Document Number: AN4478
Rev. 0, 03/2012

# Software Real Time Clock Implementation on MC9S08LG32

by: Nitin Gupta
Automotive and Industrial Solutions Group

Contents



## 1 Introduction

The MC9S08LG32 is a member of the Freescale HCS08 family MCUs. It uses the S08 core and integrates many peripherals, such as LCD, SPI, $I^2C$, SCI, ADC, and Real Time Counters. This application note describes how to initialize and maintain the Real Time Clock (RTC) in the MC9S08LG32 microcontroller family.

MC9S08LG32 Demo Board used for RTC setup is shown in Figure 1. The system communicates with the MC9S08LG32 target via a USB Background Debug Mode (BDM) interface. With BDM protocol, system can update the MC9S08LG32 firmware. RTC is implemented using the Real Time Counter IP (S08RTCV1) on MC9S08LG32. RTC can be displayed on the LCD connected on the board.

In this application note, the MC9S08LG32 RTC driver interfaces are explained. Various applications for MC9S08LG32 can make use of this driver. The following sections will describe the details and the steps for creating an application using it.

This application note does not cover how to configure the LCD. For details, please see AN3823 on http://www.freescale.com.

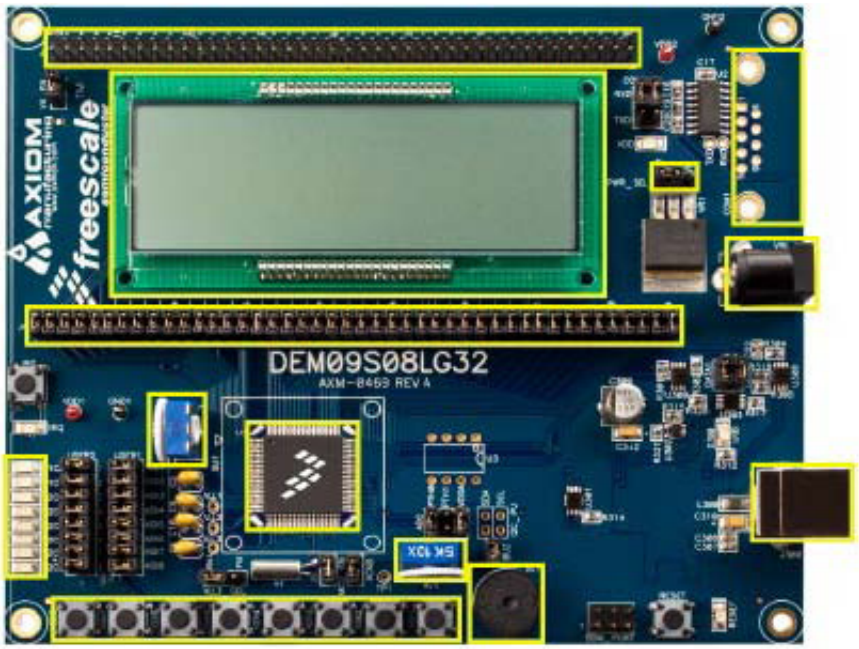


**Figure 1. MC9S08LG32 demo board setup**

# 2 Real Time Clock (RTC)

The RTC, or, sometimes referred to as time of the day, can be implemented using either a hardware or a software module. The primary function of an RTC implementation is to provide the information of time, day, week, month, and year.

A hardware RTC implementation refers to one that uses an external RTC hardware module that may be connected through $I^2C$ or SPI bus. On the other hand, some hardware RTC implementations are provided by an on-chip peripheral in the MCU. The advantage of hardware RTC is accuracy of time. Although not sought after for their accuracy, software RTCs can be a viable solution for some applications.

Software RTC can be implemented with a timer or a counter that generates an interrupt based on a specified time interval. The number of time intervals are counted and then converted to time. A one second time interval is a convenient configuration for the software RTC.

