Freescale Semiconductor
Application Note

AN3000
Rev. 0, 06/2006

AC Induction Motor Volts per Hertz Control, Driven by eTPU on MCF523x

Covers MCF523x 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 drive with open loop Volts per Hertz control, which is based on Freescale's MCF523x 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 handles the motor control processing completely, eliminating the microprocessor overhead for other duties.

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

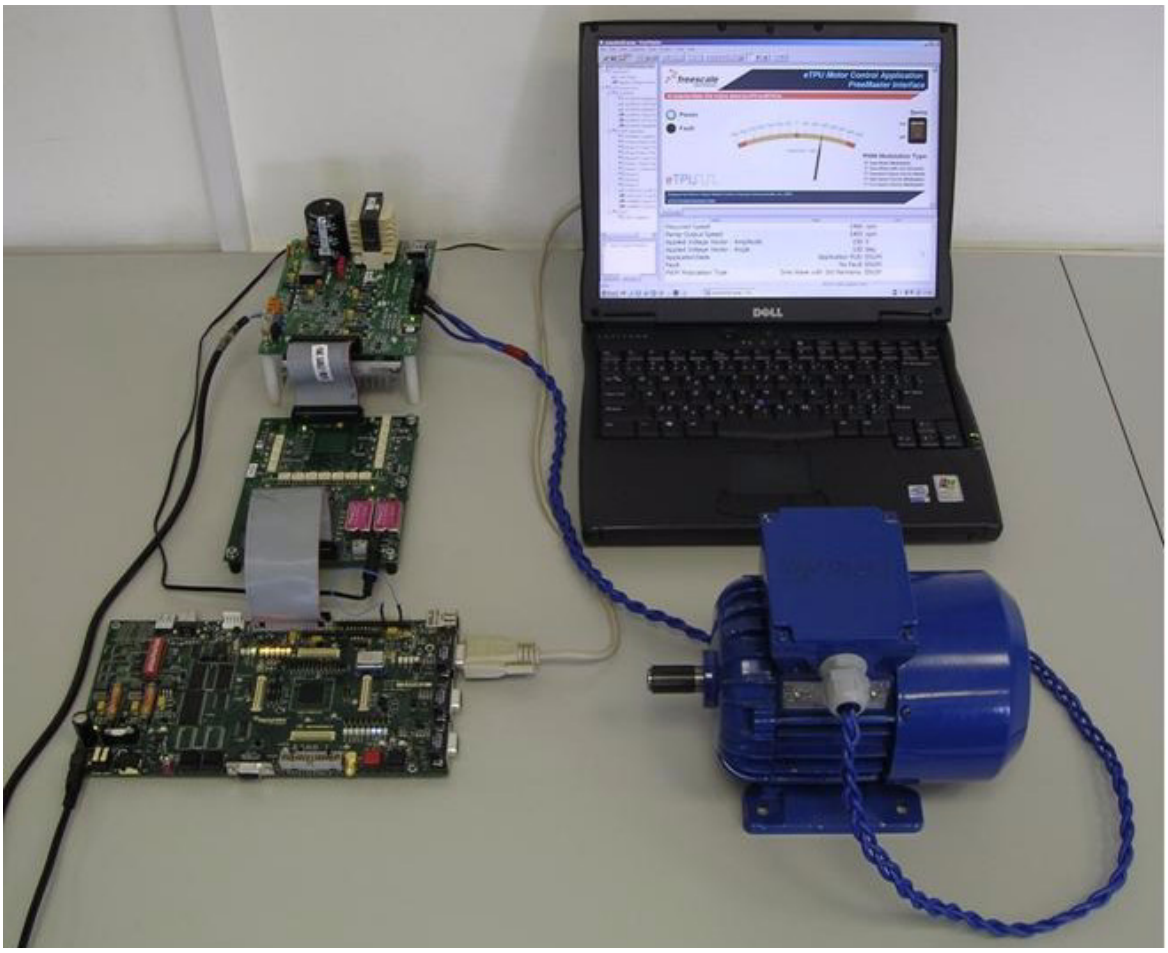


Figure 1. Using M523xEVB, 33395 Evaluation Motor Board, and DC Motor with Optical Sensors

1 ColdFire MCF523x and eTPU Advantages and Features

1.1 ColdFire MCF523x Microprocessor

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

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

On-chip modules include:

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

For more information, refer to Reference 1.

1.2 eTPU Module

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

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

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

For more information, refer to Reference 7.

2 Target Motor Theory

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.

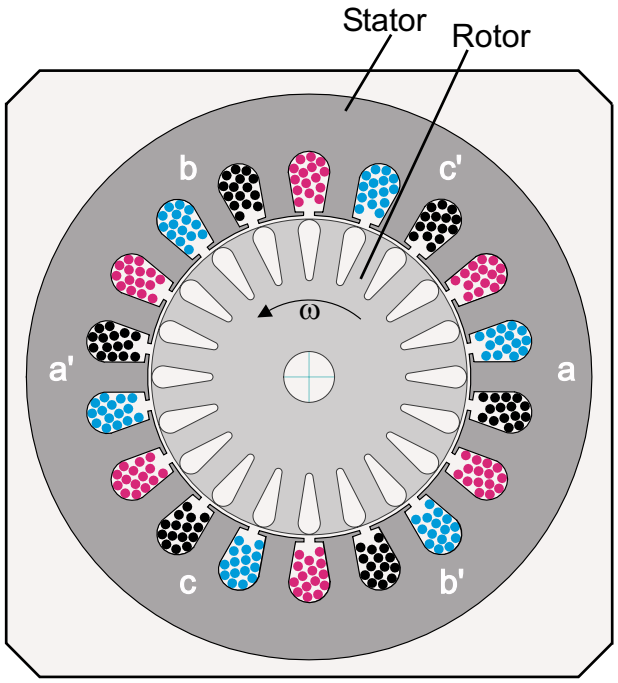


Figure 2. 3-Phase AC Induction Motor

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.

The sinusoidally-distributed flux density wave produced by the stator magnetizing currents generates a voltage in the rotor conductors as it sweeps 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 ($\omega_{r)}$ between the angular velocity ($\omega_{s)}$ 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 ($\omega_{r)}$ 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.

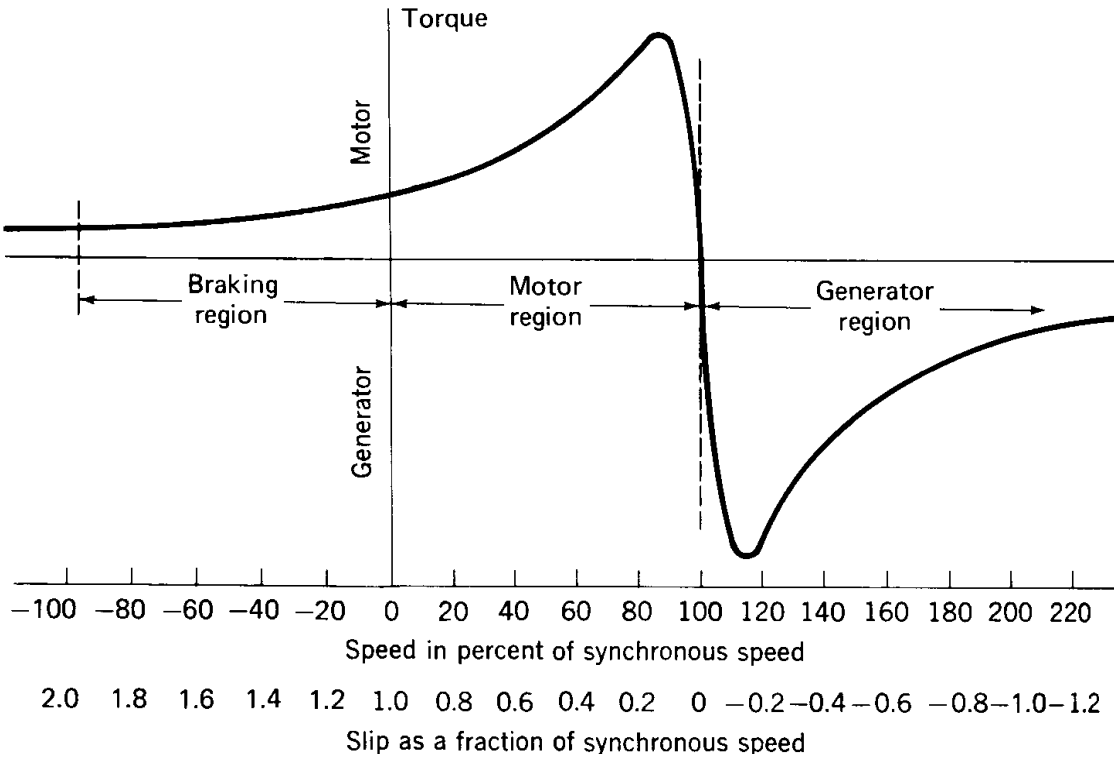


Figure 3. Speed-Torque Characteristic of an AC Induction Motor

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.

