Freescale Semiconductor
Application Note

Document Number:AN4537
Rev. 1, 10/2012

3-Phase PMSM Vector Control Using the PXS20 and Tower System

by: Libor Prokop and Marek Stulrajter
Roznov pod Radhostem
Czech Republic

Contents

1 Introduction

This application note deals with the design of a 3-phase Permanent Magnet Synchronous Motor (PMSM) vector control drive with a 3-shunt current sensing with a position sensor. It describes a modular concept that allows a single or possibly dual PMSM configuration. The design is targeted at industrial applications. This cost-effective solution benefits from the Freescale Semiconductor PXS20 device dedicated to motor control. The development cost is reduced using the Freescale TOWER modular development hardware platform.

The system is designed to drive one or more 3-phase PMSMs. System features include:

- Modular software concept of 3-phase PMSM speed field-oriented control that can be configured for single or multiple PMSM applications
- Current sensing with three shunt resistors
- Support for encoder position transducers
- Application control user interface using the FreeMASTER debugging tool

2 System Concept

The system is designed to drive either one 3-phase PMSM or possibly, more 3-phase PMSMs. The application meets the following performance specifications:

- Targeted at the PXS20 Controller; see the dedicated user manual for the PXS20 found at www.freescale.com
- Running on the Freescale Tower System based on the TWR-PXS2010 control and TWR-MC-LV3PH power stage boards; see the dedicated user manuals for the TWR-PXS2010 and TWR-MC-LV3PH
- Control technique incorporating:
  - Vector control of a 3-phase PMSM with position sensor
  - Closed-loop speed control
  - Bidirectional rotation
  - Both motor and generator modes
  - Closed-loop current control
  - Flux and torque independent control
  - Starting up with alignment
  - Field weakening is not implemented in this software version
  - Reconstruction of 3-phase motor currents from two shunt resistors
  - 100 sampling period with the FreeMASTER recorder
- FreeMASTER software control interface (motor start/stop, speed setup)
  - The control page can control the PMSM
- FreeMASTER software monitor
  - FreeMASTER software graphical control page: controls required speed, actual motor speed, start/stop status, DC-Bus voltage level, system status
  - FreeMASTER software speed scope: observes actual and desired speeds and DC-Bus voltage
  - FreeMASTER software high-speed recorders:records reconstructed motor currents, vector control algorithm quantities
- DC-Bus over-voltage and under-voltage, over-current, overload, and start-up fail protection

3 PMSM Field-Oriented Control

The field-oriented control (FOC) approach is used for the control of the PMSM.

3.1 Fundamental principle of PMSM FOC

High-performance motor control is characterized by smooth rotation over the entire speed range of the motor, full torque control at zero speed, and fast acceleration/deceleration. To achieve such control, vector control techniques are used for PMSMs. The vector control techniques are also referred to as field-oriented control (FOC).

The FOC concept is based on an efficient torque control requirement, which is essential for achieving high control dynamics. Analogous to standard DC machines, AC machines develop maximal torque when the armature current vector is perpendicular to the flux linkage vector. Therefore, if only the fundamental harmonic of the stator-mmf is considered, the torque developed by an AC machine, in vector notation, is given by:

$$T_e = \frac{3}{2} \times \mathrm{pp} \times \overline{\psi_s} \times \bar{i}_s$$

Figure 1. Equation 1

where pp is the number of motor pole-pairs, $\bar{i}_s$ is stator current vector, $\overline{\Psi}_s$ and represents the vector of the stator flux.

The constant indicates a non-power invariant form of the transformation used. In the case of DC machines, the requirement of having the rotor flux vector perpendicular to the stator current vector is satisfied by the mechanical commutator. Because there is no such mechanical commutator in AC machines, the functionality of the commutator has to be substituted electrically by enhanced current control. This therefore implies an orientation of the stator current vector in such a way so as to isolate the component of the stator current magnetizing the machine (flux component) from the torque producing component. This can be accomplished by decomposing the current vector into two components projected in the reference frame, often called the dq frame, that rotates synchronously with the rotor.

It has became a standard to position the dq reference frame such as to have the d-axis aligned with the position of the rotor flux vector, so that current in the d-axis will alter the amplitude of the rotor flux linkage vector. That requires the reference frame position to be updated such that the d-axis will be always aligned with the rotor flux axis. Because the rotor flux axis is locked to the rotor position, in the case of PMSM machines, a mechanical position transducer can be used to measure the rotor position and therefore the position of the rotor flux axis. Having the reference frame phase set such that the d-axis is aligned with the rotor flux axis, the current in the q-axis represents solely the torque producing current component.

Setting the reference frame speed to be synchronous with the rotor flux axis speed also causes both the d and q axis current components to be DC values. This implies the use of simple current controllers to control the demanded torque and magnetizing flux of the machine, therefore simplifying the control structure design.

Figure 2 shows the basic structure of the vector control algorithm for the PMSM. To perform vector control:

- Measure and obtain the motor states, variables, and quantities, for example, phase voltages, currents, rotor speed, and position.
- Transform quantities into the two-phase system (α,β) using a Clarke transformation.
- Transform stator currents into the dq reference frame using a Park transformation.

Also keep these points in mind:

- The stator current torque (i_{sq}) and flux (i_{sd}) producing components are separately controlled.
- The output stator voltage space vector is calculated using the decoupling block.
- The stator voltage space vector is transformed by an inverse Park transformation back from the d, q reference frame into the two-phase system fixed with the stator.
- The output 3-phase voltage is generated using a space vector modulation.

To decompose currents into torque and flux producing components ((i_{sq}, (i_{sd}), the position of the motor-magnetizing flux must be known. This requires accurate sensing of the rotor position and velocity. Incremental encoders or resolvers attached to the rotor are naturally used as position transducers for vector control drives.

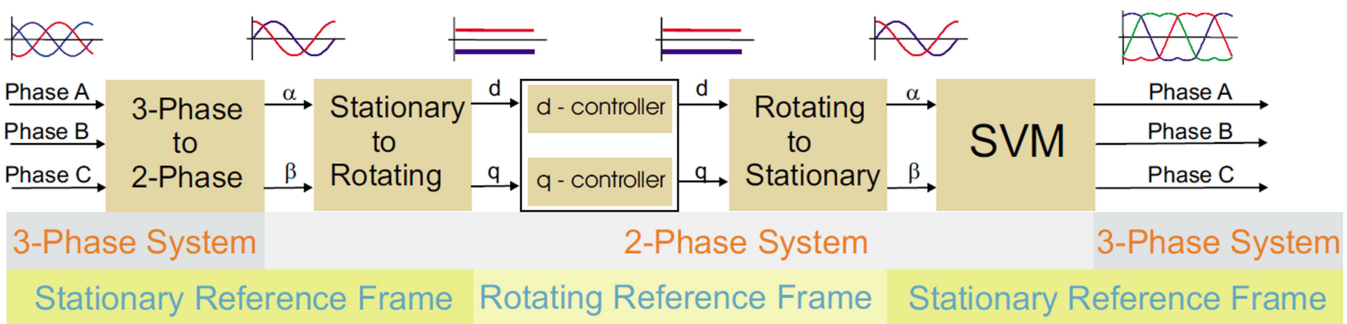

Figure 2. Field-oriented control transformations

3.2 3-Phase PMSM model in a quadrature phase synchronous reference frame

The quadrature phase model 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 only simple controllers to be used to force the machine currents into the defined states. Furthermore, full decoupling of the machine flux and torque can be achieved, which allows dynamic torque, speed, and position control.

The equations describing the voltages in the 3-phase windings of a permanent magnet synchronous machine can be written in matrix form as follows:

$$\begin{bmatrix} u_A \\ u_B \\ u_C \end{bmatrix} = R_s \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \frac{d}{dt} \begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix}$$

Figure 3. Equation 2

where the total linkage flux in each phase is given as:

$$\begin{bmatrix} \psi_A \\ \psi_B \\ \psi_C \end{bmatrix} = \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} \\ L_{ba} & L_{bb} & L_{bc} \\ L_{ca} & L_{cb} & L_{cc} \end{bmatrix} \begin{bmatrix} i_A \\ i_B \\ i_C \end{bmatrix} + \psi_{PM} \begin{bmatrix} \cos(\theta_e) \\ \cos(\theta_e - (2\pi)/3) \\ \cos(\theta_e + (2\pi)/3) \end{bmatrix}$$

Figure 4. Equation 3

where L_{aa}, L_{bb}, and L_{cc} are stator phase self-inductances, and $L_{ab} = L_{ba}$, $L_{bc} = L_{cb}$, and $L_{ca} = L_{ac}$ are mutual inductances between respective stator phases. The term Ψ_{PM} represents the magnetic flux generated by the rotor permanent magnets, and θ_e is the electrical rotor angle.

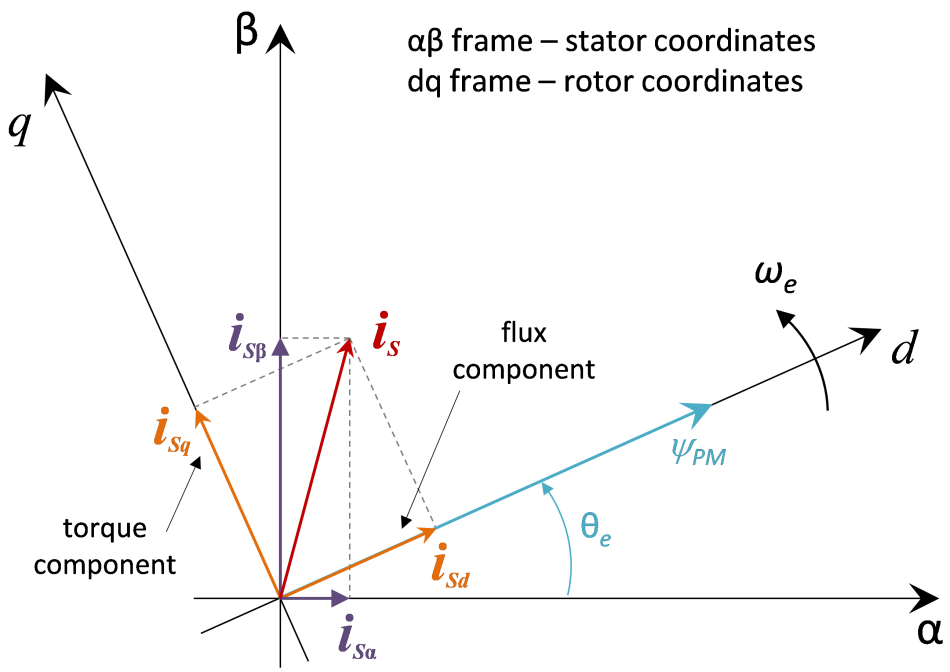

Figure 5. Orientation of stator (stationary) and rotor (rotational) reference frames, with current components transformed into both frames

The voltage equation of the quadrature phase synchronous reference frame model can be obtained by transforming the 3-phase voltage equations (Figure 1, Figure 3) into a two-phase rotational frame which is aligned and rotates synchronously with the rotor, as shown in Figure 5. Such a transformation, after some mathematical corrections, yields the following set of equations:

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = R_s \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} L_d & 0 \\ 0 & L_q \end{bmatrix} + \omega_e \begin{bmatrix} -L_q \\ L_d \end{bmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \omega_e \psi_{PM} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Figure 6. Equation 4

Figure 6 represents a non-linear cross dependent system, with cross-coupling terms in both the d and q axes, and a back-EMF voltage component in the q-axis. When the FOC concept is employed, both cross-coupling terms shall be compensated to allow independent control of the current d and q components. Design of the controllers is then governed by the following pair of equations, derived from Figure 6 after compensation:

$$u_d = R_s i_d + L_d \frac{\mathrm{d}i_d}{\mathrm{d}t}$$

Figure 7. Equation 5

$$u_q = R_s i_q + L_q \frac{\mathrm{d}i_q}{\mathrm{d}t}$$

Figure 8. Equation 6

which describes the model of the plant for the d and q current loop. Both Figure 7 and Figure 8 are structurally identical, therefore the same approach of controller design can be adopted for both the d and q controllers. The only difference is in the values of the d and q axis inductances, which results in different gains of the controllers. Considering the closed-loop feedback control of a plant model as in Figure 7 or Figure 8, using standard PI controllers, then the controller proportional and integral gains can be derived, using a pole-placement method, as follows:

$$K_p = 2\zeta\omega_0 L - R$$

Figure 9. Equation 7

$$K_I = \omega_0^2 L$$

Figure 10. Equation 8

where ω_0 represents the Natural Frequency [rad s-1] and ζ is the Damping Factor [-] of the current control loop.

