Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 002-05285

Spec Title: AN205285 - FM3, MB9AA30 SERIES, MB9AFA32N RTC SOLUTION DEVELOPMENT KIT

Replaced by: NONE

AN205285

FM3, MB9AA30 Series, MB9AFA32N RTC Solution Development Kit

The MB9AA30N series are highly integrated 32-bit Arm Cotex-M3 microcontrollers that dedicated for embedded controllers with low power consumption mode and competitive cost. As a member of MB9AA30N series MCU, MB9AA32N integrated with LCD and RTC functions can be applied in the power meter area.

Contents

1 Introduction

1.1 About MB9AFA32N

The MB9AA30N series are highly integrated 32-bit Arm Cotex-M3 microcontrollers that dedicated for embedded controllers with low power consumption mode and competitive cost. As a member of MB9AA30N series MCU, MB9AA32N integrated with LCD and RTC functions can be applied in the power meter area.

1.2 About Real Time Clock (RTC)

Integrated internally in the MB9AA30N, the RTC count block can count years, months, dates, hours, minutes, seconds, and days of the week from 01 to 99 years. Alarm and timer settings are possible as well. An alarm can be set to a specific year, month, date, hour, and minute. It can also be set to a specific year, month, date, hours, or minutes independently. A timer can be set to a period up to one day. It can be set to a desired period (with the hours, minutes, and seconds specified) or in desired intervals (with hours, minutes, and seconds specified).

An overview of the RTC count block is shown below.

- Date and time (year/month/date/hour/minute/second/day of the week) settings
- Counts dates and time (years, months, days, hours, minutes, seconds, and days of the week)
- Leap year compliant

- Alarm settings with a specific date and time (year/month/date/hour/minute)
- A specific year, month, date, hour, or minute can be set individually as well
- A timer can be set to a period of up to one day. It can be set to a desired period (with the hours, minutes, and seconds specified) or in desired intervals (with hours, minutes, and seconds specified)
- It is possible to rewrite the time by resetting the watch count of the RTC count block to make time-signal-based settings
- It is possible to rewrite the time while the watch count of the RTC count block continues for time zone changes.
- The following interrupts can be output:
  - ☐ Alarm (with an interrupt generated at a set date and time)
  - ☐ Every hour
  - ☐ Every minute
  - ☐ Every second
  - ☐ Every 0.5 seconds
  - ☐ Timer
  - ☐ Time rewrite error
  - ☐ Timer counter read completion
- Pulse output in 0.5-second or 1 second intervals.

1.3 About Frequency Correction Block

As part of RTC module, Frequency correction block can calibrate the frequency of RTC source clock. If 32768Hz oscillator is used as source clock of RTC, the calibration cycle can be set between 1 to 64 seconds, and 1 to1024 pulses can be masked in a calibration cycle.

1.4 About Document

This application note gives an example of how to implement the clock/calendar functionality using MB9AFA32N in low-power and standard applications. The firmware also provides a low-power mode demonstration and calibration process monitoring which implements a basic RTC calibration routine to compensate oscillator frequency according to temperature variations.

The MB9AFA32N RTC Solution Minimum system (the schematic is attached at the annex) supports the firmware example introduced in this application notes.

With debug and programming adaptor board (the schematic is attached), user can debug this example with IAR Embedded Workbench V6.21 or later version.

