Freescale Semiconductor
Application Note

Document Number: AN2849
Rev. 0, 11/2007

Using the eTPU Pulse Width Modulation (PWM) Function

by: David Paterson
MCD Applications EKB

This application note provides simple C interface routines for the enhanced Time Processing Unit (eTPU) PWM function. These routines can be can be used on any device that includes an eTPU module.

Sample code written for the MPC5554 device can be downloaded from www.freescale.com (AN2849SW).

This application note should be read with application note AN2864, "General C Functions for the eTPU."

Contents

1 PWM Function Overview

The PWM function generates output pulses according to a set of parameters (frequency, duty cycle, and polarity), without CPU intervention.

The eTPU PWM function is based on the TPU PWM function, with the following enhancements.

- Complete coverage of the full 0 and 100% duty cycle range
- Resolution enhanced to 24 bits
- Up to 64 eTPU channels available

2 Functional Description

Pulse width modulation involves modifying the frequency, duty cycle, and polarity of a an output pulse, resulting in variation of the average value of the resulting waveform. Applications include driving DC motors and solenoids.

The PWM function output pulses are based on a set of parameters:

- Frequencies typically between 1 Hz and 100 kHz
- Duty cycles between 0 and 100% with 0.01% resolution
- Polarity active-high or active-low

The duty cycle, or the duty cycle and the frequency, can be updated by a host service request (HSR). The duty cycle may be updated on the current cycle or on the following cycle, depending on when the HSR event occurs. If both the duty cycle and frequency are to be updated, this can be done coherently in the following cycle. These scenarios can be seen in Figure 2.

2.1 Performance and Use of eTPU PWM Function

2.1.1 Performance

As with all eTPU functions, the performance of the PWM function in an application depends, to some extent, on the service time (latency) of other active eTPU channels. This is due to the operational nature of the scheduler. The performance decreases proportionally as the number of active (requesting service) eTPU channels increases. Worst-case latency in any eTPU application can be closely estimated.

To analyze the performance of an application that appears to approach the limits of the eTPU, use the guidelines given in the eTPU reference manual and the information provided in the eTPU PWM software release available from Freescale.

2.1.2 Updating the PWM Function Parameters

2.1.2.1 Initialization

The main routine *fs_etpu_pwm_init* initializes the channel for PWM. This allows channel number, priority, frequency, duty cycle, polarity, timebase, and timebase frequency to be set.

2.1.2.2 Updating the Duty Cycle

After initialization, the duty cycle of the output pulse can be updated by using the *fs_etpu_pwm_duty* routine. Using this routine, the duty cycle is always be updated in the next cycle — there is no intermediate update.

Figure 1 provides an example of how the output pulse is updated with respect to the host service request (HSR).

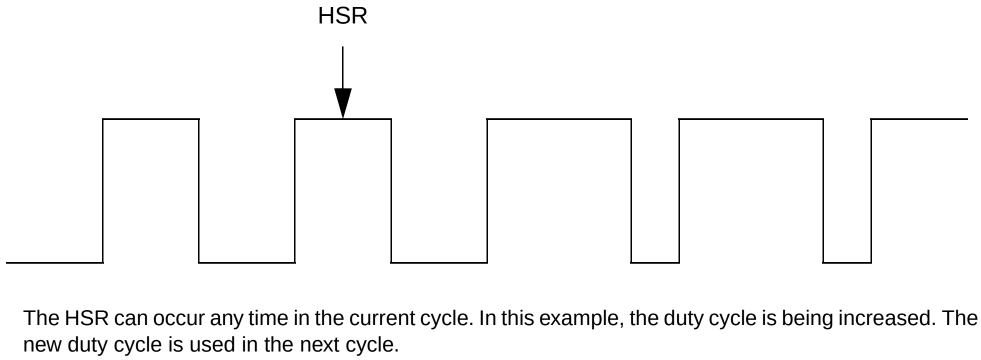
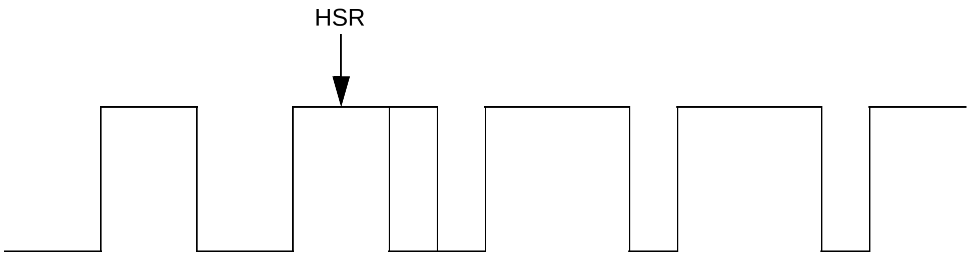


