Freescale Semiconductor
Application Note

AN2973
Rev. 0, 04/2006

Using the ACIM Vector Control eTPU Function

Covers the MCF523*x*, MPC5500, and all eTPU-Equipped Devices

by: Milan Brejl
System Application Engineer, Roznov Czech System Center
Michal Princ
System Application Engineer, Roznov Czech System Center

Table of Contents

1 Introduction

The AC induction motor vector control (ACIMVC) enhanced time processor unit (eTPU) function is one of the functions included in the AC motor control eTPU function set (set4). This eTPU application note is intended to provide simple C interface routines to the ACIMVC eTPU function. The routines are targeted at the MCF523*x* and MPC5500 families of devices, but they can easily be used with any device that has an eTPU.

2 Theory

Vector control is an elegant control method of controlling the AC induction motor (ACIM), where field oriented theory is used to control space vectors of magnetic flux, current, and voltage. It is possible to set up the co-ordinate system to decompose the vectors into a electro-magnetic field generating part and a torque generating part. Then the structure of the motor controller

freescale semiconductor

(vector control controller) is almost the same as for a separately excited DC motor, which simplifies the control of ACIM. This vector control technique was developed in the past especially to achieve similar excellent dynamic performance of ACIM.
As explained in Figure 1, the choice has been made of a widely used current control with an inner position closed loop. In this method, the decomposition of the field generating part and the torque generating part of the stator current allows separate control of the magnetic flux and the torque. To do so, we need to set up the rotary co-ordinate system connected to the rotor magnetic field. This co-ordinate system is generally called the 'd-q reference co-ordinate system.' All transformations needed for vector control are described here.

2.1 Mathematical Model of ACIM Control

For a description of the ACIM, the symmetrical three-phase smooth-air-gap machine with sinusoidally distributed windings is considered. Then the voltage equations of stator in the instantaneous form can be expressed as:

$$u_{SA} = R_S i_{SA} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{SA} \quad \textit{Eqn. 1}$$

$$u_{SB} = R_S i_{SB} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{SB} \quad \textit{Eqn. 2}$$

$$u_{SC} = R_S i_{SC} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{SC} \quad \textit{Eqn. 3}$$

where u_{SA}, $u_{SB,}$ and u_{SC} are the instantaneous values of stator voltages, i_{SA}, i_{SB} and i_{SC} are the instantaneous values of stator currents, and Ψ_{SA}, Ψ_{SB}, and Ψ_{SC} are instantaneous values of stator flux linkages in phase A, B, and C.
Due to the large number of equations in the instantaneous form of Eqn. 1, Eqn. 2 and Eqn. 3, it is more practical to rewrite the instantaneous equations using two axis theory (Clark transformation). The ACIM can then be expressed using the following equations:

- The stator voltage differential equations

$$u_{S\alpha} = R_S i_{S\alpha} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{S\alpha} \quad \textit{Eqn. 4}$$

$$u_{S\beta} = R_S i_{S\beta} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{S\beta} \quad \textit{Eqn. 5}$$

- The rotor voltage differential equations

$$u_{R\alpha} = 0 = R_R i_{R\alpha} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{R\alpha} + \omega\Psi_{R\beta} \quad \textit{Eqn. 6}$$

$$u_{R\beta} = 0 = R_R i_{R\beta} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{R\beta} - \omega\Psi_{R\alpha} \quad \textit{Eqn. 7}$$

- The stator and rotor flux linkages expressed in terms of the stator and rotor current space vectors

$$\Psi_{S\alpha} = L_S i_{S\alpha} + L_m i_{R\alpha} \quad \textit{Eqn. 8}$$

$$\Psi_{S\beta} = L_S i_{S\beta} + L_m i_{R\beta} \quad \textit{Eqn. 9}$$

$$\Psi_{R\alpha} = L_R i_{R\alpha} + L_m i_{S\alpha} \quad \textit{Eqn. 10}$$

$$\Psi_{R\beta} = L_R i_{R\beta} + L_m i_{S\beta} \quad \textit{Eqn. 11}$$

- Electromagnetic torque expressed by utilizing space vector quantities

$$t_e = \frac{3}{2} p_p (\Psi_{S\alpha} i_{S\beta} - \Psi_{S\beta} i_{S\alpha}) \quad \textit{Eqn. 12}$$

where:	$\alpha,\beta=$	Stator orthogonal coordinate system	
	$u_{S\alpha,\beta=}$	Stator voltages	[V]
	$i_{S\alpha,\beta=}$	Stator currents	[A]
	$u_{R\alpha,\beta=}$	Rotor voltages	[V]
	$i_{R\alpha,\beta=}$	Rotor currents	[A]
	$\Psi_{S\alpha,\beta=}$	Stator magnetic fluxes	[Vs]
	$\Psi_{R\alpha,\beta=}$	Rotor magnetic fluxes	[Vs]
	$R_{S=}$	Stator phase resistance	[Ohm]
	$R_{R=}$	Rotor phase resistance	[Ohm]
	$L_{S=}$	Stator phase inductance	[H]
	$L_{R=}$	Rotor phase inductance	[H]
	$L_{m=}$	Mutual (stator to rotor) inductance	[H]
	$\omega / \omega_{s=}$	Electrical rotor speed / synchronous speed	[rad/s]
	$p_{p=}$	Number of pole pairs	[-]
	$t=_e$	Electromagnetic torque	[Nm]

Equations Eqn. 4 through Eqn. 12 represent the model of a ACIM in the stationary frame α, β fixed to the stator. The main purpose of the vector control is to decompose the vectors into a magnetic field generating part and a torque generating part. To do so, it is necessary to set up a rotary co-ordinate system attached to the rotor magnetic field. This coordinate system is generally called the ‘d-q-co-ordinate system.’ Thus the equations Eqn. 4 through Eqn. 12 can be rewritten as:

$$u_{Sd} = R_S i_{Sd} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{Sd} - \omega_s \Psi_{Sq} \quad \textit{Eqn. 13}$$

$$u_{Sq} = R_S i_{Sq} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{Sq} - \omega_s \Psi_{Sd} \quad \textit{Eqn. 14}$$

$$u_{Rd} = 0 = R_R i_{Rd} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{Rd} - (\omega_s - \omega)\Psi_{Rq} \quad \textit{Eqn. 15}$$

$$u_{Rq} = 0 = R_R i_{Rq} + \frac{\mathrm{d}}{\mathrm{d}t}\Psi_{Rq} + (\omega_s - \omega)\Psi_{Rd} \quad \textit{Eqn. 16}$$

$$\Psi_{Sd} = L_S i_{Sd} + L_m i_{Rd} \quad \textit{Eqn. 17}$$

$$\Psi_{Sq} = L_S i_{Sq} + L_m i_{Rq} \quad \textit{Eqn. 18}$$

$$\Psi_{Rd} = L_R i_{Rd} + L_m i_{Sd} \quad \textit{Eqn. 19}$$

$$\Psi_{Rq} = L_R i_{Rq} + L_m i_{Sq} \quad \textit{Eqn. 20}$$

$$t_e = \frac{3}{2} p_p (\Psi_{Sd} i_{Sq} - \Psi_{Sq} i_{Sd}) \quad \textit{Eqn. 21}$$

3 Function Overview

The purpose of the ACIMVC function is to perform the current control loop of a field-oriented (vector control) drive of a ACIM.

The sequence of ACIMVC calculations consists of these steps:

1. Forward Clarke Transformation
2. Rotor flux Estimation
3. DQ establishment (including calculation of sine, cos, omega_field, and Forward Park transformation)
4. D&Q current controllers calculation
5. Decoupling
6. Circle limitation
7. Inverse Park transformation
8. DC-bus ripple elimination

The ACIMVC calculates applied voltage vector components alpha and beta based on measured phase currents and required values of phase currents in 2-phase orthogonal rotating reference frame (D-Q). The ACIMVC function optionally enables to perform the limitation of calculated D and Q components of the stator voltages into the circle.

The ACIMVC does not generate any drive signal, and can be executed even on an eTPU channel not connected to an output pin. If connected to an output pin, the ACIMVC function turns the pin high and low, so that the high-time identifies the period of time in which the ACIMVC execution is active. In this way, the ACIMVC function, as with many of the motor-control eTPU functions, supports checking eTPU timing using an oscilloscope.

