Freescale Semiconductor
Application Note

AN3002
Rev. 0, 10/2005

Permanent Magnet Synchronous Motor Vector Control, Driven by eTPU on MCF523x

Covers MCF523x and all eTPU-equipped Devices

by: Milan Brejl & Michal Princ
System Application Engineers
Roznov Czech System Center

This application note describes the design of a 3-phase permanent magnet synchronous motor (PMSM) speed and torque vector control drive based on Freescale's ColdFire MCF523x microprocessor. The application design takes advantage of the enhanced time processing unit (eTPU) module, which is used as a motor control co-processor. The eTPU handles the motor control processing, eliminating the microprocessor overhead for other duties.

PMSMs are very popular in a wide array of applications. Compared to a DC motor, the PMSM misses a commutator, therefore it is more reliable than a DC motor. The PMSM also has advantages when compared to an AC induction motor. The PMSM generates the rotor magnetic flux with rotor magnets, achieving higher efficiency. Therefore, the PMSM is used in high-end white goods (refrigerators, washing machines, dishwashers, etc.), high-end pumps, fans, and in other appliances that require high reliability and efficiency.

The concept of the application is to create a vector control PMSM driver with optional speed closed-loop, using a quadrature encoder. It serves as an example of a PMSM vector control system design using a Freescale

Table of Contents

microprocessor with the eTPU. It also illustrates the usage of dedicated motor control eTPU functions that are included in the AC motor control eTPU function set.

This application note also includes basic motor theory, system design concept, hardware implementation, and microprocessor and eTPU software design, including the FreeMASTER visualization tool.

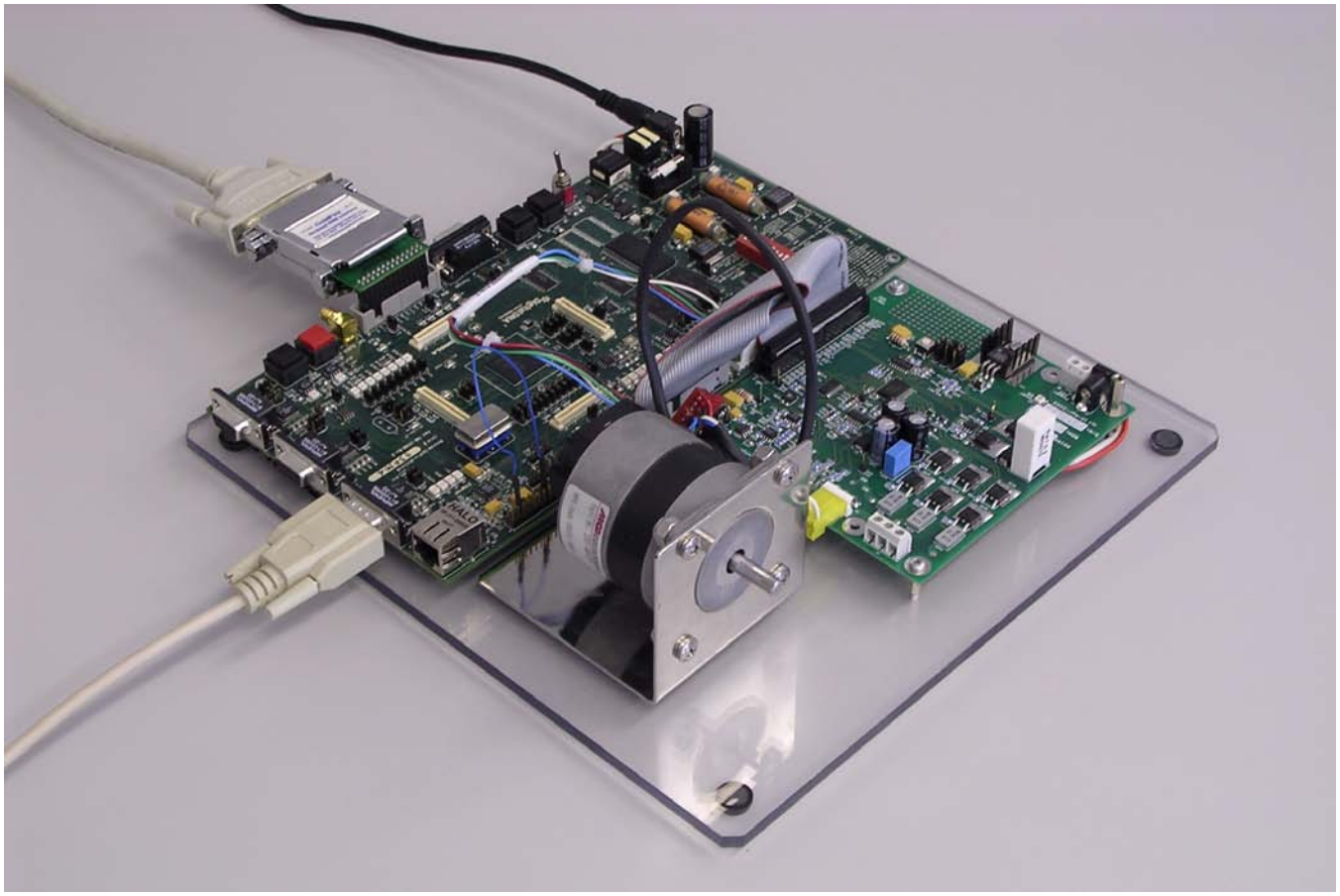

Figure 1. Using the M523xEVB, 33395 Evaluation Motor Board, and MCG BLDC Motor

1 ColdFire MCF523x and eTPU Advantages and Features

1.1 ColdFire MCF523x Microprocessor

The MCF523x family is composed of highly-integrated, 32-bit microprocessors based on the V2 ColdFire core. It features a 16- or 32-channel eTPU, 64 Kbytes of internal SRAM, a 2-bank SDRAM controller, four 32-bit timers with DMA request capability, a 4-channel DMA controller, up to two CAN modules, three UARTs, and a queued SPI. The MCF523x family has been designed for general purpose industrial control applications. It is also a high-performance upgrade for users of the MC68332.

This 32-bit device is based on the version 2 ColdFire reduced instruction set computer (RISC) core, operating at a core frequency of up to 150 MHz and a bus frequency of up to 75 MHz. On-chip modules include the following:

- V2 ColdFire core with an enhanced multiply-accumulate unit (EMAC) providing 144 Dhrystone 2.1MIPS @ 150 MHz
- eTPU with 16 or 32 channels, 6 Kbytes of code memory, and 1.5 Kbytes of data memory with eTPU debug support
- 64 Kbytes of internal SRAM
- External bus speed of half the CPU operating frequency (75-MHz bus @ 150-MHz core)
- 10/100 Mbps bus-mastering Ethernet controller
- 8 Kbytes of configurable instruction/data cache
- Three universal asynchronous receiver/transmitters (UARTs) with DMA support
- Controller area network 2.0B (FlexCAN module)
  - Optional second FlexCAN module multiplexed with the third UART
- Inter-integrated circuit (I2C) bus controller
- Queued serial peripheral interface (QSPI) module
- Hardware cryptography accelerator (optional)
  - Random number generator
  - DES/3DES/AES block cipher engine
  - MD5/SHA-1/HMAC accelerator
- 4-channel, 32-bit direct memory access (DMA) controller
- 4-channel, 32-bit input capture/output compare timers with optional DMA support
- 4-channel, 16-bit periodic interrupt timers (PITs)
- Programmable software watchdog timer
- Interrupt controller capable of handling up to 126 interrupt sources
- Clock module with phase locked loop (PLL)
- External bus interface module including a 2-bank synchronous DRAM controller
- 32-bit, non-multiplexed bus with up to 8 chip select signals that support page-mode Flash memories

For more information, refer to Reference 1.

1.2 eTPU Module

The eTPU is an intelligent, semi-autonomous co-processor designed for timing control, I/O handling, serial communications, motor control, and engine control applications. It operates in parallel with the host CPU. The eTPU processes instructions and real-time input events, performs output waveform generation, and accesses shared data without the host CPU's intervention. Consequently, the host CPU setup and service times for each timer event are minimized or eliminated.

The eTPU has up to 32 timer channels, in addition to having 6 Kbytes of code memory and 1.5 Kbytes of data memory that store software modules downloaded at boot time, and can be mixed and matched as needed for any application.

The eTPU provides more specialized timer processing than the host CPU can achieve. This is partially due to the eTPU implementation, which includes specific instructions for handling and processing time events. In addition, channel conditions are available for use by the eTPU processor, thus eliminating many branches. The eTPU creates no host CPU overhead for servicing timing events.

For more information, refer to Reference 7.

2 Target Motor Theory

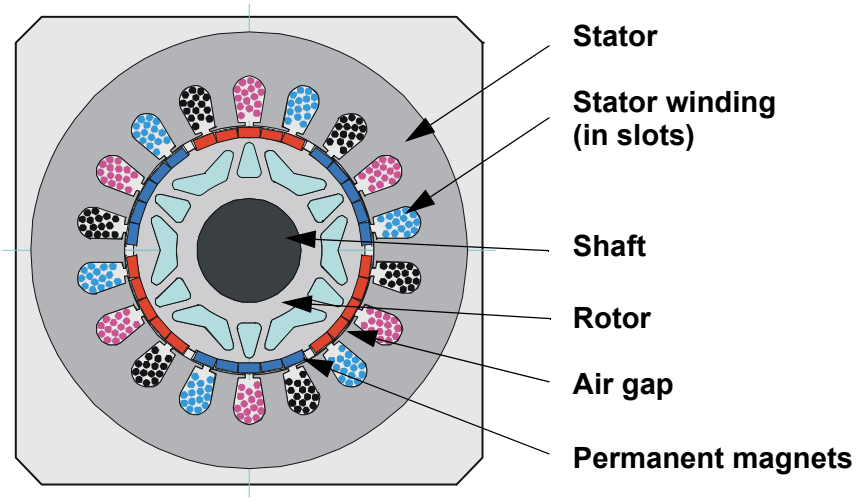

Figure 2. Permanent Magnet AC Machine - Cross Section

Permanent magnet AC (PMAC) machines provide automotive actuator designers with a unique set of features and capabilities. There are two principal classes of permanent magnet AC machines: sinusoidally excited machines or permanent magnet synchronous motors (PMSM), and trapezoidaly excited machines or brushless DC (BLDC) motors. The difference is that while stator windings of trapezoidal PMAC machines are concentrated into a narrow-phase pole, the windings of a sinusoidal machine are typically distributed over multiple slots in order to approximate a sinusoidal distribution. These differences in construction are reflected in their corresponding motion characteristics as well: the first type of PMAC provides sinusoidal back-electromotive force (back-EMF) generation, while the second type provides trapezoidal back-EMF.

PMSM machines have the unique advantages of unsurpassed efficiency and power density characteristics, which are primarily responsible for their wide appeal. On the other hand, PMSM machines are synchronous, which certainly requires accompanying power electronics, but they also provide the basis for achieving high-quality actuator control. The torque ripple associated with sinusoidal PMAC (PMSM) machines is generally less than that developed in trapezoidal machines, one reason sinusoidal motors are

preferred in high-performance motion control applications, such as electro-mechanical braking. This application note targets the PMSM only.

2.1 Digital Control of PMSM

For the common 3-phase PMSM motor, a standard 3-phase power stage is used (see Figure 3). The power stage utilizes six power transistors that operate in complementary mode.

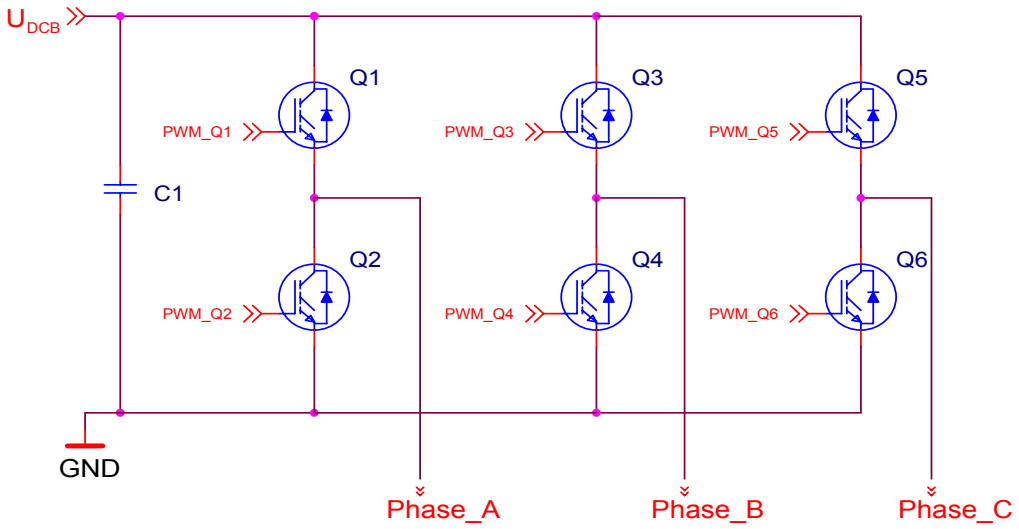

Figure 3. 3-Phase BLDC Power Stage

The inverter consists of three half-bridge units where the upper and lower switches are controlled complimentarily, meaning when the upper one is turned on, the lower one must be turned off, and vice versa. Because the power device's turn-off time is longer than its turn-on time, some dead time must be inserted between turning off one transistor of the half-bridge and turning on its complementary device. The output voltage is mostly created by the pulse width modulation (PWM) technique. The 3-phase voltage waves are shifted 120^{o} to one another; thus a 3-phase motor can be supplied.

2.2 Vector Control of PMSM

Vector control is an elegant 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 an electro-magnetic field generating part and a torque generating part. Then the structure of the motor controller (vector control controller) is almost the same as for a separately excited DC motor, which simplifies the control of the PMSM. This vector control technique was developed to achieve the same excellent, dynamic performance of the PMSM.

As explained in Figure 4, 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 torque generating part of the stator current allows separate control of the magnetic flux and the torque. In order 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.’ For more information, refer to Reference 13.

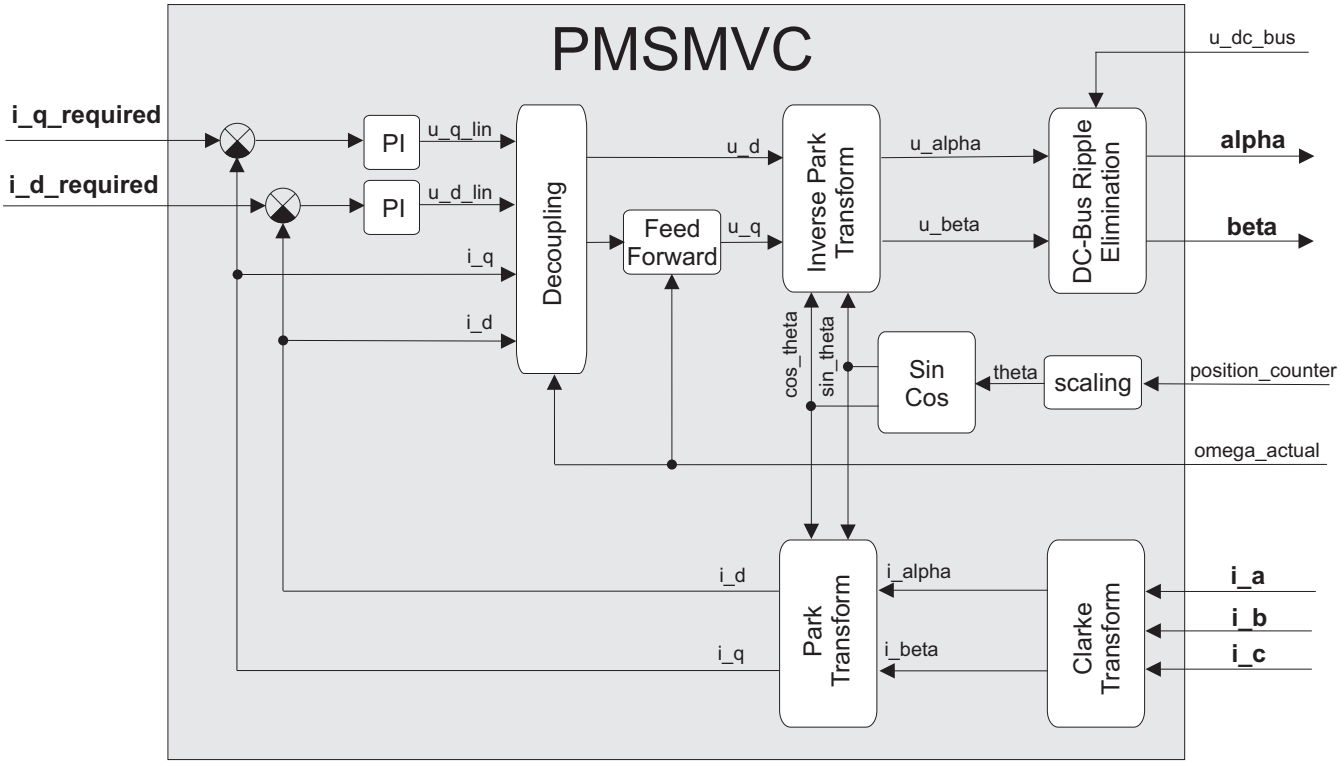

Figure 4. PMSM Vector Control Current Loop Block Diagram

2.3 Quadrature Encoder

The PMSM motor application uses the quadrature encoder for rotor position sensing. The quadrature encoder output consists of three signals. Two phases, A and B, represent the rotor position, and an index pulse defines the zero position. All quadrature encoder signals are depicted in Figure 5.