Figure 1.

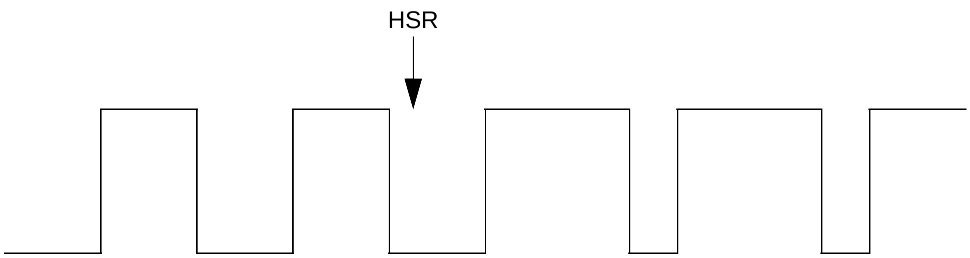
2.1.2.3 Updating the Duty Cycle Immediately

After initialization, the duty cycle of the output pulse can be updated immediately by using the *fs_etpu_pwm_duty_immed* routine. This routine updates the duty cycle immediately, if possible, by requesting a host service request.

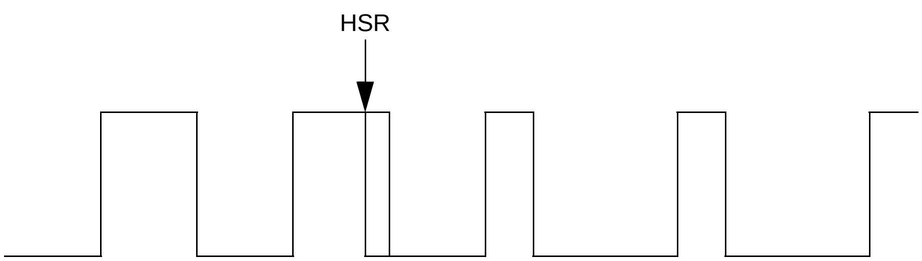
Figure 2 shows the three possible scenarios for updating the duty cycle, with respect to the timing of the immediate HSR. Case 3 shows an intermediate update; in this case, it is possible to update the duty cycle immediately on HSR. However, the full duty cycle change cannot be applied in the current cycle, and the full change is made on the next cycle.



Case 1: Active high. The HSR occurs before service of the active edge, and the duty cycle is being increased. In this case, the new duty cycle is used immediately.



Case 2: Active high. The HSR comes after service of the active edge, and the duty cycle is being increased. In this case, the new duty cycle is used in the next cycle.

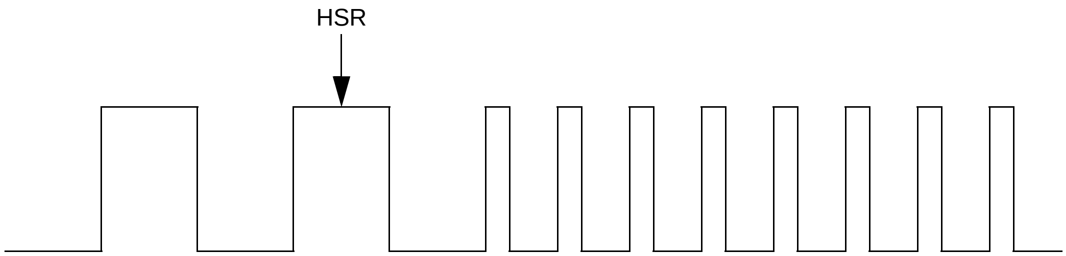


Case 3: Active high. The HSR comes before service of the active edge, and the duty cycle is being decreased. In this case, the duty cycle is partly updated in the current cycle, then fully in the next cycle.

