

Using the eTPU Spark Function

by: Bill Terry
MSG 32-bit Automotive Applications Engineering
USA, Austin

1 Introduction

This application note is intended to accompany the simple C interface functions for the eTPU SPARK function used as part of the complete eTPU Set 2 automotive function set. These functions can be used on any Freescale product that has an eTPU module. Example code is available for the MPC55xx and MPC563xM devices. This application note must be read in conjunction with application note AN2864—*General C Functions for the eTPU* and application note AN3768 — *eTPU Automotive Function Set (Set 2)*.

2 Function Overview

The eTPU SPARK function initializes and controls the generation of spark pulses synchronized to specific angular positions of a rotating engine, thus controlling the ignition timing of the engine. The function outputs one or more spark pulses during each complete engine

Contents

1	Introduction	1
2	Function Overview	1
3	Functional Description	2
3.1	Dwell Time and Minimum and Maximum Dwell	3
3.2	Performance and Use of the eTPU SPARK	4
4	C Level Application Program Interface (API) for eTPU	4
4.1	Initialization API Calls	5
4.2	Parameter Update API Calls	7
4.3	Channel Parameter Base Address	9
5	Example of Function Usage	9
6	Conclusion	10

Functional Description

cycle without CPU intervention. The properties of these pulses are programmable, and update functions allow for the pulse properties to be changed during runtime.

The SPARK function supports the generation of either one or two main spark pulses per 720° engine cycle, as well as optional multi spark pulse generation for ignition algorithms that use spark multi-strike. Minimum and maximum dwell time enforcement, selectable output polarity, and asynchronous updates are also provided.

3 Functional Description

The application programming interface (API) described in [Section 4, “C Level Application Program Interface \(API\) for eTPU SPARK Functions”](#) provides a convenient method for the application software to configure and control one or more eTPU spark channels. The parameters that must be specified for spark pulse generation include:

- Angle-based parameters that determine the main spark pulse end angle.
 - Spark_Angle — This is the angle at which the main spark pulse must end
 - Cylinder_Offset_Angle — The angle offset for each engine cylinder
 - Recalc_Offset_Angle — The angle offset for recalculation of the main spark pulse start angle
- Time-based parameters that determine the width of the spark pulse(s).
 - Spark_Dwell — Defines the targeted width of the main spark pulse
 - Min_Dwell — Defines the minimum width of a main spark pulse. See [Figure 2](#).
 - Max_Dwell — Defines the maximum width of a main spark pulse. See [Figure 3](#).
 - Multi_On_Time — Defines the width of the optional multi spark pulse(s)
 - Multi_Off_Time — Defines the time between the main pulse and any optional multi spark pulse(s)
- Other parameters that determine additional characteristics of the spark pulse(s).
 - Multi_Num_Pulses — Specifies the number of additional multi spark pulses to follow each main spark pulse

[Figure 1](#) shows a typical spark pulse and the parameters discussed above.