2 RTC Clock/Calendar Implementation Process

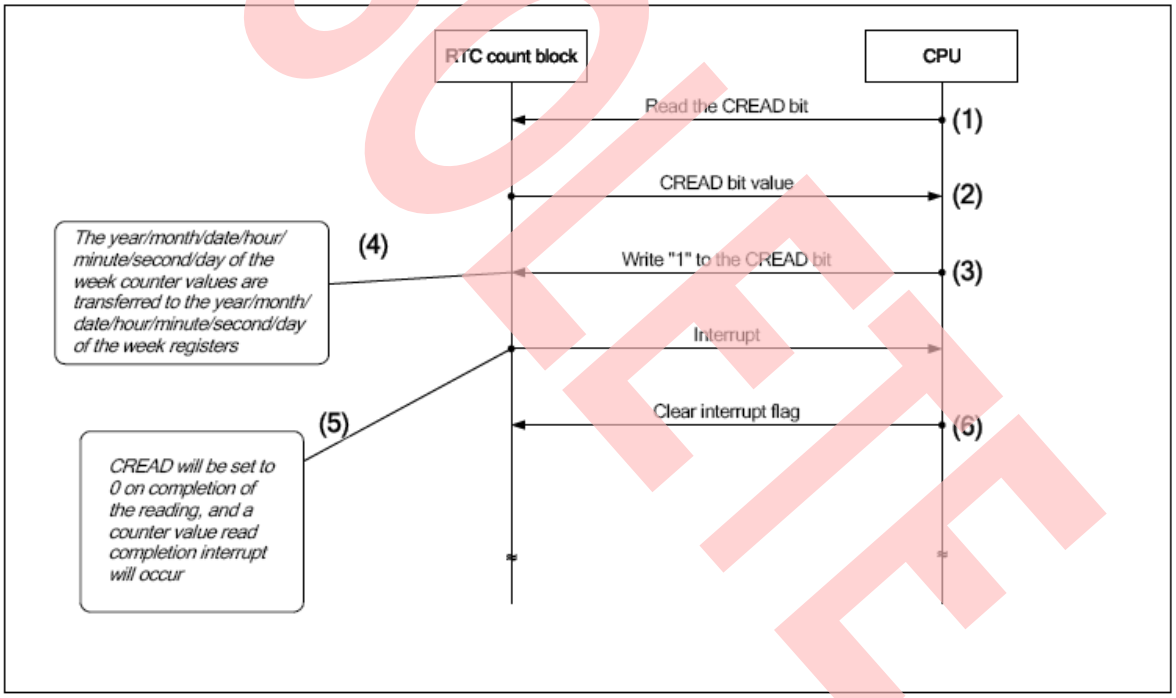
This example (RTC Development Kit) integrating RTC and low power consumption driver can demonstrates most RTC functions. Following section describes the process to implement each function.

2.1 Read Time/Date

User can use following steps to read time/date of the calendar:

1. Read the CREADbit (See WTCR2 register).
2. Check that the CREAD value becomes "0".
3. Write the following value: CREAD = "1".
4. The year/month/date/hour/minute/second/day of the week counter values will be transferred to the year, month, date, hour, minute, second, and day of the week registers (WTYR, WTMOR, WTDR, WTHR, WTMIR, WTSR , WTDW).
5. A year/month/date/hour/minute/second/day of the week counter value read completion interrupt will occur the following condition will be set: CREAD = "0".
6. Clear the year/month/date/hour/minute/second/day of the week counter value read completion interrupt

Figure 1. Read Time/Date Flow Chart



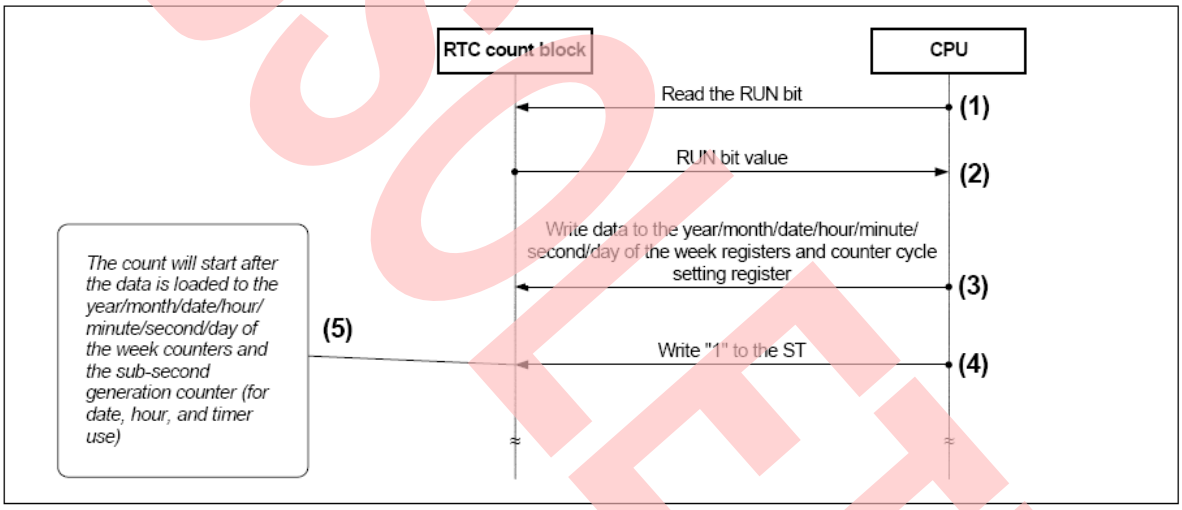
2.2 Write Time/Date

2.2.1 Initialize Time/Date

User can use following steps to initialize time/date of the calendar:

1. Read the RUN bit (See WTCR1 register).
2. The following initial time settings are possible in the following flow while the RTC count block is not in operation under the following condition: RUN = "0".
3. Write the desired time to the year/month/date/hour/minute/second/day of the week registers (WTYR, WTMOR, WTDR, WTHR, WTMIR, WTSR, and WTDW). Write a desired 0.5-second count value to the counter cycle setting register (WTBR).
4. Write the following value: ST = "1".
5. The year/month/date/hour/minute/second/day of the week register value and the value in the counter cycle setting register (WTBR) are loaded to the year/month/date/hour/minute/second/day of the week counters and the sub-second generation counters (for date, time, and timer use), and start counting.

