Freescale Semiconductor
Application Note

AN3008
Rev. 1, 05/2006

DC Motor with Speed and Current Closed Loops, 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 DC 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.

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

Table of Contents

freescale semiconductor

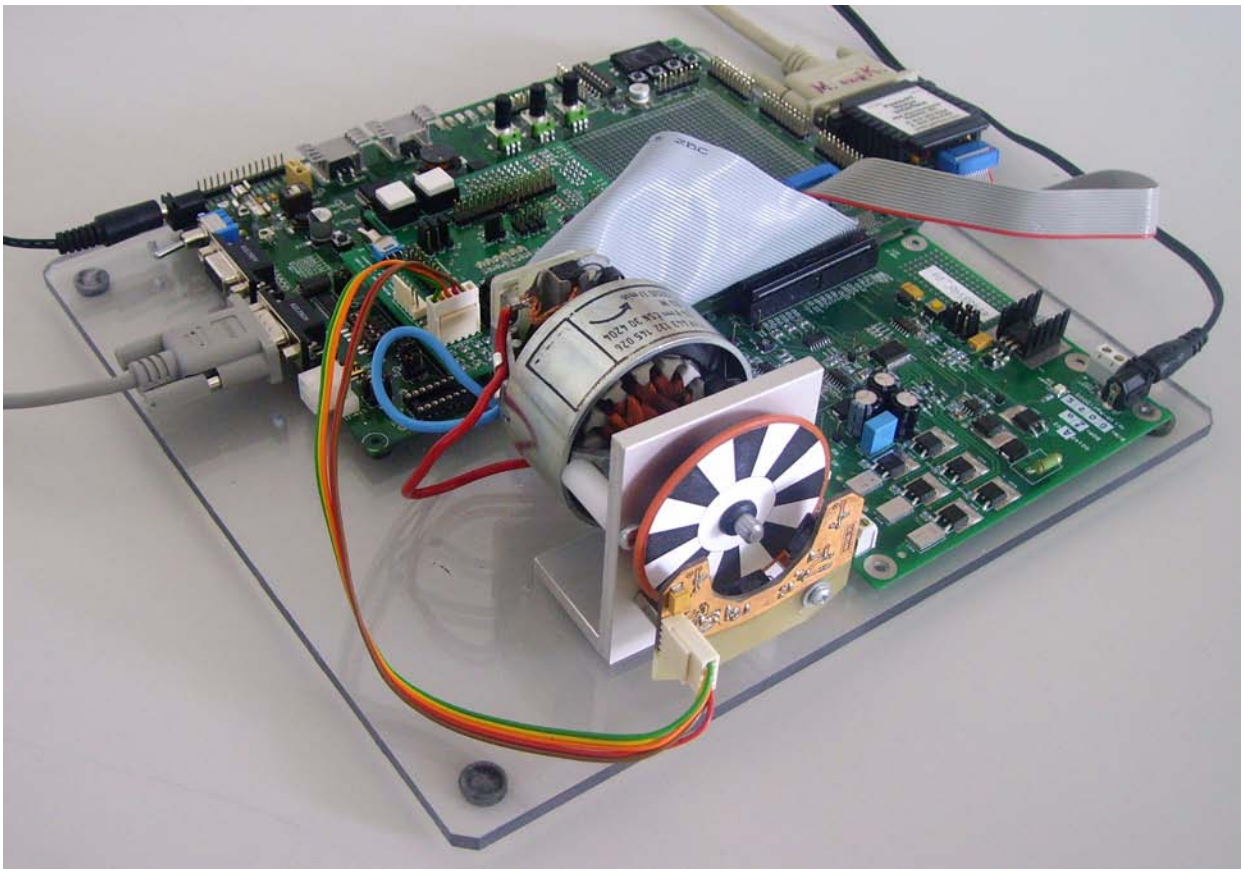

Figure 1. Using MPC5554DEMO, 33395 Evaluation Motor Board, and DC Motor

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 9.

2 Target Motor Theory

A DC motor is a rotating electric machine where the stator of a permanent magnet DC motor is composed of two or more permanent magnet pole pieces. The rotor is composed of windings which are connected to a mechanical commutator. The opposite polarities of the energized winding and the stator magnet attract and the rotor will rotate until it is aligned with the stator. Just as the rotor reaches alignment, the brushes move across the commutator contacts and energize the next winding (see Figure 2).

Notice that the commutator is staggered from the rotor poles. If the connections of a DC motor are reversed, the motor will change directions.

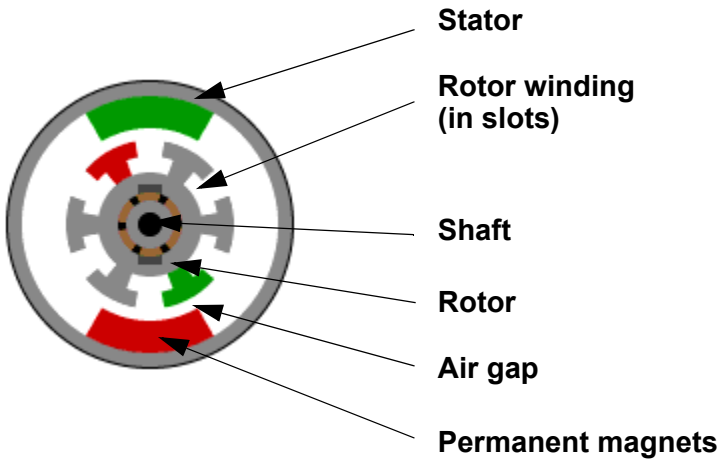

Figure 2. DC Motor—Cross Section

2.1 Digital Control of a DC Motor

For the common DC motor, a 2-phase power stage is used (see Figure 3). The power stage utilizes four power transistors that operate in either an independent or complementary mode.

In both modes, the power stage energizes two motor phases concurrently. The voltage is applied to the DC motor using a pulse width modulation (PWM) technique. 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 17.

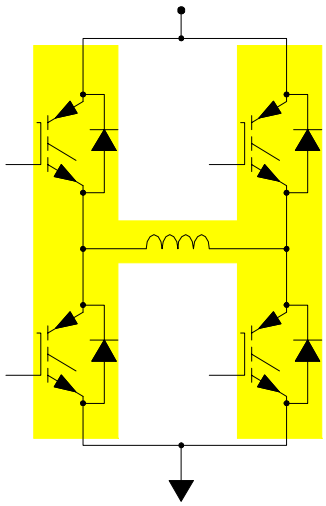

Figure 3. 2-Phase DC Motor Power Stage (H-Bridge)

2.1.1 Speed and Current Control

The motor speed 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, controllers the required DC-bus current. The required DC-bus current is controlled by a current controller, which is also implemented as a conventional proportional-integral (PI) controller. The difference between the actual and required DC-bus current 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.

The current controller, which is the inner-loop controller, is updated more frequently, for example every PWM period, compared to the speed controller, which is the outer-loop controller.

The speed controller, as well as the current 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 DC motor. The application meets the following performance specifications:

- Voltage control of a DC motor
- Targeted at PowerPC MPC5554DEMO Evaluation Board (MPC554DEMO), Interface Board with UNI-3, 33395 Evaluation Motor Board, and 24V, 3200RPM, DC motor
- Control technique incorporates:
  - Voltage DC motor control with speed and current closed loop
  - Both directions of rotation
  - 4-quadrant operation
  - Start from any motor position without rotor alignment
  - Minimum speed of 200 RPM
  - Maximum speed of 1200 RPM (limited by power supply)
- Manual interface (start/stop switch, up/down push button control, LED indication)
- FreeMASTER control interface (speed set-up)
- 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, required DC-bus current)
  - FreeMASTER current control scope (observes required and actual DC-bus currents, 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 4). The system incorporates the following hardware:

- Evaluation Board MPC5554DEMO
- Interface Board with UNI-3
- 33395 Evaluation Motor Board
- DC motor with optical, Hall-like sensors
- Power Supply 12V DC, 2.7 Amps

The eTPU module runs the main control algorithm. The 2-phase PWM output signals for a 2-phase inverter are generated according to feedback signals from optical sensors and the input variable values, provided by the microprocessor CPU.

