Freescale Semiconductor
Application Note

Document Number: AN3813
Rev. 0, 04/2009

ADC Average Driver Using the IO Processor (IOP) in the MPC5510 Family

by: Oscar Luna
PMMC Software Engineer
GDL

1 Introduction

This application note describes the implementation of an averaging sample buffer by using the eQADC module in the 5510 family. This solution provides you the way to average multiple samples from the eQADC module at a certain sample rate.

In real-world applications there are occasions when it may be inappropriate to use a single ADC conversion result. Although the result is a correct conversion of the analog value that existed at input an external disturbance may have caused it to be unrepresentative of the signal of interest.

This is of particular concern, where a short series of results is disturbed by some undesired noise at input.

One way of addressing this problem and rejecting disturbance, is to average the incoming ADC values and extract the longer-term value of the signal. This is most effective when the desired signal changes slowly.

Contents

freescale semiconductor

Averaging can be applied over a number of samples, and output based on the most recent results. Each time a result is taken from the ADC module, the oldest previous result is deleted, and the average is recalculated using the remaining results and the newest value.

2 Theory of Operation

The ADC average driver bases its operation on the boxcar filter. The boxcar filter is a simple digital filter with a finite impulse response (FIR). It is used to create an output series composed by the runtime mean of input series values.

The simplicity of a boxcar filter is due to the fact that each sample is multiplied by a unit-valued coefficient that performs a filtering task without multiples. The overhead of this filter is from the typical multiply and accumulate to accumulate.

After sample accumulation is finished, the final step in the FIR is to normalize all samples by dividing the calculated result by the sum of filter coefficients, in this case it is the number of samples to be averaged.

Figure 1 shows the impulse response of a boxcar filter.

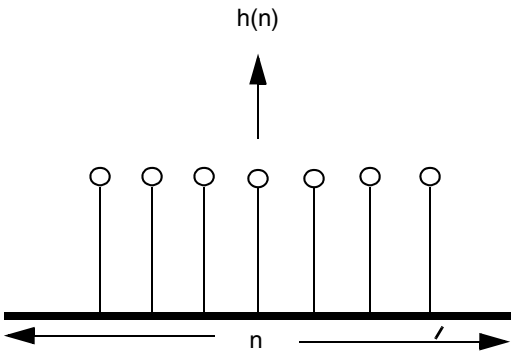

Figure 1. Impulse Response of Boxcar Filter

Figure 2 shows the frequency response of a boxcar filter using an 8-step configuration. The frequency responses of this filter gives a general low-pass characteristic with lots of lobs. See Figure 2.

This has the effect of preserving the desired, slowly changing (low frequency) signal. It is important to bear this frequency response in mind when considering the type of noise that the averaging filter rejects.

