Freescale Semiconductor
Application Note

Document Number: AN2844
Rev. 0, 04/2005

Using the Current Controller (CC) eTPU Function

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

by: Milan Brejl
System Application Engineer, Roznov Czech System Center
Andrey Butok
System Application Engineer, Kiev Embedded Software Lab

Table of Contents

1 Introduction

The current controller (CC) enhanced time processor unit (eTPU) function is one of the functions included in the DC motor control eTPU function set (set3). This application note is intended to provide simple C interface routines to the CC eTPU function. The routines are targeted at the MCF523*x* and MPC5500 families of devices, but they could be easily used with any device that has an eTPU.

2 Function Overview

The CC function is not intended to process an input or output signal. The purpose of the CC function is to control another eTPU function's input parameter. The CC function includes a controller algorithm. The controller calculates its output based on two inputs: a measured value and a desired value. The measured value is usually provided by the analog sensing for DC motors (ASDC) function, that preprocesses the measured analog values. The desired value is a CC function parameter,

freescale semiconductor

and can be provided by the CPU or another eTPU function. In the motor-control eTPU function set, this function mostly provides the current closed loop.

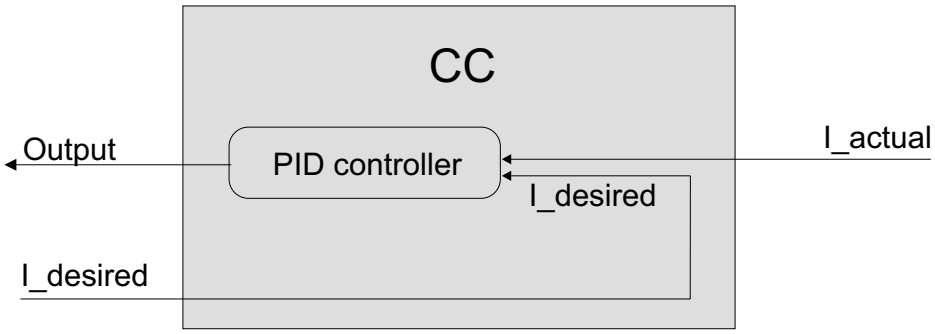

Figure 1. Functionality of CC

The controller algorithm is a general proportional-integral-derivative (PID) algorithm. It can be configured as PI or PID controller.

3 Function Description

The controller algorithm included in the CC function calculates the output according to the following equations:

$$u(k) = u_P(k) + u_I(k) + u_D(k)$$
$$e(k) = w(k) - m(k)$$
$$u_P(k) = G_P * e(k)$$
$$u_I(k) = u_I(k\text{-}1) + G_I * e(k)$$
$$u_D(k) = G_D * (e(k) - e(k\text{-}1))$$

Where:

- $u(k)$ – PID algorithm output (the output of CC) in step k.
- $u_P(k)$ – Proportional portion in step k.
- $u_I(k)$ – Integral portion in step k.
- $u_D(k)$ – Derivative portion in step k.
- $e(k)$ – Input error in step k.
- $w(k)$ – Desired value in step k.
- $m(k)$ – Measured value in step k.
- G_P – Proportional gain.
- G_I – Integral gain.
- G_D – Derivative gain.

During CC initialization, if the derivative gain is set to zero an internal flag which enables the calculation of derivative portion is cleared, resulting in a shorter calculation time.

The measured and desired values, as well as the gains, are applied with 24-bit precision. The integral portion is stored with 48-bit precision. The gain range is from 0 to 256, with a precision of 0.0000305 (30.5e-6).

Like most of the motor-control eTPU functions, the CC function also supports checking the eTPU latencies using an oscilloscope. The CC function channel, if connected to an output pin, turns the output pin high and low, so that the high-time identifies the period of time in which the CC update is executed.