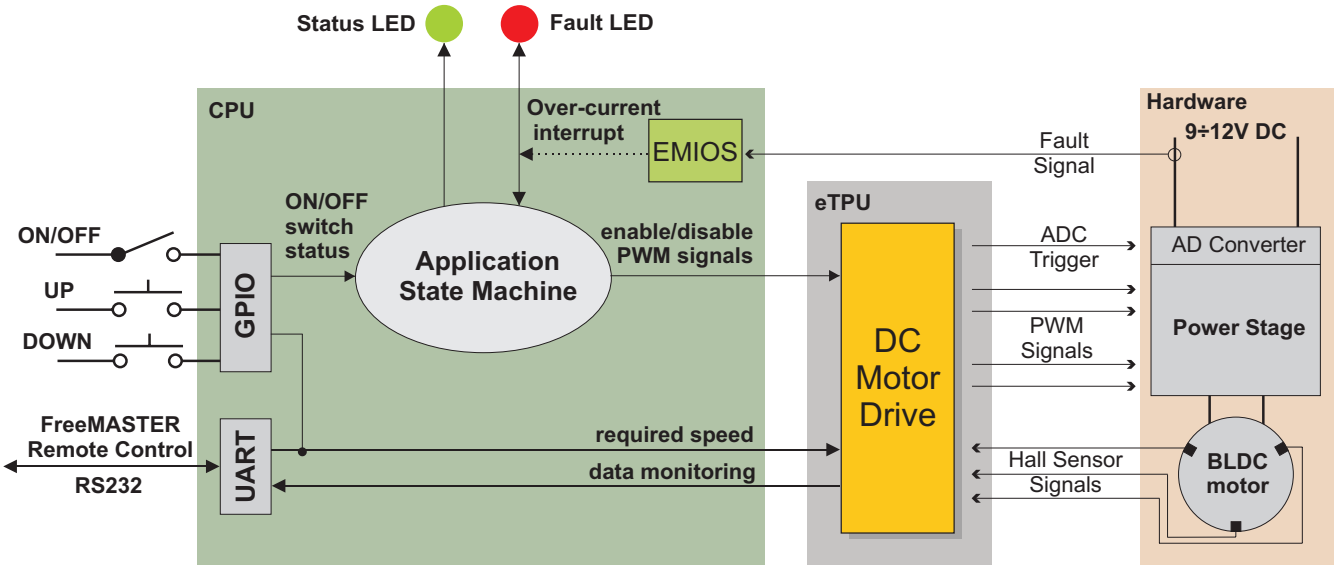

Figure 4. System Concept

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.
- 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:

- Four eTPU channels (PWMF) are used to generate PWM output signals.
- Three eTPU channels (HD) are used to process optical sensor signals. On each incoming edge, a revolution period is calculated.
- eTPU controls a speed closed loop. The actual motor speed is calculated based on the revolution period 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 current controller as a newly corrected value of the required DC-bus current value.
- eTPU controls a current closed loop. The actual DC-bus current is sampled by an external AD converter, and the converted value is transferred by DMA to the eTPU. The actual value is compared with the required speed, provided by the speed controller. The current PI control algorithm processes the error between the required and actual current. The PI controller output is passed to the PWM generator as a newly corrected value of the applied motor voltage.

Figure 5. 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 8.

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
- DC motor with optical, Hall-like sensors
- Power Supply, 12V DC, minimum 2.7 Amps

Figure 6 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 1 lists all MPC5554DEMO jumper settings used in the application.

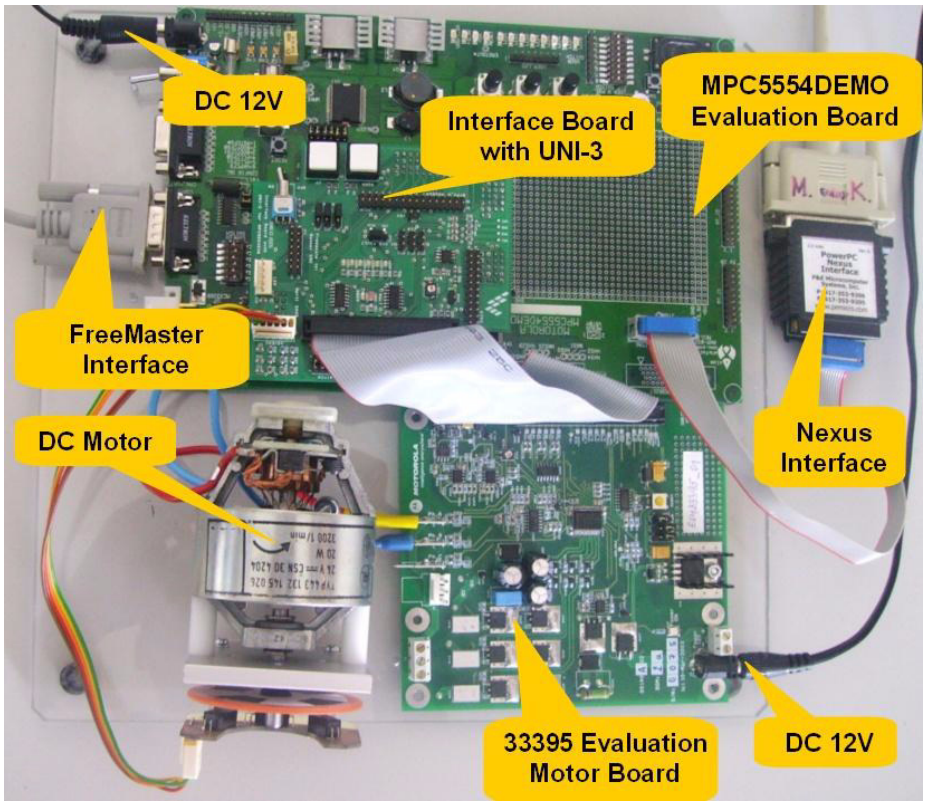

Figure 6. Connection of Application Parts

Table 1. . 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 7). Select the Binary Image, set Address as 0x0 and check the “Verify after program” option (see Figure 8). 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.

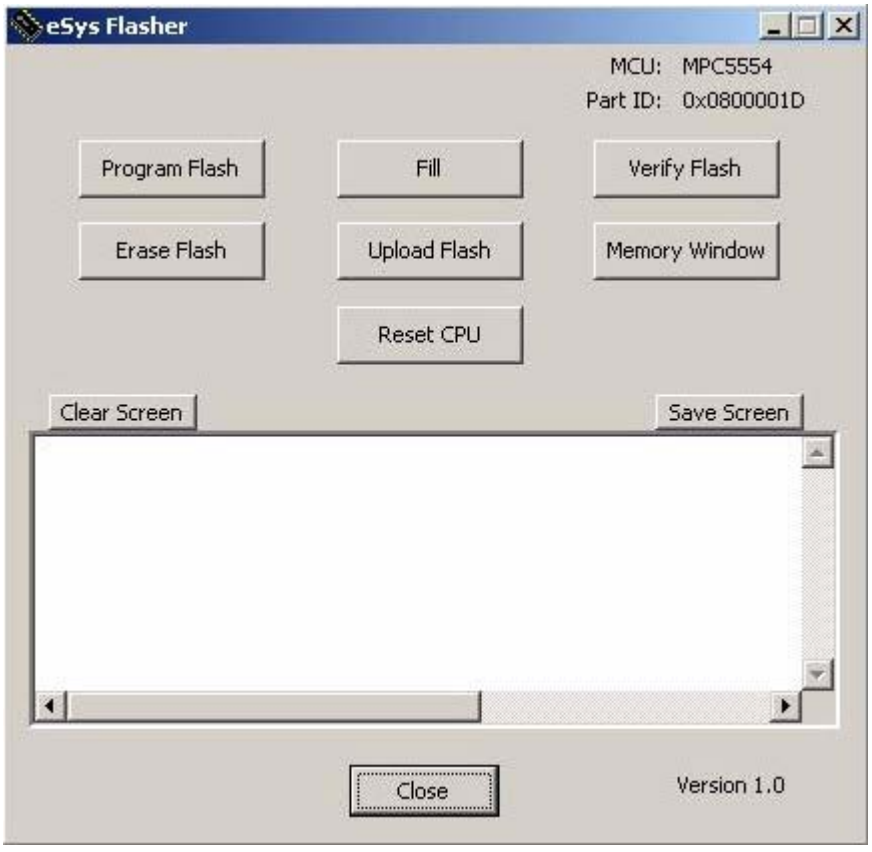

Figure 7. eSysFlasher Target Configuration Window

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

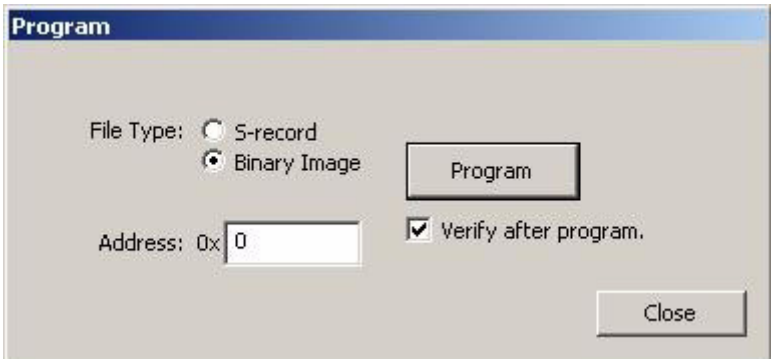

Figure 8. eSys Flasher Program Window

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.

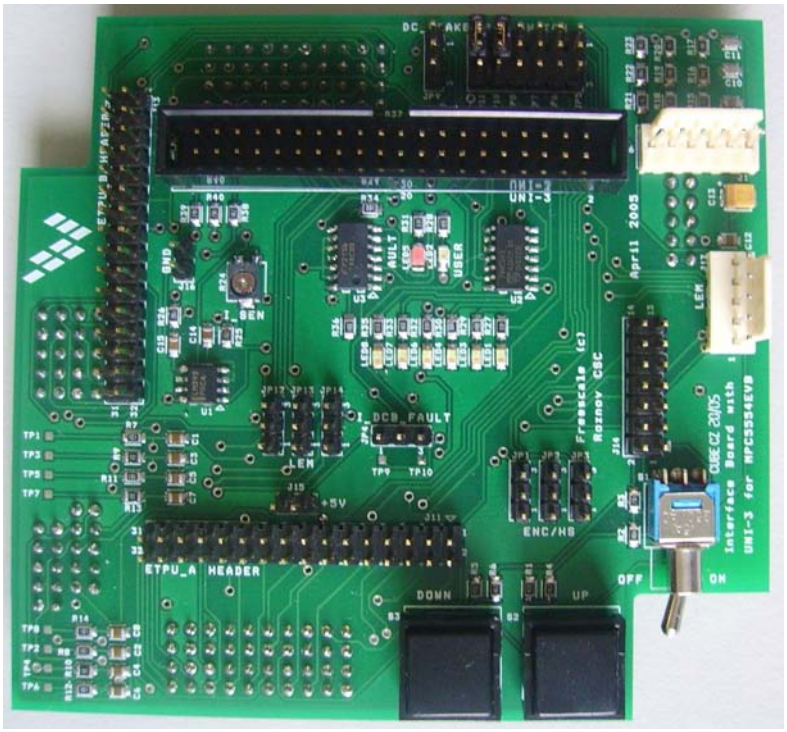

Figure 9. Interface Board with UNI-3

For more information, refer to Reference 4.