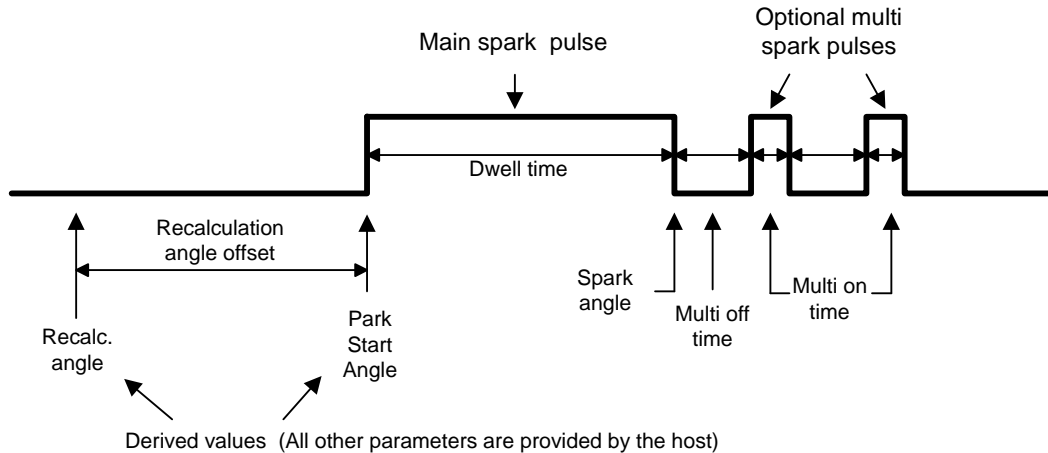


Figure 1. Spark Pulse Diagram

3.1 Dwell Time and Minimum and Maximum Dwell

Dwell time — Dwell time is the length of time the application requests the main spark pulse to be in an active state (in dwell). Start-of-dwell is the point at which the spark pulse output goes active and end-of-dwell is the point at which the output goes back to an inactive state. Because of engine dynamics (acceleration and deceleration), the time between these two points (actual dwell time) may or may not be equal to the dwell time that was specified by the application. This difference occurs because the function must ensure that the spark pulse ends at the correct engine angle.

Notice that setting the `spark2_dwell_time` to zero causes generation of a single pulse per 720°. See [Section 4.2](#).

Minimum and maximum dwell — A spark pulse must be longer than some minimum widths to ensure that the ignition coil has been charged sufficiently to generate a reliable spark after the pulse ends. Conversely, a spark pulse must be shorter than some maximum width to ensure that the ignition coil being charged is not damaged by too much current and heat.

The `Spark_Dwell` parameter specifies the optimal spark pulse width to be generated, this time is used to estimate the `Spark_Start_Angle` before the pulse begins. After the pulse begins, its width becomes secondary to the `Spark_Angle`, and the SPARK function adjusts the width (or actual dwell time) of the pulse as required to end the pulse at the correct angle.

The accuracy of the `Spark_Start_Angle` estimation degrades as a result of acceleration and deceleration of the engine. During acceleration and deceleration, the actual dwell time can vary over the range defined by the global parameters `Min_Dwell` and `Max_Dwell`.

`Min_Dwell` is the minimum time (in TCR1 timer ticks) to which the actual dwell time may be reduced as a result of engine acceleration or an update to `Spark_Angle`.

`Max_Dwell` is the maximum time (in TCR1 timer ticks) to which the actual dwell time may increase as a result of engine deceleration or an update to `Spark_Angle`.

Figure 2 and Figure 3 show the effect of the Min_Dwell and Max_Dwell parameters on the generation of spark pulses.