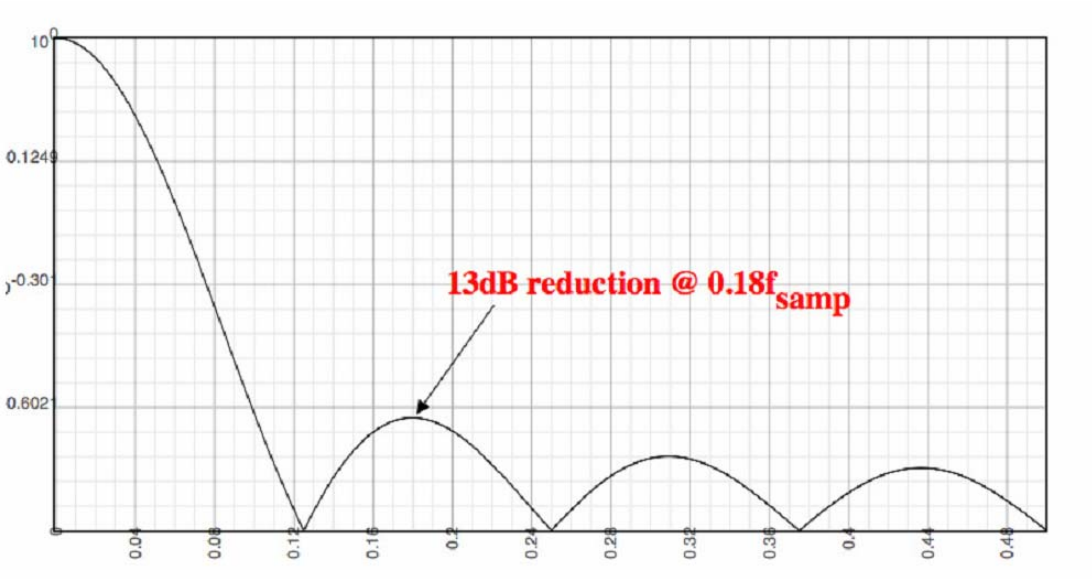

Figure 2. Frequency Response of 8-Step Boxcar Filter

Equation 1 is the mathematical expression of a boxcar filter. Describing the mathematical principle of a boxcar filter is:

$$y(n)) = \sum_{i-0}^{P} b_i x(n-i)$$

Eqn. 1

3 ADC Average Configuration and Control Parameters

This section describes how to configure the ADC average driver to operate during run-time and describes pre-compile statements.

3.1 Pre-Compile Definitions

The ADC average driver has many pre-compile definitions to enable and disable certain functionalities in the driver. All pre-compile definitions are located in file AdcFltr_Cfg.h with the exception of macro named SYS_FREQ located in the System_Cfg.h file.

3.1.1 SYS_FREQ

This definition describes the operating system frequency of the whole driver. The ADC average driver uses this definition to calculate the timing references of the PIT channel used to manage the task scheduler. This field must be captured at hertz units as shown below:

```
#define SYS_FREQ  (uint32_t) 64000000U  /* Hz units */
```

ATDFLTR_0_AVERAGECHANNELS — This macro defines the number of ADC channels that the software averages. The value of this macro must be between 1 and the maximum number of ADC channels allowed in the MPC5510 family.

```
#define ATDFLTR_0_AVERAGECHANNELS      6U
```

ATDFLTR_0_AVERAGETAPS — This macro defines the number of values to include in the average calculation. It indicates to the ADC average driver how many samples will be used to calculate the final average value per ADC channel.

The driver allows only certain predefined average taps validated by the ADC average error layer. The tap ranges allowed by the driver are: 2, 4, 8, 16, 32, and 64.

The main reason why only these ranges are allowed by the driver is to avoid using the division instruction. Instead the driver performs the shifting operation to avoid division instructions. This provides the driver with better performance.

These tap ranges are represented with the following macros:

```
#define _2_SAMPLE_DEPTH_         1U
#define _4_SAMPLE_DEPTH_         2U
#define _8_SAMPLE_DEPTH_         3U
#define _16_SAMPLE_DEPTH_        4U
#define _32_SAMPLE_DEPTH_        5U
#define _64_SAMPLE_DEPTH_        6U
```

To set the number of taps (or samples) the driver uses to calculate the average value, the user must first set the macro named `SAMPLE_AVERAGE_VALUE` with one of the parameters described above.

```
#define SAMPLE_AVERAGE_VALUE      _8_SAMPLE_DEPTH_
```

During run time, the ADC average driver uses the `ATDFLTR_0_AVERAGETAPS` macro definition to indicate the number of taps the driver averages per ADC channel.

```
#define ATDFLTR_0_AVERAGETAPS  SAMPLE_AVERAGE_VALUE
```

ATDFLTR_0_AVERAGEINTERRUPT — When enabled to the on state, this macro causes the driver to raise an interrupt by calling the function previously configured in the macro, *ATDFLTR_0_FILTER_FNC.*

```
#define ATDFLTR_0_AVERAGEINTERRUPT ON
#define ATDFLTR_0_FILTER_FNC &vfnFilter_1_End
```

ADC_FLTR_ERROR_DETECT — When enabled to the on state, the ADC average error layer enables parameter validation assigned through the ADC average functions.

```
#define ADC_FLTR_ERROR_DETECT   ON
```

3.2 Run-Time Parameters

The run-time parameters in the ADC average driver helps track information about the number of samples the driver retrieves per ADC channel. The run-time parameters contain the buffers to store each of the samples per ADC channel.