3.3 Phase current measurement

The 3-phase voltage source inverter, depicted in Figure 11, uses three shunt resistors, R7, R8, and R9, placed in each of the inverter legs as phase current sensors. Stator phase current which flows through the shunt resistor produces a voltage drop which is interfaced to the AD converter of the microcontroller through conditional circuitry. See the TWR-MC-LV3PH User's Manual for more information.

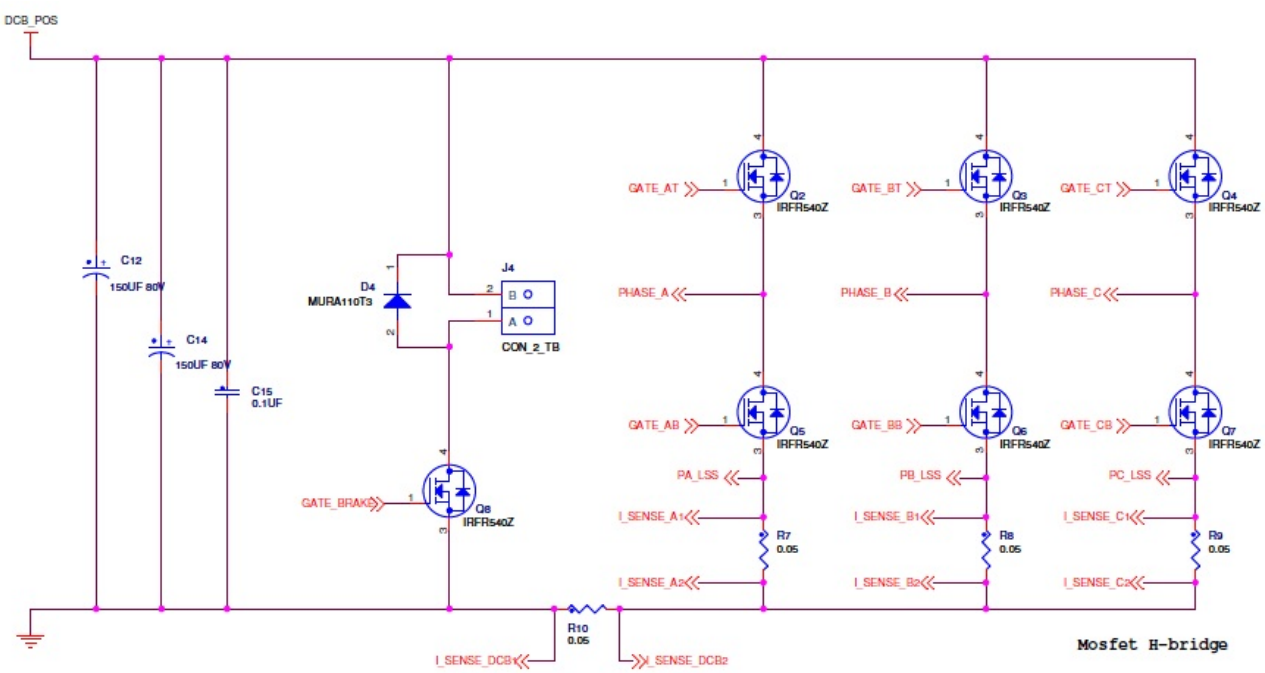

Figure 11. A 3-phase DC/AC inverter with shunt resistors for current measurement

Figure 12 shows an operational amplifier and input signal filtering circuit which provides the conditional circuitry and adjusts voltages to fit into the ADC input voltage range.

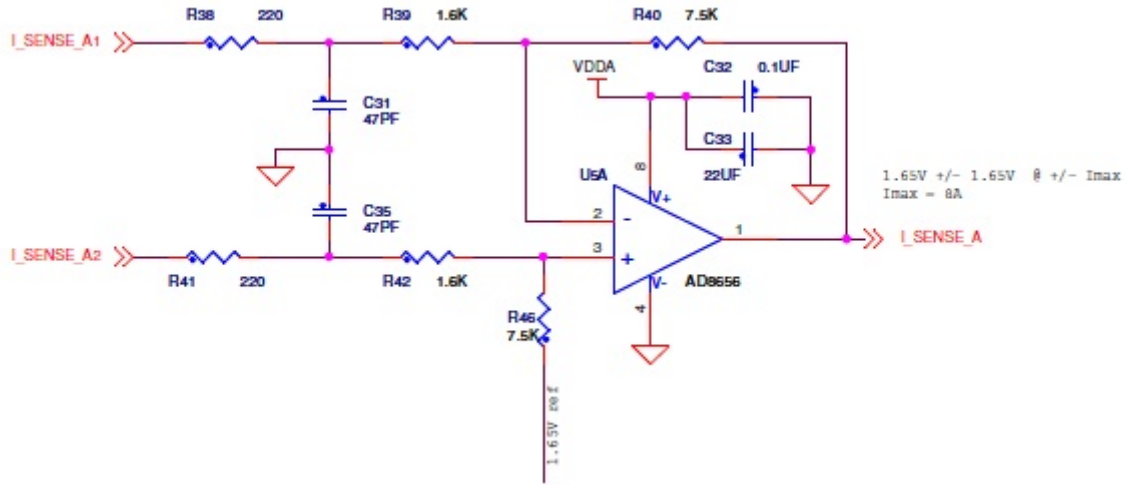

Figure 12. Phase current measurement conditional circuitry

The phase current sampling technique is a critical issue for detection of phase current differences and for acquiring full 3-phase information of the stator current by its reconstruction. A phase current flowing through shunt resistors produces a voltage drop which needs to be appropriately sampled by the AD converter when the low-side transistors are switched on. The current cannot be measured by the current shunt resistors at an arbitrary moment. This is because the current flows through the shunt resistor only when the bottom transistor of the respective inverter leg is switched on. Therefore, considering the diagram depicted in Figure 11, the phase A current is measured using the R7 shunt resistor and can be sampled only when the transistor Q5 is switched on. Correspondingly, the current in phase B can be measured only if the transistor Q6 is switched on, and the current in phase C can be measured only if the transistor Q7 is switched on. To get an actual instant of current sensing, voltage waveform analysis must be performed.

Generated duty cycles (phase A, phase B, phase C) of two different PWM periods are depicted in Figure 13. These phase voltage waveforms correspond to a center-aligned PWM with sinewave modulation. As shown in Figure 13 (PWM period I), the best sampling instant of a phase current is in the middle of the PWM period, when all bottom transistors are switched on. However, not all three currents can be measured at an arbitrary voltage shape. The PWM period II in Figure 13 shows the case when the bottom transistor of phase A is on for a very short time. If the on time is shorter than a certain critical time, depending on the hardware design, the current cannot be correctly measured.

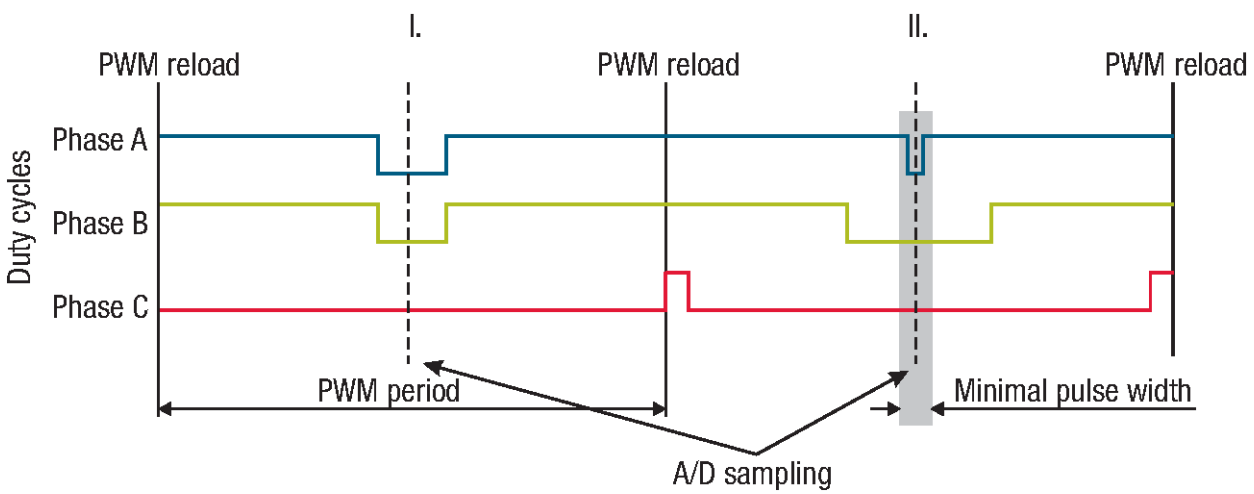

Figure 13. Generated phase duty cycles in different PWM periods

In standard motor operation, when the supplied voltage is generated using the space vector modulation, the sampling instant of the phase current takes place in the middle of the PWM period in which all bottom transistors are switched on. If the modulation index of the applied SVM technique increases, there is an instant when one of the bottom transistors is switched on for a very short time period. Therefore, only two currents are measured and the third is calculated from the equation:

$$i_A + i_B + i_C = 0$$

Figure 14. Equation 9

Therefore, a minimum on time of the low-side switch is required for 3-phase current reconstruction.

4 Hardware and PXS20 Configuration

The PMSM speed field-oriented control application software and hardware are designed to meet the following hardware configuration and technical specifications:

- Freescale Tower System based on the TWR-PXS2010 control and TWR-MC-LV3PH power stage boards is used. See the dedicated user manuals for the TWR-PXS2010 and TWR-MC-LV3PH.
- PWM output frequency = 20 kHz
- Current loop sampling period = 100 µs
- Speed loop sampling period = 1 ms
- 3-phase current measurement using three shunt resistors on bottom side of each inverter leg. Phase current measurement feedback is routed to ADC0 as follows:
  - TOWER Primary - A30 - phase A current: ADC0 – CH0
  - TOWER Primary - A29 - phase B current: ADC0 – CH2
  - TOWER Primary - A28 - phase C current: ADC0 – CH3
- DC bus voltage measurement routed to ADC0 as follows:
  - TOWER Primary - B30 - DC bus voltage: ADC0 – CH4
- Encoder position feedback routed to the eTimer0 module as follows:
  - Encoder phase A, TOWER Primary - A34: eTimer0 - CH0, Counter #2 input
  - Encoder phase B, TOWER Primary - A33: eTimer0 - CH0, Counter #3 input

The PXS20 is used to control the inverter.

