Freescale Semiconductor
Application Note

AN3205
Rev. 0, 06/2006

AC Induction Motor Volts per Hertz Control with Speed Closed Loop, Driven by eTPU on MPC5500

Covers MPC5500 and all eTPU-Equipped Devices

by: Milan Brejl, Michal Princ and Petr Uhlir
System Application Engineers
Roznov Czech System Center

This application note describes the design of a 3-phase AC induction motor (ACIM) speed closed loop Volts per Hertz control drive based on Freescale's PowerPC MPC5500 microprocessor. The application design takes advantage of the enhanced time processing unit (eTPU) module, which is used as a motor control co-processor. The eTPU completely handles the motor control processing, eliminating the microprocessor overhead for other duties.

The application is designed for driving medium power three-phase AC induction motors and is targeted for applications in both industrial and appliance fields (e.g. washing machines, compressors, air conditioning units, pumps, or simple industrial drives).

It serves as an example of an AC 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 AC motor control eTPU function set.

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

Table of Contents

freescale™ semiconductor

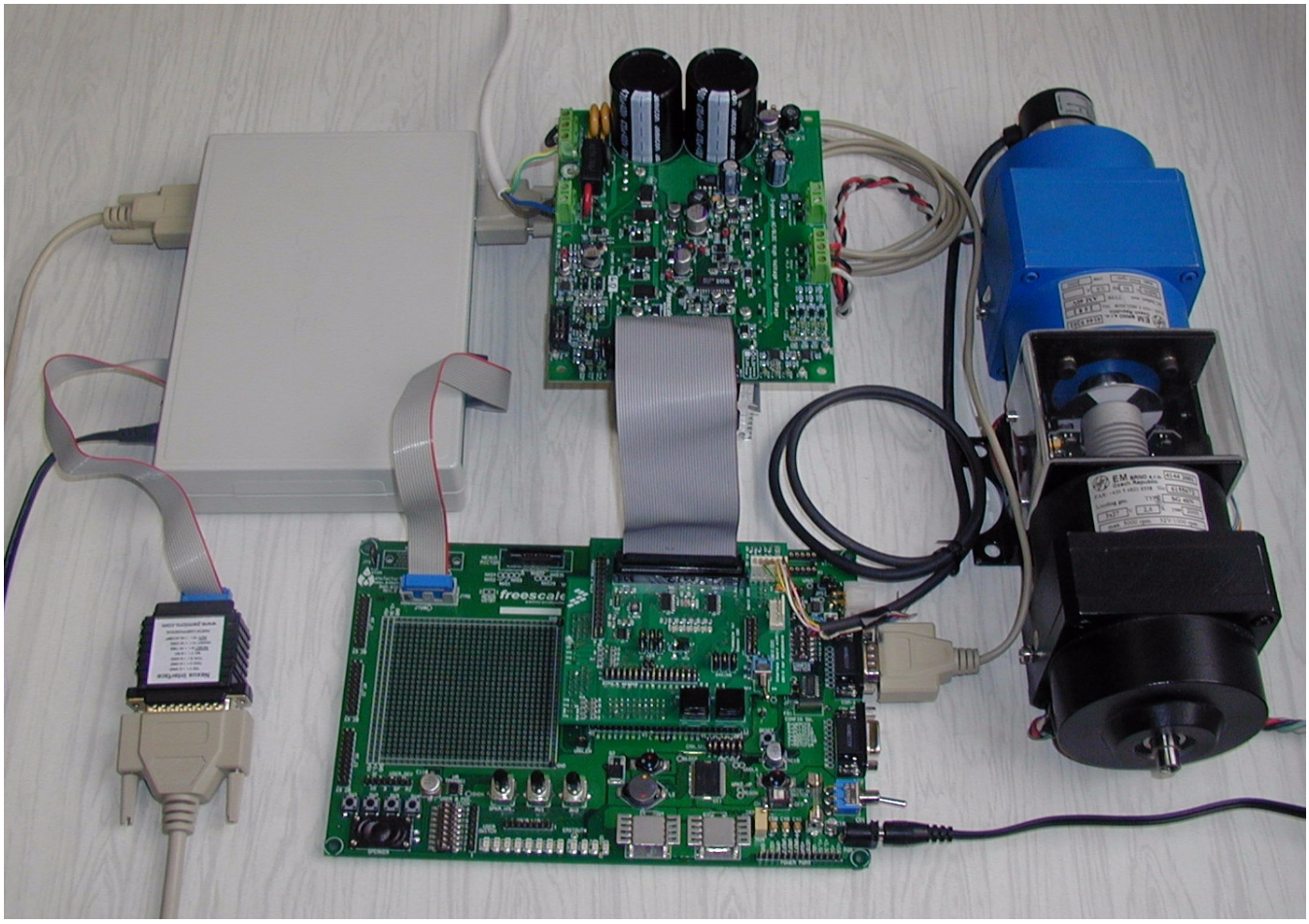


Figure 1. Using MPC5554DEMO, 3-Phase AC/BLDC High-Voltage Power Stage, Inline Optoisolation Box, and 3-Phase AC Induction Motor

1 PowerPC MPC5554 and eTPU Advantages and Features

1.1 PowerPC MPC5554 Microcontroller

The MPC5554 microcontroller belongs to a family of next generation powertrain microcontrollers based on the PowerPC Book E architecture. Featuring two 32-channel 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 has been designed for applications that require complex, real-time control.

This 32-bit device is based on the PowerPC core operating at a 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)
- 2 MB 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 enhanced direct memory access (eDMA) controller
- Interrupt controller (INTC) capable of handling 286 selectable-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 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

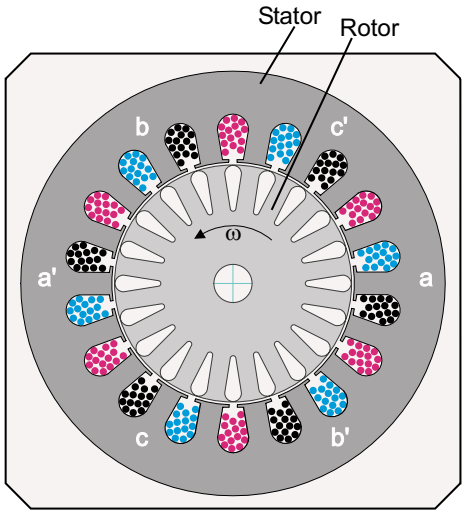


Figure 2. 3-Phase AC Induction Motor

The AC induction motor is a rotating electric machine designed to operate from a 3-phase source of alternating voltage. For variable speed drives, the source is normally an inverter that uses power switches to produce approximately sinusoidal voltages and currents of controllable magnitude and frequency.

A cross-section of a two-pole induction motor is shown in Figure 2. Slots in the inner periphery of the stator accommodate 3-phase winding a,b,c. The turns in each winding are distributed so that a current in a stator winding produces an approximately sinusoidally-distributed flux density around the periphery of the air gap. When three currents that are sinusoidally varying in time, but displaced in phase by 120° from each other, flow through the three symmetrically-placed windings, a radially-directed air gap flux density is produced that is also sinusoidally distributed around the gap and rotates at an angular velocity equal to the angular frequency, ω_s, of the stator currents.

The most common type of induction motor has a squirrel cage rotor in which aluminum conductors or bars are cast into slots in the outer periphery of the rotor. These conductors or bars are shorted together at both ends of the rotor by cast aluminum end rings, which also can be shaped to act as fans. In larger induction motors, copper or copper-alloy bars are used to fabricate the rotor cage winding.

As the sinusoidally-distributed flux density wave produced by the stator magnetizing currents sweeps past the rotor conductors, it generates a voltage in them. The result is a sinusoidally-distributed set of currents in the short-circuited rotor bars. Because of the low resistance of these shorted bars, only a small relative angular velocity (ω_r) between the angular velocity (ω_s) of the flux wave and the mechanical angular velocity (ω) of the two-pole rotor is required to produce the necessary rotor current. The relative angular velocity is called the slip velocity. The interaction of the sinusoidally-distributed air gap flux density and induced rotor currents produces a torque on the rotor. The typical induction motor speed-torque characteristic is shown in Figure 3.

Squirrel-cage AC induction motors are popular for their simple construction, low cost per horsepower and low maintenance (they contain no brushes, unlike DC motors). They are available in a wide range of power ratings. With field-oriented vector control methods, AC induction motors can fully replace standard DC motors, even in high-performance applications.

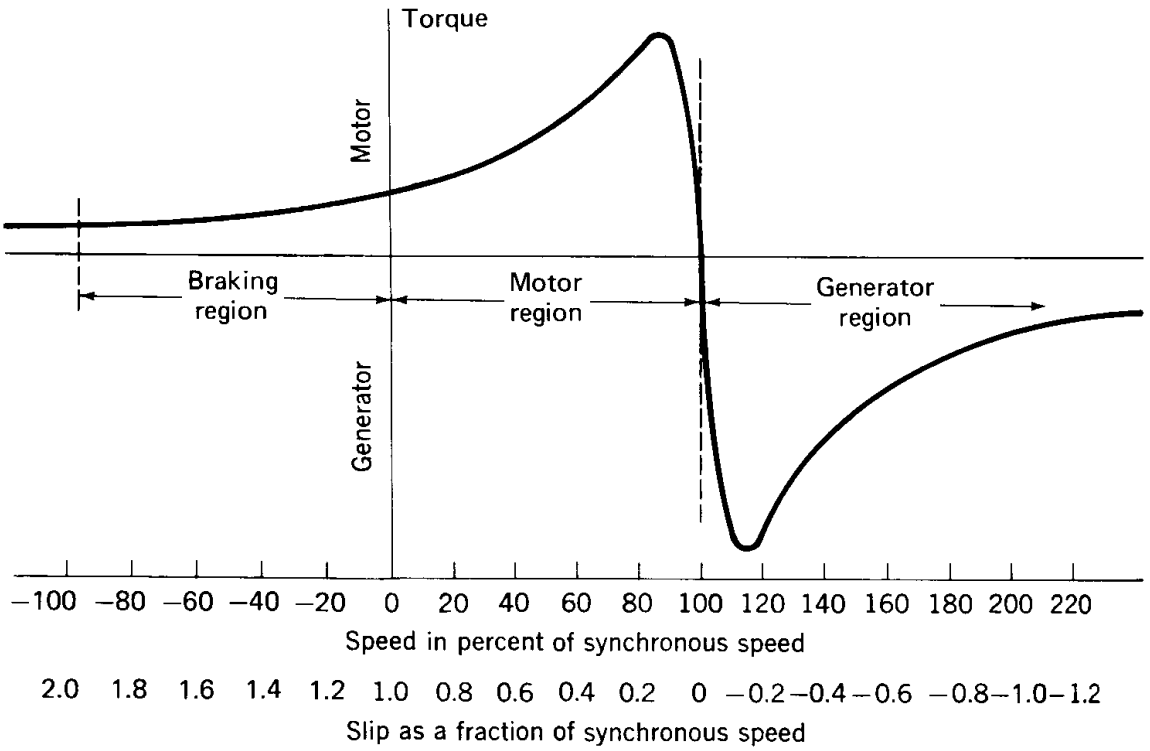


Figure 3. Speed-Torque Characteristic of an AC Induction Motor

2.1 Digital Control of an AC Induction Motor

In adjustable speed applications, AC motors are powered by inverters. The inverter converts DC power to AC power at the required frequency and amplitude. Figure 4 illustrates a typical 3-phase inverter.

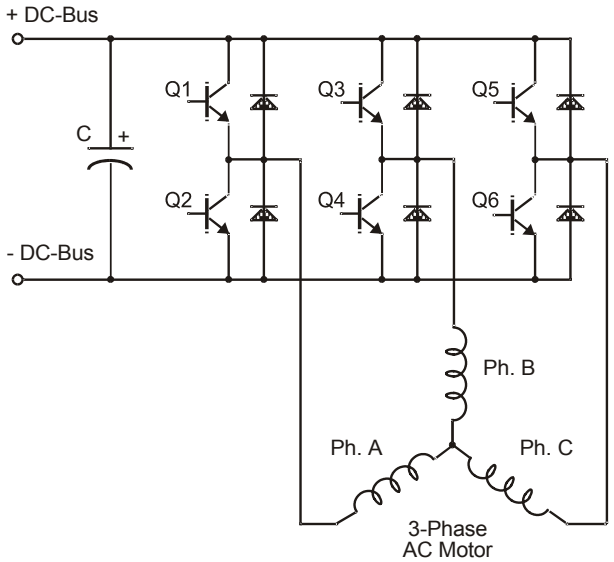


Figure 4. 3-Phase Power Stage

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

2.2 Volts per Hertz Control

The Volt pet Hertz control methods is the most popular method of scalar control that controls the magnitude of the variable like frequency, voltage, or current. The command and feedback signals are DC quantities that are proportional to the respective variables.

This scheme is defined as a Volts per Hertz control because the voltage applied command is calculated directly from the applied frequency to maintain the air-gap flux of the machine constant. In steady state operation, the machine air-gap flux is approximately related to the ratio V_s/f_s, where V_s is the amplitude of motor phase voltage and f_s is the synchronous electrical frequency applied to the motor. The control system is illustrated in Figure 5. The characteristic is defined by the base point of the motor. Below the base point, the motor operates at optimum excitation because of the constant V_s/f_s ratio. Above this point the motor operates under-excited because of the DC-bus voltage limit.