Figure 2. Write Time/Date Flow

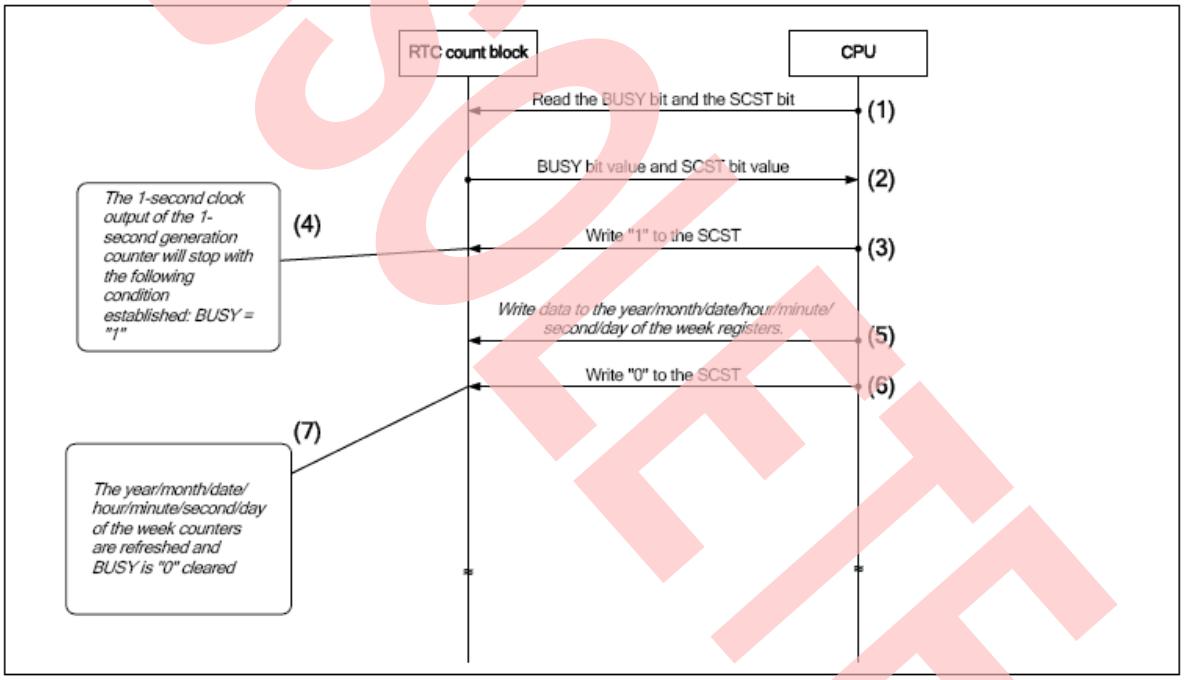


2.2.2 Rewrite Time/Date (with the time count continued)

User can use following steps to rewrite time/date of the calendar with the time count continued:

1. Read the BUSY bit and the SCST bit (See WTCR1 register).
2. Wait until the following condition is established: BUSY = "0" and SCST="0".
3. Write the following value: SCST = "1".
4. BUSY will be set to "1". The 1-second clock output of the 1-second generation counter will come to a stop.
5. Write the desired year, month, date, hour, minute, second, day of the week value to the year/month/date/hour/minute/second/day of the week registers (WTYR, WTMOR, WTDR, WTHR, WTMIR, WTSR, WTDW) while the following condition is maintained: SCST = "1".
6. Write the following value: SCST = "0".
7. Only the refreshed set values in the year/month/date/hour/minute/second/day of the week registers will be transferred to the year/month/date/hour/minute/second/day of the week counters and BUSY will be "0" cleared.