The specific peripherals, utilized used for motor control will be described in the following section:

- FlexPWM
- CTU
- eTimer0

4.1 On-chip motor control peripherals interconnection

Figure 15 shows how the CTU interacts with the following peripherals:

- ADC0 and ADC1
- DSPI1
- eTimer0, eTimer1, and eTimer2
- FlexPWM_0 and FlexPWM_1
- FlexRay

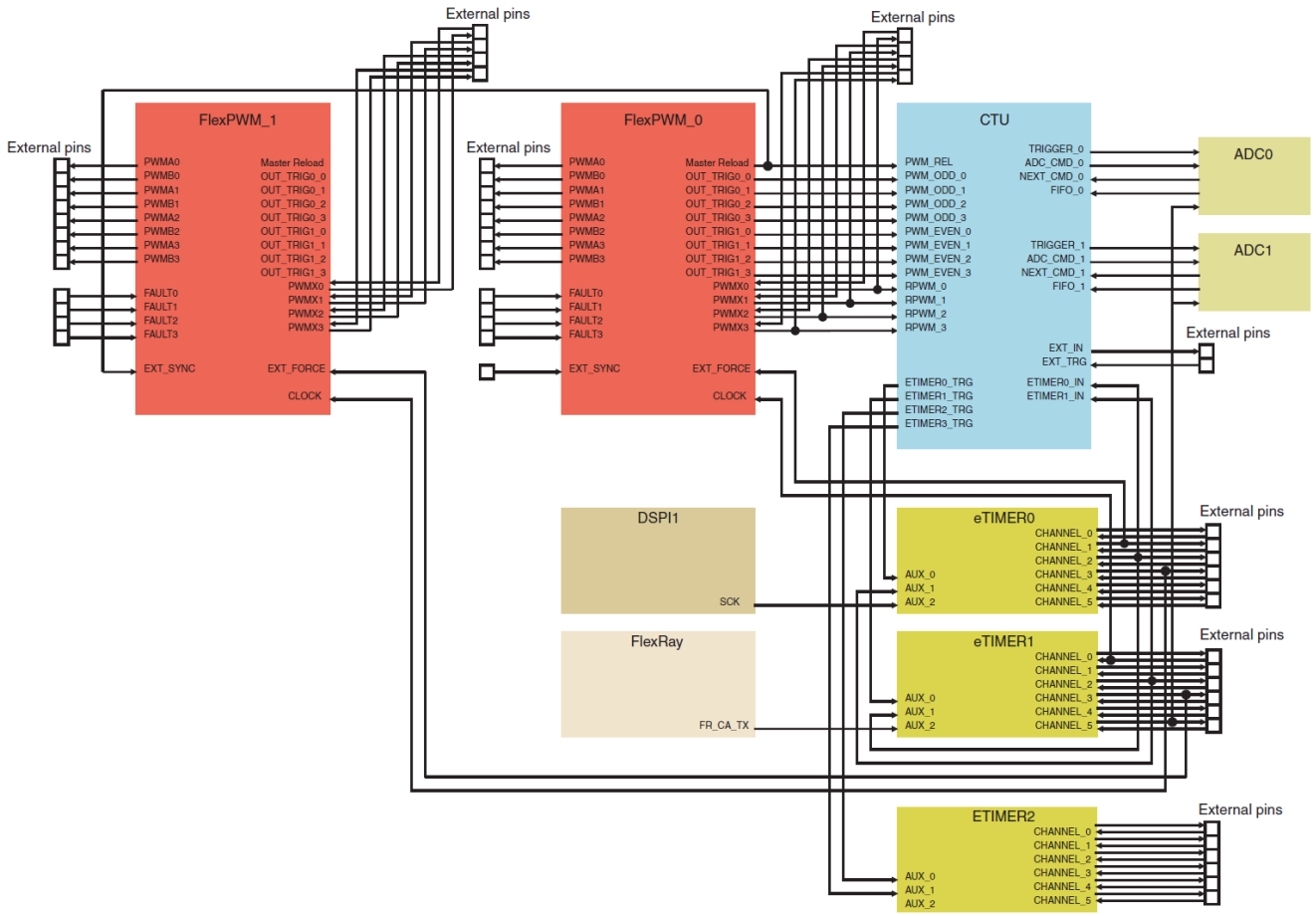

Figure 15. PXS20 motor control peripherals connection

4.2 FlexPWM

The PXS20 Clock Generation Module is configured to generate a clock signal of 120 MHz on the MC_PLL_CLK bus. The Pulse Width Modulator module (FlexPWM) is clocked from the MC_PLL_CLK, therefore it is placed behind the IPS Bus Clock Sync Bridge.

The PXS20 device contains four PWM channels, each of which is configured to control a single half-bridge power stage. Two modules are included on the 257 MAPBGA devices; on the 144 LQFP package, only one module is present. The targeted TWR-PXS2010 board is based on the 144 LQFP device and therefore one FlexPWM module is used to control one 3-phase PMSM motor. The 257 MAPBGA-based PXS20 is able to control two motors. FlexPWM_0 controls the motor.

The FlexPWM_0 submodule #0 is configured to run as a master and to generate an MRS and counter synchronization signal (master sync) for other submodules in FlexPWM module 0. The MRS signal is generated every second opportunity of submodule #0, VAL1 compare, that is, a full cycle reload. All double buffered registers, including compare registers VAL2, VAL3, VAL4, and VAL5, are updated on occurrence of the MRS, therefore, the new PWM duty cycles are updated every two PWM periods.

FlexPWM modulo counting, for generation of center-aligned PWM signals, is achieved by setting the VAL0 register to zero and INIT register to the negative value of VAL1. Considering a PWM clock of 120 MHz, a required PWM output of 20 kHz, and a PWM reload period of 100 µs, then the INIT, VAL0, and VAL1 registers of submodules 0, 1, and 2 are set as follows:

- INIT = -120000000/20000/2 = -3000 DEC = 0xF448
- VAL0 = 0 DEC
- VAL1 = - INIT = 3000 DEC = 0x0BB8

The reload frequency of FlexPWM_0 submodules 0, 1, and 2 is set to “Every two opportunities” and “Full cycle reload” is enabled. Because submodule #0 is a master that generates the MRS signal, the reload of the double buffered registers of submodule #0 is done on a “Local Reload”. Submodules 1 and 2 are slaves, therefore the reload of their double buffered registers is done on “Master Reload”, broadcasted from submodule #0. Similarly, submodules #0 counter is initialized on a “Local Sync” event, while submodules 1 and 2 on a “Master Sync” event, again broadcasted from submodule #0.

Because some registers are double buffered on occurrence of a FORCE OUT signal, all submodules of FlexPWM_0 have, as a force source, selected the “Local Reload” event.

All PWM channels are used to drive the 3-phase DC/AC inverter, therefore each PWM pair is driven in complementary mode, with dead-time automatically added on each rising edge of the respective PWM signal. The power stage used with the MC33937 pre-driver inverts the polarity of the PWM signals for the top transistors (active low logic), therefore PWM A output polarity in all submodules is set as “Inverted”. Therefore, also during a fault state, the output of PWM A of each submodule is set to logic one.

The FlexPWM modules include a Fault Protection logic, which can control any combination of PWM output pins and automatically disable PWM outputs during a fault state. Faults are generated by a logic one on any of the FAULTx pins. To enable a mapping of all of the fault pins, fault disable mapping registers (DISMAP) of all submodules must be enabled (logic one).

4.3 CTU

The MRS signal generated from the FlexPWM module is internally routed to the Cross Triggering Unit (CTU) module, where it is selected using the input selection register, Trigger Generator Subunit Input Selection (TGSISR) as the source of the master reload signal for the CTU. This signal is used for the reload trigger compare registers and reloads the TGS counter with the value stored in the TGS counter reload register. The TGS counter register is used to compare the current counter value with the values of the trigger compare registers, and where the two values are the same an event is generated. The TGS is configured in triggered mode.

Because the MRS signal is generated every two PWM periods, the CTU counter can count up to the value of 12000DEC considering the init value is set to zero.

The following TGS trigger compare register is used for trigger events:

- T0CR = 0x0 (0 DEC)

This application uses only T0CR = 0, but other trigger compare registers could eventually trigger other events, for example, when using a resolver.

The CTU Scheduler subUnit (SU) generates the trigger event output according to the occurred trigger event. The following trigger event outputs are generated:

- ADC command output: T0CR generates an ADC command event output, with the command offset initially set to one. This is used as the synchronization signal to the ADC:
  - ADC commands #0, #1, #2 for 3 motor phase currents measurement
  - ADC commands #3 for DC bus voltage measurement

The SU uses a Commands List to select the command to send to the ADC when a trigger event occurs. Each ADC command sent by the CTU into the ADC specifies:

- Whether the actual command is a first command of a new stream of consecutive commands or not
- Whether an End Of Conversion (EOC) interrupt is issued when the conversion specified by the command is finished
- Which channels are to be converted for both ADC modules
- The target FIFO register for storing the conversion results

Because the trigger compare register for trigger T0CR is set to zero, it generates an ADC start of conversion request at the beginning of each PWM reload cycle. When a T0CR trigger event occurs, the ADC command selected by the index value, T0_INDEX, in the command list control registers CLCR1 is sent to the ADC. At each T0CR trigger event, four ADC commands are executed in a stream:

- The first three commands in a stream specifies three phase currents to be sampled sequentially (PXS20 simultaneous sampling could be used with other hardware board)
- The fourth command specifies the DC bus voltage to be sampled

The index pointer to the ADC command list T0_INDEX is updated according to the sector in which the actual output voltage vector resides, calculated by the space vector modulation of the FOC algorithm. There are six sectors within the output voltage hexagon of the inverter, therefore, six different ADC command sequences are selected for one full revolution of the voltage vector. This technique is necessary when the phase current measurement is done using three shunt resistors placed on the bottom side of each inverter leg. Because the shunt resistor is placed at the bottom side of the inverter leg, the phase current can be measured only when the bottom transistor is switched on. Because the sum of the three currents in the motor windings is zero, only two currents are measured and the third is calculated. Which phases are measured and which are calculated changes according to the voltage vector angle, that is, the phases with the largest PWM on-pulse on the bottom transistors are selected to get the best current information.

Configuration of the CTU ADC commands is shown in Figure 16.