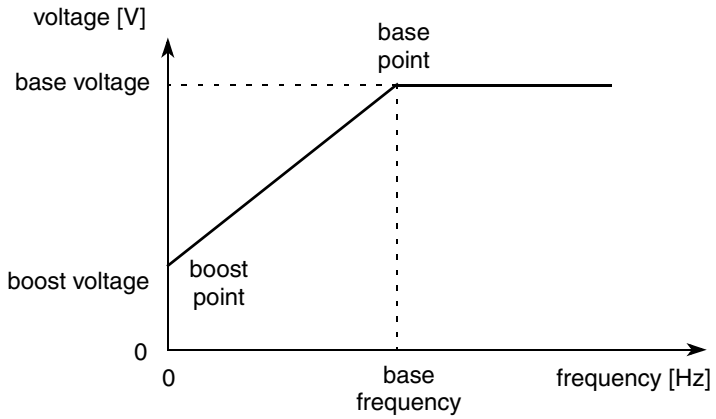


Figure 5. Volts per Hertz Control Method

2.3 Quadrature Encoder

The ACIM 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 6.

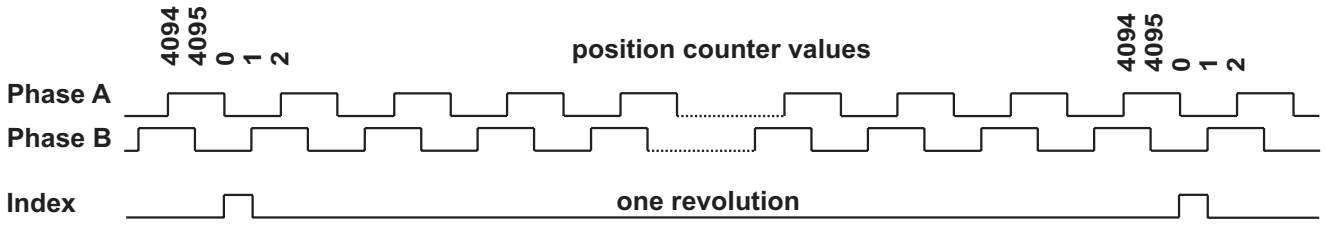


Figure 6. Quadrature Encoder Output Signals

3 System Concept

3.1 System Outline

The system is designed to drive a 3-phase AC induction motor in speed closed loop. The application meets the following performance specifications:

- Speed closed loop scalar control of a three phase general purpose induction motor
- Targeted at PowerPC MPC5554DEMO evaluation board (MPC554DEMO)
- Control technique incorporates:
  - Volts per Hertz control
  - Both directions of rotation
  - 4-quadrant operation
  - Maximal speed 3000 RPM at input power line 230 V AC
- FreeMASTER control interface (speed set-up, start/stop switch)
- FreeMASTER monitor
  - FreeMASTER graphical control page (required speed, actual motor speed, start/stop status, fault status)
  - FreeMASTER control scope (observes required 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 break
- DC bus over-current fault protection

3.2 Application Description

A standard system concept is chosen for the motor control function (see Figure 7). The system incorporates the following hardware:

- Evaluation board MPC5554DEMO
- Interface board with UNI-3
- 3-phase AC/BLDC high-voltage power stage

- Inline optoisolation box
- 3-phase AC induction motor Type AM40V, EM Brno s.r.o., Czech Republic
- Load type SG 40N, EM Brno s.r.o, Czech Republic
- Quadrature encoder BHK 16.05A 1024-I2-5, Baumer Electric, Switzerland

The eTPU module runs the main control algorithm. The 3-phase PWM output signals for a 3-phase inverter are generated, using Volts per Hertz control algorithm, according to feedback signals from quadrature encoder, and the input variable values, provided by the microprocessor CPU. The DC-bus voltage is sampled by on-chip analog-to-digital converter (eQADC module). The transfer of measured samples from eQADC to eTPU DATA RAM is provided by the eDMA module.

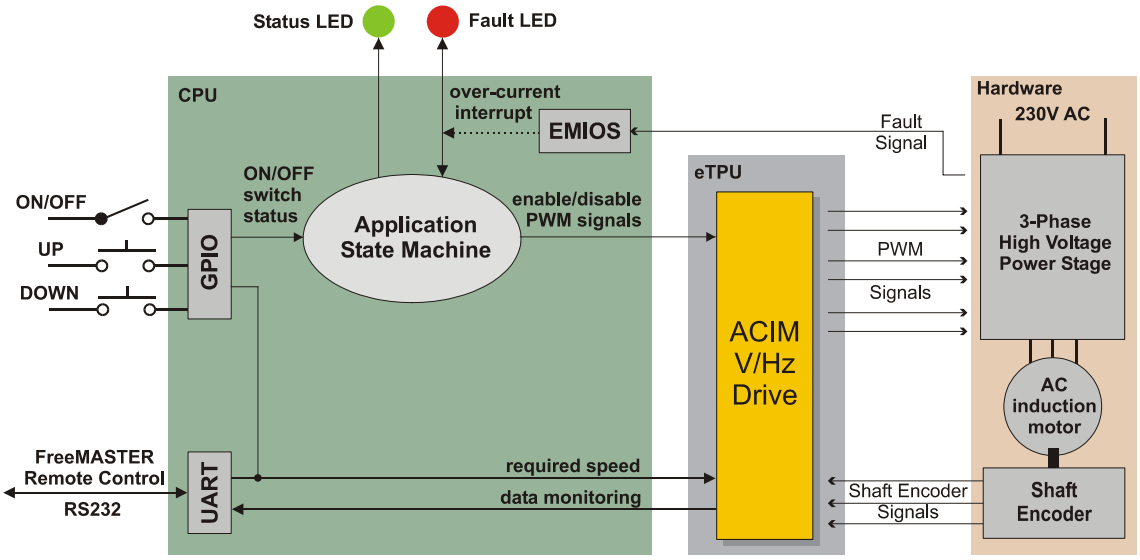


Figure 7. System Concept

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

- Periodically scans FreeMASTER interface. Based on the user input, it handles the application state machine and calculates the required speed, 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.
- Periodically, with a period of 50us, the eDMA transfers eQADC conversion command from eTPU data RAM into eQADC and the results of conversion is transferred from eQADC into eTPU data RAM.

The eTPU performs these tasks:

- Six eTPU channels (PWMF) generate PWM output signals.
- Three eTPU channels (QD) process quadrature encoder signals.

- One eTPU channel (BC) controls the DC-bus break.
- One eTPU channel (ASAC) triggers the on-chip AD convertor and preprocess the sampled values.
- One eTPU channel (PWMMAC) internally synchronizes the PWM outputs and calculate duty-cycles for individual phases from (alpha/beta) or (angle/amplitude) reference frames.
- One eTPU channel (SC) internally calculates the actual motor speed and controls a speed closed loop. The actual motor speed is calculated based on the QD position counter and QD last edge time. The required speed is provided by the CPU and passed through a ramp. The speed PI control algorithm processes the error between the required and actual speed. The PI controller output is passed to the ACIMVHZ eTPU function as a newly corrected value of the required motor speed.
- One eTPU channel (ACIMVHZ) internally performs simple V/Hz control. The ACIMVHZ calculates applied voltage vector components based on required speed (omega) and defined V/Hz ramp. The ACIMVHZ outputs, either alpha and beta vector components or angle and amplitude are passed to the PWM generator.

3.2.1 User Interface

The application is interfaced by the FreeMASTER running on a PC connected to the MPC5554DEMO via an RS232 serial cable. The on/off switch on the FreeMASTER control page 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 by the speed gauge on the FreeMASTER control page. The FreeMASTER also displays real-time values of application variables and their time behavior using scopes.

FreeMASTER software was designed to provide an application-debugging, diagnostic, and demonstration tool for the development of algorithms and applications. It runs on a PC connected to the 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 PowerPC MPC5554 microprocessor using:

- MPC5554DEMO evaluation board
- Interface board with UNI-3
- 3-phase AC/BLDC high-voltage power stage
- Inline optoisolation box, which is connected between host computer and the controller board
- 3-phase AC induction motor Type AM40V, EM Brno s.r.o., Czech Republic

- Quadrature encoder BHK 16.05A 1024-I2-5, Baumer Electric, Switzerland
- Power supply for MPC5554DEMO evaluation board, 9-12V DC, minimum 2.7 Amps
- Power supply for 3-phase AC induction motor, 230V AC

Figure 8 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)

The MPC5554 demonstration board shows an example of a minimum configuration for a MPC5554 system with the addition of external SRAM and a few optional peripherals. 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 products. The demo board shows an example of connecting the MPC5554 to the MC33394 Power Supply IC, and connectivity to other basic optional circuits (configuration for non-default boot operation and CAN, LIN, and RS-232 transceivers). The demo board also provides access to all pins of the MPC5554 and has some inputs that can be connected to MPC5554 for demonstration purposes. For more information, refer to Reference 2.

Table 1 lists all MPC5554DEMO jumper settings used in the application.

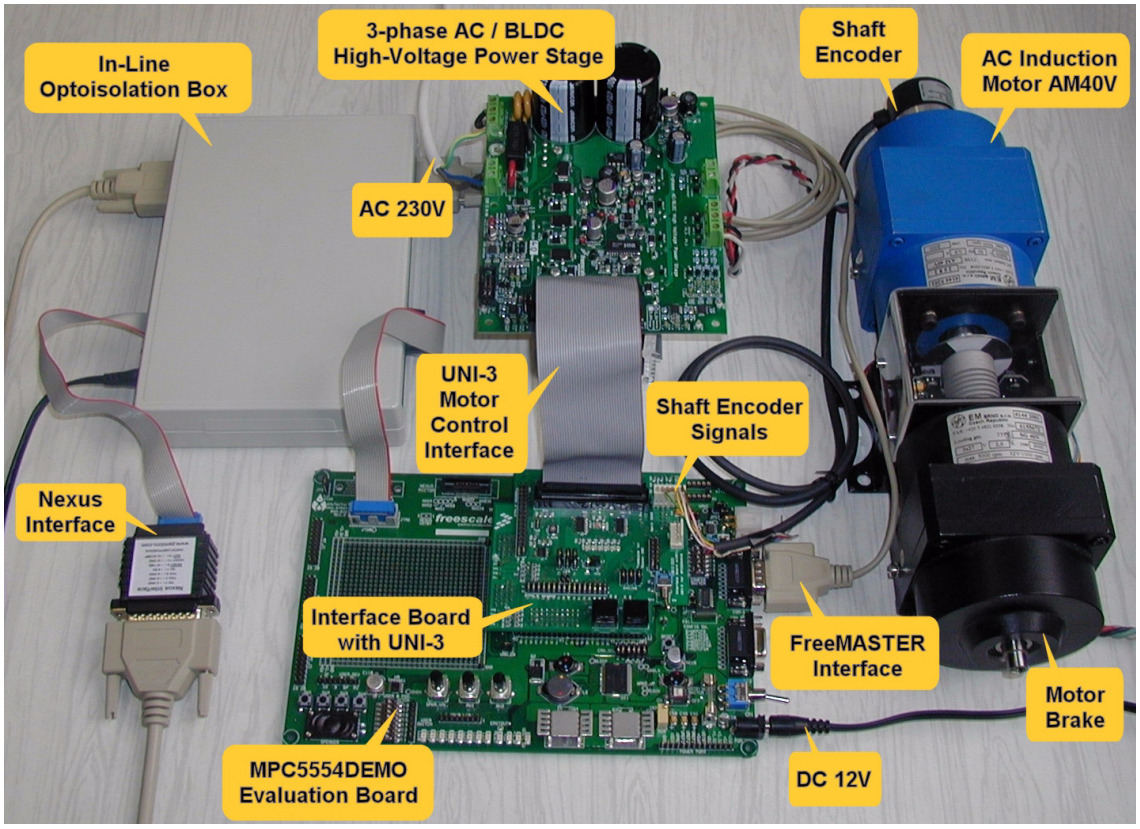


Figure 8. Connection of Application Parts

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.Here is the flashing procedure:

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 9). Select the Binary Image, set Address as 0x0 and check the “Verify after program” option (see Figure 10). 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.

