Freescale Semiconductor
Application Note

AN2971
Rev. 2, 4/2006

Using the ACIM Volts per Hertz (ACIMVHZ) eTPU Function

Covers the MCF523*x*, MPC5500, and all eTPU-Equipped Devices

by: Milan Brejl
System Application Engineer, Roznov Czech System Center
Michal Princ
System Application Engineer, Roznov Czech System Center

Table of Contents

1 Introduction

The ACIM volts per hertz (ACIMVHZ) enhanced time processor unit (eTPU) function is one of the functions included in the AC motor control eTPU function set (set4). This application note provides simple C interface routines to the ACIMVHZ eTPU function. The routines are targeted at the MCF523x and MPC5500 families of devices, but can easily be used with any device that has an eTPU.

2 Function Overview

The purpose of the ACIMVHZ function is to perform the simple V/Hz control of the AC induction motor. The ACIMVHZ calculates applied voltage vector components based on required speed (omega) and defined V/Hz ramp. The output applied voltage vector components can be either:

- Amplitude and angle for sinusoidal modulation
- Alpha and beta for space vector modulation

freescale semiconductor

The ACIMVHZ function optionally enables passing the required speed through a speed ramp to slow down step changes of the input parameter.

The ACIMVHZ function does not generate any drive signal, and can be executed even on an eTPU channel not connected to an output pin. If connected to an output pin, the ACIMVHZ function turns the pin high and low, so that the high-time identifies the period of time in which the ACIMVHZ execution is active. In this way, the ACIMVHZ function, as with many of the motor-control eTPU functions, supports checking eTPU timing using an oscilloscope.

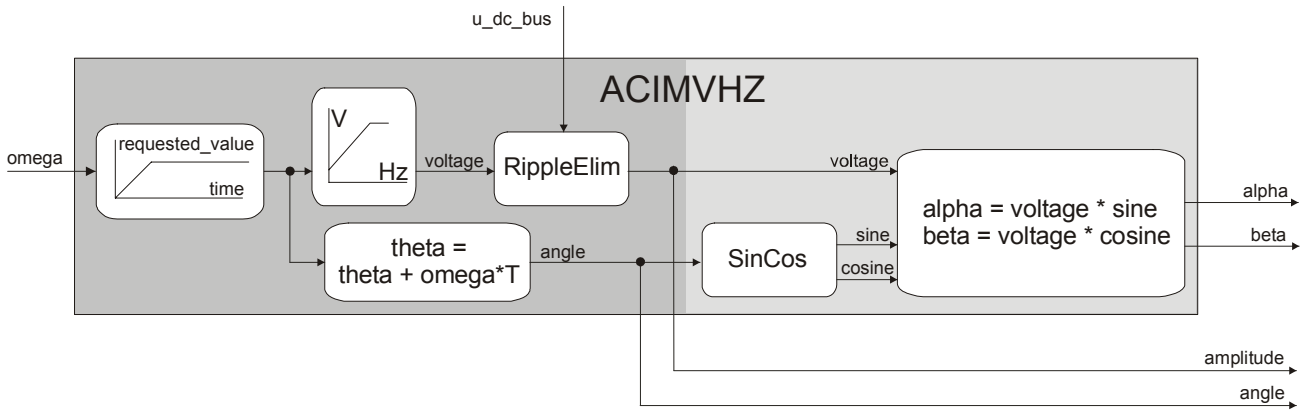

Figure 1. Functionality of ACIMVHZ

3 Function Description

The ACIMVHZ eTPU function performs the calculation of the output applied voltage vector components in the following order:

- **Optionally passes the required speed through the speed ramp.**

  This procedure refines the step changes towards the desired value. The ramp performs a linear ramp generation algorithm (see Figure 2). If the *desired_value* is greater than the *actual_value*, the ramp output steps up by *ramp_increment_up* until the *desired_value* is reached, at which point the *desired_value* is returned. And vice versa: if the *desired_value* is less than *actual_value*, the ramp output steps down by *ramp_increment_down* until the *desired_value* is reached.

