.49 Define GFLIB_ACOS_DEFAULT_F32

```
#include <GFLIB_Acos.h>
```

8.49.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F32 &f32gflibAcosCoef
```

8.49.2 Description

Default approximation coefficients for GFLIB_Acos_F32 function.

8.50 Define GFLIB_ACOS_DEFAULT_F16

```
#include <GFLIB_Acos.h>
```

8.50.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_F16 &f16gflibAcosCoef
```

8.50.2 Description

Default approximation coefficients for GFLIB_Acos_F16 function.

8.51 Define GFLIB_ACOS_DEFAULT_FLT

```
#include <GFLIB_Acos.h>
```

8.51.1 Macro Definition

```
#define GFLIB_ACOS_DEFAULT_FLT &fltgflibAcosCoef
```

8.51.2 Description

Default approximation coefficients for GFLIB_Acos_FLT function.

8.52 Define GFLIB_Asin

```
#include <GFLIB_Asin.h>
```

8.52.1 Macro Definition

```
#define GFLIB_Asin macro_dispatcher(GFLIB_Asin, __VA_ARGS__)(__VA_ARGS__)
```

8.52.2 Description

This function implements polynomial approximation of arcsine function.

8.53 Define GFLIB_ASIN_T

```
#include <GFLIB_Asin.h>
```

8.53.1 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_FLT
```

8.53.2 Description

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_FLT datatype in case the Single precision floating point implementation is selected.

8.54 Define GFLIB_ASIN_T

```
#include <GFLIB_Asin.h>
```

8.54.1 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_FLT
```

8.54.2 Description

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_FLT datatype in case the Single precision floating point implementation is selected.

8.55 Define GFLIB_ASIN_T

```
#include <GFLIB_Asin.h>
```

8.55.1 Macro Definition

```
#define GFLIB_ASIN_T GFLIB_ASIN_T_FLT
```

8.55.2 Description

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_T as alias for GFLIB_ASIN_T_FLT datatype in case the Single precision floating point implementation is selected.

8.56 Define GFLIB_ASIN_DEFAULT

```
#include <GFLIB_Asin.h>
```

8.56.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_FLT
```

8.56.2 Description

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.57 Define GFLIB_ASIN_DEFAULT

```
#include <GFLIB_Asin.h>
```

8.57.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_FLT
```

8.57.2 Description

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.58 Define GFLIB_ASIN_DEFAULT

```
#include <GFLIB_Asin.h>
```

8.58.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT GFLIB_ASIN_DEFAULT_FLT
```

8.58.2 Description

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ASIN_DEFAULT as alias for GFLIB_ASIN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.59 Define GFLIB_ASIN_DEFAULT_F32

```
#include <GFLIB_Asin.h>
```

8.59.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F32 &f32gflibAsinCoef
```

8.59.2 Description

Default approximation coefficients for GFLIB_Asin_F32 function.

8.60 Define GFLIB_ASIN_DEFAULT_F16

```
#include <GFLIB_Asin.h>
```

8.60.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_F16 &f16gflibAsinCoef
```

8.60.2 Description

Default approximation coefficients for GFLIB_Asin_F16 function.

8.61 Define GFLIB_ASIN_DEFAULT_FLT

```
#include <GFLIB_Asin.h>
```

8.61.1 Macro Definition

```
#define GFLIB_ASIN_DEFAULT_FLT &fltgflibAsinCoef
```

8.61.2 Description

Default approximation coefficients for GFLIB_Asin_FLT function.

8.62 Define GFLIB_Atan

```
#include <GFLIB_Atan.h>
```

8.62.1 Macro Definition

```
#define GFLIB_Atan macro_dispatcher(GFLIB_Atan, __VA_ARGS__)(__VA_ARGS__)
```

8.62.2 Description

This function implements polynomial approximation of arctangent function.

8.63 Define GFLIB_ATAN_T

```
#include <GFLIB_Atan.h>
```

8.63.1 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_FLT
```

8.63.2 Description

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_FLT datatype in case the Single precision floating point implementation is selected.

8.64 Define GFLIB_ATAN_T

```
#include <GFLIB_Atan.h>
```

8.64.1 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_FLT
```

8.64.2 Description

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_FLT datatype in case the Single precision floating point implementation is selected.

8.65 Define GFLIB_ATAN_T

```
#include <GFLIB_Atan.h>
```

8.65.1 Macro Definition

```
#define GFLIB_ATAN_T GFLIB_ATAN_T_FLT
```

8.65.2 Description

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_T as alias for GFLIB_ATAN_T_FLT datatype in case the Single precision floating point implementation is selected.

8.66 Define GFLIB_ATAN_DEFAULT

```
#include <GFLIB_Atan.h>
```

8.66.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_FLT
```

8.66.2 Description

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.67 Define GFLIB_ATAN_DEFAULT

```
#include <GFLIB_Atan.h>
```

8.67.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_FLT
```

8.67.2 Description

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.68 Define GFLIB_ATAN_DEFAULT

```
#include <GFLIB_Atan.h>
```

8.68.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT GFLIB_ATAN_DEFAULT_FLT
```

8.68.2 Description

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATAN_DEFAULT as alias for GFLIB_ATAN_DEFAULT_FLT default value in case the Single precision floating point implementation is selected.

8.69 Define GFLIB_ATAN_DEFAULT_F32

```
#include <GFLIB_Atan.h>
```

8.69.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F32 &f32gflibAtanCoef
```

8.69.2 Description

Default approximation coefficients for GFLIB_Atan_F32 function.

8.70 Define GFLIB_ATAN_DEFAULT_F16

```
#include <GFLIB_Atan.h>
```

8.70.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_F16 &f16gflibAtanCoef
```

8.70.2 Description

Default approximation coefficients for GFLIB_Atan_F16 function.

8.71 Define GFLIB_ATAN_DEFAULT_FLT

```
#include <GFLIB_Atan.h>
```

8.71.1 Macro Definition

```
#define GFLIB_ATAN_DEFAULT_FLT &fltgflibAtanCoef
```

8.71.2 Description

Default approximation coefficients for GFLIB_Atan_FLT function.

8.72 Define GFLIB_AtanYX

```
#include <GFLIB_AtanYX.h>
```

8.72.1 Macro Definition

```
#define GFLIB_AtanYX macro_dispatcher(GFLIB_AtanYX, __VA_ARGS__)(__VA_ARGS__)
```

8.72.2 Description

This function calculate the angle between the positive x-axis and the direction of a vector given by the (x, y) coordinates.

8.73 Define GFLIB_AtanYXShifted

```
#include <GFLIB_AtanYXShifted.h>
```

8.73.1 Macro Definition

```
#define GFLIB_AtanYXShifted macro_dispatcher(GFLIB_AtanYXShifted, __VA_ARGS__)(__VA_ARGS__)
```

8.73.2 Description

This function calculates the angle of two sine waves shifted in phase to each other.

8.74 Define GFLIB_ATANYXSHIFTED_T

```
#include <GFLIB_AtanYXShifted.h>
```

8.74.1 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_FLT
```

8.74.2 Description

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_FLT datatype in case the single precision floating point implementation is selected.

8.75 Define GFLIB_ATANYXSHIFTED_T

```
#include <GFLIB_AtanYXShifted.h>
```

8.75.1 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_FLT
```

8.75.2 Description

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_FLT datatype in case the single precision floating point implementation is selected.

8.76 Define GFLIB_ATANYXSHIFTED_T

```
#include <GFLIB_AtanYXShifted.h>
```

8.76.1 Macro Definition

```
#define GFLIB_ATANYXSHIFTED_T GFLIB_ATANYXSHIFTED_T_FLT
```

8.76.2 Description

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_ATANYXSHIFTED_T as alias for GFLIB_ATANYXSHIFTED_T_FLT datatype in case the single precision floating point implementation is selected.

8.77 Define GFLIB_ControllerPIp

```
#include <GFLIB_ControllerPIp.h>
```

8.77.1 Macro Definition

```
#define GFLIB_ControllerPIp macro_dispatcher(GFLIB_ControllerPIp, __VA_ARGS__)(__VA_ARGS__)
```

8.77.2 Description

This function calculates a parallel form of the Proportional- Integral controller, without integral anti-windup.

8.78 Define GFLIB_CONTROLLER_PI_P_T

```
#include <GFLIB_ControllerPIp.h>
```

8.78.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_FLT
```

8.78.2 Description

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_FLT datatype in case the single precision floating point implementation is selected.

8.79 Define GFLIB_CONTROLLER_PI_P_T

```
#include <GFLIB_ControllerPIp.h>
```

8.79.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_FLT
```

8.79.2 Description

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_FLT datatype in case the single precision floating point implementation is selected.

8.80 Define GFLIB_CONTROLLER_PI_P_T

```
#include <GFLIB_ControllerPIp.h>
```

8.80.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_T GFLIB_CONTROLLER_PI_P_T_FLT
```

8.80.2 Description

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_T as alias for GFLIB_CONTROLLER_PI_P_T_FLT datatype in case the single precision floating point implementation is selected.

8.81 Define GFLIB_CONTROLLER_PI_P_DEFAULT

```
#include <GFLIB_ControllerPIp.h>
```

8.81.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT GFLIB_CONTROLLER_PI_P_DEFAULT_FLT
```

8.81.2 Description

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.82 Define GFLIB_CONTROLLER_PI_P_DEFAULT

```
#include <GFLIB_ControllerPIp.h>
```

8.82.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT GFLIB_CONTROLLER_PI_P_DEFAULT_FLT
```

8.82.2 Description

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.83 Define GFLIB_CONTROLLER_PI_P_DEFAULT

```
#include <GFLIB_ControllerPIp.h>
```

8.83.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT GFLIB_CONTROLLER_PI_P_DEFAULT_FLT
```

8.83.2 Description

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_P_DEFAULT as alias for GFLIB_CONTROLLER_PI_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.84 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F32

```
#include <GFLIB_ControllerPIp.h>
```

8.84.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F32 {0,0,0,0,0,0}
```

8.84.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_F32.

8.85 Define GFLIB_CONTROLLER_PI_P_DEFAULT_F16

```
#include <GFLIB_ControllerPIp.h>
```

8.85.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_F16 {0,0,0,0,0,0}
```

8.85.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_F16.