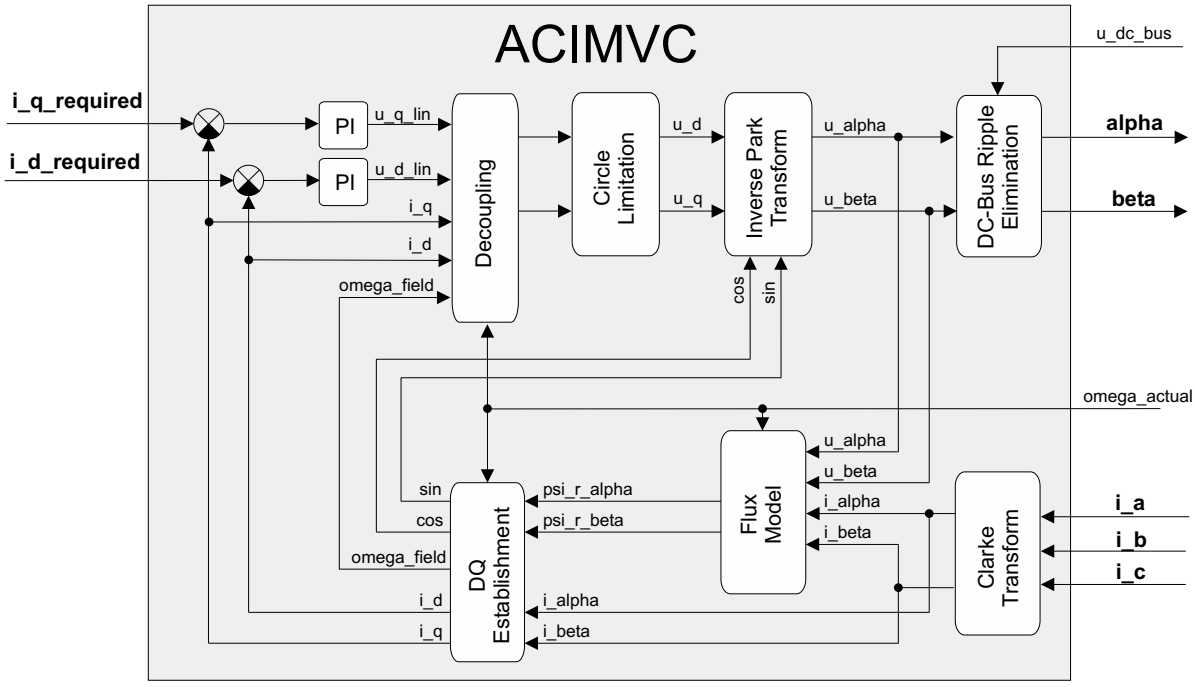

Figure 1. Functionality of ACIMVC

4 Function Description

The ACIMVC eTPU function performs the calculations of the vector control current loop in the following order:

- **Calculates Forward Clark Transformation.**

  The Forward Clark Transformation transforms a three-phase system into a two-phase orthogonal system.

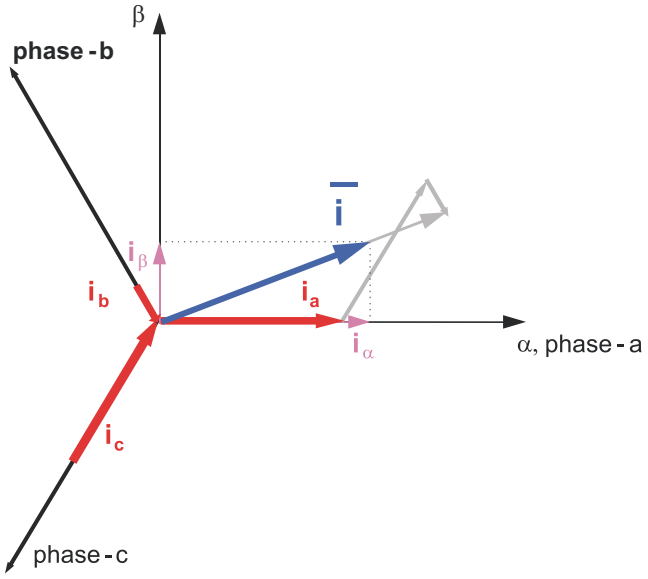

Figure 2. Clark Transformation

In most cases, the 3-phase system is symmetrical, which means that the sum of the phase quantities is always zero. To transfer the graphical representation into mathematical language:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\frac{2}{3} & -\frac{1}{3} & -\frac{1}{3}\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} = \mid\ a+b+c=0\ \mid = \begin{bmatrix}1 & 0 & 0\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} \qquad \textit{Eqn. 22}$$

The ACIMVC uses the Clark Transformation to transform the phase currents:

$$i\_alpha = i\_a$$

$$i\_beta = 1/sqrt(3) \times i\_b - 1/sqrt(3) \times i\_c = 1/sqrt(3) \times (i\_b - i\_c)$$

- **Estimates Rotor Flux.**

  Knowledge of the rotor flux space vector magnitude and position is key information for the AC induction motor vector control. With the rotor magnetic flux space vector, the rotational coordinate system (d-q) can be established. There are several methods for obtaining the rotor magnetic flux space vector. The implemented flux model utilizes monitored rotor speed and stator voltages and currents. It is calculated in the stationary reference frame (α, β) attached to the stator. The error in the calculated value of the rotor flux, influenced by the changes in temperature, is negligible for this rotor flux model.

The flux model calculates the two axis components of the rotor magnetic flux (Ψ_R) in alpha, beta stationary reference frame. The flux calculation is based on the AC induction motor mathematical model. The function solves the system of two differential equations:

$$[(1-\sigma)T_S + T_R]\frac{d\Psi_{R\alpha}}{dt} = \frac{L_m}{R_S}u_{S\alpha} - \Psi_{R\alpha} - p_p\omega T_R\Psi_{R\beta} - \sigma L_m T_S\frac{di_{S\alpha}}{dt} \quad \textit{Eqn. 23}$$

$$[(1-\sigma)T_S + T_R]\frac{d\Psi_{R\beta}}{dt} = \frac{L_m}{R_S}u_{S\beta} - \Psi_{R\beta} + p_p\omega T_R\Psi_{R\alpha} - \sigma L_m T_S\frac{di_{S\beta}}{dt} \quad \textit{Eqn. 24}$$

where:

$L_{S=}$	Self-inductance of the stator	[H]
$L_{R=}$	Self-inductance of the rotor	[H]
$L_{m=}$	Magnetizing inductance	[H]
$R_{R=}$	Resistance of a rotor phase winding	[Ohm]
$R_{S=}$	Resistance of a stator phase winding	[Ohm]
$\omega=$	Angular rotor speed	[rad.s^{-1}]
p_p	Number of motor pole-pairs	[-]
$T_R = \frac{L_R}{R_R}$	Rotor time constant	[s]
$T_S = \frac{L_S}{R_S}$	Stator time constant	[s]
$\sigma = 1 - \frac{L_m^{\ 2}}{L_S L_R}$	Resultant leakage constant	[-]

$u_{S\alpha}, u_{S\beta}, i_{S\alpha}, i_{S\beta}, \Psi_{R\alpha}, \Psi_{R\beta}$ are the α, β components of the stator voltage, currents, and rotor flux space vectors.

Eqn. 23 and Eqn. 24 can be rewritten as follows:

$$\frac{d\Psi_{R\alpha}}{dt} = (KLRKT)u_{S\alpha} - (IKT)\Psi_{R\alpha} - (TRKT)\omega\Psi_{R\beta} - (KLTKT)\frac{di_{S\alpha}}{dt} \quad \textit{Eqn. 25}$$

$$\frac{d\Psi_{R\beta}}{dt} = (KLRKT)u_{S\beta} - (IKT)\Psi_{R\beta} + (TRKT)\omega\Psi_{R\alpha} - (KLTKT)\frac{di_{S\beta}}{dt} \quad \textit{Eqn. 26}$$

where:

$$KLRKT = \frac{L_m}{R_S[(1-\sigma)T_S + T_R]}$$

$$IKT = \frac{1}{[(1-\sigma)T_S + T_R]}$$

$$TRKT = \frac{T_R p_p}{[(1-\sigma)T_S + T_R]}$$

$$KLTKT = \frac{\sigma L_m T_S}{[(1-\sigma)T_S + T_R]}$$

The numeric method of integration is based on the trapezoidal method with prediction. The prediction is implemented using the Euler method. There are two steps of the numerical integration. Lets define:

$$dx.alpha = KL\_R\_KT \times u.alpha - I\_KT \times psi\_r.alpha - TR\_KT \times psi\_r.beta \times omega\_actual$$

$$dx.beta = KL\_R\_KT \times u.beta - I\_KT \times psi\_r.beta + TR\_KT \times psi\_r.alpha \times omega\_actual$$

First step - prediction:

$$psi\_r\_pred.alpha = x.alpha[k\text{-}1] + T \times dx\_alpha[k\text{-}1] - KL\_T\_KT \times i\_alpha$$
$$psi\_r\_pred.beta = x.beta[k\text{-}1] + T \times dx\_beta[k\text{-}1] - KL\_T\_KT \times i\_beta$$

where T is the time step of the numeric integration.

