Freescale Semiconductor
Application Note

AN2972
Rev. 2, 02/2012

Using the PMSM Vector Control eTPU Function

Covers the MCF523*x*, MPC5500, MPC5600 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 permanent magnet synchronous motor vector control (PMSMVC) 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 provides simple C interface routines to the PMSMVC eTPU function. The routines are targeted at the MCF523*x,* MPC5500, and MPC5600 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 permanent magnet synchronous motor (PMSM), 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 how much electro-magnetic field is generated and how much torque is generated. Then the

freescale™ semiconductor

structure of the motor controller (vector control controller) is almost the same as for a separately excited DC motor, which simplifies the control of PMSM. This vector control technique was developed in the past especially to achieve similar excellent dynamic performance of PMSM.
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 'd-q reference co-ordinate system.' All necessary transformations for vector control are described here.

2.1 Mathematical Model of PMSM Control

For a description of the PMSM, 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:

$$u_{SA} = R_S i_{SA} + \frac{d}{dt}\psi_{SA} \quad \textit{Eqn. 1}$$

$$u_{SB} = R_S i_{SB} + \frac{d}{dt}\psi_{SB} \quad \textit{Eqn. 2}$$

$$u_{SC} = R_S i_{SC} + \frac{d}{dt}\psi_{SC} \quad \textit{Eqn. 3}$$

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 SA, SB, and SC.
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). Then the PMSM can be expressed as:

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

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

$$\Psi_{S\alpha} = L_S i_{S\alpha} + \Psi_M \cos(\Theta_r) \quad \textit{Eqn. 6}$$

$$\Psi_{S\beta} = L_S i_{S\beta} + \Psi_M \sin(\Theta_r) \quad \textit{Eqn. 7}$$

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p(\Psi_{S\alpha} i_{S\beta} - \Psi_{S\beta} i_{S\alpha}) - T_L\right] \quad \textit{Eqn. 8}$$

where: α,β - Stator orthogonal coordinate system
$u_{S\alpha,\beta}$ - Stator voltages
$i_{S\alpha,\beta}$ - Stator currents
$\Psi_{S\alpha,\beta}$ - Stator magnetic fluxes
Ψ_M - Rotor magnetic flux
R_S - Stator phase resistance

L_S - Stator phase inductance
ω / ω_F - Electrical rotor speed / fields speed
p - Number of poles per phase
J - Inertia
T_L - Load torque
Θ_r - rotor position in α,β coordinate system

Equations Eqn. 4 through Eqn. 8 represent the model of a PMSM in the stationary frame α, β fixed to the stator. The main idea of the vector control is to decompose the vectors into a magnetic field generating part and a torque generating part. In order 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 ‘d-q-co-ordinate system’ (Park transformation). Thus the equations Eqn. 4 through Eqn. 8 can be rewritten as:

$$u_{Sd} = R_S i_{Sd} + \frac{d}{dt}\Psi_{Sd} - \omega_F \Psi_{Sq} \quad \textbf{Eqn. 9}$$

$$u_{Sq} = R_S i_{Sq} + \frac{d}{dt}\Psi_{Sq} + \omega_F \Psi_{Sd} \quad \textbf{Eqn. 10}$$

$$\Psi_{Sd} = L_S i_{Sd} + \Psi_M \quad \textbf{Eqn. 11}$$

$$\Psi_{Sq} = L_S i_{Sq} \quad \textbf{Eqn. 12}$$

$$\frac{d\omega}{dt} = \frac{p}{J}\left[\frac{3}{2}p(\Psi_{Sd} i_{Sq} - \Psi_{Sq} i_{Sd}) - T_L\right] \quad \textbf{Eqn. 13}$$

The expression of electromagnetic torque is given as follows:

$$t_e = \frac{3}{2}p(\Psi_{Sd} i_{Sq} - \Psi_{Sq} i_{Sd}) \quad \textbf{Eqn. 14}$$

2.1.1 Model of PMSM in Rotating Reference Frame

In order to produce the largest torque, an optimal operation is achieved by stator current control, which ensures that the stator current space vector contains only a quadrature component, by considering that below, the base speed $i_{sd} = 0$. This is achieved in the reference frame fixed to the rotor. Equations Eqn. 9 through Eqn. 12 give a model of PMSM expressed in rotating reference frame as follows:

$$u_{Sd} = R_S i_{Sd} + L_S \frac{d}{dt} i_{Sd} - L_S \omega_F i_{Sq} \quad \textbf{Eqn. 15}$$

$$u_{Sq} = R_S i_{Sq} + L_S \frac{d}{dt} i_{Sq} + L_S \omega_F i_{Sd} + \omega_F \Psi_M \quad \textbf{Eqn. 16}$$

From equation Eqn. 14, it can be seen that the torque is dependent and can be directly controlled by the current i_{sq} only.The expression of electromagnetic torque is similar to the expression for electromagnetic torque produced by a separately excited DC motor, given as follows:

$$t_e = \frac{3}{2}p(\Psi_M i_{Sq}) \quad \textbf{Eqn. 17}$$

This analogy is a fundamental basis for various forms of vector control techniques.