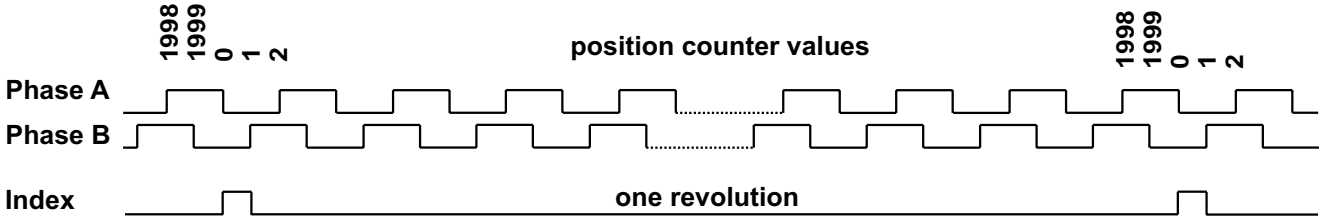

Figure 5. Quadrature Encoder Output Signals

2.3.1 Position Alignment

Since the quadrature encoder does not give the absolute position, we need to know the exact rotor position before the motor is started. One possible and very easily implemented method is the rotor alignment to a

predefined position. The motor is powered by a defined static voltage pattern and the rotor aligns to the predefined position. This alignment is done during the motor start up.

3 System Concept

3.1 System Outline

The system is designed to drive a 3-phase PMSM. The application meets the following performance specifications:

- Voltage control of a PMSM using quadrature encoder HEDS-5640 A06
- Targeted at ColdFire MCF523x evaluation board (M523xEVB), 33395 evaluation motor board (power stage), and MCG PMSM (IB23810)
- Control technique incorporates:
  - PMSM vector control with optional speed-closed loop
  - Both directions of rotation
  - 4-quadrant operation
  - Rotor alignment to the start position
  - Minimum speed of 10 RPM
  - Maximum speed of 1000 RPM (limited by power supply)
- Manual interface (start/stop switch, up/down push button control, LED indication)
- FreeMASTER control interface (speed set-up, speed control/torque control choice)
- FreeMASTER monitor
  - FreeMASTER graphical control page (required speed, required torque, actual motor speed, actual torque, start/stop status, fault status)
  - FreeMASTER control scope (observes required and actual speeds and torques, applied voltage)
  - Detail description of all eTPU functions used in the application (monitoring of channel registers and all function parameters in real time)
- DC bus over-current fault protection

3.2 Application Description

A standard system concept is chosen for the motor control function (see Figure 6). The system incorporates the following hardware:

- Evaluation board M523xEVB
- 33395 evaluation motor board
- MCG PMSM (IB23810)

- Quadrature encoder (HEDS-5640 A06)
- Power supply 9V DC, 2.7Amps

The eTPU module runs the main control algorithm. The 3-phase PWM output signals for a 3-phase inverter are generated, using the vector control algorithm according to feedback signals from the quadrature encoder, actual values of phase currents, and the input variable values provided by the microprocessor CPU. The phase currents are sampled by the external analog-to-digital converter connected to the QSPI peripheral. The transfer of phase current samples from the QSPI to eTPU DATA RAM is provided by the CPU, because a DMA transfer is not applicable here.

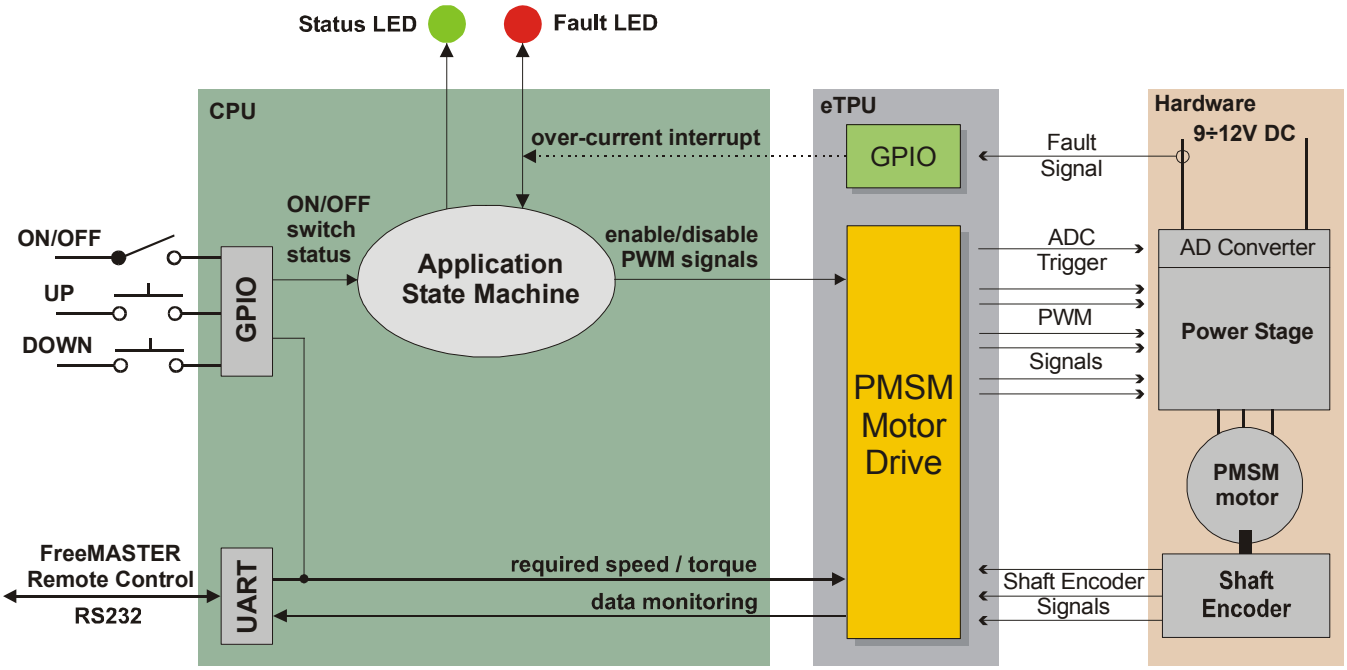

Figure 6. System Concept

The system processing is distributed between the CPU and the eTPU, which both run in parallel.

The CPU performs the following tasks:

- Periodically scans the user interface (ON/OFF switch, up and down buttons, FreeMASTER). Based on the user input, it handles the application state machine and calculates the required speed or torque, which is passed to the eTPU.
- Periodically reads application data from eTPU DATA RAM in order to monitor application variables.
- In the event of an over-current fault, immediately the PWM outputs are temporarily disabled by the eTPU hardware. Then, after an interrupt latency, the CPU disables the PWM outputs permanently and displays the fault state.
- Periodically, with a period of 50us, the CPU transfers AD converter result queue received by QSPI into eTPU DATA RAM.

The eTPU performs the following tasks:

- Six eTPU channels (PWMF) are used to generate PWM output signals.
- Three eTPU channels (QD) are used to process quadrature encoder signals.
- One eTPU channel (BC) is used for controlling the DC-bus break.
- One eTPU channel (ASAC) is used to trigger the external AD converter and preprocess the sampled values.
- One eTPU channel (GPIO) is used to generate an interrupt to the CPU when the over-current fault signal activates.
- One eTPU channel (PWMMAC) is internally used to synchronize the PWM outputs and calculate space vector modulation, based on applied motor voltage vector Alpha-Beta coordinates.
- One eTPU channel (SC) is internally used to calculate the actual motor speed and control a speed-closed loop in case of speed vector control. The actual motor speed is calculated based on the QD position counter and QD last edge time. The required speed is provided by the CPU and passed through a ramp. The speed PI control algorithm processes the error between the required and actual speed. The PI controller output is passed to the PMSMVC eTPU function as a newly corrected value of the required motor torque.
- One eTPU channel (PMSMVC) is internally used to calculate the current vector control closed loop. It takes the values of actual phase currents from ASDC, the actual rotor position from QD, and the actual motor speed from SC. The phase currents are transferred to Alpha-Beta and D-Q coordinate system, using Clark and Park transformations. One PI control algorithm processes the error between the required and actual D-current, and another one processes the error between the required and actual Q-current. The required value of the D-current, which is the flux controlling current, is set to zero. The required value of the Q-current, which is the torque controlling current, is either set directly by the CPU (torque vector control) or provided by the SC output (speed vector control). The PI controller outputs create the motor applied voltage vector in a D-Q coordinate system. It is transformed back to Alpha-Beta coordinate system and passed to PWM generator.

3.2.1 User Interface

The application is interfaced by the following:

- ON/OFF switch on M523xEVB
- Up/down buttons on M523xEVB or
  FreeMASTER running on a PC connected to the M523xEVB via an RS232 serial cable

The ON/OFF switch affects the application state and enables and disables the PWM phases. When the switch is in the off position, no voltage is applied to the motor windings. When the ON/OFF switch is in the on position, the motor speed can be controlled either by the up and down buttons on the M523xEVB, or by the FreeMASTER on the PC. The FreeMASTER also displays a control page, real-time values of application variables, and their time behavior using scopes.

FreeMASTER software was designed to provide an application-debugging, diagnostic, and demonstration tool for the development of algorithms and applications. It runs on a PC connected to the M523xEVB via

an RS232 serial cable. A small program resident in the microprocessor communicates with the FreeMASTER software to return status information to the PC and process control information from the PC. FreeMASTER software executing on a PC uses part of Microsoft Internet Explorer as the user interface.

Note that FreeMASTER version 1.2.31.1 or higher is required. The FreeMASTER application can be downloaded from http://www.freescale.com. For more information about FreeMASTER, refer to Reference 6.

3.3 Hardware Implementation and Application Setup

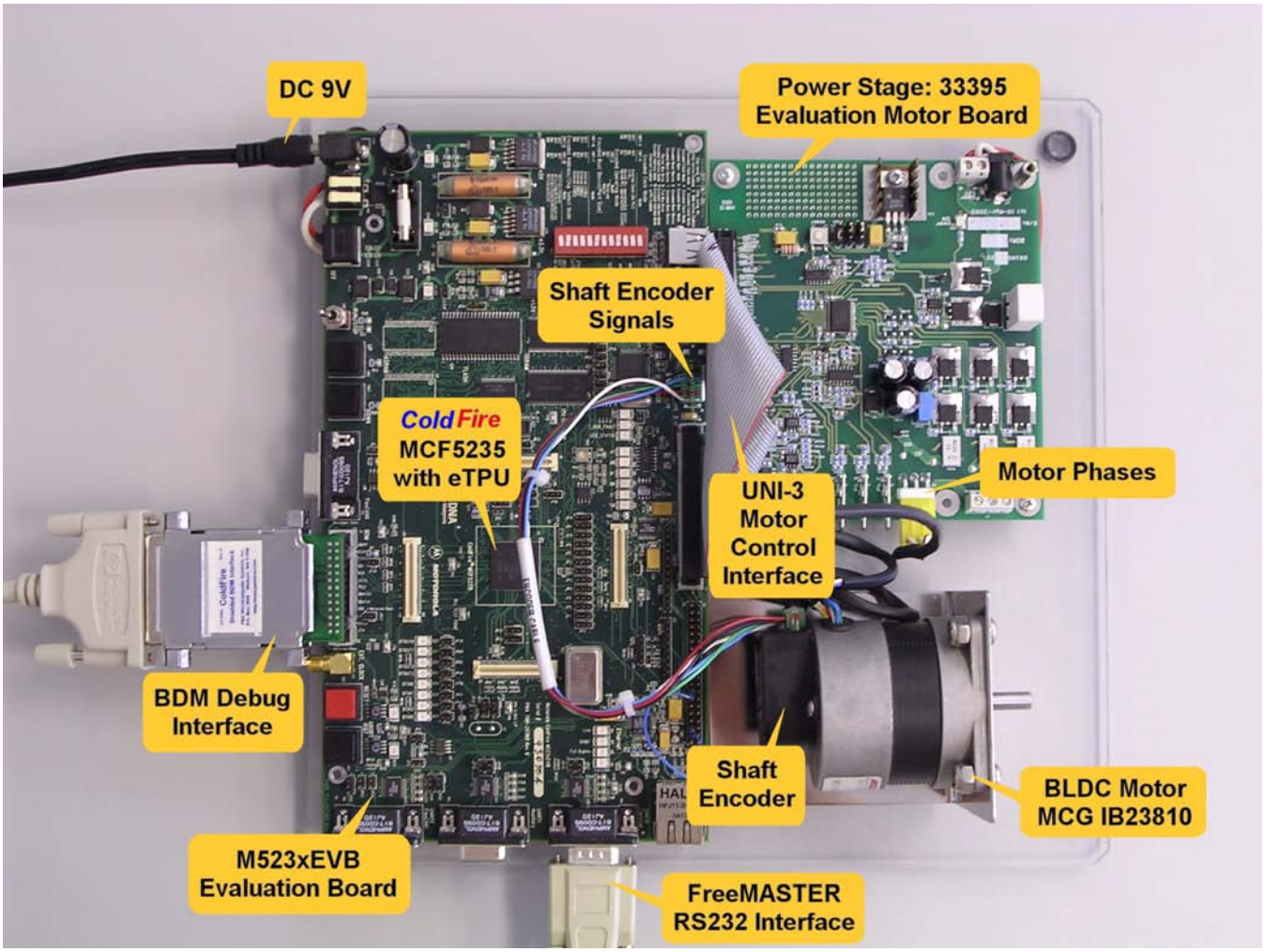

Figure 7. Connection of Application Parts

As previously stated, the application runs on the MCF523x family of ColdFire microprocessors using the following:

- M523xEVB
- 33395 evaluation motor board
- 3-phase MCG PMSM (IB23810)

- Quadrature encoder (HEDS-5640 A06)
- Power supply, 9-12V DC, minimum 2.7Amps

Figure 7 shows the connection of these parts. All system parts are documented according to references.

3.3.1 ColdFire MCF523x Evaluation Board (M523xEVB)

The EVB is intended to provide a mechanism for customers to easily evaluate the MCF523x family of ColdFire microprocessors. The heart of the evaluation board is the MCF5235; all other M523x family members have a subset of the MCF5235 features and can therefore be fully emulated using the MCF5235 device.

The M523xEVB is fitted with a single 512K x 16 page-mode Flash memory (U19), giving a total memory space of 2 Mbytes. Alternatively, a footprint is available for upgrading flash to a 512K x 32 page-mode Flash memory (U35), doubling the memory size to 4 Mbytes.

For more information, refer to Reference 2.

Table 1 lists all M523xEVB jumper settings used in the application.