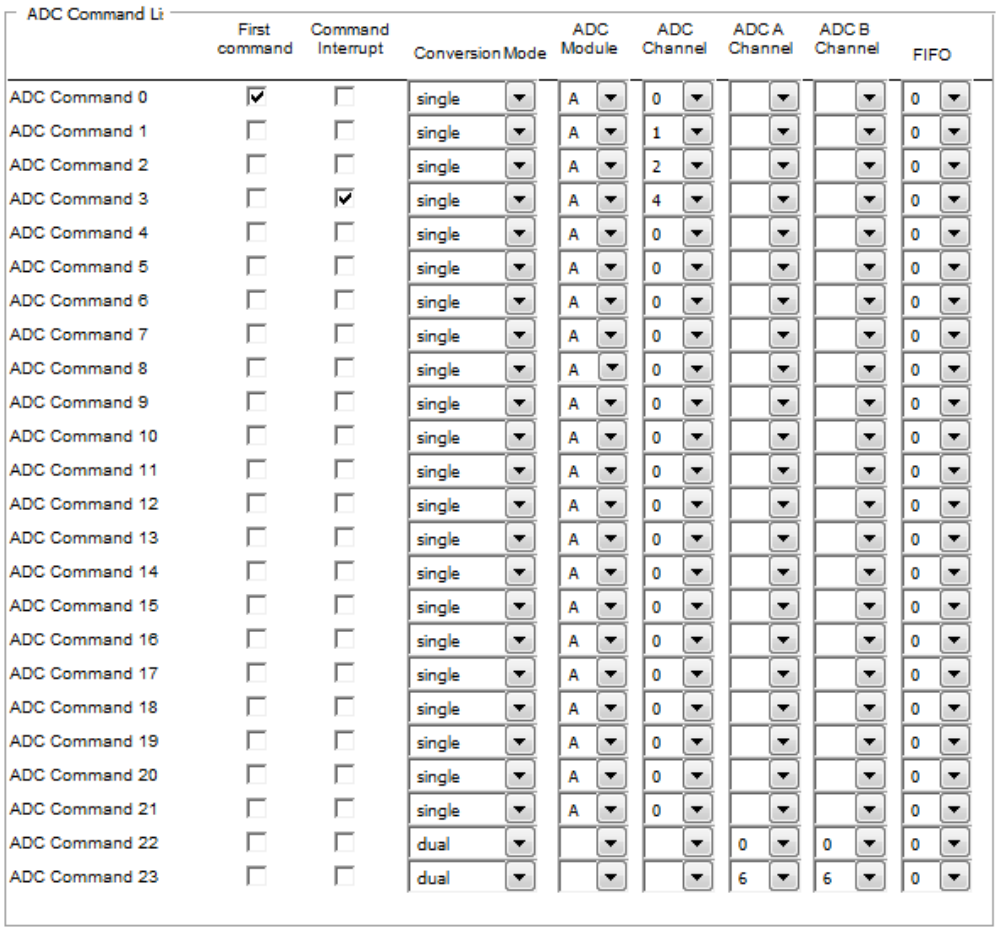

Figure 16. Configuration of CTU ADC commands for the PMSM FOC application

4.4 eTimer0

This eTimer module is used only for decoding two square-wave signals from a primary encoder.

4.4.1 Encoder

Because of the hardware design, channel #0 of this eTimer module is selected. The purpose is to decode the two 90° shifted square wave signals and count up or down all the rising/falling edges based on their sequence. The software routine then reads the associated counter value to get the information on the rotor position. The counter value is read from within the POSPE_GetPositionElEnc() function, periodically within the CTU-ADC interrupt service routine; see the data flow diagram shown in Figure 20. Parameters entering POSPE_GetPositionElEnc() represent a PMSM drive structure related to Motor #1.

To decode the encoder signals, timer channel #0 of module eTimer0 (eTimer0-ETC[0]) is configured as follows:

- Counting mode - “Quadrature count mode, uses primary and secondary sources”
- Count direction - “Count Up”
- Primary source - “Counter #2 input”
- Secondary source - “Counter #3 input”
- Count stop mode - “Count Repeatedly”
- Count length - “Count Until Compare then Reinitialize”
- Preload control for CNTR:
  - “Load CNTR with CMPLD1 upon successful compare with COMP2”
  - “Load CNTR with CMPLD2 upon successful compare with COMP1”
- Compare mode - “Use COMP1 when counting up and COMP2 when counting down"
- Output mode - “Toggle OFLAG on successful compare with COMP1 and/or COMP2”
- COMP1 = 0x07FF (2047 DEC)
- COMP2 = 0xF800 (-2048 DEC)
- CMPLD1 = 0x07FF (2047 DEC)
- CMPLD2 = 0xF800 (-2048 DEC)
- LOAD = 0x0, this value is updated by a software routine during the ALIGN phase

The compare registers of eTimer0 channel #0 are set according to the number of encoder pulses per one mechanical revolution. In this case, an encoder sensor with 1024 pulses is used. Considering quadrature mode, this means encoder capability of position recognition with a precision that is four times higher than the number of pulses in the application, and the maximum numbers of edges is 4096. Then the compare registers are calculated as follows:

- COMP1 = 4096/2 - 1 = 2047 (0x07FF HEX)
- COMP2 = -4096/2 = -2048 (0xF800 HEX)

4.5 ADC conversion and interrupt timing

Configuration of the FlexPWM, CTU, and eTimer0 peripheral modules, as described in sections FlexPWM, CTU, and eTimer0, results in a sequence of triggers/events that are shown in Figure 17. The application state machine functions are called from an interrupt service routine, which is associated with the CTU-ADC command interrupt request. As shown in the ADC command list configuration in Figure 16, the ADC command interrupts are linked with the CTU trigger T0CR, that is, when the measurement of the phase currents and DC bus voltage has finished.

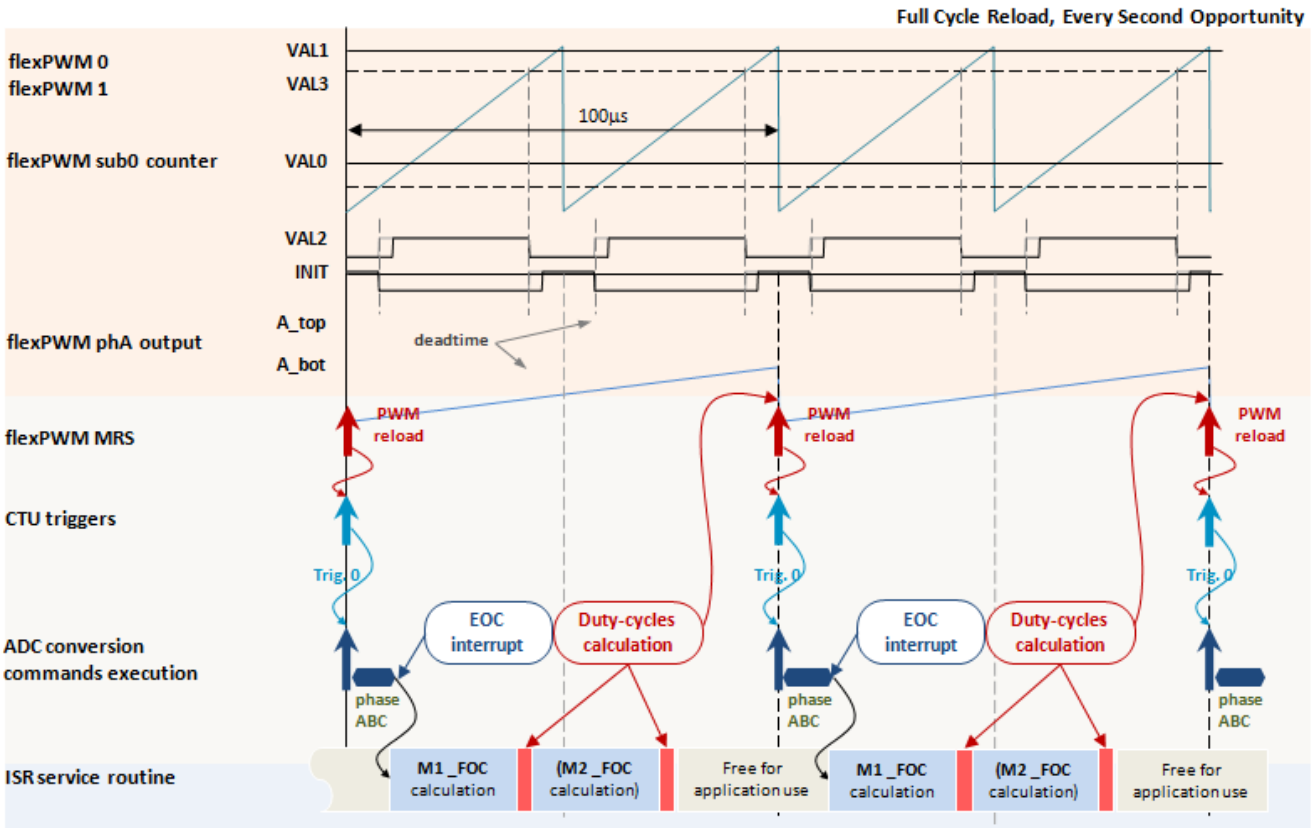

Figure 17. FlexPWM-CTU-ADC conversion timing

For the correct calculation of the field-oriented control algorithm, it and the phase current measurements shall be executed at the same time. Because there are only two independent sample and hold circuitries on the PXS20, parallel sampling of all signals is impossible. Two phase currents can be read at the same. The sequence of the measurements is handled by the CTU trigger T0CR.

5 Software Design

5.1 Introduction

This section describes the software design of the PMSM FOC framework application that can be used for either a one or two synchronous motor configuration. The application overview and a description of software implementation are provided. The aim of this chapter is to help in understanding the designed software.

5.2 Application software design

To make the software easily applicable for a one or multi-motor application, all parameters and variables are grouped in a specific data structure. In this application, the data structure pmsmDrive_t represents a predefined format that is used for organizing and storing data for the purpose of working on it with various algorithms, routines, and functions. See the following example for more information:

```
typedef struct{
    AppFaultStatus   faultID;      // Application faults
    AppFaultStatus   faultIDp;     // Application faults flags
    tU32             svmSector;    // Space Vector Modulation sector
    SWLIBS_2Syst     iDQFbck;      // dq axis current feedback
    SWLIBS_2Syst     iAlBeFbck;    // Alpha/Beta - axis current feedback
    SWLIBS_2Syst     iDQReq;       // dq axis required currents, given by speed PI
    SWLIBS_2Syst     uDQReq;       // dq axis required voltages given by current PIs
    SWLIBS_2Syst     iDQReqZC;     // Zero cancellation in current branch
    SWLIBS_2Syst     iDQErr;       // Error between the reference and feedback signal
    SWLIBS_2Syst     uAlBeReq;     // Alpha/Beta required voltages
    SWLIBS_2Syst     uAlBeReqDCB;  // Al/Be voltages after DC Bus ripple elimination
    SWLIBS_2Syst     thTransform;  // Transformation angle - enters Park transform.
    SWLIBS_3Syst     iAbcFbck;     // 3-phases current feedback
    SWLIBS_3Syst     pwm32;        // 3-phase 32bit Duty-Cycles from uAlBeReqDCB
    FLEXPWM_VAL_COMPL         pwm16;      // 3-phase 16bit Duty-Cycles feeding PWM registers
    GMCLIB_ELIMDCBUSRIP_T     elimDcbRip; // DC Bus voltage ripple elimination
    GFLIB_CONTROLLER_PIAW_R_T dAxisPI;    // d-axis current PI controller
    GFLIB_CONTROLLER_PIAW_R_T qAxisPI;    // q-axis current PI controller
    GFLIB_CONTROLLER_PIAW_P_T speedPI;    // Speed PI controller
    GFLIB_RAMP_T              speedRamp;  // Reference speed ramp generation
    GDFLIB_FILTER_IIR1_T      dAxisZC;    // d-axis current zero cancellation
    GDFLIB_FILTER_IIR1_T      qAxisZC;    // q-axis current zero cancellation
    GDFLIB_FILTER_IIR1_T      uDcbFilter; // DC bus voltage filter settings
    adcModulePmsm_t  adc;                 // ADC measurement
    pospeModule_t    pospeRes;            // Position/Speed module for resolver
    pospeModule_t    pospeEnc;            // Position/Speed module for encoder
    pospeControl_t   pospeControl;        // Position/Speed variables needed for control
    uni3PeriphCfgPmsm_t  uni3PeriphCfg;   // ADC measurement channels configuration
    pospePeriphCfg_t          pospeCfg;   // Encoder/Resolver configuration
    MC33937_SET_T    preDrv;        // PreDriver configuration
    driveStates_t    cntrState;     // State machine parameters, switches states
}pmsmDrive_t;
```

The proposed structure includes all variables related to the cascade structure for the PMSM FOC control, state variables related to the application, and data for peripherals configuration, such as base address or channel offset.