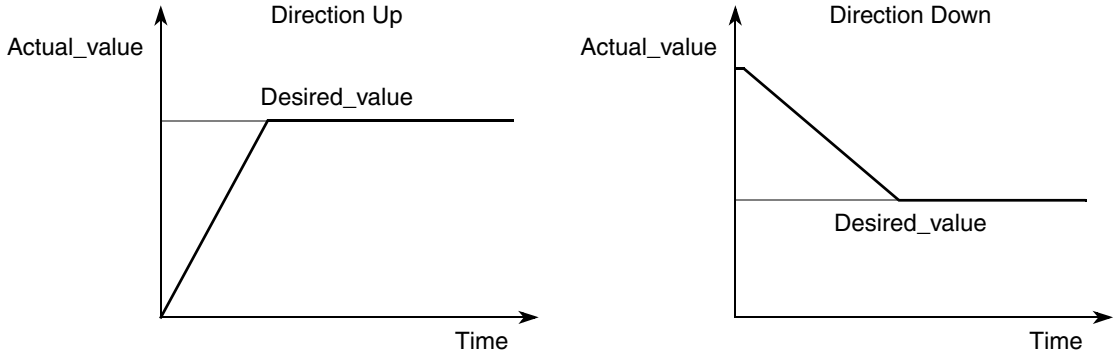

Figure 2. Ramp Generation Algorithm

- **Updates applied voltage amplitude using V/Hz ramp.**

  The volts per hertz control method is a commonly used AC induction motor control technique. Applied voltage amplitude is calculated based on the required frequency and the defined V/Hz ramp. The typical volts per hertz ramp of a 3-phase AC induction motor is illustrated in Figure 3.

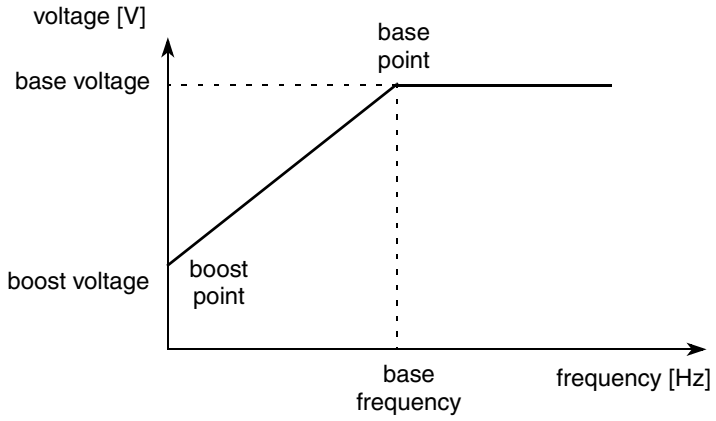

Figure 3. Volts per Hertz Ramp

The volts per hertz ramp is defined by these parameters:

— Base point—defined by base frequency (usually 50 Hz or 60 Hz) and base voltage

— Boost point—defined by boost voltage as a percentage of base voltage

The applied voltage 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 proportional 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 characteristic is defined by base point and boost point. To get the motor excited at startup, the minimal boost voltage is required. Between the boost point and the base point, the motor operates at optimum excitation called ‘constant torque operation’ because of the constant V_s/f_s ratio. Above this point, the motor operates under-excited, called ‘constant power operation’ because of the rated voltage limit.

- **Eliminates DC-bus ripples.**

  The voltage ripple elimination process eliminates the influence of the DC-bus voltage ripples on the generated phase voltage sinewaves. In fact, it lowers the 50- or 60-Hz acoustic noise of the motor. The imperfections in the DC bus voltage are eliminated by the formula shown in the following equation:

$$voltage = \begin{cases} \dfrac{inv\_mod\_index \cdot voltage}{u\_dc\_bus\_actual/2} & \text{if } |inv\_mod\_index \cdot voltage| < \dfrac{u\_dc\_bus\_actual}{2} \\ sign(voltage) \cdot 1.0 & \text{otherwise} \end{cases}$$

where the y = sign (x) function is defined as follows:

$$y = \begin{cases} 1.0 & \text{if } x \geq 0 \\ -1.0 & \text{otherwise} \end{cases}$$

Where:

voltage is calculated value of applied motor voltage, in [V].

u_dc_bus_actual is actual measured value of the DC-bus voltage, in [V].

inv_mod_index is inverse modulation index; depends on the selected modulation technique, in [-].