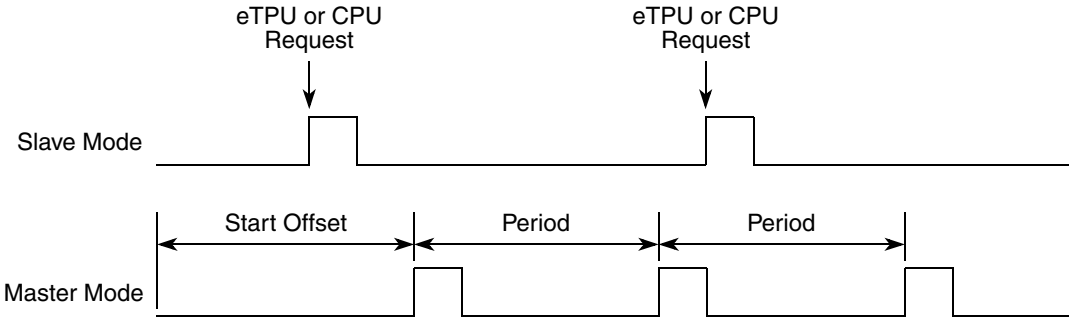

Figure 2. CC Updates in Slave Mode and Master Mode

The CC function update, in which the actual desired value and the measured value are taken and the control signal is adjusted, can be executed periodically, or by another process:

- **Master Mode**

  The CC update is executed periodically with a given period.
- **Slave Mode**

  The CC update is executed by the Analog Sensing (ASDC) eTPU function, other eTPU function, or by the CPU.

3.1 Interrupts

The CC 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 CC 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 CC function on the overall eTPU performance can be expressed by the following parameter:

- **Maximum eTPU busy-time during one update period**
  This value, compared to the update period value, determines the proportional load on the eTPU engine caused by the CC function.

Table 1 lists the maximum eTPU busy-times per update period in eTPU cycles that depend on the CC mode and controller type.

Table 1. Maximum eTPU Busy-times

Mode, Controller Type	Maximum eTPU Busy-time per CC Period [eTPU cycles]
Master Mode, PI Controller	150
Master Mode, PID Controller	160
Slave Mode, PI Controller	138
Slave Mode, PID Controller	148

The eTPU module clock is equal to the CPU clock on MPC5500 devices, as well as the peripheral clock, (half of the CPU clock) on MCF523x devices. For example, the eTPU module clock is 132 MHz on a 132-MHz MPC5554, and one eTPU cycle takes 7.58ns; eTPU module clock is only 75 MHz on a 150-MHz MCF5235, 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 1.0.0.5, 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 to the CC function for the application developer. Use of these functions eliminates the need to directly control the eTPU registers. There are 13 functions added to the application programming interface (API). The routines can be found in the `etpu_cc.h` and `etpu_cc.c` files, which should be included in the link file along with the top level development file(s).

Figure 3 shows the CC API state flow and lists API functions which can be used in each of its states.

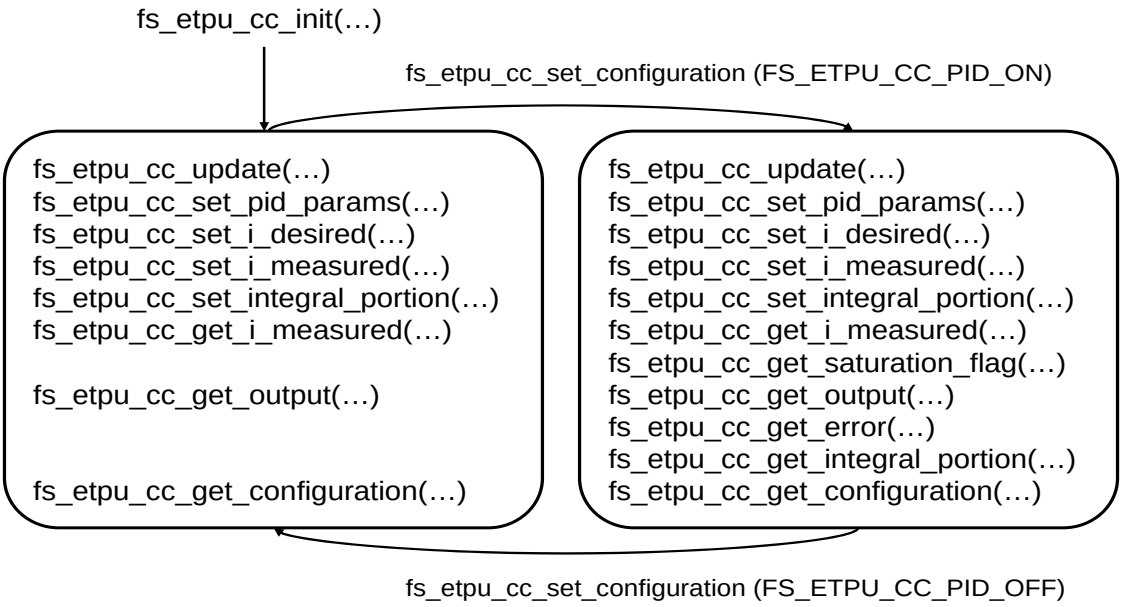

Figure 3. CC API State Flow

All CC API routines will be described in order and are listed below:

- Initialization Function:

```
int32_t fs_etpu_cc_init( uint8_t channel,
                         uint8_t priority,
                         uint8_t mode,
                         uint8_t configuration,
                        uint24_t period,
                        uint24_t start_offset,
                        uint24_t services_per_irq,
                 cc_pid_params_t* p_pid_params,
                         uint8_t output_chan,
                        uint16_t output_offset,
                         uint8_t link_chan )
```

- Change Operation Functions:

```
int32_t fs_etpu_cc_update(uint8_t channel);
int32_t fs_etpu_cc_set_configuration(uint8_t channel,
                                     uint8_t configuration);
int32_t fs_etpu_cc_set_pid_params(uint8_t  channel,
                           cc_pid_params_t* p_pid_params);
int32_t fs_etpu_cc_set_i_desired(uint8_t   channel,
                                 fract24_t i_desired);
int32_t fs_etpu_cc_set_i_measured(uint8_t   channel,
                                  fract24_t i_measured);
```

```
int32_t fs_etpu_cc_set_integral_portion(uint8_t   channel,
                                        fract24_t i_k1)
```

- Value Return Functions:

```
fract24_t fs_etpu_cc_get_i_measured(uint8_t channel);
uint8_t fs_etpu_cc_get_saturation_flag(uint8_t channel);
fract24_t fs_etpu_cc_get_output(uint8_t channel);
fract24_t fs_etpu_cc_get_error(uint8_t channel);
fract24_t fs_etpu_cc_get_integral_portion(uint8_t channel);
uint8_t fs_etpu_cc_get_configuration(uint8_t  channel);
```

4.1 Initialization Function

4.1.1 int32_t fs_etpu_cc_init(...)

This routine is used to initialize the eTPU channel for the CC function. This function has the following parameters:

- **channel (uint8_t)** - This is the CC channel number. This parameter should be assigned a value of 0-31 for ETPU_A, and 64-95 for ETPU_B.
- **priority (uint8_t)** - This is the priority to assign to the CC function. This parameter should be assigned a value of:
  - FS_ETPU_PRIORITY_HIGH
  - FS_ETPU_PRIORITY_MIDDLE
  - FS_ETPU_PRIORITY_LOW
  - FS_ETPU_PRIORITY_DISABLED
- **mode (uint8_t)** -This is the function mode. This parameter should be assigned a value of:
  - FS_ETPU_CC_MASTER
  - FS_ETPU_CC_SLAVE
- **configuration (uint8_t)** – This is the required configuration of CC. This parameter should be assigned a value of:
  - `FS_ETPU_CC_PID_OFF` (PID controller is disabled)
  - `FS_ETPU_CC_PID_ON` (PID controller is enabled)
- **period (uint24_t)** - This is the update period, as a number of TCR1 clocks. This parameter applies in master mode only (mode=FS_ETPU_CC_MASTER).
- **start_offset (uint24_t)** - This parameter is used to synchronize various eTPU functions. The first CC update starts *start_offset* TCR1 clocks after initialization. This parameter applies in master mode only (mode=FS_ETPU_CC_MASTER).
- **services_per_irq (uint24_t)** - This parameter defines the number of updates after which an interrupt service request is generated to the CPU.

- **p_pid_params (cc_pid_params_t*)** – This is the pointer to a cc_pid_params_t structure. The `cc_pid_params_t` structure is defined in `etpu_cc.h`:

```
typedef struct {
   fract24_t P_gain;
   fract24_t I_gain;
   fract24_t D_gain;
   int16_t positive_limit;
   int16_t negative_limit;
} cc_pid_params_t;
```

  Where:

  — **P_gain (fract24_t)** - This is the proportional gain and its value must be in the 24-bit signed fractional format 9.15 that means in the range of (-256, 256).

```
0x008000 corresponds to   1.0
0x000001 corresponds to   0.0000305 (30.5e-6)
0x7FFFFF corresponds to 255.9999695
```

  — **I_gain (fract24_t)** - This is the integral gain and its value must be in the 24-bit signed fractional format 9.15 that means in the range of (-256, 256).

