Freescale Semiconductor
Application Note

Document Number: AN4170
Rev. 1, 2010

IRTC Driver for MC9S08GW64

**by: Tanya Malik
Microcontroller Solutions Group
Noida
India**

Contents

1 Introduction

This document describes a driver for Digital IP Robust Real Time Clock (IRTC) allowing users to customize the possible configurations for this peripheral. These configurations include: time keeping, calendaring, auto adjustment for day light saving, programmable alarm, minute countdown timer, protection against tampering, and battery operation.

The driver was tested at DEMO9S08GW64. The software architecture was designed to provide seamless migration between devices that posses the same peripheral module.

This document is intended to be used by software development engineers and test engineers who has to use the microcontrollers with the IRTC.

In this application note, the driver interfaces are explained. Various applications for MC9S08GW64 can make use of this driver. The following section describes the details and the steps for creating an application using it.

1.1 IRTC features

- Time keeping functions by second, minute, and hour counters
- Calendaring functions via date, day-of-week, month, and year counters
- Alarm set for specific hour, minute, and second
- Countdown timer with minute resolution
- Daylight saving adjustment

- Leap year automatic adjustment
- Temperature compensation

2 Software driver description

The IRTC driver is provided as C code files. You can add these files to your applications. With the integration of IRTC driver, you can call IRTC driver API functions to use the IRTC functionality in your application.

There are four files associated with the IRTC driver. The next section in this document describes it in more detail.

- **rtc_driver_ext.c**: It is the main file for the driver. It contains the various high level API definitions exposed to the applications for IRTC functionality.
- **rtc_driver_ext.h:** This file contains the high level API declarations. It contains the macros to be used by the user while using the IRTC functions. This file is included in the application that intend to use the IRTC driver.
- **rtc_driver_int.c:** It contains the low level functions that are called inside the high level API functions.
- **rtc_driver_int.h:** It contains the declaration of all the functions defined in *rtc_driver_int.c*.

NOTE

The IRTC driver code is available in a zipped file named **AN4170SW.zip**

2.1 rtc_driver_ext.h

The macros provided in this section are passed as arguments to the respective functions to get the required configuration.

Macro	Description

`#define RTC_CLKOUT_DISABLE`
`#define RTC_CLKOUT_1HZ`
`#define RTC_CLKOUT_OSC_CLOCK`
	Selects the RTC clock out.

`#define RTC_DAYLIGHTSAVINGS_DISABLE`
`#define RTC_DAYLIGHTSAVINGS_ENABLE`
	Enables or disable the day light saving in RTC.

`#define RTC_ALARM_MATCH_SMH 0  /*Alarm matches only seconds, minutes, and hours*/`
`#define RTC_ALARM_MATCH_SMHD 1 /*Alarm matches only seconds, minutes, hours, and  days */`
`#define RTC_ALARM_MATCH_SMHDM 2 /*Alarm matches only seconds, minutes, hours, days, and months */`
`#define RTC_ALARM_MATCH_SMHDMY 3 /*Alarm matches only seconds, minutes, hours, days, months, and years*/`
	Selects the alarm match.

`#define RTC_ISR_FRE512HZ  IRTC_ISR_SAM7_MASK`
`#define RTC_ISR_FRE256HZ  IRTC_ISR_SAM6_MASK`
`#define RTC_ISR_FRE128HZ  IRTC_ISR_SAM5_MASK`
`#define RTC_ISR_FRE64HZ   IRTC_ISR_SAM4_MASK`
`#define RTC_ISR_FRE32HZ   IRTC_ISR_SAM3_MASK`
`#define RTC_ISR_FRE16HZ   IRTC_ISR_SAM2_MASK`
`#define RTC_ISR_FRE8HZ    IRTC_ISR_SAM1_MASK`
`#define RTC_ISR_FRE4HZ    IRTC_ISR_SAM0_MASK`
`#define RTC_ISR_FRE2HZ    IRTC_ISR_2HZ_MASK`
`#define RTC_ISR_FRE1HZ    IRTC_ISR_1HZ_MASK`
`#define RTC_ISR_MIN       IRTC_ISR_MIN_MASK`
`#define RTC_ISR_HR        IRTC_ISR_HR_MASK`
`#define RTC_ISR_DAY       IRTC_ISR_DAY_MASK`
`#define RTC_ISR_ALM       IRTC_ISR_ALM_MASK`
`#define RTC_ISR_CDT       IRTC_ISR_CDT_MASK`
	Interrupt Masks