The following figures depict ripple elimination functionality. Due to variations made in the actual DC-bus voltage, the ripple elimination algorithm influences the duty cycles that are generated using the standard space vector modulation technique.

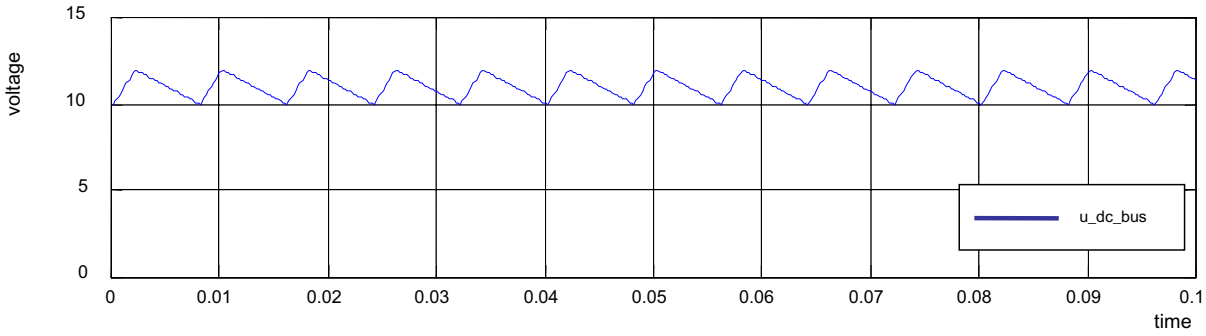

Figure 4. Measured Voltage on the DC-Bus

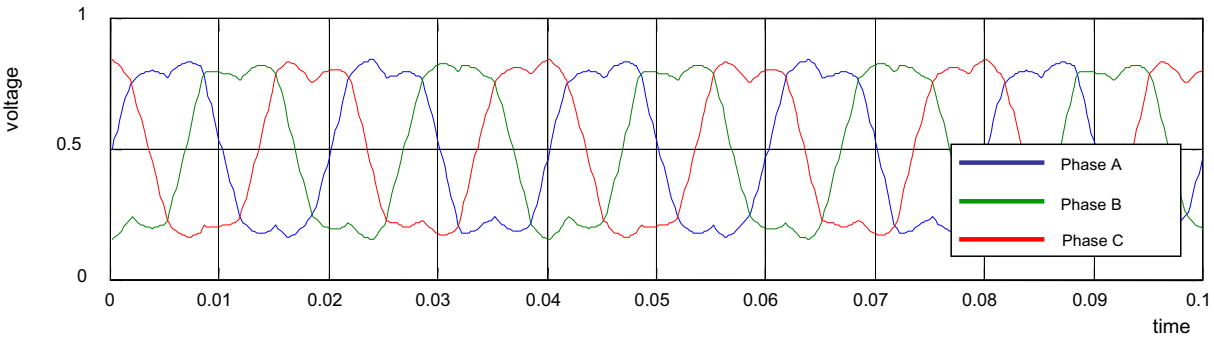

Figure 5. Standard Space Vector Modulation with Elimination of the DC_bus Ripple

- **Updates angle of the output vector.**

The angle position of output voltage vector *theta* is updated each ACIMVHZ update thread. The new *theta* value is calculated based on the following formula:

$$theta = theta + (omega \times period)$$

Where:

theta is angle position of output voltage vector, in [rad].

omega is desired value of speed after ramp, in [rad / s].

period is ACIMVHZ update period, in [s].

- **Determines sin(theta), cos(theta), α and β in case of α, β vector components calculation mode.**

  The calculation of the sine and cosine of the theta angle is performed based on a sine look-up table. The look-up table contains 129 samples of the first sine quadrant. The other quadrant values are obtained by mirroring and negating the first quadrant values. The look-up table is a two-stage process: first, the two values of the nearest angles to the desired one are fetched from the look-up table, then the linear interpolation of these two values is calculated.

