**Freescale Semiconductor**
Application Note

AN3005
Rev. 1, 05/2006

# BLDC Motor with Quadrature Encoder and Speed Closed Loop, Driven by eTPU on MPC5554

## Covers MPC5554 and all eTPU-Equipped Devices

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

This application note describes the design of a 3-phase Brushless DC (BLDC) motor drive based on Freescale's PowerPC MPC5554 microcontroller. The application design takes advantage of the Enhanced Time Processing Unit (eTPU) module, which is used as a motor control co-processor. The eTPU completely handles the motor control processing (excluding commutation processing), eliminating the microprocessor overhead for other duties.

BLDC motors are very popular in a wide array of applications. Compared to a DC motor, the BLDC motor uses an electric commutator, replacing the mechanical commutator and making it more reliable than the DC motor. In BLDC motors, rotor magnets generate the rotor's magnetic flux, allowing BLDC motors to achieve higher efficiency. Therefore, BLDC motors may be used in high-end white goods (refrigerators, washing machines, dishwashers, etc.), high-end pumps, fans, and other appliances that require high reliability and efficiency.

The concept of the application is to create a speed-closed loop BLDC driver using a Hall position sensor. It serves as an example of a BLDC motor control system design

**Table of Contents**





freescale semiconductor

using a Freescale microprocessor with the eTPU. It also illustrates the usage of dedicated motor control eTPU functions that are included in the DC 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 MPC5554DEMO, 33395 Evaluation Motor Board, and MCG BLDC Motor**

# 1 PowerPC MPC5554 and eTPU Advantages and Features

## 1.1 PowerPC MPC5554 Microcontroller

The MPC5554 microcontroller is a family of next generation powertrain microcontrollers based on the PowerPC Book E architecture. Featuring two 32 channels eTPU engines, 32 Kbytes of cache, 64 Kbytes of internal SRAM, 2 Mbytes of internal Flash memory, a 64-channel eDMA controller, 3 FlexCAN modules, 3 UARTs and four DSPI modules, the MPC5554 family has been designed for applications that require complex, real-time control.

This 32-bit device is based on the PowerPC operating at a core frequency up to 132 MHz. On-chip modules include:

- High-performance 32-bit PowerPC Book E-compliant core
- Memory management unit (MMU) with 24-entry fully associative translation look-aside buffer (TLB)
- 2MB of embedded Flash memory with Error Correction Coding (ECC)
- 64 KB on-chip L2 static RAM with ECC
- 32 KB of cache that can be configured as additional RAM
- nexus IEEE-ISTO 5001 class multicore debug capabilities
- Two enhanced time processor units (eTPUs)
- 64-channel eDMA (Enhanced Direct Memory Access) controller
- Interrupt controller (INTC) capable of handling 286 satiable-priority interrupt sources
- Frequency modulated phase-locked loop (FMPLL) to assist in electromagnetic interference (EMI) management
- Enhanced queued analog-to-digital converter (eQADC)
- Four deserial serial peripheral interface (DSPI) modules
- Three controller area network (FlexCAN) modules
- Two enhanced serial communication interface (eSCI) modules
- Eighty-eight channels of timed I/O
- Crossbar switch (XBAR)
- Enhanced modular I/O system (eMIOS)

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 on the MPC5554 microcontroller has two engines with up to 32 timer channels for each. In addition it has 16 Kbytes of code memory and 3 Kbytes of data memory that stores software modules downloaded at boot time and that can be mixed and matched as required for any specific 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 8.

# 2 Target Motor Theory

A brushless DC (BLDC) motor is a rotating electric machine where the stator is a classic three-phase stator, like that of an induction motor, and the rotor has surface-mounted permanent magnets (see Figure 2).

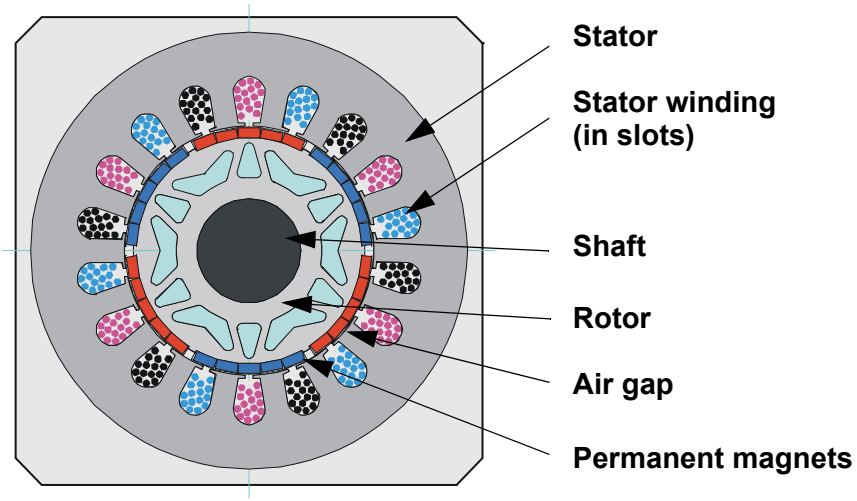



**Figure 2. BLDC Motor—Cross Section**

In this respect, the BLDC motor is equivalent to a reversed DC commutator motor, in which the magnet rotates while the conductors remain stationary. In the DC commutator motor, the current polarity is altered by the commutator and brushes. Unlike the brushless DC motor, the polarity reversal is performed by power transistors switching in synchronization with the rotor position. Therefore, BLDC motors often incorporate either internal or external position sensors to sense the actual rotor position, or the position can be detected without sensors.

## 2.1 Digital Control of a BLDC Motor

The BLDC motor is driven by rectangular voltage strokes coupled with the given rotor position (see Figure 3). The generated stator flux interacts with the rotor flux, which is generated by a rotor magnet and defines the torque and thus the speed of the motor. The voltage strokes must be properly applied to two phases of the three-phase winding system so that the angle between the stator flux and the rotor flux is kept as close to 90° as possible, to get the maximum generated torque. Therefore, the motor requires electronic control for proper operation.

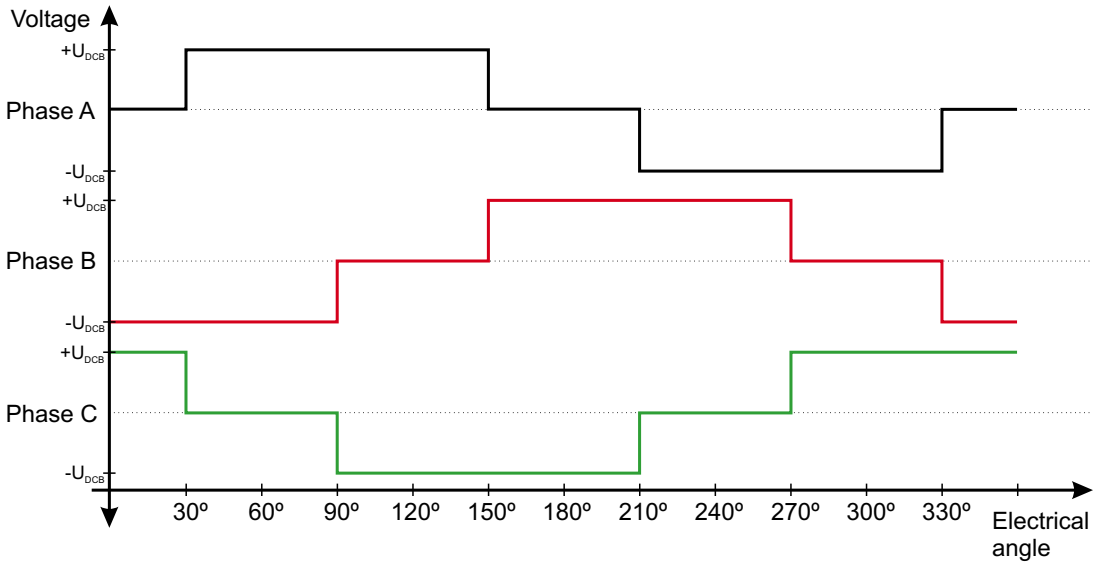



**Figure 3. Voltage Strokes Applied to the 3-Phase BLDC Motor**

For the common 3-phase BLDC motor, a standard 3-phase power stage is used (see Figure 4). The power stage utilizes six power transistors that operate in either an independent or complementary mode.

In both modes, the 3-phase power stage energizes two motor phases concurrently. The third phase is unpowered (see Figure 3). Thus, we get six possible voltage vectors that are applied to the BLDC motor using a Pulse Width Modulation (PWM) technique (see Figure 5). There are two basic types of power transistor switching schemes: independent and complementary. Both switching modes are able to work in bipolar or unipolar mode. The presented application utilizes the complementary bipolar PWM mode.

For more information about PWM techniques, refer to Reference 12.

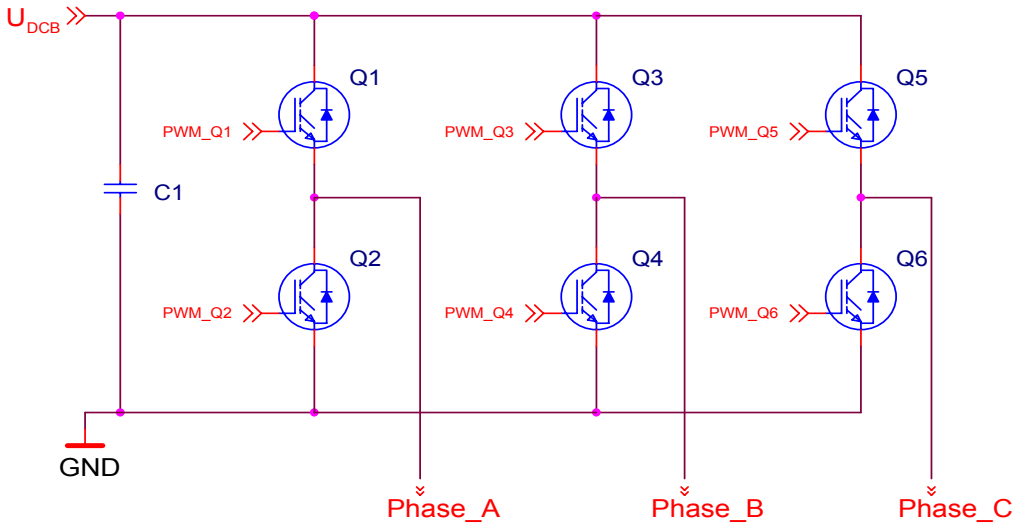



**Figure 4. 3-Phase BLDC Power Stage**

### 2.1.1 Commutation

Commutation provides the creation of a rotational field. As mentioned earlier, for proper operation of a BLDC motor, it is necessary to keep the angle between the stator and rotor flux as close to 90° as possible. We get a total of six possible stator flux vectors with a six-step control. The stator flux vector must be changed at specific rotor positions, which are usually sensed by the Hall sensors. The Hall sensors generate three signals that also consist of six states. Each of the Hall sensors' states correspond to a certain stator flux vector. All of the Hall sensors states, with corresponding stator flux vectors, are illustrated in Figure 5.

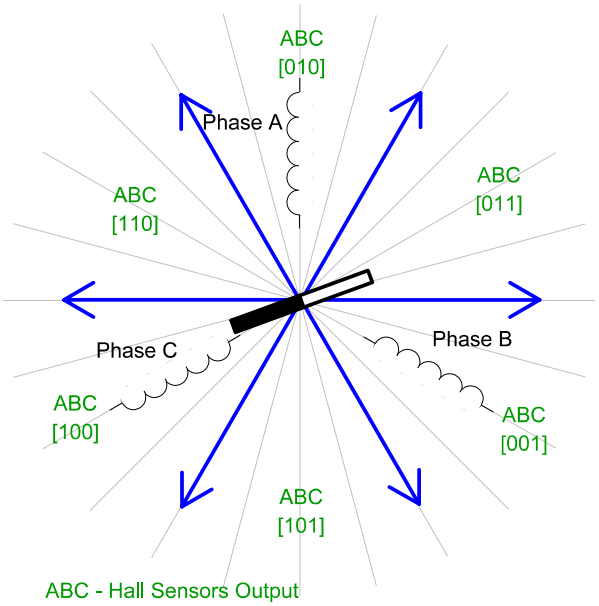



**Figure 5. Stator Flux Vectors at Six-Step Control**

The next two figures depict the commutation process. The actual rotor position in Figure 6 corresponds to the Hall sensors state ABC[110] (see Figure 5). Phase A is connected to the positive DC bus voltage by the transistor Q1; phase C is connected to the ground by transistor Q6, and phase B is unpowered.

As soon as the rotor reaches a certain position (see Figure 7), the Hall sensors state changes its value from ABC[110] to ABC[100]. A new voltage pattern is selected and applied to the BLDC motor.

As shown below, when using the six-step control technique, it is difficult to keep the angle between the rotor flux and the stator flux precisely at 90° in a six-step control technique. The actual angle varies from 60° to 120°.