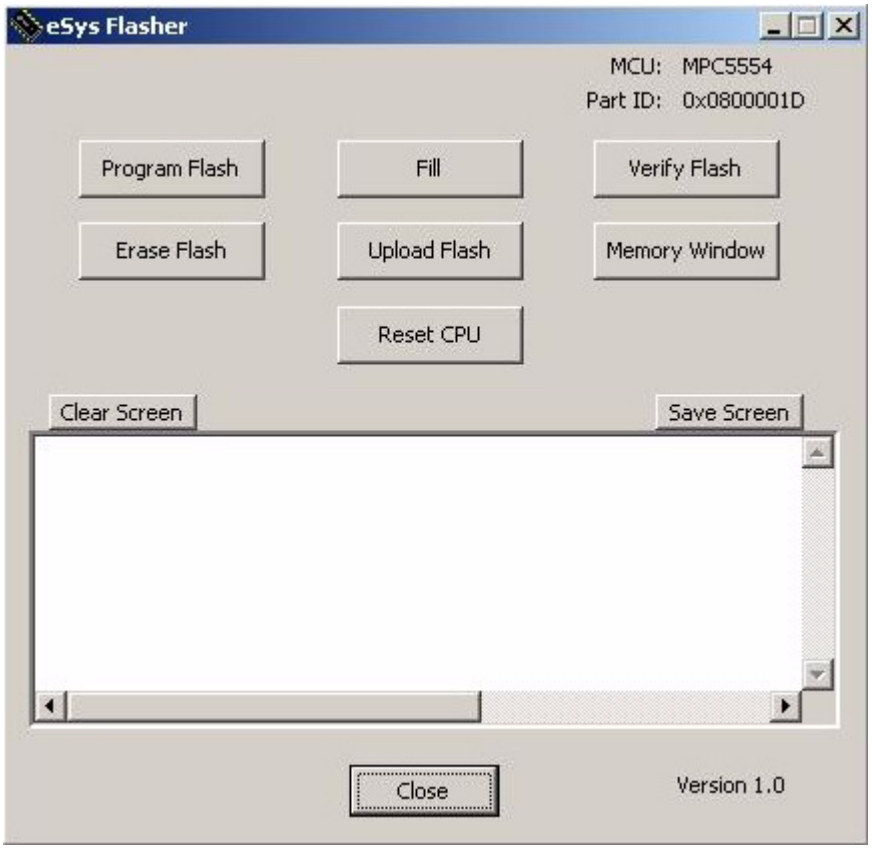


Figure 9. eSysFlasher Target Configuration Window

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

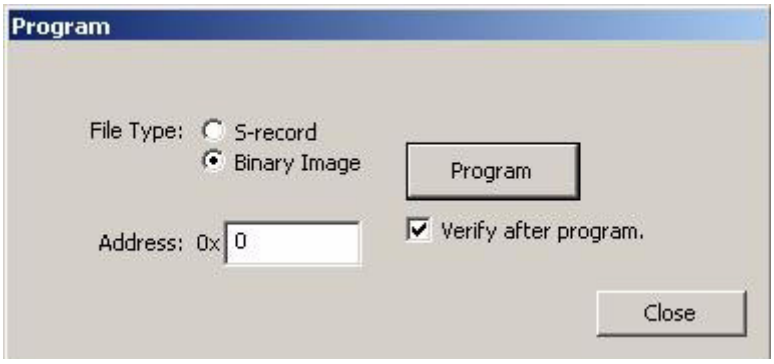


Figure 10. eSys Flasher Program Window

3.3.3 Interface Board with UNI-3

This board connects 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 the MPC5554DEMO

board. Output connections are made via 40-pin UNI-3 connector and expansion headers. Power requirements are met by input connectors. For more information, refer to Reference 3.

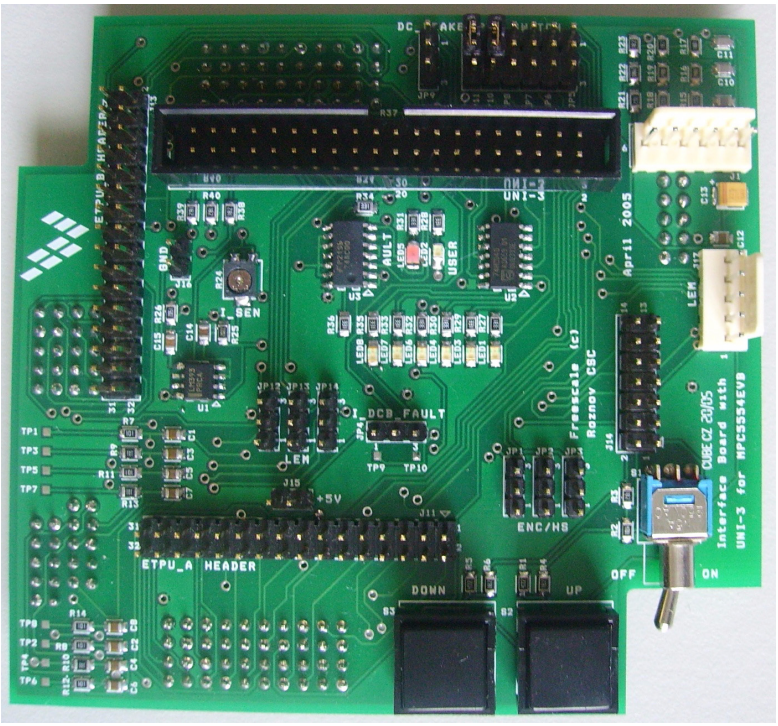


Figure 11. Interface Board with UNI-3

3.3.4 Setting Overcurrent Level

The over-current fault signal is connected to the eMIOS output disable input pin (eMIOS 21) that enables, together with a proper eTPU configuration, handling the fault by eTPU hardware. This connection is part of the MPC5554. To enable handling the fault also by a software, the fault signal, available on eMIOS 21 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 12). Reference 3 describes what voltage must the trimmer define for the over-current comparator.

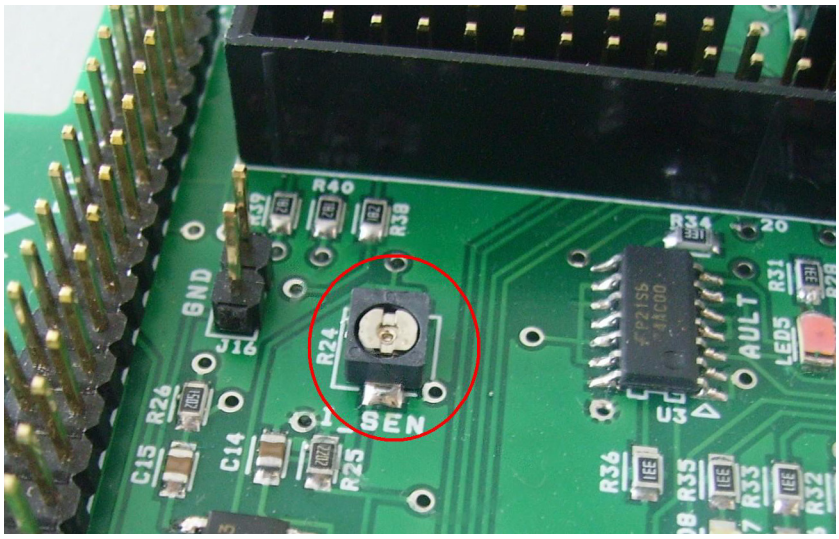


Figure 12. Overcurrent Level Trimmer on Interface Board with UNI-3 (R24)

3.3.5 3-Phase High Voltage AC/BLDC Power Stage Board

The 3-phase high voltage AC/BLDC power stage board provides together with optoisolation and controller boards, a platform for the software development of a wide variety of control algorithms for miscellaneous 3-phase drives: AC induction, brushless DC (BLDC), permanent magnet, and synchronous motors. Control techniques and algorithms can be written and tested without the need to design and build a power stage. This allows you to design the application optimal system with significant time and overall cost savings. The power stage senses a variety of feedback signals suitable for different motor control techniques. It measures all the three phase currents, dc-bus current, dc-bus voltage, back EMF voltages with zero cross sensing, PFC inductor current, and input line zero crossing for PFC. The power module temperature is measured as well. All the analog signals are accommodated to be directly sampled by the MCU's or DSP's AD converter. The 3-phase high voltage AC/BLDC power stage consists of two modules: power module and power stage. The power stage contains sensing and control circuitry and interface connectors. The power module is mounted on a heatsink and contains power devices, which needs to dissipate heat. The main power stage features are:

- Capable of supplying 3-phase AC induction, synchronous, permanent magnet and brushless DC motors
- 1-phase bridge rectifier
- dc-bus brake IGBT and brake antiparallel diode
- Power and signal connectors for external PFC board
- 3-phase bridge inverter (6-IGBT’s)
- Individual phase and dc bus current sensing shunts with Kelvin connections
- Power stage temperature sensing
- IGBT gate drivers
- Current and temperature signal conditioning
- 3-phase back-EMF voltage sensing and zero cross detection circuitry

- Low-voltage on-board power supplies
- Cooling fan
- UNI-3 motor control interface

For more information, refer to Reference 4.

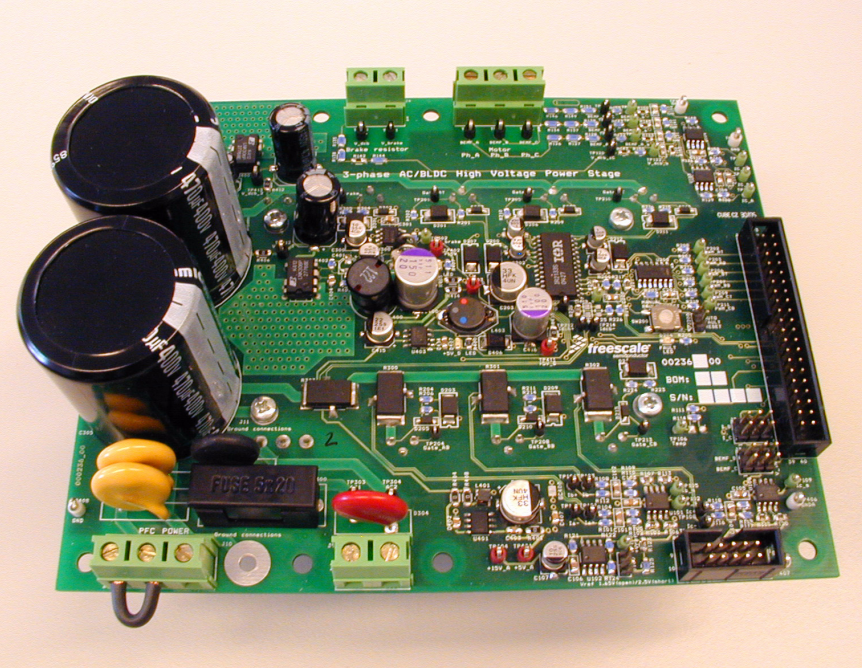


Figure 13. 3-Phase AC/BLDC High Voltage Power Stage Board

3.3.6 Inline Optoisolation Box

Freescale's embedded motion control series in-line optoisolation box links JTAG and RS-232 signals from a workstation to a controller connected to a high-voltage power stage. The box isolates the workstation, and peripherals that may be attached to the workstation, from dangerous voltages that are present on the power stage. The in-line optoisolation box's galvanic isolation barrier also isolates signals from high noise in the power stage and provides a noise-robust systems architecture. The in-line optoisolation box can be used mainly during the development phase of high-voltage and power applications such as electrical drives, power convertors, high voltage systems etc. You can use the in-line optoisolation box to connect your PC to wide range of MCU/DSP EVM boards.

For more information, refer to Reference 5.

3.3.7 AC Induction Motor with BLDC Motor Brake and Quadrature Encoder

The used AC induction motor-brake set incorporates a 3-phase AC induction motor AM40V and attached BLDC motor brake SG 40N. Detailed motor-brake specifications are listed in the following Table 2. For more motor specifications, refer to Reference 6.

Table 2. Motor—Brake Specifications

Set Manufactured	EM Brno, Czech Republic	
Motor Specification:	Motor Type:	AM40V 3-Phase AC Induction Motor
	Pole-Number:	4
	Nominal Speed:	1300 rpm
	Nominal Voltage:	3 x 200 V
	Nominal Current:	0.88 A
Brake Specification:	Brake Type:	SG40N 3-Phase BLDC Motor
	Nominal Voltage:	3 x 27 V
	Nominal Current:	2.6 A
	Pole-Number:	6
	Nominal Speed:	1500 rpm

Quadrature encoder BHK 16.05A 1024-I2-5 is attached to the motor to scan and encode shaft movement. The basic encoder features are as follows:

- Three channel quadrature output with index pulse
- Resolution 1024 counts per revolution
- -20°C to 85°C operating temperature
- TTL compatible
- Single 5-V supply

For more quadrature encoder specifications, refer to Reference 7.

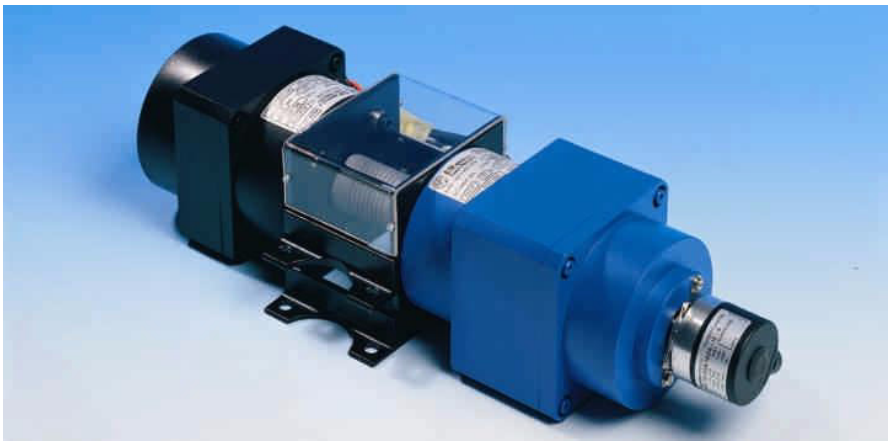


Figure 14. AC Induction Motor with BLDC Motor Brake and Quadrature Encoder

3.3.8 Power Supply

The MPC5554DEMO board is powered by a 12-V/1.2-A power supply, and the 3-phase high voltage AC/BLDC power stage board is powered by a 230-V power supply. The application is scaled for this 230-V power supply.

4 Software Design

This section describes the software design of the ACIM Volts per Hertz control drive application. The system processing is distributed between the CPU and the eTPU, which run in parallel. The CPU and eTPU tasks are described in these terms:

- 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 freemaster_protocol.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, thus ensuring the correct cooperation of eTPU functions.