  The α and β vector components are calculated as follows:

$$\alpha = voltage \times cos(theta)$$
$$\beta = voltage \times sin(theta)$$

  Where:

  α is alpha component, in [V].

  β is beta component, in [V].

  voltage is V/Hz ramp output (optionally after ripple elimination), in [V].

  theta is angle position of output voltage vector, in [rad].

The ACIMVHZ function update, in which all V/Hz control calculations are performed, can be executed periodically, or by another process:

- **Master mode**

  The ACIMVHZ update is executed periodically with a given period.

- **Slave mode**

  The ACIMVHZ update is executed by the analog sensing for AC motors (ASAC) eTPU function, other eTPU function, or by the CPU.

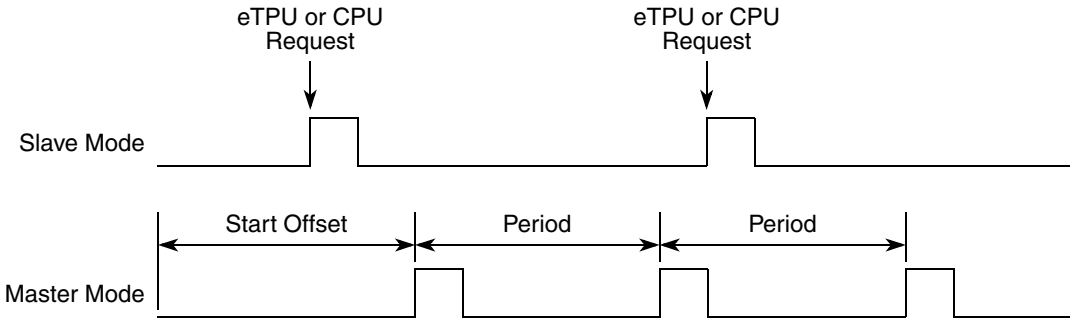

Figure 6. ACIMVHZ Updates in Slave Mode and Master Mode

3.1 Interrupts

The ACIMVHZ function generates an interrupt service request to the CPU every *n*-th update. The number of updates, after which an interrupt service request is generated, is a function parameter.

3.2 Performance

Like all eTPU functions, the ACIMVHZ function performance in an application is to some extent dependent upon the service time (latency) of other active eTPU channels. This is due to the operational nature of the scheduler.

The influence of the ACIMVHZ function on the overall eTPU performance can be expressed by this parameter:

- Maximum eTPU busy-time per one update period

  This value, compared to the update period value, determines the proportional load on the eTPU engine caused by the ACIMVHZ function.

Table 1 lists the maximum eTPU busy-times per update period in eTPU cycles that depend on the ACIMVHZ mode, ripple elimination configuration, and the output vector type option.

Table 1. Maximum eTPU Busy-Times

Mode	Ripple Elimination Configuration	Output Vector Type	Maximum eTPU Busy-time per One Update Period [eTPU Cycles]
Master mode	Speed Ramp OFF	Alpha-Beta calculation	290
Master mode	Speed Ramp OFF	Amplitude-Angle calculation	154
Master mode	Speed Ramp ON	Alpha-Beta calculation	322
Master mode	Speed Ramp ON	Amplitude-Angle calculation	186
Slave mode	Speed Ramp OFF	Alpha-Beta calculation	282
Slave mode	Speed Ramp OFF	Amplitude-Angle calculation	146
Slave mode	Speed Ramp ON	Alpha-Beta calculation	310
Slave mode	Speed Ramp ON	Amplitude-Angle calculation	174

On MPC5500 devices, the eTPU module clock is equal to the CPU clock. On MCF523x devices, it is equal to the peripheral clock (which is a half of the CPU clock). For example, on a 132-MHz MPC5554, the eTPU module clock is 132 MHz, and one eTPU cycle takes 7.58ns. On a 150-MHz MCF5235, the eTPU module clock is only 75 MHz, and one eTPU cycle takes 13.33ns.