`#define RTC_ISR_BAT_TMPR /*battery tamper*/`
`#define RTC_ISR_TMPR1  /*tamper1*/`
`#define RTC_ISR_TMPR2   /*tamper2*/`
	Specifies which tamper has occurred.

Macro	Description
enum month_names{January=1, February, March, April, May, June, July, August, September, October, November, December};	Structure which defines the month names. One is assigned to January, two to February, and so on.
enum weekday_names {Sunday=0, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};	Structure to define weekdays. 0 has been assigned to Sunday, 1 to Monday, and so on.

2.2 rtc_driver_ext.c

This file contains the various high level functions that the users can directly use.

2.2.1 rtc_setConfig()

Description:

This function is used to initialize the RTC by configuring the internal registers.

Prototype:

```
unsigned char rtc_setConfig(unsigned char rtc_clkout, unsigned char daylightsavings_enable,
unsigned char softreset,  void (*func)(unsigned int interrupt_mask, unsigned char tamper_mask));
```

Input Parameters:

1. *rtc_clkout*—Selects the clock out of RTC by using the macros RTC_CLKOUT_DISABLE, RTC_CLKOUT_1HZ, RTC_CLKOUT_OSC_CLOCK
2. *daylightsavings_enable*—Enables or disables the daylight savings in rtc using the macros: RTC_DAYLIGHTSAVINGS_DISABLE, RTC_DAYLIGHTSAVINGS_ENABLE
3. *softreset*—For software reset 1 is passed; 0 is passed for hard reset.
4. *func*—Used only in case of interrupts. The user can pass the address of the callback function, which the user wants to call in case of interrupt or pass zero, in case the user does not want to use a callback function.

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
void isr_callback() /* Call back function defined */
{}
if(! rtc_setConfig(RTC_CLKOUT_1HZ, RTC_DAYLIGHTSAVINGS_DISABLE, 0, &isr_callback))
{
/* Sucess */
}
```

Configures the RTC with enabling the RTC 1Hz clock out, disabling the day light savings, enabling the hard reset and passing the address of the callback function.

2.2.2 rtc_setAlarmConfig()

Description:

This function is used to configure the alarm settings in RTC.

Prototype:

```
unsigned char rtc_setAlarmConfig(unsigned char alarm_match);
```

Input Parameters:

alarm_match—Selects the alarm configuration using the macros

```
RTC_ALARM_MATCH_SMH    /* Alarm matches only seconds, minutes, and hours */
RTC_ALARM_MATCH_SMHD   /* Alarm matches only seconds, minutes, hours, and days */
RTC_ALARM_MATCH_SMHDM  /* Alarm matches only seconds, minutes, hours, days, and months*/
RTC_ALARM_MATCH_SMHDMY  /* Alarm matches only seconds, minutes, hours, days, months, and years
 */
```

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(!rtc_setAlarmConfig(RTC_ALARM_MATCH_SMH ))
{
       /* Success; */
}
```

Configures the RTC alarm for matching the seconds, minutes, and hours.

2.2.3 rtc_setCompensation()

Description:

This function sets the compensation settings for the RTC temperature compensation by setting the time of the interval over which the compensation is to be carried out and setting the the compensation value.