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 10). Reference 4 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 6.
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.

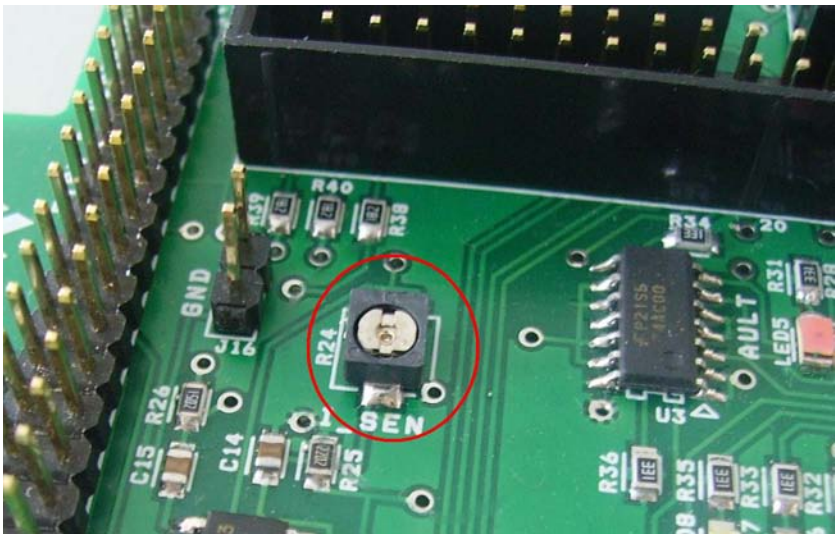

Figure 10. Overcurrent Level Trimmer on Interface Board with UNI-3 (R24)

3.3.5 33395 Evaluation 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 5.

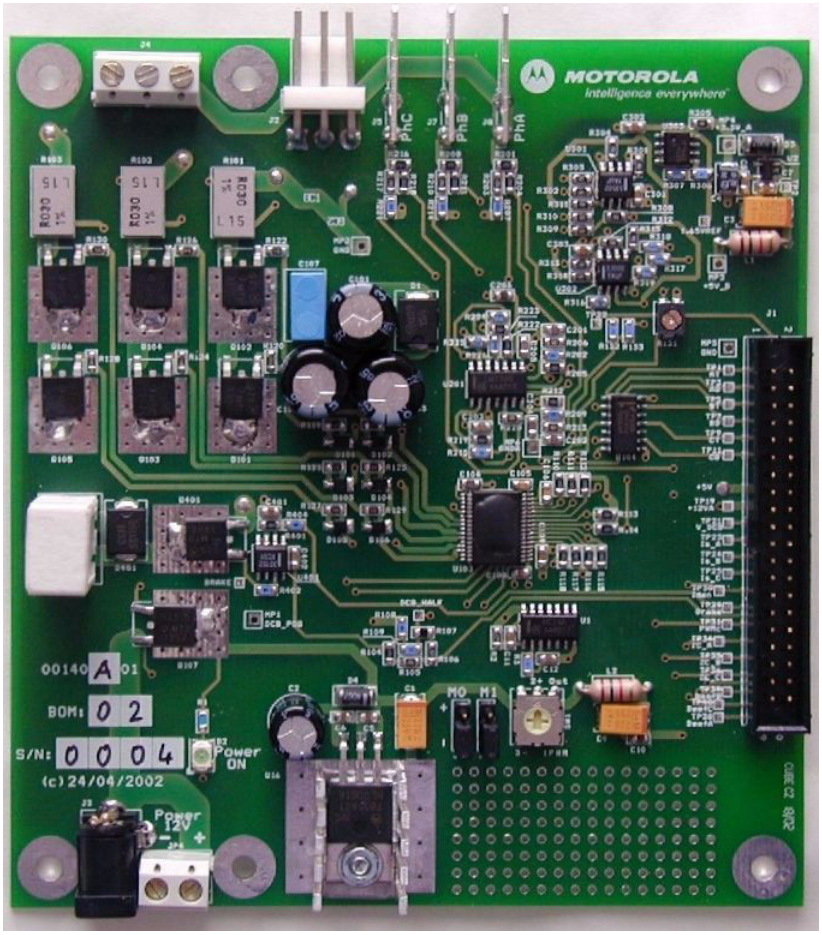

Figure 11. 33395 Evaluation Motor Board

3.3.6 DC Motor with Optical, Hall-like Sensors

The enclosed motor is a low-voltage DC motor. The additional equipment which enables to measure motor speed is attached to this motor (see Figure 12). It deals with the following parts:

- A disk with black and white sectors on its surface. The disk is attached to the motor shaft.
- A group of three photomicrosensors EE-SY313 that are placed on the special position sensor board at the distance of 5 mm from the on-shaft disk.

The described equipment replaces Hall sensors which are widely used for motion system position sensing. Each photomicrosensor generates the signal in dependence on the changing color (black or white) of the on-shaft disk. This way three Hall-like sensor signals are produced.

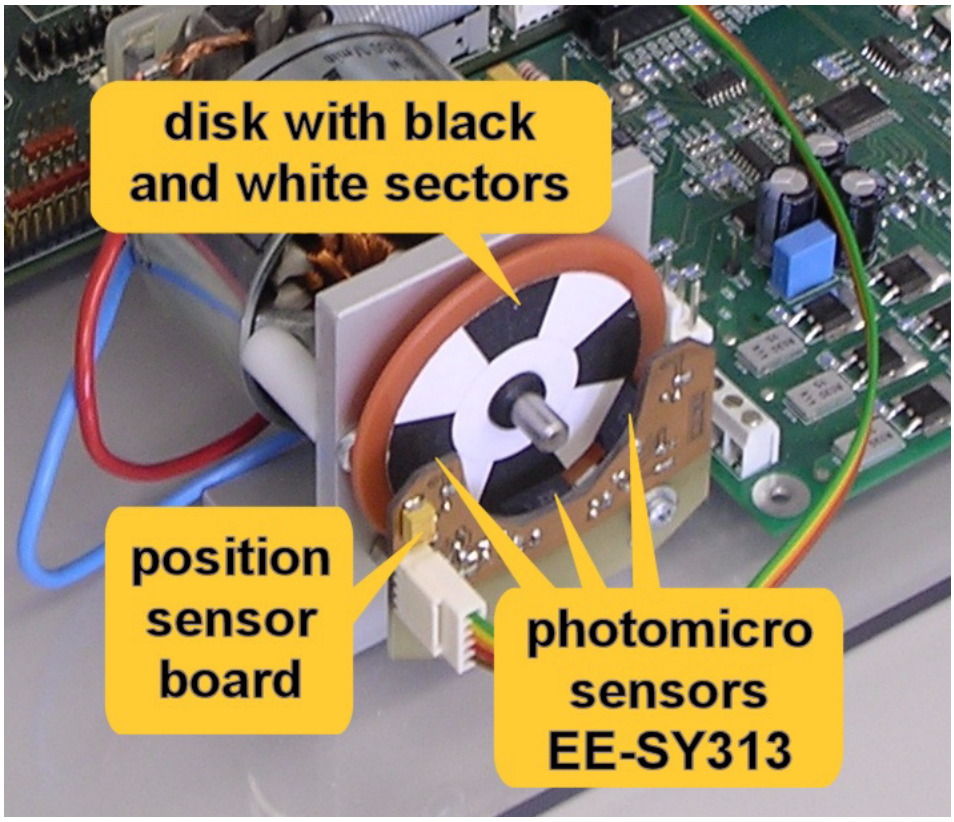

Figure 12. Optical Sensors Used for the Motion System Position Sensing

3.3.7 Power Supply

The power supply 12V/2.7A, is also used to power the 3-Phase Micro Power Stage. The application is scaled for this 12V power supply.

4 Software Design

This section describes the software design of the DC 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.