The performance is influenced by the compiler efficiency. The above numbers, measured on the code compiled by eTPU compiler version 1.0.7, are given for guidance only and are subject to change. For up-to-date information, refer to the information provided in the particular eTPU function set release available from Freescale.

4 C Level API for Function

The following routines provide easy access for an application developer to the ACIMVHZ function. Use of these functions eliminates the need to directly control the eTPU registers.

There are 11 functions added to the ACIMVHZ application programming interface (API). The routines can be found in the `etpu_acimvhz.h` and `etpu_acimvhz.c` files, which should be linked with the top level development file(s).

Figure 7 shows the ACIMVHZ API state flow and lists API functions which can be used in each of its states.

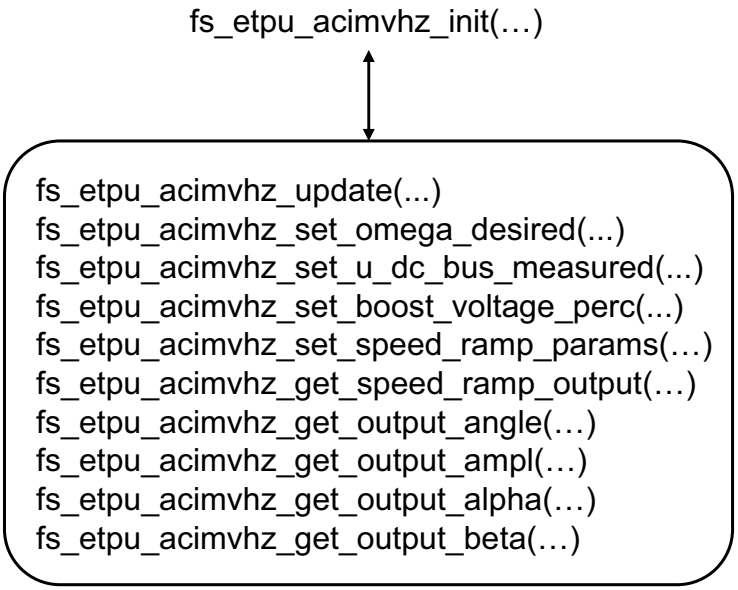

Figure 7. ACIMVHZ API State Flow

All ACIMVHZ API routines are described in order and listed below:

- Initialization functions:

```
int32_t fs_etpu_acimvhz_init( uint8_t channel,
                              uint8_t priority,
                              uint8_t mode,
                              uint8_t speed_ramp_config,
                              uint8_t outputs_type,
                             uint24_t period,
                             uint24_t start_offset,
                             uint24_t services_per_irq,
                              uint8_t base_freq,
                              uint8_t boost_voltage_perc,
                              int32_t dc_bus_voltage_mv,
                              int32_t dc_bus_voltage_range_mv,
                              int24_t inv_mod_index,
                              uint8_t pole_pairs,
                             uint24_t omega_range_rpm,
```

```
                                    speed_ramp_params_t* p_speed_ramp_params,
                                                uint8_t output_chan,
                                               uint16_t output_offset,
                                                uint8_t link_chan)
```

- Change operation functions:

```
int32_t fs_etpu_acimvhz_update(uint8_t channel)

int32_t fs_etpu_acimvhz_set_omega_desired(uint8_t  channel,
                                          fract24_t  omega_desired)

int32_t fs_etpu_acimvhz_set_u_dc_bus_measured(uint8_t  channel,
                                              ufract24_t  u_dc_bus_measured)

int32_t fs_etpu_acimvhz_set_boost_voltage_perc(uint8_t channel,
                                               uint8_t boost_voltage_perc,
                                               int32_t dc_bus_voltage_mv,
                                               int32_t dc_bus_voltage_range_mv)

int32_t fs_etpu_acimvhz_set_speed_ramp_params(uint8_t  channel,
                                      speed_ramp_params_t* p_speed_ramp_params)
```

- Value return functions:

```
fract24_t fs_etpu_acimvhz_get_speed_ramp_output(uint8_t channel)

ufract24_t fs_etpu_acimvhz_get_output_angle(uint8_t channel)

fract24_t fs_etpu_acimvhz_get_output_ampl(uint8_t channel)

fract24_t fs_etpu_acimvhz_get_output_alpha(uint8_t channel)

fract24_t fs_etpu_acimvhz_get_output_beta(uint8_t channel)
```

4.1 Initialization Function

4.1.1 int32_t fs_etpu_acimvhz_init(...)