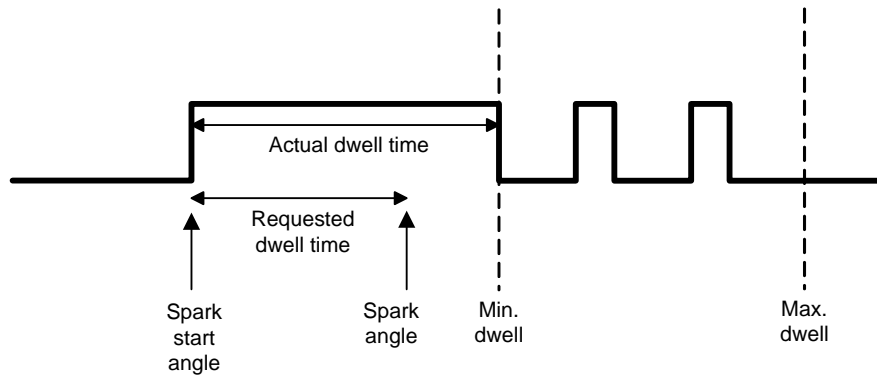


Figure 2. Minimum dwell — Spark pulse extended to meet minimum dwell requirement

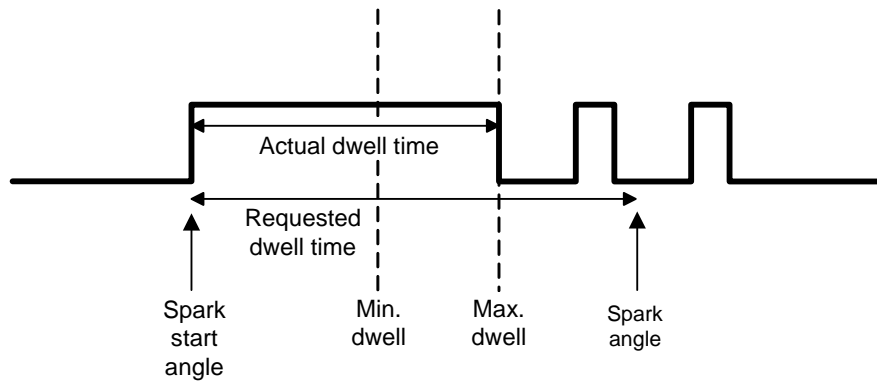


Figure 3. Maximum dwell — Spark pulse truncated to meet maximum dwell requirement

3.2 Performance and Use of the eTPU SPARK Function

Performance — Like all eTPU functions, the SPARK 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. 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 SPARK software release available from Freescale.

Limitations — The application software is responsible for ensuring that the angle and time parameters used to specify pulses spark1 and spark2 are valid and do not cause the pulses to overlap.

4 C Level Application Program Interface (API) for eTPU SPARK Functions

The following functions provide easy access to the features of the eTPU SPARK function by the application developer. Use of these functions eliminates the need to directly control the eTPU registers.

The SPARK function API consists of eight functions. These functions can be found in the *etpu_spark.c* and *etpu_spark.h* files. The functions are described below and are available from Freescale as part of this application note download. In addition, the eTPU C-compiler generates a file called *etpu_spark_auto.h*. This file contains information relating to the eTPU SPARK function including details of how the eTPU data memory is organized and definitions for various API parameters.

4.1 Initialization API Calls

These C function calls provide initialization function for eTPU SPARK channels.

```
int8_t fs_etpu_spark_init_3cylinders(  uint8_t  spark_chan_1,
                                       uint8_t  spark_chan_2,
                                       uint8_t  spark_chan_3,
                                       uint8_t  cam_chan,
                                       uint24_t cyl_offset_angle_1,
                                       uint24_t cyl_offset_angle_2,
                                       uint24_t cyl_offset_angle_3,
                                       uint8_t  priority,
                                       uint8_t  polarity,
                                       uint24_t min_dwell_time,
                                       uint24_t max_dwell_time,
                                       uint24_t multi_on_time,
                                       uint24_t multi_off_time,
                                       uint8_t  multi_num_pulses,
                                       uint24_t recalc_offset_angle,
                                       uint24_t init_dwell_time_1,
                                       uint24_t init_dwell_time_2,
                                       uint24_t init_end_angle_1,
                                       uint24_t init_end_angle_2 );

int8_t fs_etpu_spark_init_4cylinders(  uint8_t  spark_chan_1,
                                       uint8_t  spark_chan_2,
                                       uint8_t  spark_chan_3,
                                       uint8_t  spark_chan_4,
                                       uint8_t  cam_chan,
                                       uint24_t cyl_offset_angle_1,
                                       uint24_t cyl_offset_angle_2,
                                       uint24_t cyl_offset_angle_3,
                                       uint24_t cyl_offset_angle_4,
                                       uint8_t  priority,
                                       uint8_t  polarity,
                                       uint24_t min_dwell_time,
                                       uint24_t max_dwell_time,
                                       uint24_t multi_on_time,
                                       uint24_t multi_off_time,
                                       uint8_t  multi_num_pulses,
                                       uint24_t recalc_offset_angle,
                                       uint24_t init_dwell_time_1,
                                       uint24_t init_dwell_time_2,
                                       uint24_t init_end_angle_1,
                                       uint24_t init_end_angle_2 );

int8_t fs_etpu_spark_init_6cylinders(  uint8_t  spark_chan_1,
                                       uint8_t  spark_chan_2,
                                       uint8_t  spark_chan_3,
                                       uint8_t  spark_chan_4,
                                       uint8_t  spark_chan_5,
                                       uint8_t  spark_chan_6,
                                       uint8_t  cam_chan,
                                       uint24_t cyl_offset_angle_1,
```

```

uint24_t cyl_offset_angle_2,
uint24_t cyl_offset_angle_3,
uint24_t cyl_offset_angle_4,
uint24_t cyl_offset_angle_5,
uint24_t cyl_offset_angle_6,
uint8_t  priority,
uint8_t  polarity,
uint24_t min_dwell_time,
uint24_t max_dwell_time,
uint24_t multi_on_time,
uint24_t multi_off_time,
uint8_t  multi_num_pulses,
uint24_t recalc_offset_angle,
uint24_t init_dwell_time_1,
uint24_t init_dwell_time_2,
uint24_t init_end_angle_1,
uint24_t init_end_angle_2 );