Table 1. M523xEVB Jumper Settings

Jumper	Setting	Jumper	Setting	Jumper	Setting	Jumper	Setting
JP1	1 2	JP20	1 2-3	JP40	1-2	JP60	1-2
JP2	1-2	JP21	1 2-3	JP41	1-2	JP61	1-2
JP3	1 2	JP22	1 2-3	JP42	1-2	JP62	1 2
JP4	1-2	JP23	1 2-3	JP43	1-2	JP63	1 2
JP5	1 2-3	JP24	1 2-3	JP44	1-2	JP64	1 2-3
JP6	1-2 3	JP25	1-2 3	JP45	1-2		
JP7	1 2-3	JP26	1-2 3	JP46	1-2	DIP1	ON
JP8	1-2 3	JP27	1-2	JP47	1-2	DIP2	ON
JP9	1 2-3	JP28	1-2	JP48	1-2	DIP3	ON
		JP29	1-2	JP49	1-2	DIP4	ON
						DIP5	ON
JP10	1 2-3	JP30	1-2	JP50	1-2 3	DIP6	ON
JP11	1 2-3	JP31	1 2-3	JP51	1-2 3	DIP7	OFF
JP12	1 2-3	JP32	1-2 3	JP52	1-2 3	DIP8	ON
JP13	1 2-3	JP33	1-2	JP53	1 2	DIP9	ON
JP14	1 2-3	JP34	1-2	JP54	1 2	DIP10	ON
JP15	1 2-3	JP35	1-2 3	JP55	1 2	DIP11	OFF
JP16	1 2-3	JP36	1-2 3	JP56	1-2 3	DIP12	ON
JP17	1 2-3	JP37	1-2	JP57	1-2		
JP18	1 2-3	JP38	1-2	JP58	1-2		
JP19	1 2-3	JP39	1-2	JP59	1-2		

3.3.2 Flashing the M523xEVB

The CFFlasher utility can be used for programming code into the Flash memory on the MCF523xEVB. Check for correct setting of switches and jumpers: SW7-6 on, SW7-7 off, JP64 2-3, (JP31 2-3). The flashing procedure is as follows:

1. Run Metrowerks CodeWarrior for ColdFire and open the project. Choose the simple_eflash target and compile the application. A file simple_eflash.elf.S19, which will be loaded into Flash memory, is created in the project directory bin.
2. Run the CFFlasher application, click on the "Target Config" button. In the Target Configuration window, select the type of board as M523xEVB and the BDM Communication as PE_LPT (see Figure 8). Click OK to close the window.
3. Go to the Program section by clicking the "Program" button. Select the simple_eflash.elf.S19 file and check the "Verify after Program" option (see Figure 9). Finally, press the "Program" button at the bottom of the window to start loading the code into the Flash memory.
4. If the code has been programmed correctly, remove the BDM interface and push the RESET button on the M523xEVB. The application should now run from the Flash.

The CFFlasher application can be downloaded from http://www.freescale.com/coldfire.

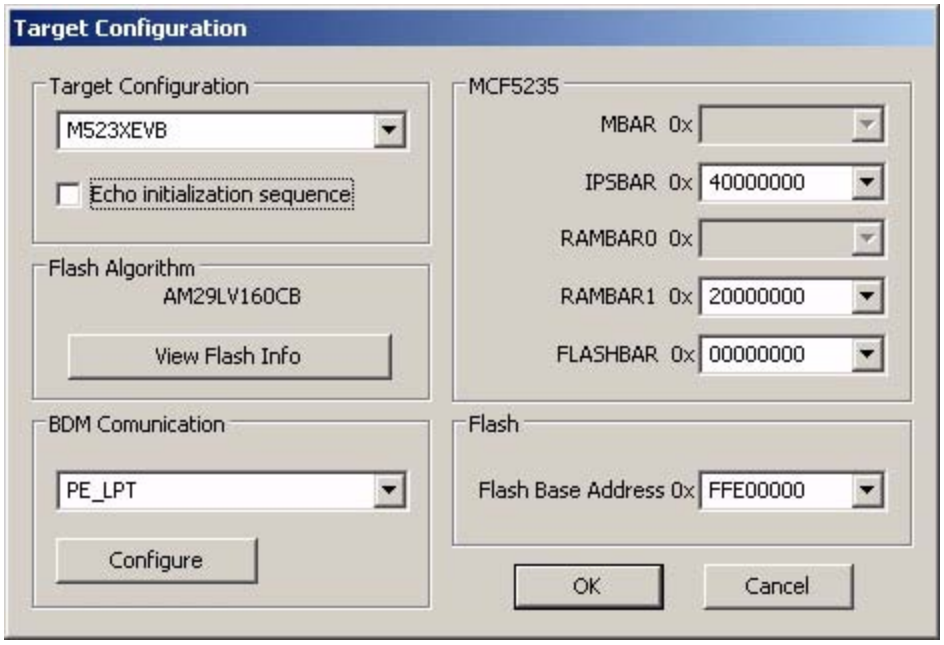

Figure 8. CFFlasher Target Configuration Window

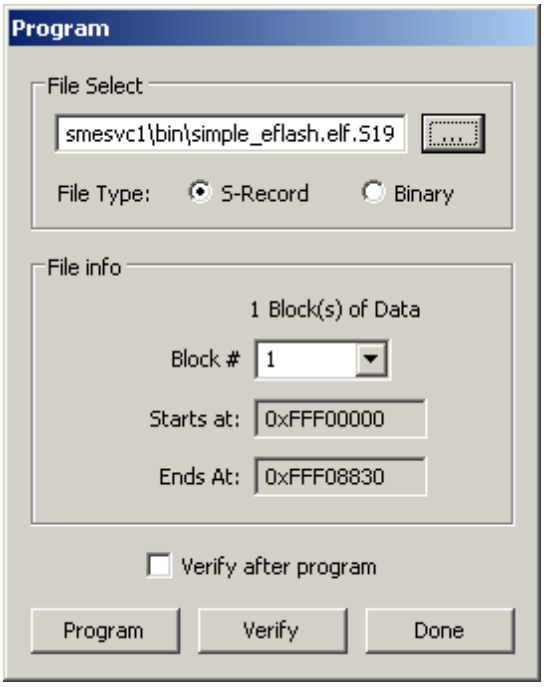

Figure 9. CFFlasher Program Window

3.3.3 Setting Over-current Level

The over-current fault signal is connected to the eTPU output disable pin (LETPUODIS) that handles eTPU hardware faults, along with the proper eTPU configuration. This connection is part of M523xEVB[1]. In order to enable handling of the fault by software, the fault signal, available on the LETPUODIS pin, must be connected to eTPU channel 4, which runs the GPIO function and generates an interrupt request to the CPU in the case of a fault. This connection must be done manually. Connect pin 6 (LETPUODIS) with pin 16 (ETPUCH4) on the eTPU header (see Figure 10).

The over-current level is set by trimmer R41 on M523xEVB (see Figure 11). Reference 3 describes what voltage the trimmer defines for the over-current comparator. Follow the steps below to set the over-current level up properly without measuring the voltage:

1. Connect all system parts according to Figure 7, connect pin 16 with pin 40 on the eTPU header. Now the over-current interrupt is disabled. The over-current fault is handled by hardware only.
2. Download and start the application.
3. Turn the ON/OFF switch ON. Using the Up and Down buttons, set the required speed to the maximum.
4. Adjust the R41 trimmer. You can find a level from which the red LED starts to light and the motor speed starts to be limited. Set the trimmer level somewhat higher, so that the motor can run at the maximum speed.
5. Turn the ON/OFF switch OFF.

1. When the eTPU is configured for 32-channels, LTPUODIS is applicable to channels 0-15. When the ethernet is enabled (SW11 on), the function of LTPUODIS then changes to channels 0-7 and UTPUODIS thus controls channels 8-15.
Therefore the UTPUODIS must be tied to LTPUODIS to enable the application to work when ethernet is enabled.

6. Connect pin 16 with pin 6 on the eTPU header. This enables the over-current interrupt. Both hardware and software handle the over-current fault.
7. Turn the ON/OFF switch ON. Using the Up and Down buttons, set the required speed to the maximum.
   If the application goes to the fault state during the acceleration, adjust the R41 trimmer level somewhat higher, so that the motor can get to the maximum speed.

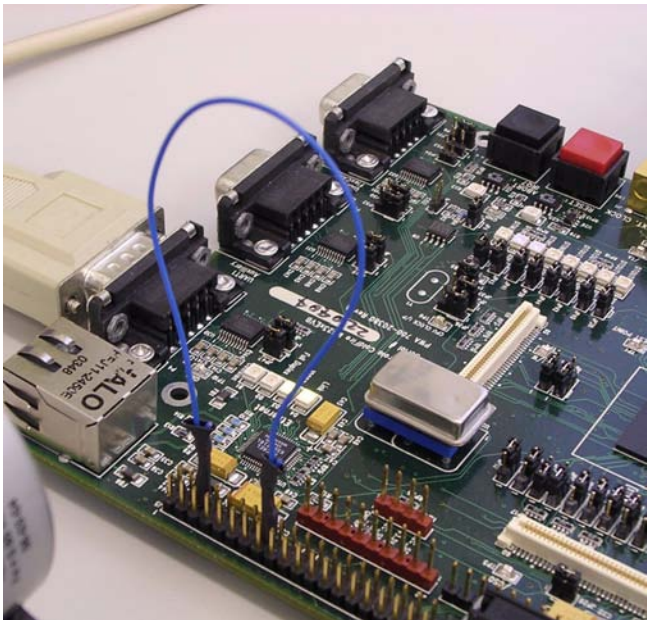

Figure 10. Connection Between LETPUODIS and ETPUCH4 on the eTPU Header

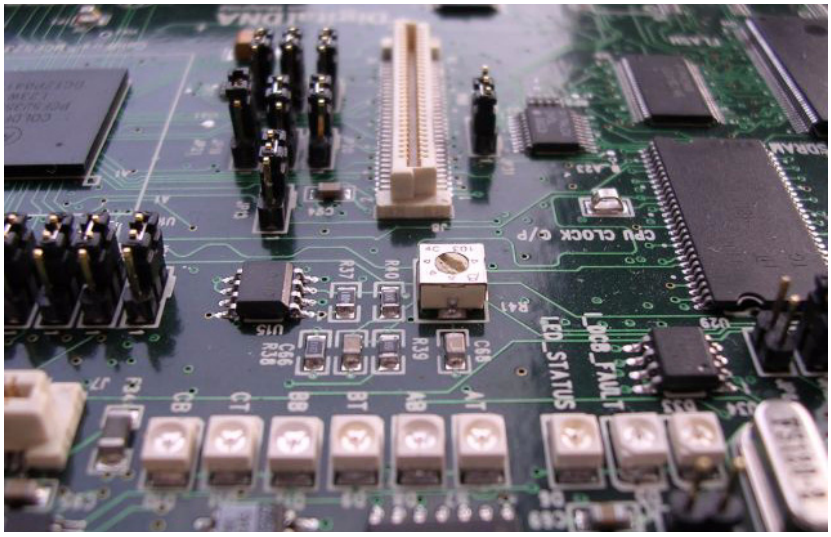

Figure 11. Over-current Level Trimmer on the M523xEVB (R41)

3.3.4 33395 Evaluation Motor Board

The 33395 evaluation motor board is a 12-volt, 8-amp power stage, which is supplied with a 40-pin ribbon cable. In combination with the M523xEVB, it provides an out-of-the-box software development platform

for small brushless DC motors. The power stage enables sensing a variety of feedback signals suitable for different motor control techniques. It measures all three-phase currents, the DC-bus voltage, and the back-EMF voltages with zero cross sensing. The DC-bus current is also reconstructed from the three-phase currents. All the analog signals are adapted to be directly sampled by the A/D converter. This single-board power stage contains an analog bridge gate driver circuitry, sensing and control circuitry, power N-MOSFET transistors, DC-bus break chopper, as well as various interface connectors for the supply and the motor.

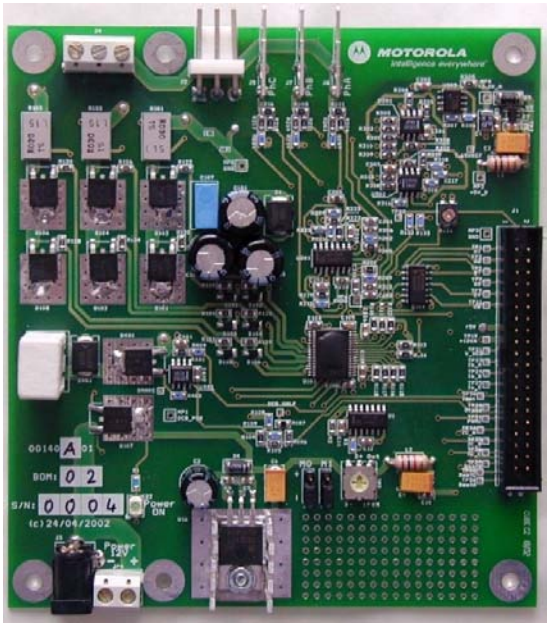

Figure 12. 33395 Evaluation Motor Board

For more information, refer to Reference 3.

3.3.5 PMSM with Quadrature Encoder

The used motor is a low-voltage MCG PMSM (IB23810). The motor characteristics in Table 2 apply to operation at 25°C.

Table 2. MCG PMSM (IB23810) Characteristics

Characteristic	Tolerance	Units	Value
Max. operating speed	MAX.	R.P.M.	5000
Continuous torque	MAX.	LC	20
Peak torque	MAX.	LC	60
Continuous current	MAX.	A	2.0
Peak current	MAX.	A	5.9
Torque sensitivity	±10%	LC/A	11.4

Table 2. MCG PMSM (IB23810) Characteristics (continued)

Characteristic	Tolerance	Units	Value
Back EMF constant	±10%	V/K R.P.M.	8.4
D.C. resistance	±10%	Ω	3.35
Inductance	±15%	mH	6.32
Rotor inertia	NOM.	LC^2	0.0011
Weight	NOM.	lbs	1.18

For more motor specifications, refer to Reference 4.

Quadrature encoder HEDS-5640 A06 is attached to the motor in order to scan and encode shaft movement. The basic encoder features are as follows:

- Three channel quadrature output with index pulse
- Resolution 500 counts per revolution
- External mounting ears
- Quick and easy assembly
- No signal adjustment required
- Small size
- -40°C to 100°C operating temperature
- TTL compatible
- Single 5V supply

For more quadrature encoder specifications, refer to Reference 5.

3.3.6 Power Supply

The power supply supplied with the M523xEVB, 9.0V/2.7A, is also used to power the 33395 evaluation motor board. The application is scaled for this 9V power supply. In case a 12V power supply is used instead, the application should be rescaled for the wider voltage and speed range.

4 Software Design

This section describes the software design of the PMSM vector control drive application. The system processing is distributed between the CPU and the eTPU, which run in parallel. The CPU and eTPU tasks are described in terms of the following:

- CPU
  - — Software flowchart
  - — Application state diagram
  - — eTPU application API

- eTPU
  - — eTPU block diagram
  - — eTPU timing

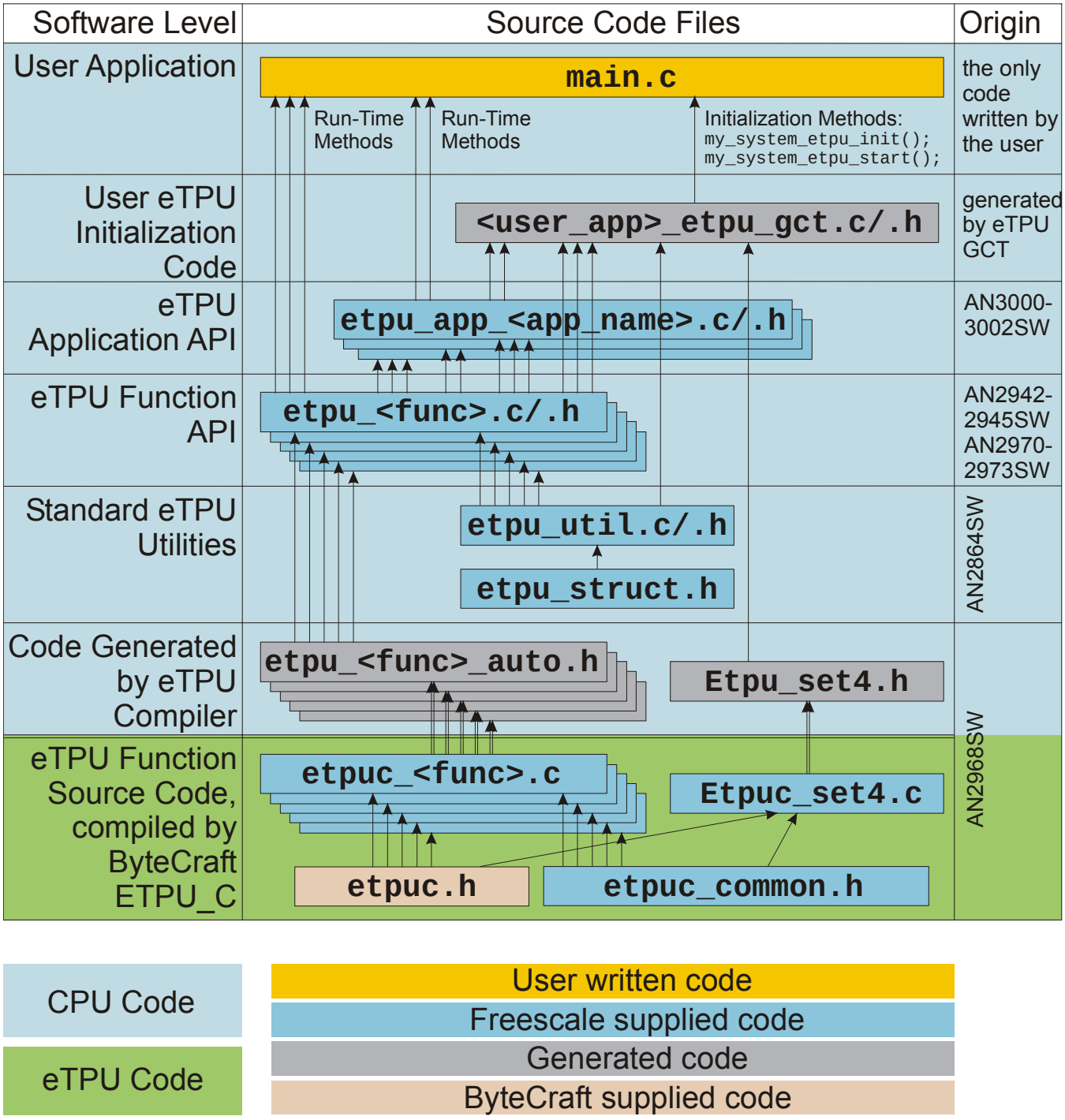

Figure 13. eTPU Project Structure

The CPU software uses several ready-to-use Freescale software drivers. The communication between the microprocessor and the FreeMASTER on PC is handled by software included in `fmaster.c/.h` files. The eTPU module uses the general eTPU utilities, eTPU function interface routines (eTPU function API), and eTPU application interface routines (eTPU application API). The general utilities, included in the etpu_util.c/.h files, are used for initialization of global eTPU module and engine settings. The eTPU function API routines are used for initialization of the eTPU channels and interfacing each eTPU function during run-time. An eTPU application API encapsulates several eTPU function APIs. The use of an eTPU application API eliminates the need to initialize each eTPU function separately and handle all eTPU function initialization settings, ensuring correct cooperation of eTPU functions.