  — **D_gain (fract24_t)** - This is the derivative gain and its value must be in the 24-bit signed fractional format 9.15 that means in the range of (-256, 256). To switch off calculation of derivative portion set this parameter to zero.

  — **positive_limit (int16_t)** - This is the positive output limit and its value must be in the 16-bit signed fractional format 1.15 that means in the range of (-1, 1).

  — **negative_limit (int16_t)** - This is the negative output limit and its value must be in the 16-bit signed fractional format 1.15 that means in the range of (-1, 1).

- **output_chan (uint8_t)** – CC writes the PID output value to a recipient function input parameter. This is the recipient function channel number. This parameter should be assigned a value of 0-31 forETPU_A, and 64-95 for ETPU_B.
- **output_offset (uint8_t)** – CC writes the PID output to a recipient function input parameter. This is the recipient function parameter offset. Function parameter offsets are defined in etpu_<func>_auto.h file.
- **link_chan (uint8_t)** – This is the channel number of a channel which receives a link after CC updates output. For example, if CC updates PWM duty-cycles it should be a PWMMDC channel. This parameter should be assigned a value of 0-31 forETPU_A, and 64-95 for ETPU_B.

4.2 Change Operation Functions

4.2.1 int32_t fs_etpu_cc_update(uint8_t channel)

This function executes the CC update. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

4.2.2 int32_t fs_etpu_cc_set_configuration(uint8_t channel, uint8_t configuration)

This function changes the CC configuration. It has the following parameters:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.
- **configuration (uint8_t)** – This is the required configuration of CC. This parameter should be assigned a value of:
  - `FS_ETPU_CC_PID_OFF` (PID controller is disabled)
  - `FS_ETPU_CC_PID_ON` (PID controller is enabled)

4.2.3 int32_t fs_etpu_cc_set_pid_params(uint8_t channel, cc_pid_params_t* p_pid_params)

This function changes the PID parameter values. It has the following parameters:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.
- **p_pid_params (cc_pid_params_t*)** - This is the pointer to the PID control structure. The `cc_pid_params_t` structure is defined in `etpu_cc.h`.

4.2.4 int32_t fs_etpu_cc_set_i_desired(uint8_t channel, fract24_t i_desired)

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

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.
- **i_desired (fract24_t)** - Desired input value. The value must be in the range MIN24 to MAX24.

4.2.5 int32_t fs_etpu_cc_set_i_measured(uint8_t channel, fract24_t i_measured)

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

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.
- **i_measured (fract24_t)** - Measured value. The value must be in the range MIN24 to MAX24.

4.2.6 int32_t fs_etpu_cc_set_integral_portion(uint8_t channel, fract24_t i_k1)

This function sets the integral portion (usually used to set the integral portion to zero). It has the following parameters

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.
- **i_k1 (fract24_t)** - This is the integral portion value in 24-bit signed fractional format 1.23, range (-1,1).

4.3 Value Return Function

4.3.1 fract24_t fs_etpu_cc_get_i_measured(uint8_t channel)

This function gets the measured value, as a portion of the maximum value. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

The value of the measured current is returned as a fract24_t in the range (-1, 1). The actual current, in [A], can be obtained by multiplying of the returned value by measurement range.

4.3.2 uint8_t fs_etpu_cc_get_saturation_flag(uint8_t channel)

This function returns the PID controller saturation flags. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

The returned value can be one of the following:

- FS_ETPU_CC_SATURATION_NO (0) ... no saturation
- FS_ETPU_CC_SATURATION_POS (1) ... saturation to positive limit
- FS_ETPU_CC_SATURATION_NEG (2) ... saturation to negative limit

4.3.3 fract24_t fs_etpu_cc_get_output(uint8_t channel)

This function returns the PID controller output. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

4.3.4 fract24_t fs_etpu_cc_get_error(uint8_t channel)

This function returns the PID controller error. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

4.3.5 fract24_t fs_etpu_cc_get_integral_portion(uint8_t channel)

This function returns the PID controller integral portion. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

4.3.6 uint8_t fs_etpu_cc_get_configuration(uint8_t channel)

This function returns the CC configuration. It has the following parameter:

- **channel (uint8_t)** - This is the current controller channel number. This parameter must be assigned the same value as was assigned to the *channel* parameter in the initialization routine. If there are more CCs running simultaneously on the eTPU(s), the channel parameter distinguishes which CC function is accessed.