Prototype:

```
void rtc_setCompensation(unsigned char comp_interval, unsigned char comp_value);
```

Input Parameters:

- *comp_interval*—Indicates the window over which the compensation has to be carried out. Minimum interval is 1 second and maximum is 255 seconds. A value of zero disables the compensation.
- *comp_value*—Two’s complement number which indicates the number of oscillator clock cycles the RTC requires to compensate for the specified compensation interval. *Range:* –128 to +127. A value of zero indicates no compensation is needed.

Output Parameters:

None

Example:

```
rtc_setCompensation(20, 28);
```

Sets the compensation interval of 20 seconds and compensation value of 28.

2.2.4 rtc_setYear()

Description:

This function is used to set the year in RTC.

Prototype:

```
unsigned char rtc_setYear(unsigned int year);
```

Input Parameters:

year—Enter year which you want to set

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(!rtc_setYear(2010))
{
/* Success */
}
```

Sets the year to 2010.

2.2.5 rtc_setMonth()

Description:

This function is used to set the month in RTC.

Prototype:

```
unsigned char rtc_setMonth(enum month_names month);
```

Input Parameters:

month—Enter month of your choice—January, February, March, April, May, June, July, August, September, October, November, December

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(!rtc_setMonth(January))
{
/* Success */
}
```

Sets the month to January.

2.2.6 rtc_setDayAndDate()

Description:

This function is used to set the day and date in RTC.

Prototype:

```
unsigned char rtc_setDayAndDate(enum weekday_names weekday, unsigned char date);
```

Input Parameters:

- *weekday*—Enter the day you want to set from the members of the structure—Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
- *date*—Enter date between 1 to 31

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setDayAndDate(Sunday, 20) )
{
/* Success*/
}
```

Sets the day to Sunday and date to 20.

2.2.7 rtc_setHour()

Description:

This function is used to set the hour in RTC Clock.

Prototype:

```
unsigned char rtc_setHour(unsigned char hour);
```

Input Parameters:

hour—Enter any value 0 - 23; Software interprets 0 - 11 as AM and 12 - 23 as PM

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setHour(11) )
{
/* Success*/
}
```

Sets the hour to 11am.

2.2.8 rtc_setMin()

Description:

This function is used to set the minutes in RTC Clock.

Prototype:

```
unsigned char rtc_setMin(unsigned char min);
```

Input Parameters:

min—Enter any value from 0 - 59

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setMin(40) )
{
/*Success*/
}
```

Sets the minute to 40.

2.2.9 rtc_setSeconds()

Description:

This function is used to set seconds in RTC clock.

Prototype:

```
unsigned char rtc_setSeconds(unsigned char sec);
```

Input Parameters:

sec—Enter any value between 0 - 59

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setSeconds(50) )
{
/* Success*/
}
```

Sets the seconds to 50.

2.2.10 rtc_setDateAndTime()

Description:

This function is used to set the year in RTC.

Prototype:

```
unsigned char rtc_setDateAndTime(unsigned int year, enum month_names month,
                          unsigned char date,enum weekday_names weekday,
                                 unsigned char hour, unsigned char minutes,
                                 unsigned char seconds);
```

Input Parameters:

- *year*
- *month*
- *weekday*
- *hour*

- *minutes*
- *seconds*

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setDateAndTime(2010, January, 20, Monday, 11, 40, 20) )
{
/*Success*/
}
```

Sets the date to 20th Jan 2010 Monday and the time to 11:40:20 in am.

2.2.11 rtc_getYear()

Description:

This function is used to get the present year from RTC.

Prototype:

```
unsigned int rtc_getYear(void);
```

Input Parameters:

None

Output Parameters:

Returns the value of the year in integer

Example:

```
unsigned int year;
year = rtc_getYear();
```

2.2.12 rtc_getMonth()

Description:

This function is used to get the present value of the month from the RTC.

Prototype:

```
enum month_names rtc_getMonth(void)
```

Input Parameters:

None

Output Parameters:

Returns value of the current month stored in the RTC

Example:

```
enum month_names month;
month = rtc_getMonth();
```

2.2.13 rtc_getDayAndDate()

Description:

This function is used to get the present day and date from the RTC.

Prototype:

```
void rtc_getDayAndDate(enum weekday_names *weekday, unsigned char *date)
```

Input Parameters:

- *weekday*—Pass the address of the variable where you want to store the weekday
- *date*—Pass the address of the variable where you want to store the date

Output Parameters:

None

Example:

```
enum weekday_names weekday;
unsigned char date;
rtc_getDayAndDate(&weekday, &date);
```

2.2.14 rtc_getMin()

Description:

This function is used to get the current minutes from the RTC.

Prototype:

```
unsigned char rtc_getMin(void)
```

Input Parameters:

None

Output Parameters:

Returns the value of the minutes in unsigned char

Example:

```
unsigned char minutes;
minutes = rtc_getMin();
```

2.2.15 rtc_getHour()

Description:

This function is used to get the current hour from the RTC.

Prototype:

```
unsigned char rtc_getHour(void)
```

Input Parameters:

None

Output Parameters:

Returns the value of the hour in unsigned char

Example:

```
unsigned char hour;
hour = rtc_getHour();
```

2.2.16 rtc_getSeconds()

Description:

This function is used to get the current seconds from the RTC.

Prototype:

```
unsigned char rtc_getSeconds(void)
```

Input Parameters:

None

Output Parameters:

Returns the value of the seconds in unsigned char

Example:

```
unsigned char seconds;
seconds = rtc_getsSeconds();
```

2.2.17 rtc_setAlarm()

Description:

This function is used to set the alarm for the RTC.

Prototype:

```
unsigned char rtc_setAlarm(unsigned int year, enum month_names month, unsigned char date,
unsigned char hour, unsigned char minutes, unsigned char seconds)
```

Input Parameters:

- *year*
- *month*
- *date*
- *hour*
- *minutes*
- *seconds*

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setAlarm(2010,January,30,11,40,10) )
{
/* Success*/
}
```

2.2.18 rtc_getAlarmYear()

Description:

This function is used to get the year set in the alarm.

Prototype:

```
unsigned int rtc_getAlarmYear(void)
```

Input Parameters:

None

Output Parameters:

Returns the value of the year in integer

Example:

```
unsigned int year;
year = rtc_getAlarmYear();
```

2.2.19 rtc_setBatteryTamperConfig()

Description:

This function is used to configure the battery tamper control.

Prototype:

```
unsigned char rtc_setBatteryTamperConfig(unsigned char state)
```

Input Parameters:

state

- **0:** To disable the tamper status bit
- **1:** To enable the tamper status bit

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
rtc_setBatteryTamperConfig(1);
```

2.2.20 rtc_setTamper1Config()

Description:

This function is used to configure the tamper1 control.

Prototype:

```
unsigned char rtc_setTamper1Config(unsigned char state, unsigned char filter_clk,
                            unsigned char filter_duration, unsigned char pol)
```

Input Parameters:

- *state*

  - **0:** To disable the tamper1 status bit
  - **1:** To enable status1 tamper bit

- *filter_clk*—Selects the clock for tamper1 by passing the following values:
  - **0:** Clock to tamper filter1 is 32.768 kHz (Oscillator clock)
  - **1:** Clock to tamper filter1 is 512 Hz

- *filter_duration*—Users can select the tamper1 filter duration. This bit indicates the number of tamper filter clock cycles for which the tamper_detect[1] signal must remain stable before being detected as a tamper. The user can pass the following values according to the requirements:
  - **0:** Disables the filtering operation
  - **1 to 63:** Number of tamper filter clock cycles to be counted when tamper is asserted

- *pol*—Users can control tamper1 polarity by passing 0 or 1
  - **0:** Tamper 1 is active high
  - **1:** Tamper 1 is active low

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if( ! rtc_setTamper1Config(1, 0, 4, 0) )
{
/*Success*/
}
```

Enables the tamper1 with oscillator clock as the tamper clock and four clock cycles as the filter duration and configures the tamper1 active high.