4.1 CPU Software Flowchart

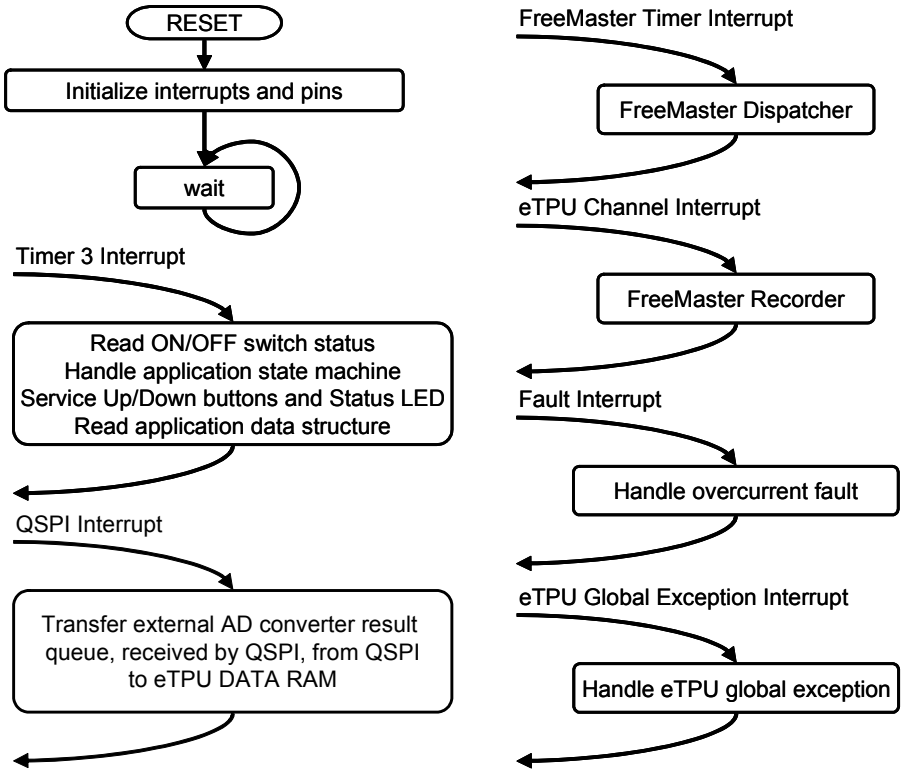

Figure 14. CPU Software Flowchart

After reset, the CPU software initializes interrupts and pins. The following CPU processing is incorporated in two periodical timer interrupts, one periodical eTPU channel interrupt, one periodical QSPI interrupt, and two fault interrupts.

4.1.1 Initialization of Interrupts and Pins

The initialization of timer 3, eTPU channel 4 and 7 interrupts, QSPI interrupt, and the eTPU global exception interrupt, together with initialization of the GPIO and LETPUODIS pins, is done by the InitInterruptsAndPins function.

4.1.2 Timer 3 Interrupt Service Routine

The timer 3 interrupt is handled by the timer3_isr function. The following actions are performed periodical, in timer3_isr:

- Read the ON/OFF switch status.
- Handle the application state machine. The application state diagram is described in detail below.
- Service the Up and Down buttons and the status LED by the ApplicationButtonsAndStatusLed function.
- Read the data structure through the eTPU application API routine. `fs_etpu_app_pmsmesvc1_get_data` (see 4.3).

4.1.3 QSPI Interrupt Service Routine

The QSPI interrupt is handled by the qspi_isr function. A queue of data received by QSPI from external AD converter is copied to eTPU DATA RAM. Then, QSPI is reinitialized to transfer a command to the external AD converter. This transfer is executed by DMA from eTPU.

4.1.4 FreeMASTER Interrupt Service Routine

The FreeMASTER interrupt service routine is called fmasterDispatcher. This function is implemented in fmaster.c.

4.1.5 eTPU Channel Interrupt Service Routine

This interrupt, which is raised every PWM period by the PWMMAC eTPU function running on eTPU channel 7, is handled by the etpu_ch7_isr function. This function calls fmasterRecorder, implemented in fmaster.c, enabling the recording of application variable time courses with a PWM-period time resolution.

4.1.6 Fault Interrupt Service Routine

The over-current fault interrupt, which is raised by the GPIO eTPU function running on eTPU channel 4, is handled by the etpu_ch4_isr function. The following actions are performed in order to switch the motor off:

- Reset the required speed/torque.
- Disable the generation of PWM signals.
- Switch the Fault LED on.
- Enter APP_STATE_MOTOR_FAULT.
- Set FAULT_OVERCURRENT.

4.1.7 eTPU Global Exception Interrupt Service Routine

The global exception interrupt is handled by the etpu_globalexception_isr function. The following situations can cause this interrupt assertion:

- Microcode global exception is asserted.
- Illegal instruction flag is asserted.
- SCM MISC flag is asserted.

The following actions are performed in order to switch the motor off:

- Reset the required speed.
- Disable the generation of PWM signals.
- Enter APP_STATE_GLOBAL_FAULT.
- Based on the eTPU global exception source, set FAULT_MICROCODE_GE, FAULT_ILLEGAL_INSTR, or FAULT_MISC.

4.2 Application State Diagram

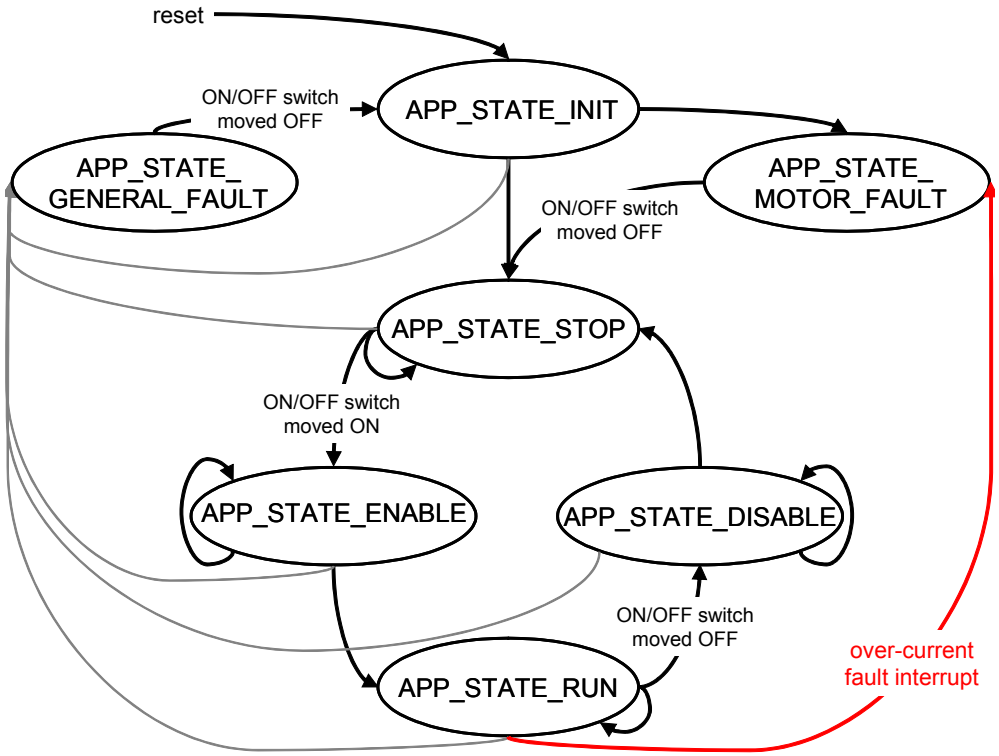

Figure 15. Application State Diagram

The application state diagram consists of seven states (see Figure 15). After reset, the application goes first to APP_STATE_INIT. Where the ON/OFF switch is in the OFF position, the APP_STATE_STOP follows, otherwise the APP_STATE_MOTOR_FAULT is entered and the ON/OFF switch must be turned OFF to get from APP_STATE_MOTOR_FAULT to APP_STATE_STOP. Then the cycle between APP_STATE_STOP, APP_STATE_ENABLE, APP_STATE_RUN, and APP_STATE_DISABLE can be repeated, depending on the ON/OFF switch position. APP_STATE_ENABLE and APP_STATE_DISABLE states are introduced in order to ensure the safe transitions between the APP_STATE_STOP and APP_STATE_RUN states. Where the over-current fault interrupt is raised (see

red line on Figure 15), the APP_STATE_MOTOR_FAULT is entered. This fault is cleared by moving the ON/OFF switch to the OFF position and thus entering the APP_STATE_STOP. Where the eTPU global exception interrupt is raised (see gray line on Figure 15), the APP_STATE_GLOBAL_FAULT is entered. The global fault is cleared by moving the ON/OFF switch to the OFF position and thus entering the APP_STATE_INIT.

The following paragraphs describe the processing in each of the application states.

4.2.1 APP_STATE_INIT

This state is passed through only. It is entered either after a reset, or after the APP_STATE_GLOBAL_FAULT. Perform the following actions to initialize (re-initialize) the application:

- Call `my_system_etpu_init` routine for eTPU module initialization.
- Get eTPU functions DATA RAM addresses for FreeMASTER.
- Get the addresses of channel configuration registers for FreeMASTER.
- Initialize QSPI and ADC.
- Set ASAC polarity to pulse high.
- Initialize DMA3.
- Initialize the UART for FreeMASTER.
- Initialize FreeMASTER.
- Call `my_system_etpu_start` routine for eTPU Start. At this point, the CPU and the eTPU run in parallel.
- Depending on the ON/OFF switch position, enter APP_STATE_STOP or APP_STATE_MOTOR_FAULT.

4.2.1.1 Initialization and Start of eTPU Module

The eTPU module is initialized using the `my_system_etpu_init` function. Later, after initialization of all other peripherals, the eTPU is started by `my_system_etpu_start`. These functions use the general eTPU utilities and eTPU function API routines. Both the `my_system_etpu_init` and `my_system_etpu_start` functions, included in `pmsmesvc1_etpu_gct.c` file, are generated by the eTPU graphical configuration tool. The eTPU graphical configuration tool can be downloaded from http://www.freescale.com/etpu. For more information, refer to Reference 16.

The `my_system_etpu_init` function first configures the eTPU module and motor settings. Some of these settings include the following:

- Channel filter mode = two-sample mode
- Channel filter clock = etpuclk div 2

  The input signals (from quadrature encoder) are filtered by channel filters. The filter settings guarantee minimum delay of input transition recognition.
- TCR1 source = etpuclk div 2

- TCR1 prescaler = 1

  The TCR1 internal eTPU clock is set to its maximum rate of 37.5 MHz (at a 150-MHz system clock), corresponding to the 27ns resolution of generated PWM signals.

After configuring the module and engine settings, the `my_system_etpu_init` function initializes the eTPU channels.

- Channel 1 - quadrature decoder (QD) - phase A channel

  Channel 2 - quadrature decoder (QD) - phase B channel
  Channel 3 - quadrature decoder (QD) - index channel
  Channel 4 - general purpose I/O (GPIO)
  Channel 5 - speed controller (SC)
  Channel 6 - PMSM vector control (PMSMVC)
  Channel 7 - PWM master for AC motors (PWMMAC)
  Channel 8 - PWM full range (PWMF) - phase A - base channel
  Channel 9 - PWM full range (PWMF) - phase A - complementary channel
  Channel 10 - PWM full range (PWMF) - phase B - base channel
  Channel 11 - PWM full range (PWMF) - phase B - complementary channel
  Channel 12 - PWM full range (PWMF) - phase C - base channel
  Channel 13 - PWM full range (PWMF) - phase C - complementary channel
  Channel 14 - analog sensing for AC motors (ASAC)
  Channel 15 - break controller (BC).

  These eTPU channels are initialized by the `fs_etpu_app_pmsmesvc1_init` eTPU application API function (see 4.3). The application settings are as follows:

  — PWM phases-type is full range complementary pairs
  — PWM frequency 20 kHz
  — PWM dead-time 500ns
  — Motor speed range 1 400 RPM
  — Motor speed minimum 5 RPM
  — DC-bus voltage 9V
  — Number of motor pole pairs 2
  — Speed controller update frequency 5 kHz
  — Speed PI controller parameters:
    Controller gain is 1.
    Integral time constant is 1ms.
    The controller parameters were experimentally tuned.
  — Ramp parameters:
    300ms to ramp up from zero to the maximum speed.

  - D-current PI controller parameters:
    Controller gain is 1.
    Integral time constant is 500µs.
    The controller parameters were experimentally tuned.
  - Q-current PI controller parameters:
    Controller gain is 1.
    Integral time constant is 500µs.
    The controller parameters were experimentally tuned.
  - Motor electrical constant 8.4V/kRPM
  - Motor induction 6.32mH
  - Number of quadrature encoder position counter increments per one revolution 2000.
  - Range AD converter sampling phase currents corresponds to 1.947A.
  - Break controller mode - PWM-based breaking signal is generated in case of over-voltage.
  - Break control signal polarity is active high.
  - DC-bus voltage level, at which break control signal is ON, is 130% of the nominal DC-bus voltage.
  - DC-bus voltage level, at which break control signal is OFF, is 110% of the nominal DC-bus voltage.
  - ASAC function triggers A/D converter on high-low edge (applies only during AD converter initialization, then this option is changed).
  - Phase currents and DC-bus voltage measurement time, including A/D conversion time and transfer time, is 15µs.
  - p_ASAC_result_queue pointer contains the address to eTPU DATA RAM, where the result queue is transferred.
  - The samples in result queue are shifted left by 12 bits to achieve bit alignment corresponding to 24-bit fractional format, which is used by eTPU functions.
  - Phase A current sample offset within ASAC_result_queue is 2.
  - Phase B current sample offset within ASAC_result_queue is 4.
  - Phase C current sample offset within ASAC_result_queue is 6.
  - DC-bus voltage sample offset within ASAC_result_queue is 0.
  - ASAC EWMA filter time constant is 200µs for phase currents.
  - ASAC EWMA filter time constant is 1ms for DC-bus voltage.
- Channel 4 - general purpose I/O (GPIO)

  This eTPU channel is initialized by the `fs_etpu_gpio_init` API function. The setting is:
  - Channel priority: high

The `my_system_etpu_start` function first applies the settings for the channel interrupt enable and channel output disable options, then enables the eTPU timers, thus starting the eTPU.