The commutation process is repeated per each 60 electrical degrees and is critical to maintain its angular (time) accuracy. Any deviation causes torque ripples, resulting in speed variation.

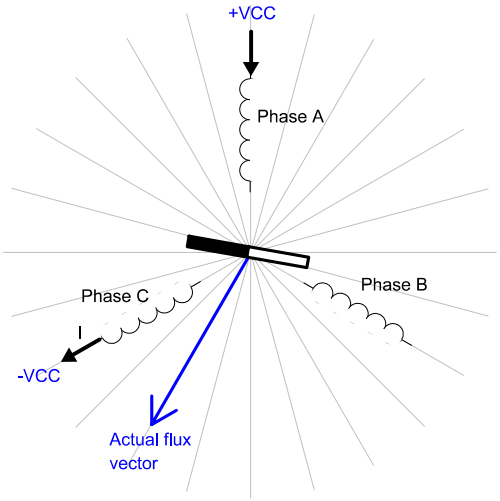



**Figure 6. Situation Right Before Commutation (Counter-Clockwise Motion)**

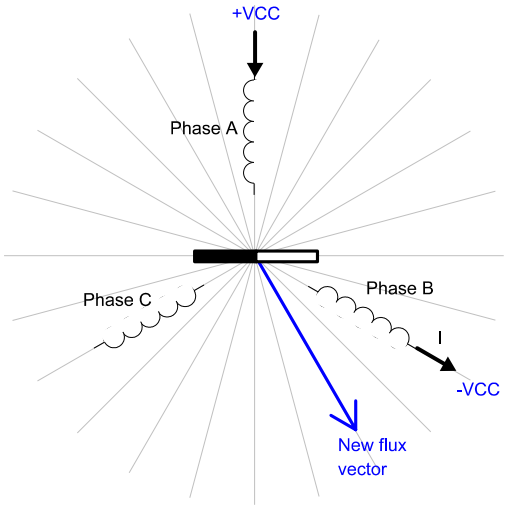



**Figure 7. Situation Right After Commutation**

### 2.1.1.1 Quadrature Encoder Versus Hall Sensors

The BLDC 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 8. Compared with Hall sensors, there are some differences, which affect the control algorithm. The main difference is that the quadrature encoder does not give commutation moment and absolute position, as do the Hall sensors.

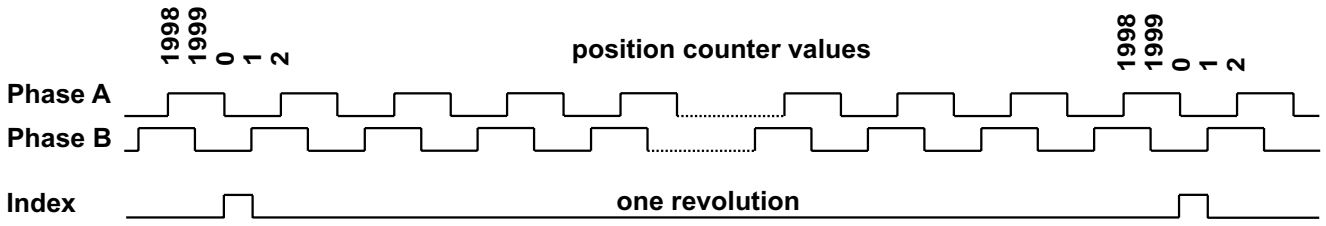



**Figure 8. Quadrature Encoder Output Signals**

The differences between the quadrature encoder and Hall sensors are summarized in Table 1.

**Table 1. Differences Between Quadrature Encoder and Hall Sensors**

| Quadrature Encoder | Hall Sensors |
|---|---|
| 3 outputs | 3 outputs |
| Does not give absolute position | Gives absolute position |
| Gives precise position | Gives 6 events per electrical revolution |

The two control algorithms are described later on with the quadrature encoder. The first solution is commonly used and periodically scans the quadrature encoder. The second solution uses specific advantages of the QD eTPU function and translates the quadrature encoder outputs directly into Hall sensors signals. These internal signals are used as an input for the commutation algorithm. This algorithm is implemented in the presented application.

### 2.1.1.2 Commutation with Periodical Scanning of Quadrature Encoder

The commutation of the BLDC motor is performed in the six defined moments. Since the quadrature encoder gives the precise position, the one electrical revolution is divided into six sectors (see Figure 9). To recognize the commutation moment, it is necessary to scan quadrature encoder position very quickly. The frequency of scanning depends on the maximal rotor speed, the number of pole pairs and on the required precision of commutation moment detection. The same scan frequency as used in the PWM is satisfactory for the common applications.

In this case, the rotor position can be scanned in the moment of a PWM reload interrupt. The algorithm translates the actual position into the one of the six sectors. If a change of sector is detected, the commutation is performed. This algorithm is not implemented in the presented application!

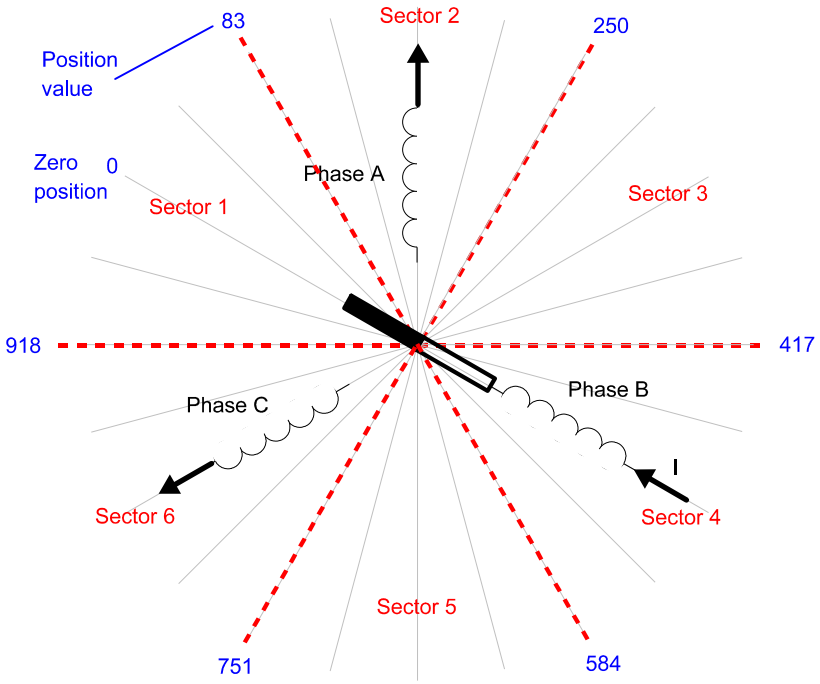



**Figure 9. Separation of One Electrical Revolution into Six Commutation Sectors**

**NOTE:**

The Figure 9 considers 500 pulses per mechanical revolution, both rising and falling edges counting, two pole pairs (500 x 4 / 2 = 1000 pulses per electrical revolution)

### 2.1.1.3 Direct Conversion of Quadrature Encoder Signals to a Commutation Sector

A different method is the direct conversion of encoder output to commutation sectors. The conversion is provided by the eTPU. The advantage of this method is that an interrupt rate depends on the actual motor speed, while in the previous method, there is a constantly high interrupt rate in order to scan the actual motor position. The direct conversion operates in following way:

The electrical revolution is divided into six sectors, as in Figure 9. The rotor position is scanned by the shaft encoder, which has its outputs connected to the dedicated eTPU input pins. The quadrature decoder (QD) eTPU function uses these pins to decode the quadrature encoder signal and to process position counting. After rotor alignment to known position, two compare parameters of the QD eTPU function are set to values that correspond to sector borders and the position counter is set to zero, see Figure 10. When the motor starts to move, the position counter starts to count the pulses of the quadrature encoder. No interrupt is generated until the position counter reaches one of the compare values.

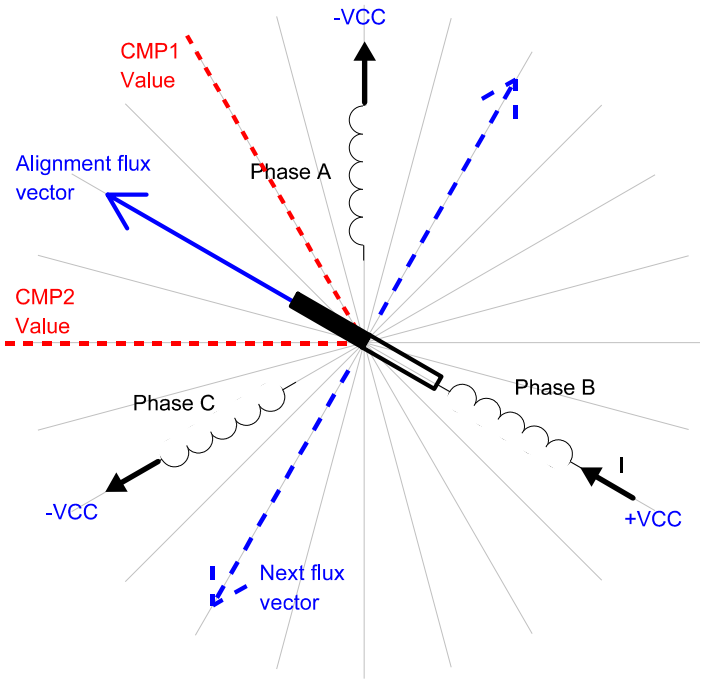



**Figure 10. Alignment Rotor Position**

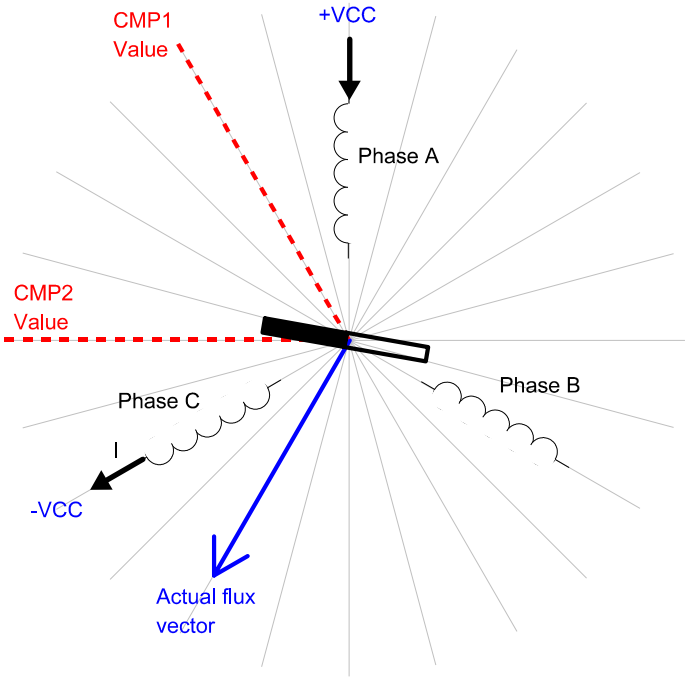



**Figure 11. Situation Prior to First Compare Following Alignment**

As soon as the position counter reaches one of the compare values, the position interrupt is called in order to start the commutation. The position interrupt sets the new values into the compare parameters of the QD eTPU function based on the spin direction. The new values corresponds to the next commutation sector in

clockwise or counter-clockwise direction according to actual spin direction. Then the control word, which saves the actual commutation sector, is updated and the new voltage pattern is applied to the motor. Figure 12 depicts the new situation after commutation.

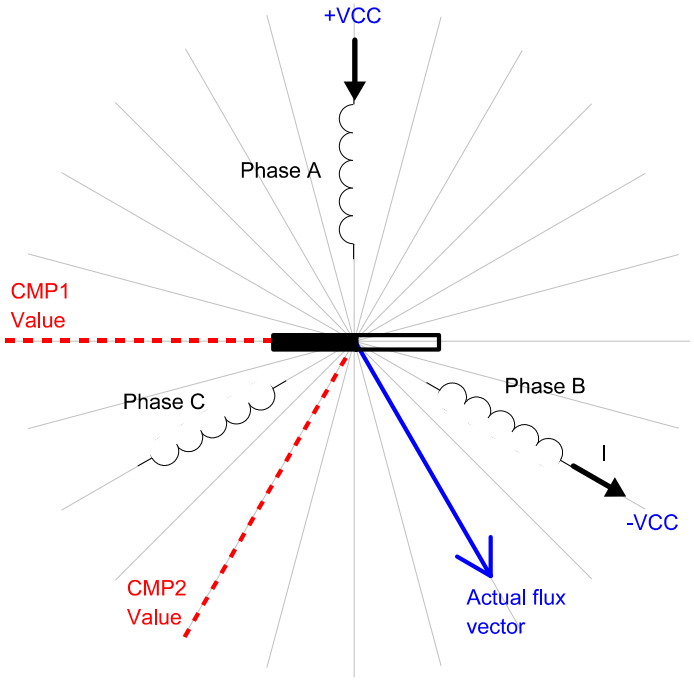



**Figure 12. Situation Following Commutation**

We can see that the position interrupt is called six times per electrical revolution. Note that the position interrupt is called in the same moment as when we use Hall sensors.