Second step - trapezoidal method:

$$dx\_pred.alpha = KL\_R\_KT \times u.alpha - I\_KT \times psi\_r\_pred.alpha - TR\_KT \times psi\_r\_pred.beta \times omega\_actual$$
$$dx\_pred.beta = KL\_R\_KT \times u.beta - I\_KT \times psi\_r\_pred.beta + TR\_KT \times psi\_r\_pred.alpha \times omega\_actual$$
$$x.alpha = x.alpha[k\text{-}1] + T \times (dx.alpha[k\text{-}1] + dx\_pred.alpha) / 2$$
$$x.beta = x.beta[k\text{-}1] + T \times (dx.beta[k\text{-}1] + dx\_pred.beta) / 2$$

Final step - Flux calculation:

$$psi\_r.alpha = x.alpha - KL\_T\_KT \times i.alpha$$
$$psi\_r.beta = x.beta - KL\_T\_KT \times i.beta$$

- **Establishes DQ-coordinate system**
  - **Calculate magnetic flux modulo.**

    The two axis components of the rotor magnetic flux in α, β stationary reference frame are used for the determination of the rotor magnetic flux magnitude Ψ_{Rd}.

$$\Psi_{Rd} = \sqrt{\Psi_{R\alpha}^2 + \Psi_{R\beta}^2} \qquad \textit{Eqn. 27}$$

  - **Calculate sin(theta_field) and cos(theta_field).**

    Based on the rotating magnetic flux space-vector modulo and the two axis components of the rotor magnetic flux in α, β stationary reference frame the position angle of the rotating magnetic flux space-vector is calculated.

$$\sin\vartheta_{Field} = \frac{\Psi_{R\beta}}{\Psi_{Rd}}$$
$$\cos\vartheta_{Field} = \frac{\Psi_{R\alpha}}{\Psi_{Rd}} \qquad \textit{Eqn. 28}$$

  - **Calculates Forward Park Transformation.**

    The Forward Park Transformation transforms a two-phase stationary system into a two-phase rotating system.

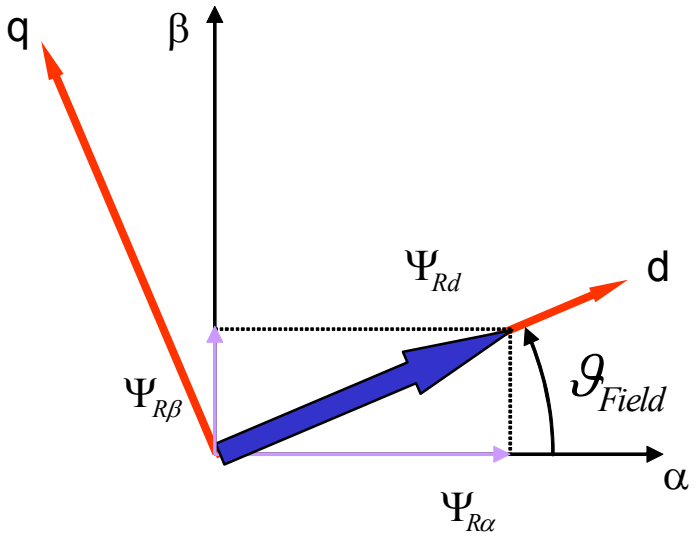

Figure 3. Park Transformation

To transfer the graphical representation into mathematical language:

$$\begin{bmatrix} d \\ q \end{bmatrix} = \begin{bmatrix} \cos\vartheta_{Field} & \sin\vartheta_{Field} \\ -\sin\vartheta_{Field} & \cos\vartheta_{Field} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \textit{Eqn. 29}$$

The ACIMVC uses the Park Transformation to transform the phase currents:

$$i\_d = i\_alpha \times cos(theta\_field) + i\_beta \times sin(theta\_field)$$
$$i\_q = -i\_alpha \times sin(theta\_field) + i\_beta \times cos(theta\_field)$$

— **Calculate synchronous speed.**

The synchronous speed of the stator magnetic flux vector is calculated as follows:

$$\omega_s = \frac{L_m i_{Sq}}{T_R p_p \Psi_{Rd}} + \omega \quad \textit{Eqn. 30}$$

where:

i_{Sq} =	Q component of the stator currents in 2-phase orthogonal rotating ref. frame	
L_m=	Magnetizing inductance	[H]
L_R=	Self-inductance of the rotor	[H]
R_R=	Resistance of a rotor phase winding	[Ohm]
$T_R = \frac{L_R}{R_R}$ =	Rotor time constant	[s]
p_p=	Number of pole pairs	[-]
Ψ_{Rq} =	Magnetic flux modulo	[Vs]
ω=	Angular rotor speed	[rad.s^{-1}]

The ACIMVC calculates synchronous speed as follows:

$$omega\_field = LM\_TR \times i\_q / psi\_r\_d + omega\_actual$$

- **D-coordinate and Q-coordinate PID controllers.**

The PID algorithm in continuous time domain can be expressed by the following equation:

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^{\tau} e(\tau)d\tau + T_D\frac{de(t)}{dt}\right] \quad \textit{Eqn. 31}$$

where

$u(t)$ – PID controller output at time t
$e(t)$ – Input error at time t
K – PID controller gain
T_I – Integral time constant
T_D – Derivative time constant

The PID algorithm in discrete time domain can be expressed by the following equation:

$$u(k) = Ke(k) + K\frac{T}{T_I}e(k) + u(k-1) + K\frac{T_D}{T}(e(k) - e(k-1)) \quad \textbf{Eqn. 32}$$

where

$u(k)$ – PID controller output in step k
$u(k\text{-}1)$ – PID controller output in step $k\text{-}1$
$e(k)$ – Input error in step k
$e(k\text{-}1)$ – Input error in step $k\text{-}1$
T – Update period

The ACIMVC PID controller algorithm calculates the output according to the following equations:

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$
$$e(k) = w(k) - m(k)$$
$$u_P(k) = G_P \times e(k)$$
$$u_I(k) = u_I(k\text{-}1) + G_I \times e(k)$$
$$u_D(k) = G_D \times (e(k) - e(k\text{-}1))$$

Where:

$u_P(k)$ – Proportional portion in step k
$u_I(k)$ – Integral portion in step k
$u_D(k)$ – Derivative portion in step k
$w(k)$ – Desired value in step k
$m(k)$ – Measured value in step k
G_P – Proportional gain $G_P = K$
G_I – Integral gain $G_I = K \times T/T_I$
G_D – Derivative gain $G_D = K \times T_D/T$

If the derivative gain is set to 0, an internal flag that enables the calculation of derivative portion is cleared, resulting in a shorter calculation time. Then controller becomes a PI-type controller.

The measured and desired values, as well as the gains, are applied with 24-bit precision. The integral portion is stored with 48-bit precision. The gain range is from 0 to 256, with a precision of 0.0000305 (30.5e-6).

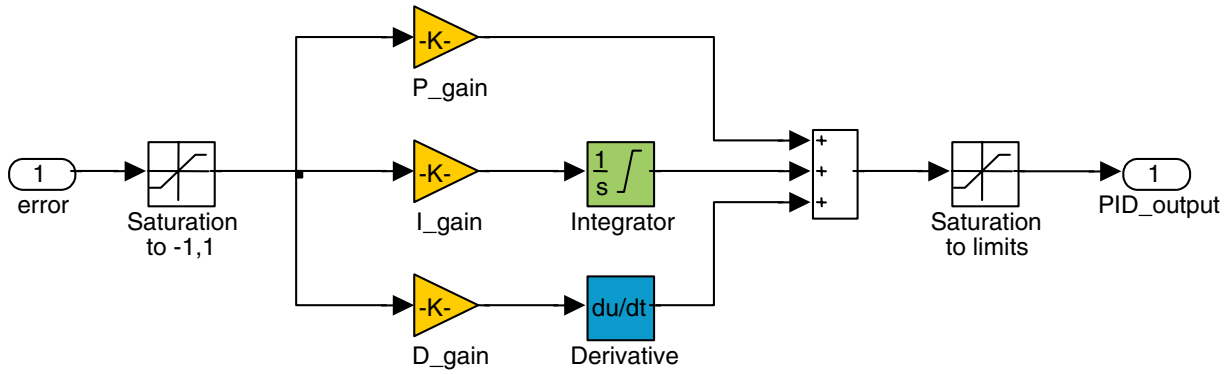

Figure 4. PID Controller Structure

The ACIMVC uses the PID controller to control the D and Q coordinates of the applied motor voltage vector, based on the error between the required and the actual D and Q phase currents:

u_d = PID_controller (i_d_required - i_d)

u_q = PID_controller (i_q_required - i_q)

- **Decoupling Circuit.**

  For purposes of the rotor flux-oriented vector control, the direct-axis stator current i_{Sd} (rotor flux-producing component) and the quadrature-axis stator current i_{Sq} (torque-producing component) must be controlled independently. However, the equations of the stator voltage components are coupled. The direct axis component u_{Sd} also depends on $i_{Sq,}$ and the quadrature axis component u_{Sq} also depends on i_{Sd}. The stator voltage components u_{Sd} and u_{Sq} cannot be considered as decoupled control variables for the rotor flux and electromagnetic torque. The stator currents i_{Sd} and i_{Sq} can only be independently controlled (decoupled) if the stator voltage equations are decoupled and the stator current components i_{Sd} and i_{Sq} are indirectly controlled by controlling the terminal voltages of the induction motor.