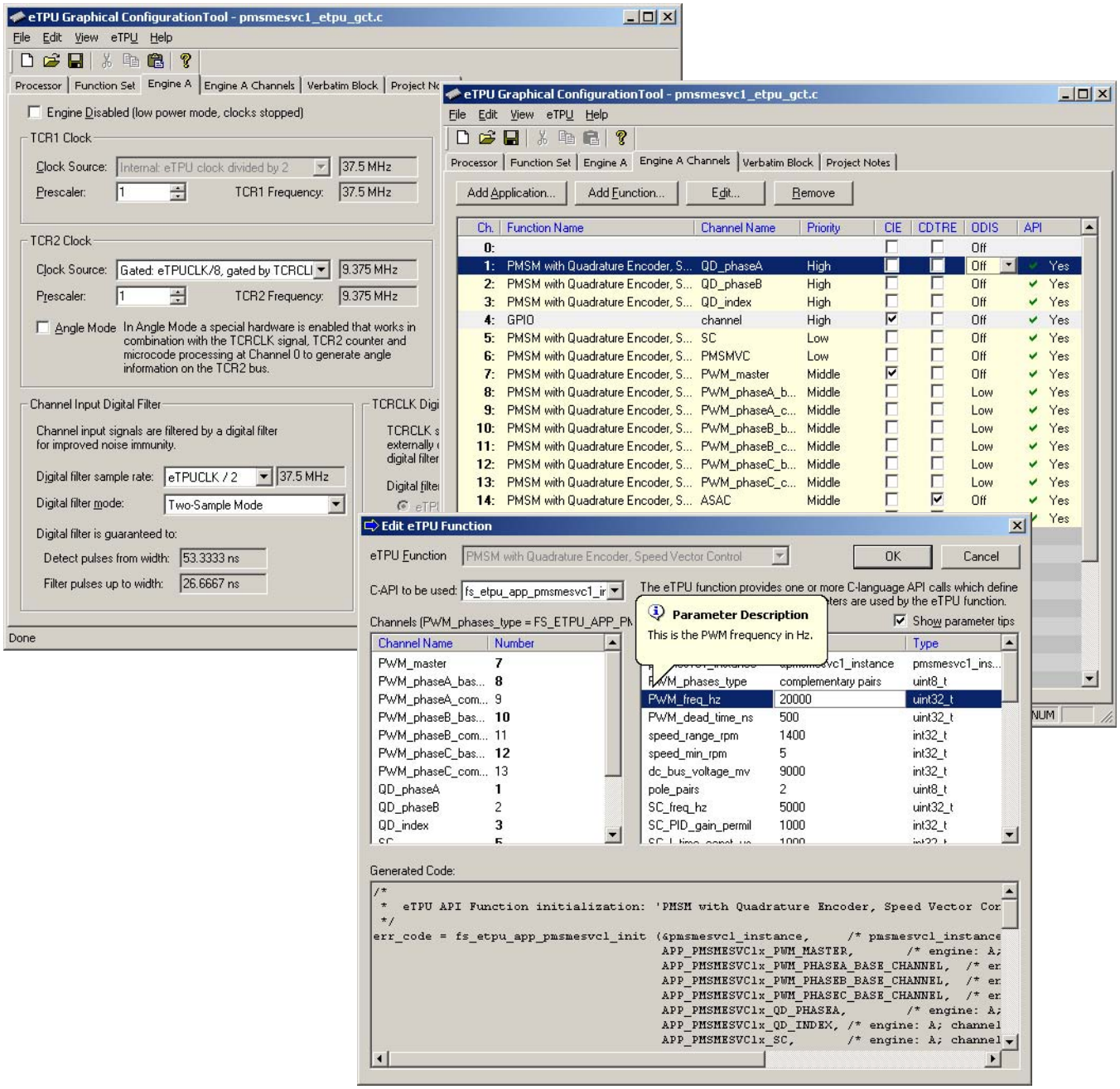

Figure 16. eTPU Configuration Using the eTPU Graphical Configuration Tool

4.2.1.2 Initialization of FreeMASTER Communication

Prior to FreeMASTER initialization, it is necessary to set pointers to the eTPU functions' DATA RAM bases and configuration register bases. Based on these pointers, which are read by FreeMASTER during the initialization, the locations of all eTPU function parameters and configuration registers are defined. This is essential for correct FreeMASTER operation.

FreeMASTER consists of software running on a PC and on the microprocessor, connected via an RS-232 serial port. A small program resident in the microprocessor communicates with the FreeMASTER on the PC in order to return status information to the PC, and processes control information from the PC. The microprocessor part of FreeMASTER is initialized by two functions: iniFmasterUart and fmasterInit. Both functions are included in fmaster.c, which automatically initializes the UART driver and installs all necessary services.

4.2.2 APP_STATE_STOP

In this state, the PWM signals are disabled and the motor is off. The motor shaft can be rotated by hand, which enables the user to explore the functionality of the quadrature decoder (QD) eTPU function, watch variables produced by the QD, and see QD signals in FreeMASTER.

When the ON/OFF switch is turned on, the application goes through APP_STATE_ENABLE to APP_STATE_RUN.

4.2.3 APP_STATE_ENABLE

This state is passed through only. The following actions are performed in order to switch the motor drive on:

- Reset the required speed.
- Enable the generation of PWM signals by calling the `fs_etpu_app_pmsmesvc1_enable` application API routine. This routine also performs the motor alignment.

If the PWM phases are successfully enabled, the GPIO eTPU function is configured as input and interrupt on rising edge, and it enters APP_STATE_RUN mode. If the PWM phases are not successfully enabled, the application state does not change.

4.2.4 APP_STATE_RUN

In this state, the PWM signals are enabled and the motor is on. The required motor speed or torque can be set using the Up and Down buttons on the M523xEVB or by using FreeMASTER. The latest value is periodically written to the eTPU.

When the ON/OFF switch is turned off, the application goes through APP_STATE_DISABLE to APP_STATE_STOP.

4.2.5 APP_STATE_DISABLE

This state is passed through only. The following actions are performed in order to switch the motor drive off:

- Reset the required speed/torque.
- Disable the generation of PWM signals.

If PWM phases were successfully disabled, APP_STATE_STOP is entered. If PWM phases are not successfully disabled, the application state remains the same.

4.2.6 APP_STATE_MOTOR_FAULT

This state is entered after the over-current fault interrupt service routine. The application waits until the ON/OFF switch is turned off. This clears the fault and the application enters the APP_STATE_STOP.

4.2.7 APP_STATE_GLOBAL_FAULT

This state is entered after the eTPU global exception interrupt service routine. The application waits until the ON/OFF switch is turned off. This clears the fault and the application enters the APP_STATE_INIT.

4.3 eTPU Application API

The eTPU application API encapsulates several eTPU function APIs. The eTPU application API includes CPU methods which enable initialization, control, and monitoring of an eTPU application. The use of eTPU application API functions eliminates the need to initialize and set each eTPU function separately, and ensures correct cooperation of the eTPU functions. The eTPU application API is device-independent and handles only the eTPU tasks.

Abbreviated application names shorten the eTPU application names:

- Motor type (DCM = DC motor, BLDCM = brushless DC motor, PMSM = permanent magnet synchronous motor, ACIM = AC induction motor, SRM = switched reluctance motor, SM = stepper motor)
- Sensor type (H = Hall sensors, E = shaft encoder, R = resolver, S = sincos, X = sensorless)
- Control type (OL = open loop, PL = position loop, SL = speed loop, CL = current loop, SVC = speed vector control, TVC = torque vector control)

Based on these definitions, the PMSMESVC1 is an abbreviation for ‘PMSM with quadrature encoder and speed vector control’ eTPU motor-control application. As there can be several applications like this, the number 1 denotes the first such application in order.

The PMSMESVC1 eTPU application API is described in the following paragraphs. There are five basic functions added to the PMSMESVC1 application API. The routines can be found in the `etpu_app_pmsmesvc1.c/.h` files. All PMSMESVC1 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_pmsmesvc1_init(
   pmsmesvc1_instance_t * pmsmesvc1_instance,
                uint8_t   PWM_master_channel,
                uint8_t   PWM_phaseA_channel,
                uint8_t   PWM_phaseB_channel,
                uint8_t   PWM_phaseC_channel,
                uint8_t   QD_phaseA_channel,
                uint8_t   QD_index_channel,
                uint8_t   SC_channel,
                uint8_t   BC_channel,
```

```
                uint8_t   PMSMVC_channel,
                uint8_t   ASAC_channel,
                uint8_t   PWM_phases_type,
                uint32_t  PWM_freq_hz,
                uint32_t  PWM_dead_time_ns,
                int32_t   speed_range_rpm,
                int32_t   speed_min_rpm,
                int32_t   dc_bus_voltage_mv,
                uint8_t   pole_pairs,
                uint32_t  SC_freq_hz,
                int32_t   SC_PID_gain_permil,
                int32_t   SC_I_time_const_us,
                uint32_t  SC_ramp_time_ms,
                int32_t   PMSMVC_D_PID_gain_permil,
                int32_t   PMSMVC_D_I_time_const_us,
                int32_t   PMSMVC_Q_PID_gain_permil,
                int32_t   PMSMVC_Q_I_time_const_us,
                int32_t   PMSM_Ke_mv_per_krpm,
                int32_t   PMSM_L_uH,
                uint32_t  QD_pc_per_rev,
                int32_t   phase_current_range_ma,
                uint8_t   BC_mode,
                uint8_t   BC_polarity,
                uint8_t   BC_u_dc_bus_ON_perc,
                uint8_t   BC_u_dc_bus_OFF_perc,
                uint8_t   ASAC_polarity,
                uint24_t  ASAC_measure_time_us,
                uint32_t *ASAC_result_queue,
                uint8_t   ASAC_bit_shift,
                uint8_t   ASAC_ia_queue_offset,
                uint8_t   ASAC_ib_queue_offset,
                uint8_t   ASAC_ic_queue_offset,
                uint8_t   ASAC_u_dcbus_queue_offset,
                uint32_t  ASAC_filter_time_constant_i_us,
                uint32_t  ASAC_filter_time_constant_u_us);
```

- Change operation functions:

```
int32_t fs_etpu_app_pmsmesvc1_enable(
   pmsmesvc1_instance_t * pmsmesvc1_instance,
                uint8_t   sc_configuration)

int32_t fs_etpu_app_pmsmesvc1_disable(
```

```
    pmsmesvc1_instance_t * pmsmesvc1_instance)

void fs_etpu_app_pmsmesvc1_set_speed_required(
    pmsmesvc1_instance_t * pmsmesvc1_instance,
                 int32_t   speed_required_rpm)
```

- Value return function:

```
void fs_etpu_app_pmsmesvc1_get_data(
    pmsmesvc1_instance_t * pmsmesvc1_instance,
        pmsmesvc1_data_t * pmsmesvc1_data)
```

4.3.1 int32_t fs_etpu_app_pmsmesvc1_init(...)

This routine is used to initialize the eTPU channels for the PMSM with quadrature encoder and torque vector control application. This function has the following parameters:

- pmsmesvc1_instance (pmsmesvc1_instance_t*) - A pointer to pmsmesvc1_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc1_init`. This structure must be declared in the user application. When there are more instances of the application running simultaneously, there must be a separate pmsmesvc1_instance_t structure for each one.
- PWM_master_channel (uint8_t) - The PWM master channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B.
- PWM_phaseA_channel (uint8_t) - The PWM phase A channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B. In the case of complementary signal generation (PWM_phases_type==FS_ETPU_APP_PMSMESVC1_COMPL_PAIRS), the complementary channel is one channel higher.
- PWM_phaseB_channel (uint8_t) - The PWM phase B channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B. In the case of complementary signal generation (PWM_phases_type==FS_ETPU_APP_PMSMESVC1_COMPL_PAIRS), the complementary channel is one channel higher.
- PWM_phaseC_channel (uint8_t) - The PWM phase C channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B. In the case of complementary signal generation (PWM_phases_type==FS_ETPU_APP_PMSMESVC1_COMPL_PAIRS), the complementary channel is one channel higher.
- QD_phaseA_channel (uint8_t) - The quadrature decoder phase A channel number; 0-30 for ETPU_A, and 64-94 for ETPU_B. The quadrature decoder phase A channel is one channel higher.
- QD_index_channel (uint8_t) - The quadrature decoder index channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B.
- SC_channel (uint8_t) - The speed controller channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B.
- BC_channel (uint8_t) - The break controller channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B.

- PMSMVC_channel (uint8_t) - The PMSM vector control function channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B.
- ASAC_channel (uint8_t) - The analog sensing for AC motors (ASAC) channel number; 0-31 for ETPU_A, and 64-95 for ETPU_B.
- PWM_phases_type (uint8_t) - Determines the type of all PWM phases; should be assigned a value of FS_ETPU_APP_PMSMESVC1_SINGLE_CHANNELS or FS_ETPU_APP_PMSMESVC1_COMPL_PAIRS.
- PWM_freq_hz (uint32_t) - The PWM frequency in Hz.
- PWM_dead_time_ns (uint32_t) - The PWM dead-time in ns.
- speed_range_rpm (int32_t) - The maximum motor speed in rpm.
- speed_min_rpm (int32_t) - The minimum (measurable) motor speed in rpm.
- dc_bus_voltage_mv (int32_t) - The DC-bus voltage in mV.
- pole_pairs (uint8_t) - The number of motor pole-pairs.
- SC_freq_hz (uint32_t) - The speed controller update frequency in Hz. The assigned value must be equal to the PWM_freq_hz divided by 1, 2, 3, 4, 5, ...
- SC_PID_gain_permil (int32_t) - The speed PI controller gain in millesimals.
- SC_I_time_constant_us (int32_t) - The speed PI controller integral time constant in µs.
- SC_ramp_time_ms (uint32_t) - Defines the required speed ramp time in ms. A step change of the required speed from 0 to speed_range_rpm is slowed down by the ramp to take the defined time.
- PMSMVC_D_PID_gain_permil (int32_t) - The D-current (flux controlling current) PI controller gain in millesimals.
- PMSMVC_D_I_time_constant_us (int32_t) - The D-current (flux controlling current) PI controller integral time constant in µs.
- PMSMVC_Q_PID_gain_permil (int32_t) - The Q-current (torque controlling current) PI controller gain in millesimals.
- PMSMVC_Q_I_time_constant_us (int32_t) - The Q-current (torque controlling current) PI controller integral time constant in µs.
- PMSM_Ke_mv_per_krpm (int32_t) - The motor electrical constant in mV/1000RPM.
- PMSM_L_uH (int32_t) - The motor induction in µH.
- QD_qd_pc_per_rev (uint32_t) - The number of QD position counter increments per one revolution.
- phase current_range_ma (int32_t) - The maximum measurable phase current in mA.
- BC_mode (uint8_t) - The BC function mode; should be assigned a value of FS_ETPU_APP_PMSMESVC1_BC_MODE_ON_OFF or FS_ETPU_APP_PMSMESVC1_BC_MODE_PWM.
- BC_polarity (uint8_t) - The BC output polarity.; should be assigned a value of FS_ETPU_APP_PMSMESVC1_BC_ON_HIGH or FS_ETPU_APP_PMSMESVC1_BC_ON_LOW.

- BC_u_dc_bus_ON_perc (uint8_t) - The proportion between U_DC_BUS (above which the BC output is ON) and the nominal U_DC_BUS, expressed in percentage (usually about 130%).
- BC_u_dc_bus_OFF_perc (uint8_t) - The proportion between U_DC_BUS (below which the BC output is OFF) and the nominal U_DC_BUS, expressed in percentage (usually about 110%).
- ASAC_polarity (uint8_t) - The polarity to assign to the ASAC function; should be assigned a value of FS_ETPU_APP_PMSMESVC1_ASAC_PULSE_HIGH or FS_ETPU_APP_PMSMESVC1_ASAC_PULSE_LOW.
- ASAC_measure_time_us (uint24_t) - Time from the first (triggering) edge to the second edge, at which the result queue is supposed to be ready in the DATA_RAM (in us). This value depends on the A/D conversion time and DMA transfer time.
- ASAC_result_queue (uint32_t *) - Pointer to the result queue in eTPU DATA RAM. Result queue is an array of 16-bit words that contains the measured values.
- ASAC_bit_shift (uint8_t) - Defines how to align data from the result queue into fract24 (or int24). This parameter should be assigned a value of: FS_ETPU_APP_PMSMESVC1_ASAC_SHIFT_LEFT_BY_8, FS_ETPU_APP_PMSMESVC1_ASAC_SHIFT_LEFT_BY_10, FS_ETPU_APP_PMSMESVC1_ASAC_SHIFT_LEFT_BY_12, or FS_ETPU_APP_PMSMESVC1_ASAC_SHIFT_LEFT_BY_16.
- ASAC_ia_queue_offset (uint8_t) - Position of the phase A current sample in the result queue. Offset is defined in bytes.
- ASAC_ib_queue_offset (uint8_t) - Position of the phase B current sample in the result queue. Offset is defined in bytes.
- ASAC_ic_queue_offset (uint8_t) - Position of the phase C current sample in the result queue. Offset is defined in bytes.
- ASAC_u_dcbus_queue_offset (uint8_t) - Position of the DC-bus voltage sample in the result queue. Offset is defined in bytes.
- ASAC_filter_time_constant_i_us (uint32_t) - The time constant of an exponentially-weighted moving average (EWMA) filter that applies when processing the phase current samples, in us.
- ASAC_filter_time_constant_i_us (uint32_t) - The time constant of an EWMA filter that applies when processing the DC-bus voltage samples, in us.

4.3.2 int32_t fs_etpu_app_pmsmesvc1_enable(...)

This routine is used to enable the generation of PWM signals, align the motor to the start position and reset position counter, initialize measurement of analog values, and start the vector control loop and the speed controller. This function has the following parameters:

- pmsmesvc1_instance (pmsmesvc1_instance_t*) - A pointer to pmsmesvc1_instance_t structure, which is filled by fs_etpu_app_pmsmesvc1_init.

- sc_configuration (uint8_t) - The required configuration of the SC; should be assigned a value of FS_ETPU_APP_PMSMESVC1_SPEED_LOOP_OPENED or FS_ETPU_APP_PMSMESVC1_SPEED_LOOP_CLOSED.
  If the speed loop is opened, the motor is controlled by torque; if closed, the motor is controlled by speed.

4.3.3 int32_t fs_etpu_app_pmsmesvc1_disable (pmsmesvcl1_instance_t * pmsmesvc1_instance)

This routine is used to disable the generation of PWM signals and stop the vector control loop and the speed controller. This function has the following parameter:

- pmsmesvc1_instance (pmsmesvc1_instance_t*) - A pointer to pmsmesvc1_instance_t structure, which is filled by `fs_etpu_app_pmsmesvc1_init`.

4.3.4 void fs_etpu_app_pmsmesvc1_set_speed_required(...)

This routine is used to set the required motor speed. This function has the following parameters:

- pmsmesvc1_instance (pmsmesvc1_instance_t*) - A pointer to pmsmesvc1_instance_t structure, which is filled by fs_etpu_app_pmsmesvc1_init.
- speed_required_rpm (int32_t) - The required motor speed in rpm.
  If the speed loop is opened (sc_configuration has been set to FS_ETPU_APP_PMSMESVC1_SPEED_LOOP_OPENED in `fs_etpu_pmsmesvc1_enable(...)`), the required speed value is passed directly to the speed controller output, which is the required motor torque. In this case, the required motor speed in RPM, as a fraction of the speed range in RPM, corresponds to the required motor torque, as a fraction of the maximum motor torque.

4.3.5 void fs_etpu_app_pmsmesvc1_get_data(...)

This routine is used to get the application state data. This function has the following parameters:

- pmsmesvc1_instance (pmsmesvc1_instance_t*) - A pointer to pmsmesvc1_instance_t structure, which is filled by fs_etpu_app_pmsmesvc1_init.
- pmsmesvc1_data (pmsmesvc1_data_t*) - A pointer to pmsmesvc1_data_t structure of application state data, which is updated.

4.4 eTPU Block Diagram

The eTPU functions used in the PMSM vector control drive are located in the AC motor-control set of eTPU functions (set4 - AC motors). The eTPU functions within the set serve as building blocks for various AC motor-control applications. The following paragraphs describe the functionality of each block.

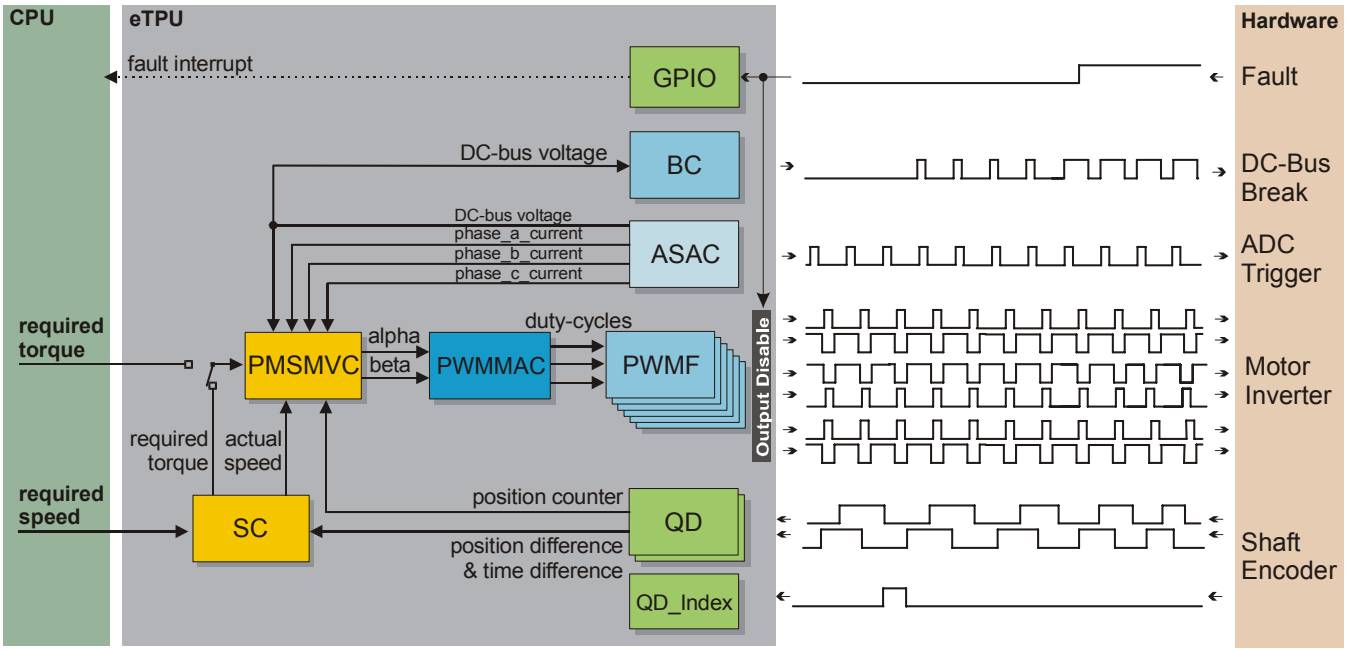

Figure 17. Block Diagram of eTPU Processing

The size of the AC motor control eTPU function set (set4) binary image is higher then 6kB of eTPU CODE RAM available on MCF523x. This is why a limited version of eTPU set4, which fits into 6kB, is used in this application. This version, set4 0.2, includes only the eTPU functions necessary for PMSM vector control application. Some of them are even modified to smaller code size. The following paragraph lists the limitations of all modified eTPU functions:

- Speed controller (SC)—HD mode not implemented
- PWM master for AC motors (PWMMAC)—only two types of modulation implemented (PWM_ICT and SVM_STD)
- PWM full range (PWMF)—half cycle reload not implemented

4.4.1 PWM Generator (PWMMAC+PWMF)

The generation of PWM signals for AC motor-control applications with eTPU is provided by two eTPU functions:

- PWM—master for AC motors (PWMMAC)
- PWM—full range (PWMF)

The PWM master for AC motors (PWMMAC) function calculates sine wave or space vector modulation resulting in PWM duty cycles, and updates the three PWM phases. The phases are driven by the PWM full range (PWMF) function, which enables a full (0% to 100%) duty-cycle range.