This routine is used to initialize the eTPU channel for the ACIMVHZ function. It function has these parameters:

- **channel (uint8_t)**—The ACIMVHZ channel number; should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **priority (uint8_t)**—The priority to assign to the ACIMVHZ function; should be assigned one of these value:
  - FS_ETPU_PRIORITY_HIGH
  - FS_ETPU_PRIORITY_MIDDLE
  - FS_ETPU_PRIORITY_LOW
- **mode (uint8_t)**—The function mode; should be assigned one of these values:
  - FS_ETPU_ACIMVHZ_MASTER
  - FS_ETPU_ACIMVHZ_SLAVE
- **speed_ramp_config (uint8_t)**—The required configuration of the speed ramp; should be assigned one of these values:
  - FS_ETPU_ACIMVHZ_SPEED_RAMP_OFF
  - FS_ETPU_ACIMVHZ_SPEED_RAMP_ON
- **outputs_type (uint8_t)**—The required type of the outputs; should be assigned one of these values:
  - FS_ETPU_ACIMVHZ_OUTPUTS_ANGLE_AMPL
  - FS_ETPU_ACIMVHZ_OUTPUTS_ALPHA_BETA
- **period (uint24_t)**—The update period, as a number of TCR1 clocks; applies in the master mode only (mode=FS_ETPU_ACIMVHZ_MASTER).
- **start_offset (uint24_t)**—Used to synchronize various eTPU functions that generate a signal. The first ACIMVHZ update starts the start_offset TCR1 clocks after initialization. This parameter applies in the master mode only (mode=FS_ETPU_ACIMVHZ_MASTER).
- **services_per_irq (uint24_t)**—Defines the number of updates, after which an interrupt service request is generated to the CPU.
- **base_freq (uint8_t)**—Defines the motor base frequency in Hz. This parameter should be assigned a value of 50 (50 Hz) or 60 (60 Hz).
- **boost_voltage_perc (uint8_t)**—Defines the motor boost voltage as a percentage of base DC-bus voltage. This parameter should be assigned a value of 0 to 100.
- **dc_bus_voltage_mv (int32_t)**—Defines the base DC-bus voltage in mV. This parameter together with *base_freq* parameter defines the base point of volts per hertz ramp.

- **dc_bus_voltage_range_mv (int32_t)**—Maximum measurable DC-bus voltage in mV. This parameter applies only when the actual DC-bus voltage is measured by AD converter. It enables to eliminate ripples on DC-bus. If the actual DC-bus voltage is not measured, set this parameter to zero.
- **inv_mod_index (int24_t)**—Defines the inverse modulation index, which depends on the type of modulation technique being used by the PWMMAC. This parameter should be assigned one of these values:
  - FS_ETPU_ACIMVHZ_INVMODINDEX_SINE
  - FS_ETPU_ACIMVHZ_INVMODINDEX_SIN3H
  - FS_ETPU_ACIMVHZ_INVMODINDEX_SVM
- **pole_pairs (uint8_t)**—Defines the number of motor pole-pairs.
- **omega_range_rpm (uint24_t)**—Defines the motor speed range in [rpm].
- **p_speed_ramp_params (speed_ramp_params_t*)**—The pointer to a speed_ramp_params_t structure.
- **output_chan (uint8_t)**—ACIMVHZ writes outputs to a recipient function's input parameters. This is the recipient function channel number. 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **output_offset (uint16_t)**—ACIMVHZ writes outputs to a recipient function's input parameters. This is the first input parameter offset of the recipient function. Function parameter offsets are defined in etpu_<func>_auto.h file.
- **link_chan (uint8_t)**—The number of the channel that receives a link after ACIMVHZ updates output. If ACIMVHZ updates PWM duty-cycles, it should be a PWMMAC channel. 0-31 for ETPU_A, and 64-95 for ETPU_B.