## 2.1.2 Position Alignment

Since the quadrature encoder doesn't 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 only once, during the initial motor start up. The Figure 10 shows the position of the aligned rotor. After alignment, the compare parameters of the position counter are set to +/- 30 electric degrees from the alignment position, in order to preset the commutation sector. The next voltage vector is set to be orthogonal to the alignment position. The resultant direction of the voltage vector is given by the polarity of applied voltage.

## 2.1.3 Speed Control

Commutation ensures the proper rotor rotation of the BLDC motor, while the motor speed only depends on the amplitude of the applied voltage. The amplitude of the applied voltage is adjusted using the PWM technique. The required speed is controlled by a speed controller, which is implemented as a conventional

Proportional-Integral (PI) controller. The difference between the actual and required speeds is input to the PI controller which then, based on this difference, controls the duty cycle of the PWM pulses which correspond to the voltage amplitude required to maintain the desired speed.

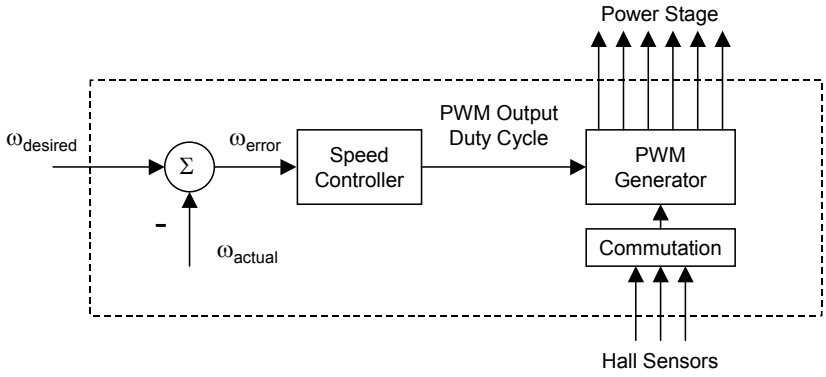



**Figure 13. Speed Controller**

The speed controller calculates the PI algorithm given in the equation below:

$$u(t) = K_c\left[e(t) + \frac{1}{T_I}\int_0^t e(\tau)d\tau\right]$$

After transforming the equation into a discrete time domain using an integral approximation with the Backward Euler method, we get the following equations for the numerical PI controller calculation:

$$u(k) = u_P(k) + u_I(k)$$

$$u_P(k) = K_c \cdot e(k)$$

$$u_I(k) = u_I(k-1) + K_c\frac{T}{T_I} \cdot e(k)$$

where:

| | | |
|---|---|---|
| e(k) | = | Input error in step k |
| w(k) | = | Desired value in step k |
| m(k) | = | Measured value in step k |
| u(k) | = | Controller output in step k |
| $u_p$(k) | = | Proportional output portion in step k |
| $u_I$(k) | = | Integral output portion in step k |
| $u_I$(k-1) | = | Integral output portion in step k-1 |
| $T_I$ | = | Integral time constant |
| T | = | Sampling time |
| $K_c$ | = | Controller gain |

# 3 System Concept

## 3.1 System Outline

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

- Voltage control of a BLDC motor using quadrature encoder HEDS-5640 A06
- Targeted at PowerPC MPC5554DEMO Evaluation Board (MPC554DEMO), Interface Board with UNI-3, 33395 Evaluation Motor Board, and MCG BLDC motor (IB23810)
- Control technique incorporates:
  - Voltage BLDC motor control with speed-closed loop
  - Both directions of rotation
  - 4-quadrant operation
  - Start from any motor position without rotor alignment
  - 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 loop close/open choice)
- FreeMASTER monitor
  - FreeMASTER graphical Control Page (required speed, actual motor speed, start/stop status, fault status)
  - FreeMASTER Speed Control Scope (observes required, ramp, and actual speeds, 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 14). The system incorporates the following hardware:

- Evaluation Board MPC5554DEMO
- Interface Board with UNI-3
- 33395 Evaluation Motor Board
- MCG BLDC motor (IB23810)
- Power Supply 12V DC, 2.7 Amps

The eTPU module runs the main control algorithm. The 3-phase PWM output signals for a 3-phase inverter are generated according to feedback signals from quadrature encoder and the input variable values, provided by the microprocessor CPU. Commutation of PWM phases is controlled by the CPU.

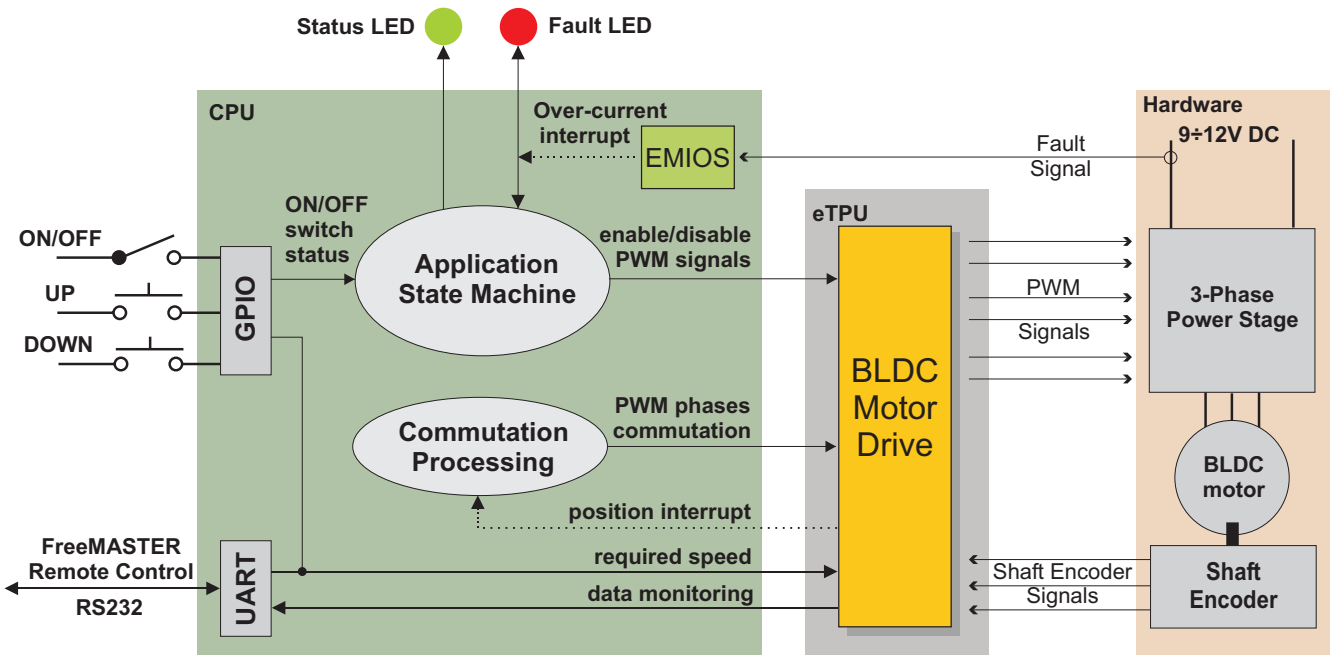



**Figure 14. System Concept**

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 speeds, which is passed to the eTPU.
- Ensures processing of position interrupts coming from eTPU QD channels and consecutive commutation of PWM phases. The interrupts are generated each time the QD position counter reaches a value which corresponds to the border between two commutation sectors.
- Periodically reads application data from eTPU DATA RAM in order to monitor application variables.
- In the event of an overcurrent fault, the PWM outputs are immediately temporarily disabled by the eTPU hardware. Then, after an interrupt latency, the CPU disables the PWM outputs permanently and displays the fault state.

The eTPU performs the following tasks:

- Six eTPU channels (PWMC) are used to generate PWM output signals.
- Three eTPU channels (QD) are used to process quadrature encoder signals.
- eTPU controls a speed closed loop. The actual motor speed is calculated based on the QD position counter and QD last edge time and compared with the required speed, 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 PWM generator as a newly corrected value of the applied motor voltage.

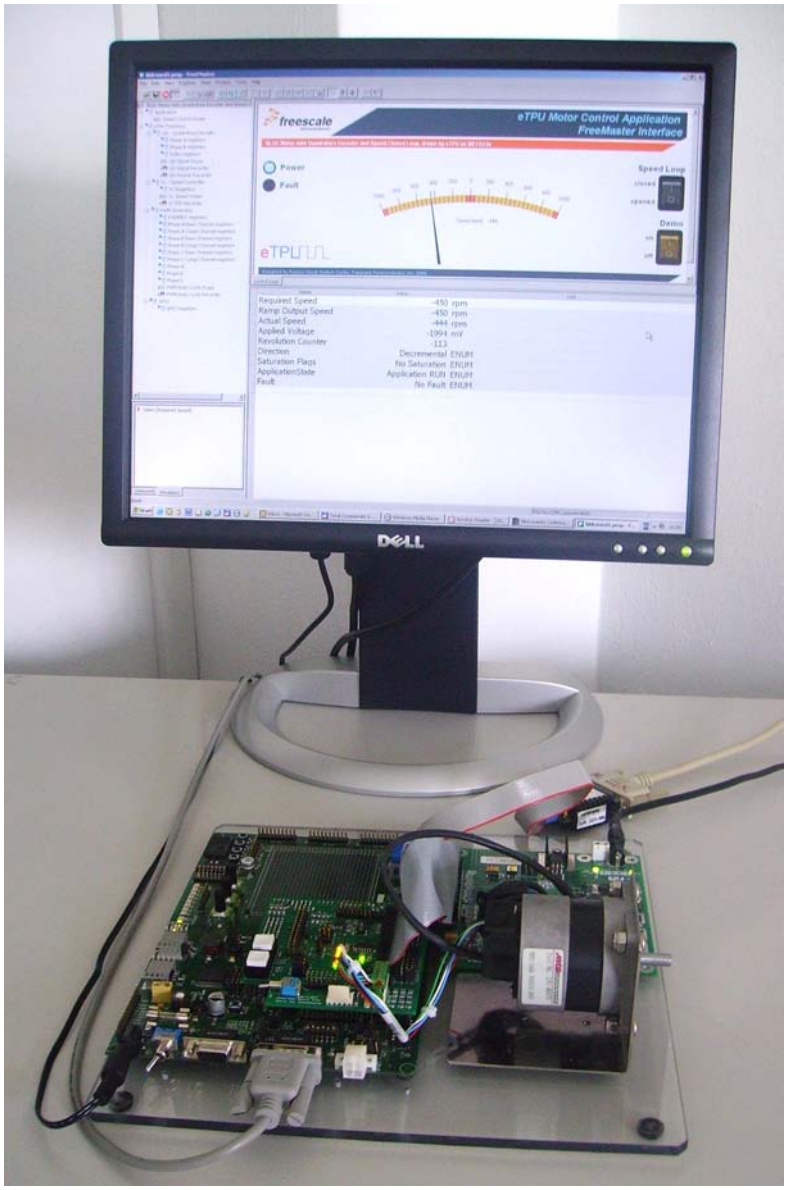



**Figure 15. The Application and FreeMASTER Screen**

## 3.2.1 User Interface

The application is interfaced by the following:

- ON/OFF switch on the Interface Board with UNI-3
- Up/Down buttons on the Interface Board with UNI-3, or FreeMASTER running on a PC connected to the MPC5554DEMO 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 Interface

Board, or by the FreeMASTER on the PC. The FreeMASTER also displays a control page, real-time values of application variables, and their time behaviour 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 MPC5554DEMO 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 7.

## 3.3 Hardware Implementation and Application Setup

As previously stated, the application runs on the MPC5554 family of PowerPC microprocessors using the following:

- MPC5554DEMO
- Interface Board with UNI-3
- 33395 Evaluation Motor Board
- 3-phase MCG BLDC motor (IB23810)
- Power Supply, 12V DC, minimum 2.7 Amps

Figure 16 shows the connection of these parts. All system parts are supplied by Freescale and documented according to references.

### 3.3.1 PowerPC MPC5554 Evaluation Board (MPC5554DEMO)

This board is not intended to be a full evaluation board for the MPC5554, but shows a minimal system for learning about the new MPC5500 family of product.

The FLASH memory placed on the MPC5554 has three address spaces. Low and mid address spaces are 256-Kbytes and high address spaces is 1.5 Mbyte in size. It gives a total memory space of 2Mbytes.

For more information, refer to Reference 2.

Table 2 lists all MPC5554DEMO jumper settings used in the application.