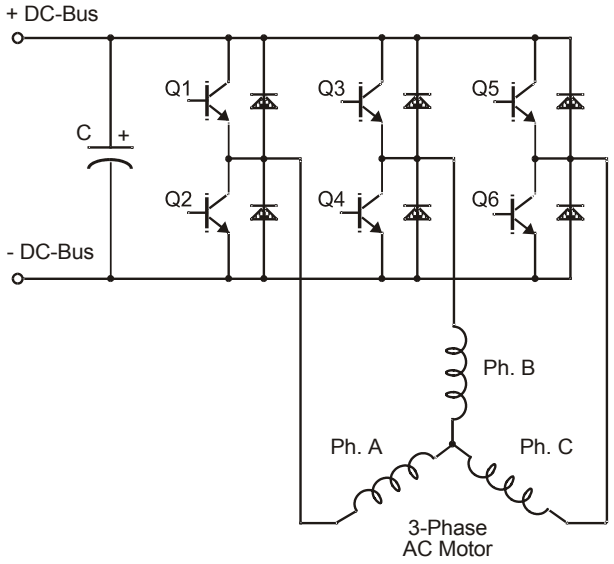


Figure 4. 3-Phase Power Stage

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.

Power devices of the inverter are the key components of the motor control. The most popular devices for motor control applications are power MOSFET's and IGBT's.

A power MOSFET is a voltage controlled transistor. It is designed for high frequency operation. It has low voltage drop and thus low power losses. However, the saturation point and its temperature sensitivity limits the MOSFET application in power circuits.

An insulated gate bipolar transistor (IGBT) is a bipolar transistor controlled by a MOSFET on its base. The IGBT requires low drive current, has fast switching time, and is suitable for high switching frequencies. Its disadvantage is the higher voltage drop of the bipolar transistor that causes higher conduction losses.

2.2 Volts per Hertz Control

The Volt pet Hertz control methods is the most popular method of scalar control which 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.

A simple open-loop Volts/Hertz speed control for an induction motor is the control technique targeted for low performance drives. This basic scheme is unsatisfactory for more demanding applications where speed precision is required.

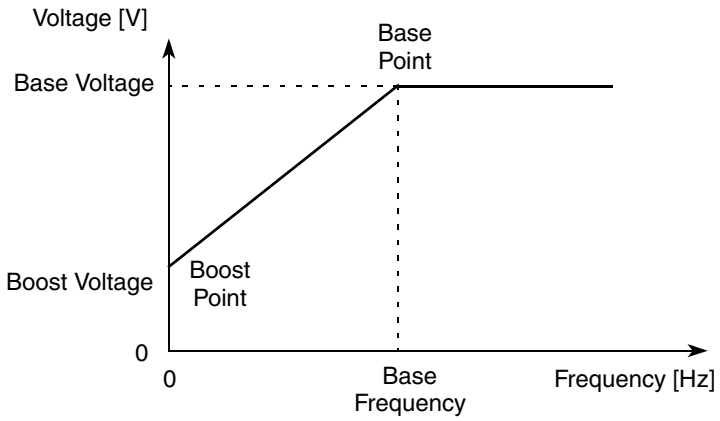


Figure 5. Volts per Hertz Control Method

3 System Concept

3.1 System Outline

The system is designed to drive a three-phase induction motor. The application meets these performance specifications:

- Opened loop scalar control of a three phase general purpose induction motor
- Targeted at ColdFire MCF523x evaluation board (M523xEVB), Optoisolation board, and 3-phase AC/BLDC high voltage power stage
- Running on three-phase general purpose induction motor variable line voltage 230/400V AC (range -15%.....+10%)
- Control technique incorporates:
  - Volts per Hertz control
  - Both directions of rotation
  - 4-quadrant operation
  - Start from any motor position without rotor alignment
  - Maximal speed 3000 RPM at input power line 230V AC
- 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, start/stop status, fault status)
  - Detail description of all eTPU functions used in the application (monitoring of channel registers and all function parameters in real time)
- DC bus over-current fault protection

3.2 Application Description

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

- Evaluation board M523xEVB
- 3-phase AC/BLDC high voltage power stage
- Optoisolation Board
- AC induction motor Sh 71-2A from Cantoni

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 the input variable values, provided by the microprocessor CPU.

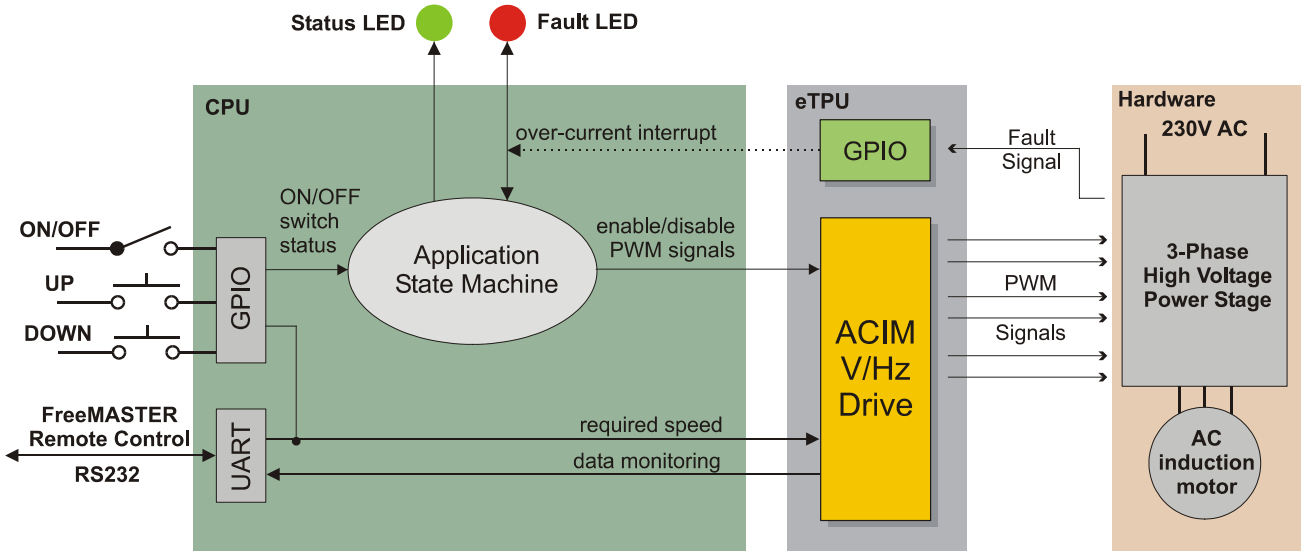


Figure 6. System Concept

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

The CPU performs these tasks:

- Periodically scans 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 these tasks:

- Six eTPU channels (PWMF) are used to generate PWM output signals.
- One eTPU channel (PWMMAC) is internally used to synchronize the PWM outputs and calculate duty-cycles for individual phases from (alpha/beta) or (angle/amplitude) reference frames.
- One eTPU channel (ACIMVHZ) is internally used to perform 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.
- One eTPU channel (GPIO) generates an interrupt to the CPU when the over-current fault signal activates.

3.2.1 User Interface

The application is interfaced by:

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

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

FreeMASTER software was designed to provide an application-debugging, diagnostic, and demonstration tool for the development of algorithms and applications. It runs on a PC connected to the M523xEVB via an RS232 serial cable. A small program resident in the microprocessor communicates with the FreeMASTER software to return status information to the PC and process control information from the PC. FreeMASTER software, executing on a PC, uses part of Microsoft Internet Explorer as the user interface.

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

3.3 Hardware Implementation and Application Setup

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

- M523xEVB
- 3-Phase AC/BLDC high voltage power stage
- AC induction motor
- Power supply for M523xEVB, 9V DC, 2.7 Amps
- Power supply for 3-phase AC induction motor, 230V AC

The Figure 7 shows the connection of these parts.

3.3.1 ColdFire MCF523x Evaluation Board (M523xEVB)

The EVB is intended to provide a mechanism for customers to easily evaluate the MCF523x family of ColdFire microprocessors. The heart of the evaluation board is the MCF5235; all other M523x family members have a subset of the MCF5235 features and can therefore be fully emulated using the MCF5235 device. The M523xEVB is fitted with a single 512K x 16 page-mode Flash memory (U19), giving a total memory space of 2 Mbytes. Alternatively, a footprint is available for upgrading flash to a 512K x 32 page-mode Flash memory (U35), doubling the memory size to 4 Mbytes.

For more information, refer to Reference 2.