Figure 3. Rewrite Time/Date with Time Continue Flow

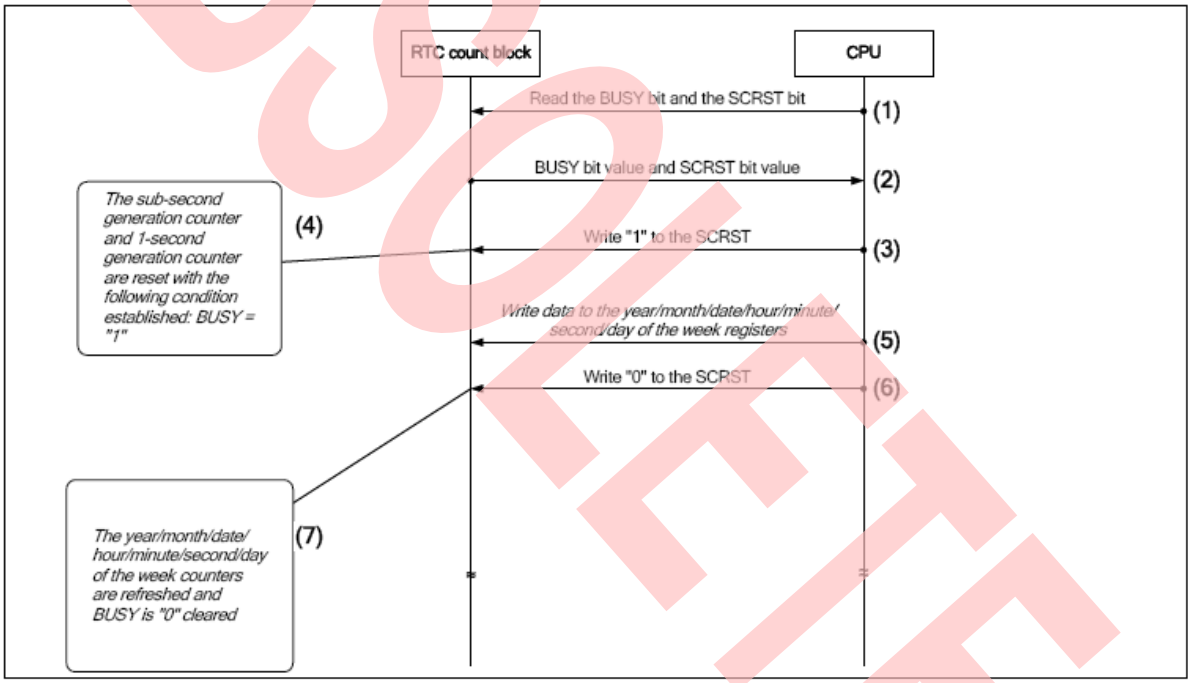


2.2.3 Rewrite Time/Date (with the time count reset)

User can use following steps to rewrite time/date of the calendar with the time count reset:

1. Read the busy bit and the SCST bit (See WTCR1 register)
2. Wait until the following condition is established: BUSY = "0" and SCST="0".
3. Write the following value: SCRST = "1".
4. BUSY will be set to "1". The sub-second generation counter and 1-second generation counter will be reset.
5. Write the desired new year, month, date, hour, minute, second, day of the week value to the year/month/date/hour/minute/second/day of the week registers (WTYR, WTMOR, WTDR, WTHR,WTMIR, WTSR, WTDW) while the following condition is maintained: SCRST = "1".
6. Write the following value: SCRST = "0".
7. Only the refreshed year/month/date/hour/minute/second/day of the week register values will be transferred to the year/month/date/hour/minute/second/day of the week counters and BUSY will be "0" cleared.