Then the declaration of possible structures used for different motors can be as follows:

```
pmsmDrive_t    M1;   //related to Motor #1
pmsmDrive_t    Mx;   //related to Motor #x - for possible dual, triple motor control
```

The names of all variables are composed of a prefix, given by the name of structure Mx, and the name of the variables/ structures which are members of the main structure pmsmDrive_t. For instance, M1.iAbcFbck.s32Arg1 represents the phase A current of Motor #1, or M2.iDQReq.s32Arg2 represents the q-axis current of Motor #2. The list of cascade structure variables can be seen in Figure 22, where the block scheme of the FOC for Motor #1 is shown and all FOC variables are described.

5.3 Application data flow overview

The application software is interrupt driven running in real time. There is one periodic interrupt service routine associated with the CTU-ADC command interrupt request, executing all motor control tasks. These include both the fast current and slow speed loop controls. All tasks are performed in the order described by the application state machine shown in Figure 20 and the application flowcharts shown in Figure 18 and Figure 19.

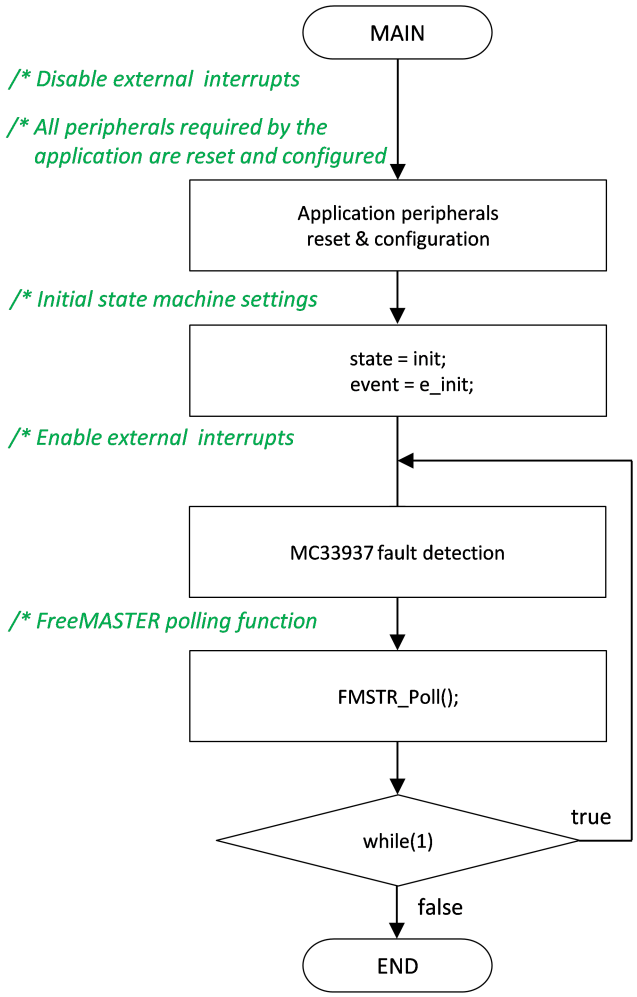

Figure 18. Flow chart diagram of main function with background loop

To achieve a precise and deterministic sampling of the analogue quantities, and to execute all necessary motor control calculations, the state machine functions are called within a periodic interrupt service routine. Therefore, to call the state machine functions, the periphery causing this periodic interrupt must be properly configured and the interrupt enabled. As described later, all peripherals are initially configured and all interrupts are enabled before calling the state machine. See Figure 20 for more information.

After interrupts are enabled and all peripheries are correctly configured, the state machine functions are called from the CTU-ADC interrupt service routine. See Hardware and PXS20 Configuration for configuration of the peripherals. The background loop handles non-critical timing tasks, such as the FreeMASTER communication polling.

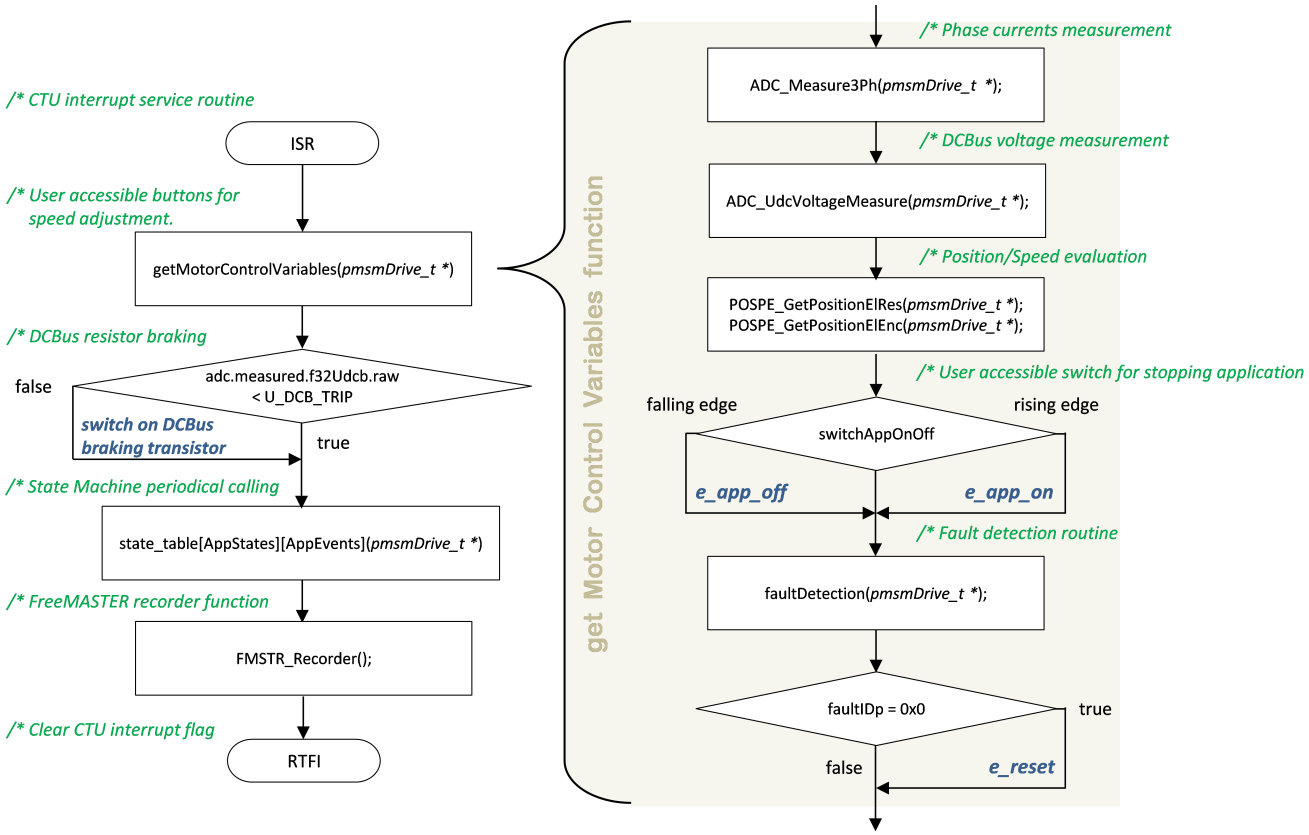

Figure 19. Flow chart diagram of periodic interrupt service routine

5.4 State machine

The application state machine is implemented using a two-dimensional array of pointers to functions variable called state_table[][](), with the first parameter describing the current application event and the second parameter describing the actual application state. These two parameters select a particular pointer to a state machine function, which causes a function call whenever state_table[][]() is called.

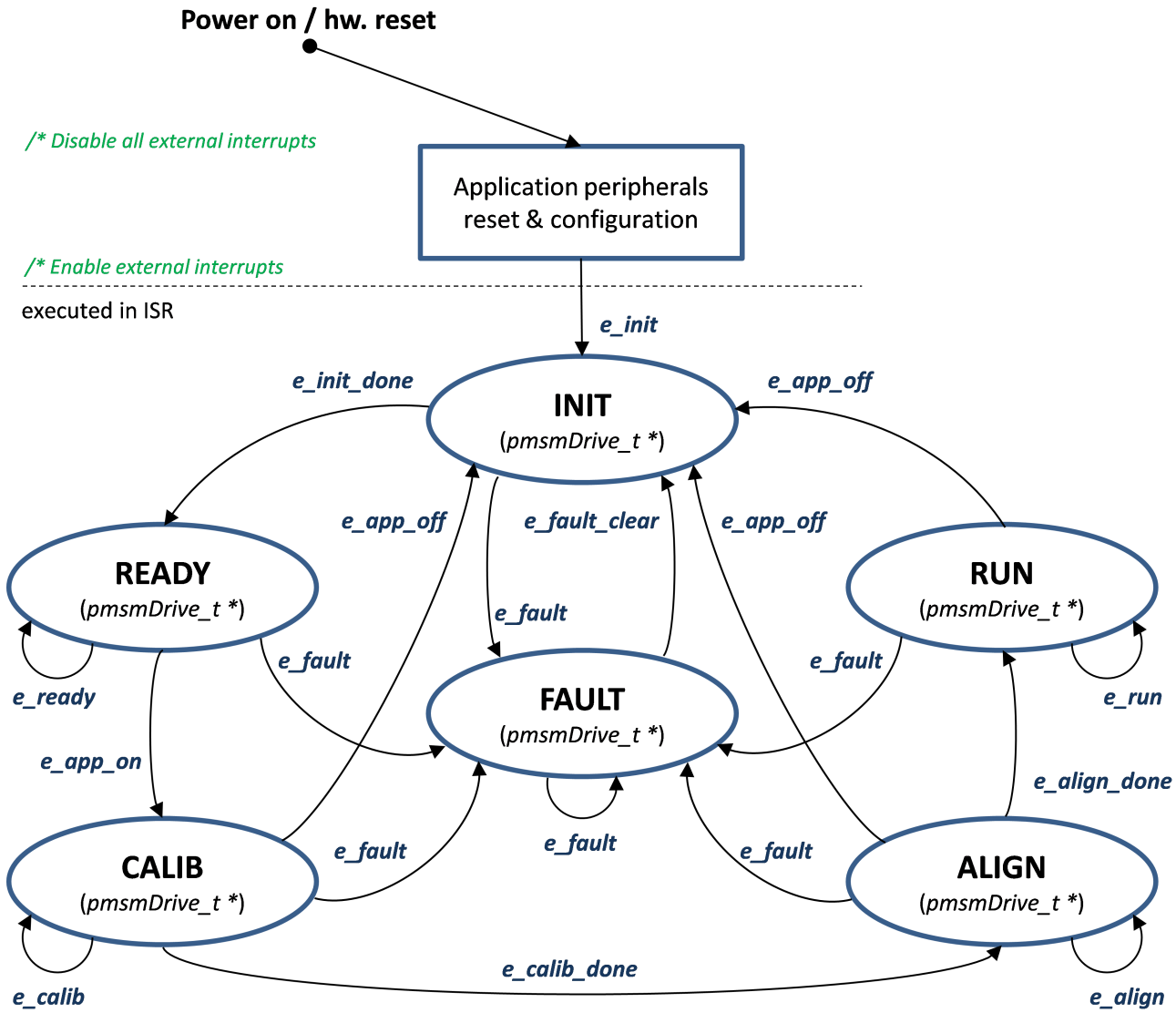

Figure 20. Application state machine

The application state machine consists of the following states, selected using the variable state defined as AppStates:

- INIT - state = 0
- FAULT - state = 1
- READY - state = 2
- CALIB - state = 3
- ALIGN - state = 4
- RUN - state = 5

To signalize/initiate a change of state, the following events are defined, selected using the variable event defined as AppEvents:

- e_fault - event = 0
- e_fault_clear - event = 1
- e_init - event = 2
- e_init_done - event = 3
- e_ready - event = 4
- e_app_on - event = 5

- e_calib - event = 6
- e_calib_done - event = 7
- e_align - event = 8
- e_align_done - event = 9
- e_run - event = 10
- e_app_off - event = 11

After power on or a reset, all used peripherals are reset and configured as required by the application. See Hardware and PXS20 Configuration for more information. To configure the application peripheral modules correctly, all the interrupts are disabled at the beginning of the peripherals reset and configuration section. Therefore, the periodic CTU-ADC interrupt is not requested and the state machine functions cannot be executed until all of the interrupts are enabled and all peripherals set.