```

4.1.1 Parameter Details

Any one or combination of three separate initialization functions (`fs_etpu_spark_init_3cylinders`, `fs_etpu_spark_init_4cylinders`, or `fs_etpu_spark_init_6cylinders`) are used to initialize the desired eTPU channels for the eTPU SPARK function. After the channel(s) have been initialized, they wait for the desired angles to occur and generate spark pulse(s) as specified.

Each initialization function has the following parameters:

- `spark_channel_n` (uint8_t) — The eTPU channel assigned to SPARK for cylinder n. For products with a single eTPU, this parameter must be assigned a value of 0 – 31. For devices with two eTPUs, this parameter must be assigned a value of 0 – 31 for eTPU_A or 64 – 95 for eTPU_B.
- `cam_chan` (uint8_t) — CAM channel number
- `cyl_offset_angle_n` (uint24_t) — Offset angle for cylinder n <0..71999>
- `priority` (uint8_t) — The priority to assign to the eTPU channel. The following eTPU priority definitions are found in utilities file *etpu_util.h*.
 - `FS_ETPU_PRIORITY_HIGH`
 - `FS_ETPU_PRIORITY_MIDDLE`
 - `FS_ETPU_PRIORITY_LOW`
 - `FS_ETPU_PRIORITY_DISABLED`
- `polarity` (uint8_t) — The polarity to assign to the eTPU channel. The following eTPU polarity definitions are found in the file *etpu_spark.h*.
 - `FS_ETPU_SPARK_ACTIVE_HIGH`
 - `FS_ETPU_SPARK_ACTIVE_LOW`
- `min_dwell_time` (uint24_t) — Minimum dwell time in μ Seconds.
- `max_dwell_time` (uint24_t) — Maximum dwell time in μ Seconds.
- `multi_on_time` (uint24_t) — Multi spark pulse active time in μ Seconds.
- `multi_off_time` (uint24_t) — Multi spark pulse inactive time in μ Seconds.
- `multi_num_pulses` (uint8_t) — Number of multi spark pulses to generate

- `recalc_offset_angle` (`uint24_t`) — Angle before estimated start angle at which start angle is recalculated <0..71999>
- `init_dwell_time_1` (`uint24_t`) — Dwell time in μ Seconds for first pulse (used for all channels at init)
- `init_dwell_time_2` (`uint24_t`) — Dwell time in μ Seconds for second pulse (used for all channels at init)
- `init_end_angle_1` (`uint24_t`) — End angle for first pulse (used for all channels at init) <0..71999>
- `init_end_angle_2` (`uint24_t`) — End angle for second pulse (used for all channels at init) <0..71999>

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

- `FS_ETPU_ERROR_MALLOC`
- `FS_ETPU_ERROR_VALUE`

Warning — This function does not configure the pin, it only configures the eTPU. In a system, a pin may need to be configured to select the eTPU functionality. See example code.