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

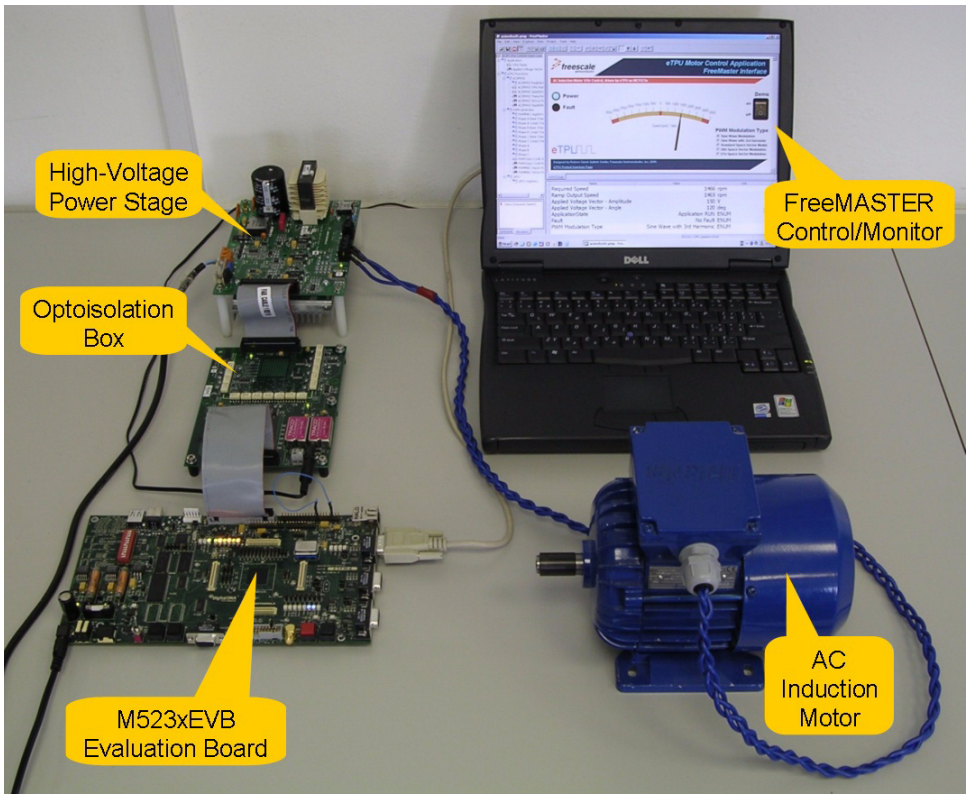


Figure 7. Connection of Application Parts

Table 1. M523xEVB Jumper Settings

Jumper	Setting
JP1	1 2
JP2	1–2
JP3	1 2
JP4	1–2
JP5	1 2–3
JP6	1–2 3
JP7	1 2–3
JP8	1–2 3
JP9	1 2–3
JP10	1 2–3
JP11	1 2–3
JP12	1 2–3
JP13	1 2–3
JP14	1 2–3
JP15	1 2–3
JP16	1 2–3
JP17	1 2–3
JP18	1 2–3
JP19	1 2–3

Jumper	Setting
JP20	1 2–3
JP21	1 2–3
JP22	1 2–3
JP23	1 2–3
JP24	1 2–3
JP25	1–2 3
JP26	1–2 3
JP27	1–2
JP28	1–2
JP29	1–2
JP30	1–2
JP31	1 2–3
JP32	1–2 3
JP33	1–2
JP34	1–2
JP35	1–2 3
JP36	1–2 3
JP37	1–2
JP38	1–2
JP39	1–2

Jumper	Setting
JP40	1–2
JP41	1–2
JP42	1–2
JP43	1–2
JP44	1–2
JP45	1–2
JP46	1–2
JP47	1–2
JP48	1–2
JP49	1–2
JP50	1–2 3
JP51	1–2 3
JP52	1–2 3
JP53	1 2
JP54	1 2
JP55	1 2
JP56	1–2 3
JP57	1–2
JP58	1–2
JP59	1–2

Jumper	Setting
JP60	1–2
JP61	1–2
JP62	1 2
JP63	1 2
JP64	1 2–3
DIP1	ON
DIP2	ON
DIP3	ON
DIP4	ON
DIP5	ON
DIP6	ON
DIP7	OFF
DIP8	ON
DIP9	ON
DIP10	ON
DIP11	OFF
DIP12	ON

3.3.2 Flashing the M523xEVB

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

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

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

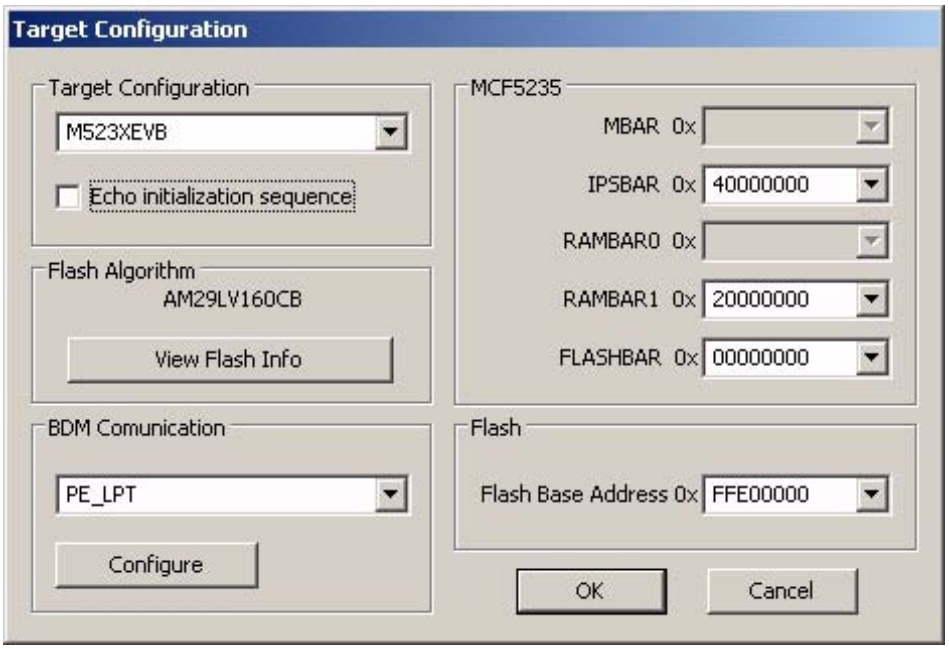


Figure 8. CFFlasher Target Configuration Window

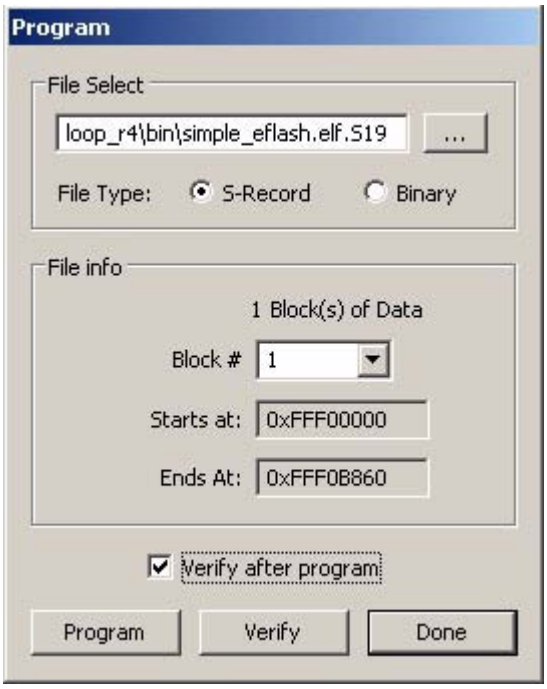


Figure 9. CFFlasher Program Window

3.3.3 Setting Over-Current Level

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

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

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

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

6. Connect pin 16 with pin 6 on the eTPU header. This enables the over-current interrupt. The over-current fault is handled by both hardware and software.
7. Turn the on/off switch on. Using the up and down buttons, set the required speed to the maximum. If the application goes to the fault state during the acceleration, adjust the R41 trimmer level somewhat higher, so that the motor can get to the maximum speed.