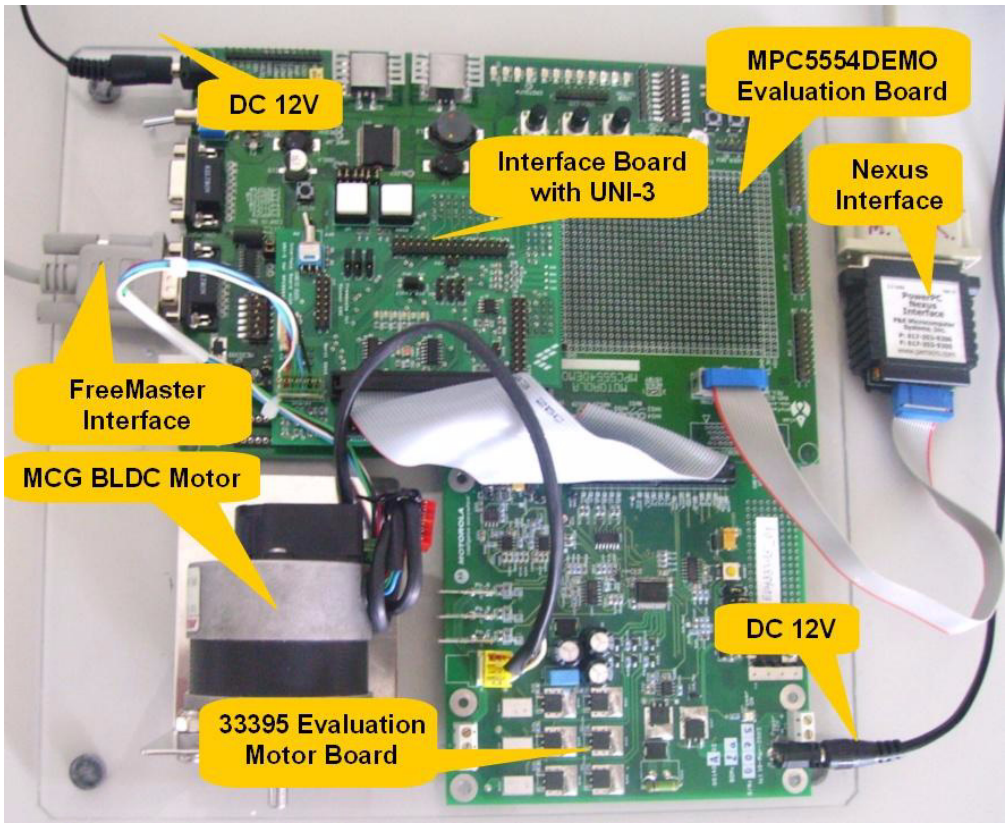



**Figure 16. Connection of Application Parts**

**Table 2. . MPC5554DEMO Jumper Settings.**

| Jumper | Setting |
|---|---|
| JP1 - 1 | 1 - 2 |
| JP1 - 2 | 1 - 2 |
| JP2 | 1 - 2 3 |
| JP3 | 1 - 2 |
| JP4 | 1 - 2 3 |
| JP5 | 1 - 2 |
| VRH_EN | 1 - 2 |
| SRAM_SEL | 1 - 2 3 |
| VSTBY_SWITCH | ON |

| CAN_SEL | Setting |
|---|---|
| 1 | 1 2 |
| 2 | 1 2 |
| 3 | 1 2 |
| 4 | 1 2 |
| 5 | 1 2 |
| 6 | 1 2 |

| CONFIG SWITCH | Setting |
|---|---|
| 1 | ON |
| 2 | OFF |
| 3 | ON |
| 4 | OFF |
| 5 | ON |
| 6 | OFF |

## 3.3.2 Flashing the MPC5554DEMO

The eSys Flasher utility can be used for programming code into the FLASH memory on the MPC5554DEMO. Check for correct setting of switches and jumpers.The flashing procedure is as follows:

1. Run Metrowerks MPC55xx V1.5b2 and open the project. Choose the Intflash 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 eSysFlasher application. In the Target Configuration window select the type of the BDM Communication as P&E Wiggler. Click OK to close the window.
3. Go to the Program section by clicking the “Program Flash” button (see Figure 17). Select the Binary Image, set Address as 0x0 and check the “Verify after program” option (see Figure 18). Press the “Program” and select intflash.bin file. Finally, press “Open” 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 MPC5554Demo. The application should now run from the FLASH.

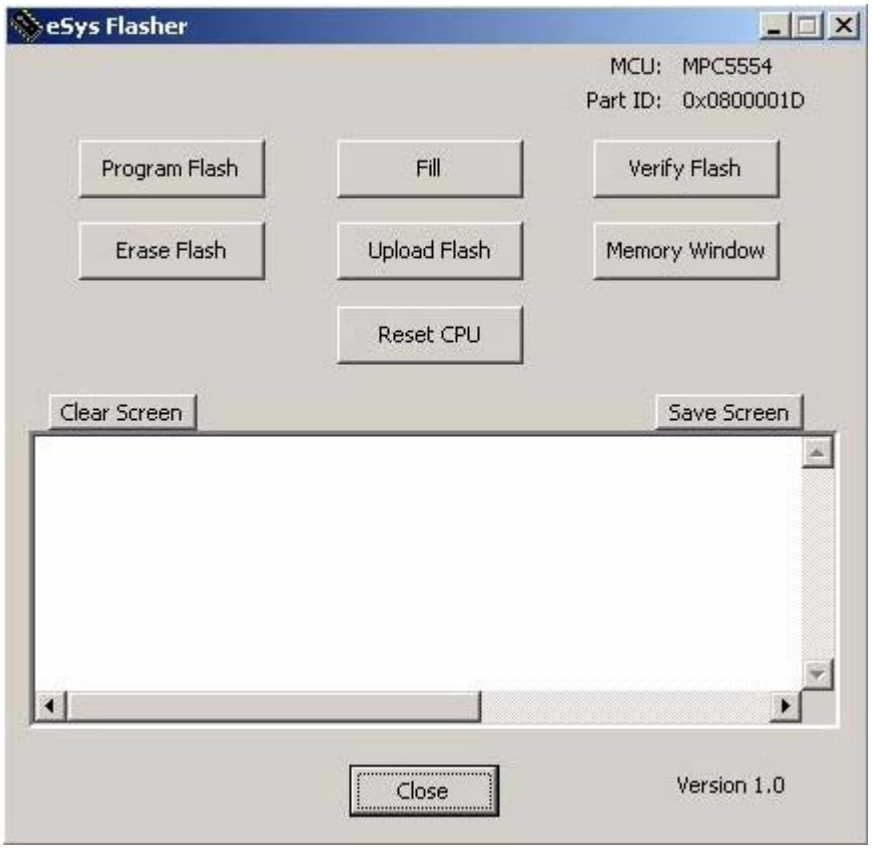



**Figure 17. eSysFlasher Target Configuration Window**

The eSYS Flasher application can be downloaded from http://www.freescale.com

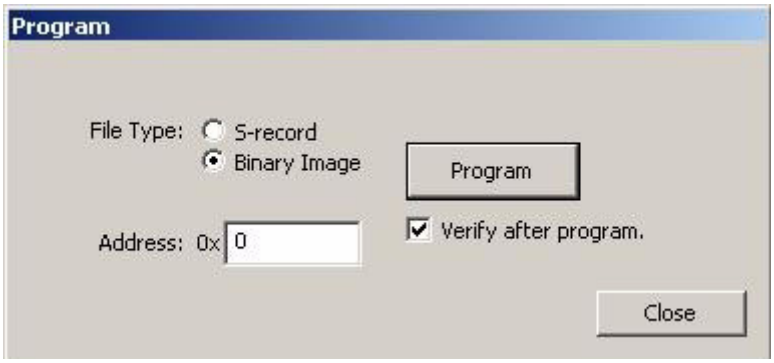



**Figure 18. eSys Flasher Program Window**

## 3.3.3 Interface Board with UNI-3

This board enables to connect the power stage with a motor to the MPC5554DEMO Board and can be used by software and hardware developers to test programs and tools. It supports algorithms that use Hall sensors, LEM sensors, encoder feedback and Back-EMF (electromotive force) signals for sensors control. Input connections are made via connectors on the bottom side of the board and headers on theMPC5554DEMO Board. Output connections are made via 40-pin UNI-3 connector and expansion headers. Power requirements are met by input connectors.

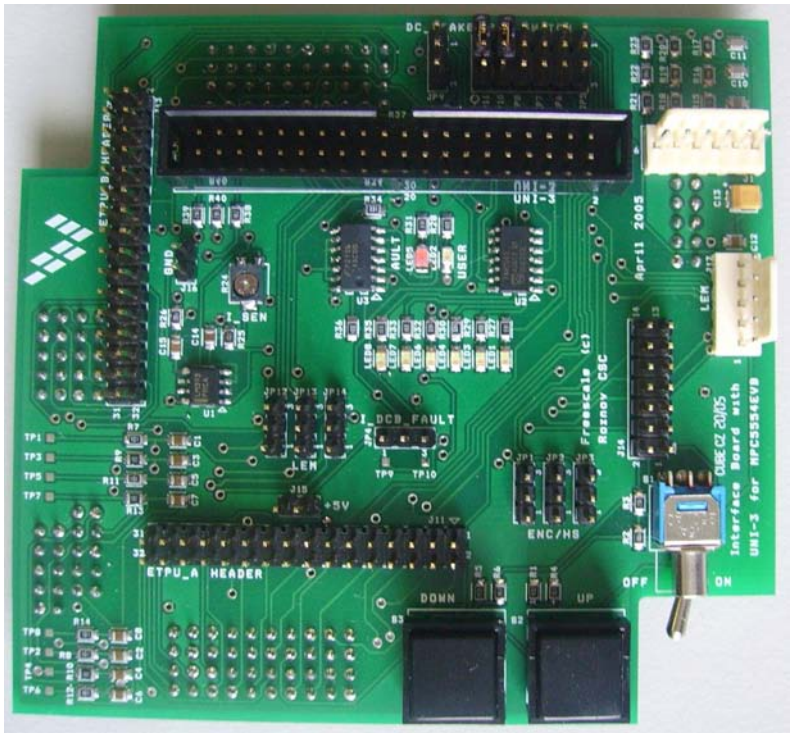



**Figure 19. Interface Board with UNI-3**

For more information, refer to Reference 3.

### 3.3.4 Setting Overcurrent Level

The over-current fault signal is connected to the eMIOS Output Disable Input pin (eMIOS 10) that enables, together with a proper eTPU configuration, handling the fault by eTPU hardware. This connection is part of the MPC5554. In order to enable handling the fault also by a software, the fault signal, available on eMIOS 10 pin generates interrupt request to the CPU in case of a fault.

The over-current level is set by the trimmer R24 (I_SEN) on the Interface Board with UNI-3 (see Figure 20). Reference 3 describes what voltage must the trimmer define for the over-current comparator. Do the following steps in order to set up the over-current level properly without measuring the voltage:

1. Connect all system parts according to Figure 16.
2. Download and start the application.
3. Turn ON/OFF switch ON. Using Up and Down buttons set the required speed to the maximum.
4. Adjust the R24 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.
6. Turn ON/OFF switch ON. Using Up and Down buttons set the required speed to the maximum.
7. If the application goes to the fault state during the acceleration, adjust the R24 trimmer level somewhat higher, so that the motor can get to the maximum speed.

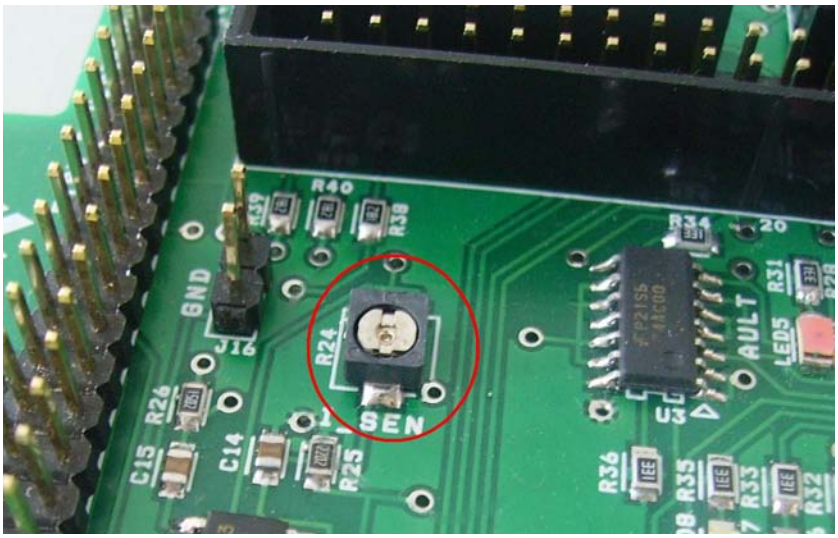



**Figure 20. Overcurrent Level Trimmer on Interface Board with UNI-3 (R24)**

### 3.3.5 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 MPC5554EVB and Interface Board with UNI-3, 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 the three phase currents, reconstructs DC-bus current from them, DC-bus voltage, Back-EMF voltages with zero cross sensing. 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 integrated circuitry, sensing and control circuitry, power N-MOSFET transistors, DC-Bus brake chopper, as well as various interface connectors for the supply and the motor.

For more information, refer to Reference 4.