4.2 Change Operation Functions

4.2.1 int32_t fs_etpu_acimvhz_update(uint8_t channel)

This function executes the ACIMVHZ update and has this parameter:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

4.2.2 int32_t fs_etpu_acimvhz_set_omega_desired(uint8_t channel, fract24_t omega_desired)

This function changes the desired value, as a portion of the maximum value. It has these parameters:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **omega_desired (fract24_t)**—Desired input value, in range MIN24 to MAX24. The sign determines the rotor direction.

4.2.3 int32_t fs_etpu_acimvhz_set_u_dc_bus_measured(uint8_t channel, ufract24_t u_dc_bus_measured)

This function sets the value of actual DC-bus voltage, as a portion of the AD convertor range and can be used in case a DMA transfer of the value from AD converter to eTPU is not used. It has these parameters:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **u_dc_bus_measured (ufract24_t)**—The actual value of DC-bus voltage, as an unsigned 24-bit portion of the AD convertor range.

4.2.4 int32_t fs_etpu_acimvhz_set_boost_voltage_perc(uint8_t channel, uint8_t boost_voltage_perc, int32_t dc_bus_voltage_mv, int32_t dc_bus_voltage_range_mv)

This function changes the V/Hz ramp boost voltage value, as a percentage of the DC-bus voltage and has these parameters:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **boost_voltage_perc (uint8_t)**—Defines the motor boost voltage as a percentage of base DC-bus voltage. This parameter should be assigned a value of 0 to 100.
- **dc_bus_voltage_mv (int32_t)**—Defines the base DC-bus voltage in mV. This parameter together with *base_freq* parameter defines the base point of volts per hertz ramp.
- **dc_bus_voltage_range_mv (int32_t)**—Maximum measurable DC-bus voltage in mV. This parameter applies only when the actual DC-bus voltage is measured by AD converter. It enables to eliminate ripples on DC-bus. If the actual DC-bus voltage is not measured, set this parameter to zero.

4.2.5 int32_t fs_etpu_acimvhz_set_speed_ramp_params(uint8_t channel, speed_ramp_params_t* p_speed_ramp_params)

This function changes the RAMP parameter values and has these parameters:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.
- **p_speed_ramp_params (speed_ramp_params_t*)**—The pointer to a speed_ramp_params_t structure.

4.3 Value Return Function

4.3.1 fract24_t fs_etpu_acimvhz_get_speed_ramp_output (uint8_t channel)

This function returns the speed ramp output and has this parameter:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

4.3.2 ufract24_t fs_etpu_acimvhz_get_output_angle(uint8_t channel)

This function returns the output applied voltage vector angle. It can be used only when outputs_type = FS_ETPU_ACIMVHZ_OUTPUTS_ANGLE_AMPL and has this parameter:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

4.3.3 fract24_t fs_etpu_acimvhz_get_output_ampl(uint8_t channel)

This function returns the output applied voltage vector amplitude and can be used only when outputs_type = FS_ETPU_ACIMVHZ_OUTPUTS_ANGLE_AMPL. It has this parameter:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

4.3.4 fract24_t fs_etpu_acimvhz_get_output_alpha(uint8_t channel)

This function returns the output applied voltage vector orthogonal component alpha. It can be used only when outputs_type = FS_ETPU_ACIMVHZ_OUTPUTS_ALPHA_BETA and has this parameter:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

4.3.5 fract24_t fs_etpu_acimvhz_get_output_beta(uint8_t channel)

This function returns the output applied voltage vector orthogonal component beta. It can be used only when outputs_type = FS_ETPU_ACIMVHZ_OUTPUTS_ALPHA_BETA and has this parameter:

- **channel (uint8_t)**—The ACIMVHZ channel number; must be assigned the same value as the channel parameter of the initialization function was assigned.

5 Example Use of Function

5.1 Demo Applications

Using the ACIMVHZ eTPU function is demonstrated in the following application note:

- "AC Induction Motor V/Hz Control, driven by eTPU on MCF523x," AN3000.