4.2 Parameter Update API Calls

The C function calls in this section provide the application with a means to update the SPARK function parameters after initialization. After the update function has been executed, the specified SPARK function is altered on the next scheduled SPARK function output.

```
int8_t fs_etpu_spark_set_dwell_times(  uint8_t channel,
                                       uint24_t spark1_dwell_time,
                                       uint24_t spark2_dwell_time );

int8_t fs_etpu_spark_set_end_angles(  uint8_t channel,
                                       uint24_t spark1_end_angle,
                                       uint24_t spark2_end_angle);

int8_t fs_etpu_spark_set_recalc_offset_angle ( uint8_t channel,
                                               uint24_t recalc_offset_angle);

int8_t fs_etpu_spark_set_min_max_dwell_times ( uint8_t channel,
                                               uint24_t min_dwell_time,
                                               uint24_t max_dwell_time);

int8_t fs_etpu_spark_set_multi_pulses ( uint8_t channel,
                                       uint24_t multi_on_time,
                                       uint24_t multi_off_time,
                                       uint8_t multi_num_pulses);
```

4.2.1 Parameter Details

This section provides details on the parameters passed to each of the update functions used to update various SPARK channel characteristics.

4.2.1.1 Set and Update Dwell Times

The *int32_t fs_etpu_spark_set_dwell_times* function is used to set or update the SPARK channel dwell times. This function is called with the following parameters:

- *channel* (uint8_t) — eTPU SPARK channel
- *spark1_dwell_time* (uint24_t) — Dwell time in μ Seconds for first pulse
- *spark2_dwell_time* (uint24_t) — Dwell time in μ Seconds for second pulse

Returns:

- 0 if the dwell time update was successful
- Sum of pending HSR numbers if the channel has pending HSRs (in this case, the function must be called again)

4.2.1.2 Set and Update End Angles

The *int32_t fs_etpu_spark_set_end_angles* function is used to set or update the SPARK channel end angles. This function is called with the following parameters:

- *channel* (uint8_t) — eTPU SPARK channel
- *spark1_end_angle* (uint24_t) — End angle for the first pulse
- *spark2_end_angle* (uint24_t) — End angle for the second pulse

Returns:

- 0 if end angle update was successful
- *FS_ETPU_ERROR_VALUE* if either SPARK end angle parameter is out of range
- Sum of pending HSR numbers if the channel has pending HSRs (in this case, the function must be called again)

4.2.1.3 Set and Recalculate Offset Angle

The *int32_t fs_etpu_spark_set_recalc_offset_angle* function is used to set or update the SPARK channel recalculate offset angle. This function is called with the following parameters:

- *channel* (uint8_t) — eTPU SPARK channel.
- *recalc_offset_angle* (uint24_t) — Angle before estimated start angle at which start angle is recalculated.

Returns:

- 0 if recalculate offset angle update was successful.
- *FS_ETPU_ERROR_VALUE* if recalculate offset angle parameter is out of range.

4.2.1.4 Set and Update Minimum and Maximum Dwell Times

The *int32_t fs_etpu_spark_set_min_max_dwell_times* function is used to set or update the SPARK channel minimum and maximum dwell times. This function is called with the following parameters:

- *channel* (uint8_t) — eTPU SPARK channel.

- `min_dwell_time (uint24_t)` — Minimum dwell time in μ Seconds.
- `max_dwell_time (uint24_t)` — Maximum dwell time in μ Seconds.

Returns:

- Always returns 0.

4.2.1.5 Set and Update Multi Pulses

The `int32_t fs_etpu_spark_set_multi_pulses` function is used to set or update the SPARK channel multi spark active times.

- `channel (uint8_t)` — eTPU SPARK channel.
- `multi_on_time (uint24_t)` — Multi spark pulse active time in μ Seconds.
- `multi_off_time (uint24_t)` — Multi spark pulse inactive time in μ Seconds.

Returns:

- Always returns to 0.

4.3 Channel Parameter Base Address

The initialization and update functions described in [Section 4.1, “Initialization API Calls”](#) and [Section 4.2, “Parameter Update API Calls”](#) dynamically allocate eTPU data memory if the channel has a zero in its channel parameter base address (CPBA) field. When a channel is first initialized, the CPBA is updated by the API with a non-zero value to point to the parameter RAM allocated to the channel. If the CPBA field is already non-zero, the SPARK API calls will not allocate new parameter RAM. The non-zero value indicates that the channel’s parameter RAM has already been allocated.