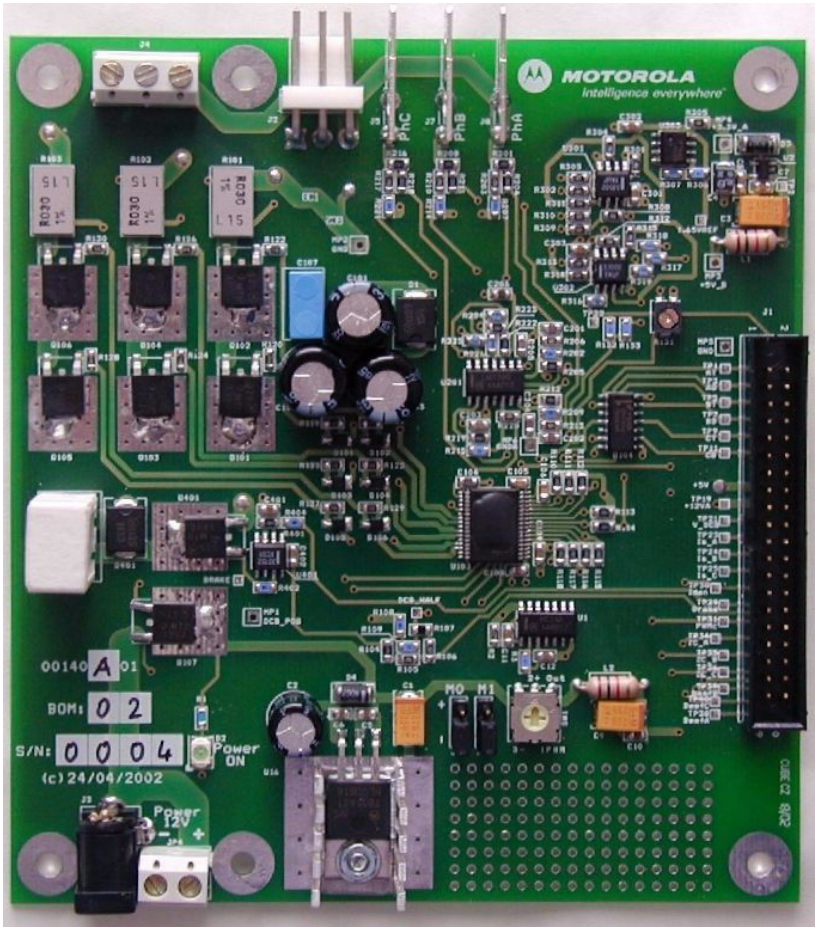



**Figure 21. 33395 Evaluation Motor Board**

## 3.3.6 BLDC Motor with Quadrature Encoder

The enclosed motor is a low-voltage MCG BLDC motor (IB23810). The motor characteristics in Table 3 apply to operation at 25°C.

**Table 3. MCG BLDC Motor (IB23810) Motor Characteristics**

| Characteristic | Tolerance | Units | Value |
|---|---|---|---|
| Max. operating speed | MAX. | R.P.M. | 5000 |
| Continuous torque | MAX. | OZ-IN | 20 |
| Peak torque | MAX. | OZ-IN | 60 |

**Table 3. MCG BLDC Motor (IB23810) Motor Characteristics (continued)**

| Characteristic | Tolerance | Units | Value |
|---|---|---|---|
| Continuous current | MAX. | AMPS | 2.0 |
| Peak current | MAX. | AMPS | 5.9 |
| Torque sensitivity | ±10% | OZ-IN/AMPS | 11.4 |
| Back EMF constant | ±10% | V/K R.P.M. | 8.4 |
| D.C. resistance | ±10% | OHMS | 3.35 |
| Inductance | ±15% | mH | 6.32 |
| Rotor inertia | NOM. | OZ-IN-SEC$^2$ | 0.0011 |
| Weight | NOM. | LBS | 1.18 |

Figure 30 depicts the motor timing. For more motor specifications, refer to Reference 5.

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 6.

### 3.3.7 Power Supply

The power supply 12V/2.7A, is also used to power the 33395 Evaluation Motor Board. The application is scaled for this 12V power supply.

# 4 Software Design

This section describes the software design of the BLDC motor 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

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 to handle all eTPU function initialization settings, and so ensures the correct cooperation of eTPU functions.

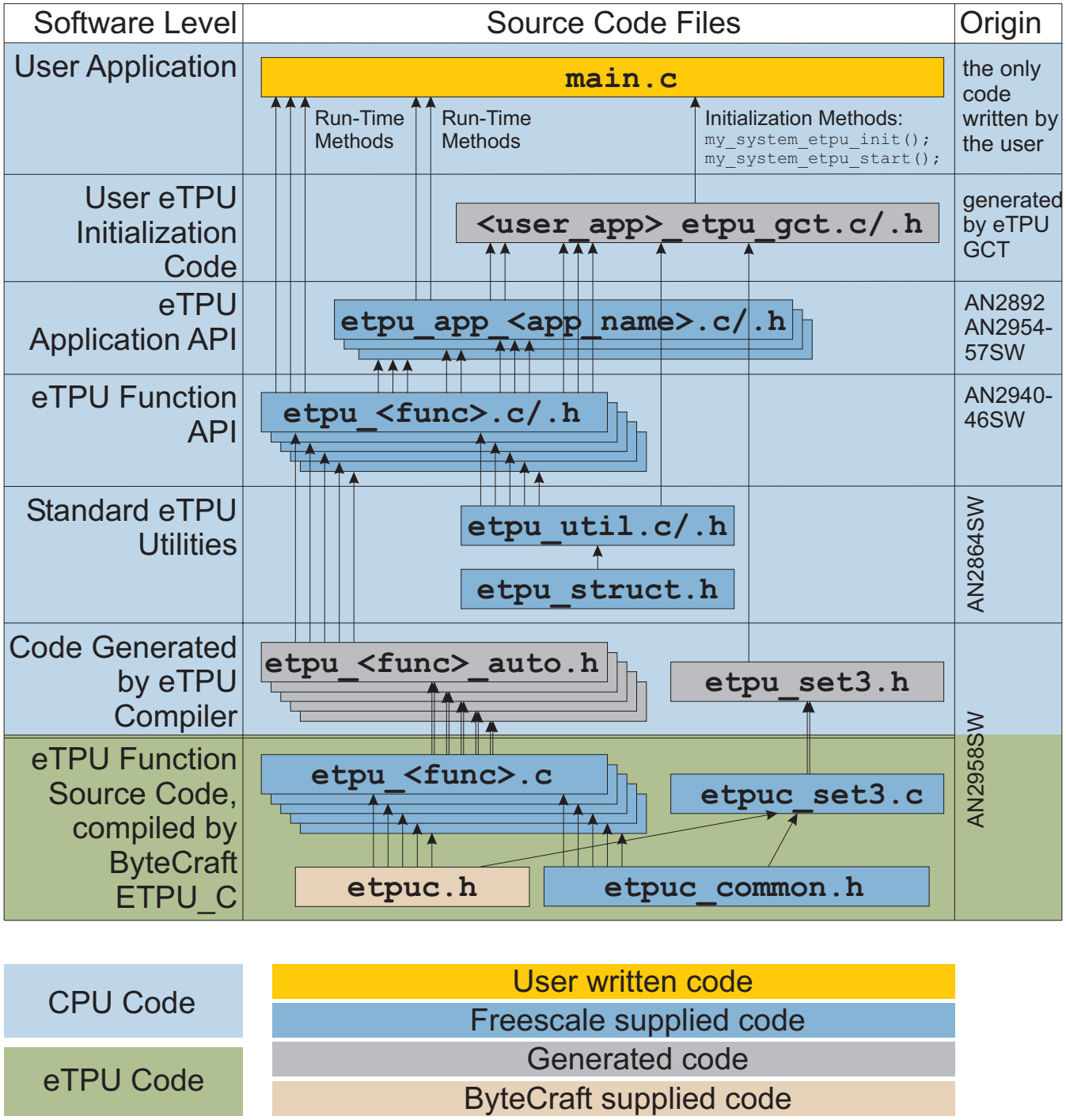



**Figure 22. eTPU Project Structure**

# 4.1 CPU Software Flowchart

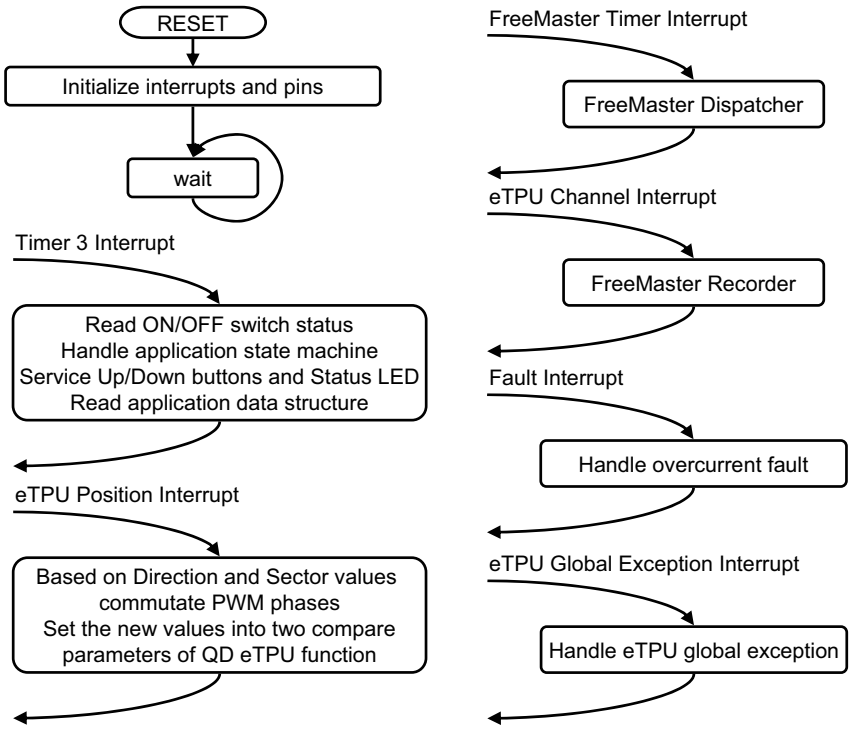



**Figure 23. CPU Software Flowchart**

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

## 4.1.1 Timer Interrupt Service Routine

The timer interrupt is handled by the timer_isr function. The following actions are performed periodically, in timer_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_bldcmesl1_get_data` (see 4.3).

### 4.1.2 Position Interrupt Service Routine

The eTPU position interrupt, which is raised by the QD eTPU function running on eTPU channels 1 and 2, is handled by the etpu_ch1_ch2_isr function. The `fs_etpu_app_bldcmesl1_commutate` application API function is called to ensure commutations of PWM phases (based on actual direction and position of the motion system). The information about the actual direction and the state of position counter is provided by the eTPU QD function. Further this application API function sets the QD eTPU function parameters *pc_interrupt1* and *pc_interrupt2* as a new compare values for QD to ensure generation of the next position interrupt.

### 4.1.3 FreeMASTER Interrupt Service Routine

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

### 4.1.4 eTPU Channel Interrupt Service Routine

This interrupt, which is raised every PWM period by the PWMMDC 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 configuration of application variable time courses with a PWM-period time resolution.

### 4.1.5 Fault Interrupt Service Routine

The over-current fault interrupt, which is raised by eMIOS input function running on eMIOS channel 10, is handled by the emios_isr function. The following actions are performed in order to switch the motor off:

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

### 4.1.6 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

The application state diagram consists of seven states (see Figure 24). After reset, the application goes firstly 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 24), 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 24), 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.

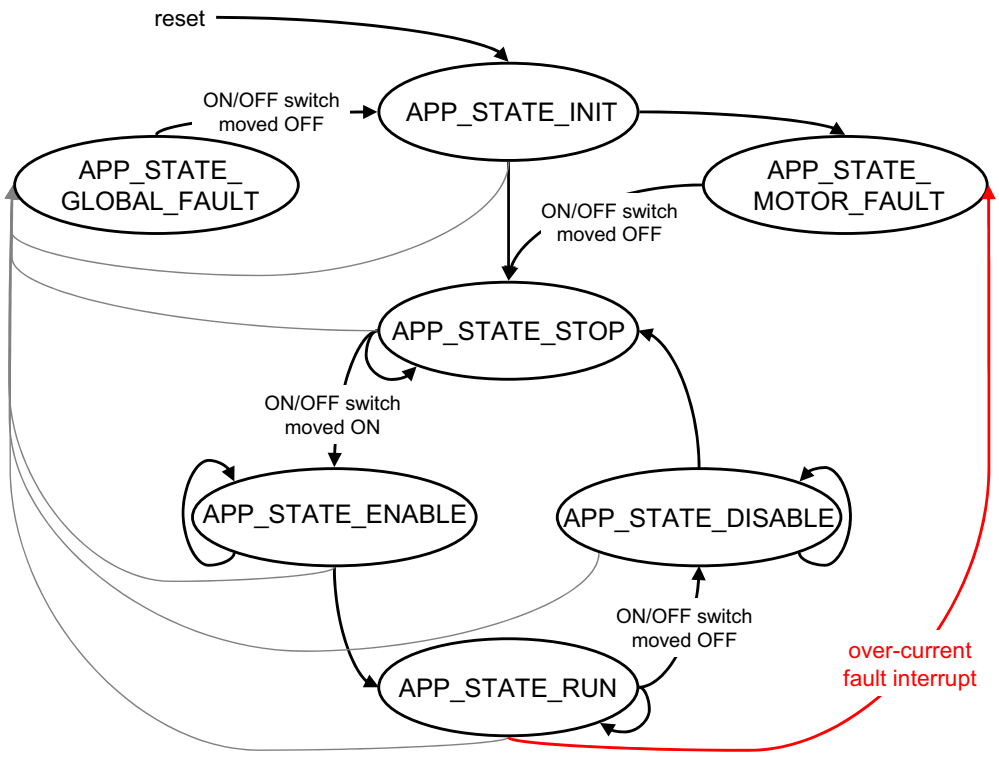



**Figure 24. Application State Diagram**

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. The following actions are performed in order 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 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 `bldcmesl1_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/webapp/sps/site/prod_summary.jsp?code=eTPU. For more information, refer to Reference 11.