  The equations of the stator voltage components in the d-q coordinate system (Eqn. 13 and Eqn. 14) can be reformulated and separated into two components: linear components $u_{Sd}^{lin}, u_{Sq}^{lin}$ and decoupling components $u_{Sd}^{decouple}, u_{Sq}^{decouple}$. The equations are decoupled as follows:

$$u_{Sd} = u_{Sd}^{lin} + u_{Sd}^{decouple} = \left[K_R i_{Sd} + K_L \frac{\mathrm{d}}{\mathrm{d}t} i_{Sd}\right] - \left[\omega_s K_L i_{Sq} + \frac{\Psi_{Rd} L_m}{L_R T_R}\right] \quad \textit{Eqn. 33}$$

$$u_{Sq} = u_{Sq}^{lin} + u_{Sq}^{decouple} = \left[K_R i_{Sq} + K_L \frac{\mathrm{d}}{\mathrm{d}t} i_{Sq}\right] + \left[\omega_s K_L i_{Sd} + \frac{L_m}{L_R} \omega \Psi_{Rd}\right] \quad \textit{Eqn. 34}$$

  where:

$$K_R = R_s + \frac{L_m^2}{L_R^2} R_R \quad \textit{Eqn. 35}$$

$$K_L = L_S - \frac{L_m^2}{L_R} \quad \textit{Eqn. 36}$$

  The voltage components $u_{Sd}^{lin}, u_{Sq}^{lin}$ are the outputs of the current controllers which control i_{Sd} and i_{Sq} components. They are added to the decoupling voltage components $u_{Sd}^{decouple}, u_{Sq}^{decouple}$. In this way, we can get direct and quadrature components of the terminal output voltage. This means the voltage on the outputs of the current controllers is:

$$u_{Sd}^{lin} = K_R i_{Sd} + K_L \frac{\mathrm{d}}{\mathrm{d}t} i_{Sd} \quad \textit{Eqn. 37}$$

$$u_{Sq}^{lin} = K_R i_{Sq} + K_L \frac{\mathrm{d}}{\mathrm{d}t} i_{Sq} \quad \textit{Eqn. 38}$$

And the decoupling components are:

$$u_{Sd}^{decouple} = -\left(\omega_s K_L i_{Sq} + \frac{L_m}{L_R T_R} \Psi_{Rd}\right) \quad \textit{Eqn. 39}$$

$$u_{Sq}^{decouple} = \left(\omega_s K_L i_{Sd} + \frac{L_m}{L_R} \omega \Psi_{Rd}\right) \quad \textit{Eqn. 40}$$

As can be seen, the decoupling algorithm transforms the non-linear motor model to linear equations, which can be controlled by general PI or PID controllers instead of complicated controllers.

The ACIMVC calculates the following in order to decouple the controllers output *u_d* and *u_q*:

$$u\_d = u\_d - KL \times omega\_field \times i\_q - LM\_LR\_TR \times psi\_r\_d$$

$$u\_q = u\_q + KL \times omega\_field \times i\_d + LM\_LR \times psi\_r\_d \times omega\_actual$$

Where:

u_d – D-coordinate of applied motor voltage
u_q – Q-coordinate of applied motor voltage
i_d – D-coordinate of phase currents
i_q – Q-coordinate of phase currents
omega_field – Field rotation speed
omega_actual – Actual motor electrical velocity
psi_r_d – Rotor flux modulo

- **Optionally limits D and Q components of the stator voltages into the circle.**

D and Q components of the stator voltages in 2-phase orthogonal rotating reference frame can be optionally limited into the circle. The process of limitation is described as follows:

$$vLim = \frac{u\_dc\_bus\_actual}{2 \cdot inv\_mod\_index}$$

$$u\_d = \begin{cases} u\_d & \text{if } -vLim < u\_d < vLim \\ vLim & \text{if } u\_d > vLim \\ -vLim & \text{if } u\_d < -vLim \end{cases}$$

$$u\_q\_tmp = \sqrt{(vLim)^2 - (u\_d)^2}$$

$$u\_q = \begin{cases} u\_q & \text{if } -u\_q\_tmp < u\_q < u\_q\_tmp \\ u\_q\_tmp & \text{if } u\_q > u\_q\_tmp \\ -u\_q\_tmp & \text{if } u\_q < -u\_q\_tmp \end{cases}$$

- **Calculates Backward Park Transformation.**

The Backward Park Transformation transforms a two-phase rotating system into a two-phase stationary system.

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \cos\vartheta_{Field} & -\sin\vartheta_{Field} \\ \sin\vartheta_{Field} & \cos\vartheta_{Field} \end{bmatrix} \begin{bmatrix} d \\ q \end{bmatrix} \quad \textit{Eqn. 41}$$

The ACIMVC uses the Backward Park Transformation to transform the motor voltages:

$$u\_alpha = u\_d \times cos(theta\_field) - u\_q \times sin(theta\_field)$$
$$u\_beta = u\_d \times sin(theta\_field) + u\_q \times cos(theta\_field)$$

- **Eliminates DC-bus ripples.**

The ripple elimination process compensates an amplitude of the direct-α and the quadrature-β component of the stator reference voltage vector for imperfections in the DC bus voltage. These imperfections are eliminated by the formula shown in the following equations:

$$u\_alpha = \begin{cases} \dfrac{inv\_mod\_index \cdot u\_alpha}{u\_dc\_bus\_actual/2} & \text{if } |inv\_mod\_index \cdot u\_alpha| < \dfrac{u\_dc\_bus\_actual}{2} \\ sign(u\_alpha) \cdot 1.0 & \text{otherwise} \end{cases}$$

$$u\_beta = \begin{cases} \dfrac{inv\_mod\_index \cdot u\_beta}{u\_dc\_bus\_actual/2} & \text{if } |inv\_mod\_index \cdot u\_beta| < \dfrac{u\_dc\_bus\_actual}{2} \\ sign(u\_beta) \cdot 1.0 & \text{otherwise} \end{cases}$$

where the y = sign (x) function is defined as follows:

$$y = \begin{cases} 1.0 & \text{if } x \geq 0 \\ -1.0 & \text{otherwise} \end{cases}$$

Where:

u_alpha is alpha component of applied motor voltage, in [V].

u_beta is beta component of applied motor voltage, in [V].

u_dc_bus_actual is actual measured value of the DC-bus voltage, in [V].

inv_mod_index is Inverse Modulation Index; depends on the selected Modulation Technique, in [-].

Figure 5 and Figure 6 depict ripple elimination functionality. Due to variations made in the actual DC-bus voltage, the ripple elimination algorithm influences the duty cycles that are generated using the standard space vector modulation technique.