3 Function Overview

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

The sequence of PMSMVC calculations consists of the following steps:

- Forward Clarke transformation
- Forward Park transformation (establishing the DQ coordinate system)
- D&Q current controllers calculation
- Decoupling and back-EMF feed forward
- Circle limitation
- Inverse Park transformation
- DC-bus ripple elimination

The PMSMVC 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 PMSMVC function optionally enables to perform the limitation of calculated D and Q components of the stator voltages into the circle.

The PMSMVC 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 PMSMVC function turns the pin high and low, so that the high-time identifies the period of time in which the PMSMVC execution is active. In this way, the PMSMVC function, as with many of the motor-control eTPU functions, supports checking eTPU timing using an oscilloscope.

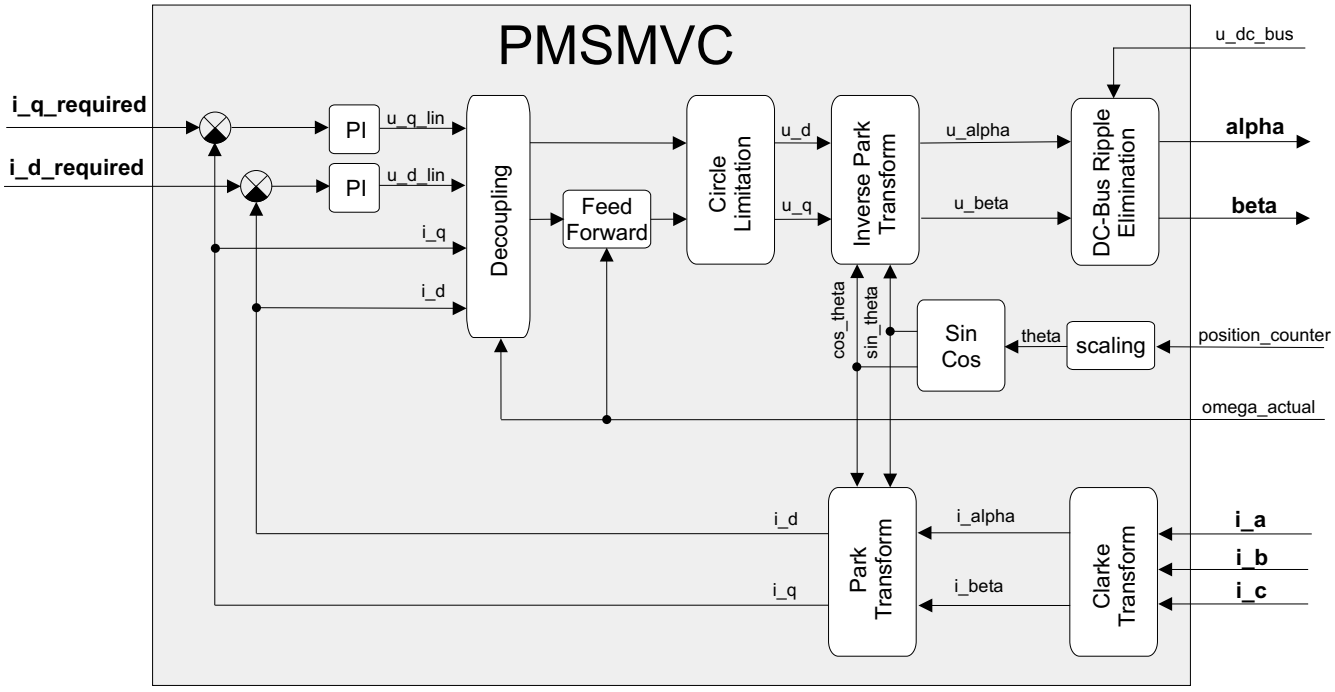

Figure 1. Functionality of PMSMVC

4 Function Description

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

- **Calculates actual rotor position theta.**

  This procedure is performed in order to transform a mechanical rotor position into the electrical rotor position.The mechanical position can be read either from a quadrature decoder or from a resolver.

  With a quadrature decoder, the input position counter value is transformed into an angle value in the range (–1, 1).

  $$theta = qd\_pc \times encoder\_scale$$

  Where:

  theta - Angle value, in [rad/pi], range (–1, 1)

  qd_pc - Position counter value from quadrature decoder, in QD increments

  encoder_scale - Scaling constant, equal to number of motor pole pairs / qd_pc range

  With a resolver, the input position counter value is transformed into an angle value in the range (–1, 1).

  $$theta = resolver\_angle \times resolver\_scale$$

  Where:

  theta - Angle value, in [rad/pi], range (–1, 1)

  resolver_angle - Angular position value from resolver, in range (–1, 1)

  resolver_scale - Scaling constant, equal to ratio between the number of motor pole pairs and the number of resolver pole pairs

- **Calculates sin(theta) and cos(theta).**

  The calculation of the sine and cosine of the theta angle is performed based on a sine look-up table. The look-up table contains 129 samples of the first sine quadrant. The other quadrant values are obtained by mirroring and negating the first quadrant values. The table look-up is a two-stage process: first, the two values of the nearest angles to the desired one are fetched from the look-up table, and then the linear interpolation of these two values is calculated.