For a detailed description of the demo application, refer to the above application note.

5.1.1 Function Calls

The ACIMVHZ function is configured to master mode and periodically calculates amplitude and angle as output parameters. The desired speed is provided by the CPU, and a step change to the desired speed is refined by the ramp algorithm. The ripple elimination is off. The controller output points to a PWMMAC input, so that it controls the duty-cycle of PWM phases.

```
/*****************************************************************************
* Parameters
*****************************************************************************/
uint8_t PWM_master_channel = 7;
uint8_t ACIMVHZ_channel = 5;
uint32_t PWM_freq_hz = 20000;
uint32_t speed_range_rpm = 4000;
uint32_t speed_ramp_time_ms = 1000;
uint32_t vhz_base_freq = 50;
uint32_t vhz_boost_voltage_perc = 15;
int32_t dc_bus_voltage_mv = 230000;
int32_t dc_bus_voltage_range_mv = 0;
uint8_t pole_pairs = 1;
```

```
/*****************************************************************************
* Initialize ACIM V/Hz Controller
*****************************************************************************/
/*****************************************************************************
 * 1) Define Speed Ramp Parameters
 *****************************************************************************/
   speed_ramp_params.ramp_incr_up   =
(0x800000/PWM_freq_hz)*1000/speed_ramp_time_ms;
   speed_ramp_params.ramp_incr_down =
(0x800000/PWM_freq_hz)*1000/speed_ramp_time_ms;

/*****************************************************************************
 * 2) Initialize ACIMVHZ channel
 *****************************************************************************/
   err_code = fs_etpu_acimvhz_init(
      ACIMVHZ_channel,/* channel */
      FS_ETPU_PRIORITY_LOW,/* priority */
      FS_ETPU_ACIMVHZ_MASTER,/* mode */
      FS_ETPU_ACIMVHZ_SPEED_RAMP_ON,/* speed_ramp_config */
      FS_ETPU_ACIMVHZ_OUTPUTS_ANGLE_AMPL,/* outputs_type */
      etpu_a_tcr1_freq/PWM_freq_hz,/* period */
      10000,/* start_offset */
      0,/* services_per_irq */
      vhz_base_freq,/* base_freq */
      vhz_boost_voltage_perc,/* boost_voltage_perc */
      dc_bus_voltage_mv, /* dc_bus_voltage_mv */
      dc_bus_voltage_range_mv, /* dc_bus_voltage_range_mv */
      FS_ETPU_ACIMVHZ_INVMODINDEX_SIN3H,/* inv_mod_index */
      pole_pairs,/* pole_pairs */
      speed_range_rpm,/* omega_range_rpm */
      &speed_ramp_params,/* p_speed_ramp_params */
      0,/* u_dc_bus_norm */
      PWM_master_channel,/* output_chan */
      FS_ETPU_PWMMAC_INPUTS_OFFSET,/* output_offset */
      PWM_master_channel/* link_chan */
   );
/*****************************************************************************
* Set required speed
*****************************************************************************/
fs_etpu_acimvhz_set_omega_desired(ACIMVHZ_channel,
          (speed_required_rpm<<15)/speed_range_rpm <<8);
```

6 Summary and Conclusions

This application note provides the user with a description of the ACIMVHZ eTPU function. The simple C interface routines to the ACIMVHZ eTPU function enable easy implementation of the ACIMVHZ in applications. The demo application is targeted at the MCF523*x* family of devices, but it can easily be reused with any device that has an eTPU.

6.1 References

1. “The Essential of Enhanced Time Processing Unit,” AN2353
2. “General C Functions for the eTPU,” AN2864
3. “Using the AC Motor Control eTPU Function Set (set4),” AN2968
4. *Enhanced Time Processing Unit Reference Manual*, ETPURM
5. eTPU Graphical Configuration Tool, http://www.freescale.com/etpu, ETPUGCT
6. “Using the AC Motor Control PWM eTPU Functions,” AN2969
7. “AC Induction Motor V/Hz Control, driven by eTPU on MCF523x,” AN3000

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

AN2971
Rev. 2
4/2006