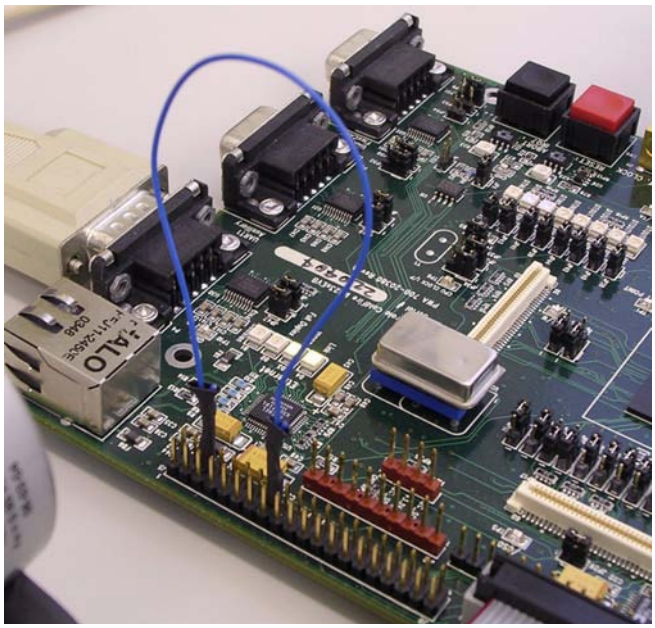


Figure 10. Connection Between LETPUODIS and ETPUCH4 on the eTPU Header

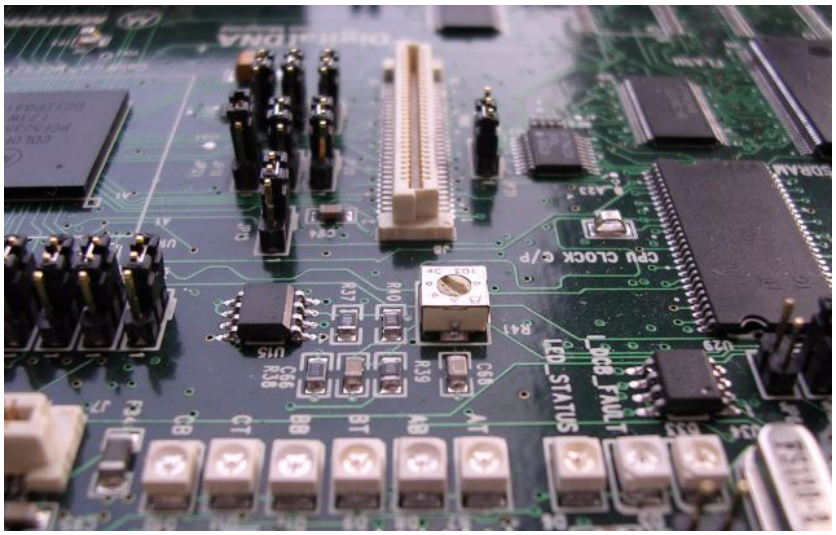


Figure 11. Overcurrent Level Trimmer on M523xEVB (R41)

3.3.4 3-Phase AC/BLDC High Voltage Power Stage

The 3-phase AC high-voltage brushless DC (BLDC) power stage (HV AC power stage) is a 115-/230- volt, 180-watt (one-fourth horsepower), off-line power stage that is an integral part of Freescale's embedded motion control series of development tools.

Features:

- The (HV) AC power stage will operate off DC input voltages from 140 to 230 volts and AC line voltages from 100 to 240 volts
- In combination with one of the embedded motion control series control boards and an embedded motion control series optoisolation board, it provides a software development platform that allows algorithms to be written and tested without the need to design and build a power stage
- It supports a wide variety of algorithms for both AC induction and brushless DC (BLDC) motors
- Input connections are made via a 40-pin ribbon cable connector; power connections to the motor are made on an output connector
- Power requirements are met with a single external 140- to 230-volt DC power supply or an AC line voltage
- Current measuring circuitry is set up for 2.93 amps full scale; both bus and phase leg currents are measured
- A cycle-by-cycle overcurrent trip point is set at 2.69 amps

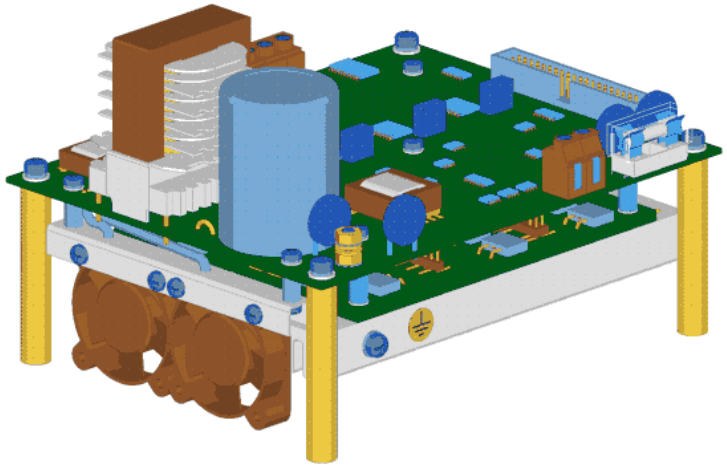


Figure 12. 3-Phase High Voltage AC/BLDC Power Stage

For more information, refer to Reference 3.

3.3.5 Optoisolation Board

Freescale's embedded motion control series optoisolation board links signals from a controller to a high-voltage power stage. The board isolates the controller, and peripherals that may be attached to the controller, from dangerous voltages that are present on the power stage. The optoisolation board's galvanic isolation barrier also isolates control signals from high noise in the power stage and provides a noise-robust systems architecture.

The optoisolation board can be used mainly during the development phase of high-voltage applications to protect the controller board and devices attached to the control board. The user can use the optoisolation board to connect the MC68HC908MR32 motor control board or DSP56F80x EVM boards to high voltage powerstage boards (3-phase high voltage SR power stage and 3-phase high voltage AC/BLDC power stage). Both input and output connections are made via standard 40-pin UNI-3 motor control interface. The pin assignments for both connectors are the same.

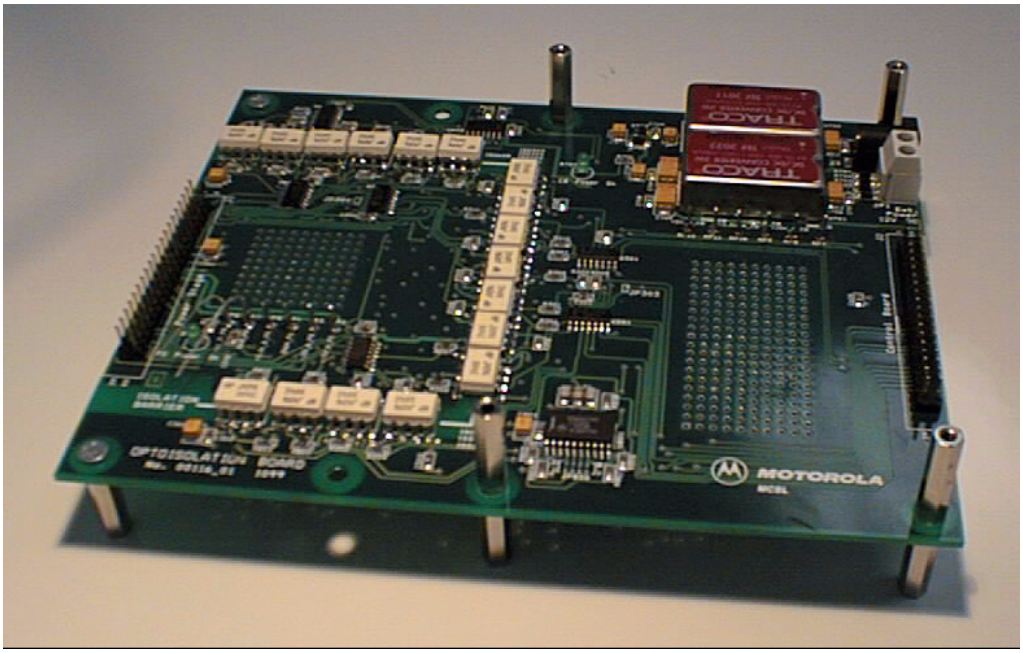


Figure 13. Optoisolation Board

For more information, refer to Reference 4.

3.3.6 3-Phase AC Induction Motor

The AC induction motor Sh 71-2A from Cantoni was used for the demo application. Detailed motor specifications are listed in the following Table 2.

Table 2. Induction Motor Specification

Parameter Name	Parameter Value
Product Category	General purpose motors
Type of motor	Sh 71-2A
Frame size	71
Degree of protection	IP 55
Insulation class	F
Type of duty	S1
Rated output [kW/HP]	0.37/0.5
Rated torque [Nm]	1.262
Number of poles	2
Synchronous speed [rpm]	3000
Rated speed [rpm]	2800
Efficiency (1/2 load) [%]	67
Efficiency (3/4 load) [%]	69
Efficiency (full load) [%]	68
Power factor (full load)	0.77
Frequency [Hz]	50
Rated voltage 1 [V]	400
Rated voltage 2 [V]	230
Current at rated voltage 1 [A]	1
Current at rated voltage 2 [A]	1.73
Star / delta start	no
Locked rotor torque / rated torque	2.2
Locked rotor current / rated current	4.4
Brakedown torque / rated torque	2.2
Sound power level [dB]	67
Sound pressure level [dB]	60
Winding connection	Y/Δ

3.3.7 Power Supply

The M523xEVB board is powered by a 9.0-V/2.7-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 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 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, and so ensures the correct cooperation of eTPU functions.