- **Calculates Forward Clark Transformation.**

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

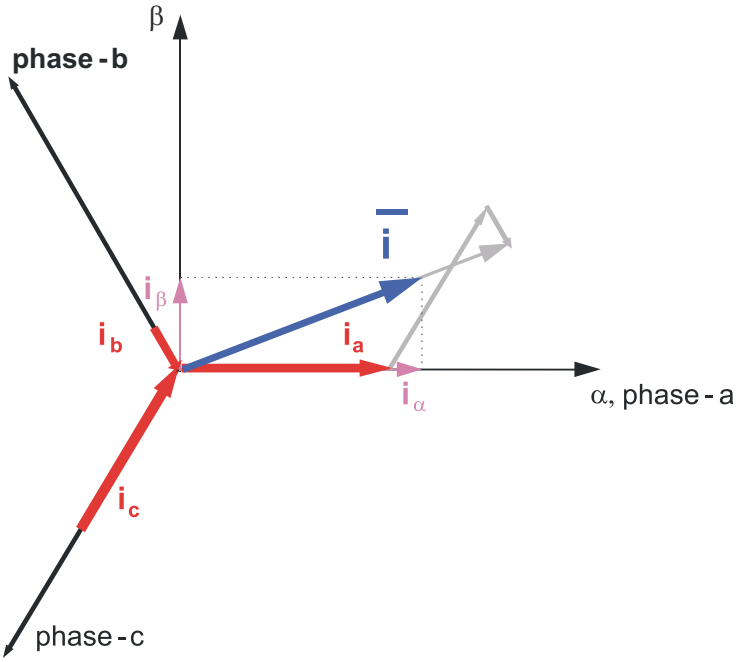

Figure 2. Clark Transformation

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:

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}\frac{2}{3} & -\frac{1}{3} & -\frac{1}{3}\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} = \mid\ a+b+c=0\ \mid = \begin{bmatrix}1 & 0 & 0\\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}}\end{bmatrix}\begin{bmatrix}a\\ b\\ c\end{bmatrix} \qquad \textit{Eqn. 18}$$

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

$$i\_alpha = i\_a$$

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

- **Calculates Forward Park Transformation.**

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

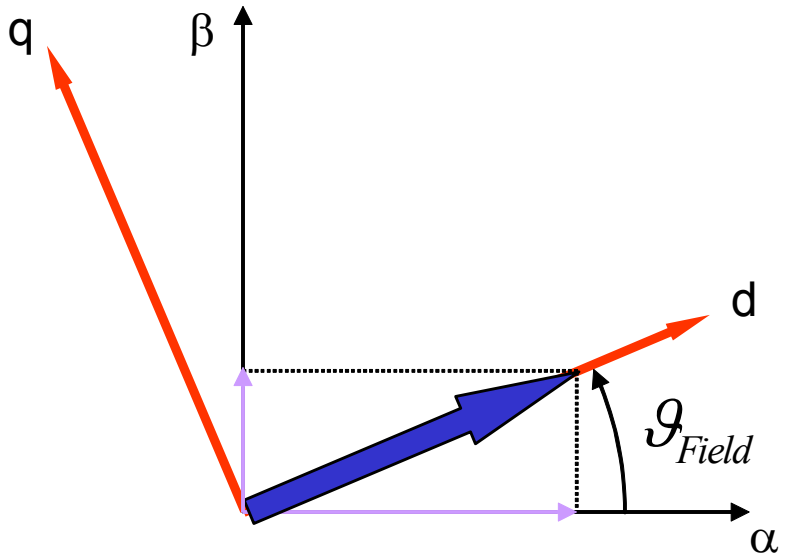

Figure 3. Park Transformation

To transfer the graphical representation into mathematical language:

$$\begin{bmatrix} d \\ q \end{bmatrix} = \begin{bmatrix} \cos\vartheta_{Field} & \sin\vartheta_{Field} \\ -\sin\vartheta_{Field} & \cos\vartheta_{Field} \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad \textit{Eqn. 19}$$

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

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

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

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

$$u(t) = K\left[e(t) + \frac{1}{T_I}\int_0^{\tau} e(\tau)d\tau + T_D\frac{de(t)}{dt}\right] \quad \textit{Eqn. 20}$$

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:

$$u(k) = Ke(k) + K\frac{T}{T_I}e(k) + u(k-1) + K\frac{T_D}{T}(e(k) - e(k-1)) \quad \textit{Eqn. 21}$$

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 PMSMVC 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 $G_I = K*T/T_I$
G_D – Derivative gain $G_D = K*T_D/T$

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. The 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).