Figure 2.

2.1.2.4 Updating the Frequency and the Duty Cycle

The routine *fs_etpu_pwm_update* allows coherent updating of the frequency and duty cycle for a specified channel (Figure 3).



In this example, both the duty cycle and frequency are modified in the next cycle.

Figure 3.

2.1.3 Changing Operating Modes

To change the operating mode of a channel that is running, the channel must first be disabled. This can be done using the *fs_etpu_disable* routine, which can be found in file *etpu_utils.h*. This is required only for changing the parameter for active-high or active-low polarity.

3 C Level API for the eTPU PWM Function

The following routines provide easy access to the PWM function. Using these routines eliminates the need to control the eTPU registers directly.

The application program interface (API) comprises five routines. These routines can be found in the *etpu_pwm.h* and *etpu_pwm.c* files; they are described in the section that follows and are available from Freescale.

In addition to the *etpu_pwm.h* and *etpu_pwm.c* files, the eTPU C-compiler generates a file called *etpu_pwm_auto.h*. This file contains information relating to the eTPU PWM function, including details of how the eTPU data memory is organized and definitions for various API parameters.

```
int32_t fs_etpu_pwm_init          (uint8_t channel,
                                   uint8_t priority,
                                   uint32_t freq,
                                   uint16_t duty,
                                   uint8_t polarity,
                                   uint8_t timebase,
                                   uint32_t timebase_freq)

void fs_etpu_pwm_duty             (uint8_t channel,
                                   uint16_t duty)

void fs_etpu_pwm_duty_immed       (uint8_t channel,
                                   uint16_t duty)
```

```
int32_t fs_etpu_pwm_update         (uint8_t channel,
                                    uint32_t freq,
                                    uint16_t duty,
                                    uint32_t timebase_freq)

uint32_t fs_etpu_pwm_get_freq      (uint8_t channel,
                                    uint32_t timebase_freq)
```

3.1 Initialization Routines

The initialization routines dynamically allocate eTPU data memory. If dynamic allocation is not required, the channel's Channel Parameter Base Address field should be written with a non-zero value before calling the *fs_etpu_pwm_init* function.

Dynamic allocation of eTPU data memory occurs if the channel has a zero in its Channel Parameter Base Address field. The Channel Parameter Base Address field is updated by the API with a non-zero value to point to the parameter RAM allocated to the channel. The *fs_etpu_pwm_init* API does not allocate new parameter RAM if the channel has a non-zero value in its Channel Parameter Base Address field (a non-zero value means that the channel has already been assigned).

The initialization routine *fs_etpu_pwm_init* is used to initialize an eTPU channel for the eTPU PWM functions. After the channel has been initialized, the channel starts to output pulses based on set parameters. Other routines can update the output pulse (*fs_etpu_pwm_duty*, *fs_etpu_pwm_duty_immed*, *fs_etpu_pwm_update*), and read back the PWM frequency (*fs_etpu_pwm_get_freq*).

The functions have the following parameters:

- **channel (uint8_t)**: The PWM channel number. For devices with two eTPUs, this parameter should be assigned a value of 0-31 for eTPU_A and 64-95 for eTPU_B. For products with a single eTPU, this parameter should be assigned a value of 0-31.
- **priority (uint8_t):** The priority to assign to the eTPU PWM channel. The following eTPU priority definitions are found in utilities file *etpu_utils.h.*:
  - — FS_ETPU_PRIORITY_HIGH
  - — FS_ETPU_PRIORITY_MIDDLE
  - — FS_ETPU_PRIORITY_LOW
  - — FS_ETPU_PRIORITY_DISABLED
- **freq (uint32_t):** The frequency of the PWM output in Hz. The range of this parameter is determined by the complete system but normally should be between 1 Hz and 100 kHz.
- **duty (uint16_t):** The duty cycle of the PWM output as a percentage. This is a unit16_t integer with a range of 0–10000, to represent 0–100% with 0.01% resolution. For example, 5000 represents 50% and 10000 represents 100%.
- **polarity (uint8_t):** The polarity of the channel. This parameter should be assigned a value of:
  - — FS_ETPU_PWM_ACTIVEHIGH
  - — FS_ETPU_PWM_ACTIVELOW