3.2.1 ADC Average Control Structure

```
typedef struct
{
  uint16_t numTAPS:6;
  uint16_t numChannels:6;
  uint16_t interruptCPU:1;
  void (*AverageEndFnc)(void);
} tAdcFltr_ChannelConfig;
```

numTAPS — Contains the number of samples to take for each ADC channel.

numChannels — Indicates the total number of ADC Channels to convert.

interruptCPU — Causes the driver to jump into a predefined function to indicate the driver has averaged the configured ADC channels.

3.2.2 ADC Average Channel Structure

```
typedef struct
{
  uint16_t*  pSampleBuffer;
  vuint16_t* pResultRegister;
  uint16_t   u16AccumulateSample;
  uint16_t*  pFilterResult;
} tAdcFltr_ChannelStruct;
```

pSampleBuffer — Points to the stored ADC samples. This pointer is used by the driver to store the samples per ADC channel.

pResultRegister — Points to the ADC result variable managed by the eDMA module. The eDMA module stores the ADC results in the rQUEUE array.

u16Accumulate — This variable accumulates the sample results aquired per ADC channel.

pFilterResult — Points to the averaged results from every configured ADC channel.

4 Required Resources that Operate ADC Average Driver

Table 1 shows the on-chip resources required to operate the ADC average driver.

Table 1. ADC Average Required Resources

Parameter	Value
Mandatory peripheral use	ADC channels as required
Mandatory peripheral use	eDMA channels 0 & 1 required
Optional peripheral use	PIT channel

The driver requires the availability of ADC channels to perform an average of each sample taken. If a specific sampling frequency is required, then the ADC average driver must make use of any available PIT channel.

5 Driver Implementation

5.1 ADC Average Initialization

The driver contains a routine that initializes the control parameters of the driver and configures the ADC module to operate at a certain conversion frequency, bit resolution, and so on.

The function, vfnAdcFltr_Init initializes three filter result parameters: the average value of each ADC channel, the sample buffer pointer, and the pointer where the driver reads the ADC converted value.

The ADC average initialization function also validates the configuration structure parameters to avoid unexpected behavior in the driver.

The second parameter validated by the function is the number of samples to convert for each ADC channel. If the function detects an invalid condition outside of the allowed tap ranges, the initialization flow aborts immediately leaving the ADC registers and buffers untouched. The allowed sample ranges are: 2, 4, 8, 16, 32, and 64.

After the initialization function validates correct configuration parameters, the control buffers are initialized and the ADC module is configured based on the parameters located in the file, Adc_Cfg.h.

5.2 ADC Average Driver Configuration

The ADC Average driver operates based on a few configuration parameters located in the file, AdcFltr_Cfg.h.

These parameters are the ADC channels to average, the number of samples taken per ADC channel, the total number of ADC channels to average, and the function to jump into, after the driver finishes averaging. The procedure to configure the driver is explained below:

1. Configuring the ADC channels needed to average. These channels are indicated in the configuration array located in the file named AdcFltr_Cfg.h

```
uint16_t u16AdcChannel_Convert[ATDFLTR_0_AVERAGECHANNELS] =
{
        ADC_CHANNEL_0,  /* Adc Channel 0 */
        ADC_CHANNEL_1,  /* Adc Channel 1 */
        ADC_CHANNEL_2,  /* Adc Channel 2 */
        ADC_CHANNEL_3,  /* Adc Channel 3 */
  ADC_CHANNEL_4,  /* Adc Channel 4 */
        ADC_CHANNEL_5   /* Adc Channel 5 */
};
```

2. The total amount of ADC channels to average

```
#define ATDFLTR_0_AVERAGECHANNELS   6
```

3. The number of samples to average by the driver.

```
#define SAMPLE_AVERAGE_VALUE   _8_SAMPLE_DEPTH_ /* 8-Step samples */
```

