**Software Real Time Clock Implementation on MC9S08LG32**

By: Nitin Gupta

Senior Application Engineer, India Solution Center

# 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, I2C, SCI, ADC and Real Time Counters[[1]](#Ref1). This document 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](#Figure1). System/Computer 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 board.

In this Application Note, the 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.

NOTE: ***This note does not cover how to configure the LCD. For details on this please refer to AN3823.***

Contents

[1. Introduction 1](#_Toc316548640)

[2. Real Time Clock (RTC) 1](#_Toc316548641)

[3. RTC driver framework 3](#_Toc316548642)

[3.1 RTC.c 3](#_Toc316548643)

[3.1.1 RTC\_Init 3](#_Toc316548644)

[3.1.2 IRQ\_Init 3](#_Toc316548645)

[3.1.3 RTC\_ISR 3](#_Toc316548646)

[3.1.4 IRQ\_ISR 4](#_Toc316548647)

[3.1.5 UpdateAndDisplayTime 4](#_Toc316548648)

[3.1.6 ClockCorrection 4](#_Toc316548649)

[3.2 RTC.h 4](#_Toc316548650)

[2.2.1 RTC\_HOURS 4](#_Toc316548651)

[2.2.2 RTC\_MINUTES 4](#_Toc316548652)

[2.2.3 RTC\_SECONDS 4](#_Toc316548653)

[2.2.4 RTCMOD\_NORMAL\_VALUE 4](#_Toc316548654)

[2.2.5 RTCMOD\_LP\_VALUE 4](#_Toc316548655)

[3.3 RTC\_main.c 4](#_Toc316548656)

[3.3.1 System\_Init 5](#_Toc316548657)

[3.3.2 main 5](#_Toc316548658)

[4. Integration of RTC Driver 5](#_Toc316548659)

[5. Test Results 5](#_Toc316548660)

[6. Conclusion 5](#_Toc316548661)

[7. References 5](#_Toc316548662)

# WT3253Real Time Clock (RTC)

**Figure 1. MC9S08LG32 Demo Board Setup[[2]](#Ref2)**

The RTC (or sometimes referred to as time of day) implementation can be either a hardware or software implementation. 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 (these are sometime connected via an I2C or a SPI). On the other hand, some hardware RTC implementations are provided by an on-chip peripheral in 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 counter that gives 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)[[1]](#Ref1), 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 reach this goal.

# RTC driver framework

**Figure 2. MC9S08LG32 RTC driver**

**Figure 2: MC9S08LG32 RTC driver**

The RTC driver is provided as “C” code files. The driver consists of three files, namely:

1. RTC.c
2. RTC.h
3. RTC\_main.c

MC9S08LG32 RTC driver project is illustrated in [Figure 2](#Figure2).

The detail of each file is as below.

## 3.1 RTC.c

RTC.c consists of API’s which enables the software real time clock on the MC9S08LG32 series. The description of each API is as follows:

### RTC\_Init

This API initializes the RTC and the system clock. It uses a 32.768 KHz external crystal, having 30 ppm tolerance. During the initialization of the clock, operating modes have been taken into consideration. As per the application operating mode can be either Normal mode or Stop3[[1]](#Ref1) mode. In Normal mode, RTC interrupt will occur every 1 second and in Stop3 Mode, RTC interrupt will occur after every 8 seconds.

**Syntax**

*void RTC\_Init(void)*

**Parameters**

*None*

**Return Value**

*None*

### IRQ\_Init

This API initializes the IRQ, 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*

### RTC\_ISR

This ISR is called each time RTC interrupt occurs. In this, clock-time is updated as per the mode and interrupt flag is cleared. If the mode is normal mode, the time interval of interrupt 1 second (frequency - 1Hz). In low power mode (Stop3), this act as a wake-up source for MCU, and the time interval of 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 µs - 10 µs, depending on the number of instructions which needs to be executed in this ISR.

*Note: The above time is as per the present RTC driver.*

**Syntax**

*void interrupt VectorNumber\_Vrtc RTC\_ISR(void)*

**Parameters**

*None*

**Return Value**

*None*

### 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*

### 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*

### ClockCorrection

This API should be called, each time there is a correction required. This API should be changed as per the crystal characteristics. e.g. the crystal characteristics are such that our clock is gaining 1 sec every 8 hours under the ambient temperature conditions, then this API should be called when the hours are equal to 8, 16 and 24, but only once and one 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 initializes the clock structures, which are described in detail below:

### RTC\_HOURS

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

### 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

### 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.

### RTCMOD\_NORMAL\_VALUE

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

### 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 (triggered, 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*

# Integration of RTC Driver

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

1. RTC.c
2. RTC.h

From the main of the existing application, call RTC\_Init() and IRQ\_Init(). Present RTC driver presumes the external crystal used is 32.768 kHz and configures the prescaler value to 210 (1024), thus, making the RTC clock to 32 Hz (32.768 kHz/ 1024). If, Application uses a crystal with different frequency, then the prescaler needs to be re-configured and the RTC driver needs to be re-calibrated. Refer to [section 3.1.6 ClockCorrection](#_ClockCorrection) for more details

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Parameters** | **Result without Correction** | **Results with 1 second correction / 8 Hours** |
| 1 | Temperature | Ambient | Ambient |
| Number of hours | 185 hours | 185 hours |
| Time gain | 24 seconds | 1 second |
| 2 | Temperature | 40.1°C | 40.1°C |
| Number of hours | 65 Hours | 65 Hours |
| Time gain | 10 seconds | 2 seconds |
| 3 | Temperature | 60.9°C | 60.9°C |
| Number of hours | 70 hours | 70 hours |
| Time gain | 12 seconds | 3 seconds |
| 4 | Temperature | 80.5°C | 80.5°C |
| Number of hours | 24 hours | 24 hours |
| Time gain | 8 seconds | 5 seconds |

Table 5-1: RTC driver test results under various   
 Temperature Conditions

# Test Results

RTC soaking results under various temperature conditions, with and without corrections are displayed in [Table 5-1](#Table5_1). All the readings are noted, after the temperature was stabilized. Results may show lot of variation in time due to PCB characteristics, crystal characteristics and the components used on the board.

**Table 5-1: Test Results of RTC soaking on MC9S08LG32 Demo Board using 32.768 KHz (30ppm)**

# Conclusion

RTC Driver can be added into the application, by easily adding 2 files (described in [section 4](#_Integration_of_RTC)), but care must be taken to calibrate the driver as per the crystal characteristics in order to get accurate clock in the vehicle. 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.

# References

1. MC9S08LG32 RM: <http://cache.freescale.com/files/microcontrollers/doc/ref_manual/MC9S08LG32RM.pdf?fr=g>
2. DEMO9S08LG32 Schematics: <http://cache.freescale.com/files/DEMO9S08LG32SCH.pdf?fsrch=1&sr=3>