8.86 Define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT

```
#include <GFLIB_ControllerPIp.h>
```

8.86.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_P_DEFAULT_FLT {0,0,0,0}
```

8.86.2 Description

Default value for GFLIB_CONTROLLER_PI_P_T_FLT.

8.87 Define GFLIB_ControllerPIpAW

```
#include <GFLIB_ControllerPIpAW.h>
```

8.87.1 Macro Definition

```
#define GFLIB_ControllerPIpAW macro_dispatcher(GFLIB_ControllerPIpAW, __VA_ARGS__)
(__VA_ARGS__)
```

8.87.2 Description

This function calculates a standard recurrent form of the Proportional- Integral controller, with integral anti-windup.

8.88 Define GFLIB_CONTROLLER_PIAW_P_T

```
#include <GFLIB_ControllerPIpAW.h>
```

8.88.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_FLT
```

8.88.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_FLT datatype in case the single precision floating point implementation is selected.

8.89 Define GFLIB_CONTROLLER_PIAW_P_T

```
#include <GFLIB_ControllerPIpAW.h>
```

8.89.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_FLT
```

8.89.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_FLT datatype in case the single precision floating point implementation is selected.

8.90 Define GFLIB_CONTROLLER_PIAW_P_T

```
#include <GFLIB_ControllerPIpAW.h>
```

8.90.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_T GFLIB_CONTROLLER_PIAW_P_T_FLT
```

8.90.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_T as alias for GFLIB_CONTROLLER_PIAW_P_T_FLT datatype in case the single precision floating point implementation is selected.

8.91 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT

```
#include <GFLIB_ControllerPIpAW.h>
```

8.91.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT
```

8.91.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.92 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT

```
#include <GFLIB_ControllerPIpAW.h>
```

8.92.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT
```

8.92.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.93 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT

```
#include <GFLIB_ControllerPIpAW.h>
```

8.93.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT
```

8.93.2 Description

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_P_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.94 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32

```
#include <GFLIB_ControllerPIpAW.h>
```

8.94.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F32 {0,0,0,0,INT32_MIN,INT32_MAX,0,0,0}
```

8.94.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_F32.

8.95 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16

```
#include <GFLIB_ControllerPIpAW.h>
```

8.95.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_F16 {0,0,0,0,INT16_MIN,INT16_MAX,0,0,0}
```

8.95.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_F16.

8.96 Define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT

```
#include <GFLIB_ControllerPIpAW.h>
```

8.96.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_P_DEFAULT_FLT {0,0,FLOAT_MIN,FLOAT_MAX,0,0,0}
```

8.96.2 Description

Default value for GFLIB_CONTROLLER_PIAW_P_T_FLT.

8.97 Define GFLIB_ControllerPIr

```
#include <GFLIB_ControllerPIr.h>
```

8.97.1 Macro Definition

```
#define GFLIB_ControllerPIr macro_dispatcher(GFLIB_ControllerPIr, __VA_ARGS__)(__VA_ARGS__)
```

8.97.2 Description

This function calculates a standard recurrent form of the Proportional- Integral controller, without integral anti-windup.

8.98 Define GFLIB_CONTROLLER_PI_R_T

```
#include <GFLIB_ControllerPIr.h>
```

8.98.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_FLT
```

8.98.2 Description

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_FLT datatype in case the single precision floating point implementation is selected.

8.99 Define GFLIB_CONTROLLER_PI_R_T

```
#include <GFLIB_ControllerPIr.h>
```

8.99.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_FLT
```

8.99.2 Description

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_FLT datatype in case the single precision floating point implementation is selected.

8.100 Define GFLIB_CONTROLLER_PI_R_T

```
#include <GFLIB_ControllerPIr.h>
```

8.100.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_T GFLIB_CONTROLLER_PI_R_T_FLT
```

8.100.2 Description

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_T as alias for GFLIB_CONTROLLER_PI_R_T_FLT datatype in case the single precision floating point implementation is selected.

8.101 Define GFLIB_CONTROLLER_PI_R_DEFAULT

```
#include <GFLIB_ControllerPIr.h>
```

8.101.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT GFLIB_CONTROLLER_PI_R_DEFAULT_FLT
```

8.101.2 Description

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.102 Define GFLIB_CONTROLLER_PI_R_DEFAULT

```
#include <GFLIB_ControllerPIr.h>
```

8.102.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT GFLIB_CONTROLLER_PI_R_DEFAULT_FLT
```

8.102.2 Description

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.103 Define GFLIB_CONTROLLER_PI_R_DEFAULT

```
#include <GFLIB_ControllerPIr.h>
```

8.103.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT GFLIB_CONTROLLER_PI_R_DEFAULT_FLT
```

8.103.2 Description

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PI_R_DEFAULT as alias for GFLIB_CONTROLLER_PI_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.104 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F32

```
#include <GFLIB_ControllerPIr.h>
```

8.104.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F32 {0,0,0,0,0}
```

8.104.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_F32.

8.105 Define GFLIB_CONTROLLER_PI_R_DEFAULT_F16

```
#include <GFLIB_ControllerPIr.h>
```

8.105.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_F16 {0,0,0,0,0}
```

8.105.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_F16.

8.106 Define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT

```
#include <GFLIB_ControllerPIr.h>
```

8.106.1 Macro Definition

```
#define GFLIB_CONTROLLER_PI_R_DEFAULT_FLT {0,0,0,0}
```

8.106.2 Description

Default value for GFLIB_CONTROLLER_PI_R_T_FLT.

8.107 Define GFLIB_ControllerPIrAW

```
#include <GFLIB_ControllerPIrAW.h>
```

8.107.1 Macro Definition

```
#define GFLIB_ControllerPIrAW macro_dispatcher(GFLIB_ControllerPIrAW, __VA_ARGS__)
(__VA_ARGS__)
```

8.107.2 Description

This function calculates a standard recurrent form of the Proportional-Integral controller, with integral anti-windup.

8.108 Define GFLIB_CONTROLLER_PIAW_R_T

```
#include <GFLIB_ControllerPIrAW.h>
```

8.108.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_FLT
```

8.108.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_FLT datatype in case the single precision floating point implementation is selected.

8.109 Define GFLIB_CONTROLLER_PIAW_R_T

```
#include <GFLIB_ControllerPIrAW.h>
```

8.109.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_FLT
```

8.109.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_FLT datatype in case the single precision floating point implementation is selected.

8.110 Define GFLIB_CONTROLLER_PIAW_R_T

```
#include <GFLIB_ControllerPIrAW.h>
```

8.110.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_T GFLIB_CONTROLLER_PIAW_R_T_FLT
```

8.110.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_T as alias for GFLIB_CONTROLLER_PIAW_R_T_FLT datatype in case the single precision floating point implementation is selected.

8.111 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT

```
#include <GFLIB_ControllerPIrAW.h>
```

8.111.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT
```

8.111.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.112 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT

```
#include <GFLIB_ControllerPIrAW.h>
```

8.112.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT
```

8.112.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.113 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT

```
#include <GFLIB_ControllerPIrAW.h>
```

8.113.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT
```

8.113.2 Description

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_CONTROLLER_PIAW_R_DEFAULT as alias for GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.114 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32

```
#include <GFLIB_ControllerPIrAW.h>
```

8.114.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F32 {0,0,0,0,INT32_MIN,INT32_MAX,0}
```

8.114.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_F32.

8.115 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16

```
#include <GFLIB_ControllerPIrAW.h>
```

8.115.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_F16 {0,0,0,0,INT16_MIN,INT16_MAX,0}
```

8.115.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_F16.

8.116 Define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT

```
#include <GFLIB_ControllerPIrAW.h>
```

8.116.1 Macro Definition

```
#define GFLIB_CONTROLLER_PIAW_R_DEFAULT_FLT {0,0,0,0,FLOAT_MIN,FLOAT_MAX }
```

8.116.2 Description

Default value for GFLIB_CONTROLLER_PIAW_R_T_FLT.

8.117 Define GFLIB_Cos

```
#include <GFLIB_Cos.h>
```

8.117.1 Macro Definition

```
#define GFLIB_Cos macro_dispatcher(GFLIB_Cos, __VA_ARGS__)(__VA_ARGS__)
```

8.117.2 Description

This function implements polynomial approximation of cosine function.

8.118 Define GFLIB_COS_T

```
#include <GFLIB_Cos.h>
```

8.118.1 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_FLT
```

8.118.2 Description

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_FLT datatype in case the single precision floating point implementation is selected.

8.119 Define GFLIB_COS_T

```
#include <GFLIB_Cos.h>
```

8.119.1 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_FLT
```

8.119.2 Description

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_FLT datatype in case the single precision floating point implementation is selected.

8.120 Define GFLIB_COS_T

```
#include <GFLIB_Cos.h>
```

8.120.1 Macro Definition

```
#define GFLIB_COS_T GFLIB_COS_T_FLT
```

8.120.2 Description

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_T as alias for GFLIB_COS_T_FLT datatype in case the single precision floating point implementation is selected.

8.121 Define GFLIB_COS_DEFAULT

```
#include <GFLIB_Cos.h>
```

8.121.1 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_FLT
```

8.121.2 Description

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.122 Define GFLIB_COS_DEFAULT

```
#include <GFLIB_Cos.h>
```

8.122.1 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_FLT
```

8.122.2 Description

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.123 Define GFLIB_COS_DEFAULT

```
#include <GFLIB_Cos.h>
```

8.123.1 Macro Definition

```
#define GFLIB_COS_DEFAULT GFLIB_COS_DEFAULT_FLT
```

8.123.2 Description

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_COS_DEFAULT as alias for GFLIB_COS_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.124 Define GFLIB_COS_DEFAULT_F32

```
#include <GFLIB_Cos.h>
```

8.124.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_F32 &f32gflibCosCoef
```

8.124.2 Description

Default approximation coefficients for GFLIB_Cos_F32 function.

8.125 Define GFLIB_COS_DEFAULT_F16

```
#include <GFLIB_Cos.h>
```

8.125.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_F16 &f16gflibCosCoef
```

8.125.2 Description

Default approximation coefficients for GFLIB_Cos_F32 function.

8.126 Define GFLIB_COS_DEFAULT_FLT

```
#include <GFLIB_Cos.h>
```

8.126.1 Macro Definition

```
#define GFLIB_COS_DEFAULT_FLT &fltgflibCosCoef
```

8.126.2 Description

Default approximation coefficients for GFLIB_Cos_FLT function.

8.127 Define GFLIB_Hyst

```
#include <GFLIB_Hyst.h>
```

8.127.1 Macro Definition

```
#define GFLIB_Hyst macro_dispatcher(GFLIB_Hyst, __VA_ARGS__)(__VA_ARGS__)
```

8.127.2 Description

The function implements the hysteresis functionality.

8.128 Define GFLIB_HYST_T

```
#include <GFLIB_Hyst.h>
```

8.128.1 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_FLT
```

8.128.2 Description

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_FLT datatype in case the single precision floating point implementation is selected.

8.129 Define GFLIB_HYST_T

```
#include <GFLIB_Hyst.h>
```

8.129.1 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_FLT
```

8.129.2 Description

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_FLT datatype in case the single precision floating point implementation is selected.