4. Enable or disable the end filter interrupt function. This condition indicates the driver to jump into a specific predefined function when the channel averaging finishes.

```
#define ATDFLTR_0_AVERAGEINTERRUPT     ON
#define ATDFLTR_0_FILTER_FNC           &vfnFilter_1_End
```

Figure 3shows the ADC average configuration structure.

```
tAdcFltr_FilterStruct sATD0 =
 {
AdcFltr_0_ChannelStruct,
{
ATDFLTR_0_AVERAGETAPS,
ATDFLTR_0_AVERAGECHANNELS,
ATDFLTR_0_AVERAGEINTERRUPT,
      ATDFLTR_0_FILTER_FNC
}
  };
```

Figure 3. ADC Average Configuration Structure.

6 ADC Averaging

The driver uses the function `vfnAdc_Sample_Acquire` to trigger the ADC channels to sample. The function triggers the eDMA module to start sending commands to the ADC module and reads the converted results.

Every time this function is called, the eDMA sends commands to convert four ADC channels at the same time. This feature provides better performance to the driver when many ADC channels are requested to average.

The ADC averaging function figures out sending commands to the ADC module when less than four channels are configured to be averaged. The function uses a circular command buffer to store conversion commands. If only two ADC channels need to be averaged, the function fills the rest of the two command buffers with the same commands from first ADC channels.

If a specific sampling frequency is required, a PIT channel is initialized to call this function (`vfnAdc_Sample_Acquire`), periodically.

7 Sample Application

A sample application is provided within this application note. It demonstrates the ADC average driver operation. A multi-task scheduler has been mounted in the sample application to retrieve the averaged results.

7.1 Initialization

Figure 4 shows the sample application flow chart.