Since the software RTC function is not a part of the hardware, legacy systems can implement software RTC functionality with a firmware update. If the RTC is implemented in software, it will have lower system cost, requires fewer external components, and requires less power.

For MC9S08LG32, software RTC is implemented using Real Time Counter (S08RTCV1), configured to use the external oscillator of 32.768 kHz. S08RTCV1 accurately keeps the track of time in low-power mode, which is a critical parameter in automotive applications to prevent high battery discharge rates and conserve the battery charge between vehicle starts (recharge cycles).

Vehicle manufacturers expect that a vehicle should start even if the battery is not charged for long period (say 6-8 weeks), and still keep a very good accuracy on the daily digital clock. The MC9S08LG32 family of microcontrollers provides a range of features to enable users to achieve this goal.

# 3 RTC driver framework

The RTC driver is provided as “C” code files.

The driver consists of three files, namely:

- RTC.c
- RTC.h
- RTC_main.c

MC9S08LG32 RTC driver project is illustrated in Figure 2

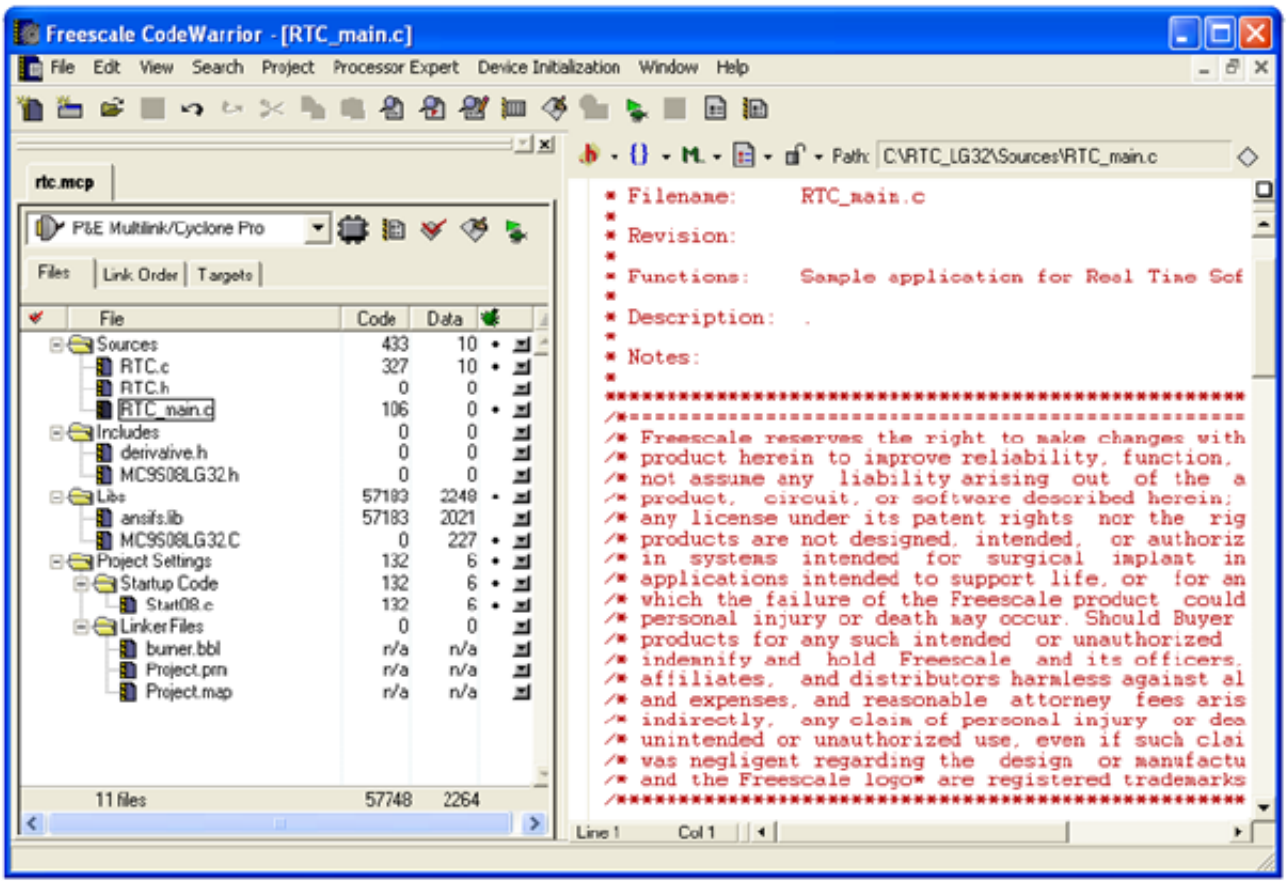


**Figure 2. MC9S08LG32 RTC driver**

The details of each of the files are given in the following subsections.

## 3.1 RTC.c

RTC.c consists of application programming interfaces (APIs) which enable the software RTC on the MC9S08LG32 series. The description of each API is as follows:

### 3.1.1 RTC_Init

This API initializes the RTC and the system clock. It uses a 32.768 kHz external crystal oscillator, having 30 ppm tolerance. During the initialization of the clock, the operating modes have been taken into consideration. As per the application, the operating mode can be either of the following:

- Normal mode: In this mode, the RTC interrupt will occur every 1 second.
- Stop3 mode: In this mode, the RTC interrupt will occur after every 8 seconds.

**Syntax**

```
void RTC_Init(void)
```

**Parameters**

None

**Return Value**

None

## 3.1.2 IRQ_Init

This API initializes the IRQ pin, which is connected to the ignition. It is assumed that a proper signal conditioning is present on the pin.

**Syntax**

```
void IRQ_Init(void)
```

**Parameters**

None

**Return Value**:

None