Figure 4. Rewrite Time/Date with Time Reset Flow

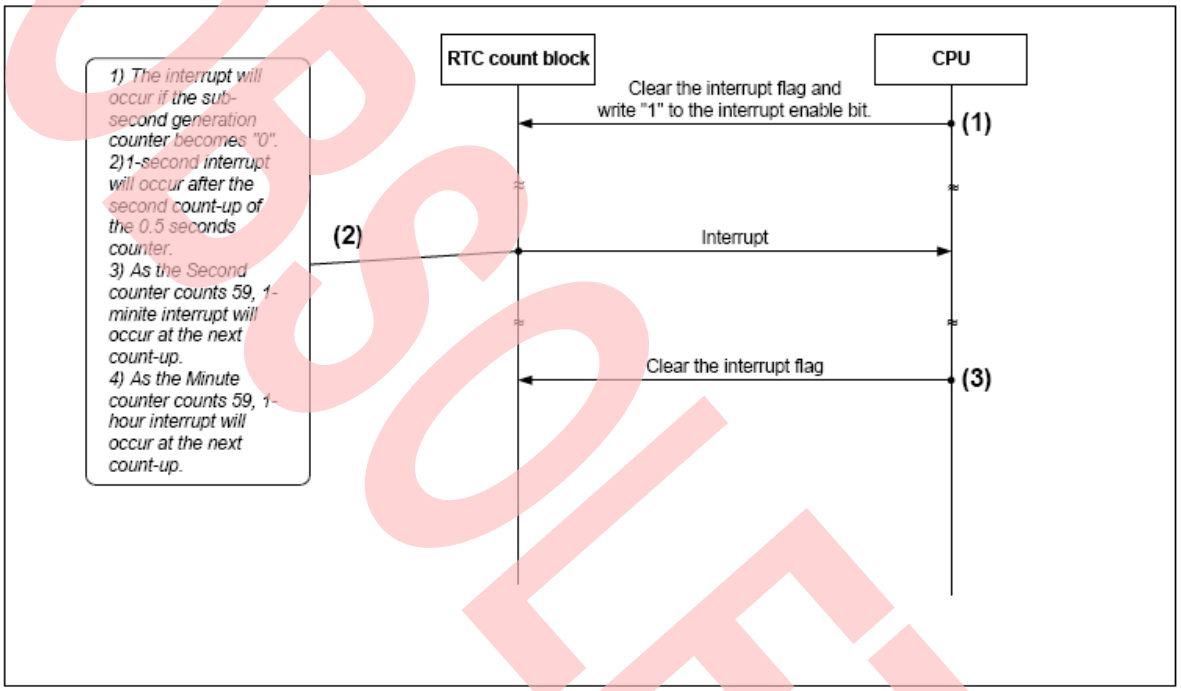


2.3 Set Every 0.5 Second/Second/Minute/Hour Interrupt

User can use following steps to set every 0.5 second/second/minute/hour interrupt:

1. Write INTSSI/ INTSI/ INTMI/ INTHI="0" to clear the interrupt flag bit. Write "1" to the interrupt enable bit of INSSIE, INTSIE, INTMIE, or INHIE to be used to enable the interrupt.
2. If either of 0.5 seconds/ 1-second/1-minute/1-hour interrupt occurs, an interrupt will be generated.
3. Write INTSSI/ INTSI/ INTMI/ INTHI="0" to clear the interrupt flag bit