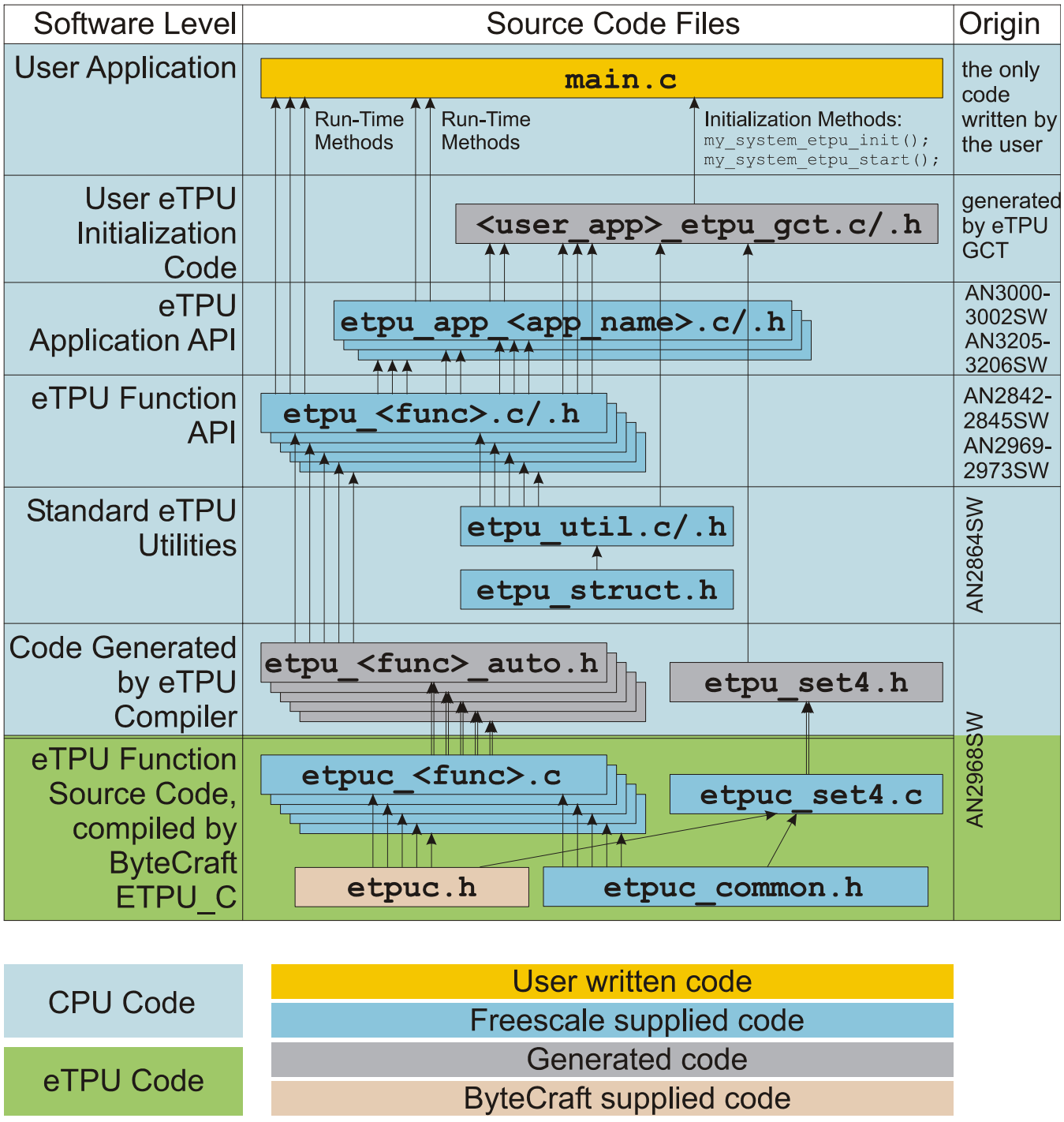


Figure 14. eTPU Project Structure

4.1 CPU Software Flowchart

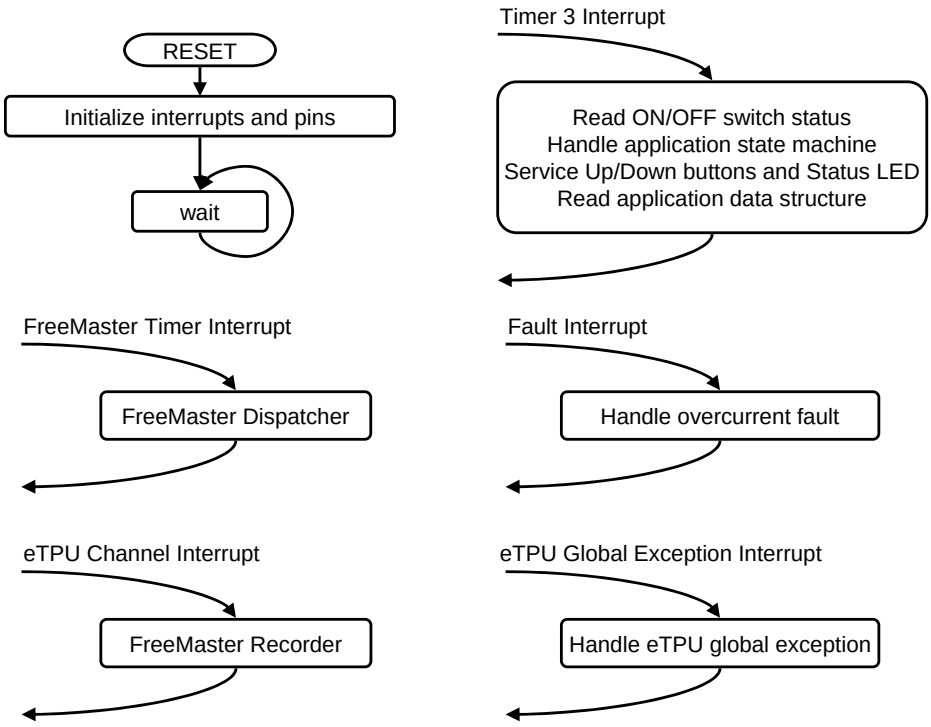


Figure 15. CPU Software Flowchart

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, one periodical eTPU channel interrupt, and two fault interrupts.

4.1.1 Initialization of Interrupts and Pins

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

4.1.2 Timer 3 Interrupt Service Routine

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

- Read the ON/OFF switch status
- Handle the application state machine

  The application state diagram is described in Section 4.2, “Application State Diagram” 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_acimvhzol1_get_data` (see Section 4.3, “eTPU Application API”).

4.1.3 FreeMASTER Interrupt Service Routine

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

4.1.4 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.

4.1.5 Fault Interrupt Service Routine

The over-current fault interrupt, which is raised by the GPIO eTPU function running on eTPU channel 4, is handled by the etpu_ch4_isr function. 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

4.1.6 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 16). 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 16), 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 16), 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.

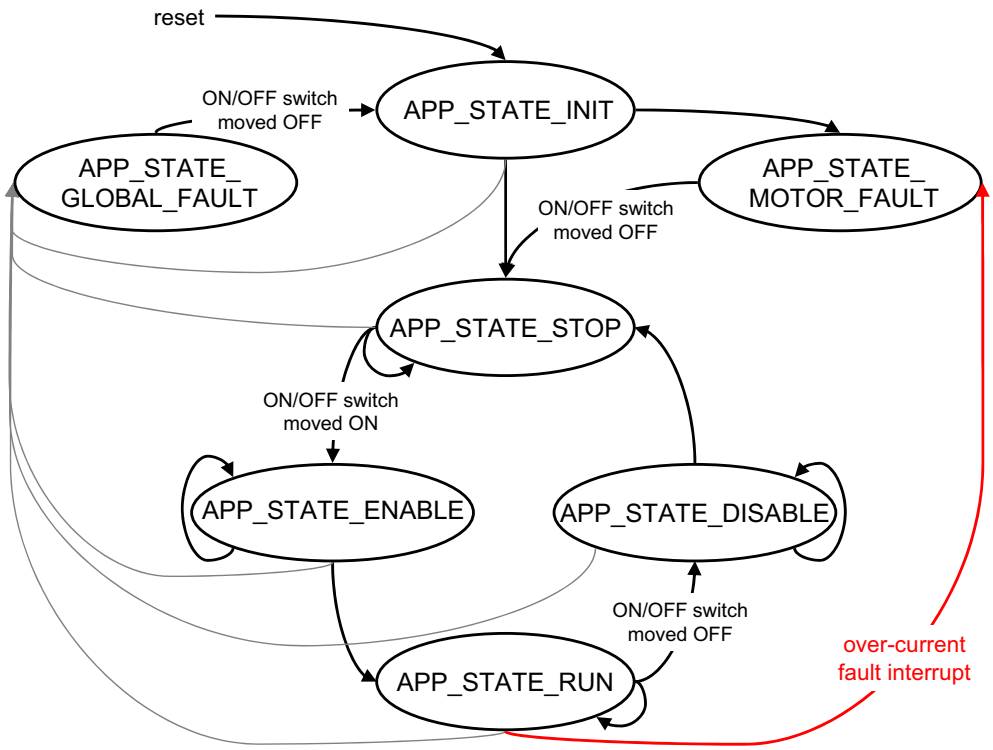


Figure 16. Application State Diagram

These 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:

- 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 the UART for FreeMASTER
- Initialize FreeMASTER
- Call `my_system_etpu_start` routine for eTPU start. At this point, the CPU and the eTPU run in parallel.
- Depending on the ON/OFF switch position, enter APP_STATE_STOP or APP_STATE_MOTOR_FAULT

4.2.1.1 Initialization and Start of eTPU Module

The eTPU module is initialized using the `my_system_etpu_init` function. Later, after initialization of all other peripherals, the eTPU is started by `my_system_etpu_start`. These functions use the general eTPU utilities and eTPU function API routines. Both the `my_system_etpu_init` and `my_system_etpu_start` functions, included in `acimvhzol1_etpu_gct.c` file, are generated by the eTPU graphical configuration tool. The eTPU graphical configuration tool can be downloaded from http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=eTPU. For more information, refer to Reference 12.

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 32
- TCR1 source = etpuclk div 2
- TCR1 prescaler = 1

  The TCR1 internal eTPU clock is set to its maximum rate of 37.5 MHz (at 150-MHz system clock), corresponding to the 27ns resolution of generated PWM signals.
- TCR2 source = etpuclk div 8
- TCR2 prescaler = 64

  The TCR2 internal eTPU clock is set to a rate of 146.484 kHz (at 150-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 5 - ACIM V/Hz control (ACIMVHZ)
- 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

  These eTPU channels are initialized by the `fs_etpu_app_acimvhzol1_init` eTPU application API function (see Section 4.3, "eTPU Application API"). The application settings are:

  — PWM phases-type is full range complementary pairs
  — PWM frequency 20kHz
  — PWM dead-time 1µs
  — Motor speed range 4000 RPM
  — Speed Ramp parameters:
  2000 ms to ramp up from zero to the maximum speed
  — Base frequency 50 Hz
  — Boost voltage percentage 10%
  — DC Bus voltage 400 V
  — Number of motor pole pairs 1

- Channel 4 - general purpose I/O (GPIO)

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

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

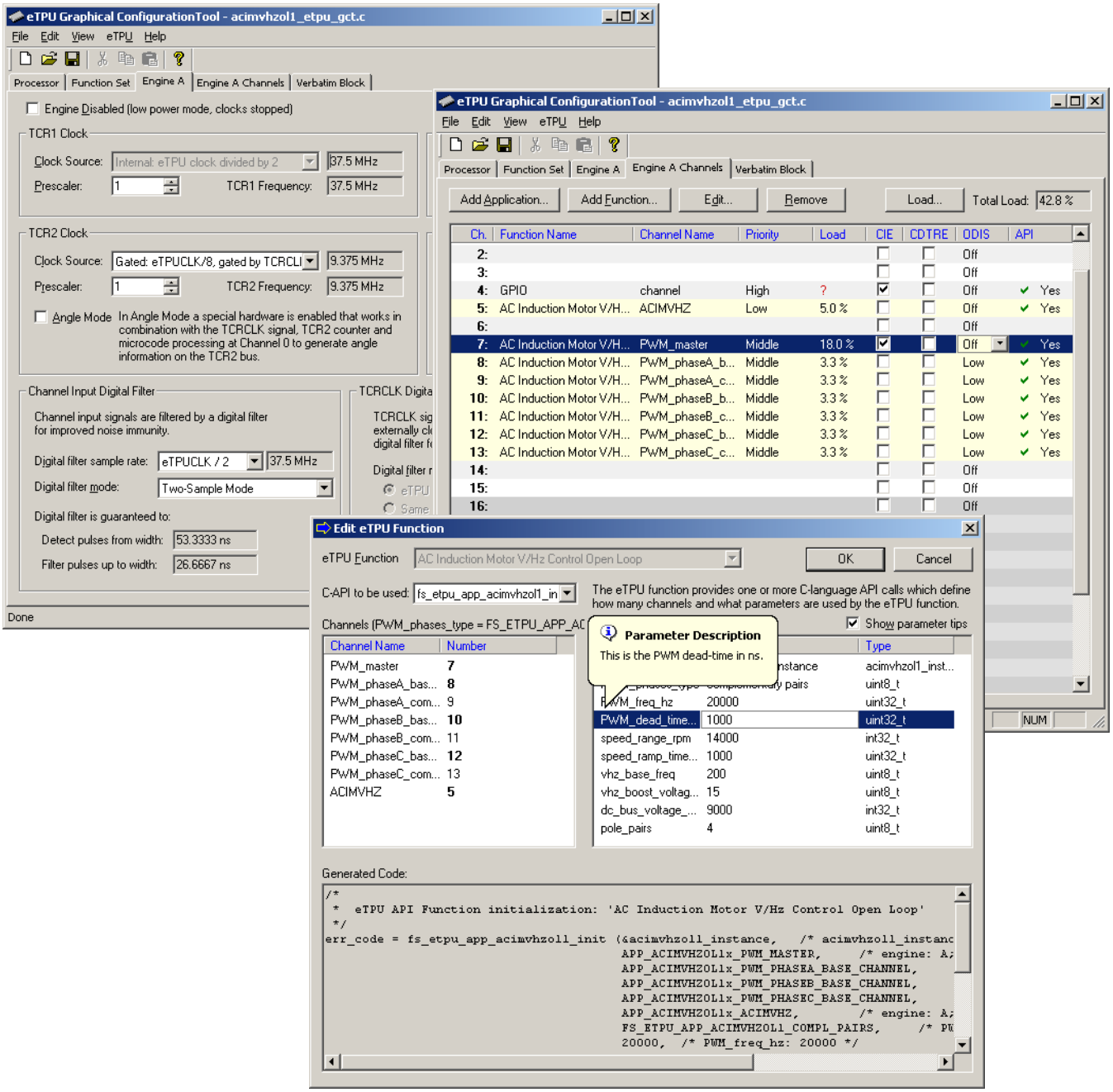


Figure 17. eTPU Configuration Using the eTPU Graphical Configuration Tool

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 FMSTR_Init() function.

4.2.2 APP_STATE_STOP

In this state, the PWM signals are disabled and the motor is off. 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. These actions switch the motor drive on:

- Reset the required speed
- Enable the generation of PWM signals

If the PWM phases were successfully enabled, the GPIO eTPU function is configured as input, interrupt on rising 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 M523xEVB or by using FreeMASTER. The latest value is periodically written to the eTPU.

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

4.2.5 APP_STATE_DISABLE

This state is passed through only. 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 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.

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 ACIMVHZOL1 is an abbreviation for ‘AC induction motor open loop 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 ACIMVHZOL1 eTPU application API is described in the following paragraphs. There are 5 basic functions added to the ACIMVHZOL1 application API. The routines can be found in the `etpu_app_acimvhzol1.c(.h)` files. All ACIMVHZOL1 application API routines will be described in order and are listed below:

- Initialization function:

```
int32_t fs_etpu_app_acimvhzol1_init(
                acimvhzol1_instance_t * acimvhzol1_instance,
                            uint8_t   PWM_master_channel,
                            uint8_t   PWM_phaseA_channel,
                            uint8_t   PWM_phaseB_channel,
                            uint8_t   PWM_phaseC_channel,
                            uint8_t   ACIMVHZ_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,
                                    uint8_t   pole_pairs)
```

- Change operation functions:

```
int32_t fs_etpu_app_acimvhzol1_enable(
                acimvhzol1_instance_t * acimvhzol1_instance,
                                    uint8_t   PWM_modulation_type)


int32_t fs_etpu_app_acimvhzol1_disable(
                acimvhzol1_instance_t * acimvhzol1_instance)


void fs_etpu_app_acimvhzol1_set_speed_required(
                acimvhzol1_instance_t * acimvhzol1_instance,
                                    int32_t   speed_required_rpm)
```

- Value return functions:

```
void fs_etpu_app_acimvhzol1_get_data(
                acimvhzol1_instance_t * acimvhzol1_instance,
                    acimvhzol1_data_t * acimvhzol1_data)
```

4.3.1 int32_t fs_etpu_acimvhzol1_init(...)