2.2.21 rtc_setTamper2Config()

Description:

This function is used to configure the tamper2 control.

Prototype:

```
unsigned char rtc_setTamper2Config(unsigned char state, unsigned char filter_clk, unsigned char
 filter_duration, unsigned char pol)
```

Input Parameters:

1. *state*—0 to disable the tamper2 status bit and 1 to enable the tamper2 status bit
2. *filter_clk*—Used to select the clock for tamper2 by passing the following values:
   a. **0:** Clock to tamper filter2 is 32.768 kHz (Oscillator clock)
   b. **1:** Clock to tamper filter2 is 512 Hz

3. *filter_duration*—Users can select the tamper2 filter duration. This bit indicates the number of tamper filter clock cycles for which the tamper_detect[2] signal must remain stable before being detected as a tamper. The user can pass the following values according to the requirements:
   a. **0:** To disable the filtering operation
   b. **1 to 63:** Number of tamper filter clock cycles to be counted when tamper is asserted

4. *pol*—Users can control the tamper2 polarity by passing 0 or 1
   a. **0:** Tamper2 is active high

b. **1:** Tamper2 is active low

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_setTamper2Config(1,0,4,0) )
{
/*Success*/
}
```

Enables the tamper2 with oscillator clock as the tamper clock and four clock cycles as the filter duration and configures the tamper2 active high.

2.2.22 rtc_setCountDownTimer()

Description:

This function is used to configure the countdown timer to generate an interrupt after specified minutes.

Prototype:

```
unsigned char rtc_setCountDownTimer(unsigned char minutes);
```

Input Parameters:

minutes—Number of minutes to be countdown

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(!rtc_setCountDownTimer(5))
{
/*Success*/
}
```

Sarts a countdown timer in minutes.

2.2.23 rtc_EnableInterrupt()

Description:

This function enables the particular interrupt.

Prototype:

```
unsigned char rtc_EnableInterrupt(unsigned int interrupt_byte);
```

Input Parameters:

interrupt_byte—Masks the specific interrupt from the following masks

```
RTC_ISR_FRE512HZ\RTC_ISR_FRE256HZ\RTC_ISR_FRE128HZ
  \ RTC_ISR_FRE64HZ \RTC_ISR_FRE32HZ \RTC_ISR_FRE16HZ \RTC_ISR_FRE8HZ
 \RTC_ISR_FRE4HZ\RTC_ISR_FRE2HZ\RTC_ISR_FRE1HZ\RTC_ISR_MIN
\ RTC_ISR_HR \RTC_ISR_DAY\RTC_ISR_ALM\RTC_ISR_CDT
```

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_EnableInterrupt(RTC_ISR_FRE1HZ))
{
/*Success*/
}
```

2.2.24 rtc_DisableInterrupt()

Description:

This function disables the particular interrupt.

Prototype:

```
unsigned char rtc_DisableInterrupt(unsigned int interrupt_byte)
```

Input Parameters:

interrupt_byte—Masks the specific interrupt from the following masks

```
RTC_ISR_FRE512HZ\RTC_ISR_FRE256HZ\RTC_ISR_FRE128HZ
  \ RTC_ISR_FRE64HZ \RTC_ISR_FRE32HZ \RTC_ISR_FRE16HZ \RTC_ISR_FRE8HZ
 \RTC_ISR_FRE4HZ\RTC_ISR_FRE2HZ\RTC_ISR_FRE1HZ\RTC_ISR_MIN
\ RTC_ISR_HR \RTC_ISR_DAY\RTC_ISR_ALM\RTC_ISR_CDT
```

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_DisableInterrupt(RTC_ISR_FRE1HZ))
{
/* Success*/
}
```

2.2.25 rtc_ClearInterrupt()

Description:

This function clears the particular interrupt.