8.130 Define GFLIB_HYST_T

```
#include <GFLIB_Hyst.h>
```

8.130.1 Macro Definition

```
#define GFLIB_HYST_T GFLIB_HYST_T_FLT
```

8.130.2 Description

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_T as alias for GFLIB_HYST_T_FLT datatype in case the single precision floating point implementation is selected.

8.131 Define GFLIB_HYST_DEFAULT

```
#include <GFLIB_Hyst.h>
```

8.131.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_FLT
```

8.131.2 Description

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.132 Define GFLIB_HYST_DEFAULT

```
#include <GFLIB_Hyst.h>
```

8.132.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_FLT
```

8.132.2 Description

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.133 Define GFLIB_HYST_DEFAULT

```
#include <GFLIB_Hyst.h>
```

8.133.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT GFLIB_HYST_DEFAULT_FLT
```

8.133.2 Description

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_HYST_DEFAULT as alias for GFLIB_HYST_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.134 Define GFLIB_HYST_DEFAULT_F32

```
#include <GFLIB_Hyst.h>
```

8.134.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F32 {0,0,0,0,0}
```

8.134.2 Description

Default value for GFLIB_HYST_T_F32.

8.135 Define GFLIB_HYST_DEFAULT_F16

```
#include <GFLIB_Hyst.h>
```

8.135.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_F16 {0,0,0,0,0}
```

8.135.2 Description

Default value for GFLIB_HYST_T_F16.

8.136 Define GFLIB_HYST_DEFAULT_FLT

```
#include <GFLIB_Hyst.h>
```

8.136.1 Macro Definition

```
#define GFLIB_HYST_DEFAULT_FLT {0,0,0,0,0}
```

8.136.2 Description

Default value for GFLIB_HYST_T_FLT.

8.137 Define GFLIB_IntegratorTR

```
#include <GFLIB_IntegratorTR.h>
```

8.137.1 Macro Definition

```
#define GFLIB_IntegratorTR macro_dispatcher(GFLIB_IntegratorTR, __VA_ARGS__)(__VA_ARGS__)
```

8.137.2 Description

The function calculates a discrete implementation of the integrator (sum), discretized using a trapezoidal (Bilinear) transformation.

8.138 Define GFLIB_INTEGRATOR_TR_T

```
#include <GFLIB_IntegratorTR.h>
```

8.138.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_FLT
```

8.138.2 Description

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_FLT datatype in case the single precision floating point implementation is selected.

8.139 Define GFLIB_INTEGRATOR_TR_T

```
#include <GFLIB_IntegratorTR.h>
```

8.139.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_FLT
```

8.139.2 Description

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_FLT datatype in case the single precision floating point implementation is selected.

8.140 Define GFLIB_INTEGRATOR_TR_T

```
#include <GFLIB_IntegratorTR.h>
```

8.140.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_T GFLIB_INTEGRATOR_TR_T_FLT
```

8.140.2 Description

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_T as alias for GFLIB_INTEGRATOR_TR_T_FLT datatype in case the single precision floating point implementation is selected.

8.141 Define GFLIB_INTEGRATOR_TR_DEFAULT

```
#include <GFLIB_IntegratorTR.h>
```

8.141.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT GFLIB_INTEGRATOR_TR_DEFAULT_FLT
```

8.141.2 Description

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.142 Define GFLIB_INTEGRATOR_TR_DEFAULT

```
#include <GFLIB_IntegratorTR.h>
```

8.142.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT GFLIB_INTEGRATOR_TR_DEFAULT_FLT
```

8.142.2 Description

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.143 Define GFLIB_INTEGRATOR_TR_DEFAULT

```
#include <GFLIB_IntegratorTR.h>
```

8.143.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT GFLIB_INTEGRATOR_TR_DEFAULT_FLT
```

8.143.2 Description

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_INTEGRATOR_TR_DEFAULT as alias for GFLIB_INTEGRATOR_TR_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.144 Define GFLIB_INTEGRATOR_TR_DEFAULT_F32

```
#include <GFLIB_IntegratorTR.h>
```

8.144.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F32 {0,0,0,0}
```

8.144.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_F32.

8.145 Define GFLIB_INTEGRATOR_TR_DEFAULT_F16

```
#include <GFLIB_IntegratorTR.h>
```

8.145.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_F16 {0,0,0,0}
```

8.145.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_F16.

8.146 Define GFLIB_INTEGRATOR_TR_DEFAULT_FLT

```
#include <GFLIB_IntegratorTR.h>
```

8.146.1 Macro Definition

```
#define GFLIB_INTEGRATOR_TR_DEFAULT_FLT {0,0,0}
```

8.146.2 Description

Default value for GFLIB_INTEGRATOR_TR_T_FLT.

8.147 Define GFLIB_Limit

```
#include <GFLIB_Limit.h>
```

8.147.1 Macro Definition

```
#define GFLIB_Limit macro_dispatcher(GFLIB_Limit, __VA_ARGS__)(__VA_ARGS__)
```

8.147.2 Description

This function tests whether the input value is within the upper and lower limits.

8.148 Define GFLIB_LIMIT_T

```
#include <GFLIB_Limit.h>
```

8.148.1 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_FLT
```

8.148.2 Description

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.149 Define GFLIB_LIMIT_T

```
#include <GFLIB_Limit.h>
```

8.149.1 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_FLT
```

8.149.2 Description

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.150 Define GFLIB_LIMIT_T

```
#include <GFLIB_Limit.h>
```

8.150.1 Macro Definition

```
#define GFLIB_LIMIT_T GFLIB_LIMIT_T_FLT
```

8.150.2 Description

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_T as alias for GFLIB_LIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.151 Define GFLIB_LIMIT_DEFAULT

```
#include <GFLIB_Limit.h>
```

8.151.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_FLT
```

8.151.2 Description

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.152 Define GFLIB_LIMIT_DEFAULT

```
#include <GFLIB_Limit.h>
```

8.152.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_FLT
```

8.152.2 Description

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.153 Define GFLIB_LIMIT_DEFAULT

```
#include <GFLIB_Limit.h>
```

8.153.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT GFLIB_LIMIT_DEFAULT_FLT
```

8.153.2 Description

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LIMIT_DEFAULT as alias for GFLIB_LIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.154 Define GFLIB_LIMIT_DEFAULT_F32

```
#include <GFLIB_Limit.h>
```

8.154.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F32 {INT32_MIN,INT32_MAX }
```

8.154.2 Description

Default value for GFLIB_LIMIT_T_F32.

8.155 Define GFLIB_LIMIT_DEFAULT_F16

```
#include <GFLIB_Limit.h>
```

8.155.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_F16 {INT16_MIN,INT16_MAX }
```

8.155.2 Description

Default value for GFLIB_LIMIT_T_F16.

8.156 Define GFLIB_LIMIT_DEFAULT_FLT

```
#include <GFLIB_Limit.h>
```

8.156.1 Macro Definition

```
#define GFLIB_LIMIT_DEFAULT_FLT {FLOAT_MIN,FLOAT_MAX }
```

8.156.2 Description

Default value for GFLIB_LIMIT_T_FLT.

8.157 Define GFLIB_LowerLimit

```
#include <GFLIB_LowerLimit.h>
```

8.157.1 Macro Definition

```
#define GFLIB_LowerLimit macro_dispatcher(GFLIB_LowerLimit, __VA_ARGS__)(__VA_ARGS__)
```

8.157.2 Description

This function tests whether the input value is above the lower limit.

8.158 Define GFLIB_LOWERLIMIT_T

```
#include <GFLIB_LowerLimit.h>
```

8.158.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_FLT
```

8.158.2 Description

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.159 Define GFLIB_LOWERLIMIT_T

```
#include <GFLIB_LowerLimit.h>
```

8.159.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_FLT
```

8.159.2 Description

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.160 Define GFLIB_LOWERLIMIT_T

```
#include <GFLIB_LowerLimit.h>
```

8.160.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_T GFLIB_LOWERLIMIT_T_FLT
```

8.160.2 Description

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_T as alias for GFLIB_LOWERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.161 Define GFLIB_LOWERLIMIT_DEFAULT

```
#include <GFLIB_LowerLimit.h>
```

8.161.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_FLT
```

8.161.2 Description

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.162 Define GFLIB_LOWERLIMIT_DEFAULT

```
#include <GFLIB_LowerLimit.h>
```

8.162.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_FLT
```

8.162.2 Description

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.163 Define GFLIB_LOWERLIMIT_DEFAULT

```
#include <GFLIB_LowerLimit.h>
```

8.163.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT GFLIB_LOWERLIMIT_DEFAULT_FLT
```

8.163.2 Description

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LOWERLIMIT_DEFAULT as alias for GFLIB_LOWERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.164 Define GFLIB_LOWERLIMIT_DEFAULT_F32

```
#include <GFLIB_LowerLimit.h>
```

8.164.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F32 {INT32_MIN }
```

8.164.2 Description

Default value for GFLIB_LOWERLIMIT_T_F32.

8.165 Define GFLIB_LOWERLIMIT_DEFAULT_F16

```
#include <GFLIB_LowerLimit.h>
```

8.165.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_F16 {INT16_MIN }
```

8.165.2 Description

Default value for GFLIB_LOWERLIMIT_T_F16.

8.166 Define GFLIB_LOWERLIMIT_DEFAULT_FLT

```
#include <GFLIB_LowerLimit.h>
```

8.166.1 Macro Definition

```
#define GFLIB_LOWERLIMIT_DEFAULT_FLT {FLOAT_MIN }
```

8.166.2 Description

Default value for GFLIB_LOWERLIMIT_T_FLT.

8.167 Define GFLIB_Lut1D

```
#include <GFLIB_Lut1D.h>
```

8.167.1 Macro Definition

```
#define GFLIB_Lut1D macro_dispatcher(GFLIB_Lut1D, __VA_ARGS__)(__VA_ARGS__)
```

8.167.2 Description

This function implements the one-dimensional look-up table.

8.168 Define GFLIB_LUT1D_T

```
#include <GFLIB_Lut1D.h>
```

8.168.1 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_FLT
```

8.168.2 Description

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_FLT datatype in case the single precision floating point implementation is selected.

8.169 Define GFLIB_LUT1D_T

```
#include <GFLIB_Lut1D.h>
```

8.169.1 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_FLT
```

8.169.2 Description

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_FLT datatype in case the single precision floating point implementation is selected.

8.170 Define GFLIB_LUT1D_T

```
#include <GFLIB_Lut1D.h>
```

8.170.1 Macro Definition

```
#define GFLIB_LUT1D_T GFLIB_LUT1D_T_FLT
```

8.170.2 Description

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_T as alias for GFLIB_LUT1D_T_FLT datatype in case the single precision floating point implementation is selected.