To initialize all of the application variables and parameters, a state INIT is called as the first state/function of the state machine. To fulfill this condition, the application state and event are set to state = init and event = e_init before entering the background loop.

All interrupts are enabled after a successful peripherals configuration. From this point, the CTU-ADC interrupts are enabled, therefore, considering a correct configuration of the peripherals, the first call of the state machine function will be from within the CTU-ADC interrupt service routine.

5.5 State – INIT

The state INIT is the first called state/function of the state machine and can be entered from all states except the READY state, provided there are no faults detected. All application variables and parameters are initialized in the state INIT.

After a successful initialization, the application event is automatically set to event=e_init_done at the end of the execution of the INIT state, and the state READY is selected as the next state to enter.

According to the data flow diagram of the CTU-ADC interrupt service routine, shown in Figure 19, the routine for the 3-phase current reconstruction, ADC_Measure2Ph(), is executed first, followed by the DC bus voltage measurement and the rotor position measurement routines. The fault detection function is always called before the state machine function call, ensuring a correct transition to the FAULT state in case there is a fault detected. If there is no fault detected, the application event remains set to event=e_init_done, hence the READY state will be selected as the next state to execute.

5.6 State – FAULT

The application goes into the FAULT state immediately when a fault is detected. The system allows all states to pass into the FAULT state by setting event = e_fault. The state FAULT is a state that allows transition back to itself if a fault is present in the system and the user does not request a clearing of the fault flags.

There are two variables which signal a fault occurrence in the application. The actual fault register, faultID, represents the current state of the fault pin/variable, and so on, and the pending fault register, faultIDp, represents a fault flag, which is set when the actual fault is/was true. Even if the actual fault is reset and the fault source disappears, the pending fault remains set until the user manually clears it. Such a mechanism allows the user to stop the application and analyze the cause of the failure even if the fault was caused by a short glitch on the monitored pins/variables.

The state FAULT can be exited only when the application variable switchFaultClear is manually set to true using FreeMASTER. That is, the user has acknowledged that the fault source has been removed and the application can be restarted. When the user sets switchFaultClear = true, the following sequence is automatically executed:

```
faultID.R               = 0x0;              // Clear Fault register
faultIDp.R              = 0x0;              // Clear Pending Fault register
switchFaultClear        = false;            // Reset fault clearing switch
event                   = e_fault_clear;    // new application event
```

Setting event to event = e_fault_clear while in the FAULT state represents a new request to proceed to the INIT state. This request is purely a user action and does not depend on the actual fault status, that is, it is up to the user when to decide to set switchFaultClear true. However, according to the interrupt data flow diagram shown in Figure 11, function faultDetection() is called before the state machine function state_table[event][state](). Therefore, all faults will be checked again, and if there is any fault condition remaining in the system, the respective bits in the faultID and faultIDp variables will be set. As a consequence of faultID and faultIDp not equaling zero, function faultDetection() will modify the application event from e_fault_clear back to e_fault, which means jumping to the FAULT state when the state machine function state_table[event][state]() is called. Therefore, the INIT state will not be entered even though the user tried to clear the fault flags using switchFaultClear.

The next state (INIT) is set to be entered when all fault bits are cleared, meaning no fault is detected (faultIDp.R = 0x0) and the application variable switchFaultClear is manually set to true.

5.6.1 Application faults

Both faultID and faultIDp are defined as AppFaultStatus, which is a 32-bit long data type. Application faults are bit mapped in the AppFaultStatus type as follows:

Table 1. AppFaultStatus type

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FLT 31	FLT 30	FLT 29	RESERVED				FLT 24	FLT 23	FLT 22	FLT 21	RESERVED		FLT 18	FLT 17	FLT 16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLT 15	FLT 14	FLT 13	FLT 12	FLT 11	FLT 10	FLT9	FLT8	FLT7	FLT6	FLT5	FLT4	FLT3	FLT2	FLT1	FLT0

- FLT0 - OverDCBusVoltage (Over-voltage on the DC bus)
- FLT1 - UnderDCBusVoltage (Under-voltage on the DC bus)
- FLT2 - OverDCBusCurrent (Over-current on the DC bus)
- FLT3 - OverLoad (Overload Flag)
- FLT4 - MainsFault (Mains out of range)
- FLT5 - WrongHardware (Wrong hardware fault flag)
- FLT6 - OverHeating (Overheating fault flag)
- FLT7 - OverPhaseACurrent (Over-current on phase A)
- FLT8 - OverPhaseBCurrent (Over-current on phase B)
- FLT9 - OverPhaseCCurrent (Over-current on phase C)
- FLT10 - OffCancError (Offset cancellation error)
- FLT11 - MC33937_TLIM (over temperature) - not used in the current software version
- FLT12 - MC33937_DESAT (desaturation detected) - not used in the current software version
- FLT13 - MC33937_VLS (low VLS detected) - not used in the current software version
- FLT14 - MC33937_OC (over current detected) - not used in the current software version
- FLT15 - MC33937_PhaseE (phase error) - not used in the current software version
- FLT16 - MC33937_FrameE (SPI communication frame error) - not used in the current software version
- FLT17 - MC33937_WriteE (SPI communication write error) - not used in the current software version
- FLT18 - MC33937_RST (reset event) - not used in the current software version
- FLT21 - FOCError (error in the FOC calculation function)
- FLT22 - AlignError (error during alignment)
- FLT23 - CalibError (error during ADC calibration)
- FLT24 - InitError (error during app initialization)
- FLT29 - FLEXPWM_Error (error in FlexPWM hardware initialization)

- FLT30 - ADC_Error (error in ADC hardware initialization)
- FLT31 - CTU_Error (error in CTU hardware initialization)

5.7 State – READY

In the CALIB state, the ADC DC offset calibration is performed. When the state machine enters the CALIB state, all PWM outputs are enabled.

The DC offset is calibrated by generating a 50% duty-cycle on the PWM outputs and taking several measurements of all the configured ADC channels. These measurements are then averaged and the average value for each channel is stored. This value will be subtracted from the measured value when in normal operation. This removes the half range DC offset, caused by a voltage shift of 1.65 V in the conditional circuitry, in all three phases. See Figure 12 for more information.

The state CALIB is a state that allows transition back to itself, provided that no faults are present, the user has not requested a stop of the application by switchAppOnOff=true, and the calibration process has not finished.

The number of samples for averaging is set by default to 2^8=256 and can be modified in the state INIT.

Once all 256 samples have been taken and the averaged values successfully saved, the application event is automatically set to event=e_calib_done and the state machine can proceed to the state ALIGN.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting event to event=e_app_off, which is done automatically on a falling edge of switchAppOnOff=false using FreeMASTER.

5.8 State – CALIB

In the CALIB state, the ADC DC offset calibration is performed. When the state machine enters the CALIB state, all PWM outputs are enabled.

The DC offset is calibrated by generating a 50% duty-cycle on the PWM outputs and taking several measurements of all the configured ADC channels. These measurements are then averaged and the average value for each channel is stored. This value will be subtracted from the measured value when in normal operation. This removes the half range DC offset, caused by a voltage shift of 1.65 V in the conditional circuitry, in all three phases. See Figure 12 for more information.

The state CALIB is a state that allows transition back to itself, provided that no faults are present, the user has not requested a stop of the application by switchAppOnOff=true, and the calibration process has not finished.

The number of samples for averaging is set by default to 2^8=256 and can be modified in the state INIT.

Once all 256 samples have been taken and the averaged values successfully saved, the application event is automatically set to event=e_calib_done and the state machine can proceed to the state ALIGN.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting event to event=e_app_off, which is done automatically on a falling edge of switchAppOnOff=false using FreeMASTER.

5.9 State – ALIGN

Alignment of the rotor and stator flux vectors is used to mark the zero position necessary only for relative position sensors such as encoders and so on. When using a relative position sensor such as an encoder, the zero position is not known. Therefore, counting of the encoder edges must be correctly reset at the start of an operation. This is done in the ALIGN state,

where a DC voltage is applied in phase A for a certain period. This will force the rotor to rotate to an align position, that is, the stator and rotor fluxes are aligned. The rotor position in which the rotor stabilizes after applying this DC voltage is set as the zero position, therefore the timers eTimer0-CH0 and/or eTimer1-CH0, depending on the configuration, are reset to zero.

To wait for the rotor to stabilize in an aligned position, a certain time period is selected during which the DC voltage is constantly applied. This time and amplitude of DC voltage can be modified in the INIT state. Timing is implemented using a software counter that counts from a predefined value down to zero. During this time, event remains set to event=e_align. When the counter reaches zero, the counter is reset back to the predefined value, timers eTimer0-CH0 and/or eTimer1-CH0, depending on the configuration, are reset, and event is automatically set to event=e_align_done. This enables transition to the RUN state.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting event to event=e_app_off, which is done automatically on a falling edge of switchAppOnOff=false using FreeMASTER.

5.10 State – RUN

In the RUN state, all calculations for the FOC algorithm as described in PMSM Field-Oriented Control are performed. Calculation of the fast current loop is executed on every CTU-ADC interrupt when in the RUN state. Calculation of the slow speed loop is executed on every Nth CTU-ADC interrupt. Arbitration is done using a counter that counts from the value N down to zero. When zero is reached, the counter is reset back to N and the slow speed loop calculation is performed. In this way, only one interrupt is needed for both loops and the timing of both loops is synchronized, that is, slow loop calculations are finished before entering the fast loop calculations.

Figure 21 shows the implementation of the FOC algorithm and the functions and variables used. As shown in the diagram in Figure 21, the position/speed measurement is prepared for an encoder sensor. The encoder sensor feedback is selected as default.

Transition to the FAULT state is performed automatically when a fault occurs.

Transition to the INIT state is performed by setting event to event=e_app_off, which is done automatically on a falling edge of switchAppOnOff=false using FreeMASTER.