Prototype:

```
unsigned char rtc_ClearInterrupt(unsigned int interrupt_byte)
```

Input Parameters:

interrupt_byte—Masks the specific interrupt from the following masks

```
RTC_ISR_FRE512HZ\RTC_ISR_FRE256HZ\RTC_ISR_FRE128HZ
  \ RTC_ISR_FRE64HZ \RTC_ISR_FRE32HZ \RTC_ISR_FRE16HZ \RTC_ISR_FRE8HZ
 \RTC_ISR_FRE4HZ\RTC_ISR_FRE2HZ\RTC_ISR_FRE1HZ\RTC_ISR_MIN
\ RTC_ISR_HR \RTC_ISR_DAY\RTC_ISR_ALM\RTC_ISR_CDT
```

Output Parameters:

Returns 0/1

0: Success

1: Failure

Example:

```
if(! rtc_ClearInterrupt(RTC_ISR_FRE1HZ) )
{
/* Success*/
}
```

2.2.26 rtc_ISR()

Description:

This is the interrupt subroutine for RTC interrupts. The subroutine clears the rtc interrupt and jumps to the callback function, only if, the address of the call back function is passed in the RTC_Init function

Prototype:

```
void interrupt VectorNumber_Virtc rtc_ISR()
```

Input Parameters:

None

Output Parameters:

None

3 Assumptions

The descriptions in this document assumes that you have full knowledge of all the configuration registers of all the blocks in MC9S08GW64, especially RTC and Internal Clock Source (ICS) blocks.

4 Use Case

Include *rtc_driver_ext.h* in the main file and perform the following steps.

1. Define a callback function and write the action where you want to take on RTC interrupt.

   ```
   void isr_callback(unsigned int interrupt_mask, unsigned char tamper_mask)
   {
          return;
   }
   ```

2. Declare the variables needed to store the year, month, day, time, and so on as shown.

   ```
   unsigned int y;
   enum month_names m;
   unsigned char dat;
   enum weekday_names wday;
   unsigned char hr, mnts, secs;
   ```

3. Configure the RTC with the desired configuration by calling the following function.

   ```
   (void)rtc_setConfig(RTC_CLKOUT_1HZ, RTC_DAYLIGHTSAVINGS_DISABLE, 0, isr_callback);
   ```

   It configures the RTC with 1 HZ clock out, day light savings disabled, soft reset enabled and passes the address of the callback function declared.

4. Set the compensation window by calling the function.

   ```
   rtc_setCompensation(20, 28);
   ```

   Sets the compensation window of 20 seconds and 28 is the two’s complement of the number of oscillator cycles required for compensation

5. Set the date and time using the function.

   ```
   if(!rtc_setDateAndTime(2010, December, 3, Thursday, 15, 30, 30))
   {
       //success
   }
   ```

6. Read back the current date and time set.

   ```
   y = rtc_getYear();   /* gets back the year */
   m = rtc_getMonth();  /* reads back the month */
   rtc_getDayAndDate(&wday, &dat); /* reads back the day and date */
   hr = (unsigned char)rtc_getHour(); /* reads back the hour */
   mnts = (unsigned char)rtc_getMin(); /* reads back the minutes */
   secs = (unsigned char)rtc_getSeconds(); /* reads back the secs */
   ```

7. Sets the alarm

   ```
   if(!rtc_setAlarm(2010, June, 20, 12, 41, 45))
   {
   /* Success */
   }
   ```

5 Conclusion

This driver provides a software base for applications that needs the implementation of IRTC.

6 References

MC9S08GW64/MC9S08GW32 Reference Manual (document: MC9S08GW64RM)

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor
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 +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and sofware implementers to use Freescale Semiconductors 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 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 prodcuts 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 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 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 claims 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-complaint 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.

Document Number: AN4170
Rev. 1, 2010