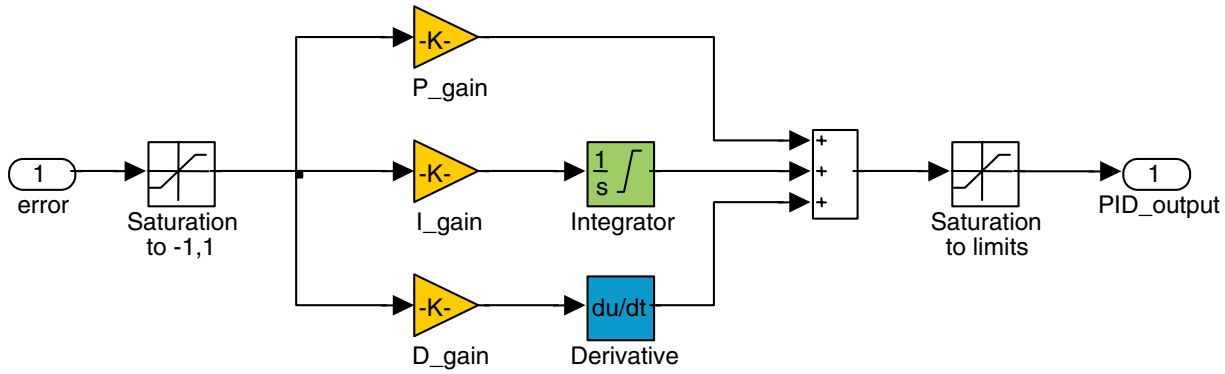

Figure 4. PID Controller Structure

The PMSMVC 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:

$$u\_d = PID\_controller(i\_d\_required - i\_d)$$
$$u\_q = PID\_controller(i\_q\_required - i\_q)$$

- **Calculates DQ decoupling and Q feed forward.**

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 control) if the stator voltage equations are decoupled. Then they are indirectly controlled by controlling the terminal voltages of the induction motor.

The equations of the stator voltage components in the d-q co-ordinate system Eqn. 15 and Eqn. 16 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:

$$u_{Sd} = u_{sd}^{lin} + u_{sd}^{decouple} = \left[R_S i_{Sd} + L_S \frac{d}{dt} i_{Sd}\right] - [L_S \omega_F i_{Sq}] \quad \textbf{Eqn. 22}$$

$$u_{Sq} = u_{sq}^{lin} + u_{sq}^{decouple} = \left[R_S i_{Sq} + L_S \frac{d}{dt} i_{Sq}\right] + [L_S \omega_F i_{Sd} + \omega_F \Psi_M] \quad \textbf{Eqn. 23}$$

The voltage components $u_{sd}^{lin}, u_{sq}^{lin}$ are the outputs of the current controllers which control the 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 that the voltage on the outputs of the current controllers is:

$$u_{sd}^{lin} = R_S i_{Sd} + L_S \frac{d}{dt} i_{Sd} \quad \textbf{Eqn. 24}$$

$$u_{sq}^{lin} = R_S i_{Sq} + L_S \frac{d}{dt} i_{Sq} \quad \textbf{Eqn. 25}$$

And the decoupling components are:

$$u_{sd}^{decouple} = -L_S \omega_F i_{Sq} \quad \textbf{Eqn. 26}$$

$$u_{sq}^{decouple} = L_S \omega_F i_{Sd} + \omega_F \Psi_M \quad \textbf{Eqn. 27}$$

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

The PMSMVC calculates the following in order to decouple the controller outputs *u_d* and *u_q*:

$$u\_d = u\_d - omega\_actual \times Lq \times i\_q$$

$$u\_q = u\_q + omega\_actual \times Ld \times i\_d + omega\_actual \times Ke$$

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_actual – Actual motor electrical velocity
Ld – Motor induction in D-coordinate
Lq – Motor induction in Q-coordinate
The *Ld* and *Lq* are equal for most type of motors
Ke – Motor electrical constant

- **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.

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \cos\vartheta_{Field} & -\sin\vartheta_{Field} \\ \sin\vartheta_{Field} & \cos\vartheta_{Field} \end{bmatrix} \begin{bmatrix} d \\ q \end{bmatrix} \qquad \textit{Eqn. 28}$$

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

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

- **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 DCBus 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 [-].

The following figures 5 and 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.