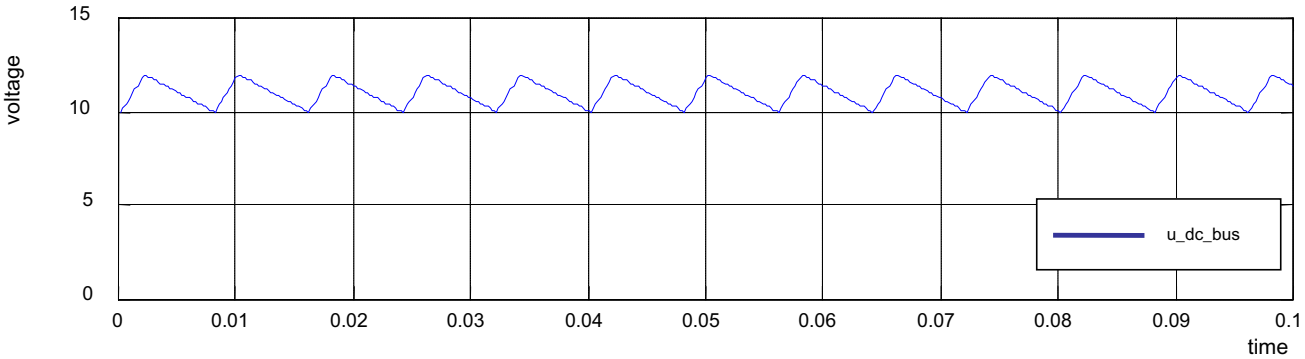

Figure 5. Measured Voltage on the DC-Bus

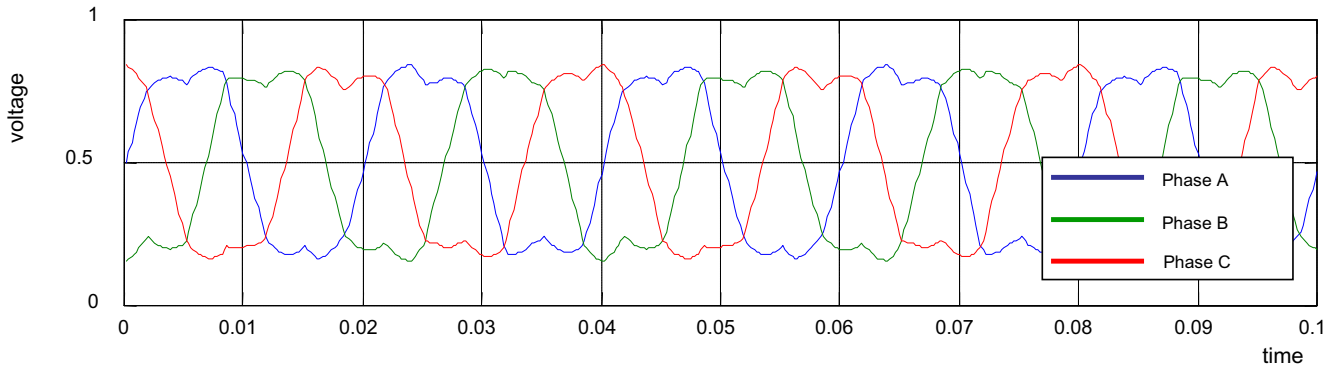

Figure 6. Standard Space Vector Modulation with Elimination of the DC_bus Ripple

The ACIMVC function update, in which all vector control calculations are performed, can be executed periodically, or by another process:

- **Master mode**

  The ACIMVC update is executed periodically with a given period.
- **Slave mode**

  The ACIMVC update is executed by the analog sensing for AC motors (ASAC) eTPU function, other eTPU function, or by the CPU.

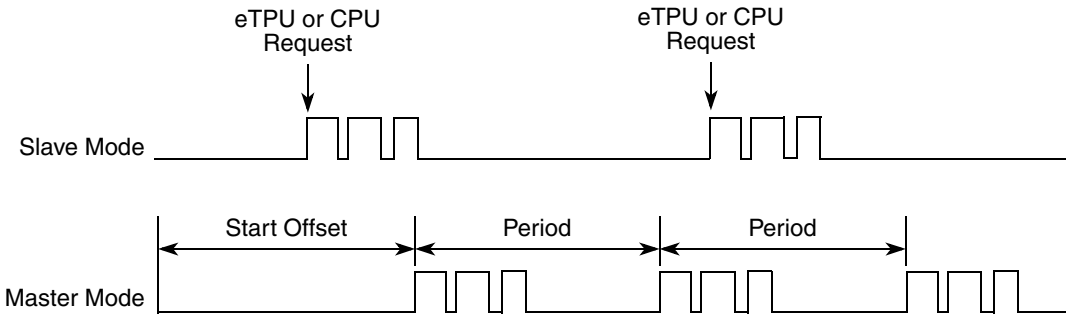

Figure 7. ACIMVC updates in Slave Mode and Master Mode

The ACIMVC update is divided in three consecutive threads. It enables to interrupt the ACIMVC calculations by another channel activity, and it keeps the latency caused by ACIMVC low.

4.1 Interrupts

The ACIMVC function generates an interrupt service request to the CPU at the end of every n-th update. The number of updates, after which an interrupt service request is generated, is a function parameter.

4.2 Performance

Like all eTPU functions, the ACIMVC function performance in an application is, to some extent, dependent upon the service time (latency) of other active eTPU channels. This is due to the operational nature of the scheduler.

The influence of the ACIMVC function on the overall eTPU performance can be expressed by the following parameter:

Maximum eTPU busy-time per one update
This value, compared to the update period value, determines the proportional load on the eTPU engine caused by ACIMVC function.

Longest thread time
This value determines the longest latency which can be caused by ACIMVC function.

Table 1 lists the maximum eTPU busy-times per update period in eTPU cycles that depend on the ACIMVC mode, and ripple elimination configuration.

Table 1. Maximum eTPU Busy-Times

Mode, Ripple Elimination, and Controller Type	Maximum eTPU Busy-Time per One Update Period (eTPU Cycles)	Longest Thread Time (eTPU Cycles)
Master mode, Circle limitation OFF	1304	584
Master mode, Circle limitation OFF	1550	584
Slave mode, Circle limitation OFF	1292	584
Slave mode, Circle limitation OFF	1538	584

On MPC5500 devices, the eTPU module clock is equal to the CPU clock. On MCF523x devices, it's equal to the peripheral clock, which is a half of the CPU clock,. For example, on a 132-MHz MPC5554, the eTPU module clock is 132 MHz, and one eTPU cycle takes 7.58 ns. On a 150-MHz MCF5235, the eTPU module clock is only 75 MHz and one eTPU cycle takes 13.33 ns.

The performance is influenced by the compiler efficiency. The above numbers, measured on the code compiled by eTPU compiler version 1.0.7, are given for guidance only and are subject to change. For up-to-date information, refer to the information provided in the particular eTPU function set release available from Freescale.

5 C Level API for Function

The following routines provide easy access for the application developer to the ACIMVC function. Use of these functions eliminates the need to directly control the eTPU registers.

There are 18 functions added to the ACIMVC application programming interface (API). The routines can be found in the `etpu_acimvc.h` and `etpu_acimvc.c` files, which should be linked with the top level development file(s).

Figure 8 shows the ACIMVC API state flow and lists API functions that can be used in each of its states.

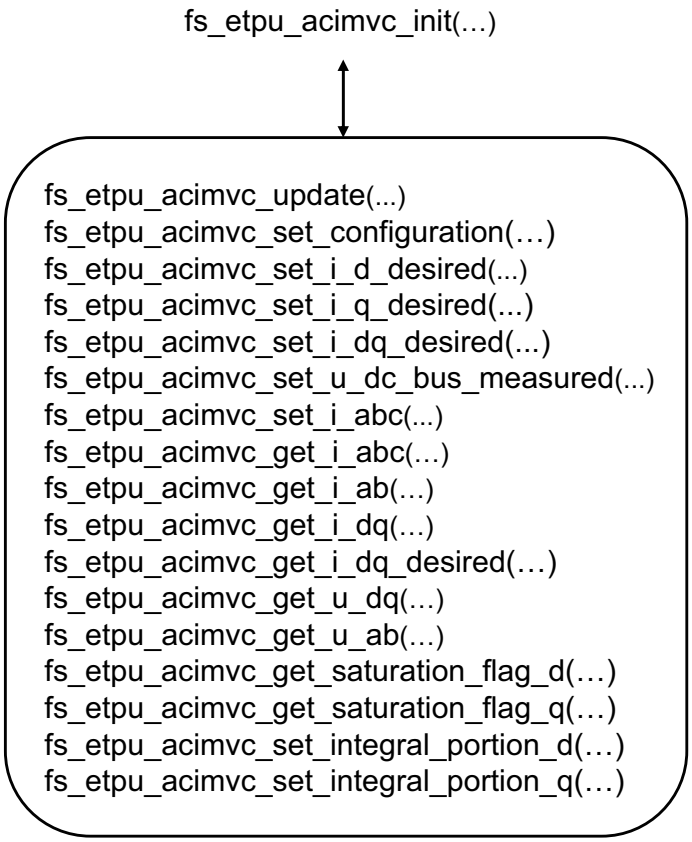

Figure 8. ACIMVC API State Flow

All ACIMVC API routines will be described in order and are listed below:

- Initialization functions:

```
int32_t fs_etpu_acimvc_init( uint8_t channel,
                             uint8_t priority,
                             uint8_t mode,
                             uint8_t circle_limitation_config,
                            uint24_t period,
                            uint24_t start_offset,
                            uint24_t services_per_irq,
                             uint8_t SC_chan,
                  acimvc_motor_params_t* p_motor_params,
                    acimvc_pid_params_t* p_pid_d_params,
                    acimvc_pid_params_t* p_pid_q_params,
                             int24_t inv_mod_index,
                             uint8_t output_chan,
                            uint16_t output_offset,
```

```
                                        uint8_t link_chan)
```