This routine initializes the eTPU channel for the AC induction motor with open loop Volts per Hertz application. This function has these parameters:

- **acimvhzol1_instance (acimvhzol1_instance_t*)** - This is a pointer to acimvhzol1_instance_t structure, which is filled by `fs_etpu_app_acimvhzol1_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 acimvhzol1_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_ACIMVHZOL1_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_ACIMVHZOL1_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_ACIMVHZOL1_COMPL_PAIRS) the complementary channel is a channel one higher.
- **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_ACIMVHZOL1_SINGLE_CHANNELS, or
  FS_ETPU_APP_ACIMVHZOL1_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.

4.3.2 int32_t fs_etpu_app_acimvhzol1_enable(...)

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

- **acimvhzol1_instance (acimvhzol1_instance_t*)** - This is a pointer to acimvhzol1_instance_t structure, which is filled by fs_etpu_app_acimvhzol1_init function.
- **PWM_modulation_type (uint8_t)** - This is the required PWM modulation type. This parameter should be assigned a values of:
  FS_ETPU_APP_ACIMVHZOL1_MOD_SIN - Pure sinewave modulation, or
  FS_ETPU_APP_ACIMVHZOL1_MOD_SIN3H - Sinewave modulation with third harmonic injection, or
  FS_ETPU_APP_ACIMVHZOL1_MOD_SVMSTD - Standard space Vector Modulation, or
  FS_ETPU_APP_ACIMVHZOL1_MOD_SVMU0N - U0N Space Vector Modulation, or
  FS_ETPU_APP_ACIMVHZOL1_MOD_SVMU7N - U7N Space Vector Modulation.

4.3.3 int32_t fs_etpu_app_acimvhzol1_disable (...)

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

- **acimvhzol1_instance (acimvhzol1_instance_t*)** - This is a pointer to acimvhzol1_instance_t structure, which is filled by fs_etpu_app_acimvhzol1_init function.

4.3.4 void fs_etpu_app_acimvhzol1_set_speed_required(...)

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

- **acimvhzol1_instance (acimvhzol1_instance_t*)** - This is a pointer to acimvhzol1_instance_t structure, which is filled by fs_etpu_app_acimvhzol1_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_acimvhzol1_get_data(...)

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

- **acimvhzol1_instance (acimvhzol1_instance_t*)** - This is a pointer to acimvhzol1_instance_t structure, which is filled by fs_etpu_app_acimvhzol1_init function.
- **acimvhzol1_data (acimvhzol1_data_t*)** - This is a pointer to acimvhzol1_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. These paragraphs describe the functionality of each block.

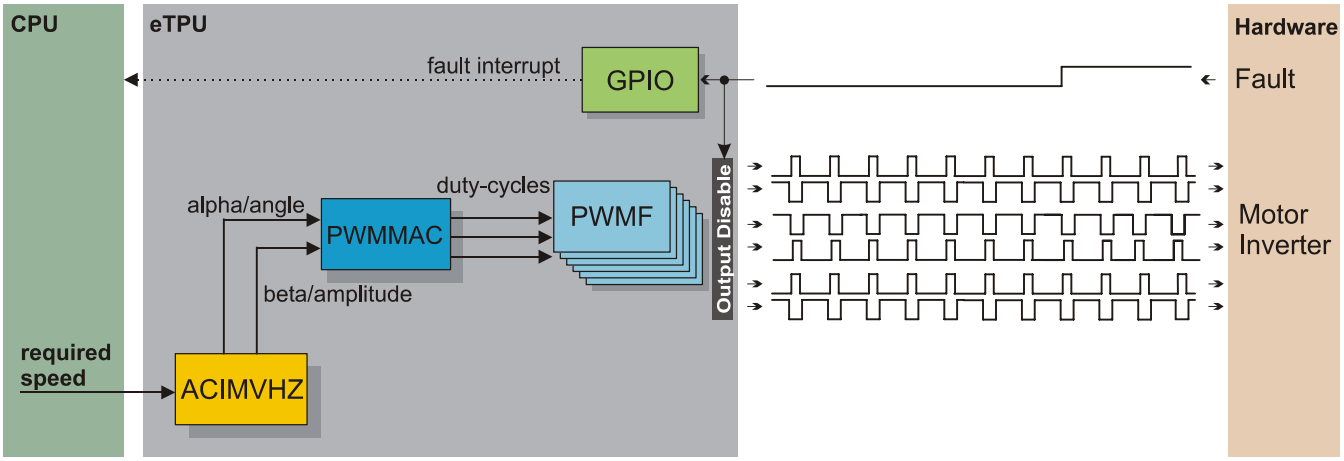


Figure 18. Block Diagram of eTPU Processing

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 (channels 16 to 32).