Figure 5. Every 0.5 Second/Second/Minute/Hour Interrupt Flow

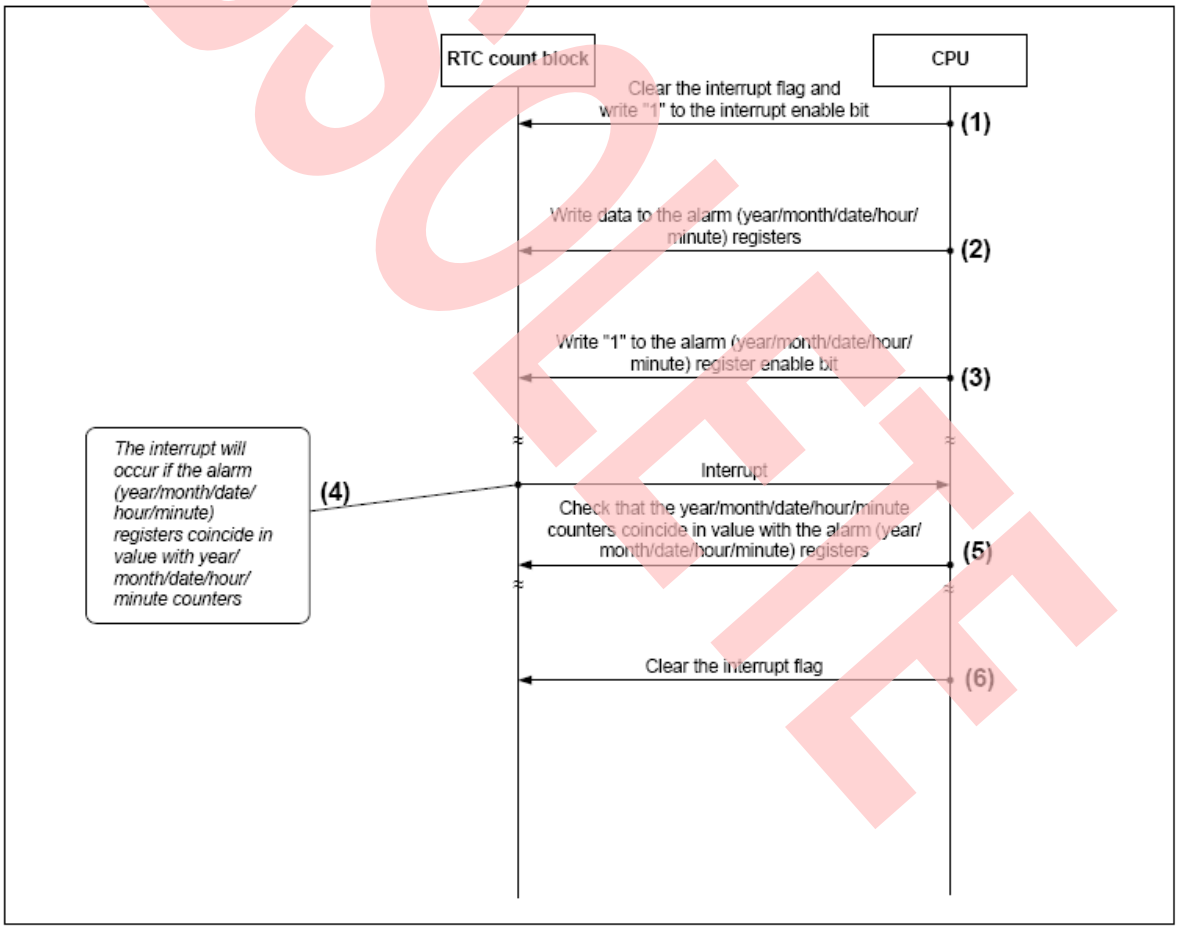


2.4 Set Alarm

User can use following steps to set the alarm:

1. Clear the alarm interrupt flag bit with the following condition written: INTALI = "0".
2. The alarm interrupt is enabled with the following condition written: INTALIE = "1".
3. Write the desired time value to the alarm (year/month/date/hour/minute) registers so that the alarm interrupt will occur at the desired time.
4. Write "1" to the alarm (year/month/date/hour/minute) register enable bit. The RTC count block interrupt request will be generated when the alarm (year/month/date/hour/minute) register values coincide with the year/month/date/hour/minute counter values.
5. Read the time by following the example of the time read setting procedures, and check that the year/month/date/hour/minute counter values coincides with the alarm (year/month/date/hour/minute) register values.
6. Clear the alarm interrupt flag bit with the following condition written: INTALI = "0".