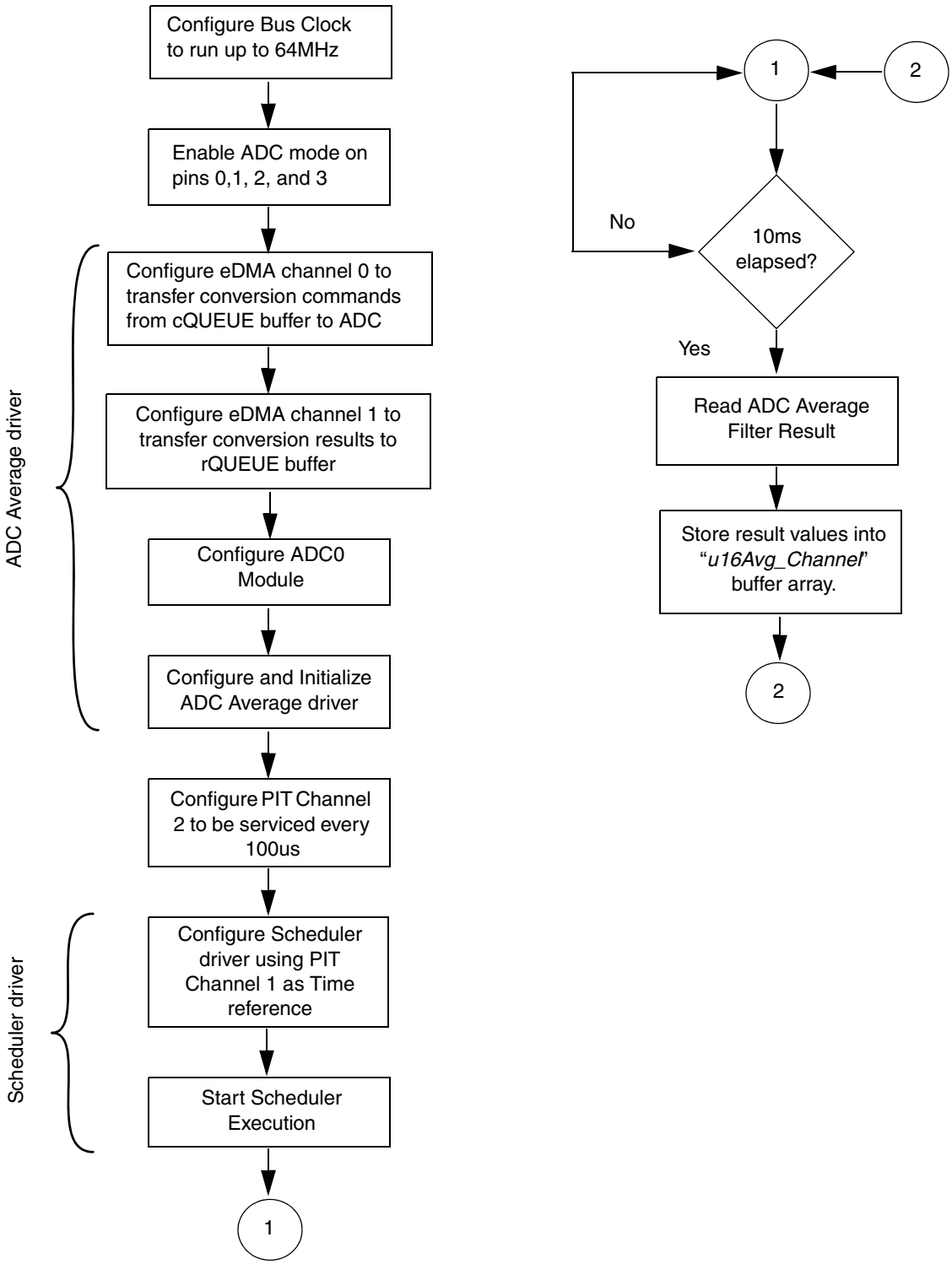

Figure 4. Sample Application Flow Chart

The first action the sample application performs, is to configure the microcontroller bus clock to run-up to 64 MHz. After the bus clock is set, the microcontroller pin numbers 0, 1, 2, and 3 are configured to enable the ADC mode.

When ADC functionality pins are enabled, the eDMA channels 0 and 1 are configured to handle transfers and store conversions from the ADC module.

eDMA channel 0 is used to handle conversion command transfers to the ADC module and eDMA channel 1 is configured to store result conversions. A conversion command buffer named cQUEUE is used to transfer commands from the buffer to the ADC module. A result conversion buffer rQUEUE is used to store converted results from ADC channels.

When the eDMA channels are properly configured, the ADC initialization is executed to start configuring the general parameters of the ADC module, such as the ADC conversion rate, bit resolution, and result alignment. All these parameters are taken from the ADC configuration file Adc_Cfg.h.

PIT channel 1 is enabled to be serviced every 100 µs. Every 100 µs the ADC averaged function is called to start averaging the configured ADC channels. This interrupt time base provides a frequency conversion rate of 10 KHz.

Configuring the ADC average routine is the last initialization step. The routine configures the cQUEUE buffer to indicate to the ADC module that channels 0, 1, 2, and 3 need to be converted. The following configuration steps are set to operate with this sample application:

1. Maximum number of ADC channels equal to 4
2. Number of samples per ADC channel equal to 8
3. The driver generates an interrupt after all averaged ADC channels. This condition is true whenever the interrupt macro in the driver is enabled.