The PWMF function generates the PWM signals. The PWMMAC function controls three PWMF functions and three PWM phases, and does not generate any drive signal. The PWMMAC can even be executed on an eTPU channel not connected to an output pin (channels 16 to 32).

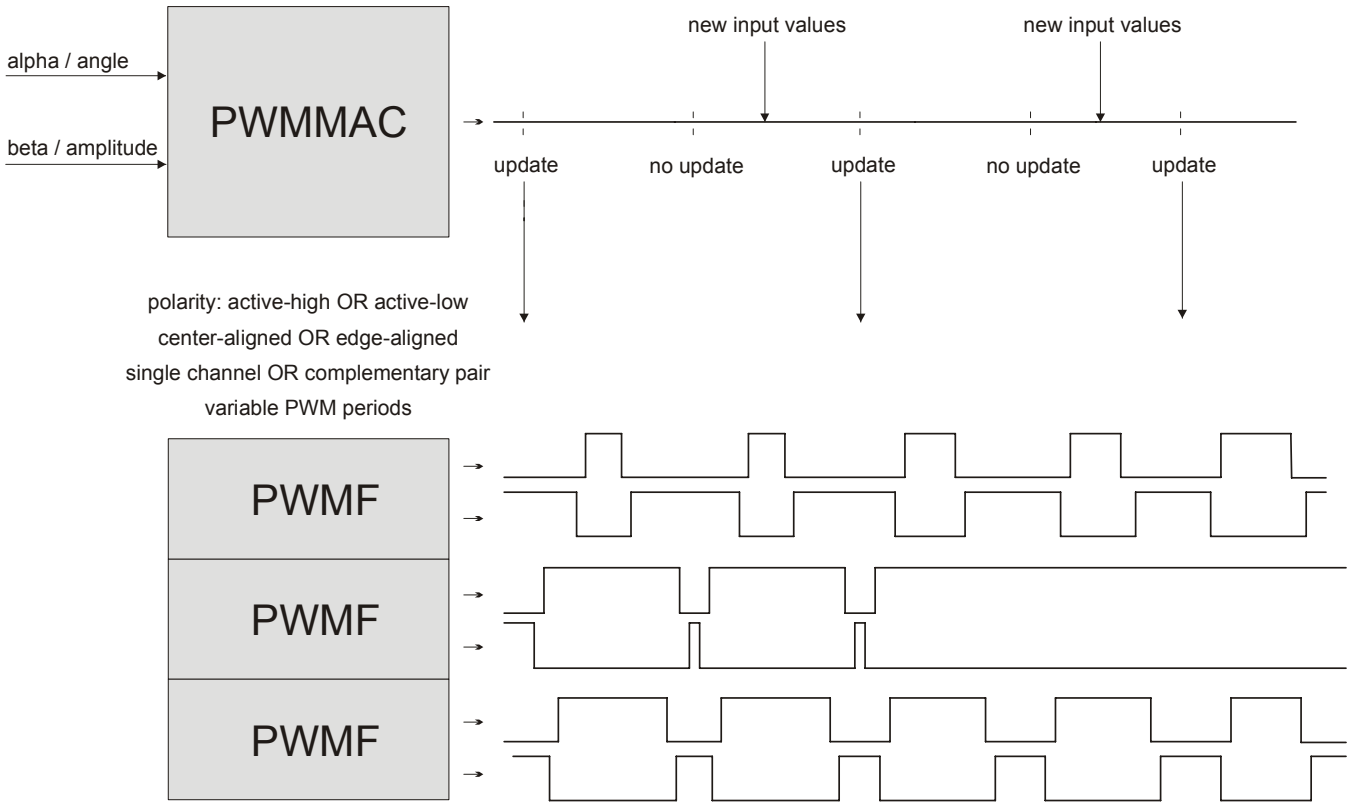

Figure 18. Functionality of PWMMAC+PWMF

For more details about the PWMMAC and PWMF eTPU functions, refer to Reference 10.

4.4.2 Quadrature Decoder (QD)

The quadrature decoder eTPU function set is intended to process signals generated by a shaft encoder in a motion control system. It uses two channels to decode a pair of out-of-phase encoder signals and produce a 24-bit bi-directional position counter, together with direction information, for the CPU. An additional input channel can also be processed. The index channel receives a pulse on each revolution. Based on the

actual direction, a revolution counter is incremented or decremented on the index pulse. A further additional input channel can indicate a home position, but it is not used in this application.

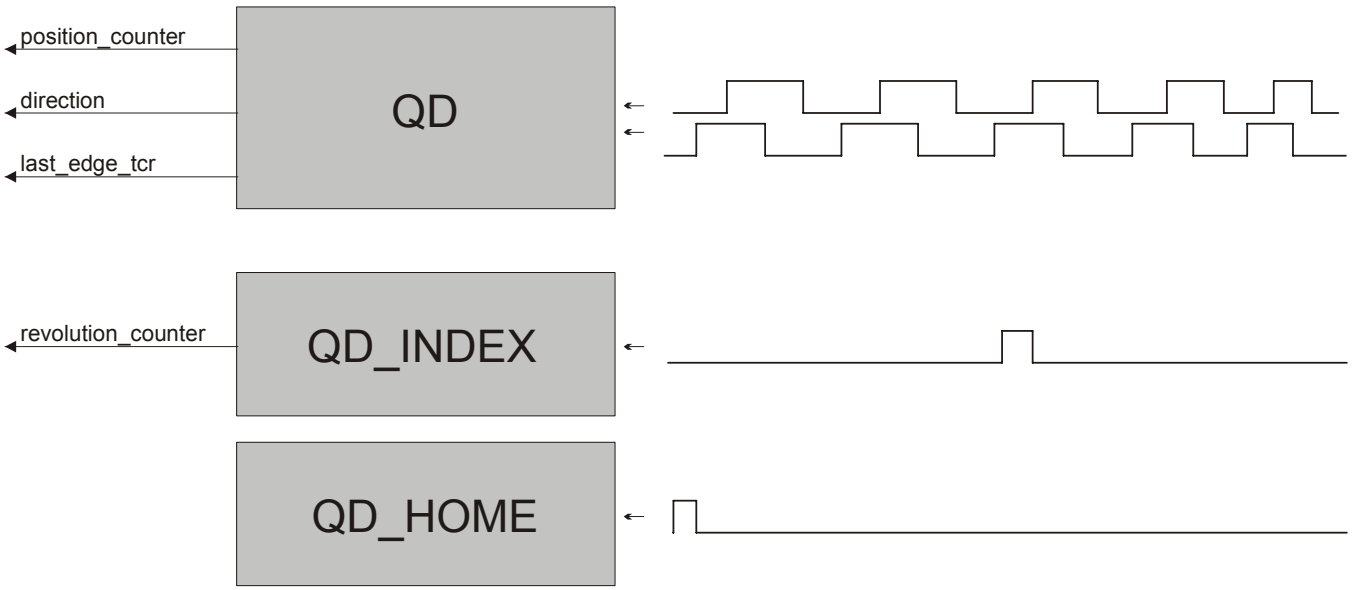

Figure 19. Functionality of QD

For more details about the QD eTPU function, refer to Reference 8.

4.4.3 Analog Sensing for AC Motors (ASAC)

The analog sensing for AC motors eTPU function (ASAC) is useful for preprocessing analog values that are measured by the AD converter and transferred to the eTPU data memory by DMA transfer. The ASAC function is also useful for triggering the AD converter and synchronizing other eTPU functions.

All of the above mentioned ASAC features are utilized in the application. The ASAC is initialized to run in PWM synchronized mode, e.g. the first ASAC edge is synchronized with the beginning of the PWM period. Simultaneously, the ASAC manages to synchronize the SC function by generating the link to the SC channel every fourth ASAC period.

The ASAC function preprocesses the phase currents and DC-bus voltage analog values and passes the adjusted values as an input to the PMSMVC and BC functions. Processing of the DC-bus voltage sample includes bit shifting, dc-offset removing, and filtering. Processing of phase current samples includes also computation of the third phase current from the other two, based on actual motor position in one of six sectors and computation of dead-time compensation parameters.

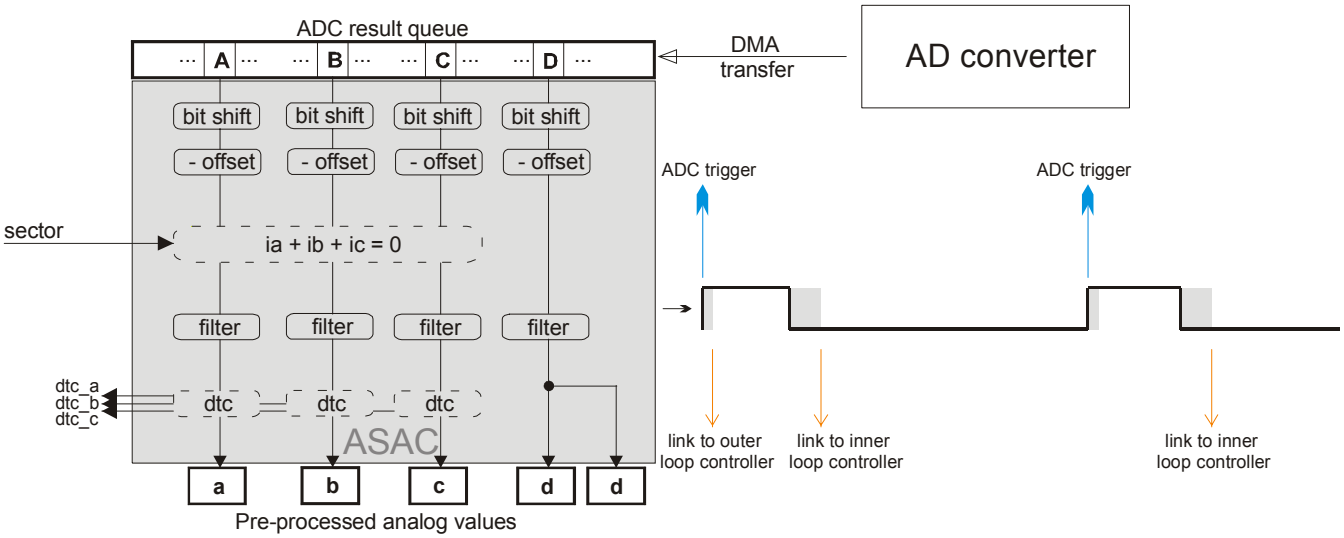

Figure 20. Functionality of ASDC

For more details about the ASAC eTPU function, refer to Reference 12.

In order to ensure periodic sampling and the quick transfer of the measured data from the AD converter to the ETPU DATA RAM, several peripheral modules are used:

- An external analog to digital converter (AD7928BRA) is used for sampling of the analog values.
- A queued serial peripheral interface (QSPI) module is used to interface the AD converter. It enables initialization of the AD converter control registers, triggering the AD converter, and receiving measured samples of phase currents and DC-bus voltage. The QSPI module generates an interrupt to the CPU each time it receives data from the AD converter. Dedicated interrupt service routine qspi_isr ensures that the external AD converter result queue is transferred from QSPI to eTPU DATA RAM.
- Direct memory access (DMA channel 3) is used for the transfer of one auxiliary byte (0x80) from the eTPU DATA RAM to the QDLYR register of the QSPI module. This operation triggers the QSPI transfer and the consequent AD converter sampling. The DMA channel 3 transfer is initiated by the DMA request generated by the ASAC eTPU function.

The DMA transfer decrements the DMA BCR3 register by one. Re-initializing of this register must be done regularly to enable the continuous operation of the DMA channel in this way.

4.4.4 PMSM Vector Control (PMSMVC)

The PMSM vector control eTPU function is not intended to process input or output signals, but to control another eTPU function's input parameter. The PMSMVC function can even be executed on an eTPU channel not connected to an output pin.

The purpose of the PMSMVC function is to perform the current control loop of a field-oriented (vector control) drive of a permanent magnet synchronous motor (PMSM). The sequence of PMSMVC calculations consists of the following steps:

- Forward Clarke transformation

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

The PMSMVC calculates applied voltage vector components alpha and beta based on measured phase currents and required values of phase currents in a 2-phase, orthogonal rotating reference frame (D-Q). The PMSMVC function optionally enables to perform the DC-bus ripple elimination based on the actual measured value of DC-bus voltage.

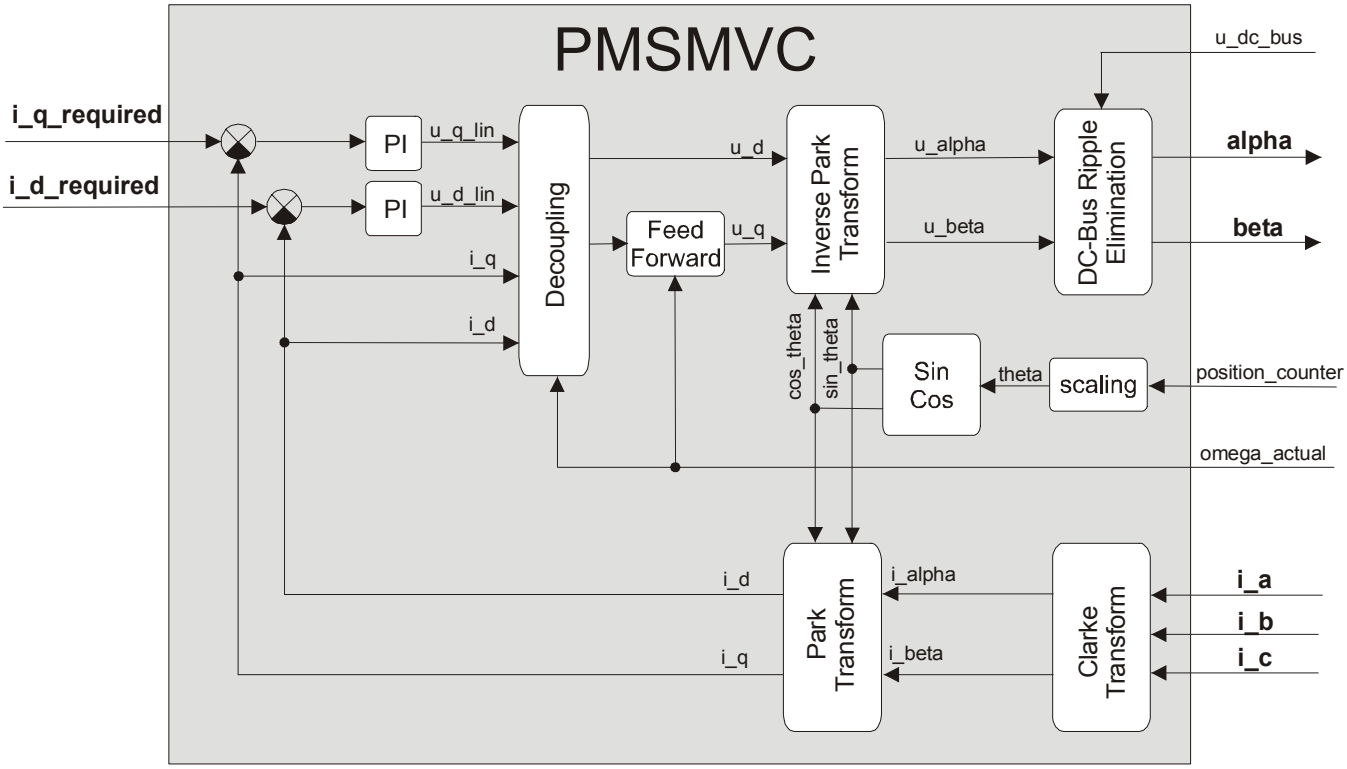

Figure 21. Functionality of PMSMVC

For more details about the PMSMVC eTPU function, refer to Reference 13.

4.4.5 Speed Controller (SC)

The speed controller eTPU function is not intended to process input or output signals. Its purpose is to control another eTPU function's input parameter. The SC function can be executed even on an eTPU channel not connected to an output pin. The SC function includes a general PID controller algorithm. The controller calculates its output based on two inputs: a measured value and a required value. The measured value (the actual motor speed) is calculated based on inputs provided by the QD function. The required value is an output of the speed ramp, whose input is a SC function parameter, and can be provided by the

CPU or another eTPU function. In the motor-control eTPU function set, this function mostly provides the speed outer-loop.

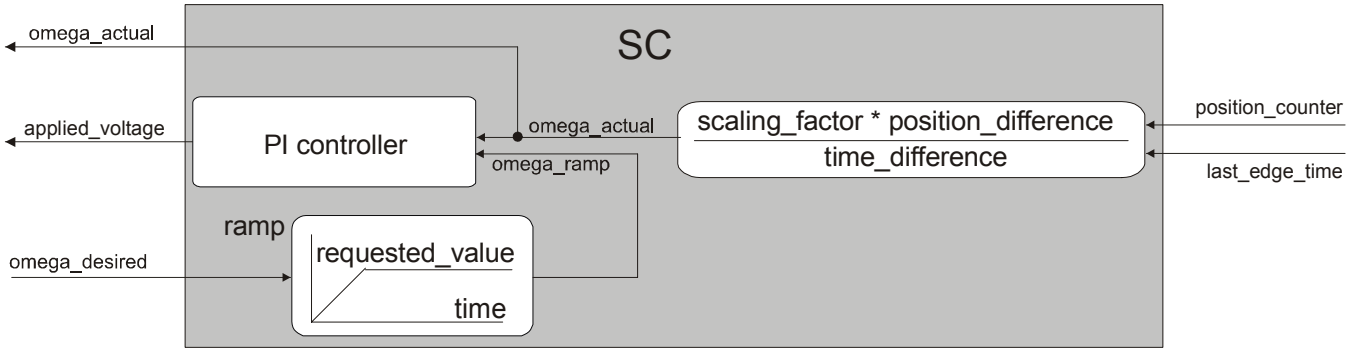

Figure 22. Functionality of SC

For more details about the SC eTPU function, refer to Reference 9.

4.4.6 Break Controller (BC)

The purpose of the break controller (BC) eTPU function is to eliminate DC-bus over-voltage when a motor is driven in the generating mode. The BC function generates the DC-bus break control signal (see Figure 23) based on the actual DC-bus voltage.

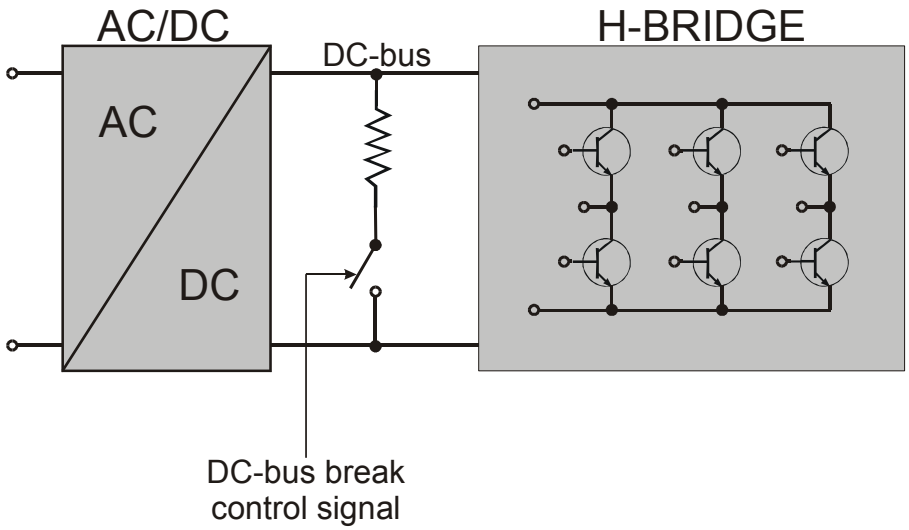

Figure 23. Functionality of BC

The described application uses the PWM mode of the BC function. In this mode, the BC function switches softly using a PWM signal. The *u_dc_bus_ON* and *u_dc_bus_OFF* thresholds define a ramp (see Figure 24). When the DC-bus voltage is lower than *u_dc_bus_OFF*, the control signal is turned off. Between the *u_dc_bus_OFF* and *u_dc_bus_ON* thresholds, a PWM signal with a duty-cycle linearly increasing from 0% to 100% is generated. Above the *u_dc_bus_ON* threshold, the control signal is turned on.

The functionality of the BC is shown in Figure 25, Figure 26, and Figure 27. Signal 1 (dark blue line) represents the DC-bus voltage, and signal 2 (light blue line) reflects the DC-bus break control signal generated by the BC.

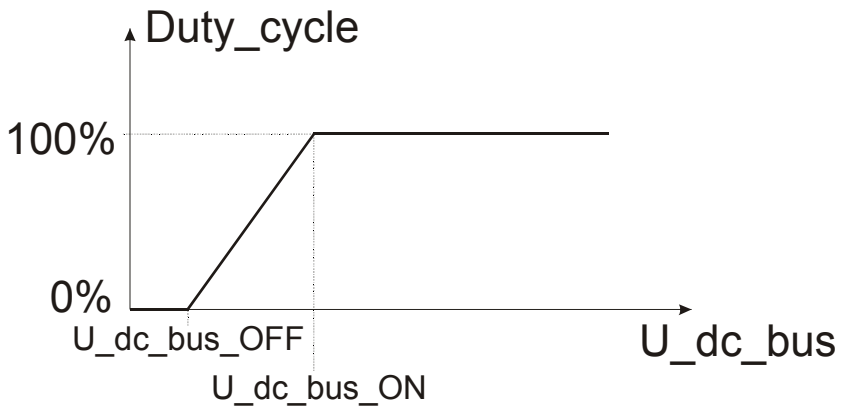

Figure 24. PWM Mode of the Break Controller Function

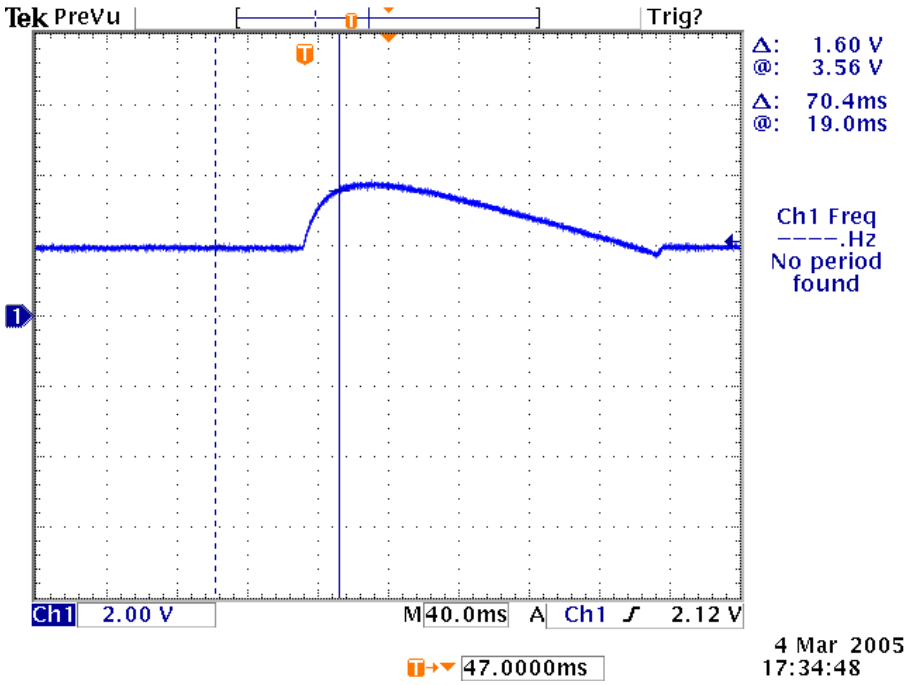

Figure 25. Oscilloscope Screen-shot Showing DC-bus Voltage Course when a Motor Reaches Generating Mode (Without Action of Break Controller)

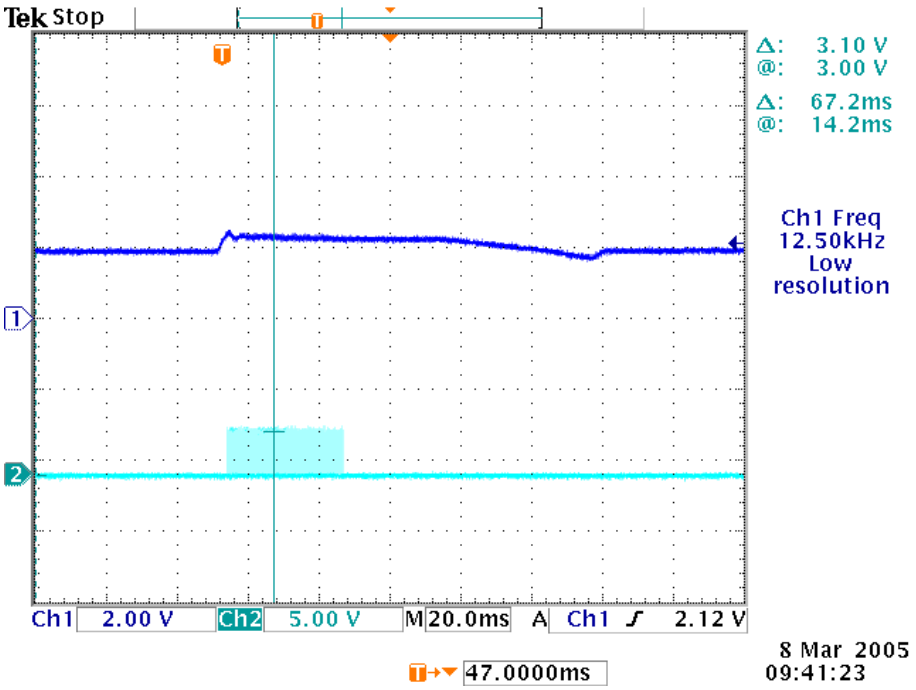

Figure 26. Oscilloscope Screen-shot Showing Functionality of the Break Controller

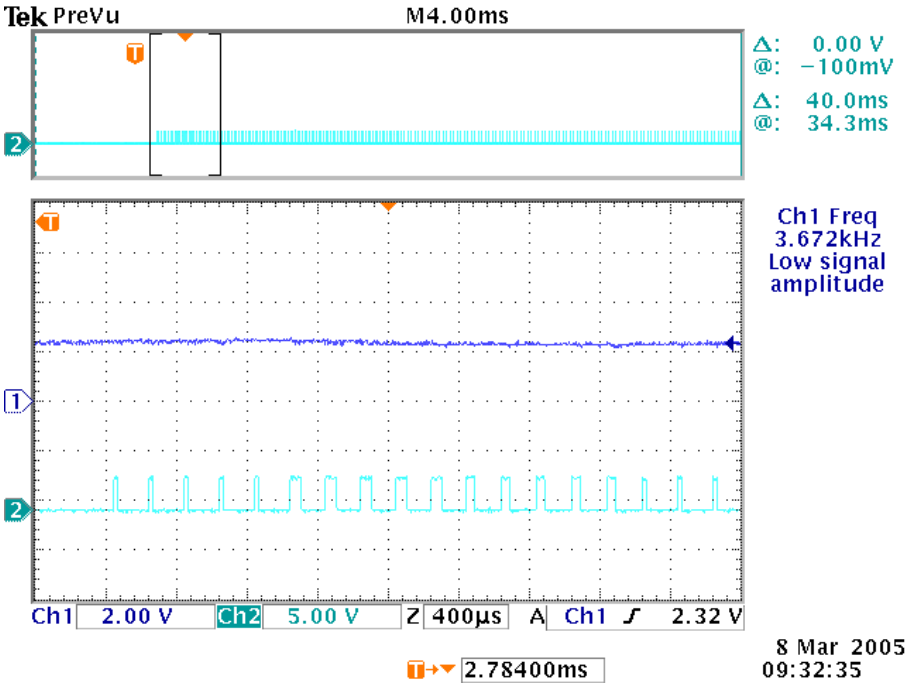

Figure 27. Oscilloscope Screen-shot showing Functionality of the Break Controller in Detail (Zoom of DC-bus Break Control Signal)

For more details about the BC eTPU function, refer to Reference 14.

4.4.7 General Purpose I/O (GPIO)

This function originates from the general eTPU function set (set1) and is included in the DC motor control eTPU function set as well. It allows the user to configure an eTPU channel as a general purpose input or output. There are 7 supported 7 function:

- Input mode—update periodically
- Input mode—update on transition—either edge
- Input mode—update on transition—falling edge
- Input mode—update on transition—rising edge
- Input mode—update on request/disable transition and match updates
- Output mode—output low
- Output mode—output high

GPIO function is configured to generate an interrupt to the CPU in the case of an over-current fault. The fault signal, which is usually connected to the eTPU output disable input, can be also connected to the eTPU channel assigned a GPIO function. When the fault signal turns active, selected output channels are immediately disabled by the eTPU hardware. At the same time, the GPIO function generates an interrupt to notify the CPU about the fault occurrence.

For more details about the GPIO eTPU function, refer to Reference 11.

4.5 eTPU Timing

eTPU processing is event-driven. Once an event service begins, its execution cannot be interrupted by another event service. The other event services have to wait, which causes a service request latency. The maximum service request latency, or worst case latency (WCL), differs for each eTPU channel. The WCL is affected by the channel priority and activity on other channels. The WCL of each channel must be kept below a required limit. For example, the WCL of the PWMF channels must be lower than the PWM period.

A theoretical calculation of WCLs, for a given eTPU configuration, is not a trivial task. The motor control eTPU functions introduce a debugging feature that enables the user to check channel latencies using an oscilloscope, and eliminates the necessity of theoretical WCL calculations.

As mentioned earlier, some eTPU functions are not intended to process any input or output signals for driving the motor. These functions turn the output pin high and low, so that the high-time identifies the period of time in which the function execution is active. An oscilloscope can be used to determine how much the channel activity pulse varies in time, which indicates the channel service latency range. For example, when the oscilloscope time base is synchronized with the PWM periods, the behavior of a tested channel activity pulse can be described by one of the following cases:

- The pulse is asynchronous with the PWM periods. This means that the tested channel activity is not synchronized with the PWM periods.
- The pulse is synchronous with the PWM periods and stable. This means that the tested channel activity is synchronous with the PWM periods and is not delayed by any service latency.

- The pulse is synchronous with the PWM periods, but its position varies in time. This means that the tested channel activity is synchronous with the PWM periods and the service latency varies in this time range.

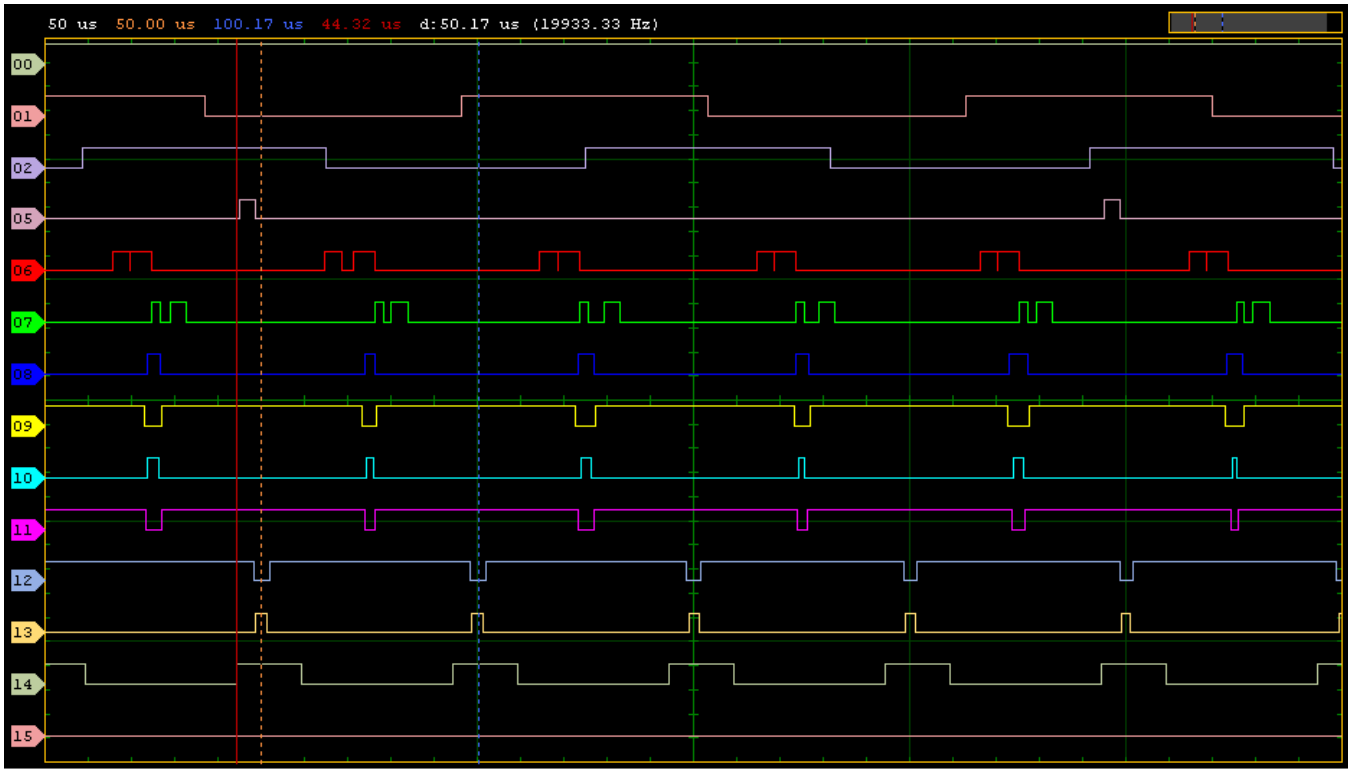

Figure 28. Logic Analyzer Screen-shot of eTPU Timing