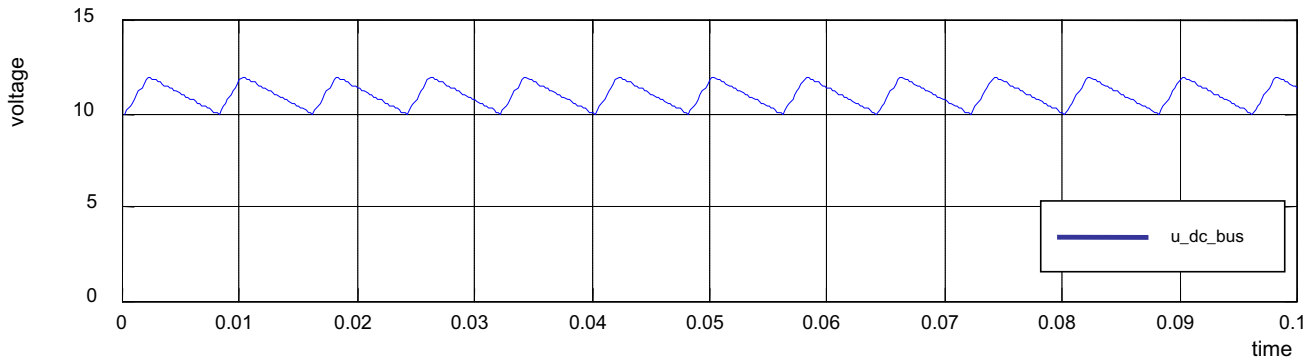

Figure 5. Measured Voltage on the DC-Bus

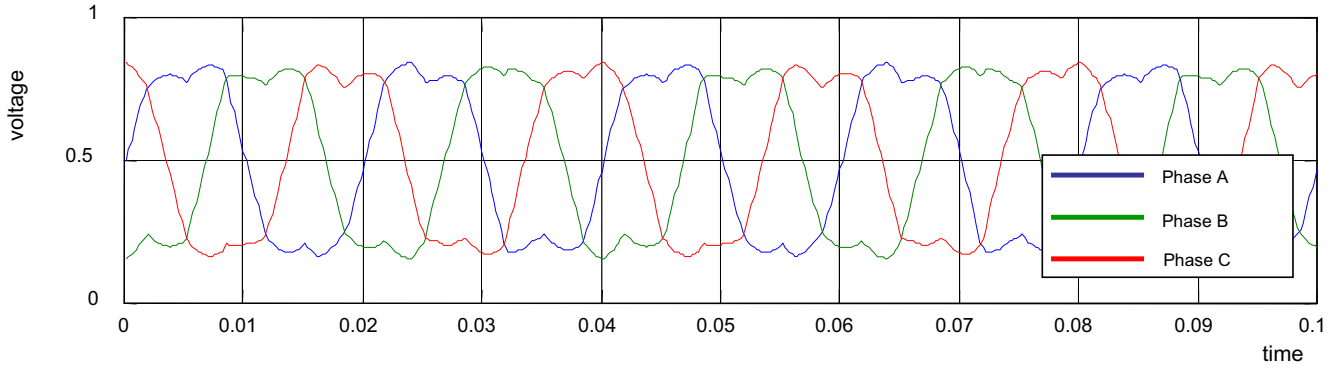

Figure 6. Standard Space Vector Modulation with Elimination of the DC_bus Ripple

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

- **Master Mode**

  The PMSMVC update is executed periodically with a given period.

- **Slave Mode**

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

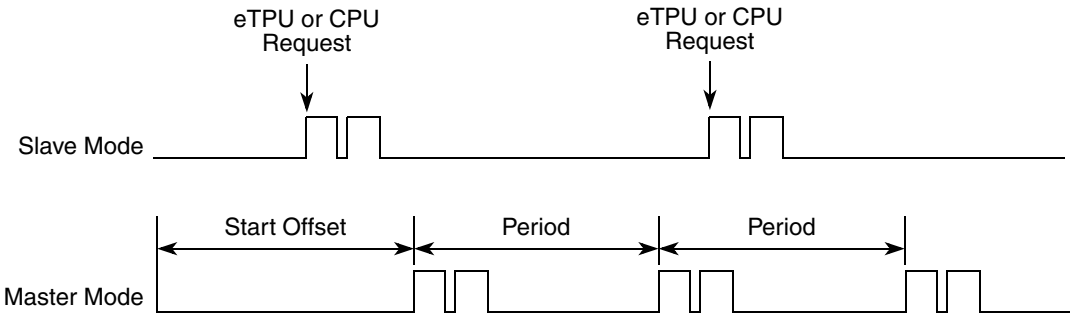

Figure 7. PMSMVC Updates in Slave Mode and Master Mode

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

4.1 Interrupts

The PMSMVC function generates an interrupt service request to the CPU 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 PMSMVC 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 PMSMVC 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 PMSMVC function.

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

Table 1 lists the maximum eTPU busy-times per update period in eTPU cycles that depend on the PMSMVC 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	766	546
Master mode, Circle limitation ON	1010	546
Slave mode, Circle limitation OFF	754	546

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]
Slave mode, Circle limitation ON	998	546

On MPC5500 devices, the eTPU module clock is equal to the CPU clock. On MCF523x devices, it is 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 PMSMVC function. Use of these functions eliminates the need to directly control the eTPU registers.

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

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

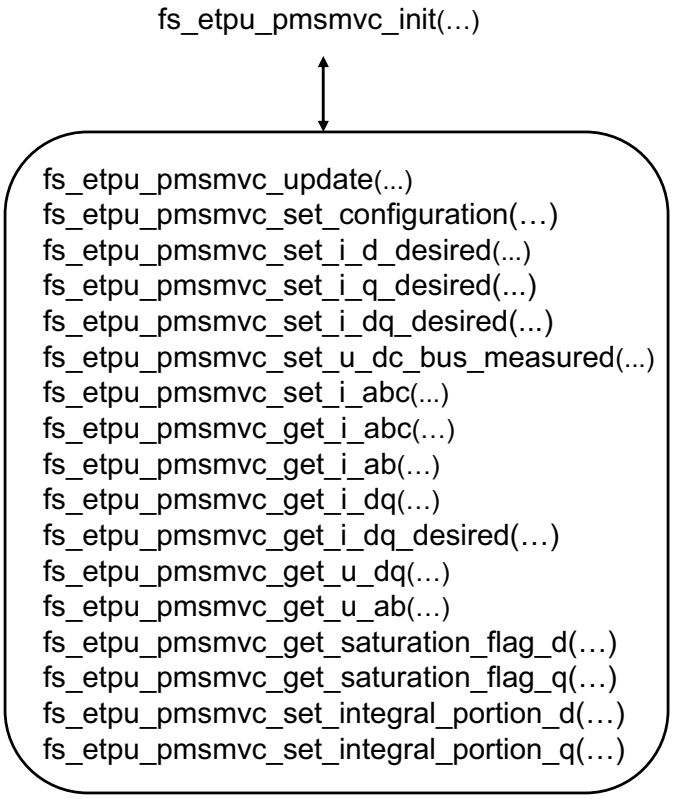

Figure 8. PMSMVC API State Flow

All PMSMVC API routines are described in order and listed below:

- Initialization functions:

```
int32_t fs_etpu_pmsmvc_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,
                             uint8_t QD_RSLV_chan,
                            uint24_t qd_pc_per_rev,
                             uint8_t rslv_pole_pairs,
                             uint8_t motor_pole_pairs,
                pmsmvc_motor_params_t* p_motor_params,
                  pmsmvc_pid_params_t* p_pid_d_params,
```

```
                                pmsmvc_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_pmsmvc_set_configuration( uint8_t  channel,
                                          uint8_t configuration)

int32_t fs_etpu_pmsmvc_update(uint8_t channel)

int32_t fs_etpu_pmsmvc_set_i_d_desired(uint8_t  channel,
                                       fract24_t  i_d_desired)

int32_t fs_etpu_pmsmvc_set_i_q_desired(uint8_t  channel,
                                       fract24_t  i_q_desired)

int32_t fs_etpu_pmsmvc_set_i_dq_desired(uint8_t  channel,
                                        pmsmvc_dq_t * p_i_dq_desired)

int32_t fs_etpu_pmsmvc_set_u_dc_bus_measured(uint8_t  channel,
                                          ufract24_t  u_dc_bus_measured)

int32_t fs_etpu_pmsmvc_set_i_abc(uint8_t  channel,
                                 pmsmvc_abc_t * p_i_abc)

int32_t fs_etpu_pmsmvc_set_integral_portion_d( uint8_t channel,
                                               fract24_t i_k1)

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

- Value return functions:

```
int32_t fs_etpu_pmsmvc_get_i_abc(uint8_t channel,
                                 pmsmvc_abc_t * p_i_abc)

int32_t fs_etpu_pmsmvc_get_i_ab(uint8_t channel,
                                pmsmvc_ab_t * p_i_ab)
```

```
int32_t fs_etpu_pmsmvc_get_i_dq(uint8_t channel,
                                pmsmvc_dq_t * p_i_dq)

int32_t fs_etpu_pmsmvc_get_i_dq_desired(uint8_t channel,
                                        pmsmvc_dq_t * p_i_dq_desired);

int32_t fs_etpu_pmsmvc_get_u_dq(uint8_t channel,
                                pmsmvc_dq_t * p_u_dq)

int32_t fs_etpu_pmsmvc_get_u_ab(uint8_t channel,
                                pmsmvc_ab_t * p_u_ab)

uint8_t fs_etpu_pmsmvc_get_saturation_flag_d(uint8_t channel)

uint8_t fs_etpu_pmsmvc_get_saturation_flag_q(uint8_t channel)
```

5.1 Initialization Function

5.1.1 int32_t fs_etpu_pmsmvc_init(...)

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

- **channel (uint8_t)**—The PMSMVC 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 PMSMVC 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_PMSMVC_MASTER
  - FS_ETPU_PMSMVC_SLAVE
- **circle_limitation_config (uint8_t)**—The required configuration of circle limitation; should be assigned one of these values:
  - FS_ETPU_PMSMVC_CIRCLE_LIMITATION_OFF
  - FS_ETPU_PMSMVC_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_PMSMVC_MASTER).
- **start_offset (uint24_t)**—Used to synchronize various eTPU functions that generate a signal. The first PMSMVC update starts the start_offset TCR1 clocks after initialization. This parameter applies in the master mode only (mode=FS_ETPU_PMSMVC_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 PMSMVC 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.
- **QD_RSLV_chan (uint8_t)**—The number of a channel the QD or RSLV function is assigned to. The PMSMVC reads the actual motor position from QD or RSLV. This parameter should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **qd_pc_per_rev (uint24_t)**—The number of QD position counter increments per revolution.
- **rslv_pole_pairs (uint8_t)**—Defines the number of resolver pole-pairs.
- **motor_pole_pairs (uint8_t)**—Defines the number of motor pole-pairs.
- **p_motor_params (pmsmvc_motor_params_t*)**—The pointer to a pmsmvc_motor_params_t structure of motor constants. The pmsmvc_motor_params_t structure is defined in `etpu_pmsmvc.h`:

```
typedef struct {
   fract24_t Ke;  /* motor electrical constant */
   fract24_t Ld;  /* motor induction in D-coordinates in fractional format 2.22*/
   fract24_t Lq;  /* motor induction in Q-coordinates in fractional format 2.22*/
} pmsmvc_motor_params_t;
```

  Where:
  - **Ke (fract24_t)** is the motor electrical constant. Its value must be scaled to nominal range:
    Ke[-] = 2*pi * Ke[V/RPM] * speed_range[RPM] / dc_bus_voltage_range[V]
    and then expressed in fractional format 3.21:
    Ke[fract 3.21] = 0x200000 * Ke[-]
  - **Ld (fract24_t)** is the motor induction in D-coordinate. Its value must be scaled to nominal range:
    Ld[-] = 2*pi * Ld[H] * speed_range[RPM] * current_range[A] / (60 * dc_bus_voltage_range[V])

and then expressed in fractional format 3.21:
Ld[fract 3.21] = 0x200000 * Ld[-]

— **Lq (fract24_t)** is the motor induction in Q-coordinate. Its value must be scaled the same way as Ld.

The Ld and Lq are usually (but not always) equal.

- **p_pid_d_params (pmsmvc_pid_params_t*)**—The pointer to a pmsmvc_pid_params_t structure of D-coordinate PID controller parameters. The pmsmvc_pid_params_t structure is defined in `etpu_pmsmvc.h`:

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

Where:

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

— **D_gain (fract24_t)** is the derivative gain and its value must be in the 24-bit signed fractional format 9.15, which 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 and its value must be in the 16-bit signed fractional format 1.15, which means in the range of (-1, 1).

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

- **p_pid_q_params (pmsmvc_pid_params_t*)**—The pointer to a pmsmvc_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_PMSMVC_INVMODINDEX_SINE

— FS_ETPU_PMSMVC_INVMODINDEX_SIN3H

— FS_ETPU_PMSMVC_INVMODINDEX_SVM

- **output_chan (uint8_t)**—PMSMVC 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)**—PMSMVC 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 PMSMVC updates output. Usually PMSMVC updates PWMMAC inputs, and this is why it should be a PWMMAC channel. 0-31 for ETPU_A and 64-95 for ETPU_B.
- **compensation_delay (uint24_t)**—The delay from the point where the phase currents and motor position are measured to the point where the new PWM duty-cycles are applied, expressed as: compensation_delay = Comp_Delay[TCR1] × Omega_Range[RPM] / (30 × freq_tcr1[Hz]) Typically, 150% of the PWM period length is used for Comp_Delay[TCR1]. If this parameter is assigned 0, the delay compensation task is turned off.

5.2 Change Operation Functions

5.2.1 int32_t fs_etpu_pmsmvc_set_configuration(uint8_t channel, uint8_t configuration)

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

- **channel (uint8_t)**—The PMSMVC 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 PMSMVC; should be assigned one of these values:
  - FS_ETPU_PMSMVC_PID_OFF (DQ PID controllers are disabled)
  - FS_ETPU_PMSMVC_PID_ON (DQ PID controllers are enabled)

5.2.2 int32_t fs_etpu_pmsmvc_update(uint8_t channel)

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

- **channel (uint8_t)**—The PMSMVC 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_pmsmvc_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 PMSMVC 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 the range MIN24 to MAX24.

5.2.4 int32_t fs_etpu_pmsmvc_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. This function has the following parameters:

- **channel (uint8_t)** - This is the PMSMVC channel number. This parameter 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_pmsmvc_set_i_dq_desired(uint8_t channel, pmsmvc_dq_t * p_i_dq_desired)

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

- **channel (uint8_t)** - This is the PMSMVC channel number. This parameter must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_dq_desired (pmsmvc_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_pmsmvc_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. This function has these parameters:

- **channel (uint8_t)**—The PMSMVC 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 convertor range.

5.2.7 int32_t fs_etpu_pmsmvc_set_i_abc(uint8_t channel, pmsmvc_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 PMSMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_abc (pmsmvc_abc_t*)**—Pointer to structure of phase currents in 3-phase stationary reference frame.

5.2.8 int32_t fs_etpu_pmsmvc_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 PMSMVC 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_pmsmvc_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 PMSMVC 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_pmsmvc_get_i_abc(uint8_t channel, pmsmvc_abc_t * p_i_abc)

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

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

5.3.2 int32_t fs_etpu_pmsmvc_get_i_ab(uint8_t channel, pmsmvc_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 PMSMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_ab (pmsmvc_ab_t*)**—Pointer to structure of phase currents in 2-phase orthogonal stationary reference frame.

5.3.3 int32_t fs_etpu_pmsmvc_get_i_dq(uint8_t channel, pmsmvc_dq_t * p_i_dq)

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

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

5.3.4 int32_t fs_etpu_pmsmvc_get_i_dq_desired(uint8_t channel, pmsmvc_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 PMSMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_i_dq_desired (pmsmvc_dq_t*)**—Pointer to return structure of phase currents in 2-phase orthogonal rotating reference frame.