- Change operation functions:

```
int32_t fs_etpu_acimvc_set_configuration( uint8_t  channel,
                                          uint8_t configuration)

int32_t fs_etpu_acimvc_update(uint8_t channel)

int32_t fs_etpu_acimvc_set_i_d_desired(uint8_t  channel,
                                       fract24_t  i_d_desired)

int32_t fs_etpu_acimvc_set_i_q_desired(uint8_t  channel,
                                       fract24_t  i_q_desired)

int32_t fs_etpu_acimvc_set_i_dq_desired(uint8_t  channel,
                                        acimvc_dq_t * p_i_dq_desired)

int32_t fs_etpu_acimvc_set_u_dc_bus_measured(uint8_t  channel,
                                          ufract24_t  u_dc_bus_measured)

int32_t fs_etpu_acimvc_set_i_abc(uint8_t  channel,
                                 acimvc_abc_t * p_i_abc)

int32_t fs_etpu_acimvc_set_integral_portion_d( uint8_t channel,
                                               fract24_t i_k1)

int32_t fs_etpu_acimvc_set_integral_portion_q( uint8_t channel,
                                               fract24_t i_k1)
```

- Value return functions:

```
int32_t fs_etpu_acimvc_get_i_abc(uint8_t channel,
                                 acimvc_abc_t * p_i_abc)


int32_t fs_etpu_acimvc_get_i_ab(uint8_t channel,
                                acimvc_ab_t * p_i_ab)
int32_t fs_etpu_acimvc_get_i_dq(uint8_t channel,
                                acimvc_dq_t * p_i_dq)


int32_t fs_etpu_acimvc_get_i_dq_desired(uint8_t channel,
                                        acimvc_dq_t * p_i_dq_desired)
```

```
int32_t fs_etpu_acimvc_get_u_dq(uint8_t channel,
                                acimvc_dq_t * p_u_dq)

int32_t fs_etpu_acimvc_get_u_ab(uint8_t channel,
                                acimvc_ab_t * p_u_ab)

uint8_t fs_etpu_acimvc_get_saturation_flag_d(uint8_t channel)

uint8_t fs_etpu_acimvc_get_saturation_flag_q(uint8_t channel)
```

5.1 Initialization Function

5.1.1 int32_t fs_etpu_acimvc_init(...)

This routine is used to initialize the eTPU channel for the ACIMVC function. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **priority (uint8_t)**—The priority to assign to the ACIMVC function; should be assigned one of these values:
  - FS_ETPU_PRIORITY_HIGH
  - FS_ETPU_PRIORITY_MIDDLE
  - FS_ETPU_PRIORITY_LOW
- **mode (uint8_t)**—The function mode; should be assigned one of these values:
  - FS_ETPU_ACIMVC_MASTER
  - FS_ETPU_ACIMVC_SLAVE
- **circle_limitation_config (uint8_t)**—The required configuration of circle limitation; should be assigned one of these values:
  - FS_ETPU_ACIMVC_CIRCLE_LIMITATION_OFF,
  - FS_ETPU_ACIMVC_CIRCLE_LIMITATION_ON
- **period (uint24_t)**—The update period, as a number of TCR1 clocks. This parameter applies in the master mode only (mode=FS_ETPU_ACIMVC_MASTER).
- **start_offset (uint24_t)**—Used to synchronize various eTPU functions that generate a signal. The first ACIMVC update starts the start_offset TCR1 clocks after initialization. This parameter applies in the master mode only (mode=FS_ETPU_ACIMVC_MASTER).
- **services_per_irq (uint24_t)**—Defines the number of updates after which an interrupt service request is generated to the CPU.

- **SC_chan (uint8_t)**—The number of a channel the SC function is assigned to. The ACIMVC reads the actual speed from SC. This parameter should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **p_motor_params (acimvc_motor_params_t*)**—The pointer to a acimvc_motor_params_t structure of motor constants. The acimvc_motor_params_t structure is defined in `etpu_acimvc.h`:

```
typedef struct {
   fract24_t KL_T_KT; /* motor dependent constant 1, in fract. format (3.21) */
   fract24_t KL_R_KT; /* motor dependent constant 2, in fract. format (1.23) */
   fract24_t I_KT;    /* motor dependent constant 3, in fract. format (1.23) */
   fract24_t TR_KT;   /* motor dependent constant 4, in fract. format (1.23) */
   fract24_t T;       /* motor dependent constant 5, in fract. format (1.23) */
   fract24_t LM_TR;   /* motor dependent constant 6, in fract. format (3.21) */
   fract24_t LM_LR_TR;/* motor dependent constant 7, in fract. format (1.23) */
   fract24_t LM_LR;   /* motor dependent constant 8, in fract. format (1.23) */
   fract24_t KL;      /* motor dependent constant 9, in fract. format (3.21) */
} acimvc_motor_params_t;
```

The following motor parameters are essential for motor dependent constants setting:

Ls	Self-inductance of the stator [H]
Lr	Self-inductance of the rotor [H]
Lm	Magnetizing inductance [H]
Rr	Resistance of a rotor phase winding [Ohm]
Rs	Resistance of a stator phase winding [Ohm]
pp	Number of motor pole-pairs [-]
Tr=Lr/Rr	Rotor time constant [s]
Ts=Ls/Rs	Stator time constant [s]
sigma = 1-(Lm*Lm/Ls*Lr)	Resultant leakage constant [-]

The motor dependent constants are calculated and scaled to the nominal range as follows:

— KL_T_KT[-] = ((sigma[-] ¥ Lm[H] ¥ Ts[s]) / (Tr[s] + Ts[s] ¥ (1 - sigma[-]))) ¥ phase_current_range[A] / rotor_flux_range[Vs]
KL_T_KT[fract 3.21] = 0x200000 ¥ KL_T_KT[-]

— KL_R_KT[-] = ((Lm[H] / Rs[Ohm]) / (Tr[s] + Ts[s] ¥ (1 - sigma[-]))) ¥ dc_bus_voltage[V] / (pp[-] ¥ rotor_flux_range[Vs] ¥ omega_range[rad/s])
KL_R_KT[fract 1.23] = 0x800000 ¥ KL_R_KT[-]

— I_KT[-] = (1 / (Tr[s] + Ts[s] ¥ (1 - sigma[-]))) ¥ rotor_flux_range[Vs] / (pp[-] ¥ rotor_flux_range[Vs] ¥ omega_range[rad/s])
I_KT[fract 1.23] = 0x800000 ¥ I_KT[-]

— TR_KT[-] = ((pp[-] ¥ Lr[H] / Rr[Ohm]) / (Tr[s] + Ts[s] ¥ (1 - sigma[-]))) ¥ rotor_flux_range[Vs] ¥ omega_range[rad/s] / (pp[-] ¥ rotor_flux_range[Vs] ¥ omega_range[rad/s])
TR_KT[fract 1.23] = 0x800000 ¥ TR_KT[-]

— T[-] = (pp[-] ¥ rotor_flux_range[Vs] ¥ omega_range[rad/s]) / (PWM_freq[hz] ¥ rotor_flux_range[Vs]
T[fract 1.23] = 0x800000 ¥ T[-]

— LM_TR[-] = Lm[H] / (Tr[s] ¥ pp[-]) ¥ phase_current_range[A] / (dc_bus_voltage[V] ¥ rotor_flux_range[Vs])
LM_TR[fract 3.21] = 0x200000 ¥ LM_TR[-]

— LM_LR_TR[-] = Lm[H] / (Lr[H] ¥ Tr[s]) ¥ rotor_flux_range[Vs] / (2 ¥ dc_bus_voltage[V])
LM_LR_TR[fract 1.23] = 0x800000 ¥ LM_LR_TR[-]

— LM_LR[-] = pp[-] ¥ Lm[H] / Lr[H] ¥ rotor_flux_range[Vs] ¥ omega_range[rad/s] / (2 ¥ dc_bus_voltage[V])
LM_LR[fract 1.23] = 0x800000 ¥ LM_LR[-]

— KL[-] = pp[-] ¥ (Ls[H] - Lm[H] ¥ Lm[H] / Lr[H]) ¥ phase_current_range[A] ¥ omega_range[rad/s] / (2 ¥ dc_bus_voltage[V])
KL[fract 3.21] = 0x200000 ¥ KL[-]

- **p_pid_d_params (acimvc_pid_params_t*)**×The pointer to a acimvc_pid_params_t structure of D-coordinate PID controller parameters. The acimvc_pid_params_t structure is defined in `etpu_acimvc.h`:

```
typedef struct {
   fract24_t P_gain;
   fract24_t I_gain;
   fract24_t D_gain;
   int16_t positive_limit;
   int16_t negative_limit;
} acimvc_pid_params_t;
```

Where:

— **P_gain (fract24_t)** is the proportional gain whose value must be in the 24-bit signed fractional format 9.15. This means in the range of (-256, 256).

```
0x008000 corresponds to   1.0
0x000001 corresponds to   0.0000305 (30.5e-6)
0x7FFFFF corresponds to 255.9999695
```

— **I_gain (fract24_t)** is the integral gain whose value must be in the 24-bit signed fractional format 9.15. This means in the range of (-256, 256).