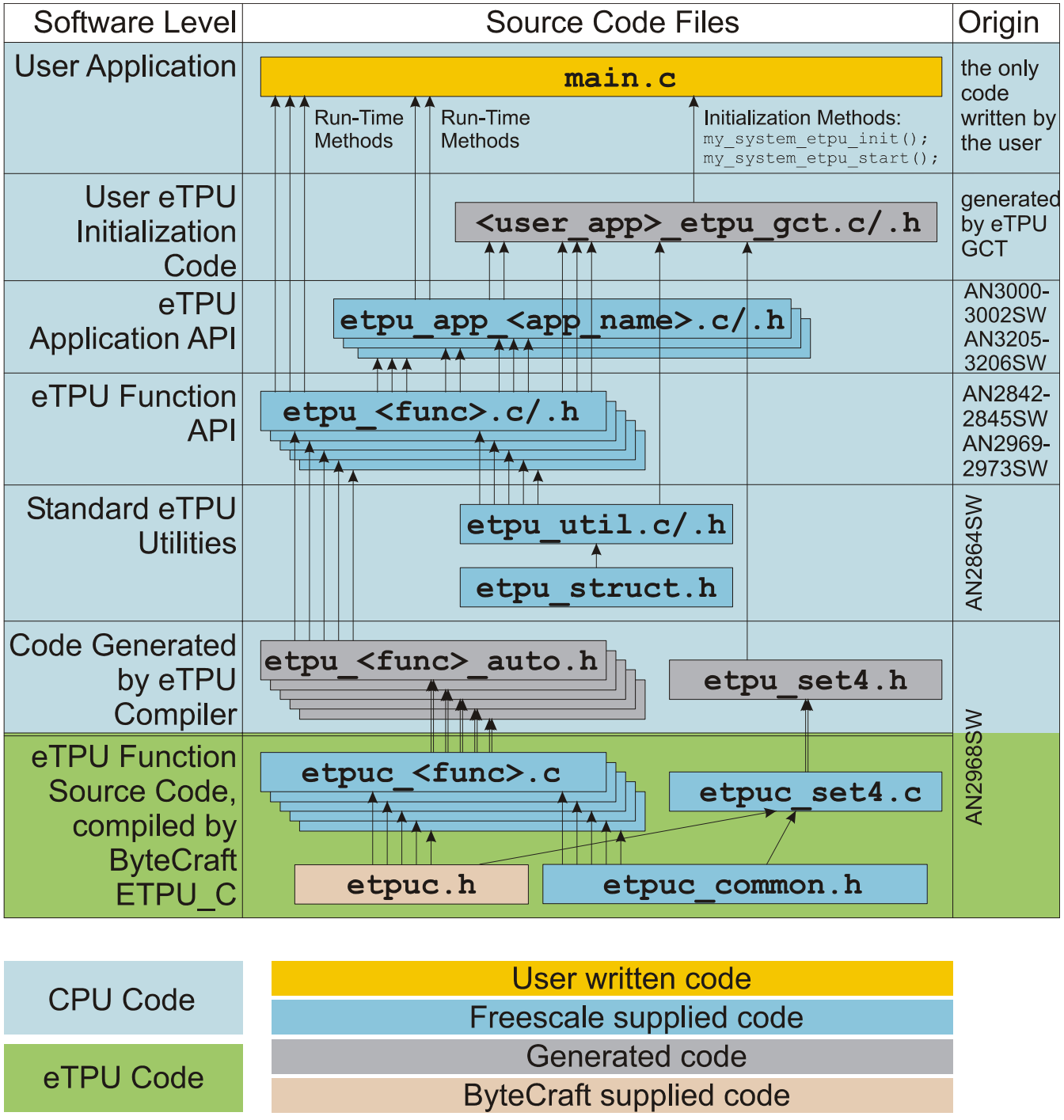


Figure 15. eTPU Project Structure

4.1 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.

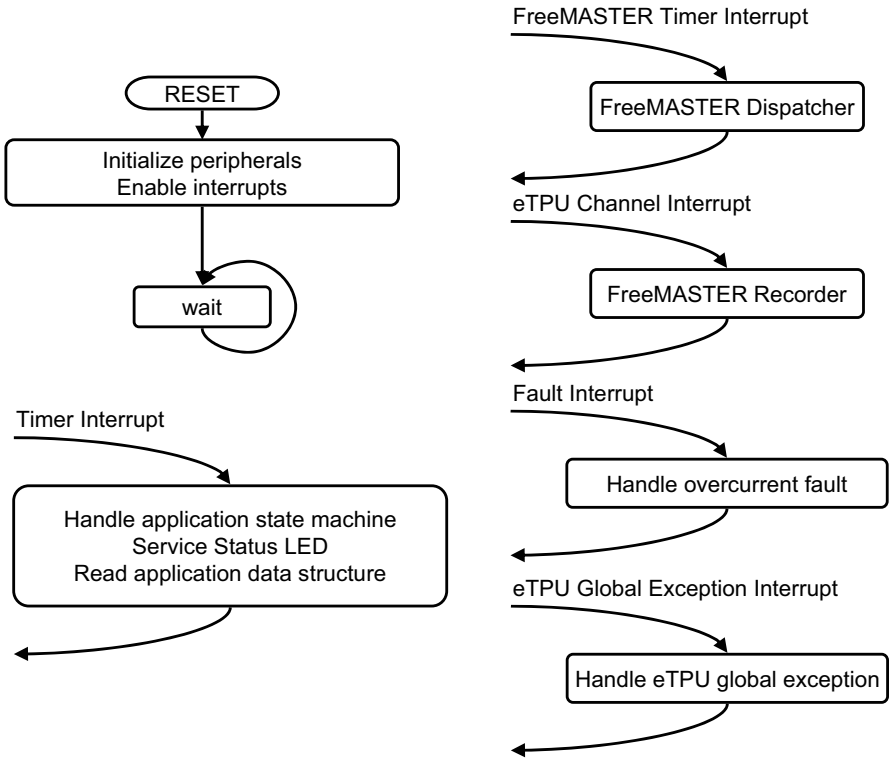


Figure 16. CPU Software Flowchart

4.1.1 Timer Interrupt Service Routine

The timer interrupt is handled by the timer_isr function. These actions are performed periodically, in timer_isr:

- Handle the application state machine.

  The application state diagram is described in detail below.
- Service the status LED by the ApplicationButtonsAndStatusLed function.
- Read the data structure through the eTPU application API routine. `fs_etpu_app_acimvhzsl1_get_data` (see 4.3).

4.1.2 FreeMASTER Interrupt Service Routine

The FreeMASTER interrupt service routine is called FMSTR_Isr. This function is implemented in freemaster_protocol.c.

4.1.3 eTPU Channel Interrupt Service Routine

This interrupt, which is raised every PWM period by the PWMMAC eTPU function running on eTPU channel 7, is handled by the etpu_ch7_isr function. This function calls FMSTR_Recorder, implemented in freemaster_rec.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 21, is handled by the emios_isr function. These actions 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. These situations can cause this interrupt assertion:

- Microcode Global Exception is asserted.
- Illegal Instruction Flag is asserted.
- SCM MISC Flag is asserted.

These actions 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 17) that are controlled by the power on/off switch placed on FreeMASTER control page. 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 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 17), 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 17), 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.

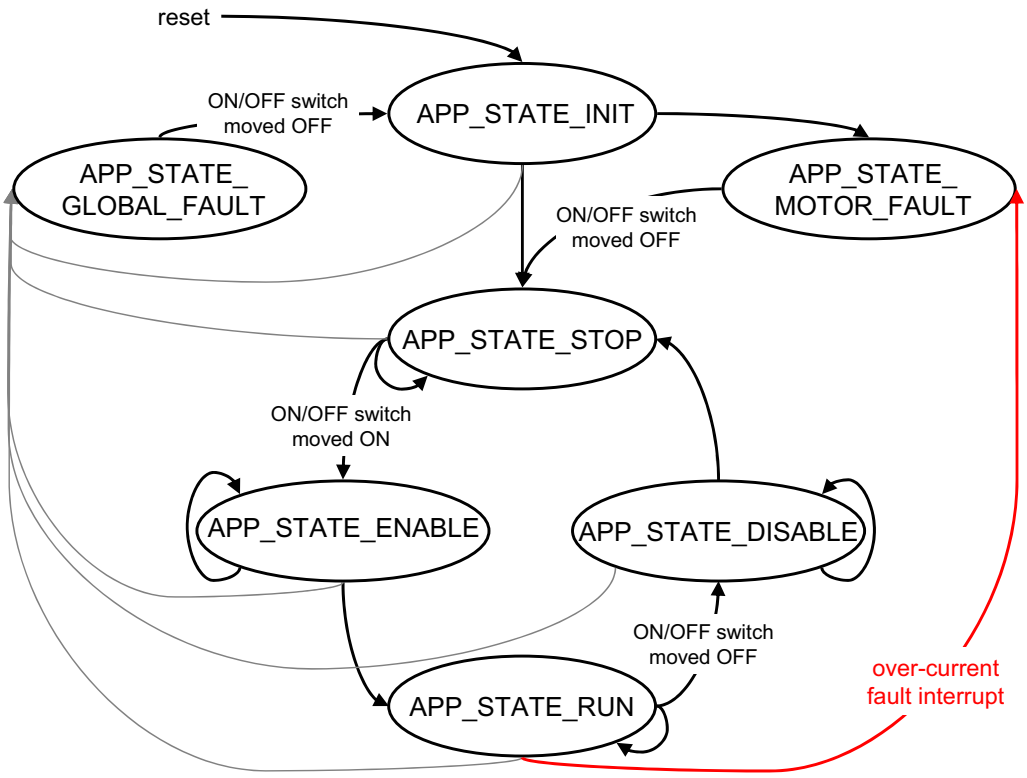


Figure 17. Application State Diagram

The next 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. These actions initialize (re-initialize) the application:

- Disable over_current interrupt.
- Call `my_system_etpu_init` routine for eTPU module initialization.
- Calibrate eQADC.
- Set destination addresses for eDMA1.
- Set source address for eDMA47.
- 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 `acimvhzsl1_etpu_gct.c` file, are generated by the eTPU Graphical Configuration Tool. The eTPU Graphical Configuration Tool can be downloaded from http://www.freescale.com/etpu. For more information, refer to Reference 17.

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

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

  The input signals (from quadrature encoder) are filtered by channel filters. The filter settings guarantee minimum delay of input transition recognition
- TCR1 source = etpuclk div 2
- TCR1 prescaler = 1

  The TCR1 internal eTPU clock is set to its maximum rate of 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 128-MHz 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 - ACIM V/Hz control (ACIMVHZ)
Channel 6 - speed controller (SC)
Channel 7 - PWM master for AC motors (PWMMAC)
Channel 8 - PWM full range (PWMF) - phase A - base channel
Channel 9 - PWM full range (PWMF) - phase A - complementary channel
Channel 10 - PWM full range (PWMF) - phase B - base channel
Channel 11 - PWM full range (PWMF) - phase B - complementary channel
Channel 12 - PWM full range (PWMF) - phase C - base channel
Channel 13 - PWM full range (PWMF) - phase C - complementary channel
Channel 15 - break controller (BC)
Channel 29 - analog sensing for AC motors (ASAC)

These eTPU channels are initialized by the `fs_etpu_app_acimvhzsl1_init` eTPU application API function (see 4.3).

The application settings are as follows:

— PWM phases-type is full range complementary pairs
— PWM frequency 20 kHz
— PWM dead-time 500 ns
— Motor speed range 4000 RPM
— Base frequency 50 Hz
— Boost voltage percentage 10%
— DC Bus voltage 230 V
— DC Bus voltage range 618 V
— Number of motor pole pairs 2
— Speed controller update frequency 1 kHz
— Speed PI controller parameters:
Controller gain is 1, and
Integral time constant is 1 ms
The controller parameters were experimentally tuned.
— Speed ramp parameters:
1000 ms to ramp up from zero to the maximum speed.
— Number of quadrature encoder position counter increments per one revolution 4096
— Break controller mode—PWM-based breaking signal is generated in case of over-voltage.
— Break control signal polarity is active high.
— DC-bus voltage level, at which break control signal is ON, is 130% of the nominal DC-bus voltage
— DC-bus voltage level, at which break control signal is OFF, is 110% of the nominal DC-bus voltage
— ASAC function triggers A/D converter on low-high edge
— DC-bus voltage measurement time, including A/D conversion time and eDMA transfer time, is 8µs
— p_ASAC_result_queue pointer contains the address to eTPU DATA RAM, where the result queue is transferred.
— The samples in result queue are shifted left by 10 bits to achieve bit alignment corresponding to 24-bit fractional format, which is used by eTPU functions.
— DC-bus voltage sample offset within ASAC_result_queue is 0
— ASAC EWMA filter time constant is 1000 µs

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.