The returned value can be one of the following:

— `FS_ETPU_CC_PID_OFF` (PID controller is disabled)

— `FS_ETPU_CC_PID_ON` (PID controller is enabled)

5 Example Use of Function

5.1 Demo Application

The usage of the CC eTPU function is demonstrated in the application “DC Motor with Speed and Current Closed Loop, driven by eTPU on MCF523x”. For a detailed description of the demo, refer to AN2955.

5.1.1 Function Calls

The CC function is configured to slave mode. The ASDC function triggers the DC-bus current sampling, requests a DMA transfer of the conversion result to the eTPU DATA RAM, preprocess the obtained value by removing the DC offset and aligning to 24-bit fractional format, and writes the result into the CC parameter I_MEASURED. A link from the ASDC function is sent to the CC function in turn, in order to execute the CC update. The I_DESIRED is provided by the SC function, which is used as an outer loop controller. The CC controller output points to a PWMMDC input, so that it controls the duty-cycle of PWM phases.

```
/*****************************************************************************
 * Parameters
 *****************************************************************************/
uint8_t   PWMMDC_channel  = 7;
uint8_t   CC_channel      = 4;
fract24_t CC_P_gain       = 0x080000; /* 1.0 */
fract24_t CC_I_gain       = 0x000100; /* 0.0078125 */
```

```
/*************************************************************************
 *
 * Initialize Current Controller
 *
 *************************************************************************/
/*************************************************************************
 * 1) Define Current Controller PID Parameters
 *************************************************************************
 *  The P-gain and I-gain are given by parameters. The D-gain is set to
 *  zero in order to have PI-type speed controller.
 *  The positive and negative limits, which are set in 16-bit fractional
 *  format (1.15), can be adjusted in order to limit the speed controller
 *  output range, and also the integral portion range.
 *************************************************************************/
  cc_pid_params.P_gain = CC_P_gain;
  cc_pid_params.I_gain = CC_I_gain;
  cc_pid_params.D_gain = 0;
  cc_pid_params.positive_limit = 0x7FFF;
  cc_pid_params.negative_limit = 0x8000;
/*************************************************************************
 * 2) Initialize CC channel
 *************************************************************************/
  err_code = fs_etpu_cc_init(
     CC_channel,/* channel */
     FS_ETPU_PRIORITY_LOW,/* priority */
     FS_ETPU_CC_SLAVE,/* mode */
     FS_ETPU_CC_PID_OFF,/* configuration */
     0,/* period */
     0,/* start_offset */
     0,/* services_per_irq */
     &cc_pid_params,/* p_pid_params */
     PWMMDC_channel,/* output_chan */
     FS_ETPU_PWMMDC_VOLTAGE_OFFSET,/* output_offset */
     PWMMDC_channel/* link_chan */
  );

/*************************************************************************
 *
 * Enable Current Controller
 *  - Reset PID integral portion and set PID on
 *
 *************************************************************************/
  fs_etpu_cc_set_integral_portion(CC_channel, 0);
  fs_etpu_cc_set_configuration(CC_channel, FS_ETPU_CC_PID_ON);

/*************************************************************************
 *
 * Disable Current Controller
 *  - Set PID off and reset PID integral portion
```

```
 *
 ****************************************************************************/
   fs_etpu_cc_set_configuration(CC_channel, FS_ETPU_CC_PID_OFF);
   fs_etpu_cc_set_integral_portion(CC_channel, 0);
```

6 Summary and Conclusions

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

References:

1. “The Essentials of Enhanced Time Processing Unit,” AN2353.
2. “General C Functions for the eTPU,” AN2864.
3. “Using the DC Motor Control eTPU Function Set (set3),” AN2958.
4. “DC Motor with Speed and Current Closed Loop, driven by eTPU on MCF523x,” AN2955.
5. *Enhanced Time Processing Unit Reference Manual*, ETPURM/D.
6. eTPU Graphical Configuration Tool, http://www.freescale.com/etpu, ETPUGCT.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

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

Document Number: AN2844
Rev. 0
04/2005