- **timebase (uint8_t):** The timebase the eTPU PWM channel will use. The following eTPU timebase definitions are found in utilities file *etpu_utils.h*:
  — FS_ETPU_TCR1
  — FS_ETPU_TCR2
- **timebase_freq (uint32_t):** The system frequency in Hz. The range of this is the same as the range of the timebase frequency on the device.

Return Notes: Returns an error code if the channel could not be initialized. The error codes that can be returned are found in utilities file *etpu_utils.h*:

— FS_ETPU_ERROR_MALLOC (memory allocation)
— FS_ETPU_ERROR_FREQ (frequency is too low or too high)

NOTE

This PWM function configures the eTPU channel only; it does not configure the pin. In a system, a pin may have to be configured to select the eTPU functionality. For example, the pad configuration register (PCR) must be configured within the system integration unit (SIU). See the MPC5500 example code in the software file AN2849SW.

4 Examples of Using the Function

4.1 Simple Example

4.1.1 Description

This section describes a simple use of the PWM function; it shows how to initialize the eTPU module and assign the eTPU PWM function to an eTPU channel.

4.1.2 Example Code

The example consists of two files:

- pwm_example1.c
- pwm_example1.h

File pwm_example1.c contains the main() routine. This routine initializes the MPC5554 device for 128 MHz CPU operation, and initializes the eTPU according to the information in the my_etpu_config struct (stored in file pwm_example1.h).

One pin is used in this example. ETPUA0 is configured for eTPU functionality, then the PWM function is initialized on this channel (defined as PWM0 in pwm_example1.h).

The channel is configured to produce a 1 kHz pulse with a 25% duty cycle, with middle priority <<Is middle priority a defined term?>> and a timebase frequency of 64 MHz:

```
error_code = fs_etpu_pwm_init (PWM0, FS_ETPU_PRIORITY_MIDDLE, 1000, 2500,
FS_ETPU_PWM_ACTIVEHIGH, FS_ETPU_TCR1, etpu_a_tcr1_freq);
```

After a delay, the pulse is updated to produce a 2 kHz pulse with a 60% duty cycle.

```
for(x=0; x<0x1000000; x++);         // Delay
error_code = fs_etpu_pwm_update (PWM0, 2000, 6000, etpu_a_tcr1_freq);
```

4.1.3 Program Output

The updated frequency is read into a variable *new_freq* in the source code, to verify that the frequency has been updated, using the *fs_etpu_pwm_get_freq* routine.

Figure 4 shows simulated oscilloscope traces of the output pulse trains.