8.171 Define GFLIB_LUT1D_DEFAULT

```
#include <GFLIB_Lut1D.h>
```

8.171.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_FLT
```

8.171.2 Description

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.172 Define GFLIB_LUT1D_DEFAULT

```
#include <GFLIB_Lut1D.h>
```

8.172.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_FLT
```

8.172.2 Description

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.173 Define GFLIB_LUT1D_DEFAULT

```
#include <GFLIB_Lut1D.h>
```

8.173.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT GFLIB_LUT1D_DEFAULT_FLT
```

8.173.2 Description

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT1D_DEFAULT as alias for GFLIB_LUT1D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.174 Define GFLIB_LUT1D_DEFAULT_F32

```
#include <GFLIB_Lut1D.h>
```

8.174.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F32 {0,0,0}
```

8.174.2 Description

Default value for GFLIB_LUT1D_T_F32.

8.175 Define GFLIB_LUT1D_DEFAULT_F16

```
#include <GFLIB_Lut1D.h>
```

8.175.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_F16 {0,0,0}
```

8.175.2 Description

Default value for GFLIB_LUT1D_T_F16.

8.176 Define GFLIB_LUT1D_DEFAULT_FLT

```
#include <GFLIB_Lut1D.h>
```

8.176.1 Macro Definition

```
#define GFLIB_LUT1D_DEFAULT_FLT {0,0,0}
```

8.176.2 Description

Default value for GFLIB_LUT1D_T_FLT.

8.177 Define GFLIB_Lut2D

```
#include <GFLIB_Lut2D.h>
```

8.177.1 Macro Definition

```
#define GFLIB_Lut2D macro_dispatcher(GFLIB_Lut2D, __VA_ARGS__)(__VA_ARGS__)
```

8.177.2 Description

This function implements the two-dimensional look-up table.

8.178 Define GFLIB_LUT2D_T

```
#include <GFLIB_Lut2D.h>
```

8.178.1 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_FLT
```

8.178.2 Description

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_FLT datatype in case the single precision floating point implementation is selected.

8.179 Define GFLIB_LUT2D_T

```
#include <GFLIB_Lut2D.h>
```

8.179.1 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_FLT
```

8.179.2 Description

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_FLT datatype in case the single precision floating point implementation is selected.

8.180 Define GFLIB_LUT2D_T

```
#include <GFLIB_Lut2D.h>
```

8.180.1 Macro Definition

```
#define GFLIB_LUT2D_T GFLIB_LUT2D_T_FLT
```

8.180.2 Description

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_T as alias for GFLIB_LUT2D_T_FLT datatype in case the single precision floating point implementation is selected.

8.181 Define GFLIB_LUT2D_DEFAULT

```
#include <GFLIB_Lut2D.h>
```

8.181.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_FLT
```

8.181.2 Description

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.182 Define GFLIB_LUT2D_DEFAULT

```
#include <GFLIB_Lut2D.h>
```

8.182.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_FLT
```

8.182.2 Description

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.183 Define GFLIB_LUT2D_DEFAULT

```
#include <GFLIB_Lut2D.h>
```

8.183.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT GFLIB_LUT2D_DEFAULT_FLT
```

8.183.2 Description

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_LUT2D_DEFAULT as alias for GFLIB_LUT2D_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.184 Define GFLIB_LUT2D_DEFAULT_F32

```
#include <GFLIB_Lut2D.h>
```

8.184.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F32 {0,0,0,0,0}
```

8.184.2 Description

Default value for GFLIB_LUT2D_T_F32.

8.185 Define GFLIB_LUT2D_DEFAULT_F16

```
#include <GFLIB_Lut2D.h>
```

8.185.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_F16 {0,0,0,0,0}
```

8.185.2 Description

Default value for GFLIB_LUT2D_T_F16.

8.186 Define GFLIB_LUT2D_DEFAULT_FLT

```
#include <GFLIB_Lut2D.h>
```

8.186.1 Macro Definition

```
#define GFLIB_LUT2D_DEFAULT_FLT {0,0,0,0,0}
```

8.186.2 Description

Default value for GFLIB_LUT2D_T_FLT.

8.187 Define GFLIB_Ramp

```
#include <GFLIB_Ramp.h>
```

8.187.1 Macro Definition

```
#define GFLIB_Ramp macro_dispatcher(GFLIB_Ramp, __VA_ARGS__)(__VA_ARGS__)
```

8.187.2 Description

The function calculates the up/down ramp with the step increment/decrement defined in the pParam structure.

8.188 Define GFLIB_RAMP_T

```
#include <GFLIB_Ramp.h>
```

8.188.1 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_FLT
```

8.188.2 Description

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_FLT datatype in case the single precision floating point implementation is selected.

8.189 Define GFLIB_RAMP_T

```
#include <GFLIB_Ramp.h>
```

8.189.1 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_FLT
```

8.189.2 Description

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_FLT datatype in case the single precision floating point implementation is selected.

8.190 Define GFLIB_RAMP_T

```
#include <GFLIB_Ramp.h>
```

8.190.1 Macro Definition

```
#define GFLIB_RAMP_T GFLIB_RAMP_T_FLT
```

8.190.2 Description

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_T as alias for GFLIB_RAMP_T_FLT datatype in case the single precision floating point implementation is selected.

8.191 Define GFLIB_RAMP_DEFAULT

```
#include <GFLIB_Ramp.h>
```

8.191.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_FLT
```

8.191.2 Description

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.192 Define GFLIB_RAMP_DEFAULT

```
#include <GFLIB_Ramp.h>
```

8.192.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_FLT
```

8.192.2 Description

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.193 Define GFLIB_RAMP_DEFAULT

```
#include <GFLIB_Ramp.h>
```

8.193.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT GFLIB_RAMP_DEFAULT_FLT
```

8.193.2 Description

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_RAMP_DEFAULT as alias for GFLIB_RAMP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.194 Define GFLIB_RAMP_DEFAULT_F32

```
#include <GFLIB_Ramp.h>
```

8.194.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F32 {0,0,0}
```

8.194.2 Description

Default value for GFLIB_RAMP_T_F32.

8.195 Define GFLIB_RAMP_DEFAULT_F16

```
#include <GFLIB_Ramp.h>
```

8.195.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_F16 {0,0,0}
```

8.195.2 Description

Default value for GFLIB_RAMP_T_F16.

8.196 Define GFLIB_RAMP_DEFAULT_FLT

```
#include <GFLIB_Ramp.h>
```

8.196.1 Macro Definition

```
#define GFLIB_RAMP_DEFAULT_FLT {0,0,0}
```

8.196.2 Description

Default value for GFLIB_RAMP_T_FLT.

8.197 Define GFLIB_Sign

```
#include <GFLIB_Sign.h>
```

8.197.1 Macro Definition

```
#define GFLIB_Sign macro_dispatcher(GFLIB_Sign, __VA_ARGS__)(__VA_ARGS__)
```

8.197.2 Description

This function returns the signum of input value.

8.198 Define GFLIB_Sin

```
#include <GFLIB_Sin.h>
```

8.198.1 Macro Definition

```
#define GFLIB_Sin macro_dispatcher(GFLIB_Sin, __VA_ARGS__)(__VA_ARGS__)
```

8.198.2 Description

This function implements polynomial approximation of sine function.

8.199 Define GFLIB_SIN_T

```
#include <GFLIB_Sin.h>
```

8.199.1 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_FLT
```

8.199.2 Description

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_FLT datatype in case the single precision floating point implementation is selected.

8.200 Define GFLIB_SIN_T

```
#include <GFLIB_Sin.h>
```

8.200.1 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_FLT
```

8.200.2 Description

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_FLT datatype in case the single precision floating point implementation is selected.

8.201 Define GFLIB_SIN_T

```
#include <GFLIB_Sin.h>
```

8.201.1 Macro Definition

```
#define GFLIB_SIN_T GFLIB_SIN_T_FLT
```

8.201.2 Description

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_T as alias for GFLIB_SIN_T_FLT datatype in case the single precision floating point implementation is selected.

8.202 Define GFLIB_SIN_DEFAULT

```
#include <GFLIB_Sin.h>
```

8.202.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_FLT
```

8.202.2 Description

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.203 Define GFLIB_SIN_DEFAULT

```
#include <GFLIB_Sin.h>
```

8.203.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_FLT
```

8.203.2 Description

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.204 Define GFLIB_SIN_DEFAULT

```
#include <GFLIB_Sin.h>
```

8.204.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT GFLIB_SIN_DEFAULT_FLT
```

8.204.2 Description

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_SIN_DEFAULT as alias for GFLIB_SIN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.205 Define GFLIB_SIN_DEFAULT_F32

```
#include <GFLIB_Sin.h>
```

8.205.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F32 &f32gflibSinCoef
```

8.205.2 Description

Default approximation coefficients for GFLIB_Sin_F32 function.

8.206 Define GFLIB_SIN_DEFAULT_F16

```
#include <GFLIB_Sin.h>
```

8.206.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_F16 &f16gflibSinCoef
```

8.206.2 Description

Default approximation coefficients for GFLIB_Sin_F16 function.

8.207 Define GFLIB_SIN_DEFAULT_FLT

```
#include <GFLIB_Sin.h>
```

8.207.1 Macro Definition

```
#define GFLIB_SIN_DEFAULT_FLT &fltgflibSinCoef
```

8.207.2 Description

Default approximation coefficients for GFLIB_Sin_FLT function.

8.208 Define GFLIB_Sqrt

```
#include <GFLIB_Sqrt.h>
```

8.208.1 Macro Definition

```
#define GFLIB_Sqrt macro_dispatcher(GFLIB_Sqrt, __VA_ARGS__)(__VA_ARGS__)
```

8.208.2 Description

This function returns the square root of input value.

8.209 Define GFLIB_TAN_LIMIT_FLT

```
#include <GFLIB_Tan.c>
```

8.209.1 Macro Definition

```
#define GFLIB_TAN_LIMIT_FLT ((tFloat)1000)
```

8.209.2 Description

Max output value for the tangent function in single precision floating point implementation.

8.210 Define GFLIB_Tan

```
#include <GFLIB_Tan.h>
```

8.210.1 Macro Definition

```
#define GFLIB_Tan macro_dispatcher(GFLIB_Tan, __VA_ARGS__)(__VA_ARGS__)
```

8.210.2 Description

This function implements polynomial approximation of tangent function.