Figure 5 shows configuration parameters used in the sample application.

```
uint16_t u16AdcChannel_Convert[ATDFLTR_0_AVERAGECHANNELS] =
{
        ADC_CHANNEL_0,  /* Adc Channel 0 */
        ADC_CHANNEL_1,  /* Adc Channel 1 */
        ADC_CHANNEL_2,  /* Adc Channel 2 */
        ADC_CHANNEL_3  /* Adc Channel 3 */
};

/* Filter struct     */
tAdcFltr_FilterStruct sATD0 =
{
  AdcFltr_0_ChannelStruct,
  {
    ATDFLTR_0_AVERAGETAPS,
    ATDFLTR_0_AVERAGECHANNELS,
    ATDFLTR_0_AVERAGEINTERRUPT,
    ATDFLTR_0_FILTER_FNC
  }
};

/** Indicates the driver the number of samples to be acquired per channel          */
#define SAMPLE_AVERAGE_VALUE      _8_SAMPLE_DEPTH_
/** 1..8/16 ; number of active channels used on this ADC            */
#define ATDFLTR_0_AVERAGECHANNELS      6U
/* 1..64 ; size of filter - shared among all filters on this ADC    */
#define ATDFLTR_0_AVERAGETAPS          SAMPLE_AVERAGE_VALUE
/** If TRUE then XGATE interrupts CPU when done                     */
#define ATDFLTR_0_AVERAGEINTERRUPT     ON
/** Filter End function used to indicate that all samples were averaged */
#define ATDFLTR_0_FILTER_FNC           &vfnFilter_1_End
/** Enables or Disables the Error Layer                             */
#define ADC_FLTR_ERROR_DETECT          ON
```

Figure 5. ADC Average Configuration Parameters

7.2 Scheduler Initialization

After initialization, the scheduler function `vfnScheduler_Init` runs to configure PIT channel number 2 as a time base reference.

A macro definition `LOOP_TIME_5ms` located at Pit.h file calculates the value of the PIT channel 2 counter to generate an interrupt service every 5 ms. This macro makes use of the actual system clock used by the sample application that is 64 MHz.

After the `vfnScheduler_Init` function is called, the scheduler execution remains in hold state until the function `vfnStart_Scheduler` starts executing the scheduler. This function enables PIT channel 2 timer to start counting and enables complete operation of the scheduler.

This is the schedule initialization procedure. The correct initialization and execution of the scheduler.

```
vfnScheduler_Init();  /* Initialize Scheduler timebase         */
vfnStart_Scheduler(); /* Start Tasks execution                 */
```

NOTE

It is important to mention, the scheduler driver is not part of the ADC average driver. The scheduler driver helps demonstrate the functionality of the ADC average driver.

7.3 Sample Application

Every 10 ms the scheduler reads the ADC average filter result buffer to retrieve the values averaged from channels 0, 1, 2, and 3. This process is executed every 10 ms and is mounted under a function-like macro EXECUTE_10MS_TASKS.

Figure 6 shows scheduler execution and the 10 ms thread called to retrieve the averaged channel values.

```
while (gu8SleepModeEnabled == 0)
  {
       if ((gu8Scheduler_Flag & (uint8_t)0x01) == (uint8_t)0x01)
        {
            /*-- Allow 10 ms periodic tasks to be executed --*/
            EXECUTE_10MS_TASKS();

            /* Scheduled tasks finished, clear control flag */
            gu8Scheduler_Flag = (uint8_t)0x00;
        }
        else
        {
            if ((gu8Scheduler_Flag & (uint8_t)0x02) == (uint8_t)0x02)
            {

                /*-- Allow 20 ms periodic tasks to be executed --*/
                 EXECUTE_20MS_TASKS();

                      /* Scheduled tasks finished, clear control flag */
                gu8Scheduler_Flag = (uint8_t)0x00;
            }
            else
            {
                if ((gu8Scheduler_Flag & (uint8_t)0x04) == (uint8_t)0x04)
                {
                    /*-- Allow 40 ms periodic tasks to be executed --*/
                     EXECUTE_40MS_TASKS();

                    /* Scheduled tasks finished, clear control flag */
                    gu8Scheduler_Flag = (uint8_t)0x00;
                }
                else
                {
                    if ((gu8Scheduler_Flag & (uint8_t)0x08) == (uint8_t)0x08)
                    {
                        /*-- Allow 80 ms group A periodic tasks to be executed --*/
                         EXECUTE_80MS_A_TASKS();
                              /* Scheduled tasks finished, clear control flag */
                        gu8Scheduler_Flag = (uint8_t)0x00;
                    }
```

Figure 6. Scheduler Execution

The averaged results are stored in the buffer variable `u16Avg_Channel`. The buffer array contains the results from the four configured ADC channels.

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
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN3813
Rev. 0
04/2009