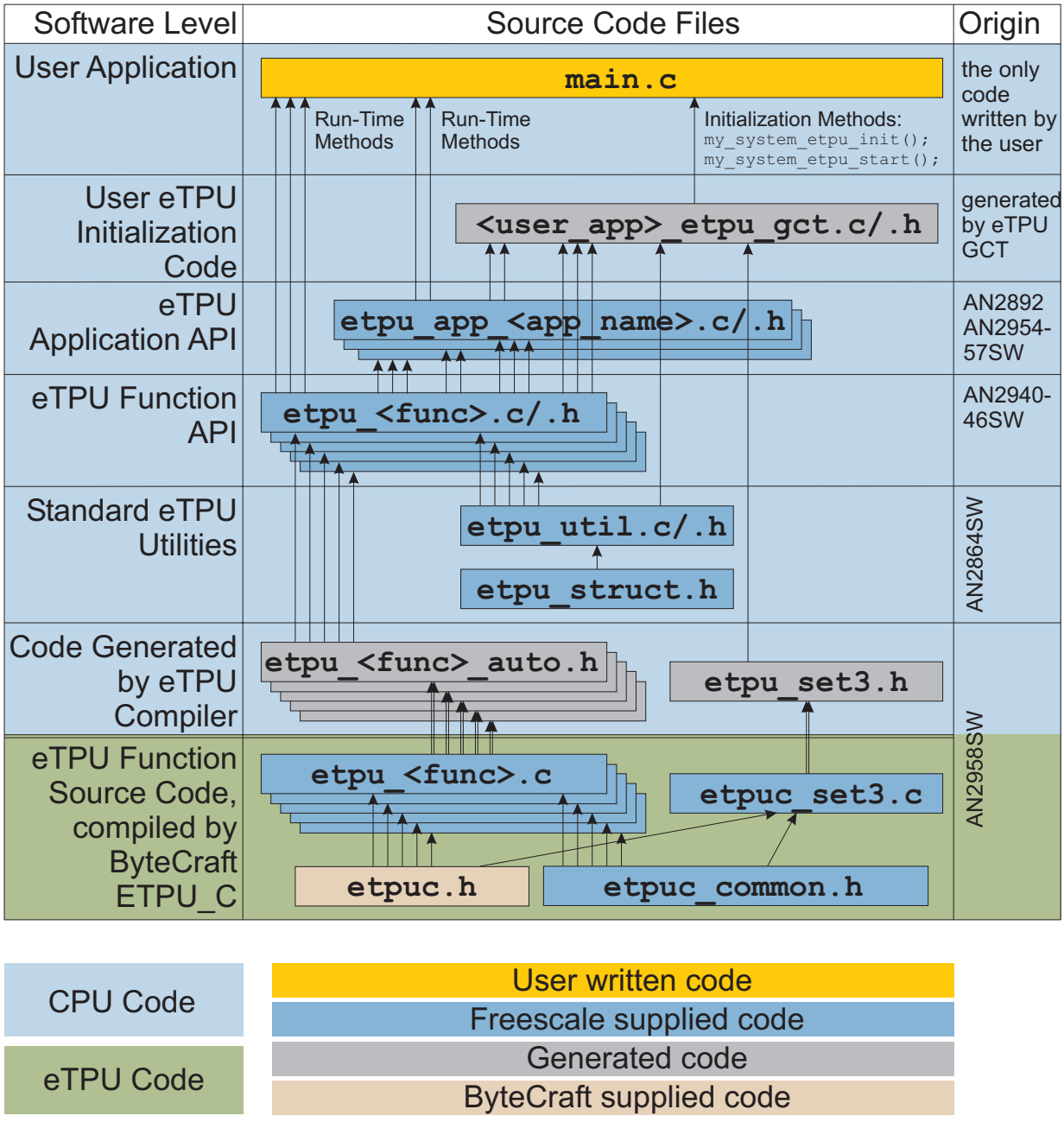

Figure 13. eTPU Project Structure

4.1 CPU Software Flowchart

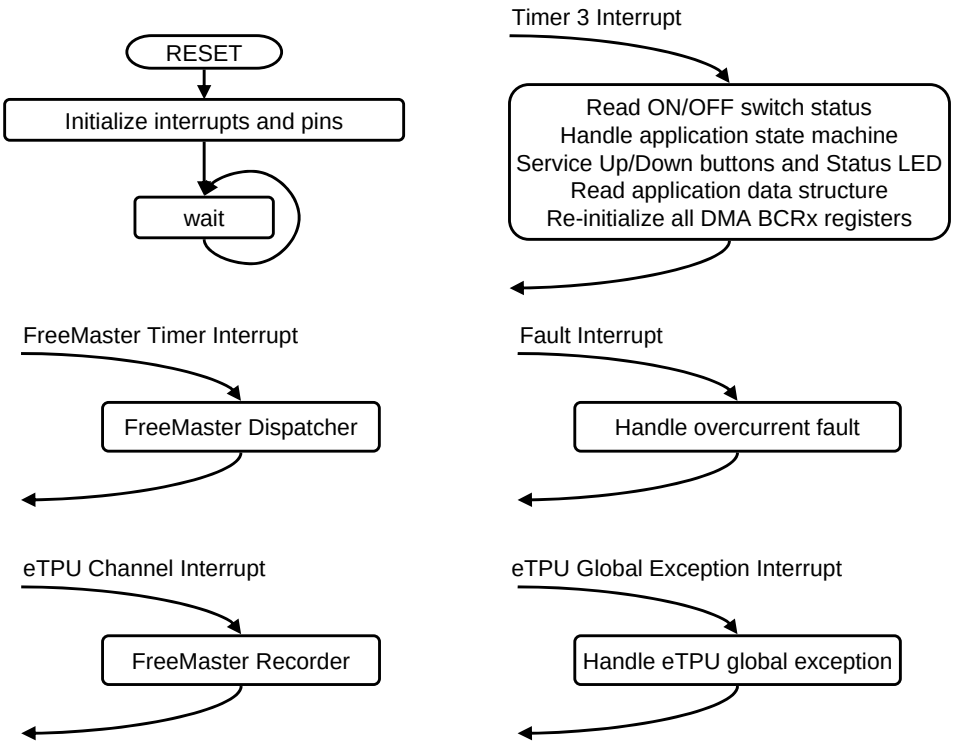

Figure 14. CPU Software Flowchart

After reset, the CPU software initializes peripherals, interrupts, and pins, and starts the eTPU module. At this point, the CPU and the eTPU run in parallel. The following CPU processing is incorporated in two periodic timer interrupts and one fault interrupt.

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_dcmhscl1_get_data` (see 4.3).

4.1.2 FreeMASTER Interrupt Service Routine

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

4.1.3 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.

4.1.4 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

4.1.5 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 15). 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 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.

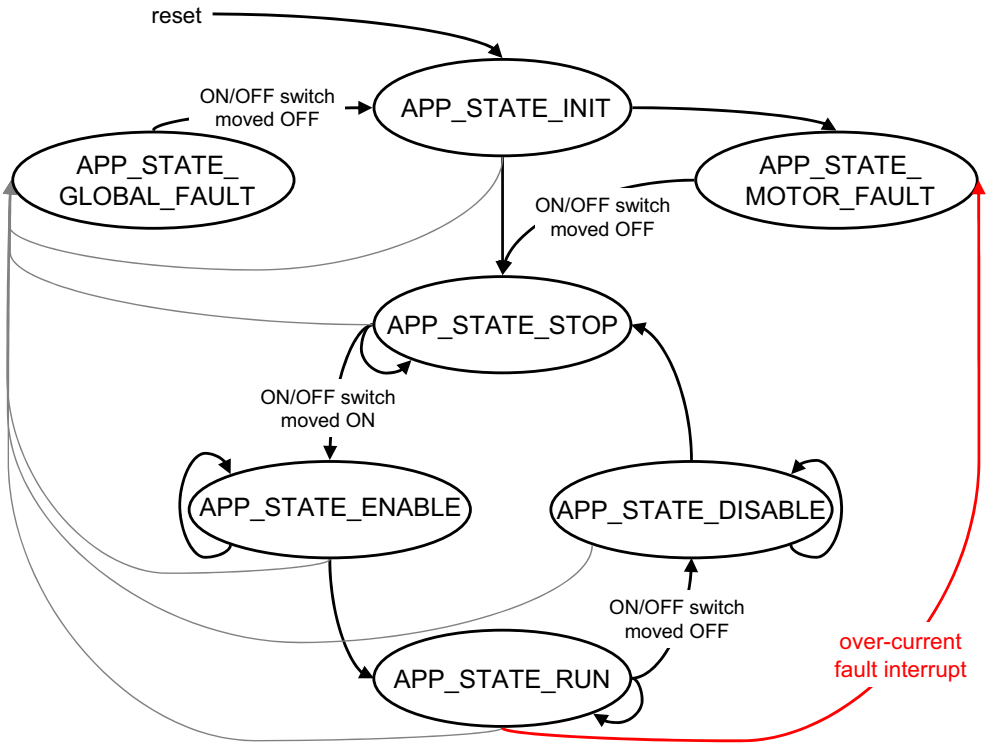

Figure 15. Application State Diagram

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 `dcmhscl1_etpu_gct.c` file, are generated by 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 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 = 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 = 64

  The TCR2 internal eTPU clock is set to a rate of 250 kHz (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 - Hall decoder (HD) - Phase A

  Channel 2 - Hall decoder (HD) - Phase B

  Channel 3 - Hall decoder (HD) - Phase C

  Channel 5 - speed controller (SC)

  Channel 6 - current controller (CC)

  Channel 7 - PWM master for DC motors (PWMMDC)

  Channel 8 - PWM full range (PWMF) - Phase A - base channel

  Channel 10 - PWM full range (PWMF) - Phase B - base channel

Channel 14 - analog sensing for DC motors (ASDC)

These eTPU channels are initialized by the `fs_etpu_app_dcmhscl1_init` eTPU application API function (see 4.3). The application settings are as follows:

- PWM phases-type is full range complementary pairs
- PWM frequency 20kHz
- PWM dead-time 1µs
- Motor speed range 1 400 RPM
- Motor speed minimum 50 RPM
- DC-bus voltage 9V
- Number of motor pole pairs 8
- Motor speed calculated using HD revolution period
- Speed controller update frequency 1250 Hz
- Speed controller PI parameters:
  P-gain is 0.256 (0x0020C4 * 2^{-15}), and
  I-gain is 0.005127 (0x0000A8 * 2^{-15}).
  The controller parameters were experimentally tuned.
- Ramp parameters:
  0.3s to ramp up from zero to the maximum speed,
  0.3s to ramp down from the maximum speed to zero.
- Current controller PI parameters:
  P-gain is 25.6 (0x0CCCCC * 2^{-15}), and
  I-gain is 0 (0x000000 * 2^{-15}).
  The controller parameters were experimentally tuned.
- DC-bus current measurement range is 14.55A
- ASDC function triggers A/D converter on high-low edge
- DC-bus current measurement time including A/D conversion time and DMA transfer time is 11us
- p_ASDC_result_queue pointer contains the address of the measured sample of DC-bus current
- Measured sample of DC-bus current is shifted left by 8 bits
- DC-bus current sample offset within ASDC_result_queue is 0

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.