8.211 Define GFLIB_TAN_T

```
#include <GFLIB_Tan.h>
```

8.211.1 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_FLT
```

8.211.2 Description

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_FLT datatype in case the single precision floating point implementation is selected.

8.212 Define GFLIB_TAN_T

```
#include <GFLIB_Tan.h>
```

8.212.1 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_FLT
```

8.212.2 Description

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_FLT datatype in case the single precision floating point implementation is selected.

8.213 Define GFLIB_TAN_T

```
#include <GFLIB_Tan.h>
```

8.213.1 Macro Definition

```
#define GFLIB_TAN_T GFLIB_TAN_T_FLT
```

8.213.2 Description

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_T as alias for GFLIB_TAN_T_FLT datatype in case the single precision floating point implementation is selected.

8.214 Define GFLIB_TAN_DEFAULT

```
#include <GFLIB_Tan.h>
```

8.214.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_FLT
```

8.214.2 Description

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.215 Define GFLIB_TAN_DEFAULT

```
#include <GFLIB_Tan.h>
```

8.215.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_FLT
```

8.215.2 Description

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.216 Define GFLIB_TAN_DEFAULT

```
#include <GFLIB_Tan.h>
```

8.216.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT GFLIB_TAN_DEFAULT_FLT
```

8.216.2 Description

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_TAN_DEFAULT as alias for GFLIB_TAN_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.217 Define GFLIB_TAN_DEFAULT_F32

```
#include <GFLIB_Tan.h>
```

8.217.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F32 &f32gflibTanCoef
```

8.217.2 Description

Default approximation coefficients for GFLIB_Tan_F32 function.

8.218 Define GFLIB_TAN_DEFAULT_F16

```
#include <GFLIB_Tan.h>
```

8.218.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_F16 &f16gflibTanCoef
```

8.218.2 Description

Default approximation coefficients for GFLIB_Tan_F16 function.

8.219 Define GFLIB_TAN_DEFAULT_FLT

```
#include <GFLIB_Tan.h>
```

8.219.1 Macro Definition

```
#define GFLIB_TAN_DEFAULT_FLT &fltgflibTanCoef
```

8.219.2 Description

Default approximation coefficients for GFLIB_Tan_FLT function.

8.220 Define GFLIB_UpperLimit

```
#include <GFLIB_UpperLimit.h>
```

8.220.1 Macro Definition

```
#define GFLIB_UpperLimit macro_dispatcher(GFLIB_UpperLimit, __VA_ARGS__)(__VA_ARGS__)
```

8.220.2 Description

This function tests whether the input value is below the upper limit.

8.221 Define GFLIB_UPPERLIMIT_T

```
#include <GFLIB_UpperLimit.h>
```

8.221.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_FLT
```

8.221.2 Description

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.222 Define GFLIB_UPPERLIMIT_T

```
#include <GFLIB_UpperLimit.h>
```

8.222.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_FLT
```

8.222.2 Description

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.223 Define GFLIB_UPPERLIMIT_T

```
#include <GFLIB_UpperLimit.h>
```

8.223.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_T GFLIB_UPPERLIMIT_T_FLT
```

8.223.2 Description

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_T as alias for GFLIB_UPPERLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.224 Define GFLIB_UPPERLIMIT_DEFAULT

```
#include <GFLIB_UpperLimit.h>
```

8.224.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_FLT
```

8.224.2 Description

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.225 Define GFLIB_UPPERLIMIT_DEFAULT

```
#include <GFLIB_UpperLimit.h>
```

8.225.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_FLT
```

8.225.2 Description

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.226 Define GFLIB_UPPERLIMIT_DEFAULT

```
#include <GFLIB_UpperLimit.h>
```

8.226.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT GFLIB_UPPERLIMIT_DEFAULT_FLT
```

8.226.2 Description

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_UPPERLIMIT_DEFAULT as alias for GFLIB_UPPERLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.227 Define GFLIB_UPPERLIMIT_DEFAULT_F32

```
#include <GFLIB_UpperLimit.h>
```

8.227.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F32 {INT32_MAX }
```

8.227.2 Description

Default value for GFLIB_UPPERLIMIT_T_F32.

8.228 Define GFLIB_UPPERLIMIT_DEFAULT_F16

```
#include <GFLIB_UpperLimit.h>
```

8.228.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_F16 {INT16_MAX }
```

8.228.2 Description

Default value for GFLIB_UPPERLIMIT_T_F16.

8.229 Define GFLIB_UPPERLIMIT_DEFAULT_FLT

```
#include <GFLIB_UpperLimit.h>
```

8.229.1 Macro Definition

```
#define GFLIB_UPPERLIMIT_DEFAULT_FLT {FLOAT_MAX }
```

8.229.2 Description

Default value for GFLIB_UPPERLIMIT_T_FLT.

8.230 Define GFLIB_VectorLimit

```
#include <GFLIB_VectorLimit.h>
```

8.230.1 Macro Definition

```
#define GFLIB_VectorLimit macro_dispatcher(GFLIB_VectorLimit, __VA_ARGS__)(__VA_ARGS__)
```

8.230.2 Description

This function limits the magnitude of the input vector.

8.231 Define GFLIB_VECTORLIMIT_T

```
#include <GFLIB_VectorLimit.h>
```

8.231.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_FLT
```

8.231.2 Description

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.232 Define GFLIB_VECTORLIMIT_T

```
#include <GFLIB_VectorLimit.h>
```

8.232.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_FLT
```

8.232.2 Description

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.233 Define GFLIB_VECTORLIMIT_T

```
#include <GFLIB_VectorLimit.h>
```

8.233.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_T GFLIB_VECTORLIMIT_T_FLT
```

8.233.2 Description

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_T as alias for GFLIB_VECTORLIMIT_T_FLT datatype in case the single precision floating point implementation is selected.

8.234 Define GFLIB_VECTORLIMIT_DEFAULT

```
#include <GFLIB_VectorLimit.h>
```

8.234.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_FLT
```

8.234.2 Description

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.235 Define GFLIB_VECTORLIMIT_DEFAULT

```
#include <GFLIB_VectorLimit.h>
```

8.235.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_FLT
```

8.235.2 Description

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.236 Define GFLIB_VECTORLIMIT_DEFAULT

```
#include <GFLIB_VectorLimit.h>
```

8.236.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT GFLIB_VECTORLIMIT_DEFAULT_FLT
```

8.236.2 Description

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GFLIB_VECTORLIMIT_DEFAULT as alias for GFLIB_VECTORLIMIT_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.237 Define GFLIB_VECTORLIMIT_DEFAULT_F32

```
#include <GFLIB_VectorLimit.h>
```

8.237.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F32 {0}
```

8.237.2 Description

Default value for GFLIB_VECTORLIMIT_T_F32.

8.238 Define GFLIB_VECTORLIMIT_DEFAULT_F16

```
#include <GFLIB_VectorLimit.h>
```

8.238.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_F16 {0}
```

8.238.2 Description

Default value for GFLIBVECTORLIMIT_T_F16.

8.239 Define GFLIB_VECTORLIMIT_DEFAULT_FLT

```
#include <GFLIB_VectorLimit.h>
```

8.239.1 Macro Definition

```
#define GFLIB_VECTORLIMIT_DEFAULT_FLT {0}
```

8.239.2 Description

Default value for GFLIB_VECTORLIMIT_T_FLT.

8.240 Define GMCLIB_Clark

```
#include <GMCLIB_Clark.h>
```

8.240.1 Macro Definition

```
#define GMCLIB_Clark macro_dispatcher(GMCLIB_Clark, __VA_ARGS__)(__VA_ARGS__)
```

8.240.2 Description

This function implements the Clarke transformation.

8.241 Define GMCLIB_ClarkInv

```
#include <GMCLIB_ClarkInv.h>
```

8.241.1 Macro Definition

```
#define GMCLIB_ClarkInv macro_dispatcher(GMCLIB_ClarkInv, __VA_ARGS__)(__VA_ARGS__)
```

8.241.2 Description

This function implements the inverse Clarke transformation.

8.242 Define GMCLIB_DecouplingPMSM

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.242.1 Macro Definition

```
#define GMCLIB_DecouplingPMSM macro_dispatcher(GMCLIB_DecouplingPMSM, __VA_ARGS__)
(__VA_ARGS__)
```

8.242.2 Description

This function calculates the cross-coupling voltages to eliminate the dq axis coupling causing non-linearity of the field oriented control.

8.243 Define GMCLIB_DECOUPLINGPMSM_T

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.243.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_FLT
```

8.243.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_FLT datatype in case the single precision floating point implementation is selected.

8.244 Define GMCLIB_DECOUPLINGPMSM_T

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.244.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_FLT
```

8.244.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_FLT datatype in case the single precision floating point implementation is selected.

8.245 Define GMCLIB_DECOUPLINGPMSM_T

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.245.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_T GMCLIB_DECOUPLINGPMSM_T_FLT
```

8.245.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_T as alias for GMCLIB_DECOUPLINGPMSM_T_FLT datatype in case the single precision floating point implementation is selected.

8.246 Define GMCLIB_DECOUPLINGPMSM_DEFAULT

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.246.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT
```

8.246.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.247 Define GMCLIB_DECOUPLINGPMSM_DEFAULT

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.247.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT
```

8.247.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.248 Define GMCLIB_DECOUPLINGPMSM_DEFAULT

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.248.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT
```

8.248.2 Description

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_DECOUPLINGPMSM_DEFAULT as alias for GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.249 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.249.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F32 {0,0,0,0}
```

8.249.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_F32.

8.250 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.250.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_F16 {0,0,0,0}
```

8.250.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_F16.

8.251 Define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT

```
#include <GMCLIB_DecouplingPMSM.h>
```

8.251.1 Macro Definition

```
#define GMCLIB_DECOUPLINGPMSM_DEFAULT_FLT {0,0}
```

8.251.2 Description

Default value for GMCLIB_DECOUPLINGPMSM_T_FLT.

8.252 Define GMCLIB_ElimDcBusRip

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.252.1 Macro Definition

```
#define GMCLIB_ElimDcBusRip macro_dispatcher(GMCLIB_ElimDcBusRip, __VA_ARGS__)(__VA_ARGS__)
```

8.252.2 Description

This function implements the DC Bus voltage ripple elimination.

8.253 Define GMCLIB_ELIMDCBUSRIP_T

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.253.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_FLT
```

8.253.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_FLT datatype in case the single precision floating point implementation is selected.

8.254 Define GMCLIB_ELIMDCBUSRIP_T

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.254.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_FLT
```

8.254.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_FLT datatype in case the single precision floating point implementation is selected.

8.255 Define GMCLIB_ELIMDCBUSRIP_T

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.255.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_T GMCLIB_ELIMDCBUSRIP_T_FLT
```

8.255.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F32 datatype in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_F16 datatype in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_T as alias for GMCLIB_ELIMDCBUSRIP_T_FLT datatype in case the single precision floating point implementation is selected.