## 3.1.3 RTC_ISR

This interrupt service routine (ISR) is called each time the RTC interrupt occurs. In this, clock-time is updated as per the mode and interrupt flag is cleared. There can be two possibilities:

- Normal mode: In this mode, the time interval of the interrupt is 1 second (frequency ~ 1 Hz).
- Stop3 mode: In this low-power Stop3 mode, the RTC interrupt acts as a wakeup source for the MCU and the time interval of the interrupt is increased to 8 seconds (frequency ~ 0.125 Hz). This, in-turn will reduce the overall current consumption, as the execution of the ISR will take 2-10 µs, depending on the number of instructions which need to be executed in this ISR.

**NOTE**

The time interval given above is as per the present RTC driver.

**Syntax**

```
void interrupt VectorNumber_Vrtc RTC_ISR(void)
```

**Parameters**

None

**Return Value**

None

## 3.1.4 IRQ_ISR

This ISR is called each time a falling edge is detected at the IRQ/Ignition pin. It is assumed that the falling edge will be generated, each time ignition is ON/OFF, which is done through a signal conditioning on the Ignition pin.

**Syntax**

```
void interrupt VectorNumber_Vrtc IRQ_ISR(void)
```

**Parameters**

None

**Return Value**

None

### 3.1.5 UpdateAndDisplayTime

This API should be called each time there is an interrupt from the RTC or from the IRQ to update the clock structures to the latest values.

**Syntax**

```
void UpdateAndDisplayTime(void)
```

**Parameters**

None

**Return Value**

None

### 3.1.6 ClockCorrection

This API should be called, each time there is a correction required. This API should be changed as per the crystal characteristics. For example, when the crystal characteristics are such that the clock is gaining 1 second every 8 hours under the ambient temperature conditions, this API should be called when the hours are equal to 8, 16, and 24, but only once and consequently, 1 should be subtracted from the seconds variable in order to provide a correct and accurate clock.

**Syntax**

```
void ClockCorrection(void)
```

**Parameters**

None

**Return Value**

None

## 3.2 RTC.h

This header file contains the clock structure and the function prototypes. This file also contains the macros, which are used to initialize the clock structures, which are described in detail below:

### 3.2.1 RTC_HOURS

This macro is used to initialize the hours value in the clock structure and can be changed as per the requirement of the application.