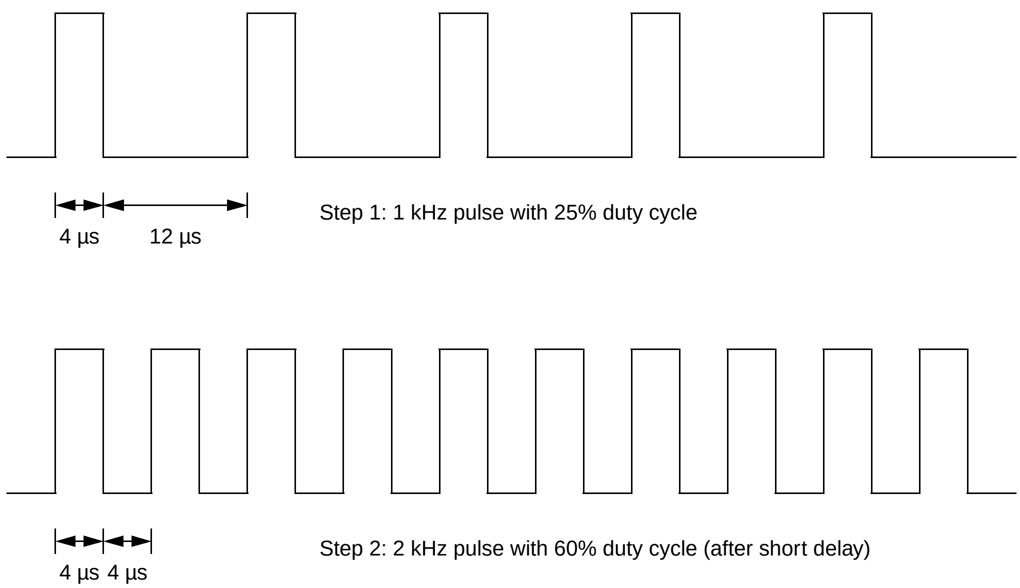


Figure 4.

4.2 Complex Example

4.2.1 Description

This section describes a more complex use of the PWM function. Two separate channels are used to output pulses. All five PWM API routines are exercised.

4.2.2 Example Code

This example consists of two files:

- pwm_example2.c
- pwm_example2.h

File pwm_example2.c contains the main() routine. This routine initializes the MPC5554 device for 128 MHz CPU operation and initializes the eTPU according to the information in the my_etpu_config struct (stored in file pwm_example2.h).

This example uses two pins: ETPUA0 (defined as PWM0 in pwm_example2.h) and ETPUA1 (defined as PWM1 in pwm_example2.h). These are set up as follows, where the timebase frequency is 64 MHz:

```
/* 10Khz, 50%, active high, middle priority using TCR1 */
error_code = fs_etpu_pwm_ini t(PWM0, FS_ETPU_PRIORITY_MIDDLE, 10000, 5000,\
         FS_ETPU_PWM_ACTIVEHIGH, FS_ETPU_TCR1, etpu_a_tcr1_freq);

/* 10Khz, 25%, active low, middle priority using TCR1 */
error_code = fs_etpu_pwm_init (PWM1, FS_ETPU_PRIORITY_MIDDLE, 10000, 2500,\
         FS_ETPU_PWM_ACTIVELOW, FS_ETPU_TCR1, etpu_a_tcr1_freq);
```

Then, PWM0 is updated to 22 kHz in increments of 1 kHz. After each increment, the *fs_etpu_pwm_get_freq* routine is used to measure the output frequency into program variable *get_freq*.

```
         fs_etpu_pwm_update (PWM0, 11000, 5000, etpu_a_tcr1_freq);
         get_freq = fs_etpu_pwm_get_freq (PWM0, etpu_a_tcr1_freq);
```

Then, the duty cycle of PWM1 is continuously updated from 0% to 100% and PWM0 is finally settled at 5 kHz with a 5% duty cycle.

4.2.3 Program Output

The updated frequencies are read into the program variable *get_freq* using the *fs_etpu_pwm_get_freq* routine, to check the actual output frequency. Figure 5 shows simulated oscilloscope traces of the output pulse trains.