8.256 Define GMCLIB_ELIMDCBUSRIP_DEFAULT

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.256.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT
```

8.256.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.257 Define GMCLIB_ELIMDCBUSRIP_DEFAULT

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.257.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT
```

8.257.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.258 Define GMCLIB_ELIMDCBUSRIP_DEFAULT

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.258.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT
```

8.258.2 Description

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 default value in case the 32-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 default value in case the 16-bit fractional implementation is selected.

Definition of GMCLIB_ELIMDCBUSRIP_DEFAULT as alias for GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT default value in case the single precision floating point implementation is selected.

8.259 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.259.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F32 {0,0}
```

8.259.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_F32.

8.260 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.260.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_F16 {0,0}
```

8.260.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_F16.

8.261 Define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT

```
#include <GMCLIB_ElimDcBusRip.h>
```

8.261.1 Macro Definition

```
#define GMCLIB_ELIMDCBUSRIP_DEFAULT_FLT {0,0}
```

8.261.2 Description

Default value for GMCLIB_ELIMDCBUSRIP_T_FLT.

8.262 Define GMCLIB_Park

```
#include <GMCLIB_Park.h>
```

8.262.1 Macro Definition

```
#define GMCLIB_Park macro_dispatcher(GMCLIB_Park, __VA_ARGS__)(__VA_ARGS__)
```

8.262.2 Description

This function implements the calculates Park transformation.

8.263 Define GMCLIB_ParkInv

```
#include <GMCLIB_ParkInv.h>
```

8.263.1 Macro Definition

```
#define GMCLIB_ParkInv macro_dispatcher(GMCLIB_ParkInv, __VA_ARGS__)(__VA_ARGS__)
```

8.263.2 Description

This function implements the inverse Park transformation.

8.264 Define GMCLIB_SvmStd

```
#include <GMCLIB_SvmStd.h>
```

8.264.1 Macro Definition

```
#define GMCLIB_SvmStd macro_dispatcher(GMCLIB_SvmStd, __VA_ARGS__)(__VA_ARGS__)
```

8.264.2 Description

This function calculates the duty-cycle ratios using the Standard Space Vector Modulation technique.

8.265 Define MLIB_Abs

```
#include <MLIB_Abs.h>
```

8.265.1 Macro Definition

```
#define MLIB_Abs macro_dispatcher(MLIB_Abs, __VA_ARGS__)(__VA_ARGS__)
```

8.265.2 Description

This function returns absolute value of input parameter.

8.266 Define MLIB_AbsSat

```
#include <MLIB_AbsSat.h>
```

8.266.1 Macro Definition

```
#define MLIB_AbsSat macro_dispatcher(MLIB_AbsSat, __VA_ARGS__)(__VA_ARGS__)
```

8.266.2 Description

This function returns absolute value of input parameter and saturate if necessary.

8.267 Define MLIB_Add

```
#include <MLIB_Add.h>
```

8.267.1 Macro Definition

```
#define MLIB_Add macro_dispatcher(MLIB_Add, __VA_ARGS__)(__VA_ARGS__)
```

8.267.2 Description

This function returns sum of two input parameters.

8.268 Define MLIB_AddSat

```
#include <MLIB_AddSat.h>
```

8.268.1 Macro Definition

```
#define MLIB_AddSat macro_dispatcher(MLIB_AddSat, __VA_ARGS__)(__VA_ARGS__)
```

8.268.2 Description

This function returns sum of two input parameters and saturate if necessary.

8.269 Define MLIB_Convert

```
#include <MLIB_Convert.h>
```

8.269.1 Macro Definition

```
#define MLIB_Convert macro_dispatcher(MLIB_Convert, __VA_ARGS__)(__VA_ARGS__)
```

8.269.2 Description

This function converts the input value to different representation.

8.270 Define MLIB_ConvertPU

```
#include <MLIB_ConvertPU.h>
```

8.270.1 Macro Definition

```
#define MLIB_ConvertPU macro_dispatcher(MLIB_ConvertPU, __VA_ARGS__)(__VA_ARGS__)
```

8.270.2 Description

This function converts the input value to different representation with scale.

8.271 Define MLIB_Div

```
#include <MLIB_Div.h>
```

8.271.1 Macro Definition

```
#define MLIB_Div macro_dispatcher(MLIB_Div, __VA_ARGS__)(__VA_ARGS__)
```

8.271.2 Description

This function divides the first parameter by the second one.

8.272 Define MLIB_DivSat

```
#include <MLIB_DivSat.h>
```

8.272.1 Macro Definition

```
#define MLIB_DivSat macro_dispatcher(MLIB_DivSat, __VA_ARGS__)(__VA_ARGS__)
```

8.272.2 Description

This function divides the first parameter by the second one as saturate.

8.273 Define MLIB_Mac

```
#include <MLIB_Mac.h>
```

8.273.1 Macro Definition

```
#define MLIB_Mac macro_dispatcher(MLIB_Mac, __VA_ARGS__)(__VA_ARGS__)
```

8.273.2 Description

This function implements the multiply accumulate function.

8.274 Define MLIB_MacSat

```
#include <MLIB_MacSat.h>
```

8.274.1 Macro Definition

```
#define MLIB_MacSat macro_dispatcher(MLIB_MacSat, __VA_ARGS__)(__VA_ARGS__)
```

8.274.2 Description

This function implements the multiply accumulate function saturated if necessary.

8.275 Define MLIB_Mul

```
#include <MLIB_Mul.h>
```

8.275.1 Macro Definition

```
#define MLIB_Mul macro_dispatcher(MLIB_Mul, __VA_ARGS__)(__VA_ARGS__)
```

8.275.2 Description

This function multiply two input parameters.

8.276 Define MLIB_MulSat

```
#include <MLIB_MulSat.h>
```

8.276.1 Macro Definition

```
#define MLIB_MulSat macro_dispatcher(MLIB_MulSat, __VA_ARGS__)(__VA_ARGS__)
```

8.276.2 Description

This function multiply two input parameters and saturate if necessary.

8.277 Define MLIB_Neg

```
#include <MLIB_Neg.h>
```

8.277.1 Macro Definition

```
#define MLIB_Neg macro_dispatcher(MLIB_Neg, __VA_ARGS__)(__VA_ARGS__)
```

8.277.2 Description

This function returns negative value of input parameter.

8.278 Define MLIB_NegSat

```
#include <MLIB_NegSat.h>
```

8.278.1 Macro Definition

```
#define MLIB_NegSat macro_dispatcher(MLIB_NegSat, __VA_ARGS__)(__VA_ARGS__)
```

8.278.2 Description

This function returns negative value of input parameter and saturate if necessary.

8.279 Define MLIB_Norm

```
#include <MLIB_Norm.h>
```

8.279.1 Macro Definition

```
#define MLIB_Norm macro_dispatcher(MLIB_Norm, __VA_ARGS__)(__VA_ARGS__)
```

8.279.2 Description

This function returns the number of left shifts needed to normalize the input parameter.

8.280 Define MLIB_Round

```
#include <MLIB_Round.h>
```

8.280.1 Macro Definition

```
#define MLIB_Round macro_dispatcher(MLIB_Round, __VA_ARGS__)(__VA_ARGS__)
```

8.280.2 Description

This function rounds the first input value for number of digits defined by second parameter and saturate automatically.

8.281 Define MLIB_ShBi

```
#include <MLIB_ShBi.h>
```

8.281.1 Macro Definition

```
#define MLIB_ShBi macro_dispatcher(MLIB_ShBi, __VA_ARGS__)(__VA_ARGS__)
```

8.281.2 Description

Based on sign of second parameter this function shifts the first parameter to right or left. If the sign of second parameter is negative, shift to right.

8.282 Define MLIB_ShBiSat

```
#include <MLIB_ShBiSat.h>
```

8.282.1 Macro Definition

```
#define MLIB_ShBiSat macro_dispatcher(MLIB_ShBiSat, __VA_ARGS__)(__VA_ARGS__)
```

8.282.2 Description

Based on sign of second parameter this function shifts the first parameter to right or left and saturate if necessary. If the sign of second parameter is negative, shift to right.

8.283 Define MLIB_ShL

```
#include <MLIB_ShL.h>
```

8.283.1 Macro Definition

```
#define MLIB_ShL macro_dispatcher(MLIB_ShL, __VA_ARGS__)(__VA_ARGS__)
```

8.283.2 Description

This function shifts the first parameter to left by number defined by second parameter.

8.284 Define MLIB_ShLSat

```
#include <MLIB_ShLSat.h>
```

8.284.1 Macro Definition

```
#define MLIB_ShLSat macro_dispatcher(MLIB_ShLSat, __VA_ARGS__)(__VA_ARGS__)
```

8.284.2 Description

This function shifts the first parameter to left by number defined by second parameter and saturate if necessary.

8.285 Define MLIB_ShR

```
#include <MLIB_ShR.h>
```

8.285.1 Macro Definition

```
#define MLIB_ShR macro_dispatcher(MLIB_ShR, __VA_ARGS__)(__VA_ARGS__)
```

8.285.2 Description

This function shifts the first parameter to right by number defined by second parameter.

8.286 Define MLIB_Sub

```
#include <MLIB_Sub.h>
```

8.286.1 Macro Definition

```
#define MLIB_Sub macro_dispatcher(MLIB_Sub, __VA_ARGS__)(__VA_ARGS__)
```

8.286.2 Description

This function subtracts the second parameter from the first one.

8.287 Define MLIB_SubSat

```
#include <MLIB_SubSat.h>
```

8.287.1 Macro Definition

```
#define MLIB_SubSat macro_dispatcher(MLIB_SubSat, __VA_ARGS__)(__VA_ARGS__)
```

8.287.2 Description

This function subtracts the second parameter from the first one and saturate if necessary.

8.288 Define MLIB_VMac

```
#include <MLIB_VMac.h>
```

8.288.1 Macro Definition

```
#define MLIB_VMac macro_dispatcher(MLIB_VMac, __VA_ARGS__)(__VA_ARGS__)
```

8.288.2 Description

This function implements the vector multiply accumulate function.

8.289 Define MCLIB_VERSION

```
#include <SWLIBS_Config.h>
```

8.289.1 Macro Definition

```
#define MCLIB_VERSION {0,95,0}
```

8.289.2 Description

8.290 Define MCLIB_STD_ON

```
#include <SWLIBS_Config.h>
```

8.290.1 Macro Definition

```
#define MCLIB_STD_ON 0x01
```

8.290.2 Description

8.291 Define MCLIB_STD_OFF