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

- channel filter mode = three-sample mode
- channel filter clock = etpuclk div 64

  The input signals (from Hall sensors) are filtered by channel filters. The filter settings guarantee filtering all noise pulses up to a width of 1us and pass pulses from a width of 1.5us (at 128 MHz system clock).
- TCR1 source = etpuclk div 2
- TCR1 prescaler = 1

  The TCR1 internal eTPU clock is set to its maximum rate of 64 MHz (at 128 MHz system clock), corresponding to the 16ns resolution of generated PWM signals.
- TCR2 source = etpuclk div 8
- TCR2 prescaler = 2

  The TCR2 internal eTPU clock is set to a rate of 8 MHz (at 128MHz system clock). The TCR2 clock settings are optimized for motor speed calculation precision.

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 5 - speed controller (SC)
  Channel 7 - PWM master for DC motors (PWMMDC)
  Channel 8 - PWM commuted (PWMC) - phase A - base channel
  Channel 10 - PWM commuted (PWMC) - phase B - base channel
  Channel 12 - PWM commuted (PWMC) - phase C - base channel

  These eTPU channels are initialized by the `fs_etpu_app_bldcmesl1_init` eTPU application API function (see 4.3). The application settings are as follows:
  - PWM phases-type is commuted complementary pairs
  - PWM frequency 20kHz
  - PWM dead-time 1µs
  - Motor speed range 1 200 RPM
  - Motor speed minimum 10 RPM
  - DC-bus voltage 12V
  - Number of motor pole pairs 2
  - Speed controller update frequency 500Hz
  - PI controller parameters:
    P-gain is 0.5 ($0x004000 * 2^{-15}$), and
    I-gain is 0.125 ($0x001000 * 2^{-15}$).
    The controller parameters were experimentally tuned.
  - Ramp parameters:
    0.25s to ramp up from zero to the maximum speed,
    0.25s to ramp down from the maximum speed to zero.
  - Number of quadrature encoder position counter increments per one revolution 2000

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, so starting the eTPU.

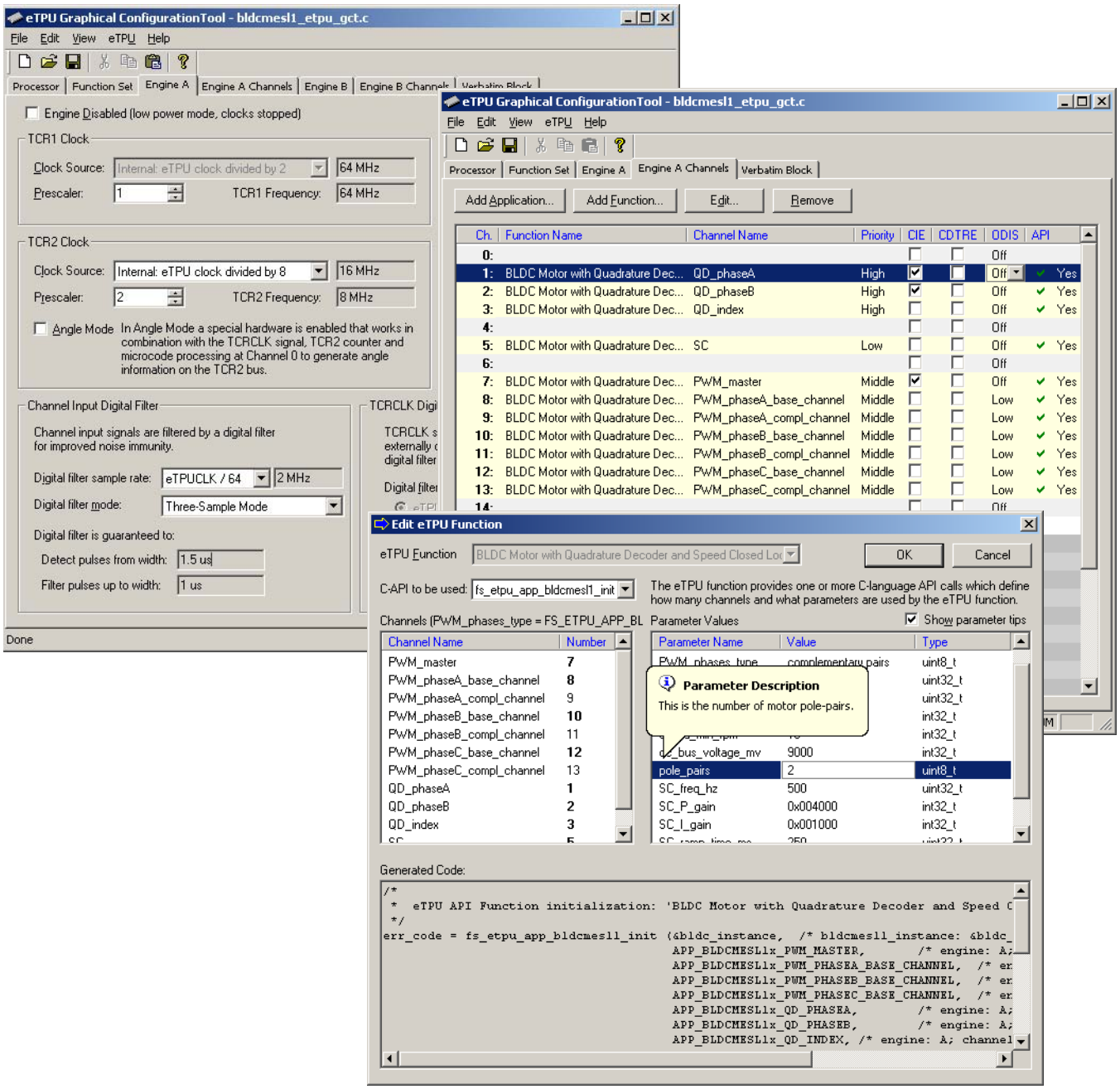



**Figure 25. eTPU Configuration Using the eTPU Graphical Configuration Tool**

## 4.2.1.2 Initialization of FreeMASTER Communication

Prior to the 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 the 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, to watch variables produced by the QD, and to 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_bldcmesl1_enable` application API routine. This routine also performs the motor alignment.

If the PWM phases were successfully enabled, the eMIOS channel 10 is configured as input, interrupt on falling edge, and APP_STATE_RUN is entered. Where the PWM phases were 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 can be set using the Up and Down buttons on the Interface 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
- Disable the generation of PWM signals

If PWM phases were successfully disabled, APP_STATE_STOP is entered. Where PWM phases were 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.

In order to shorten the eTPU application names, abbreviated application names are introduced. The abbreviations include:

- 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 BLDCMESL1 is an abbreviation for ‘BLDC motor with quadrature encoder and speed closed loop’ eTPU motor-control application. As there are several BLDC motor applications with quadrature encoder and speed closed loop, the number 1 denotes the first such application in order.

The BLDCMESL1 eTPU application API is described in the following paragraphs. There are 6 basic functions added to the BLDCMESL1 application API. The routines can be found in the `etpu_app_bldcmesl1.c/.h` files. All BLDCMESL1 application API routines will be described in order and are listed below:

- Initialization Function:

```
int32_t fs_etpu_app_bldcmesl1_init(
   bldcmesl1_instance_t * bldcmesl1_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_phaseB_channel,
uint8_t   QD_index_channel,
uint8_t   SC_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_P_gain,
int32_t   SC_I_gain,
uint32_t  SC_ramp_time_ms,
uint24_t  QD_pc_per_rev)
```

### 4.3.1 int32_t fs_etpu_app_bldcmesl1_init(...)

This routine is used to initialize the eTPU channels for the BLDC motor with quadrature encoder and speed closed loop application. This function has the following parameters:

- **bldcmesl1_instance (bldcmesl1_instance_t*)** - This is a pointer to bldcmesl1_instance_t structure, which is filled by `fs_etpu_app_bldcmesl1_init`. This structure must be declared in the user application. Where there are more instances of the application running simultaneously, there must be a separate bldcmesl1_instance_t structure for each one.
- **PWM_master_channel (uint8_t)** - This is the PWM master channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **PWM_phaseA_channel (uint8_t)** - This is 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_BLDCMESL1_COMPL_PAIRS), the complementary channel is one channel higher.
- **PWM_phaseB_channel (uint8_t)** - This is 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_BLDCMESL1_COMPL_PAIRS), the complementary channel is one channel higher.
- **PWM_phaseC_channel (uint8_t)** - This is 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_BLDCMESL1_COMPL_PAIRS), the complementary channel is one channel higher.
- **QD_phaseA_channel (uint8_t)** - This is the quadrature decoder phase A channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **QD_phaseB_channel (uint8_t)** - This is the quadrature decoder phase B channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.

- **QD_index_channel (uint8_t)** - This is the quadrature decoder index channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **SC_channel (uint8_t)** - This is the speed controller channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **PWM_phases_type (uint8_t)** - This parameter determines the type of all PWM phases. This parameter should be assigned a value of:
  FS_ETPU_APP_BLDCMESL1_SINGLE_CHANNELS, or
  FS_ETPU_APP_BLDCMESL1_COMPL_PAIRS.
- **PWM_freq_hz (uint32_t)** - This is the PWM frequency in Hz.
- **PWM_dead_time_ns (uint32_t)** - This is the PWM dead-time in ns.
- **speed_range_rpm (int32_t)** - This is the maximum motor speed in rpm.
- **speed_min_rpm (int32_t)** - This is the minimum (measurable) motor speed in rpm.
- **dc_bus_voltage_mv (int32_t)** - This is the DC-bus voltage in mV.
- **pole_pairs (uint8_t)** - This is the number of motor pole-pairs.
- **SC_freq_hz (uint32_t)** - This is 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_P_gain (fract24_t)** - This is the speed controller P-gain in 24-bit signed fractional format (9.15).
  0x008000 corresponds to 1.0
  0x000001 corresponds to 0.0000305 ($30.5*10^{-6}$)
  0x7FFFFF corresponds to 255.9999695
- **SC_I_gain (fract24_t)** - This is the speed controller I-gain in 24-bit signed fractional format (9.15).
  0x008000 corresponds to 1.0
  0x000001 corresponds to 0.0000305 ($30.5*10^{-6}$)
  0x7FFFFF corresponds to 255.9999695
- **SC_ramp_time_ms (uint32_t)** - This parameter 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.
- **QD_qd_pc_per_rev (uint24_t)** - This is the number of QD position counter increments per one revolution.

### 4.3.2 int32_t fs_etpu_app_bldcmesl1_enable(...)

This routine is used to enable the generation of PWM signals, to align motor to the start position, to initialize and enable position interrupts from the quadrature encoder and to start the speed controller. This function has the following parameters:

- **bldcmesl1_instance (bldcmesl1_instance_t*)** - This is a pointer to bldcmesl1_instance_t structure, which is filled by `fs_etpu_app_bldcmesl1_init`.

- **configuration (uint8_t)** - This is the required configuration of the SC. This parameter should be assigned a value of:
  FS_ETPU_APP_BLDCMESL1_SPEED_LOOP_OPENED, or
  FS_ETPU_APP_BLDCMESL1_SPEED_LOOP_CLOSED.
- **qd_pc_interrupt1 (int32_t)** - This is the first out of two QD pc_interrupt parameters to be set. QD eTPU function generates the position interrupt to the CPU when the QD position counter reaches this value. This is used for commutation purposes.
- **qd_pc_interrupt2 (int32_t)** - This is the second out of two QD pc_interrupt parameters to be set. QD eTPU function generates the position interrupt to the CPU when the QD position counter reaches this value. This is used for commutation purposes.