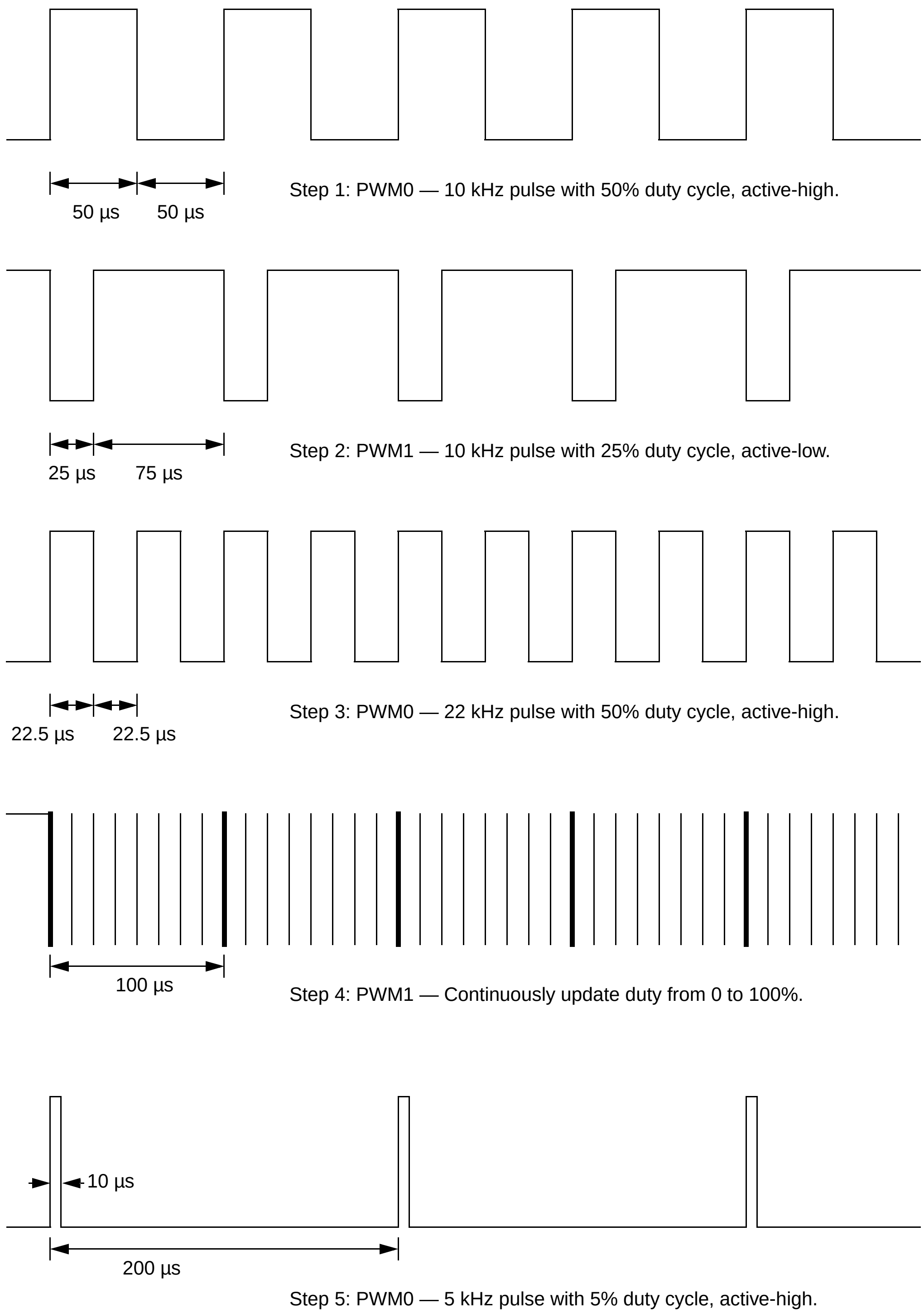

50 µs
50 µs
Step 1: PWM0 — 10 kHz pulse with 50% duty cycle, active-high.
25 µs
75 µs
Step 2: PWM1 — 10 kHz pulse with 25% duty cycle, active-low.
22.5 µs
22.5 µs
Step 3: PWM0 — 22 kHz pulse with 50% duty cycle, active-high.
100 µs
Step 4: PWM1 — Continuously update duty from 0 to 100%.
10 µs
200 µs
Step 5: PWM0 — 5 kHz pulse with 5% duty cycle, active-high.

Figure 5.

5 Summary and Conclusions

This eTPU PWM application note describes how to use the pulse width modulation eTPU function, and illustrates its use with working examples. The simple C interface routines of the eTPU PWM function enable easy implementation of the PWM function in applications. The routines are aimed at the MPC5500 family of devices, but they can be used with any device that has an eTPU.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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: AN2849
Rev. 0
11/2007

freescale™
semiconductor