```
#include <SWLIBS_Config.h>
```

8.291.1 Macro Definition

```
#define MCLIB_STD_OFF 0x00
```

8.291.2 Description

8.292 Define F32

```
#include <SWLIBS_Config.h>
```

8.292.1 Macro Definition

```
#define F32 Define F32
```

8.292.2 Description

8.293 Define F16

```
#include <SWLIBS_Config.h>
```

8.293.1 Macro Definition

```
#define F16 Define F16
```

8.293.2 Description

8.294 Define FLT

```
#include <SWLIBS_Config.h>
```

8.294.1 Macro Definition

```
#define FLT Define FLT
```

8.294.2 Description

8.295 Define MCLIB_DEFAULT_IMPLEMENTATION_F32

```
#include <SWLIBS_Config.h>
```

8.295.1 Macro Definition

```
#define MCLIB_DEFAULT_IMPLEMENTATION_F32 1
```

8.295.2 Description

8.296 Define MCLIB_DEFAULT_IMPLEMENTATION_F16

```
#include <SWLIBS_Config.h>
```

8.296.1 Macro Definition

```
#define MCLIB_DEFAULT_IMPLEMENTATION_F16 2
```

8.296.2 Description

8.297 Define MCLIB_DEFAULT_IMPLEMENTATION_FLT

```
#include <SWLIBS_Config.h>
```

8.297.1 Macro Definition

```
#define MCLIB_DEFAULT_IMPLEMENTATION_FLT 3
```

8.297.2 Description

8.298 Define MCLIB_SUPPORT_F32

```
#include <SWLIBS_Config.h>
```

8.298.1 Macro Definition

```
#define MCLIB_SUPPORT_F32 Define MCLIB_STD_ON
```

8.298.2 Description

Enables/disables support of 32-bit fractional implementation.

8.299 Define MCLIB_SUPPORT_F16

```
#include <SWLIBS_Config.h>
```

8.299.1 Macro Definition

```
#define MCLIB_SUPPORT_F16 Define MCLIB_STD_ON
```

8.299.2 Description

Enables/disables support of 16-bit fractional implementation.

8.300 Define MCLIB_SUPPORT_FLT

```
#include <SWLIBS_Config.h>
```

8.300.1 Macro Definition

```
#define MCLIB_SUPPORT_FLT Define MCLIB_STD_ON
```

8.300.2 Description

Enables/disables support of single precision floating point implementation.

8.301 Define MCLIB_SUPPORTED_IMPLEMENTATION

```
#include <SWLIBS_Config.h>
```

8.301.1 Macro Definition

```
#define MCLIB_SUPPORTED_IMPLEMENTATION {Define MCLIB_SUPPORT_F32,\ Define MCLIB_SUPPORT_F16,\
Define MCLIB_SUPPORT_FLT, \ 0,0,0,0,0}
```

8.301.2 Description

Array of supported implementations

8.302 Define MCLIB_DEFAULT_IMPLEMENTATION

```
#include <SWLIBS_Config.h>
```

8.302.1 Macro Definition

```
#define MCLIB_DEFAULT_IMPLEMENTATION Define MCLIB_DEFAULT_IMPLEMENTATION_F32
```

8.302.2 Description

Selection of default implementation.

8.303 Define inline

```
#include <SWLIBS_Defines.h>
```

8.303.1 Macro Definition

```
#define inline
```

8.303.2 Description

Definition of inline directive for all supported compilers.

8.304 Define SFRACT_MIN

```
#include <SWLIBS_Defines.h>
```

8.304.1 Macro Definition

```
#define SFRACT_MIN (-1.0)
```

8.304.2 Description

Constant representing the maximal negative value of a signed 16-bit fixed point fractional number, equal to -1.0.

8.305 Define SFRACT_MAX

```
#include <SWLIBS_Defines.h>
```

8.305.1 Macro Definition

```
#define SFRACT_MAX (0.999969482421875)
```

8.305.2 Description

Constant representing the maximal positive value of a signed 16-bit fixed point fractional number, equal to 0.999969482421875.

8.306 Define FRACT_MIN

```
#include <SWLIBS_Defines.h>
```

8.306.1 Macro Definition

```
#define FRACT_MIN (-1.0)
```

8.306.2 Description

Constant representing the maximal negative value of signed 32-bit fixed point fractional number, equal to -1.0.

8.307 Define FRACT_MAX

```
#include <SWLIBS_Defines.h>
```

8.307.1 Macro Definition

```
#define FRACT_MAX (0.99999999953433871269226074218750)
```

8.307.2 Description

Constant representing the maximal positive value of a signed 32-bit fixed point fractional number, equal to 0.99999999953433871269226074218750.

8.308 Define FRAC32_0_5

```
#include <SWLIBS_Defines.h>
```

8.308.1 Macro Definition

```
#define FRAC32_0_5 ((tFrac32) 0x40000000)
```

8.308.2 Description

Value 0.5 in 32-bit fixed point fractional format.

8.309 Define FRAC16_0_5

```
#include <SWLIBS_Defines.h>
```

8.309.1 Macro Definition

```
#define FRAC16_0_5 ((tFrac16) 0x4000)
```

8.309.2 Description

Value 0.5 in 16-bit fixed point fractional format.

8.310 Define FRAC32_0_25

```
#include <SWLIBS_Defines.h>
```

8.310.1 Macro Definition

```
#define FRAC32_0_25 ((tFrac32) 0x20000000)
```

8.310.2 Description

Value 0.25 in 32-bit fixed point fractional format.

8.311 Define FRAC16_0_25

```
#include <SWLIBS_Defines.h>
```

8.311.1 Macro Definition

```
#define FRAC16_0_25 ((tFrac16) 0x2000)
```

8.311.2 Description

Value 0.25 in 16-bit fixed point fractional format.

8.312 Define INT16_MAX

```
#include <SWLIBS_Defines.h>
```

8.312.1 Macro Definition

```
#define INT16_MAX ((tS16) 0x7fff)
```

8.312.2 Description

Constant representing the maximal positive value of a signed 16-bit fixed point integer number, equal to $2^{15}-1 = 0x7fff$.

8.313 Define INT16_MIN

```
#include <SWLIBS_Defines.h>
```

8.313.1 Macro Definition

```
#define INT16_MIN ((tS16) 0x8000)
```

8.313.2 Description

Constant representing the maximal negative value of a signed 16-bit fixed point integer number, equal to 2^{15} = 0x8000.

8.314 Define INT32_MAX

```
#include <SWLIBS_Defines.h>
```

8.314.1 Macro Definition

```
#define INT32_MAX ((tS32) 0x7fffffff)
```

8.314.2 Description

Constant representing the maximal positive value of a signed 32-bit fixed point integer number, equal to 2^{31}-1 = 0x7fff ffff.

8.315 Define INT32_MIN

```
#include <SWLIBS_Defines.h>
```

8.315.1 Macro Definition

```
#define INT32_MIN ((tS32) 0x80000000)
```

8.315.2 Description

Constant representing the maximal negative value of a signed 32-bit fixed point integer number, equal to 2^{31} = 0x8000 0000.

8.316 Define FLOAT_MIN

```
#include <SWLIBS_Defines.h>
```

8.316.1 Macro Definition

```
#define FLOAT_MIN ((tFloat)(-3.40282346638528860e+38))
```

8.316.2 Description

Constant representing the maximal negative value of the 32-bit float type.

8.317 Define FLOAT_MAX

```
#include <SWLIBS_Defines.h>
```

8.317.1 Macro Definition

```
#define FLOAT_MAX ((tFloat)(3.40282346638528860e+38))
```

8.317.2 Description

Constant representing the maximal positive value of the 32-bit float type.

8.318 Define INT16TOINT32

```
#include <SWLIBS_Defines.h>
```

8.318.1 Macro Definition

```
#define INT16TOINT32 ((tS32) (x))
```

8.318.2 Description

Type casting - signed 16-bit integer value cast to a signed 32-bit integer.

8.319 Define INT32TOINT16

```
#include <SWLIBS_Defines.h>
```

8.319.1 Macro Definition

```
#define INT32TOINT16 ((tS16) (x))
```

8.319.2 Description

Type casting - signed 32-bit integer value cast to a signed 16-bit integer.

8.320 Define INT32TOINT64

```
#include <SWLIBS_Defines.h>
```

8.320.1 Macro Definition

```
#define INT32TOINT64 ((tS64) (x))
```

8.320.2 Description

Type casting - signed 32-bit integer value cast to a signed 64-bit integer.

8.321 Define INT64TOINT32

```
#include <SWLIBS_Defines.h>
```

8.321.1 Macro Definition

```
#define INT64TOINT32 ((tS32) (x))
```

8.321.2 Description

Type casting - signed 64-bit integer value cast to a signed 32-bit integer.

8.322 Define F16TOINT16

```
#include <SWLIBS_Defines.h>
```

8.322.1 Macro Definition

```
#define F16TOINT16 ((tS16) (x))
```

8.322.2 Description

Type casting - signed 16-bit fractional value cast to a signed 16-bit integer.

8.323 Define F32TOINT16

```
#include <SWLIBS_Defines.h>
```

8.323.1 Macro Definition

```
#define F32TOINT16 ((tS16) (x))
```

8.323.2 Description

Type casting - lower 16 bits of a signed 32-bit fractional value cast to a signed 16-bit integer.

8.324 Define F64TOINT16

```
#include <SWLIBS_Defines.h>
```

8.324.1 Macro Definition

```
#define F64TOINT16 ((tS16) (x))
```

8.324.2 Description

Type casting - lower 16 bits of a signed 64-bit fractional value cast to a signed 16-bit integer.

8.325 Define F16TOINT32

```
#include <SWLIBS_Defines.h>
```

8.325.1 Macro Definition

```
#define F16TOINT32 ((tS32) (x))
```

8.325.2 Description

Type casting - a signed 16-bit fractional value cast to a signed 32-bit integer, the value placed at the lower 16-bits of the 32-bit result.

8.326 Define F32TOINT32

```
#include <SWLIBS_Defines.h>
```

8.326.1 Macro Definition

```
#define F32TOINT32 ((tS32) (x))
```

8.326.2 Description

Type casting - signed 32-bit fractional value cast to a signed 32-bit integer.

8.327 Define F64TOINT32

```
#include <SWLIBS_Defines.h>
```

8.327.1 Macro Definition

```
#define F64TOINT32 ((tS32) (x))
```

8.327.2 Description

Type casting - lower 32 bits of a signed 64-bit fractional value cast to a signed 32-bit integer.

8.328 Define F16TOINT64

```
#include <SWLIBS_Defines.h>
```

8.328.1 Macro Definition

```
#define F16TOINT64 ((tS64) (x))
```

8.328.2 Description

Type casting - signed 16-bit fractional value cast to a signed 64-bit integer, the value placed at the lower 16-bits of the 64-bit result.

8.329 Define F32TOINT64