5.3.5 int32_t fs_etpu_pmsmvc_get_u_dq(uint8_t channel, pmsmvc_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 PMSMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_u_dq (pmsmvc_dq_t*)**—Pointer to structure of stator voltages in 2-phase orthogonal rotating reference frame.

5.3.6 int32_t fs_etpu_pmsmvc_get_u_ab(uint8_t channel, pmsmvc_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 PMSMVC channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_u_ab (pmsmvc_ab_t*)**—Pointer to structure of stator voltages in 2-phase orthogonal stationary reference frame.

5.3.7 uint8_t fs_etpu_pmsmvc_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 PMSMVC 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_PMSMVC_SATURATION_NO (0)... no saturation

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

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

5.3.8 uint8_t fs_etpu_pmsmvc_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 PMSMVC 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_PMSMVC_SATURATION_NO (0)... no saturation

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

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

6 Example Use of Function

6.1 Demo Applications

The use of the PMSMVC eTPU function is demonstrated in the following application notes:

- “Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MCF523x,” AN3002.
- “Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MPC5500,” AN3206.
- “Permanent Magnet Synchronous Motor with Resolver, Vector Control, Driven by eTPU on MPC5500,” AN4480.

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

6.1.1 Function Calls

The PMSMVC 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 = 1400;
int32_t dc_bus_voltage_range_mv = 24240;
int32_t phase_current_range_ma = 1947;
int32_t PMSM_Ke_mv_per_krpm = 8400;
int32_t PMSM_L_uH = 6320;
uint8_t PMSM_pole_pairs = 2;
int32_t PMSMVC_D_PID_gain_permil = 500;
int32_t PMSMVC_D_I_time_const_us = 2500;
int32_t PMSMVC_Q_PID_gain_permil = 500;
int32_t PMSMVC_Q_I_time_const_us = 2500;
uint8_t PMSMVC_channel = 5;
uint8_t SC_channel = 6;
uint8_t PWM_master_channel = 7;
int32_t PWM_freq_hz = 20000;
uint8_t QD_phaseA_channel = 1;
int24_t QD_pc_per_rev = 2000;
/*************************************************************************
* Initialize PMSM Vector Control
*************************************************************************/
/*************************************************************************
 * 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.




 *************************************************************************/
  pmsmvc_pid_d_params.P_gain = PMSMVC_D_PID_gain_permil*0x001000/125;
  pmsmvc_pid_d_params.I_gain = 0x008000*1000/PWM_freq_hz*PMSMVC_D_PID_gain_permil
```

```
                                /PMSMVC_D_I_time_const_us;
   pmsmvc_pid_d_params.D_gain = 0;
   pmsmvc_pid_d_params.positive_limit = 0x7FFF;
   pmsmvc_pid_d_params.negative_limit = 0x8000;
/*************************************************************************
 * 2) Define Q-Current Controller PID Parameters
 *************************************************************************/
   pmsmvc_pid_q_params.P_gain = PMSMVC_Q_PID_gain_permil*0x001000/125;
   pmsmvc_pid_q_params.I_gain = 0x008000*1000/PWM_freq_hz*PMSMVC_Q_PID_gain_permil
                                /PMSMVC_Q_I_time_const_us;
   pmsmvc_pid_q_params.D_gain = 0;
   pmsmvc_pid_q_params.positive_limit = 0x7FFF;
   pmsmvc_pid_q_params.negative_limit = 0x8000;
/*************************************************************************
 * 3) Define Motor Parameters
*************************************************************************/
   pmsmvc_motor_params.Ke = 2*3.1415927*PMSM_Ke_mv_per_krpm*speed_range_rpm/
                            dc_bus_voltage_range_mv*(0x400000/1000);
   pmsmvc_motor_params.Ld = 2*3.1415927*PMSM_L_uH*speed_range_rpm/
             dc_bus_voltage_range_mv*phase_current_range_ma/(60000000/0x400000);
   pmsmvc_motor_params.Lq = pmsmvc_motor_params.Ld;
/*************************************************************************
 * 4) Initialize PMSMVC channel
 *************************************************************************/
   err_code = fs_etpu_pmsmvc_init(
      PMSMVC_channel,/* channel */
      FS_ETPU_PRIORITY_LOW,/* priority */
      FS_ETPU_PMSMVC_SLAVE,/* mode */
      FS_ETPU_PMSMVC_CIRCLE_LIMITATION_ON, /* circle_limitation_config */
      0, /* period */
      0, /* start_offset */
      0, /* services_per_irq */
      SC_channel, /* SC_chan */
      QD_phaseA_channel, /* QD_RSLV_chan */
      QD_pc_per_rev, /* qd_pc_per_rev */
      0, /* rslv_pole_pairs */
      PMSM_pole_pairs, /* motor_pole_pairs */
      &pmsmvc_motor_params, /* p_motor_params */
      &pmsmvc_pid_d_params, /* p_pid_d_params */
      &pmsmvc_pid_q_params, /* p_pid_q_params */
      FS_ETPU_PMSMVC_INVMODINDEX_SINE, /* inv_mod_index */
      PWM_master_channel, /* output_chan */
      FS_ETPU_PWMMAC_INPUTS_OFFSET,/* output_offset */
      PWM_master_channel,/* link_chan */
       0x800000*1.5*speed_range_rpm/(30*PWM_freq_hz)
   );
```

7 Summary and Conclusions

This application note provides the user with a description of the PMSMVC eTPU function. The simple C interface routines to the PMSMVC eTPU function enable easy implementation of the PMSMVC 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. “Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MCF523x,” AN3002
8. “Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MPC5500,” AN3206
9. “Permanent Magnet Synchronous Motor with Resolver, Vector Control, Driven by eTPU on MPC5500,” AN4480

8 Revision history

Table 2. Revision history

Revision number	Revision date	Description of changes
2	02 May 2012	Updated for support of motor drives with resolver position sensor.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN2972
Rev. 2
02/2012