5 Example of Function Usage

Simple Example

- **Description** — The example code sets up three eTPU channels (5, 6, and 7) to implement a SPARK function. The three channels are configured to generate a spark pulse at 30° , 270° , and 510° respectively. The polarity is configured to be active high for all three channels. Two multi-pulses are generated after each spark pulse with an on-time of $209 \mu\text{S}$ and an off-time of $209 \mu\text{S}$.
- **Example Code** — The example code is partly generated by Freescale's eTPU graphical configuration tool (GCT). See output files `eng_pos_etpu_gct.c` and `eng_pos_etpu_gct.h`. A tooth generator function is used to simulate crank and cam signals and the engine position functions are used to generate the angle clock.

The `high level main.c` file configures the pins, sets up the various spark parameters, base addresses, and loads and initializes the eTPU.

Conclusion

The SPARK function initialization call in this example looks like:

```
fs_etpu_spark_init_3cylinders (SPARK0_SPARK_1, /* engine: A; channel: 5 */
                               SPARK0_SPARK_2, /* engine: A; channel: 6 */
                               SPARK0_SPARK_3, /* engine: A; channel: 7 */
                               14, /* cam_chan: 14 */
                               3000, /* cyl_offset_angle_1: 3000 */
                               27000, /* cyl_offset_angle_2: 27000 */
                               51000, /* cyl_offset_angle_3: 51000 */
                               FS_ETPU_PRIORITY_MIDDLE, /* priority: Middle */
                               FS_ETPU_SPARK_ACTIVE_HIGH, /* polarity active high */
                               834, /* min_dwell_time: 834 */
                               1042, /* max_dwell_time: 1042 */
                               209, /* multi_on_time: 209 */
                               209, /* multi_off_time: 209 */
                               2, /* multi_num_pulses: 2 */
                               20, /* recalc_offset_angle: 20 */
                               625, /* init_dwell_time_1: 625 */
                               0, /* init_dwell_time_2: 0 */
                               6000, /* init_end_angle_1: 6000 */
                               0); /* init_end_angle_2: 0 */
```

- Program Output — The outputs of the three SPARK functions will look similar to the plot shown in Figure 4.

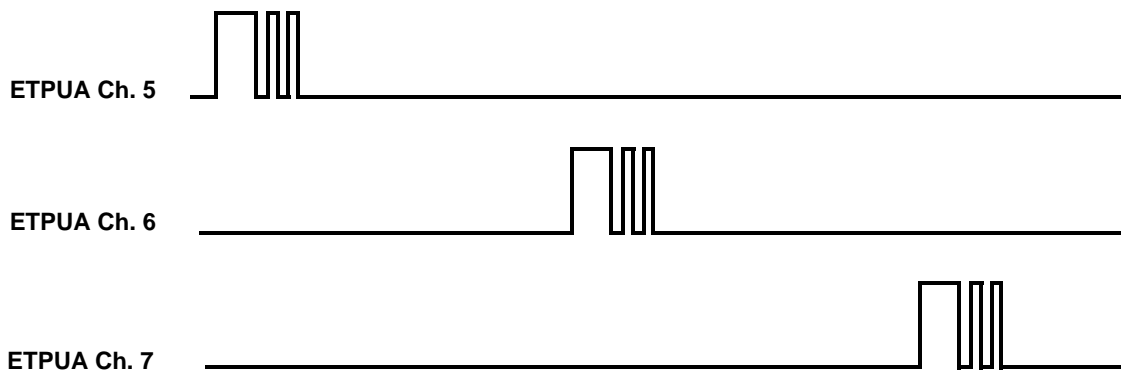


Figure 4. Spark function example – output capture

6 Conclusion

This application note provides the user with a description of the eTPU SPARK function usage and examples. The simple C interface functions that interface to the SPARK eTPU function enable easy implementation of the SPARK function in applications. The functions are targeted for the MPC55xx and MPC563xM family of devices, but they can be used with any device that has an eTPU.

THIS PAGE IS INTENTIONALLY BLANK

Because of an order from the United States International Trade Commission, BGA-packaged product lines and part numbers indicated here currently are not available from Freescale for import or sale in the United States prior to September 2010: MPC55xx, MPC563xM in MAPBGA packages

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3771
Rev. 0
07/2009

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2009. All rights reserved.