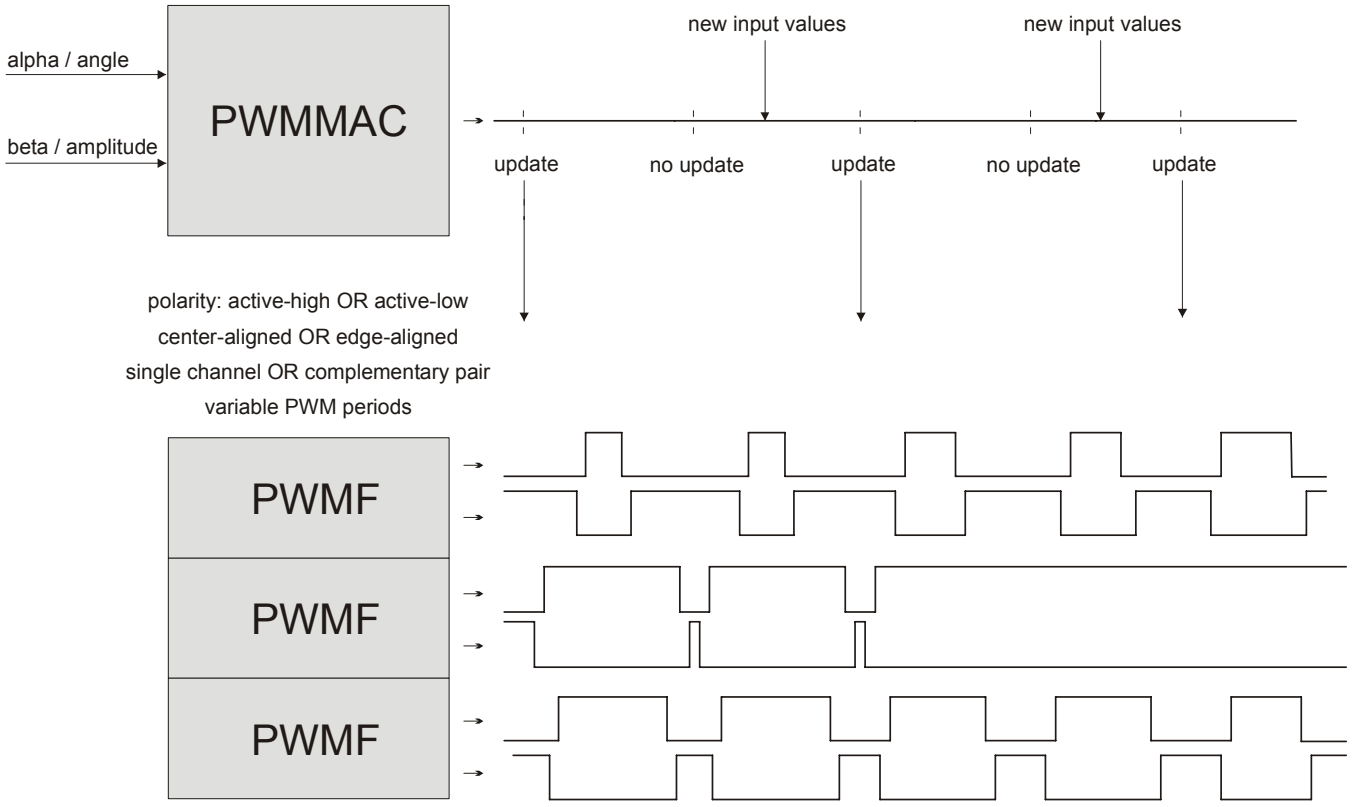


Figure 19. Functionality of PWMMAC+PWMF

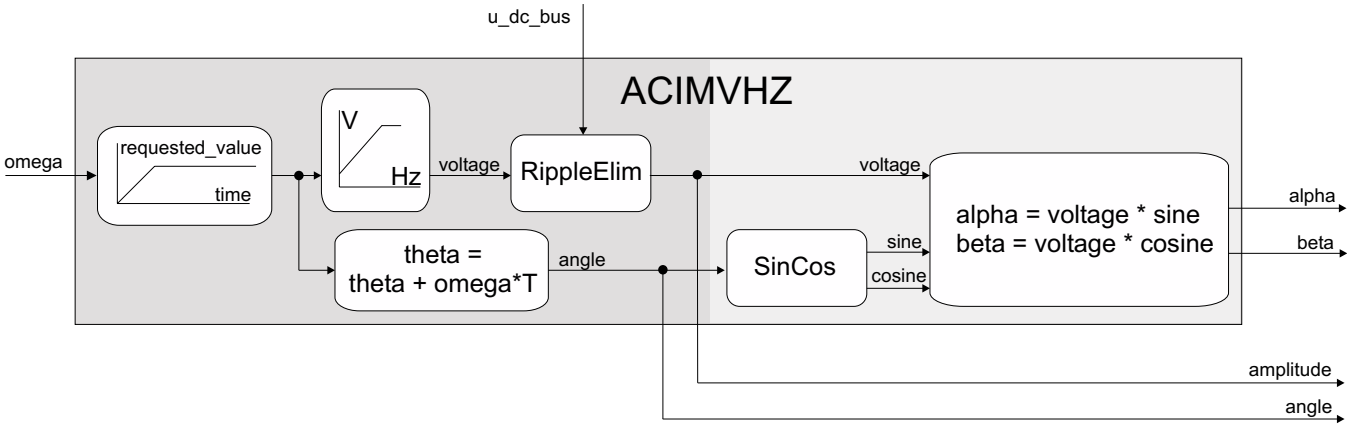
For more details about the PWMMAC and PWMF eTPU functions, refer to Reference 9.

4.4.2 ACIM Volts per Hertz (ACIMVHZ)

The ACIM Volts per Hertz eTPU function is not intended to 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 8.

4.4.3 General Purpose I/O (GPIO)

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

- Input mode - update periodically
- Input mode - update on transition - either edge
- Input mode - update on transition - falling edge

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

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

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

4.5 eTPU Timing

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

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

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

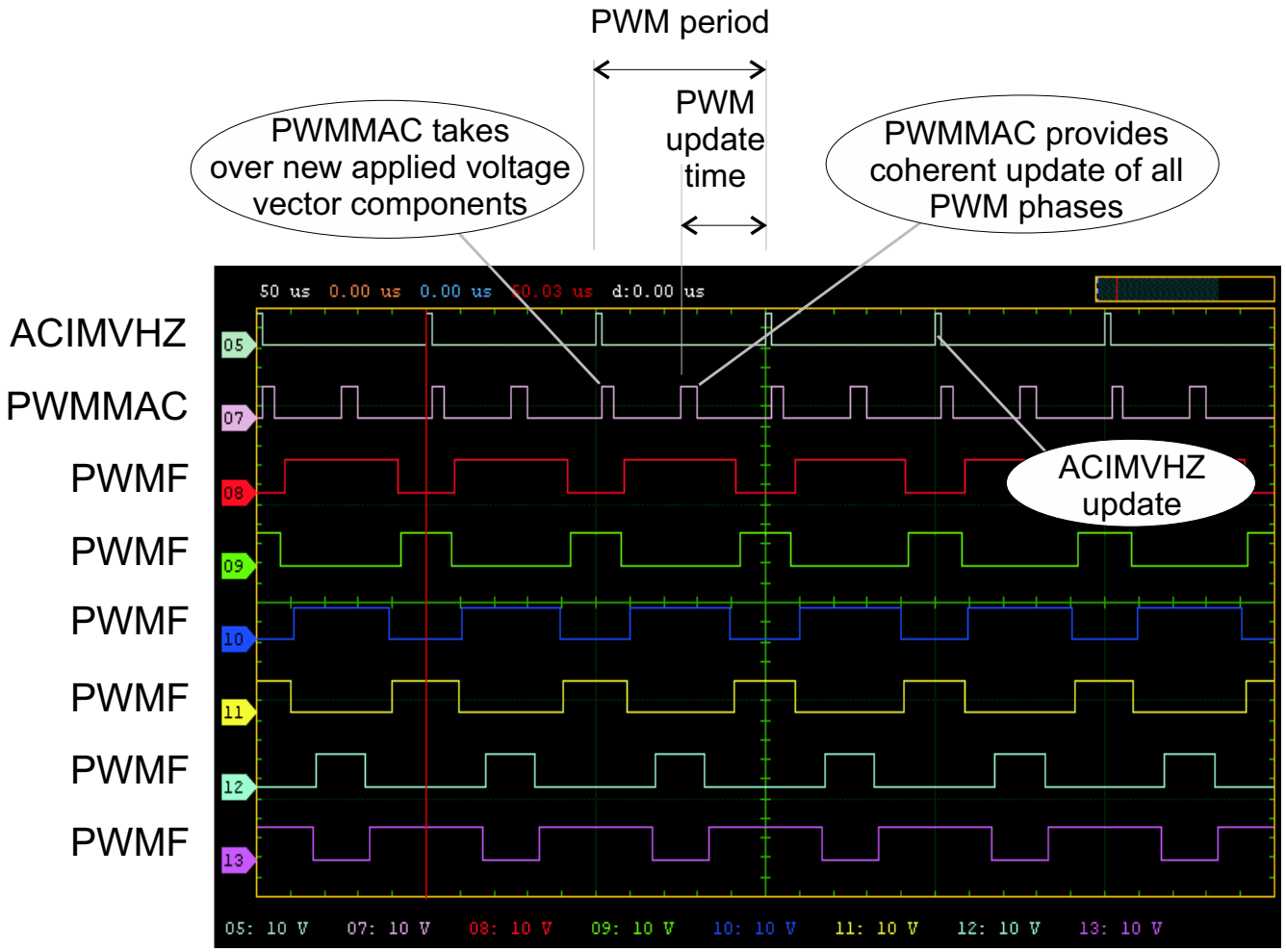


Figure 20. Logic Analyzer Screenshot and Explanation of PWMMAC Timing

Figure 20 explains the application eTPU timing. The logic analyzer screen-shot depicts a typical situation described below.

Figure 20 shows all input and output eTPU signals. The signals are numbered according to eTPU channels. The time scale is adjusted so that the whole picture shows a time section of 6 PWM periods.

Signal 5 is generated by the ACIM Volts per Hertz control (ACIMVHZ) eTPU function. The ACIMVHZ update is performed every PWM period.

Signal 7 is generated by the PWM master for AC motors (PWMMAC) eTPU function. Its pulses determine the activity of the PWMMAC. Immediately after 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.

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

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 $1.0 - 2^{-23}$.

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

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

where:

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

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

6 Microprocessor Usage

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

Table 3. Memory Usage in Bytes

Memory	Available	Used
Flash (external)	2M	25,040
RAM	6K	5,784
eTPU code RAM	6K	5,864
eTPU data RAM	1.5K	688

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

- According to Reference 9, the maximum eTPU load produced by PWM generation is 1384 eTPU cycles per one PWM period. The PWM frequency is set to 20 kHz, thus the PWM period is 3750 eTPU cycles (eTPU module clock is 75 MHz, half of the 150-MHz CPU clock).

- According to Reference 8, the ACIM Volts per Hertz control calculation takes 186 eTPU cycles. The calculation is performed every PWM period.
- The GPIO function processing does not affect the eTPU time load.

The average time load takes 41.9%. 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.

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 ColdFire MCF523x, which results in a CPU independent motor drive. Lastly, the demo application is targeted at the MCF523x family of devices, but it could be easily reused with any device that has an eTPU.

8 References

Table 4. References

1. *MCF5235 Reference Manual*, MCF5235RM
2. *M523xEVB User's Manual*, M5235EVBUM
3. *3-Phase AC/BLDC High-Voltage Power Stage User's Manual* MEMC3PBLDCPSUM, http://www.freescale.com, search keyword "ECOPTHIVACBLDC" refine "Design Tools"
4. *Embedded Motion Optoisolation Board User's Manual* MEMCOBUM, http://www.freescale.com, search keyword "ECOPT" refine "Design Tools"
5. 3-Phase AC Induction Motor Sh 71-2A, Cantoni Motor http://www.cantonimotor.com
6. FreeMASTER web page, http://www.freescale.com, search keyword "FreeMASTER"
7. *Enhanced Time Processing Unit Reference Manual*, ETPURM
8. "Using the ACIM Volts per Hertz (ACIMVHZ) eTPU Function" AN2971
9. "Using the AC Motor Control PWM eTPU Functions," AN2969
10. "Using the General Purpose I/O eTPU functions (GPIO)," AN2850
11. "Using the AC Motor Control eTPU Function Set (set4)," AN2968
12. eTPU Graphical Configuration Tool, http://www.freescale.com, search keyword "ETPUGCT"

9 Revision History

Table 5 provides a revision history of this document.

Table 5. 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

AN3000
Rev. 0
06/2006