Figure 28 explains the application eTPU timing. The logic analyzer screen-shot depicts a typical situation described below:

The Figure 28 shows all input and output eTPU signals. The signals are numbered according to eTPU channels. The time scale is adjusted so that the dark green grid lines correspond to the PWM period frames. The whole picture shows a time section of six PWM periods.

Signals 1 and 2 are coming from the shaft encoder and processed by quadrature decoder (QD) eTPU function. During this time, the motor speed is 1000 RPM.

Signal 5 is generated by the speed controller (SC) eTPU function. Its pulses determine the activity of the SC. The pulse width determines the time necessary to calculate the motor speed from the QD position counter and QD last edge time, calculate the required speed ramp, and apply the PI controller algorithm. This calculation is performed periodically at a 5-kHz rate, which is every fourth PWM period.

Signal 6 is generated by the PMSM vector control (PMSMVC) eTPU function. The pairs of pulses show that the update of the PMSMVC is divided in two consecutive threads. The PMSM update is performed every PWM period.

Signal 7 is generated by the PWM master for AC motors (PWMMAC) eTPU function. Its pulses determine the activity of the PWMMAC. Immediately after PMSMVC update is finished, a narrow PWMMAC pulse occurs. These pulses determine the service time of an PMSMVC request to update the applied motor

voltage vector. This service also includes the calculation of space vector modulation. Apart from these pulses, wider pulses signal the actual update for the next PWM period.

Signals 8 to 13 are three phases of PWM signals—three center-aligned complementary pairs. The base channels (8, 10, 12) are of active-high polarity, while the complementary channels (9, 11, 13) are active-low. The PWM period is 50µs, which corresponds to a PWM frequency of 20 kHz.

Signal 14 is generated by the analog sensing for AC motors (ASAC) eTPU function. The ASAC function triggers the AD converter by generating a DMA request on the first (low-high) edge. The ASAC pulse width determines the time necessary to sample the phase currents and DC-bus voltage and transfer the sampled values to the eTPU data memory. ASAC starts measured samples preprocessing at the time of the second edge when samples are supposed to be ready in the eTPU data memory.

Signal 15 is generated by the break controller (BC) eTPU function. When the break controller channel receives the link, the actual value of DC-bus voltage is compared with the defined over-voltage thresholds, and according to this comparison, generates the PWM-based break control signal.

The ASAC thread executed on the first ASAC edge requests an SC update every fourth period by sending a link to the SC channel. Consequently, the SC requests a BC update. Both SC and BC updates are finished prior to the ASAC second edge. The ASAC thread executed on the second edge requests a PMSMVC update every period, in order to apply the new values of phase currents. The PMSMVC requests PWMMAC to update the applied motor voltage vector. All of these ASAC, SC, BC, and PMSMVC activities have to be finished prior to the start of PWMMAC update. If they are not finished, the PWMMAC update is postponed to the next PWM period. The update starts an update_time prior to the end-of-the-period frame, so that the update is finished by the end-of-the-period frame, even in the worst case latency case. Reference 10 describes how to set the update_time value.

5 Implementation Notes

5.1 Scaling of Quantities

The PMSM vector control algorithm running on eTPU uses a 24-bit fractional representation for all real quantities except time. The 24-bit signed fractional format is mostly represented using 1.23 format (1 sign bit, 23 fractional bits). The most negative number that can be represented is -1.0, whose internal representation is 0x800000. The most positive number is 0x7FFFFF or $1.0 - 2^{-23}$.

The following equation shows the relationship between real and fractional representations:

$$\text{Fractional Value} = \frac{\text{Real Value}}{\text{Real Quatity Range}}$$

where:

Fractional value is a fractional representation of the real value [fract24].
Real value is the real value of the quantity [V, A, RPM, etc.].
Real quantity range is the maximal range of the quantity, defined in the application [V, RPM, etc.].

Some quantities are represented using 3.21 format (3 signed integer bits, 21 fractional bits) in order to eliminate the need of saturation to range (-1, 1). The most negative number that can be represented is -4.0,

whose internal representation is 0x800000. The most positive number is 0x7FFFFF or $4.0 - 2^{-21}$. In this format the components of applied motor voltage vector are passed from PMSMVC to PWMMAC.

5.2 Speed Calculation

The speed controller (SC) eTPU function calculates the angular motor speed using pc_sc and last_edge parameters of the QD eTPU function. The following equation applies:

$$\text{omega\_actual} = \frac{\text{position\_difference}}{\text{time\_difference}} \cdot \text{scaling\_factor}$$

where:

omega_actual [fract24] is the actual angular speed as a fraction of the maximum speed range.

position_difference [int24] is the difference between the updated value of QD position counter and the previous value, which was captured by SC in the previous SC period. In fact the position_difference is readable from pc_sc parameter of the QD function. After SC reads the new updated value, it resets this pc_sc parameters, ensuring that the position_difference is available in the pc_sc parameter next time SC reads it.

time_difference [int24] is the difference between the updated value of QD last_edge and the previous value, which was captured by SC in the previous SC period.

scaling_factor is pre-calculated using the following equation:

$$\text{scaling\_factor} = \frac{30 \cdot 256 \cdot \text{etpu\_tcr\_freq}}{\text{omega\_max} \cdot \text{pc\_per\_rev}}$$

where:

etpu_tcr_freq [Hz] is a frequency of the internal eTPU timer (TCR1 or TCR2) used.

omega_max [RPM] is a maximal speed range.

pc_per_rev is a number of QD position counter increments per one revolution.

The internal eTPU timer (TCR1 or TCR2) frequency must be set so that the calculation of omega_actual both fits into the 24-bits arithmetic and its resolution is sufficient.

6 Microprocessor Usage

Table 3 shows how much memory is needed to run the application.

Table 3. Memory Usage in Bytes

Memory	Available	Used
Flash (external)	2M	34,800
RAM	6K	5,832
eTPU code RAM	6K	6,088
eTPU data RAM	1.5K	1,024

The eTPU module usage in terms of time load can easily be determined based on the following facts:

- According to Reference 10, the maximum eTPU load produced by PWM generation is 1364 eTPU cycles per one PWM period. The PWM frequency is set to 20 kHz, thus the PWM period is 3750 eTPU cycles (eTPU module clock is 75 MHz, half of the 150 MHz CPU clock).
- According to Reference 8, the contribution of QD function to the overall eTPU time load can be calculated. It depends on the number of shaft encoder pulses (500) and the motor speed (maximum 1000rpm). The maximum eTPU busy time per one encoder pulse is 732 eTPU cycles.
- According to Reference 13, the PMSM vector control calculation takes 690 eTPU cycles. The calculation is performed every PWM period.
- According to Reference 9, the speed controller calculation takes 308 eTPU cycles. The calculation is performed every fourth PWM period.
- According to Reference 14, the BC maximum eTPU load per one update in slave PWM switching mode is 64 eTPU cycles, and the BC maximum eTPU load per one PWM edge is 20 eTPU cycles. The BC update is performed every four PWM periods. PWM frequency of the DC-bus break control signal is 10 kHz, which means that the BC-PWM update is performed every two PWM periods.
- According to Reference 12, the ASAC maximum eTPU load takes 42 + 386 eTPU cycles (both the first and then the second edge processing is performed). The ASAC function processing is executed every PWM period.
- The GPIO function processing does not affect the eTPU time load.

The values of eTPU load by each of the functions are influenced by compiler efficiency. The above numbers are given for guidance only and are subject to change. For up-to-date information, refer to the information provided in the latest release available from Freescale.

The peak of the eTPU time load occurs within the PWM period when the speed controller and the break controller calculations are executed. This peak value must be kept below 100%, which ensures that all processing fits into the PWM period, no service latency is longer than the PWM period, and thus the generated PWM signals are not affected.

Table 4 shows the eTPU module time load in several typical situations. For more information, refer to Reference 15.

Table 4. eTPU Time Load

Situation	Average Time Load [%]	Peak Time Load Within PWM Period [%]
Motor Speed 10 RPM (20.8 QD pulses per second)	68.7	84.8
Motor Speed 1000 RPM (2083 QD pulses per second)	76.8	84.8

7 Summary and Conclusions

This application note provides the user with a description of the demo application PMSM vector control. The application also demonstrates usage of the eTPU module on the ColdFire MCF523x, which results in a CPU independent motor drive. The demo application is targeted at the MCF523x family of devices, but it could be easily reused with any device that has an eTPU.

8 References

Table 5. References

1. *MCF5235 Reference Manual*, MCF5235RM
2. *M523xEVB User's Manual*, M5235EVBUM
3. *33395 Evaluation Motor Board Designer Reference Manual* DRM33395/D
4. MCG's Motors web: http://www.mcg-net.com
5. Quadrature Encoder HEDS-5640 A06 distributor's web: http://www.agilent.com/semiconductors
6. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
7. *Enhanced Time Processing Unit Reference Manual*, ETPURM
8. "Using the Quadrature Decoder (QD) eTPU Function," AN2842
9. "Using the Speed Controller (SC) eTPU Function," AN2843
10. "Using the AC Motor Control PWM eTPU Functions," AN2969
11. "Using the General Purpose I/O eTPU functions (GPIO)," AN2850
12. "Using the Analog Sensing for AC Motors (ASAC) eTPU Function," AN2970
13. "Using the PMSM Vector Control (PMSMVC) eTPU Function," AN2972
14. "Using the Break Controller (BC) eTPU Function," AN2845
15. "Using the AC Motor Control eTPU Function Set (set4)," AN2968
16. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT"

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

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

AN3002
Rev. 0
10/2005