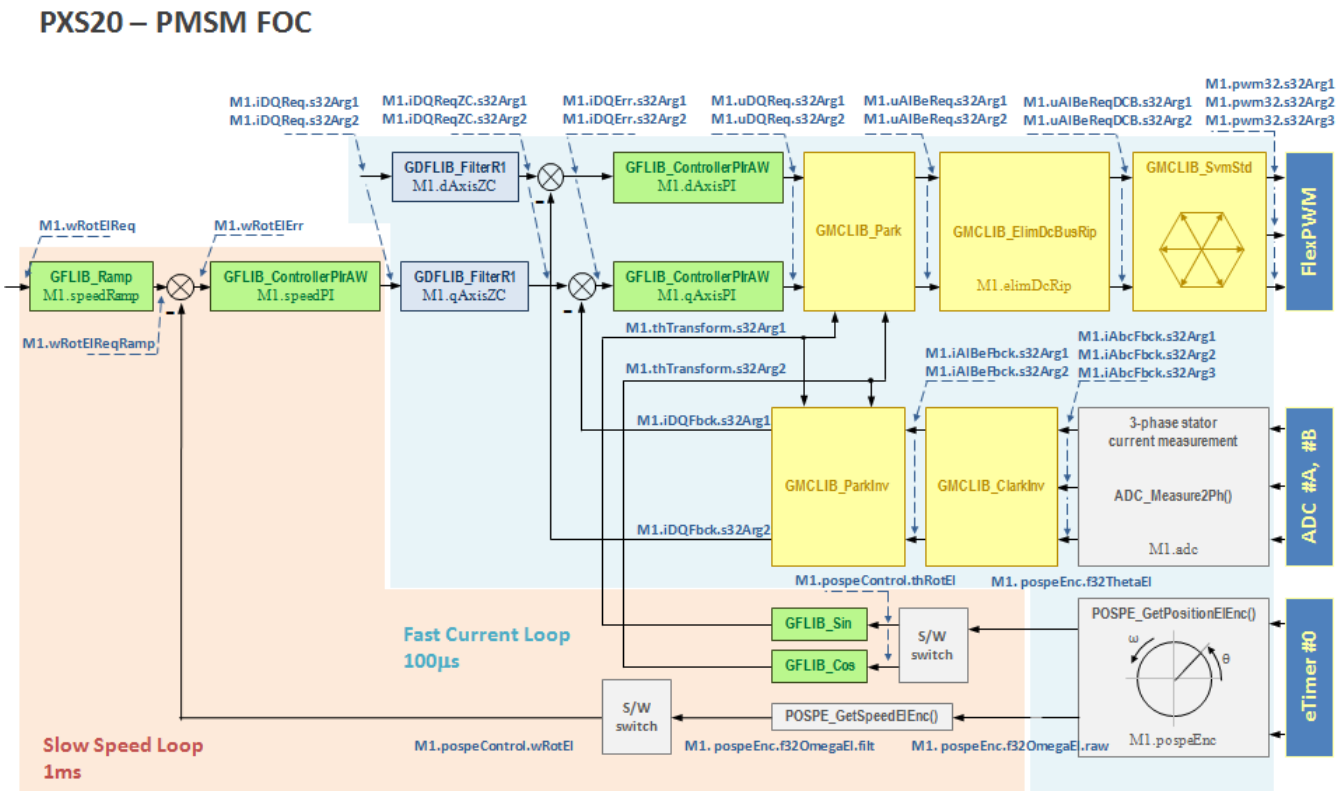

Figure 21. Variables and functions as implemented in the FOC calculation

The variables and functions shown in Figure 21 represent the cascade control structure for motor #1 whose data structure is named M1. For FOC for motor #2, all variables and functions have the same name but prefixed by an M2, which is taken from the name of the data structure M2 for motor #2.

5.10.1 Current measurement

Three phase currents are obtained by calling the ADC_Measure2Ph() function. As described in Phase current measurement, only two currents are measured at a time and the third current is calculated. Which currents are measured and which are calculated depends on the sector in which lies the actual output voltage vector. The sector is calculated by the GMCLIB_SvmStd() function, which generates the three phase duty-cycles for the inverter by employing a Space Vector Modulation technique.

The 3-phase inverter can switch six active voltage vectors and two zero vectors. These are given by combinations of the corresponding power switches. Plotting all six active vectors in a complex plane results in a hexagon with six sectors.

5.10.2 Position/speed measurement

The information on the rotor position is obtained by calling function POSPE_GetPosElEnc() if an encoder is used as a position sensor. Both functions calculate the angle tracking observer, which is implemented using the PI controller and an integrator. Output of the integrator represents the estimated tracked position, which is subtracted from the measured position from the sensor and the resulting difference is used as an error signal for the PI controller. Because the integrator is connected to the PI controller output, the PI controller output represents the estimated angular rotor velocity.

To obtain the information on the rotor position/speed, the parameters of the tracking observer, that is, the Kp and Ki gains of the PI controller, and the input and output scales of the integrator, must be properly configured.

5.10.3 Encoder sensor

If an encoder is used as a position sensor, the information on the rotor position is obtained by reading the value of the selected timer. In a single PMSM configuration, two encoder/timer peripheries are available and each of them can be used for encoder signal processing. The encoder/timer peripheries are used as shown in Figure 22. For timer configurations, see eTimer0.

Because the encoder is a relative position sensor, the particular counter/timer used for counting the encoder edges must be correctly reset at the start of an operation. This is done during the ALIGN state, where aligned rotor position is declared as a zero position and the counters eTimer0-CH0 and/or eTimer1-CH0, depending on the configuration, are set to zero.

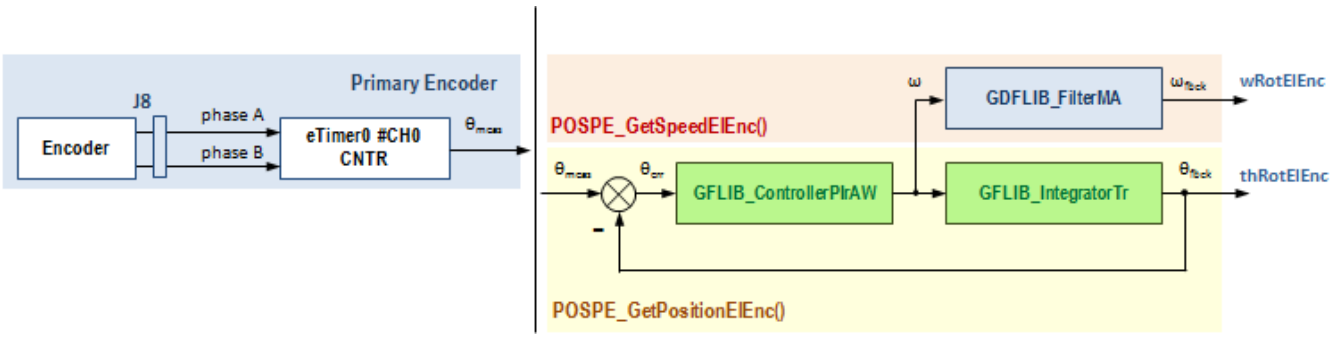

Figure 22. Angle tracking observer used for position/speed estimation using encoder

6 Application Control User Interface

To control the application and monitor variables during run time, the Freescale run-time debugging tool FreeMASTER from www.freescale.com is used.

Communication with the host PC is via USB. However, because FreeMASTER supports RS232 communication, the Serial to USB Bridge on the TWR-PXS2010 is used. See the board Quick Start Guide for more information. This provides a virtual COM port on the PC which can then be used for FreeMASTER communication.

The application configures the LINFlex module of the PXS20 for a communication speed of 19200bps. Therefore, FreeMASTER must also be set up for this speed. This can be done in the FreeMASTER menu \Project>Options> and selecting the tag Comm.

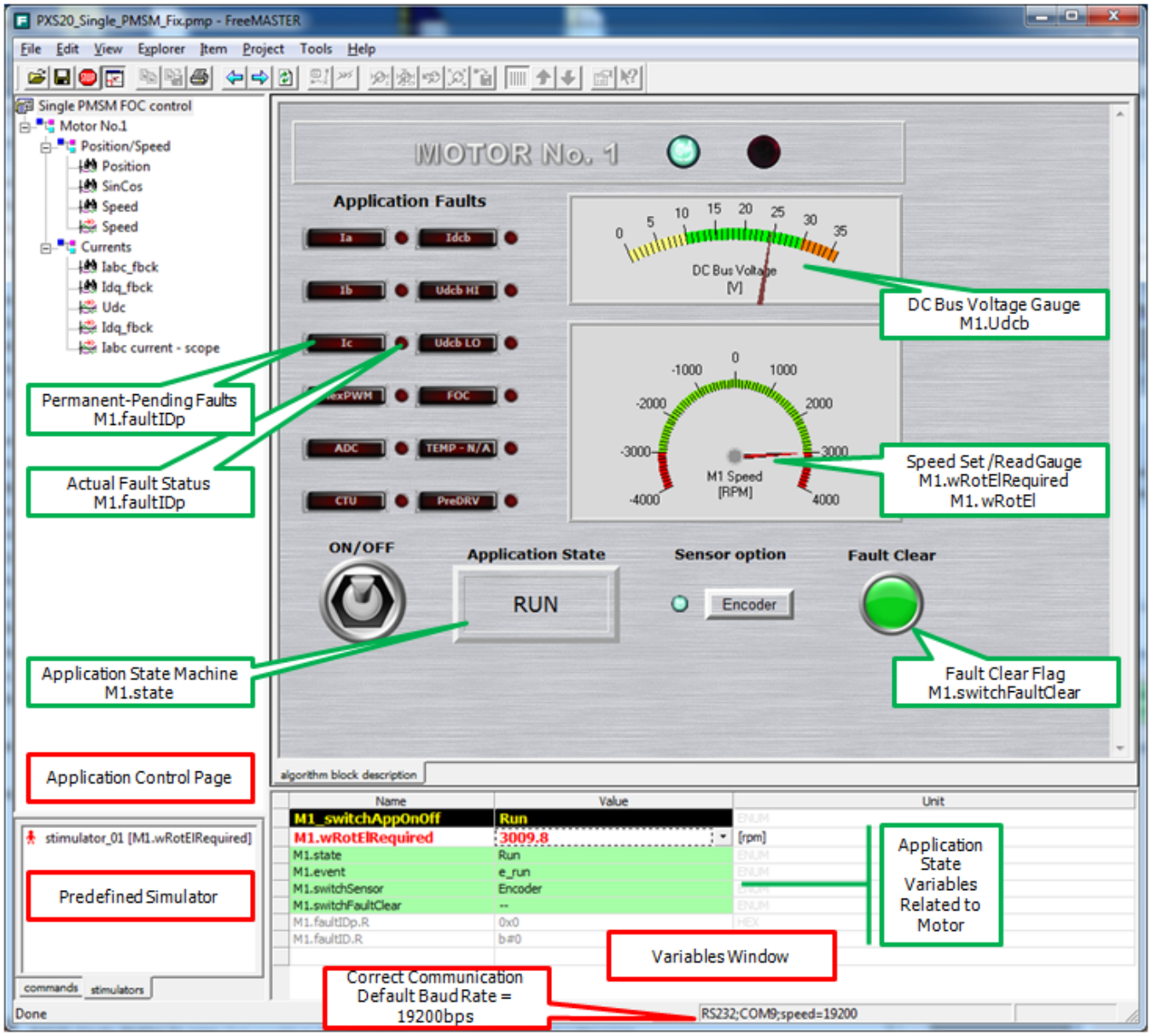

Figure 23. FreeMASTER Control Page for controlling the application

All application state machine variables can be seen on the FreeMASTER control page, as shown in Figure 23. Permanent/ pending faults are signaled by a highlighted red bar with the name of the fault source. The actual faults are signaled by the circular LED-like indicator, which is placed next to the bar with the name of the fault source. The actual presence of any fault is signaled when the respective LED-like indicator is lit. In Figure 23, for example, it can be seen that there are three pending faults and three actual faults. “Ia, Ib, Ic” - phase currents of the first motor are over the maximum current limit. That means there is an over current still present in the system. When pending faults are highlighted but actual faults are off, an error has been latched but it is not present in the system anymore. In that case, the application state FAULT shall be displayed and indicated by a pink frame indicator. The FAULT state is also indicated by a red LED-like indicator in the header bar. If a fault state occurs for whatever reason, the application is automatically switched off.