— **D_gain (fract24_t)** is the derivative gain whose value must be in the 24-bit signed fractional format 9.15. This means in the range of (-256, 256). To switch off calculation of derivative portion, set this parameter to zero.

— **positive_limit (int16_t)** is the positive output limit whose value must be in the 16-bit signed fractional format 1.15. This means in the range of (-1, 1).

— **negative_limit (int16_t)** is the negative output limit whose value must be in the 16-bit signed fractional format 1.15. This means in the range of (-1, 1).

- **p_pid_q_params (acimvc_pid_params_t*)**—The pointer to a acimvc_pid_params_t structure of Q-coordinate PID controller parameters.

- **inv_mod_index (int24_t)**—Defines the Inverse Modulation Index. Inverse Modulation Index is dependent on the type of modulation technique being used by the PWMMAC. This parameter should be assigned one of these values:
  - FS_ETPU_ACIMVC_INVMODINDEX_SINE
  - FS_ETPU_ACIMVC_INVMODINDEX_SIN3H
  - FS_ETPU_ACIMVC_INVMODINDEX_SVM
- **output_chan (uint8_t)**—ACIMVC writes outputs to a recipient function's input parameters. This is the recipient function channel number. 0-31 for ETPU_A and 64-95 for ETPU_B.
- **output_offset (uint16_t)**—ACIMVC writes outputs to a recipient function's input parameters. This is the first input parameter offset of the recipient function. Function parameter offsets are defined in etpu_<func>_auto.h file.
- **link_chan (uint8_t)**—The number of the channel that receives a link after ACIMVC updates output. Usually ACIMVC updates PWMMAC inputs, and that is why it should be a PWMMAC channel. 0-31 for ETPU_A and 64-95 for ETPU_B.

5.2 Change Operation Functions

5.2.1 int32_t fs_etpu_acimvc_set_configuration(uint8_t channel, uint8_t configuration)

This function changes the ACIMVC configuration. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **configuration (uint8_t)**—The required configuration of ACIMVC; should be assigned one of these values:
  - FS_ETPU_ACIMVC_PID_OFF (DQ PID controllers are disabled)
  - FS_ETPU_ACIMVC_PID_ON (DQ PID controllers are enabled)

5.2.2 int32_t fs_etpu_acimvc_update(uint8_t channel)

This function executes the ACIMVC update. It has this parameter:

- **channel (uint8_t)**—T ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

5.2.3 int32_t fs_etpu_acimvc_set_i_d_desired(uint8_t channel, fract24_t i_d_desired)

This function changes the value of D-component of desired phase currents in 2-phase orthogonal rotating reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **i_d_desired (fract24_t)**—D-component of desired phase currents in 2-phase orthogonal rotating reference frame, in range MIN24 to MAX24.

5.2.4 int32_t fs_etpu_acimvc_set_i_q_desired(uint8_t channel, fract24_t i_q_desired)

This function changes the value of Q-component of desired phase currents in 2-phase orthogonal rotating reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **i_q_desired (fract24_t)**—Q-component of desired phase currents in 2-phase orthogonal rotating reference frame, in range MIN24 to MAX24.

5.2.5 int32_t fs_etpu_acimvc_set_i_dq_desired(uint8_t channel, acimvc_dq_t * p_i_dq_desired)

This function changes the value of desired phase currents in 2-phase orthogonal rotating reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_dq_desired (acimvc_dq_t *)**—Pointer to structure of desired phase currents in 2-phase orthogonal rotating reference frame, in range MIN24 to MAX24.

5.2.6 int32_t fs_etpu_acimvc_set_u_dc_bus_measured(uint8_t channel, ufract24_t u_dc_bus_measured)

This function sets the value of actual DC-bus voltage, as a portion of the AD convertor range. It can be used in case a DMA transfer of the value from AD converter to eTPU is not used and has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number.; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **u_dc_bus_measured (ufract24_t)**—The actual value of DC-bus voltage, as an unsigned 24 bit portion of the AD converter range.

5.2.7 int32_t fs_etpu_acimvc_set_i_abc(uint8_t channel, acimvc_abc_t * p_i_abc)

This function sets the values of i_abc - input phase currents in 3-phase stationary reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

- **p_i_abc (acimvc_abc_t*)**—Pointer to structure of phase currents in 3-phase stationary reference frame.

5.2.8 int32_t fs_etpu_acimvc_set_integral_portion_d(uint8_t channel, fract24_t i_k1)

This function sets the D component PID controller integral portion (usually used to set the integral portion to zero). It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **i_k1 (fract24_t)**—The integral portion value in 24-bit signed fractional format 1.23, range (-1,1).

5.2.9 int32_t fs_etpu_acimvc_set_integral_portion_q(uint8_t channel, fract24_t i_k1)

This function sets the Q component PID controller integral portion (usually used to set the integral portion to zero). It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **i_k1 (fract24_t)**—The integral portion value in 24-bit signed fractional format 1.23, range (-1,1).

5.3 Value Return Function

5.3.1 int32_t fs_etpu_acimvc_get_i_abc(uint8_t channel, acimvc_abc_t * p_i_abc)

This function gets the values of i_abc - input phase currents in 3-phase stationary reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_abc (acimvc_abc_t*)**—Pointer to structure of phase currents in 3-phase stationary reference frame.

5.3.2 int32_t fs_etpu_acimvc_get_i_ab(uint8_t channel, acimvc_ab_t * p_i_ab)

This function gets the values of i_ab - phase currents in 2-phase orthogonal stationary reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

- **p_i_ab (acimvc_ab_t*)**—Pointer to structure of phase currents in 2-phase orthogonal stationary reference frame.

5.3.3 int32_t fs_etpu_acimvc_get_i_dq(uint8_t channel, acimvc_dq_t * p_i_dq)

This function gets the values of i_dq - phase currents in 2-phase orthogonal rotating reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_dq (acimvc_dq_t*)**—Pointer to structure of phase currents in 2-phase orthogonal rotating reference frame.

5.3.4 int32_t fs_etpu_acimvc_get_i_dq_desired(uint8_t channel, acimvc_dq_t * p_i_dq_desired)

This function gets the values of i_dq_desired - desired phase currents in 2-phase orthogonal rotating reference frame. It has these parameters:

- **channel (uint8_t)** —The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_dq_desired (acimvc_dq_t*)**—Pointer to return structure of phase currents in 2-phase orthogonal rotating reference frame.

5.3.5 int32_t fs_etpu_acimvc_get_u_dq(uint8_t channel, acimvc_dq_t * p_u_dq)

This function gets the values of u_dq - stator voltages in 2-phase orthogonal rotating reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_u_dq (acimvc_dq_t*)**—Pointer to structure of of stator voltages in 2-phase orthogonal rotating reference frame.

5.3.6 int32_t fs_etpu_acimvc_get_u_ab(uint8_t channel, acimvc_ab_t * p_u_ab)

This function gets the values of u_ab - stator voltages in 2-phase orthogonal stationary reference frame. It has these parameters:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_u_ab (acimvc_ab_t*)**—Pointer to structure of of stator voltages in 2-phase orthogonal stationary reference frame.

5.3.7 uint8_t fs_etpu_acimvc_get_saturation_flag_d(uint8_t channel)

This function returns the D component PID controller saturation flags.It has this parameter:

- **channel (uint8_t)** —The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

The returned value can be:

— FS_ETPU_ACIMVC_SATURATION_NO (0) ... no saturation

— FS_ETPU_ACIMVC_SATURATION_POS (1) ... saturation to positive limit

— FS_ETPU_ACIMVC_SATURATION_NEG (2) ... saturation to negative limit

5.3.8 uint8_t fs_etpu_acimvc_get_saturation_flag_q(uint8_t channel)

This function returns the Q component PID controller saturation flags. It has this parameter:

- **channel (uint8_t)**—The ACIMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

The returned value can be:

— FS_ETPU_ACIMVC_SATURATION_NO (0) ... no saturation

— FS_ETPU_ACIMVC_SATURATION_POS (1) ... saturation to positive limit

— FS_ETPU_ACIMVC_SATURATION_NEG (2) ... saturation to negative limit

6 Example Use of Function

6.1 Demo Applications

The use of the ACIMVC eTPU function is demonstrated in the following application note:

- “AC Induction Motor Vector Control, Driven by eTPU on MPC5500,” AN3001

For a detailed description of the demo application, refer to the above application note.