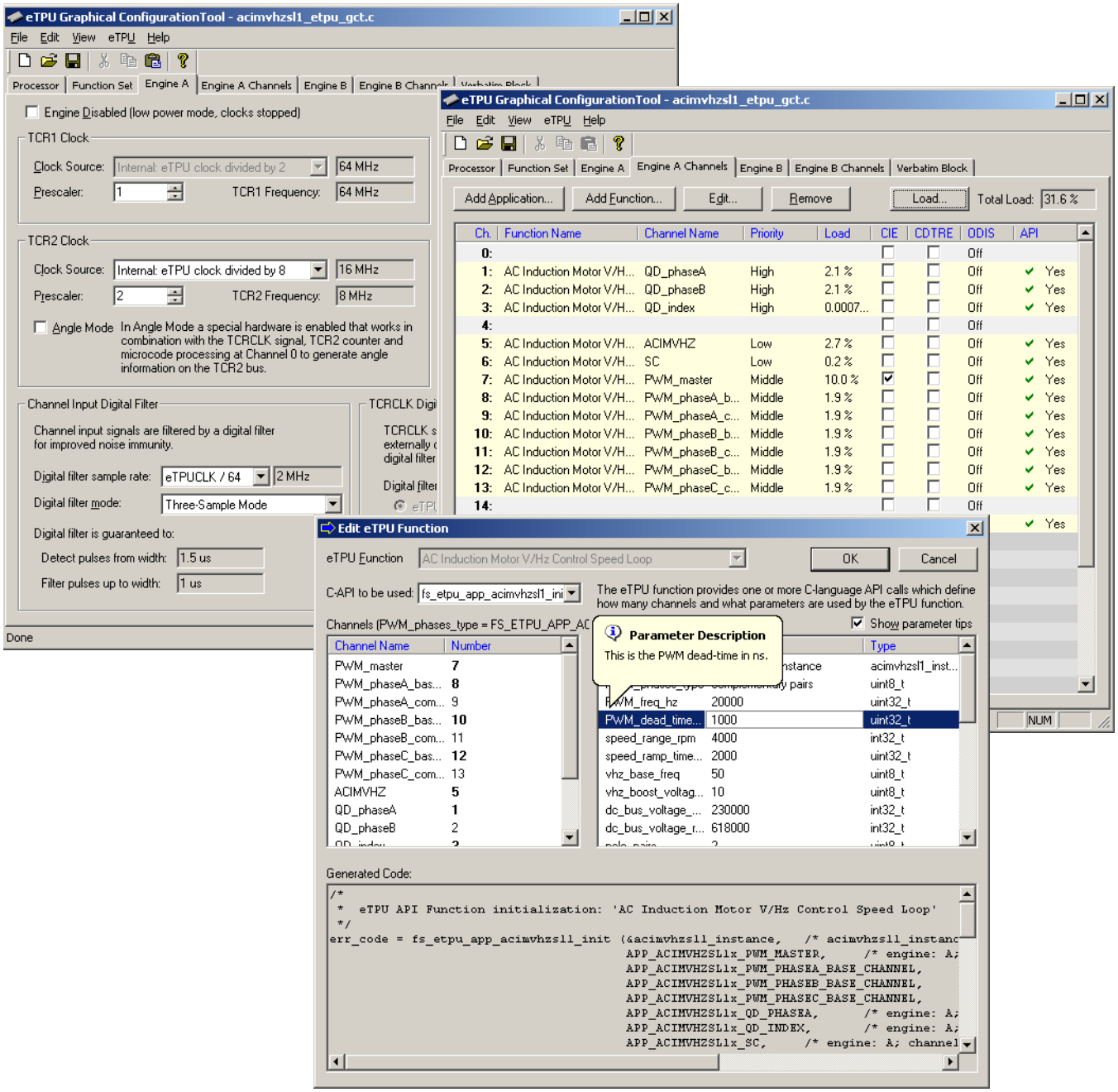


Figure 18. eTPU Configuration Using the eTPU Graphical Configuration Tool

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 to return status information to the PC, and processes control information from the PC. The microprocessor part of the FreeMASTER is initialized by FMSTR_Init() function.

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 you to explore the functionality of the quadrature decoder (QD) eTPU function, watch variables produced by the QD, and see QD signals in FreeMASTER. When the on/off switch on FreeMASTER control page 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. These actions switch the motor drive on:

- Reset the required speed.
- Enable overcurrent interrupt.
- Enable the generation of PWM signals by calling the `fs_etpu_app_acimvhzsl1_enable` application API routine.

If the PWM phases were successfully enabled, the eMIOS channel 21 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 speed gauge on the FreeMASTER control page. The latest value is periodically written to the eTPU.

When the on/off switch on FreeMASTER control page 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. These actions 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 that 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.

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 ACIMVHZSL1 is an abbreviation for ‘ACIM with quadrature encoder and speed Volts per Hertz control’ eTPU motor-control application. As there can be several applications like this, the number 1 denotes the first such application in order.

The ACIMVHZSL1 eTPU application API is described in the following paragraphs. There are 5 basic functions added to the ACIMVHZSL1 application API. The routines can be found in the etpu_app_acimvhzsl1.c/.h files. All ACIMVHZSL1 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_acimvhzsl1_init(
                acimvhzsl1_instance_t * acimvhzsl1_instance,
                            uint8_t   PWM_master_channel,
                            uint8_t   PWM_phaseA_channel,
                            uint8_t   PWM_phaseB_channel,
                            uint8_t   PWM_phaseC_channel,
                            uint8_t   QD_phaseA_channel,
                            uint8_t   QD_index_channel,
                            uint8_t   SC_channel,
```

```
                                    uint8_t   ACIMVHZ_channel,
                                    uint8_t   BC_channel,
                                    uint8_t   ASAC_channel,
                                    uint8_t   PWM_phases_type,
                                    uint32_t  PWM_freq_hz,
                                    uint32_t  PWM_dead_time_ns,
                                    int32_t   speed_range_rpm,
                                    uint32_t  speed_ramp_time_ms,
                                    uint8_t   vhz_base_freq,
                                    uint8_t   vhz_boost_voltage_perc,
                                    int32_t   dc_bus_voltage_mv,
                                    int32_t   dc_bus_voltage_range_mv,
                                    uint8_t   pole_pairs,
                                    uint32_t  SC_freq_hz,
                                    int32_t   SC_PID_gain_permil,
                                    int32_t   SC_I_time_const_us,
                                    uint32_t  QD_pc_per_rev,
                                    uint8_t   BC_mode,
                                    uint8_t   BC_polarity,
                                    uint8_t   BC_u_dc_bus_ON_perc,
                                    uint8_t   BC_u_dc_bus_OFF_perc,
                                    uint8_t   ASAC_polarity,
                                    uint24_t  ASAC_measure_time_us,
                                    uint32_t *ASAC_result_queue,
                                    uint8_t   ASAC_bit_shift,
                                    uint8_t   ASAC_u_dcbus_queue_offset,
                                    uint32_t  ASAC_filter_time_constant_u_us)
```

- Change operation functions:

```
int32_t fs_etpu_app_acimvhzsl1_enable(
                acimvhzsl1_instance_t * acimvhzsl1_instance,
                                uint8_t   PWM_modulation_type)

int32_t fs_etpu_app_acimvhzsl1_disable(
                acimvhzsl1_instance_t * acimvhzsl1_instance)

void fs_etpu_app_acimvhzsl1_set_speed_required(
                acimvhzsl1_instance_t * acimvhzsl1_instance,
                                int32_t   speed_required_rpm)
```

- Value return functions:

```
void fs_etpu_app_acimvhzsl1_get_data(
                acimvhzsl1_instance_t * acimvhzsl1_instance,
                    acimvhzsl1_data_t * acimvhzsl1_data)
```

4.3.1 int32_t fs_etpu_app_acimvhzsl1_init(...)

This routine initializes the eTPU channels for the AC induction motor with quadrature encoder and speed closed loop application. This function has these parameters:

- **acimvhzsl1_instance (acimvhzsl1_instance_t*)** - This is a pointer to acimvhzsl1_instance_t structure, which is filled by `fs_etpu_app_acimvhzsl1_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 acimvhzsl1_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 case of complementary signals generation (PWM_phases_type=FS_ETPU_APP_ACIMVHZSL1_COMPL_PAIRS) the complementary channel is a channel one 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 case of complementary signals generation (PWM_phases_type=FS_ETPU_APP_ACIMVHZSL1_COMPL_PAIRS) the complementary channel is a channel one 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 case of complementary signals generation (PWM_phases_type=FS_ETPU_APP_ACIMVHZSL1_COMPL_PAIRS) the complementary channel is a channel one 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.
- **ACIMVHZ_channel (uint8_t)** - This is the ACIM V/Hz 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_ACIMVHZSL1_SINGLE_CHANNELS, or
  FS_ETPU_APP_ACIMVHZSL1_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 synchronous motor speed in rpm.

- **speed_ramp_time_ms (uint32_t)** - This parameter defines the required speed ramp time in ms. A step change of required speed from 0 to speed_range_rpm is slowed down by the ramp to take the defined time.
- **vhz_base_freq (uint8_t)** - This parameter defines the motor base frequency in Hz. This parameter should be assigned a value of 50 (50Hz) or 60 (60Hz).
- **vhz_boost_voltage_perc (uint8_t)** - This parameter defines the motor boost voltage as a percentage of base voltage. This parameter should be assigned a value of 0 to 100.
- **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_PID_gain_permil (int32_t)** - This is the speed PI controller gain in millesimals.
- **SC_I_time_const_us (int32_t)** - This is the speed PI controller integral time constant in µs.
- **QD_qd_pc_per_rev (uint24_t)** - This is the number of QD position counter increments per one revolution.
- **BC_mode (uint8_t)** - This is the function mode. This parameter should be assigned a value of: FS_ETPU_APP_ACIMVHZSL1_BC_MODE_ON_OFF, or FS_ETPU_APP_ACIMVHZSL1_BC_MODE_PWM.
- **BC_polarity (uint8_t)** - This is the BC output polarity. This parameter should be assigned a value of: FS_ETPU_APP_ACIMVHZSL1_BC_ON_HIGH, or FS_ETPU_APP_ACIMVHZSL1_BC_ON_LOW.
- **BC_u_dc_bus_ON_perc (uint8_t)** - This is the proportion between U_DC_BUS, above which the BC output is ON, and the nominal U_DC_BUS, expressed in percentage (usually about 130%).
- **BC_u_dc_bus_OFF_perc (uint8_t)** - This is the proportion between U_DC_BUS, below which the BC output is OFF, and the nominal U_DC_BUS, expressed in percentage (usually about 110%).
- **ASAC_polarity (uint8_t)** - This is the polarity to assign to the ASAC function. This parameter should be assigned a value of: FS_ETPU_APP_ACIMVHZSL1_ASAC_PULSE_HIGH or FS_ETPU_APP_ACIMVHZSL1_ASAC_PULSE_LOW.
- **ASAC_measure_time_us (uint24_t)** - Time from the first (triggering) edge to the second edge, at which the result queue is supposed to be ready in the DATA_RAM (in us). This value depends on the A/D conversion time and DMA transfer time.
- **ASAC_result_queue (uint32_t *)** - Pointer to the result queue. Result queue is an array of 16-bit words that contains the measured values.
- **ASAC_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_ACIMVHZSL1_ASAC_SHIFT_LEFT_BY_8,

FS_ETPU_APP_ACIMVHZSL1_ASAC_SHIFT_LEFT_BY_10,
FS_ETPU_APP_ACIMVHZSL1_ASAC_SHIFT_LEFT_BY_12, or
FS_ETPU_APP_ACIMVHZSL1_ASAC_SHIFT_LEFT_BY_16.

- **ASAC_queue_offset (uint8_t)** - Position of the U_DC_BUS sample in the result queue. Offset is defined in bytes.
- **ASAC_filter_time_constant_us (uint32_t)** - This is the time constant of an EWMA filter which applies when processing the DC-bus voltage samples, in us.

4.3.2 int32_t fs_etpu_app_acimvhzsl1_enable(...)

This routine sets the PWM modulation type and enable the generation of PWM signals.
This function has these parameters:

- **acimvhzsl1_instance (acimvhzsl1_instance_t *)** - This is a pointer to acimvhzsl1_instance_t structure, which was filled by fs_etpu_app_acimvhzsl1_init function during initialisation.
- **PWM_modulation_type (uint8_t)** - This is the required PWM modulation type. This parameter should be assigned a values of:
  FS_ETPU_APP_ACIMVHZSL1_MOD_SIN - Pure sinewave modulation, or
  FS_ETPU_APP_ACIMVHZSL1_MOD_SIN3H - Sinewave modulation with third harmonic injection, or
  FS_ETPU_APP_ACIMVHZSL1_MOD_SVMSTD - Standard space Vector Modulation, or
  FS_ETPU_APP_ACIMVHZSL1_MOD_SVMU0N - U0N Space Vector Modulation, or
  FS_ETPU_APP_ACIMVHZSL1_MOD_SVMU7N - U7N Space Vector Modulation.

4.3.3 int32_t fs_etpu_app_acimvhzsl1_disable (...)

This routine disables the generation of PWM signals. It has these parameter:

- **acimvhzsl1_instance (acimvhzsl1_instance_t*)** - This is a pointer to acimvhzsl1_instance_t structure, which is filled by fs_etpu_app_acimvhzsl1_init function.

4.3.4 void fs_etpu_app_acimvhzsl1_set_speed_required(...)

This routine sets the required motor speed. This function has these parameters:

- **acimvhzsl1_instance (acimvhzsl1_instance_t*)** - This is a pointer to acimvhzsl1_instance_t structure, which is filled by fs_etpu_app_acimvhzsl1_init function.
- **speed_required_rpm (int32_t)** - This is the required synchronous speed of the motor in rpm.

4.3.5 void fs_etpu_app_acimvhzsl1_get_data(...)

This routine gets the application state data. This function has these parameters:

- **acimvhzsl1_instance (acimvhzsl1_instance_t*)** - This is a pointer to acimvhzsl1_instance_t structure, which is filled by fs_etpu_app_acimvhzsl1_init function.
- **acimvhzsl1_data (acimvhzsl1_data_t*)** - This is a pointer to acimvhzsl1_data_t structure of application state data, which is updated.

4.4 eTPU Block Diagram

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

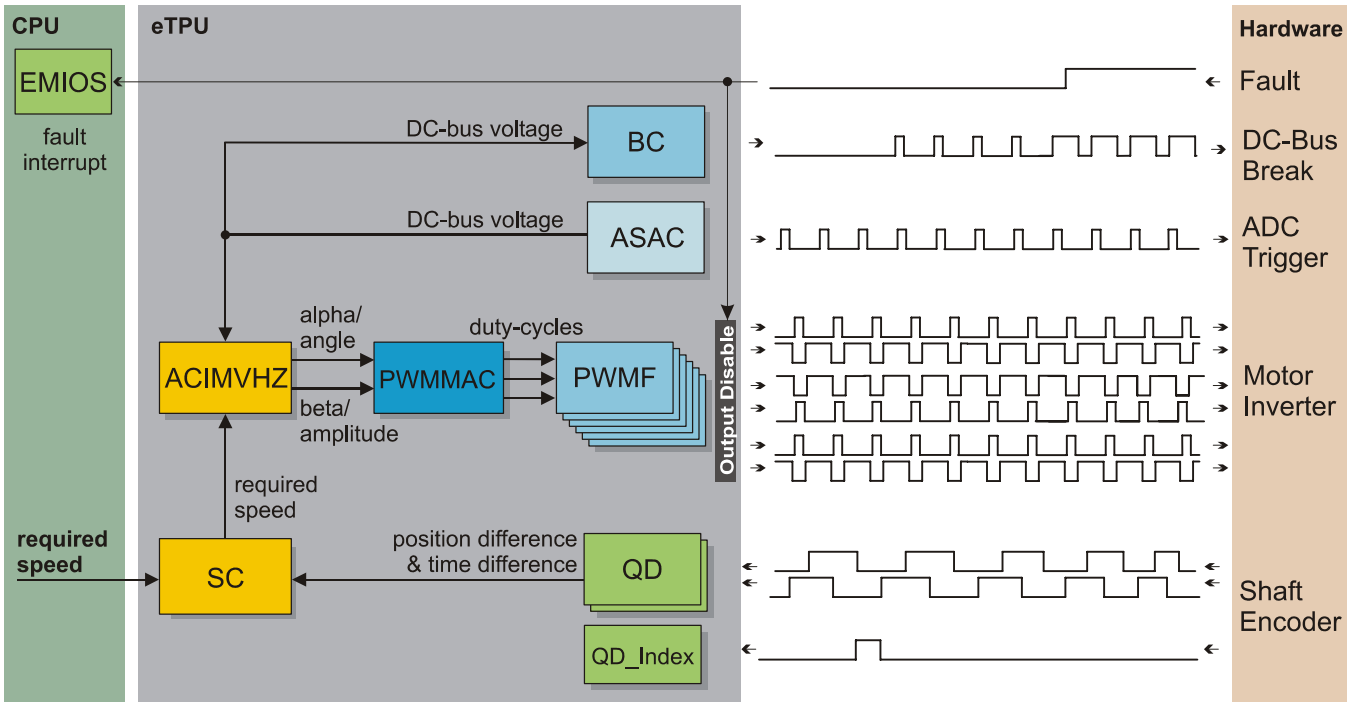


Figure 19. Block Diagram of eTPU Processing

4.4.1 PWM Generator (PWMMAC+PWMF)

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

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

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

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

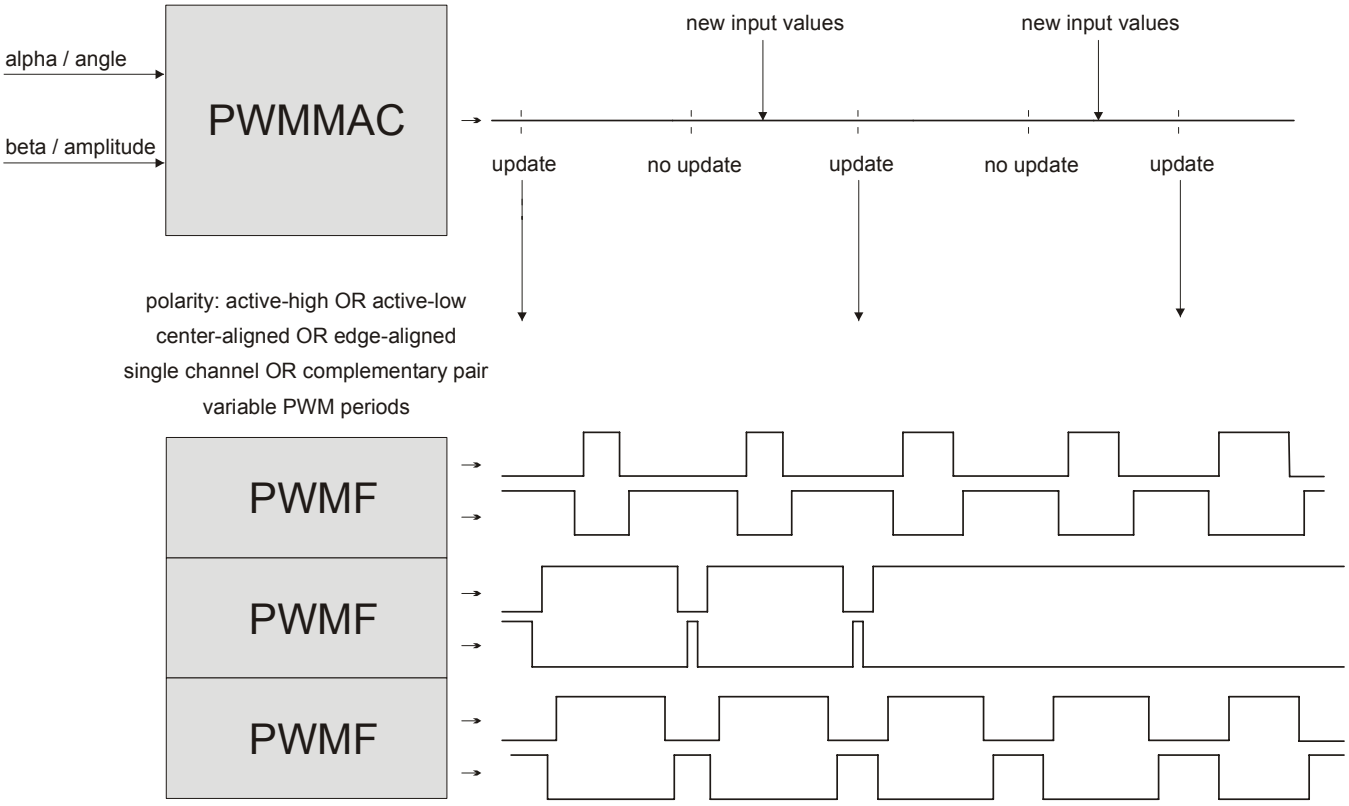
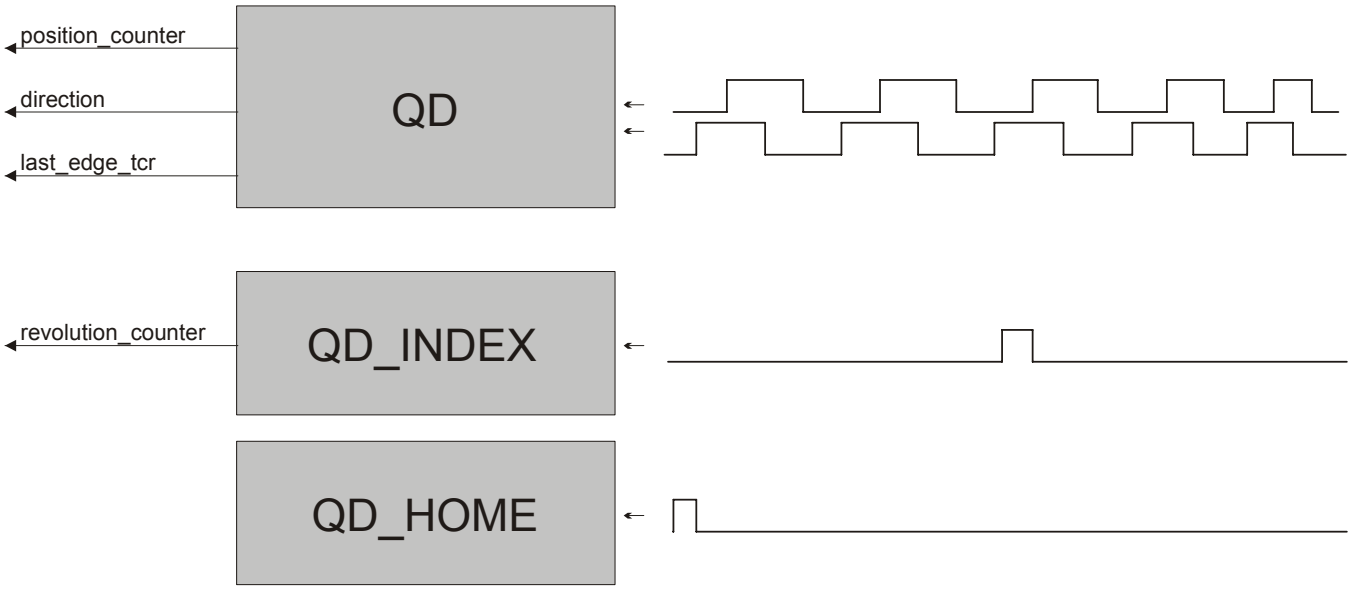


Figure 20. Functionality of PWMMAC+PWMF

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

4.4.2 Quadrature Decoder (QD)

The quadrature decoder eTPU function set processes 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 10.