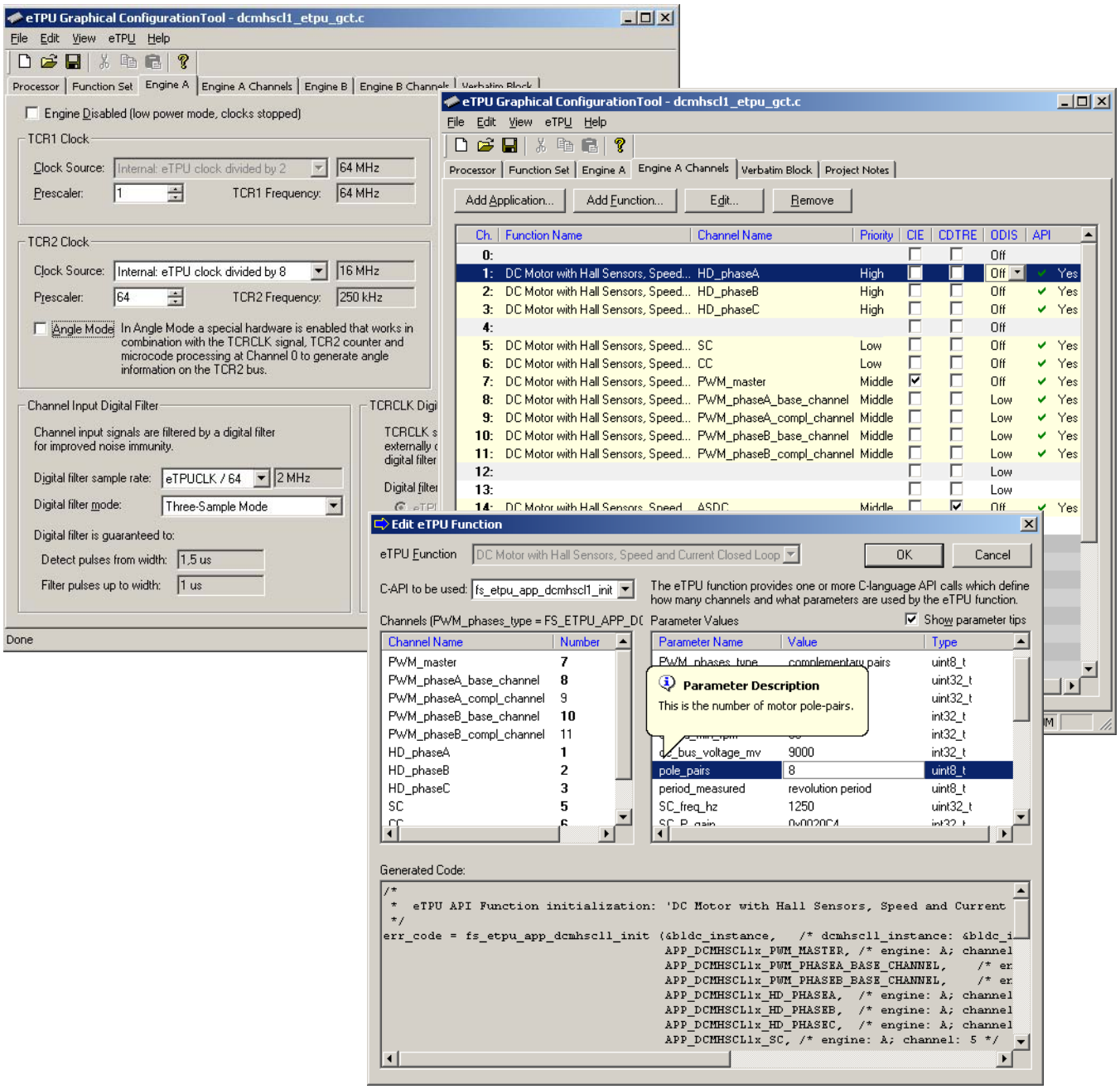

Figure 16. eTPU Configuration Using the eTPU Graphical Configuration Tool

4.2.1.2 Initialization of FreeMASTER Communication

Prior to 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 Hall decoder (HD) eTPU function, to watch variables produced by the HD, and to see optical, Hall-like sensor 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_dcmhscl1_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 DCMHSCL1 is an abbreviation for ‘DC Motor with Hall Sensors, Speed and Current Closed Loops’ eTPU motor - control application. As there are several DC motor applications with Hall sensors, speed and current closed loops, the numeral 1 denotes the first such application in order.

The DCMHSCL1 eTPU application API is described in the following paragraphs. There are 5 basic functions added to the DCMHSCL1 application API. The routines can be found in the `etpu_app_dcmhscl1.c/.h` files. All DCMHSCL1 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_dcmhscl1_init(
  dcmhscl1_instance_t * dcmhscl1_instance,
              uint8_t   PWM_master_channel,
              uint8_t   PWM_phaseA_channel,
              uint8_t   PWM_phaseB_channel,
              uint8_t   HD_phaseA_channel,
              uint8_t   HD_phaseB_channel,
              uint8_t   HD_phaseC_channel,
              uint8_t   SC_channel,
```

```
                    uint8_t   CC_channel,
                    uint8_t   ASDC_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,
                    uint8_t   period_measured,
                    uint32_t  SC_freq_hz,
                    int32_t   SC_P_gain,
                    int32_t   SC_I_gain,
                    uint32_t  SC_ramp_time_ms,
                    int32_t   CC_P_gain,
                    int32_t   CC_I_gain,
                    int32_t   dc_bus_current_range_ma,
                    uint8_t   ASDC_polarity,
                    uint24_t  ASDC_measure_time_us,
                    uint32_t *ASDC_result_queue,
                    uint8_t   ASDC_bit_shift,
                    uint8_t   ASDC_queue_offset,
                    uint32_t  ASDC_filter_time_constant_us)
```

- Change operation functions:

```
int32_t fs_etpu_app_dcmhscl1_enable(
  dcmhscl1_instance_t * dcmhscl1_instance,
              uint8_t   sc_configuration,
              uint8_t   cc_configuration)

int32_t fs_etpu_app_dcmhscl1_disable(
  dcmhscl1_instance_t * dcmhscl1_instance)

void fs_etpu_app_dcmhscl1_set_speed_required(
  dcmhscl1_instance_t * dcmhscl1_instance,
              int32_t   speed_required_rpm)
```

- Value return functions:

```
void fs_etpu_app_dcmhscl1_get_data(
  dcmhscl1_instance_t * dcmhscl1_instance,
      dcmhscl1_data_t * dcmhscl1_data)
```

4.3.1 int32_t fs_etpu_app_dcmhscl1_init(...)

This routine is used to initialize the eTPU channels for the DC motor with speed and current closed loops application. This function has the following parameters:

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_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 dcmhscl1_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_DCMHSCL1_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_DCMHSCL1_COMPL_PAIRS), the complementary channel is one channel higher.
- **HD_phaseA_channel (uint8_t)** - This is the Hall decoder phase A channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **HD_phaseB_channel (uint8_t)** - This is the Hall decoder phase B channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **HD_phaseC_channel (uint8_t)** - This is the Hall decoder phase C 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.
- **CC_channel (uint8_t)** - This is the current controller channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **ASDC_channel (uint8_t)** - This is the analog sensing for DC motors (ASDC) 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_DCMHSCL1_SINGLE_CHANNELS, or FS_ETPU_APP_DCMHSCL1_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.

- **period_measured (uint8_t)** - This option defines the type of period measurement for speed calculation. This parameter should be assigned a value of:
  FS_ETPU_APP_DCMHSCL1_REV_PERIOD, or
  FS_ETPU_APP_DCMHSCL1_SECTOR_PERIOD.
- **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.
- **CC_P_gain (fract24_t)** - This is the current 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
- **CC_I_gain (fract24_t)** - This is the current 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
- **dc_bus_current_range_ma (int32_t)** - This is the DC-bus current measurement range in mA.
- **ASDC_polarity (uint8_t)** - This is the polarity to assign to the ASDC function. This parameter should be assigned a value of: FS_ETPU_APP_DCMHSCL1_ASDC_PULSE_HIGH or FS_ETPU_APP_DCMHSCL1_ASDC_PULSE_LOW.
- **ASDC_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.
- **ASDC_result_queue (uint32_t *)** - Pointer to the result queue. Result queue is an array of 16-bit words that contains the measured values.
- **ASDC_bit_shift (uint8_t)** - This parameter defines how to align data from the result queue into fract24 (or int24). This parameter should be assigned a values of:
  FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_8,

FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_10,
FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_12, or
FS_ETPU_APP_DCMHSCL1_ASDC_SHIFT_LEFT_BY_16.

- **ASDC_queue_offset (uint8_t)** - Position of the I_DC_BUS sample in the result queue. Offset is defined in bytes.
- **ASDC_filter_time_constant_us (uint32_t)** - This the time constant of the filter which applies when processing the I_DC_BUS samples, in us.

4.3.2 int32_t fs_etpu_app_dcmhscl1_enable(...)

This routine is used to enable the generation of PWM signals and to start speed and current controllers. This function has the following parameters:

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.
- **sc_configuration (uint8_t)** - This is the required configuration of the SC. This parameter should be assigned a value of:
  FS_ETPU_APP_DCMHSCL1_SPEED_LOOP_OPENED, or
  FS_ETPU_APP_DCMHSCL1_SPEED_LOOP_CLOSED.
- **cc_configuration (uint8_t)** - This is the required configuration of the CC. This parameter should be assigned a value of:
  FS_ETPU_APP_DCMHSCL1_CURRENT_LOOP_OPENED, or
  FS_ETPU_APP_DCMHSCL1_CURRENT_LOOP_CLOSED.

4.3.3 int32_t fs_etpu_app_dcmhscl1_disable (dcmhscl1_instance_t * dcmhscl1_instance)

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

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.

4.3.4 void fs_etpu_app_dcmhscl1_set_speed_required(...)

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

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.
- **speed_required_rpm (int32_t)** - This is the required motor speed in rpm.

4.3.5 void fs_etpu_app_dcmhscl1_get_data(...)

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

- **dcmhscl1_instance (dcmhscl1_instance_t*)** - This is a pointer to dcmhscl1_instance_t structure, which is filled by `fs_etpu_app_dcmhscl1_init`.
- **dcmhscl1_data (dcmhscl1_data_t*)** - This is a pointer to dcmhscl1_data_t structure of application state data, which is updated.

4.4 eTPU Block Diagram

The eTPU functions used to drive the DC motor with speed and current closed loops 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.

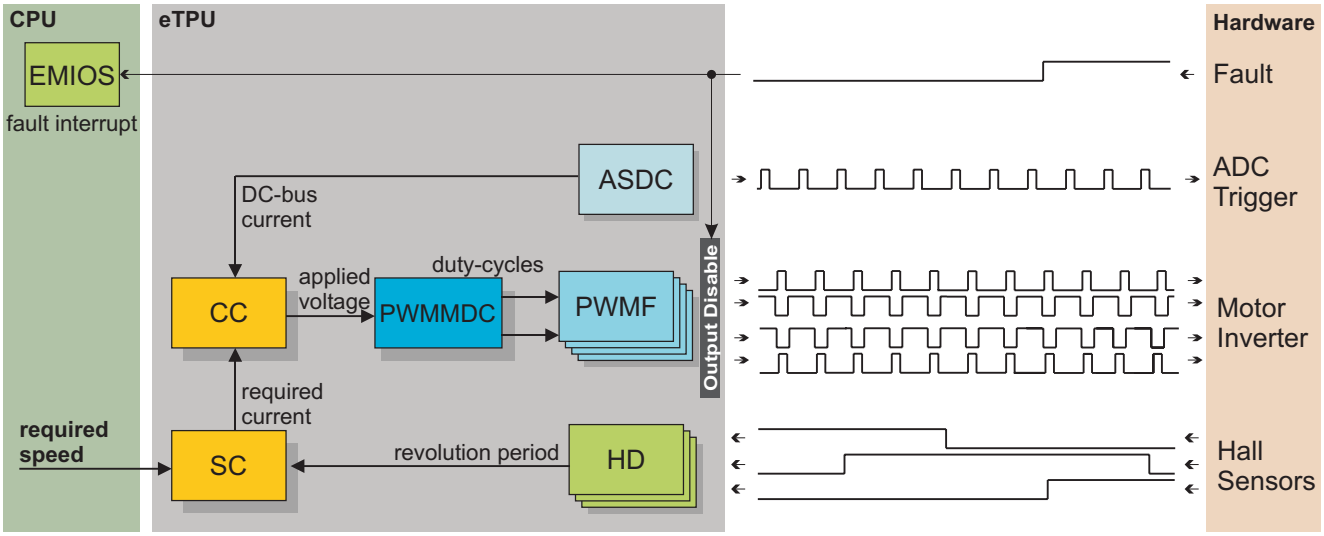

Figure 17. Block Diagram of eTPU Processing

4.4.1 PWM Generator (PWMMDC+PWMF)

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 PWMF function is used in the described application.

The PWMF function generates the PWM signals. The PWMMDC function controls two PWMF functions, two 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.

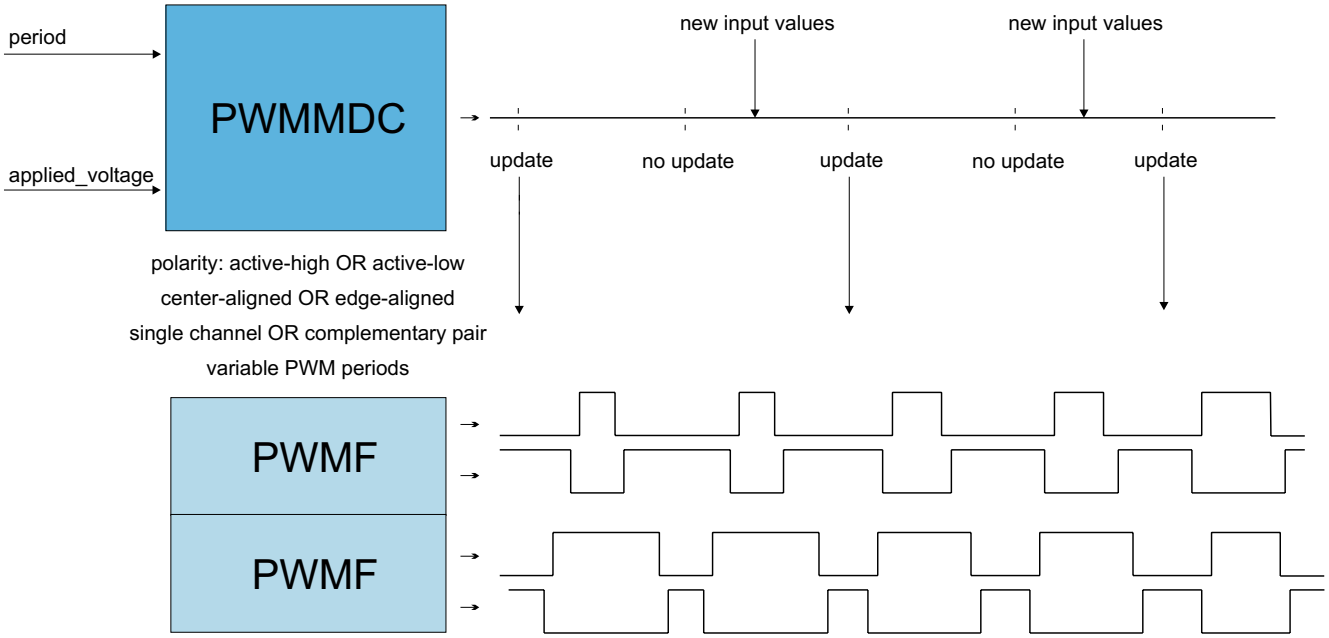

Figure 18. Functionality of PWMMDC+PWMF

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

4.4.2 Hall Decoder (HD)