### 4.3.3 int32_t fs_etpu_app_bldcmesl1_disable (bldcmesl1_instance_t * bldcmesl1_instance)

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

- **bldcmesl1_instance (bldcmesl1_instance_t*)** - This is a pointer to bldcmesl1_instance_t structure, which is filled by `fs_etpu_app_bldcmesl1_init`.

### 4.3.4 void fs_etpu_app_bldcmesl1_set_speed_required(...)

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

- **bldcmesl1_instance (bldcmesl1_instance_t*)** - This is a pointer to bldcmesl1_instance_t structure, which is filled by `fs_etpu_app_bldcmesl1_init`.
- **speed_required_rpm (int32_t)** - This is the required motor speed in rpm.

### 4.3.5 void fs_etpu_app_bldcmesl1_commutate(...)

This routine is used to commutate PWM phases. This function has the following parameters:

- **bldcmesl1_instance (bldcmesl1_instance_t*)** - This is a pointer to bldcmesl1_instance_t structure, which is filled by `fs_etpu_app_bldcmesl1_init`.

### 4.3.6 void fs_etpu_app_bldcmesl1_get_data(...)

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

- **bldcmesl1_instance (bldcmesl1_instance_t*)** - This is a pointer to bldcmesl1_instance_t structure, which is filled by `fs_etpu_app_bldcmesl1_init`.
- **bldcmesl1_data (bldcmesl1_data_t*)** - This is a pointer to bldcmesl1_data_t structure of application state data, which is updated.

## 4.4 eTPU Block Diagram

The eTPU functions used to drive the BLDC motor with speed closed loop are located in the motor-control set of eTPU functions (set3 - DC motors). The eTPU functions within the set serve as building blocks for various motor-control applications. The following paragraphs describe the functionality of each block.

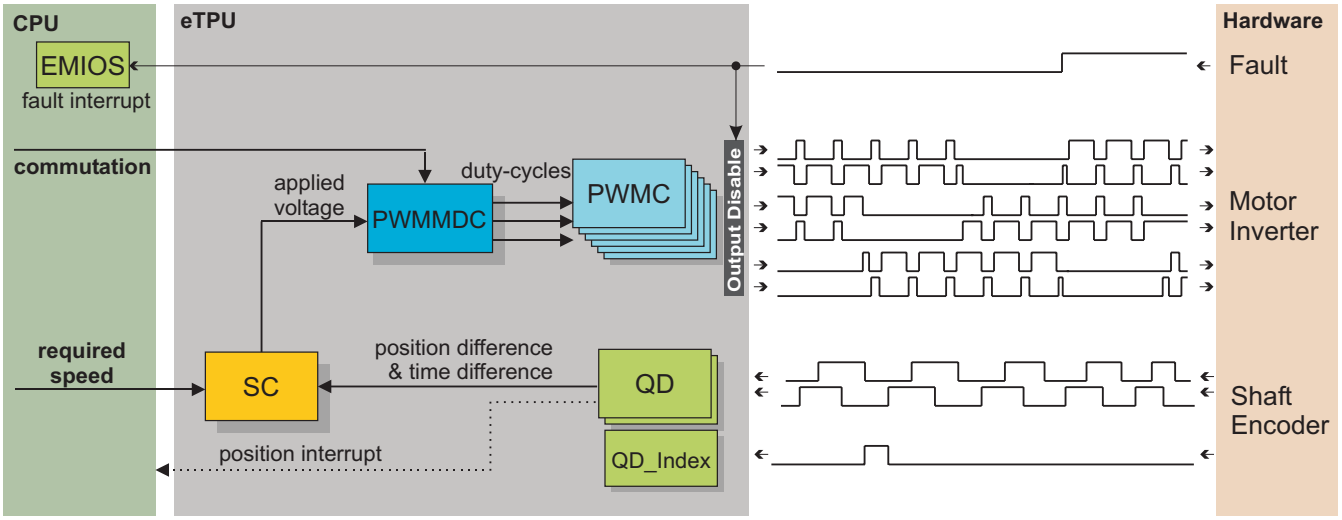



**Figure 26. Block Diagram of eTPU Processing**

### 4.4.1 PWM Generator (PWMMDC+PWMC)

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

- PWM - Master for DC Motors (PWMMDC)
- PWM - Full Range (PWMF)
- PWM - Commuted (PWMC)

The PWM Master for DC Motors (PWMMDC) function calculates a PWM duty cycle and updates the three PWM phases. The phases may be driven either by the PWM Full Range (PWMF) function, which enables a full (0% to 100%) duty-cycle range, or by the PWM Commuted (PWMC) function, which enables switching the phase ON and OFF. The PWMC function is used in the described application.

The PWMC function generates the PWM signals. The PWMMDC function controls three PWMC functions, three PWM phases, and does not generate any drive signal. The PWMMDC can be executed even on an eTPU channel not connected to an output pin.

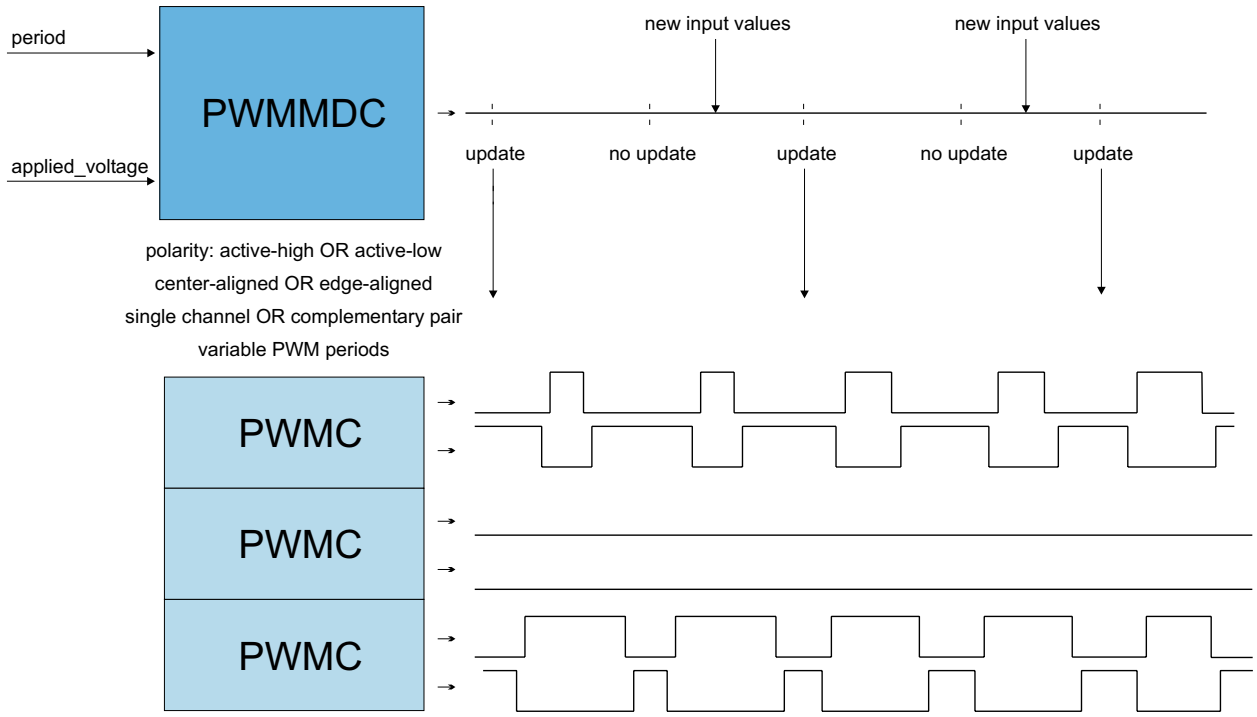
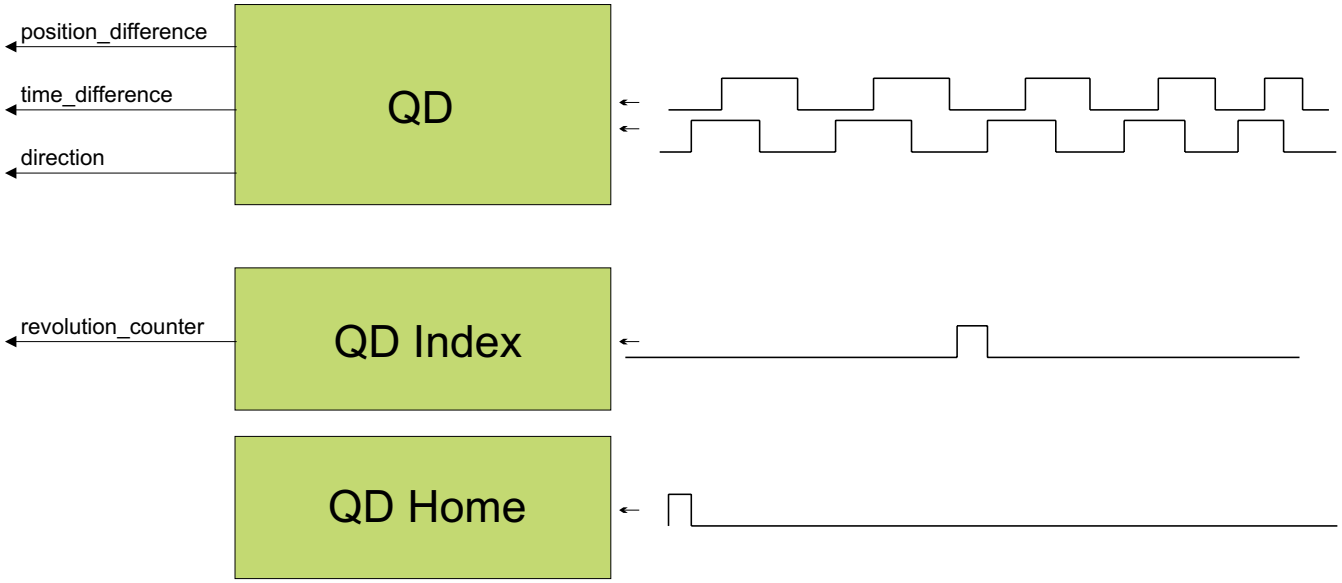



**Figure 27. Functionality of PWMMDC+PWMC**

For more details about the PWMMDC, PWMF, and PWMC eTPU functions, refer to Reference 12.

## 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 systems. It uses two channels to decode a pair of out-of-phase encoder signals and to produce a 24-bit bi-directional position counter, together with direction information, for the CPU. An additional input channels 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.




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

### 4.4.3 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 HD 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.

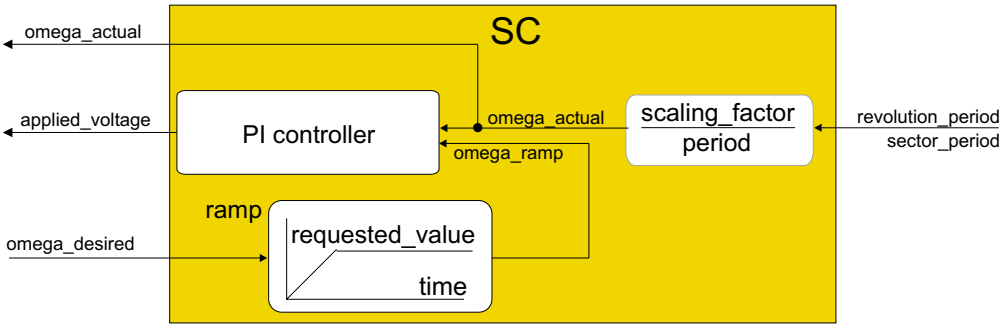



**Figure 28. Functionality of SC**

For more details about the SC 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 PWMC 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.