Figure 6. Set Alarm Flow



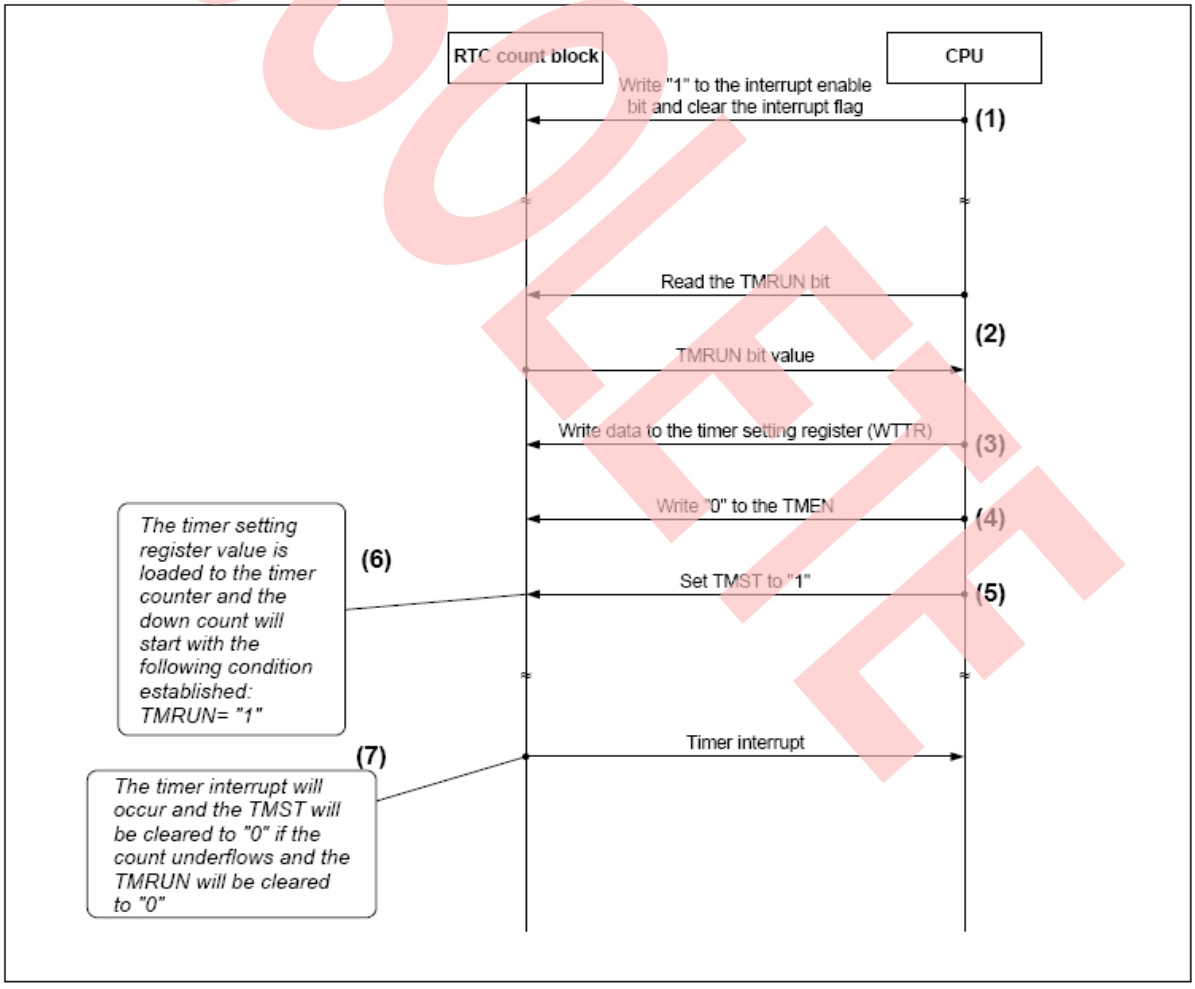
2.5 Set Timer

2.5.1 Set Timer (with (hours, minutes, and seconds) elapsed)

User can use following steps to set the timer with hours, minutes and seconds elapsed:

1. Clear the timer interrupt flag bit with the following condition written: INTTMI = "0". The timer interrupt is enabled with the following condition written: INTTMIE = "1".
2. Read the timer counter operation bit (TMRUN) and check whether the value is "0" (not in operation).
3. Write the desired timer set value to the timer setting register (WTTR).
4. Write "0" to the timer counter control bit (TMEN).
5. Write "1" to the timer counter start bit (TMST).
6. The set value in the timer setting register value will be transferred to the timer counter with an elapse of 7 MCLK cycles, and the down count will start.
7. If the down count underflows, an RTC count block interrupt request will be generated, and TMST will be "0" cleared. Then the TMRUN will be "0" cleared after an elapse of 12 MCLK cycles.