The Hall decoder eTPU function is intended to process signals generated by Hall sensors in motion control systems. The HD function uses three adjacent eTPU channels configured as inputs. The HD function calculates the following parameters for the CPU:

- Sector - determines the position of the motion system in one of the sectors.
- Direction - determines the direction of the motion system. A direction value 0 means a positive (incremental) direction, other values mean a negative (decremental) direction.
- Revolution counter - determines the number of motion system electrical revolutions. The revolution counter is incremented or decremented on each revolution, based on the current direction.
- Revolution period - determines the TCR time of the last revolution. The parameter value is updated each time the sector is changed. The revolution period is measured from the last edge of a similar type (low-high / high-low), on the same channel, to the current edge.
- Sector period - determines the TCR time between the last two changes of the sector. The parameter value is updated each time the sector is changed. The sector period is measured from the last edge to the current edge.
- Last edge time - stores the TCR time of the last incoming edge.

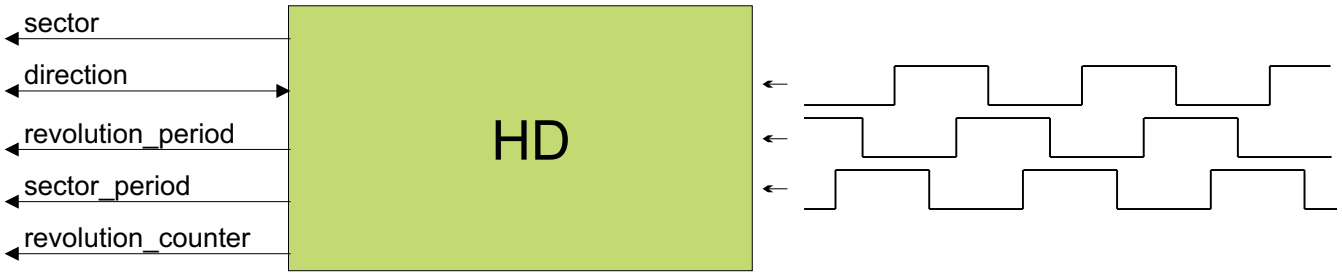

Figure 19. Functionality of HD

For more details about the HD eTPU function, refer to Reference 10.

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.

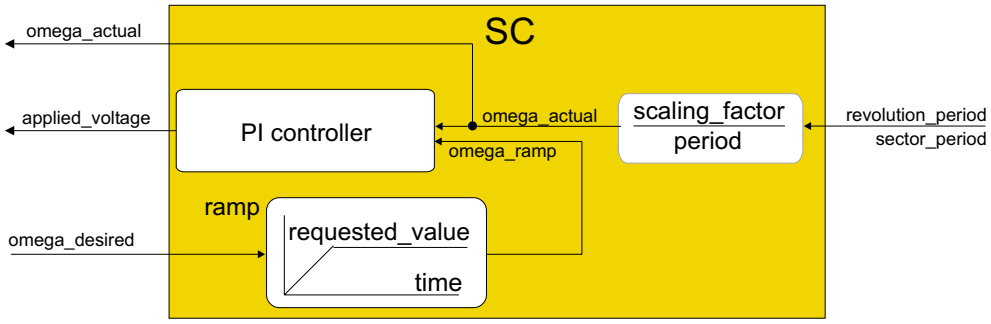

Figure 20. Functionality of SC

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

4.4.4 Current Controller (CC)

The current controller eTPU function is not intended to process input or output signals. Its purpose is to control another eTPU function's input parameter. The CC function can be executed even on an eTPU channel not connected to an output pin. The CC function includes a general PID controller algorithm. The controller calculates its output based on two inputs: a measured value, and a desired value. The measured value (the actual DC-bus current) is usually provided by the analog sensing for DC motors (ASDC) function, that preprocesses the measured analog values. The desired value is a CC 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 current closed loop.

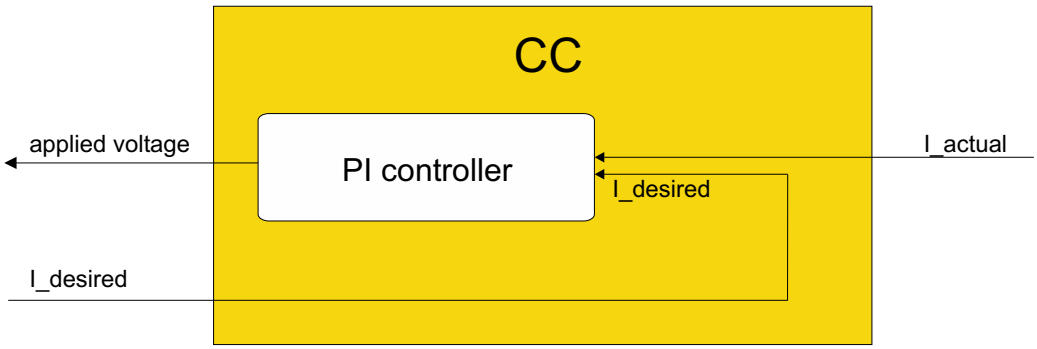

Figure 21. Functionality of CC

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

4.4.5 Analog Sensing for DC Motors (ASDC)

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

All the above mentioned ASDC features are utilized in the application. The ASDC is initialized to run in PWM synchronized mode, e.g. the first ASDC edge is synchronized with the beginning of the PWM period. Simultaneously, the ASDC manages to synchronize the SC function (outer loop controller) by generating the link to the SC channel every 16th ASDC period and to synchronize the CC function (inter loop controller) by generating the link to the CC channel every ASDC period.

The ASDC function preprocesses the DC-bus current analog value and passes the adjusted value as an input to the CC function. Processing of the DC-bus current analog value includes bit shifting and filtering.

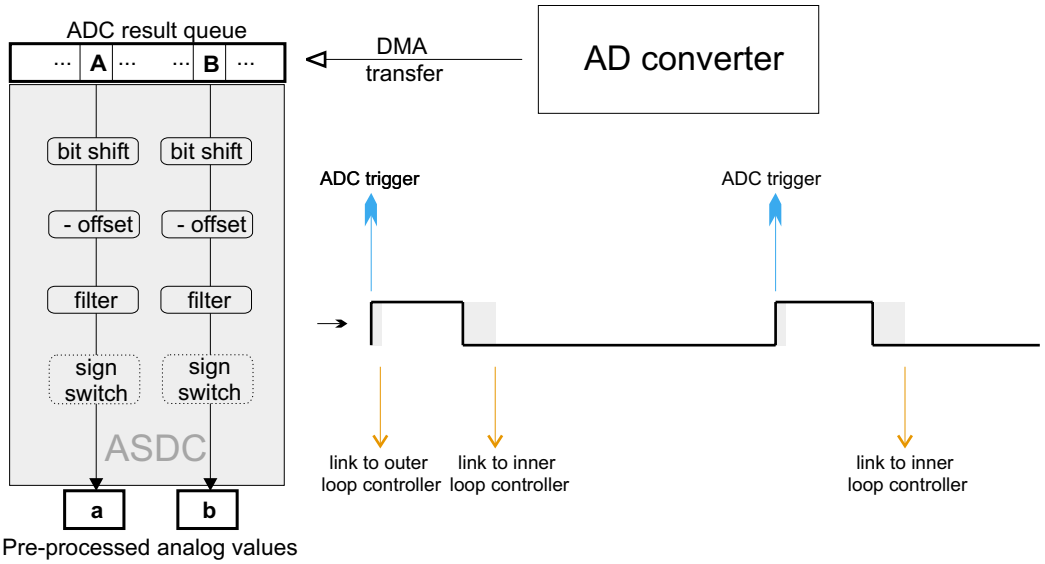

Figure 22. Functionality of ASDC

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

- An internal analog to digital converter is used for sampling of the DC-bus current analog value.
- 3 direct memory access (DMA) channels are used as follows:
  - DMA channel 0 is used for the transfer of the configuration word (0x80000000) from eqadcCQueue0 to the CFPR (CFIFO Push Register) of the eQADC module. This operation triggers the eQADC converter sampling. The DMA channel 0 transfer is initiated by a DMA request generated by DMA channel 30.
  - DMA channel 1 is used for the transfer of the result from the RFPR (result FIFO pop register) to the eTPU memory where points p_ASDC_result_queue pointer. The DMA channel 30 transfer is initiated by a DMA request generated by QADC module after the conversion is finished.
  - DMA channel 30 is used for the transfer of the configuration word (0x0410) to the CFCR (command FIFO control register). This set the start bit and the QADC single scan mode. The DMA channel 30 transfer is initiated by a DMA request generated by eTPU_A channel 14. When the transfer is complete, the link to DMA channel 0 is made

All setting is made using the Quick Start Configuration Tool, refer to Reference 3.

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.

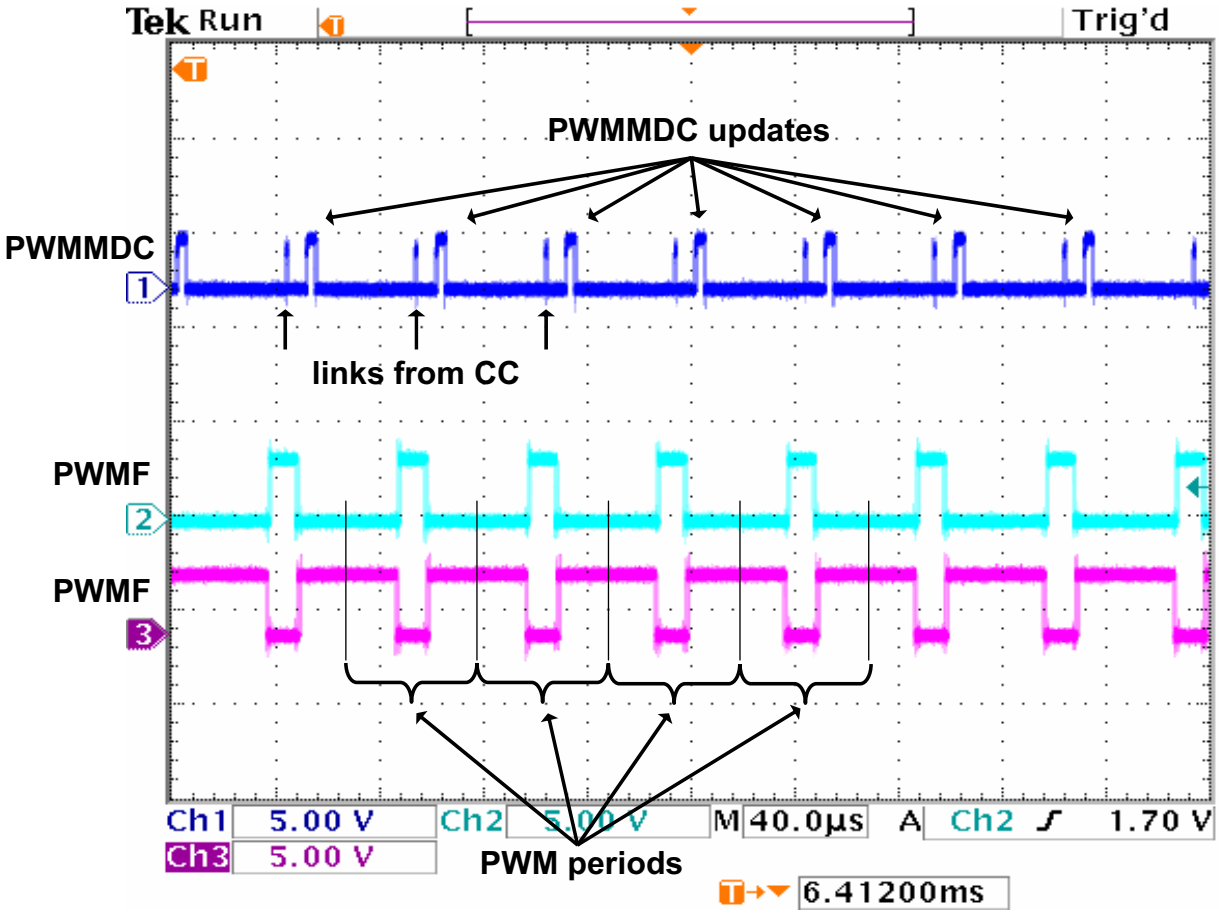

Figure 23. Oscilloscope Screenshot and Explanation of PWMMDC Timing

Figure 23 and Figure 24 explain the application eTPU timing. The oscilloscope screen-shots depict a typical situation described below. A live view on the oscilloscope screen enables the user to see the variation of SC, CC and PWMMDC activity pulses, which determines the channel service latency ranges.

In Figure 23, signals 2 (cyan) and 3 (pink) are PWM signals of one phase. It is a complementary pair of centre-aligned PWM signals. The base channel (2) is of active-high polarity, while the complementary channel (3) is active-low. The PWM period is 50µs, which corresponds to a PWM frequency of 20kHz.