4.4.3 Analog Sensing for AC Motors (ASAC)

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

All the above mentioned ASAC features are utilized in the application. The ASAC is initialized to run in PWM synchronized mode, e.g. the first ASAC edge is synchronized with the beginning of the PWM period. Simultaneously, the ASAC manages to synchronize the SC function by generating the link to the SC channel every 20^{th} ASAC period and to synchronize the ACIMVHZ function by generating the link to the ACIMVHZ channel each ASAC period.

The ASAC function preprocesses the DC-bus voltage analog values and passes the adjusted values as an input to the ACIMVHZ and BC functions. Processing of the DC-bus voltage sample includes bit shifting, dc-offset removing, and filtering.

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

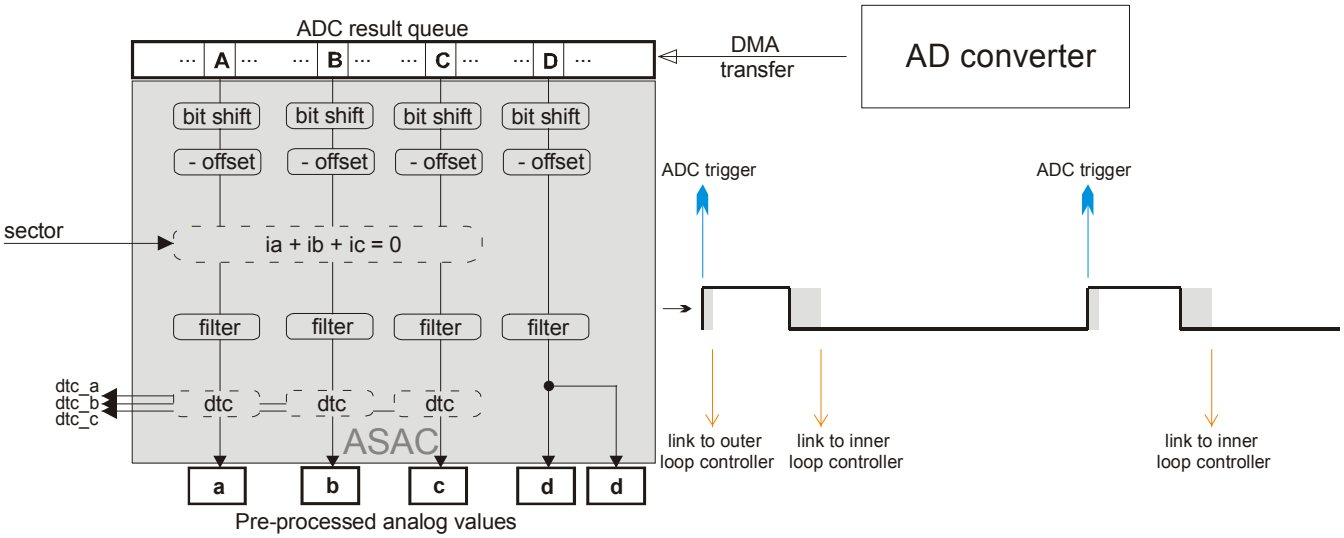


Figure 21. Functionality of ASAC

In order to ensure periodic sampling and the quick transfer of the measured data from the AD converter to the eTPU data RAM, several peripheral modules are used, see Figure 22:

- On-chip analog to digital converter (eQADC) is used for sampling of the analog values. Sampling of analog values is triggered by an eTRIG signal generated internally by the ASAC eTPU function running on eTPU channel 29.
- 2 direct memory access (eDMA) channels are used:
  - eDMA channel 47 transfers a 32-bit eQADC conversion command from the eTPU data RAM (p_ASAC_command_queue) to the EQADC_CFPR2 (CFIFO push register 2) of the eQADC module. The eDMA channel 47 transfer is initiated by a DMA request generated by ASAC eTPU function.
  - eDMA channel 1 transfers the 16-bits DC bus voltage sampling result from the EQADC_RFPR0 (result FIFO pop register 0) to the eTPU data RAM. The DMA channel 1 transfer is initiated by a DMA request generated by eQADC module after the conversion is finished.

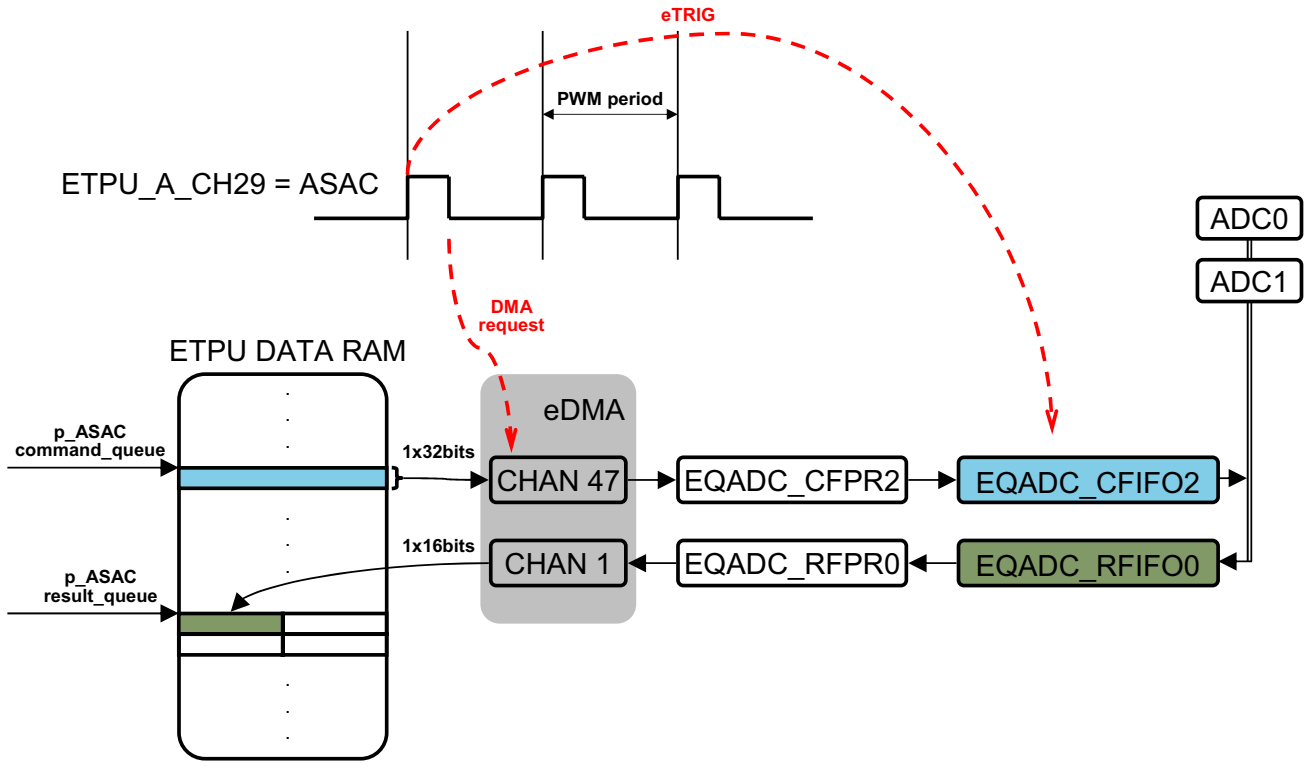


Figure 22. Analog Values Sampling, Cooperation of eTPU, eDMA and eQADC Modules

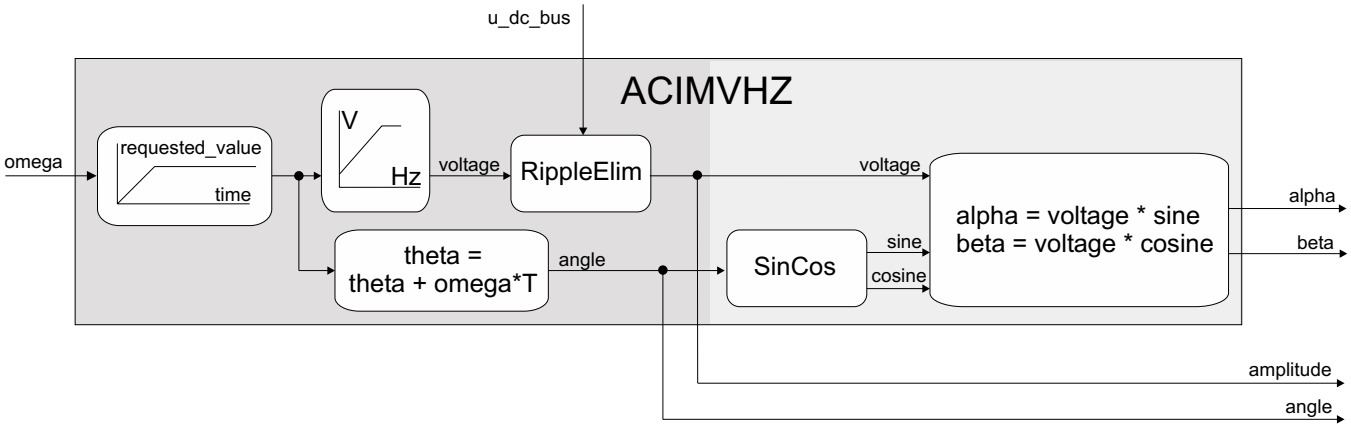
eDMA and eQADC settings are made using the Quick Start Configuration Tool, refer to Reference 18.

4.4.4 ACIM Volts per Hertz (ACIMVHZ)

The ACIM Volts per Hertz eTPU function does not process input or output signals. Its purpose is to control another eTPU function's input parameter. The ACIMVHZ function can be executed even on an eTPU channel not connected to an output pin. The purpose of the ACIMVHZ function is to perform the simple V/Hz control of the AC induction motor. The ACIMVHZ eTPU function performs the calculation of the output applied voltage vector components in this order:

- Optionally passes the required speed through the speed ramp
- Updates applied voltage amplitude using V/Hz ramp
- Eliminates DC-bus ripples
- Updates angle of the output vector
- Determines sin(theta), cos(theta), α and β in case of α, β vector components calculation mode

The ACIMVHZ calculates applied voltage vector components based on the required speed (omega) and the defined V/Hz ramp.



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

4.4.5 Speed Controller (SC)

The speed controller eTPU function does not process input or output signals. It controls another eTPU function's input parameter. The SC function can be executed even on an eTPU channel not connected to an output pin. The SC function includes a general PID controller algorithm. The controller calculates its output based on two inputs: a measured value and a required value. The measured value (the actual motor speed) is calculated based on inputs provided by the QD function. The required value is an output of the speed ramp, whose input is a SC function parameter, and can be provided by the CPU or another eTPU function. In the motor-control eTPU function set, this function mostly provides the speed outer-loop.

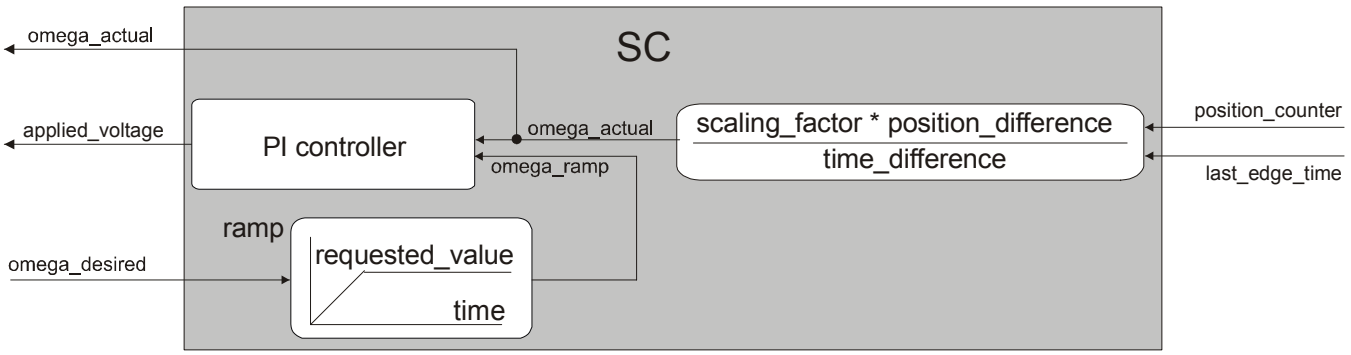


Figure 23. Functionality of SC

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

4.4.6 Break Controller (BC)

The purpose of the break controller (BC) eTPU function is to eliminate DC-bus overvoltage when a motor is driven in the generating mode. The BC function generates the DC-bus break control signal (see Figure 24) based on the actual DC-bus voltage.

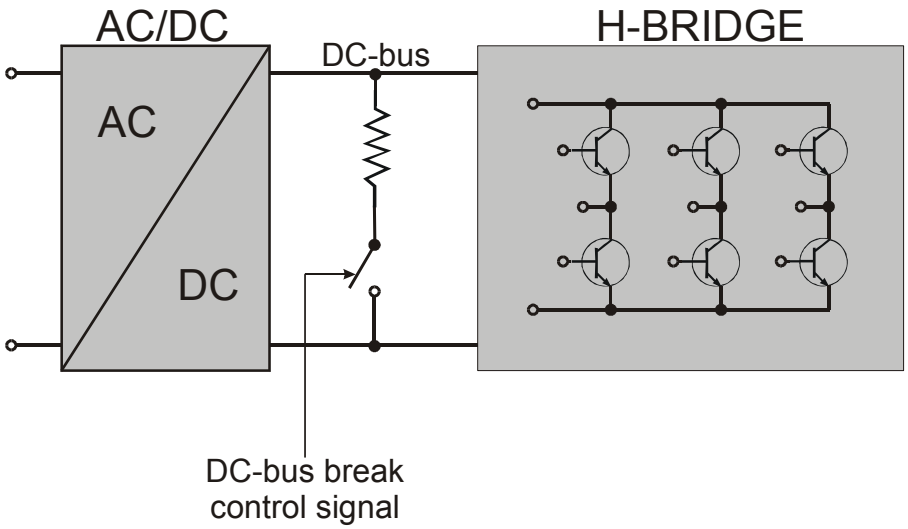


Figure 24. Functionality of BC

The described application uses the PWM mode of the BC function. In this mode, the BC function switches softly using a PWM signal. The *u_dc_bus_ON* and *u_dc_bus_OFF* thresholds define a ramp (see Figure 25). When the DC-bus voltage is lower than *u_dc_bus_OFF*, the control signal is turned off. Between the *u_dc_bus_OFF* and *u_dc_bus_ON* thresholds, a PWM signal with a duty-cycle linearly increasing from 0% to 100% is generated. Above the *u_dc_bus_ON* threshold, the control signal is turned on.

The functionality of the BC is shown in Figure 26, Figure 27, and Figure 28. Signal 1 (dark blue line) represents the DC-bus voltage, signal 2 (light blue line) reflects the DC-bus break control signal generated by the BC.

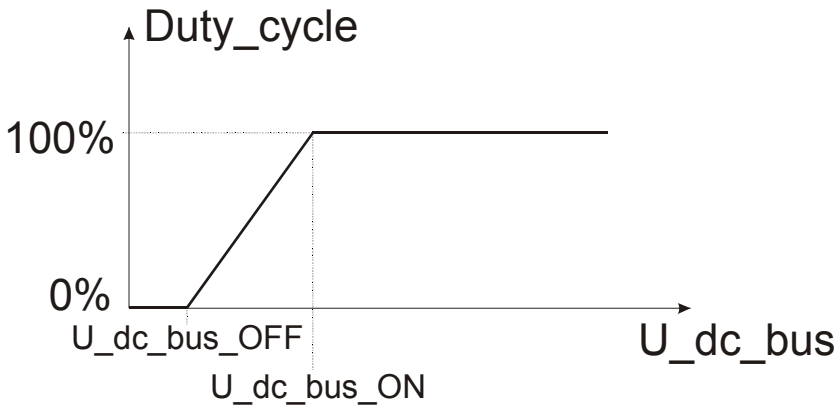


Figure 25. PWM Mode of the Break Controller Function

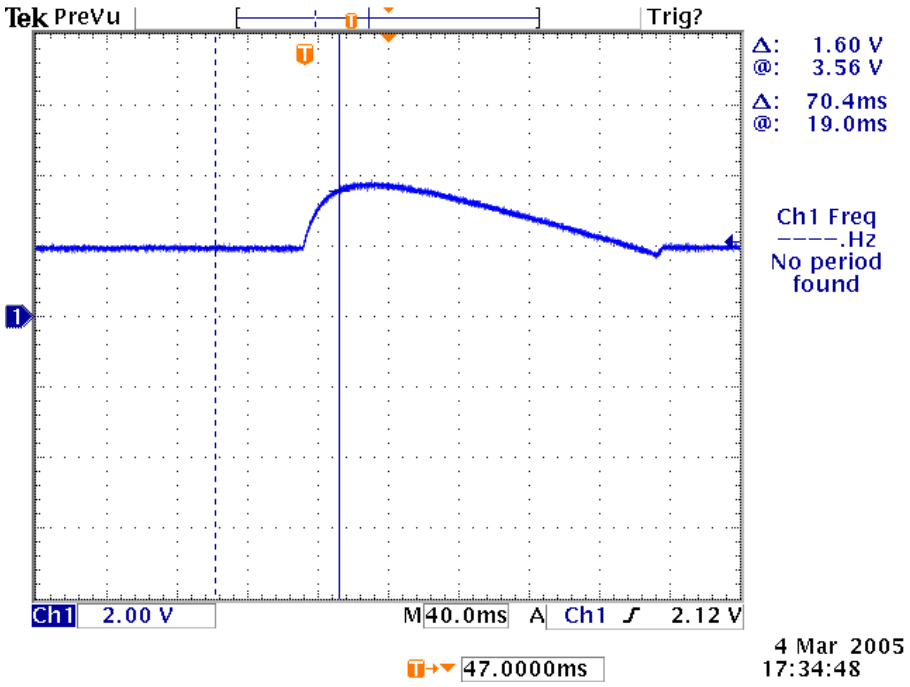


Figure 26. Oscilloscope Screenshot Showing DC-Bus Voltage Course when a Motor Reaches to Generating Mode (Without Action of Break Controller)

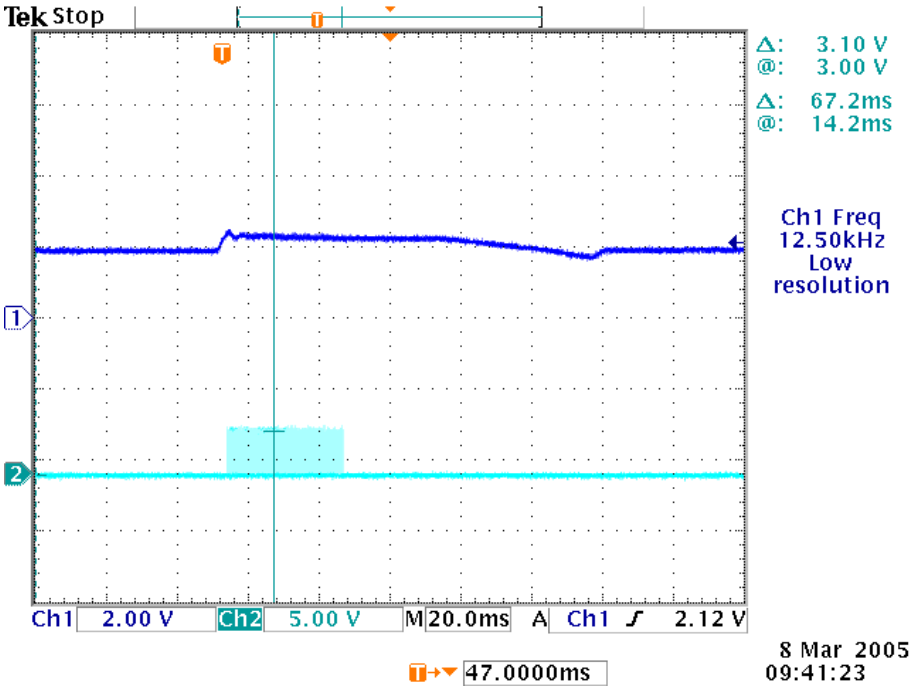


Figure 27. Oscilloscope Screenshot Showing Functionality of the Break Controller

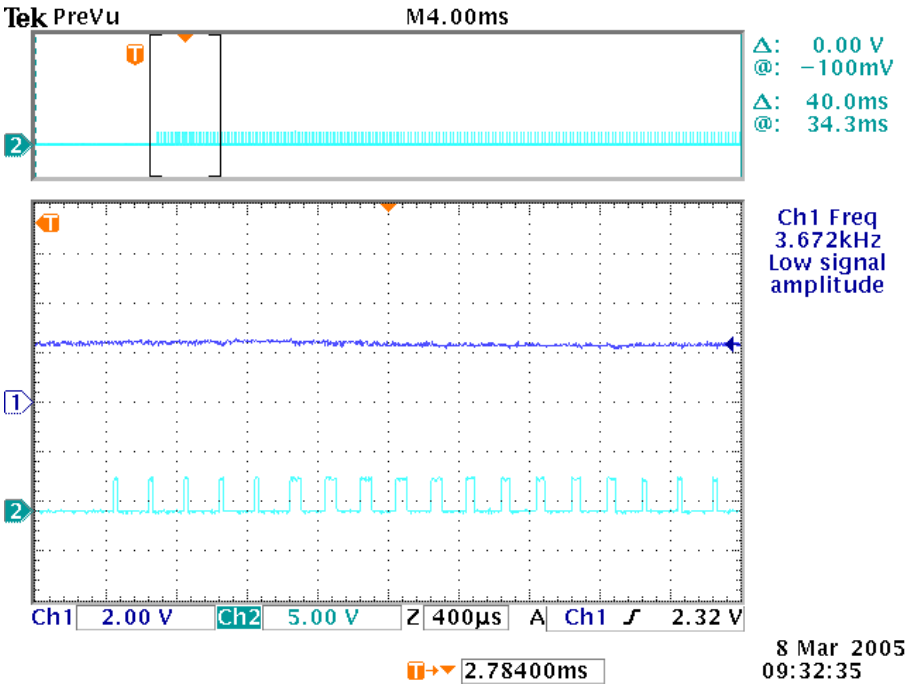


Figure 28. Oscilloscope Screenshot Showing Functionality of the Break Controller in Detail (Zoom of DC-Bus Break Control Signal)

For more details about the BC eTPU function, refer to Reference 15.

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 these 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.

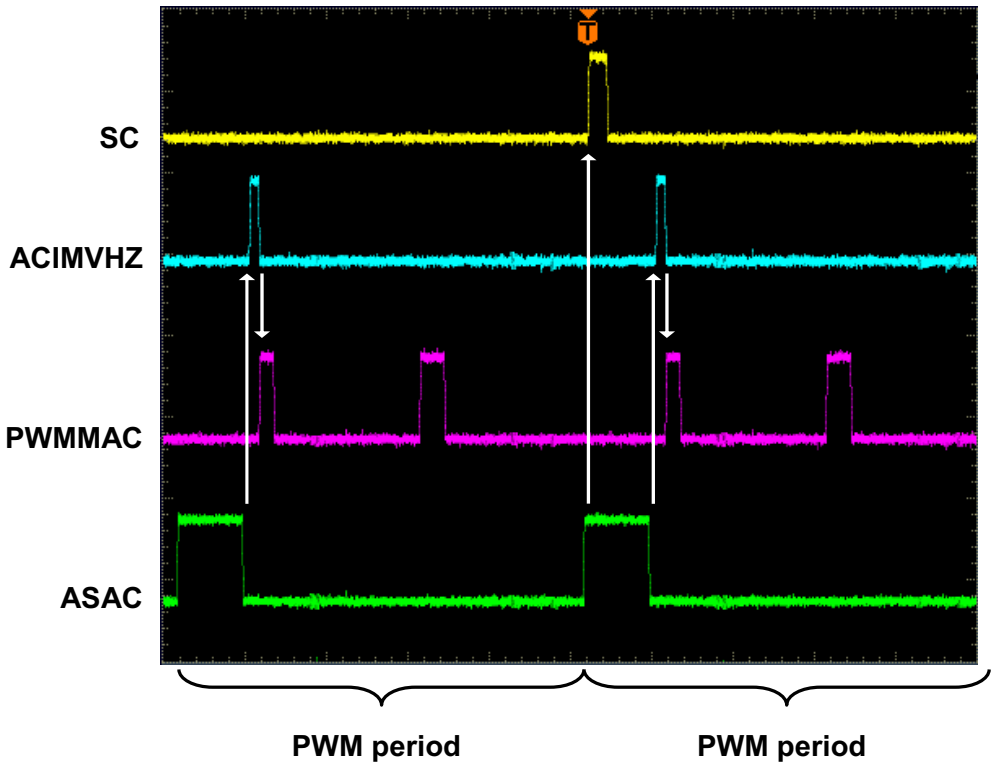


Figure 29. Oscilloscope Screenshot of eTPU Timing

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 you to see the variation of SC, ACIMVHZ, PWMMAC, and ASAC activity pulses.

The yellow signal (1) is generated by the speed controller (SC) eTPU function. Its pulses determine the activity of the SC. The pulse width determines the time necessary to calculate the motor speed from the QD position counter and QD last edge time, calculate the required speed ramp, and apply the PI controller algorithm. This calculation is performed periodically at a 1-kHz rate, which is every 20th PWM period.

The blue (2) signal is generated by the ACIM Volts per Hertz control (ACIMVHZ) eTPU function. The ACIMVHZ update is performed every PWM period.

The violet signal (3) is generated by the PWM master for AC motors (PWMMAC) eTPU function. Its pulses determine the activity of the PWMMAC. Immediately after ACIMVHZ update is finished, a narrow PWMMAC pulse occurs. These pulses determine the service time of an ACIMVHZ request to update the applied motor voltage vector. This service includes also the calculation of space vector modulation. Apart from these pulses, a wider pulses signal the actual update for the next PWM period.