```
#include <SWLIBS_Defines.h>
```

8.329.1 Macro Definition

```
#define F32TOINT64 ((tS64) (x))
```

8.329.2 Description

Type casting - signed 32-bit fractional value cast to a signed 64-bit integer, the value placed at the lower 32-bits of the 64-bit result.

8.330 Define F64TOINT64

```
#include <SWLIBS_Defines.h>
```

8.330.1 Macro Definition

```
#define F64TOINT64 ((tS64) (x))
```

8.330.2 Description

Type casting - signed 64-bit fractional value cast to a signed 64-bit integer.

8.331 Define INT16TOF16

```
#include <SWLIBS_Defines.h>
```

8.331.1 Macro Definition

```
#define INT16TOF16 ((tFrac16) (x))
```

8.331.2 Description

Type casting - signed 16-bit integer value cast to a signed 16-bit fractional.

8.332 Define INT16TOF32

```
#include <SWLIBS_Defines.h>
```

8.332.1 Macro Definition

```
#define INT16TOF32 ((tFrac32) (x))
```

8.332.2 Description

Type casting - signed 16-bit integer value cast to a signed 32-bit fractional, the value placed at the lower 16 bits of the 32-bit result.

8.333 Define INT32TOF16

```
#include <SWLIBS_Defines.h>
```

8.333.1 Macro Definition

```
#define INT32TOF16 ((tFrac16) (x))
```

8.333.2 Description

Type casting - lower 16-bits of a signed 32-bit integer value cast to a signed 16-bit fractional.

8.334 Define INT32TOF32

```
#include <SWLIBS_Defines.h>
```

8.334.1 Macro Definition

```
#define INT32TOF32 ((tFrac32) (x))
```

8.334.2 Description

Type casting - signed 32-bit integer value cast to a signed 32-bit fractional.

8.335 Define INT64TOF16

```
#include <SWLIBS_Defines.h>
```

8.335.1 Macro Definition

```
#define INT64TOF16 ((tFrac16) (x))
```

8.335.2 Description

Type casting - lower 16-bits of a signed 64-bit integer value cast to a signed 16-bit fractional.

8.336 Define INT64TOF32

```
#include <SWLIBS_Defines.h>
```

8.336.1 Macro Definition

```
#define INT64TOF32 ((tFrac32) (x))
```

8.336.2 Description

Type casting - lower 32-bits of a signed 64-bit integer value cast to a signed 32-bit fractional.

8.337 Define F16_1_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

8.337.1 Macro Definition

```
#define F16_1_DIVBY_SQRT3 ((tFrac16) 0x49E7)
```

8.337.2 Description

One over sqrt(3) with a 16-bit result, the result rounded for a better precision, i.e. round(1/ $\sqrt{(3)}$*2 ^{15}).

8.338 Define F32_1_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

8.338.1 Macro Definition

```
#define F32_1_DIVBY_SQRT3 ((tFrac32) 0x49E69D16)
```

8.338.2 Description

One over sqrt(3) with a 32-bit result, the result rounded for a better precision, i.e. round(1/ $\sqrt{(3)}$*2 ^{31}).

8.339 Define F16_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

8.339.1 Macro Definition

```
#define F16_SQRT3_DIVBY_2 ((tFrac16) 0x6EDA)
```

8.339.2 Description

Sqrt(3) divided by two with a 16-bit result, the result rounded for a better precision, i.e. round(√(3)/2*2 ^{15}).

8.340 Define F32_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

8.340.1 Macro Definition

```
#define F32_SQRT3_DIVBY_2 ((tFrac32) 0x6ED9EBA1)
```

8.340.2 Description

Sqrt(3) divided by two with a 32-bit result, the result rounded for a better precision, i.e. round(√(3)/2*2 ^{31}).

8.341 Define F16_SQRT2_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

8.341.1 Macro Definition

```
#define F16_SQRT2_DIVBY_2 ((tFrac16) 0x5A82)
```

8.341.2 Description

Sqrt(2) divided by two with a 16-bit result, the result rounded for a better precision, i.e. round(√(2)/2*2 ^{15}).

8.342 Define F32_SQRT2_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

8.342.1 Macro Definition

```
#define F32_SQRT2_DIVBY_2 ((tFrac32) 0x5A82799A)
```

8.342.2 Description

Sqrt(2) divided by two with a 32-bit result, the result rounded for a better precision, i.e. round($\sqrt{(2)}/2*2^{31}$).

8.343 Define FRAC16

```
#include <SWLIBS_Defines.h>
```

8.343.1 Macro Definition

```
#define FRAC16 ((tFrac16) ((x) < (SFRACT_MAX) ? ((x) >= SFRACT_MIN ? (x)*0x8000 :
INT16_MIN) : INT16_MAX))
```

8.343.2 Description

Macro converting a signed fractional [-1,1) number into a 16-bit fixed point number in format Q1.15.

8.344 Define FRAC32

```
#include <SWLIBS_Defines.h>
```

8.344.1 Macro Definition

```
#define FRAC32 ((tFrac32) ((x) < (FRACT_MAX) ? ((x) >= FRACT_MIN ? (x)*0x80000000 :
INT32_MIN) : INT32_MAX))
```

8.344.2 Description

Macro converting a signed fractional [-1,1) number into a 32-bit fixed point number in format Q1.31.

8.345 Define FLOAT_DIVBY_SQRT3

```
#include <SWLIBS_Defines.h>
```

8.345.1 Macro Definition

```
#define FLOAT_DIVBY_SQRT3 ((tFloat) 0.5773502691896258)
```

8.345.2 Description

One over sqrt(3) in single precision floating point format.

8.346 Define FLOAT_SQRT3_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

8.346.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_2 ((tFloat) 0.866025403784439)
```

8.346.2 Description

Sqrt(3) divided by two in single precision floating point format.

8.347 Define FLOAT_SQRT3_DIVBY_4

```
#include <SWLIBS_Defines.h>
```

8.347.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4 ((tFloat) 0.4330127018922190)
```

8.347.2 Description

Sqrt(3) divided by four in single precision floating point format.

8.348 Define FLOAT_SQRT3_DIVBY_4_CORRECTION

```
#include <SWLIBS_Defines.h>
```

8.348.1 Macro Definition

```
#define FLOAT_SQRT3_DIVBY_4_CORRECTION ((tFloat)0)
```

8.348.2 Description

Sqrt(3) divided by four correction constant.

8.349 Define FLOAT_2_PI

```
#include <SWLIBS_Defines.h>
```

8.349.1 Macro Definition

```
#define FLOAT_2_PI ((tFloat) 6.28318530717958)
```

8.349.2 Description

2* π in single precision floating point format.

8.350 Define FLOAT_PI

```
#include <SWLIBS_Defines.h>
```

8.350.1 Macro Definition

```
#define FLOAT_PI ((tFloat) 3.14159265358979)
```

8.350.2 Description

π in single precision floating point format.

8.351 Define FLOAT_PI_DIVBY_2

```
#include <SWLIBS_Defines.h>
```

8.351.1 Macro Definition

```
#define FLOAT_PI_DIVBY_2 ((tFloat) 1.57079632679490)
```

8.351.2 Description

π/2 in single precision floating point format.

8.352 Define FLOAT_PI_SINGLE_CORRECTION

```
#include <SWLIBS_Defines.h>
```

8.352.1 Macro Definition

```
#define FLOAT_PI_SINGLE_CORRECTION ((tFloat)4.37102068E-8)
```

8.352.2 Description

Double to single precision correction constant for π, equal to (π(Double) - π(Single)).

8.353 Define FLOAT_PI_CORRECTION

```
#include <SWLIBS_Defines.h>
```

8.353.1 Macro Definition

```
#define FLOAT_PI_CORRECTION ((tFloat)8.74204136E-8)
```

8.353.2 Description

Double to single precision correction constant for π, equal to (2*(π(Double) - π(Single)).

8.354 Define FLOAT_PI_DIVBY_4

```
#include <SWLIBS_Defines.h>
```

8.354.1 Macro Definition

```
#define FLOAT_PI_DIVBY_4 ((tFloat) 0.7853981633974480)
```

8.354.2 Description

π/4 in single precision floating point format.

8.355 Define FLOAT_4_DIVBY_PI

```
#include <SWLIBS_Defines.h>
```

8.355.1 Macro Definition

```
#define FLOAT_4_DIVBY_PI ((tFloat) 1.2732395447351600)
```

8.355.2 Description

Number four divided by π in single precision floating point format.

8.356 Define FLOAT_0_5

```
#include <SWLIBS_Defines.h>
```

8.356.1 Macro Definition

```
#define FLOAT_0_5 ((tFloat) 0.5)
```

8.356.2 Description

Value 0.5 in single precision floating point format.

8.357 Define FLOAT_PLUS_1

```
#include <SWLIBS_Defines.h>
```

8.357.1 Macro Definition

```
#define FLOAT_PLUS_1 ((tFloat) 1)
```

8.357.2 Description

Value 1 in single precision floating point format.

8.358 Define FLOAT_MINUS_1

```
#include <SWLIBS_Defines.h>
```

8.358.1 Macro Definition

```
#define FLOAT_MINUS_1 ((tFloat) -1)
```

8.358.2 Description

Value -1 in single precision floating point format.

8.359 Define MCLIB_VERSION_DEFAULT

```
#include <SWLIBS_Version.c>
```

8.359.1 Macro Definition

```
#define MCLIB_VERSION_DEFAULT {MCLIB_ID, Define MCLIB_VERSION, Define
MCLIB_SUPPORTED_IMPLEMENTATION, Define MCLIB_DEFAULT_IMPLEMENTATION }
```

8.359.2 Description

8.360 Define MCLIB_MCID_SIZE

```
#include <SWLIBS_Version.h>
```

8.360.1 Macro Definition

```
#define MCLIB_MCID_SIZE 4
```

8.360.2 Description

8.361 Define MCLIB_MCVERSION_SIZE

```
#include <SWLIBS_Version.h>
```

8.361.1 Macro Definition

```
#define MCLIB_MCVERSION_SIZE 3
```

8.361.2 Description

8.362 Define MCLIB_MCIMPLEMENTATION_SIZE

```
#include <SWLIBS_Version.h>
```

8.362.1 Macro Definition

```
#define MCLIB_MCIMPLEMENTATION_SIZE 8
```

8.362.2 Description

8.363 Define MCLIB_ID

```
#include <SWLIBS_Version.h>
```

8.363.1 Macro Definition

```
#define MCLIB_ID {90,71,77,68}
```

8.363.2 Description

Library identification string.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: PXR40XXMCLUG
Rev. 1.0