Signal 2 (cyan) is generated by the PWM master for DC motors (PWMMDC) eTPU function. Its pulses determine the activity of the PWMMDC. The narrow PWMMDC pulses occur after each CC activity and they determine the service time of an CC request to update the new value of applied motor voltage. The wide pulses occur each PWM period and they determine the PWM update. A new value of applied motor voltage is processed during the PWM update.

The `fs_etpu_pwmmdc_init_3ph` function parameter update_time enables the user to adjust the position of the PWM update 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.

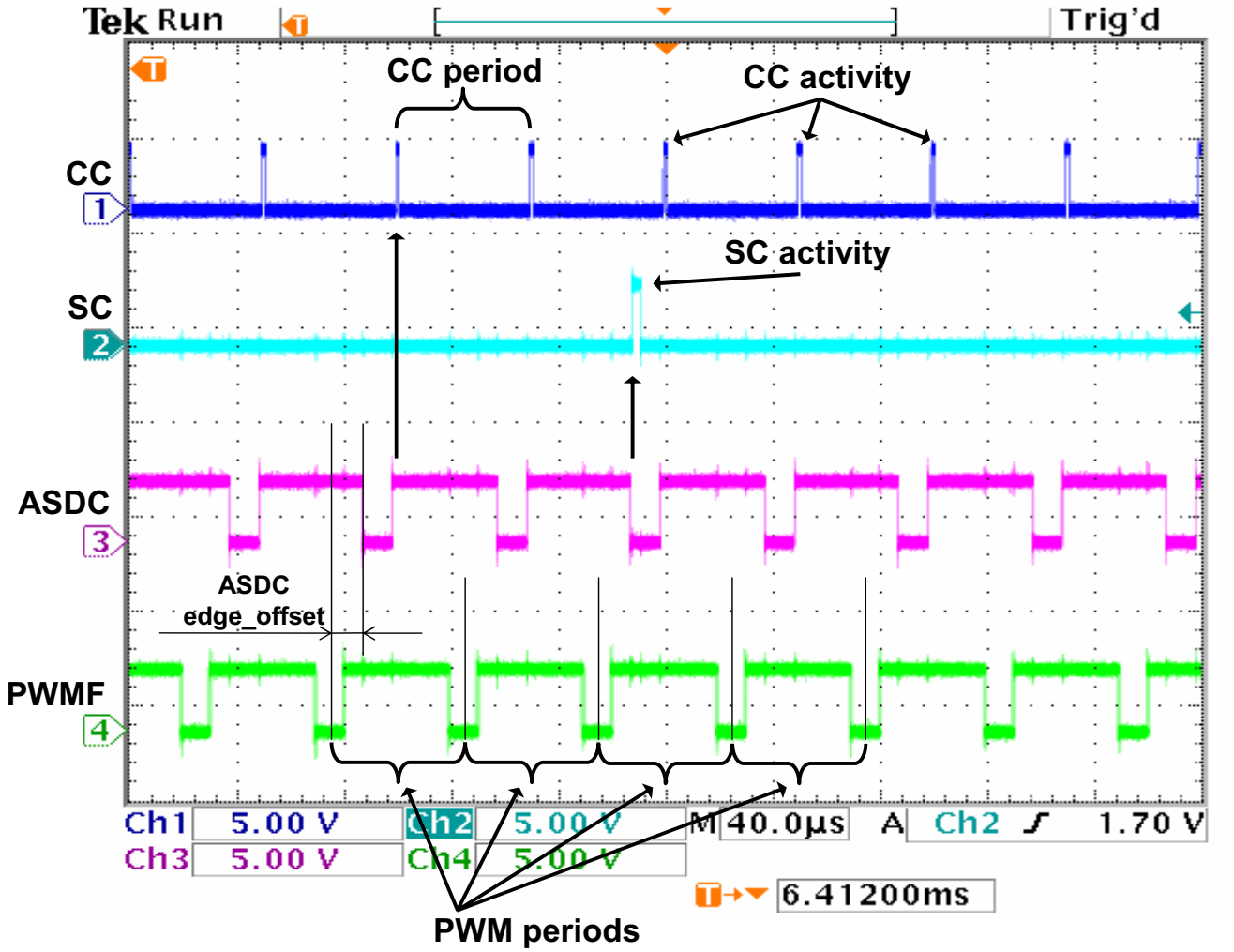

Figure 24. Oscilloscope Screenshot and Explanation of ASDC, SC, and CC Timing

Figure 24 explains the timing of ASDC, SC, and CC eTPU functions. Signal 4 (green) is the PWM signal of one base channel, as in Figure 23. Signal 3 (pink) is generated by the ASDC eTPU function. The ASDC function triggers the AD converter by generating a DMA request on a high-low edge (active low polarity of ASDC) and simultaneously sending the link to the SC channel every 16th ASDC period (see Figure 24). The position of the ASDC first edge is synchronized with the beginning of the PWM period. The time between the PWM period beginning and the ASDC first edge equals to one-quarter of the PWM period. The ASDC pulse width determines the time necessary to sample the DC-bus current and to transfer this sampled value to the eTPU data memory. ASDC starts measured sample preprocessing at the time of the second edge when a sample is supposed to be ready in the eTPU data memory. Immediately after the DC-bus current sample preprocessing, ASDC function sends the link to the CC channel to start PI controller calculation. This way the CC synchronization is ensured.

Signal 2 (cyan) 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 a revolution

period measured by the Hall decoder (HD), calculate the required speed ramp, and apply the SC-PI controller algorithm. The SC output - the new value of desired current - is passed to the current controller. This calculation is performed periodically at a 1250Hz rate, which is every 16th PWM period.

Signal 1 (blue) is generated by the current controller (CC) eTPU function. Its pulses determine the activity of the CC. The pulse width determines the time necessary to apply the CC-PI controller algorithm. The CC output - the new value of applied motor voltage - is passed to the PWM generator. This calculation is performed periodically at a 20kHz rate, which is every PWM period.

The signals from the optical sensors come asynchronously with the PWM periods. The Hall decoder (HD) eTPU function processes these signals transitions and provides the calculated data to the speed controller eTPU function.

5 Implementation Notes

5.1 Scaling of Quantities

The DC 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.

6 Microprocessor Usage

Table 2 shows how much memory is needed to run the application.

Table 2. Memory Usage in Bytes

Memory	Available	Used
FLASH	2M	33 116
RAM	64K	3 460
eTPU code RAM	16K	6 120
eTPU data RAM	3K	632

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 According to Reference 11, the speed controller calculation takes 232 eTPU cycles. The calculation is performed every 16th PWM period.
- According to Reference 10, the processing of one Hall signal transition takes 56 eTPU cycles. The Hall signal transitions come asynchronously to the PWM periods. Six transitions are processed per one electrical motor revolution.

- According to Reference 14, the ASDC maximum eTPU load takes 42 + 80 eTPU cycles (both the first and then the second edge processing is performed). The ASDC function processing is executed every PWM period.
- According to Reference 13, the current controller calculation takes 138 eTPU cycles. The calculation is performed every PWM period.
- 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 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 the speed controller calculation, the current controller calculation, ASDC processing, and a Hall 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 3 shows the eTPU module time load in several typical situations.

Table 3. eTPU Time Load

Situation	Average Time Load [%]	Peak Time Load Within PWM Period [%]
Motor Speed 100 RPM	30.10	37.39
Motor Speed 1000 RPM	30.12	37.39

7 Summary and Conclusions

This application note provides the user with a description of the demo application DC motor with speed and current 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 4. References

1. MPC5554 Reference Manual, MPC5554RM
2. MPC5554DEMO User's Manual, MPC5554DEMO EVBUM
3. MPC5550 Quick Start User's Manual
4. Interface Board with UNI-3 User's Manual
5. 33395 Evaluation Motor Board Designer Reference Manual DRM33395/D
6. MCG's Motors web: http://www.mcg-net.com
7. Quadrature Encoder HEDS-5640 A06 distributor's web: http://www.agilent.com/semiconductors
8. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
9. Enhanced Time Processing Unit Reference Manual, ETPURM
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 Current Controller (CC) eTPU Function," AN2844
14. "Using the Analog Sensing for DC Motors (ASDC) eTPU Function," AN2846
15. "Using the DC Motor Control eTPU Function Set (set3)," AN2958
16. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT"
17. DSP56F80x MC PWM Module in Motor Control Applications, AN1927

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

AN3008
Rev. 1
05/2006