6.1.1 Function Calls

The ACIMVC function is configured to the slave mode and calculates current control loop on a link from ASAC function. The desired value of Q-component of phase currents in 2-phase orthogonal rotating reference frame (i_q_required) is provided by the SC function. The desired value of D-component of phase currents in 2-phase orthogonal rotating reference frame (i_d_required) is set to 0. The circle limitation is on. The controller output points to a PWMMAC input, so that it controls the duty-cycle of PWM phases.

```
/*****************************************************************************
* Parameters
*****************************************************************************/
```

```
int32_t speed_range_rpm = 4000;
int32_t dc_bus_voltage_range_mv = 618000;
int32_t phase_current_range_ma = 8000;
int32_t rotor_flux_range_mVs = 1000;
uint8_t ACIM_pole_pairs = 2;
int32_t ACIM_resist_stator_mOhm = 32250;
int32_t ACIM_resist_rotor_mOhm = 31170;
int32_t ACIM_Lm_uH = 537800;
int32_t ACIM_Ls_leak_uH = 28100;
int32_t ACIM_Lr_leak_uH = 65500;
int32_t ACIM_Ke_mv_per_krpm = 150000;
int32_t ACIMVC_D_PID_gain_permil = 1000;
int32_t ACIMVC_D_I_time_const_us = 100000;
int32_t ACIMVC_Q_PID_gain_permil = 2000;
int32_t ACIMVC_Q_I_time_const_us = 1000;
uint8_t ACIMVC_channel = 6;
uint8_t SC_channel = 5;
uint8_t PWM_master_channel = 7;
int32_t PWM_freq_hz = 20000;
uint8_t QD_phaseA_channel = 1;
int24_t QD_pc_per_rev = 4096;

/*****************************************************************************
 *
 * 4) Initialize ACIM Vector Control Loop
 *
 *****************************************************************************/
/*****************************************************************************
 * 4.1) Define D-Current Controller PID Parameters
 *****************************************************************************
 *  The P-gain and I-gain are calculated from the controller gain and
 *  integral time constant, given by parameters, and transformed to 24-bit
 *  fractional format 9.15:
 *     P_gain = PID_gain_permil/1000 * 0x008000;
 *     I_gain = PID_gain_permil/1000 * 1/update_freq_hz *
 *              * 1000000/I_time_const_us * 0x008000;
 *  The D-gain is set to zero in order to have a PI-type controller.
 *  The positive and negative limits, which are set in 16-bit fractional
 *  format (1.15), can be adjusted in order to limit the speed controller
 *  output range, and also the integral portion range.
 *****************************************************************************/
  acimvc_pid_d_params.P_gain = ACIMVC_D_PID_gain_permil*0x001000/125;
  acimvc_pid_d_params.I_gain = 0x008000*1000/PWM_freq_hz*ACIMVC_D_PID_gain_permil
                               /ACIMVC_D_I_time_const_us;
  acimvc_pid_d_params.D_gain = 0;
  acimvc_pid_d_params.positive_limit = 0x7FFF;
  acimvc_pid_d_params.negative_limit = 0x8000;
```

```
/*************************************************************************
 * 4.2) Define Q-Current Controller PID Parameters
 *************************************************************************/
   acimvc_pid_q_params.P_gain = ACIMVC_Q_PID_gain_permil*0x001000/125;
   acimvc_pid_q_params.I_gain = 0x008000*1000/PWM_freq_hz*ACIMVC_Q_PID_gain_permil
                                /ACIMVC_Q_I_time_const_us;
   acimvc_pid_q_params.D_gain = 0;
   acimvc_pid_q_params.positive_limit = 0x7FFF;
   acimvc_pid_q_params.negative_limit = 0x8000;
/*************************************************************************
 * 4.3) Define Motor Parameters
 *************************************************************************/
/* omega range in mrad/s */
omega_range_rad_s = 2*3.1415927f*speed_range_rpm/60;
/* Motor self-inductance of the stator in uH */Ls_mH = (ACIM_Lm_uH +
ACIM_Ls_leak_uH)/1000.0f;

/* Motor self-inductance of the rotor in uH */
Lr_mH = (ACIM_Lm_uH + ACIM_Lr_leak_uH)/1000.0f;

/* Motor stator time constant in ms */
Ts_s = Ls_mH/ACIM_resist_stator_mOhm;

/* Motor rotor time constant in ms */
Tr_s = Lr_mH/ACIM_resist_rotor_mOhm;

/* Motor resultant leakage constant */
sigma = (1.0f - (ACIM_Lm_uH/1000.0f)*(ACIM_Lm_uH/1000.0f)/(Ls_mH*Lr_mH));

/* Rotor magnetic flux calculation constants */
dx_range = rotor_flux_range_mVs/1000.0f * ACIM_pole_pairs * omega_range_rad_s;
i_kt_s = 1.0f/(Tr_s + Ts_s*(1.0f-sigma));
acimvc_motor_params.KL_T_KT = (int32_t)((float)0x200000*sigma*ACIM_Lm_uH*Ts_s*
   i_kt_s*phase_current_range_mA/(rotor_flux_range_mVs*1e6));
acimvc_motor_params.KL_R_KT = (int32_t)((float)0x800000*i_kt_s*ACIM_Lm_uH*
   dc_bus_voltage_range_mV/(ACIM_resist_stator_mOhm*dx_range*1e6));
acimvc_motor_params.I_KT    = (int32_t)((float)0x800000*i_kt_s*
   rotor_flux_range_mVs/(dx_range*1e3));
acimvc_motor_params.TR_KT   = (int32_t)((float)0x800000*i_kt_s*ACIM_pole_pairs*Tr_s*
   omega_range_rad_s*rotor_flux_range_mVs/(dx_range*1e3));
acimvc_motor_params.T       = (int32_t)((float)0x800000*dx_range*1e3/(PWM_freq_hz*
   rotor_flux_range_mVs));

/* D-Q system establishment constants */
acimvc_motor_params.LM_TR   = (int32_t)((float)0x200000*ACIM_Lm_uH*
   phase_current_range_mA/(Tr_s*ACIM_pole_pairs*rotor_flux_range_mVs*
   dc_bus_voltage_range_mV*1e3));
```

```
/* ACIM decouplinq constants */
acimvc_motor_params.LM_LR_TR= (int32_t)((float)0x800000*ACIM_Lm_uH*
   rotor_flux_range_mVs/(Lr_mH*Tr_s*dc_bus_voltage_range_mV*2e3));
acimvc_motor_params.LM_LR   = (int32_t)((float)0x800000*ACIM_pole_pairs*ACIM_Lm_uH*
   rotor_flux_range_mVs*omega_range_rad_s/(Lr_mH*dc_bus_voltage_range_mV*2e3));
acimvc_motor_params.KL      = (int32_t)((float)0x200000*ACIM_pole_pairs*
   ((float)Ls_mH -((float)ACIM_Lm_uH*ACIM_Lm_uH)/(Lr_mH*1e6))*
   phase_current_range_mA*omega_range_rad_s/(dc_bus_voltage_range_mV*2e3));

/*************************************************************************
 * 4.4) Initialize ACIMVC channel
 *************************************************************************/
   err_code = fs_etpu_acimvc_init(
      ACIMVC_channel,/* channel */
      FS_ETPU_PRIORITY_LOW,/* priority */
      FS_ETPU_ACIMVC_SLAVE,/* mode */
      FS_ETPU_ACIMVC_CIRCLE_LIMITATION_ON, /* circle_limitation_config */
      0, /* period */
      0, /* start_offset */
      0, /* services_per_irq */
      SC_channel, /* SC_chan */
      &acimvc_motor_params, /* p_motor_params */
      &acimvc_pid_d_params, /* p_pid_d_params */
      &acimvc_pid_q_params, /* p_pid_q_params */
      FS_ETPU_ACIMVC_INVMODINDEX_SINE, /* inv_mod_index */
      PWM_master_channel, /* output_chan */
      FS_ETPU_PWMMAC_INPUTS_OFFSET,/* output_offset */
      0/*PWM_master_channel*//* link_chan */
   );
   if (err_code != 0)
        return (err_code);
```

7 Summary and Conclusions

This application note provides the user with a description of the ACIMVC eTPU function. The simple C interface routines to the ACIMVC eTPU function enable easy implementation of the ACIMVC in applications. The demo application is targeted at the MPC5500 family of devices, but it can easily be reused with any device that has an eTPU.

7.1 References

1. “The Essential of Enhanced Time Processing Unit,” AN2353
2. “General C Functions for the eTPU,” AN2864
3. “Using the AC Motor Control eTPU Function Set (set4),” AN2968

4. *Enhanced Time Processing Unit Reference Manual*, ETPURM
5. eTPU Graphical Configuration Tool, http://www.freescale.com/etpu, ETPUGCT
6. “Using the AC Motor Control PWM eTPU Functions,” AN2969
7. “AC Induction Motor Vector Control, Driven by eTPU on MPC5500,” AN3001

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2973
Rev. 0
04/2006