The green signal (4) corresponds to the analog sensing for AC motors (ASAC) eTPU function. This signal triggers the AD converter on the first (low-high) edge. The ASAC pulse width determines the time necessary to sample the DC-bus voltage analog value and transfer the sampled values to the eTPU data

memory. ASAC starts measured samples preprocessing at the time of the second edge when samples are supposed to be ready in the eTPU data memory.

The ASAC thread executed on the first ASAC edge requests SC update every 20th period, by sending a link to the SC channel. Consequently, the SC requests BC update (not displayed on the oscilloscope screen). Both SC and BC updates are finished prior to the ASAC second edge. The ASAC thread executed on the second edge requests ACIMVHZ update. The ACIMVHZ requests PWMMAC to update the applied motor voltage vector. All of these ASAC, SC, BC, and ACIMVHZ activities have to be finished prior to the start of PWMMAC update. If they are not finished, the PWMMAC update is postponed to the next PWM period. The update starts an update_time prior to the end of the period frame, so that the update is finished by the end of the period frame, even in the worst case latency case. Reference 16 describes how to set the update_time value.

5 Implementation Notes

5.1 Scaling of Quantities

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

This equation shows the relationship between real and fractional representations:

$$\text{Fractional Value} = \frac{\text{Real Value}}{\text{Real Quatity Range}} \qquad \textit{Eqn. 1}$$

where:

Fractional value is a fractional representation of the real value [fract24]

Real value is the real value of the quantity [V, A, RPM, etc.]

Real quantity range is the maximal range of the quantity, defined in the application [V, RPM, etc.]

Some quantities are represented using 3.21 format (3 signed integer bits, 21 fractional bits) to eliminate the need of saturation to range (-1, 1). The most negative number that can be represented is -4.0, whose internal representation is 0x800000. The most positive number is 0x7FFFFF or 4.0 - 2^{-21}. In this format, the components of applied motor voltage vector are passed from ACIMVHZ to PWMMAC.

5.2 Speed Calculation

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

$$\text{omega\_actual} = \frac{\text{position\_difference}}{\text{time\_difference}} \cdot \text{scaling\_factor} \qquad \textit{Eqn. 2}$$

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:

$$\text{scaling\_factor} = \frac{30 \cdot 256 \cdot \text{etpu\_tcr\_freq}}{\text{omega\_max} \cdot \text{pc\_per\_rev}} \qquad \textit{Eqn. 3}$$

where:

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

omega_max [RPM] is a maximal speed range

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

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

6 Microprocessor Usage

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

Table 3. Memory Usage in Bytes

Memory	Available	Used
Flash	2M	39 912
RAM	64K	3 492
eTPU code RAM	16K	9 800
eTPU data RAM	3K	1 144

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

- According to Reference 12, the maximum eTPU load produced by PWM generation is 1384 eTPU cycles per one PWM period. The PWM frequency is set to 20kHz, thus the PWM period is 6400 eTPU cycles (eTPU module clock is 128 MHz, at 128 MHz CPU clock).
- According to Reference 10, the contribution of QD function to the overall eTPU time load can be calculated. It depends on the number of shaft encoder pulses (1024) and the motor speed (maximum 3000 rpm). The maximum eTPU busy time per one encoder pulse is 720 eTPU cycles.
- According to Reference 14, the ACIM Volts per Hertz control calculation takes 146 eTPU cycles. The calculation is performed every PWM period.
- According to Reference 11, the speed controller calculation takes 310 eTPU cycles. The calculation is performed every 20th PWM period.

- According to Reference 15, the BC maximum eTPU load per one update in slave PWM switching mode is 64 eTPU cycles, and the BC maximum eTPU load per one PWM edge is 20 eTPU cycles. The BC update is performed every 20 PWM periods. PWM frequency of the DC-bus break control signal is 10 kHz, which means that the BC-PWM update is performed every 2 PWM periods.
- According to Reference 13, the ASAC maximum eTPU load takes 54+96 eTPU cycles (both the first and then the second edge processing is performed). The ASAC function processing is executed 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 the Freescale web site.

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

Table 4 shows the eTPU module time load in several typical situations. For more information, refer to Reference 16.

Table 4. eTPU Time Load

Situation	Average Time Load [%]	Peak Time Load Within PWM Period [%]
Motor Speed 100 RPM (1707 QD pulses per second)	27.7	33.3
Motor Speed 3000 RPM (51200 QD pulses per second)	53.3	58.8

7 Summary and Conclusions

This application note provides a description of the demo application ACIM Volts per Hertz control. 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 MPC5500 family of devices, but it could be easily reused with any device that has an eTPU.

8 References

Table 5. References

1. *MPC5554 Reference Manual*, MPC5554RM
2. *MPC5554DEMO User's Manua*l, MPC5554DEMO EVBUM
3. *Interface Board with UNI-3 User's Manua*l
4. *3-phase AC/BLDC High Voltage Power Stage User's Manual*

Table 5. References (continued)

5. *Freescale Embedded Motion In-Line Optoisolation Box User's Manual*, MEMCILOBUM
6. 3-Phase AC Induction Motor AM40V, EM Brno, web page: http://www.embrno.cz
7. Quadrature Encoder BHK 16.05A 1024-I2-5, Baumer Electric, web page: http://www.baumerelectric.com
8. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
9. *Enhanced Time Processing Unit Reference Manual*, ETPURM
10. "Using the Quadrature Decoder (QD) eTPU Function," AN2842
11. "Using the Speed Controller (SC) eTPU Function," AN2843
12. "Using the AC Motor Control PWM eTPU Functions," AN2969
13. "Using the Analog Sensing for AC Motors (ASAC) eTPU Function," AN2970
14. "Using the ACIM Volts per Hertz Control (ACIMVHZ) eTPU Function," AN2971
15. "Using the Break Controller (BC) eTPU Function," AN2845
16. "Using the AC Motor Control eTPU Function Set (set4)," AN2968
17. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT"
18. *MPC5550 Quick Start User's Manual*

9 Revision History

Table 6 provides a revision history of this document.

Table 6. Revision History

Revision	Location(s)	Substantive Change(s)
Rev. 0		This is the first released version of this document.

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

AN3205
Rev. 0
06/2006