### 3.2.2 RTC_MINUTES

This macro is used to initialize the minutes value in the clock structure and can be changed as per the requirement of the application.

### 3.2.3 RTC_SECONDS

This macro is used to initialize the seconds value in the clock structure and can be changed as per the requirement of the application.

### 3.2.4 RTCMOD_NORMAL_VALUE

This macro is used in Normal mode to load the RTCMOD register.

### 3.2.5 RTCMOD_LP_VALUE

This macro is used in Stop3 mode to load the RTCMOD register.

## 3.3 RTC_main.c

This file contains the main function, which acts as an application for the present RTC driver. This can be removed, when the driver is integrated into the application.

### 3.3.1 System_Init

This API initializes the overall system. It disables the watchdog and enables the Stop3 mode only when the ignition is stopped and is captured through the IRQ, and Low Voltage Detect (LVD) reset is disabled.

**Syntax**

```
void System_Init(void)
```

**Parameters**

None

**Return Value**

None

### 3.3.2 Main

This API is the entry point.

**Syntax**

```
void main(void)
```

**Parameters**

None

**Return Value**

None

# 4 Integration of RTC driver

In order to integrate the RTC driver in the existing code, add the following files:

- RTC.c
- RTC.h

From the main of the existing application, call RTC_Init() and IRQ_Init(). The present RTC driver presumes the external crystal oscillator used is 32.768 kHz and configures the prescaler value to $2^{10}$ (1024), thus, making the RTC clock to 32 Hz (32.768 kHz/ 1024). If an application uses a crystal with a different frequency, then the prescaler needs to be reconfigured and the RTC driver needs to be recalibrated. See ClockCorrection for more details.

The application should always wait for the oscillator clock to stabilize and then initialize other drivers.

In the main loop, the application should always check the MCU operating mode, so that when the MCU wakes up due to ignition, the application should correct the clock, by loading the correct value in milliseconds.

# 5 Test results

RTC soaking results under various temperature conditions, with and without corrections are displayed in Table 1. All the readings are noted, after the temperature was stabilized. The results may show lot of variation in time due to PCB characteristics, crystal characteristics and the components used on the board.

**Table 1. RTC driver test results under various temperature conditions**

| S. No. | Parameters | Results without correction | Results with 1 second correction / 8 hours |
|---|---|---|---|
| 1 | Temperature | Ambient | Ambient |
| | Number of hours | 185 h | 185 h |
| | Time gain | 24 s | 1 s |
| 2 | Temperature | 40.1°C | 40.1°C |
| | Number of hours | 65 h | 65 h |
| | Time gain | 10 s | 2 s |

*Table continues on the next page...*

**Table 1. RTC driver test results under various temperature conditions (continued)**

| S. No. | Parameters | Results without correction | Results with 1 second correction / 8 hours |
|---|---|---|---|
| 3 | Temperature | 60.9°C | 60.9°C |
| | Number of hours | 70 h | 70 h |
| | Time gain | 12 s | 3 s |
| 4 | Temperature | 80.5°C | 80.5°C |
| | Number of hours | 24 h | 24 h |
| | Time gain | 8 s | 5 s |

# 6 Conclusion

The RTC driver can be added into the application, by easily adding 2 files (described in Integration of RTC driver), but care must be taken to calibrate the driver as per the crystal characteristics in order to get accurate clock in the vehicle. The application can also add compensation algorithm based on the temperature characteristics of the crystal for better accuracy of the clock with temperature.

The accuracy of the software RTC is affected by the frequency tolerance of the microcontroller clock source. If the clock source is an external crystal (for instance), a high ppm frequency tolerance would be preferred.

# 7 References

1. MC9S08LG32RM available at http://www.freescale.com
2. DEMO9S08LG32 Schematic available at http://www.freescale.com

## *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





Document Number: AN4478
Rev. 0, 03/2012