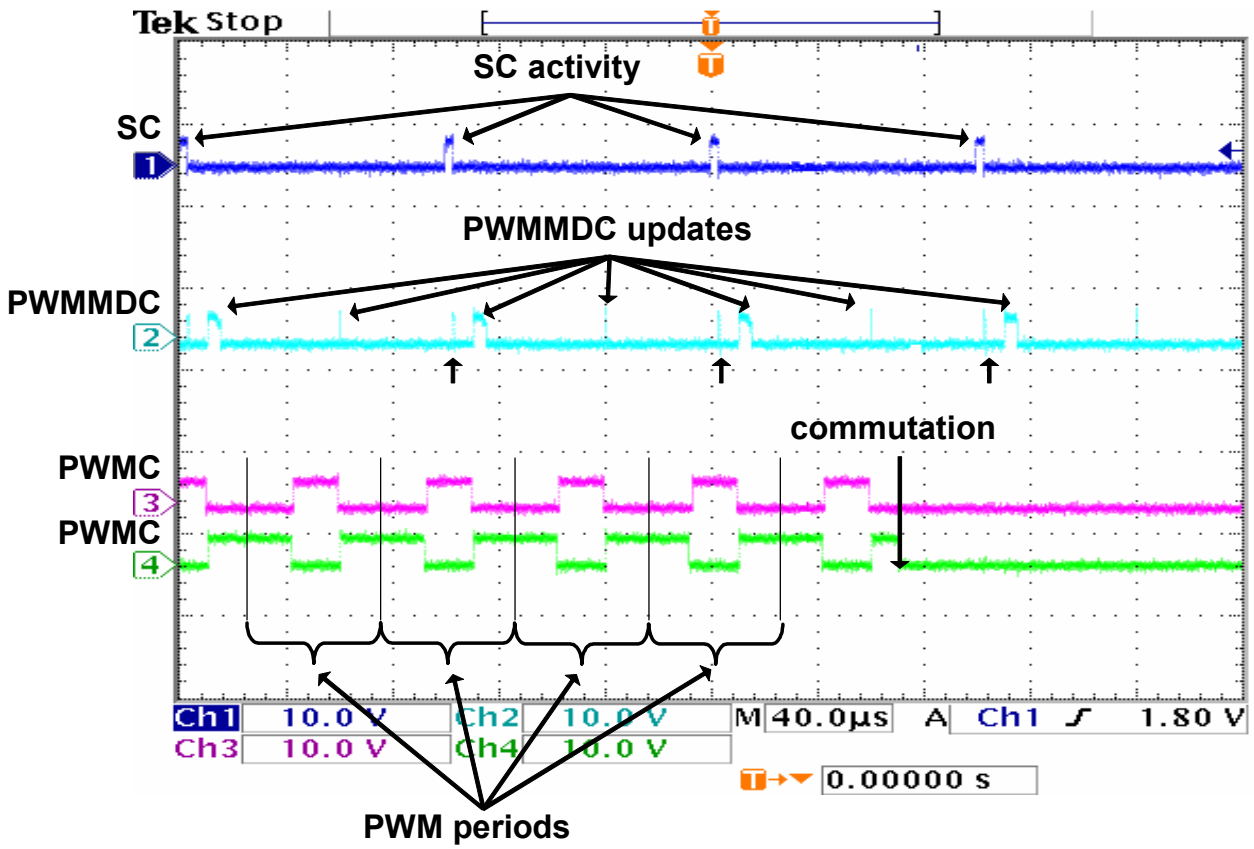



**Figure 29. Oscilloscope Screenshot and Explanation of eTPU Timing**

Figure 29 explains the application eTPU timing. The oscilloscope screen-shot depicts a typical situation described below. A live view on the oscilloscope screen enables the user to see the variation of SC and PWMMDC activity pulses, which determines the channel service latency ranges.

In Figure 29, signals 3 (pink) and 4 (green) are PWM signals of one phase. It is a complementary pair of center-aligned PWM signals. The base channel (3) is of active-high polarity, while the complementary channel (4) is active-low. The PWM phase commutation is recognizable on the screen. The PWM period is 50µs, which corresponds to a PWM frequency of 20kHz.

Signal 1 (blue) 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 output is the new value of applied motor voltage. This calculation is performed periodically at a 500Hz rate, which is every 40th PWM period.

Signal 2 (cyan) is generated by the PWM master for DC motors (PWMMDC) eTPU function. Its pulses determine the activity of the PWMMDC. Immediately after each SC pulse, a very narrow PWMMDC pulse occurs. These pulses determine the service time of an SC request to update the new value of applied motor voltage. Apart from these pulses, for every PWM period, a pulse will appear which signals a PWM

update. The PWM update activity pulse is wide when a new value of applied motor voltage has been processed; it is narrow when no new value has been processed and the PWM duty-cycles are not updated.

The `fs_etpu_pwmmdc_init_3ph` function parameter update_time enables the user to adjust the position of the PWMMDC activity pulse relative to the PWM period frame. The activity pulse has a scheduled 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. Reference 12 describes how to set the update_time value. The difference between the values of the `fs_etpu_pwmmdc_init_3ph` function parameter start_offset, and the `fs_etpu_sc_init` function parameter start_offset, determines the position of the SC activity pulse relative to the PWM period frame. The SC activity precedes the PWMMDC activity, so that the worst case SC latency does not affect the PWMMDC latency.

# 5 Implementation Notes

## 5.1 Scaling of Quantities

The BLDC motor control algorithm running on eTPU uses a 24-bit fractional representation for all real quantities except time. The 24-bit signed fractional format is 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.]

### 5.1.1 PI Controller Parameters

The PI controller parameters are set in a 32-bit extended fractional format 9.23. This format enables the user to set values in the range of -256.0 to $256.0 - 2^{-23}$. Internally, the parameter value is transformed into one of two 24-bit formats, either 9.15, or 1.23, based on the value.

## 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 which ensures 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:

*e*tpu_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.

## 5.3 Definition of Commutation Tables

The PWM phases are commuted by CPU on each position interrupt generated by one of two QD channels. This is actually done by applying two commutation commands. The first command turns a phase off, and the second turns another phase on. Such pairs of commutation commands must be defined for each sector border crossing, on each phase, and for both motor directions. The commutation command is a 32-bit word that consists of the following 8-bit parts.

- Channel number of the PWM phase base channel
- New base channel commutation state. It can be:
  - ON_ACTIVE_HIGH
  - ON_ACTIVE_LOW
  - OFF_LOW
  - OFF_HIGH
- New complementary channel commutation state. It can be:
  - ON_ACTIVE_HIGH
  - ON_ACTIVE_LOW
  - OFF_LOW
  - OFF_HIGH

- New phase options:
  - DUTY_POS
  - DUTY_NEG.

For a full description of all commutation command options, refer to Reference 10.

Figure 30 depicts the MCG BLDC motor (IB23810) motor timing diagram. The following example describes how to define several of the commutation commands based on this timing diagram:

For incremental direction, read the motor timing diagram from right to left. QD position interrupt, which indicates crossing border between sectors 3 and 1, comes at 120 electrical degrees (red dotted line). The PWM phase B is turned off, with the pin in low state, and phase C on, with active-high polarity on the base channel and active-low polarity on the complementary channel, on this position interrupt. Further more, the PWM phase B option is set to not change the calculated duty-cycle value (negative duty cycle), and the phase C option is set to positive duty-cycle value, which generates a positive voltage. So, the commutation commands associated with the sector 1 in an incremental motor direction are defined, using predefined macros, as follows:

```
((FS_ETPU_PWMMDC_DUTY_NEG       <<24)+
 (FS_ETPU_PWMMDC_OFF_LOW        <<16)+
 (FS_ETPU_PWMMDC_OFF_LOW        << 8)+
 APP_BLDCMESL10_PWM_PHASEB_BASE_CHANNEL)


((FS_ETPU_PWMMDC_DUTY_POS       <<24)+
 (FS_ETPU_PWMMDC_ON_ACTIVE_LOW  <<16)+
 (FS_ETPU_PWMMDC_ON_ACTIVE_HIGH << 8)+
 APP_BLDCMESL10_PWM_PHASEC_BASE_CHANNEL)
```

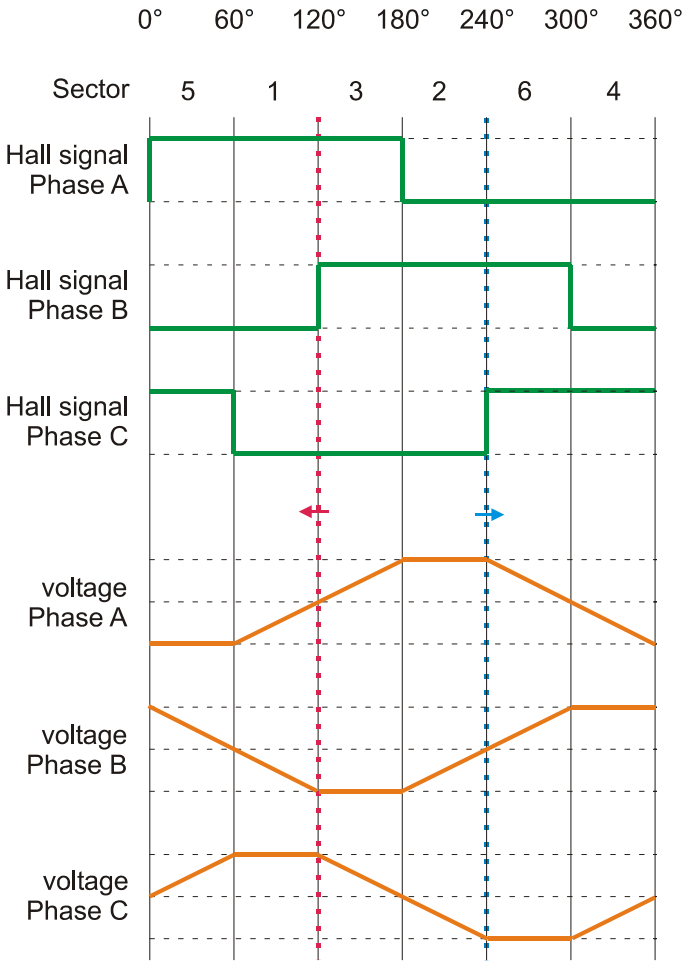



**Figure 30. MCG BLDC Motor (IB23810) Timing Diagram**

For decremental direction, read the motor timing diagram from left to right. QD position interrupt, which indicates crossing border between sectors 2 and 6, comes at 240 electrical degrees (blue dotted line). The PWM phase A is turned off and phase C on during this position interrupt. The PWM phase C option is set to negate the duty-cycle in order to generate negative phase voltage. So, the commutation commands associated with the sector 6 in a decremental motor direction are defined, using predefined macros, as follows:

```
((FS_ETPU_PWMMDC_DUTY_NEG       <<24)+
 (FS_ETPU_PWMMDC_OFF_LOW        <<16)+
 (FS_ETPU_PWMMDC_OFF_LOW        << 8)+
 APP_BLDCMESL10_PWM_PHASEA_BASE_CHANNEL)


((FS_ETPU_PWMMDC_DUTY_POS       <<24)+
 (FS_ETPU_PWMMDC_ON_ACTIVE_LOW  <<16)+
```

```
(FS_ETPU_PWMMDC_ON_ACTIVE_HIGH << 8)+
APP_BLDCMESL10_PWM_PHASEC_BASE_CHANNEL)
```

This way all commutation commands can be defined.

# 6 Microprocessor Usage

Table 4 shows how much memory is needed to run the application.

**Table 4. Memory Usage in Bytes**

| Memory | Available | Used |
|---|---|---|
| FLASH | 2M | 33 124 |
| RAM | 64K | 3 432 |
| eTPU code RAM | 16K | 6 120 |
| eTPU data RAM | 3K | 288 |

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

- According to Reference 12, the maximum eTPU load produced by PWM generation is 946 eTPU cycles per one PWM period. The PWM frequency is set to 20kHz, thus the PWM period is 3750 eTPU cycles (eTPU module clock is 64 MHz, half of the 128 MHz CPU clock).
- According to Reference 11, the speed controller calculation takes 324 eTPU cycles. The calculation is performed every 40th PWM period.
- According to Reference 9, 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.

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 when both the speed controller calculation and a QD signal transition are processed within one PWM period. 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 5 shows the eTPU module time load in several typical situations. For more information, refer to Reference 13.

**Table 5. eTPU Time Load**

| Situation | Average Time Load [%] | Peak Time Load Within PWM Period [%] |
|---|---|---|
| Motor Speed 300 RPM (120 commutations per second) | 25.5 | 42.0 |
| Motor Speed 10000 RPM (4000 commutations per second) | 33.6 | 42.0 |

# 7 Summary and Conclusions

This application note provides the user with a description of the demo application BLDC motor with quadrature encoder and speed closed loop. The application also demonstrates usage of the eTPU module on the PowerPC MPC5554, which results in a CPU independent motor drive. Lastly, the demo application is targeted at the MPC5554 family of devices, but it could be easily reused with any device that has an eTPU.

# 8 References

**Table 6. References**


| |
|---|
| 1. MPC5554 Reference Manual, MPC5554RM |
| 2. MPC5554DEMO User's Manual, MPC5554DEMO EVBUM |
| 3. Interface Board with UNI-3 User's Manual |
| 4. 33395 Evaluation Motor Board Designer Reference Manual DRM33395/D |
| 5. MCG's Motors web: http://www.mcg-net.com |
| 6. Quadrature Encoder HEDS-5640 A06 distributor's web: http://www.agilent.com/semiconductors |
| 7. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER" |
| 8. Enhanced Time Processing Unit Reference Manual, ETPURM |
| 9. "Using the Quadrature Decoder (QD) eTPU Function," AN2842 |
| 10. "Using the Hall Decoder (HD) eTPU Function," AN2841 |
| 11. "Using the Speed Controller (SC) eTPU Function," AN2843 |
| 12. "Using the DC Motor Control PWM eTPU Functions," AN2480 |
| 13. "Using the DC Motor Control eTPU Function Set (set3)," AN2958 |
| 14. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT" |
| 15. DSP56F80x MC PWM Module in Motor Control Applications, AN1927 |
| 16. "3-Phase BLDC Motor Control with Quadrature Encoder using DSP56F80x," AN1915/D |

## 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






AN3005
Rev. 1
05/2006