When all actual fault sources have been removed, none of the LED-like fault indicators will be lit, and pending faults can be cleared by pressing the green circled button named FAULT CLEAR. This will clear all pending faults and will enable transition of the state machine into the INIT and then the READY states. Because INIT is a one-pass state, transitions FAULT-INIT-READY happen so fast that it is not visible on the control page. Instead, it seems as if the state machine went from FAULT to READY directly. This is not an error, but is caused by slow communication via RS232 and/or slow refresh rate of the control page.

When the application faults have been cleared and application is in the READY state, all variables must be set to their default values. Now the application can be started by selecting ON, the upper position on the application On/Off switch, which is a flip/flop type of switch. Successful selection is signaled by the green LED-like indicator in the header bar.

If there is no fault detected in the system, after starting the application by selecting ON with the flip/flop switch, the application should proceed to the CALIB state for DC offset calibration, then to the ALIGN state for marking the zero position, and the state RUN shall be entered.

When in the RUN state, all control loops of the FOC algorithm are active. That means, three phase currents are measured and used to close the current loop and the actual rotor speed is measured for closing the speed loop. The user can now select the desired speed of the rotor in the variable watch window. The variable for controlling the speed is called Nreq, which is the required speed recalculated to the mechanical speed in revolutions per minute.

The application can be switched off by selecting OFF, the lower position on the application On/Off switch, and should proceed to the INIT state and consequently to the READY state. Both switches can be controlled independently.

Because of the motor used, the required speed can be selected from -3000[rpm] to 3000[rpm].

6.1 Application quick start

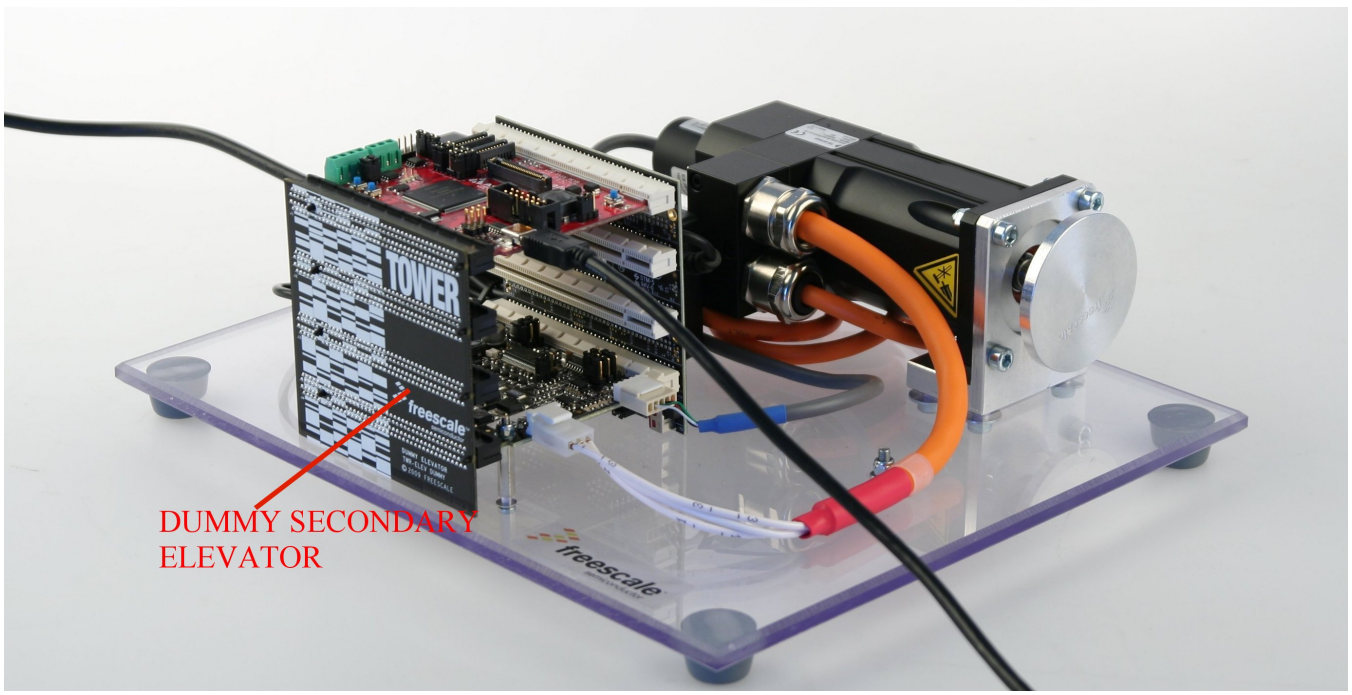

Figure 24. Hardware setup with TGT2-0032-30-24 motor

To get started with this application:

- Compose the hardwareTOWER system with the TWR-PXS2010, TWR-MC-LV3PH primary elevator and secondary dummy elevator . The second elevator does not need to be connected.

NOTE

The TWR-MC-LV3PH board needs to be modified: Primary connector B56 wired to DRV_OC (R90 removed). The SECONDARY DUMMY elevator should be used for correct operation of the Power Stage board with the TWR-PXS2010 (the SECONDARY elevator needs to be REPLACED with the DUMMY elevator, or not installed)

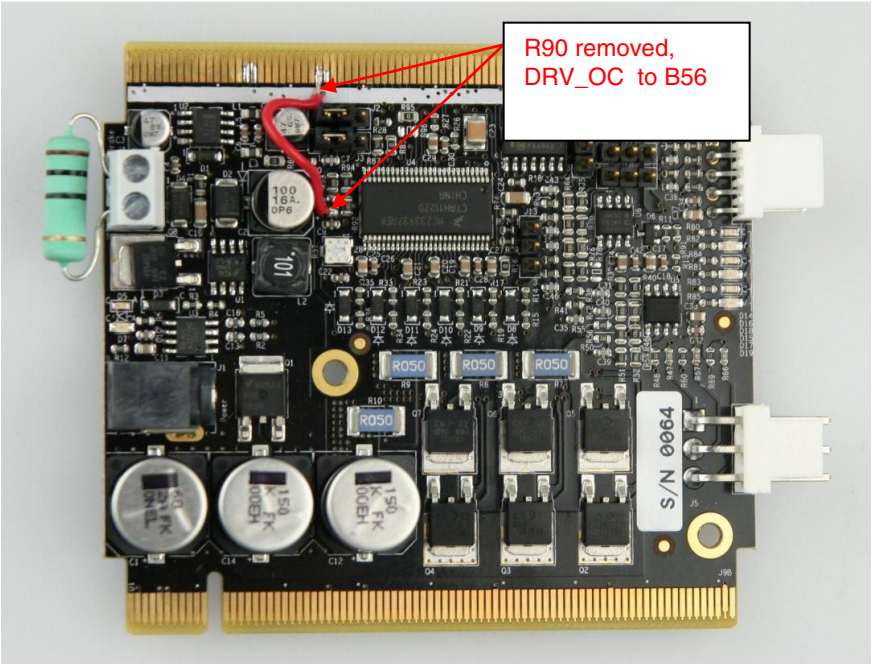

Figure 25. TWR-MC-LV3PH board with required modification

1. Connect the motor TGT2-0032-30-24/TOPS1KX-1M (and Inducoder ES 28-6-1024-05-D-R encoder).

NOTE

The software attached with this application note is set up to use the TGT2-0032-30-24/TOPS1KX-1M (with encoder Inducoder ES 28-6-1024-05-D-R) motor from an external vendor (therefore it does not use the motor LINIX 45WN24-40 from the TWR-MC-LV3PH kit). The TGT2-0032-30-24/ TOPS1KX-1M motor parameters:

- 0.32 Nm
- 30 V
- 5.2 A
- 3000 rpm

(See ww.tgdrives.cz/en for detail.) The encoder Inducoder ES 28-6-1024-05-D-R has 1024 impulses.

2. Program the application software to the microcontroller (the software PXS20_PMSM_Development_Kit_Fix_CW10_2.exe can be programmed from CodeWarrior Development Studio 10.2 via JTAG interface, for example, using USB PowerPC Nexus Multilink or other debugging hardware (see the TWR-PXS2010 manual).
3. Install the FreeMASTER software, located at www.freescale.com.
4. Connect the USB cable to the TWR-PXS2010 controller board and to the host PC.
5. Connect the power supply to the TWR-MC-LV3PH power-stage. The 5 V controller board power supply is taken from the TWR-MC-LV3PH power stage via the PRIMARY ELEVATOR board (VDD_5V_ELEV—see the TWR-PXS2010 manual). The PMSM motors are designed for DC-bus voltage = 24 V, which is suitable for the TWR-MC-LV3PH but not for the controller board.
6. Start the FreeMASTER project located in \ FreeMASTER_control\ PXS20_Single_PMSM_Fix.pmp
7. Ensure that the correct virtual port, COMx is selected (according to your PC) and communication speed 19200 Baud is selected in the FreeMASTER Project/Options window.
8. Enable communication by pressing the STOP button in the toolbar in FreeMASTER, or by pressing CTRL+K.
9. The successful communication is signaled in the status bar. See Figure 23 for an example.
10. If no actual faults are present in the system, then all of the LED-like indicators will be dark red. If there is a fault present, identify the source of the fault and remove it. Successful removal is signaled by the switching off of the respective LED-like indicator.

11. If all of the LED-like indicators are off, clear the pending faults by pressing the green circular button FAULT CLEAR.
12. Start the application by pressing ON, theupper position on the flip/flop ON/OFF switch.
13. Enter the required speed by clicking on the speed gauge, or by assigning the value to the M1.wRotElRequired variable in the variables watch window. The value is in revolutions per minute.
14. Stop the application by pressing OFF, the lower position on the flip/flop ON/OFF switch.

7 References

1. Quick Start Guide - Dual Core Power Architecture MCU for Industrial Control and Safety Applications, PXS20QSG REV 0, Freescale Semiconductor 2011
2. TWR-MC-LV3PH User's Manual, Freescale Semiconductor 2009
3. TWR-MC-LV3PH Quick Setup Guide - TWRMCLV3PHQSG, Freescale Semiconductor 2009
4. www.tgdrives.cz/en
5. www.inducoder.de/dat_en/datsheet/es28.shtml
6. www.freescale.com/codewarrior

8 Conclusion

This application note describes a PMSM control application using the Freescale PXS20 Power Architecture MCU together with the TOWER hardware system based on the TWR-PXS2010 control board. In the future, a newer PXS20 controller board might be available. Some details might differ from this application note. Search the board manuals for such details.

9 Revision history

Table 2. Revision history

Revision number	Date	Description
0	July 2012	Initial version.
1	October 2012	Updated information in section "Application quick start" and in "References."

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: AN4537
Rev. 1, 10/2012