Figure 7. Set Timer with (hours, minutes, and seconds) elapsed Flow

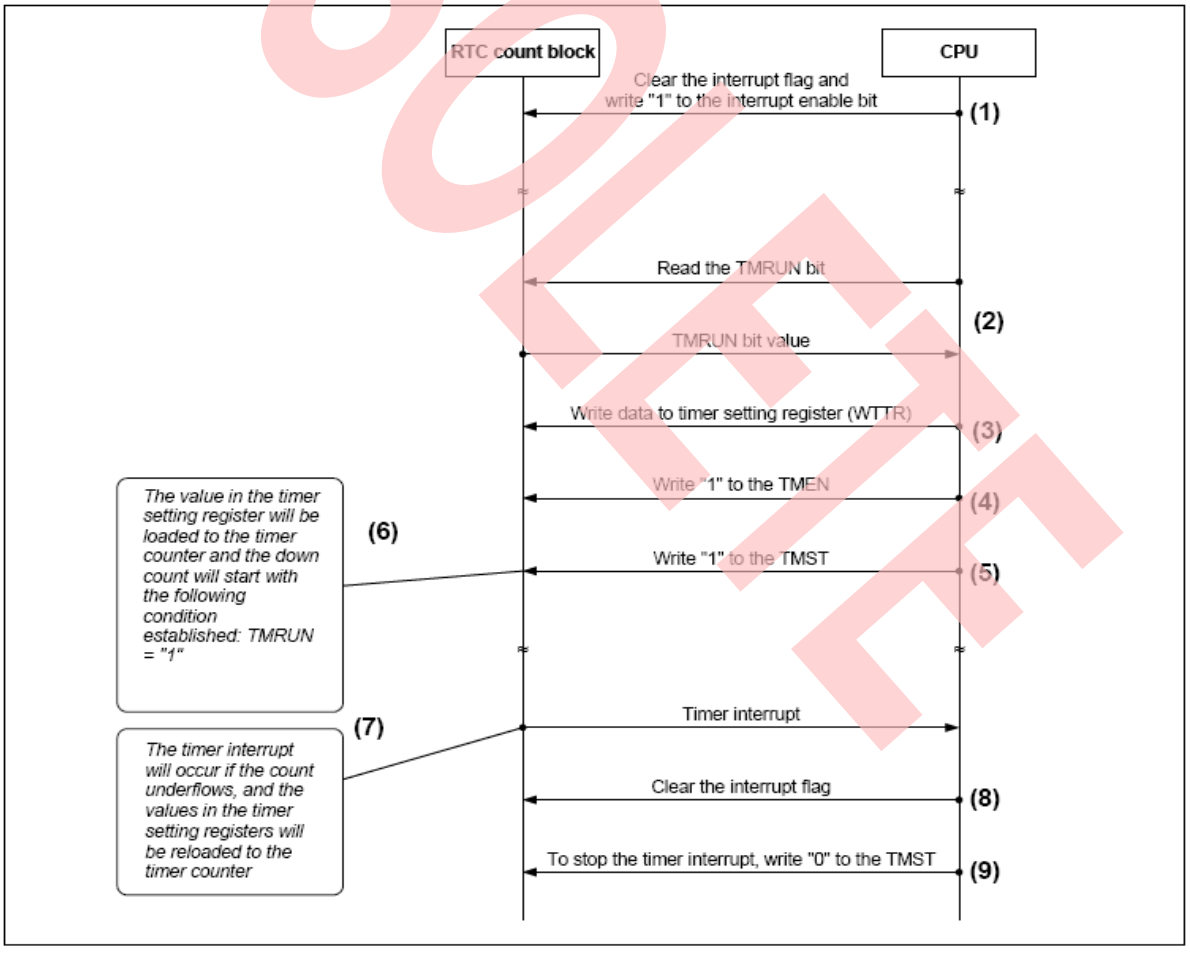


2.5.2 Set Timer (in interval of (hours, minutes, and seconds))

User can use following steps to set the timer in interval of hours, minutes and seconds elapsed:

1. Clear the timer interrupt flag bit with the following condition written: INTTMI = "0". The timer interrupt is enabled with the following condition written: INTTMIE = "1".
2. Read the timer counter operation bit (TMRUN) and check whether the value is "0" (not in operation).
3. Write the desired timer set value to the timer setting register (WTTR).
4. Write "1" to the timer counter control bit (TMEN).
5. Write "1" to the timer counter start bit (TMST).
6. The set value in the timer setting register value will be transferred to the timer counter with an elapse of 7 MCLK cycles, and the down count will start.
7. The RTC count block interrupt request will be generated on completion of the count, and the timer setting register value will be reloaded to the timer counter to continue operation.
8. Clear the timer interrupt flag bit with the following condition written: INTTMI = "0".
9. To stop the timer interrupt, write "0" to the TMST.

Figure 8. Set Timer in interval of (hours, minutes, and seconds) Flow



2.6 Calibration

2.6.1 Calibration Overview

RTC precision is a requirement in most embedded applications. The external crystal oscillator used to provide the source clock to RTC is subject to frequency variations due to external temperature variation. It is therefore necessary to compensate the oscillator frequency according to temperature variations.

Following figure is a typical temperature-frequency curve of a 32768Hz oscillator. The relationship of frequency deviation and temperature can be described by following formula:

$$\Delta f / f_0 = \beta(T - T_i)^2 + \Delta f_0$$

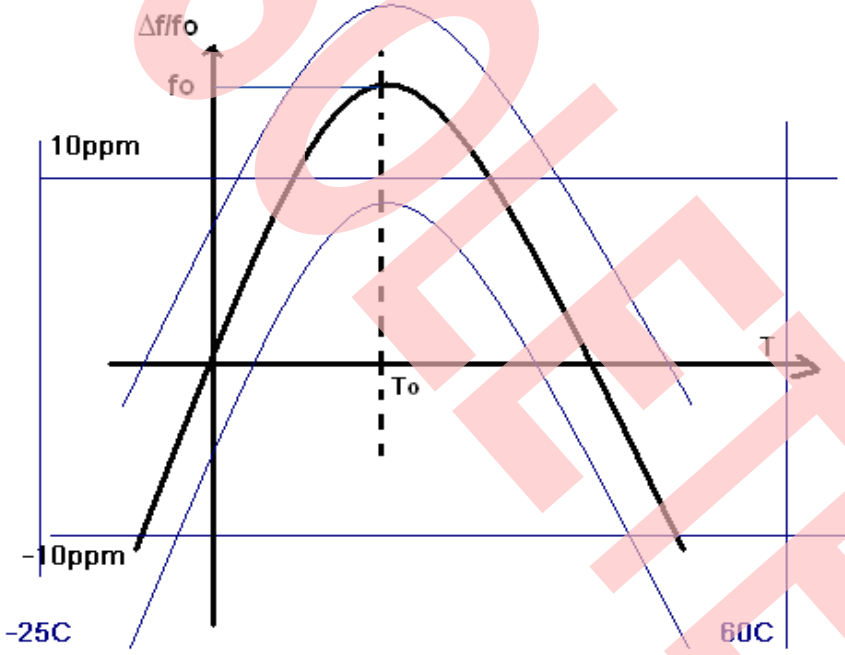
β : Parabolic coefficient

Ti: Turnover temperature

Δf_0 : Frequency tolerance

f_0 : Nominal frequency

Figure 9. Frequency-Temperature Curve



Usually the above 4 parameter can be provided in the oscillator manufacture's SPEC, and with this data, the frequency deviation can be gotten in each temperature point, so the temperature compensation can be implemented based on it.

2.6.2 Calibration Process

There are two steps to make the calibration:

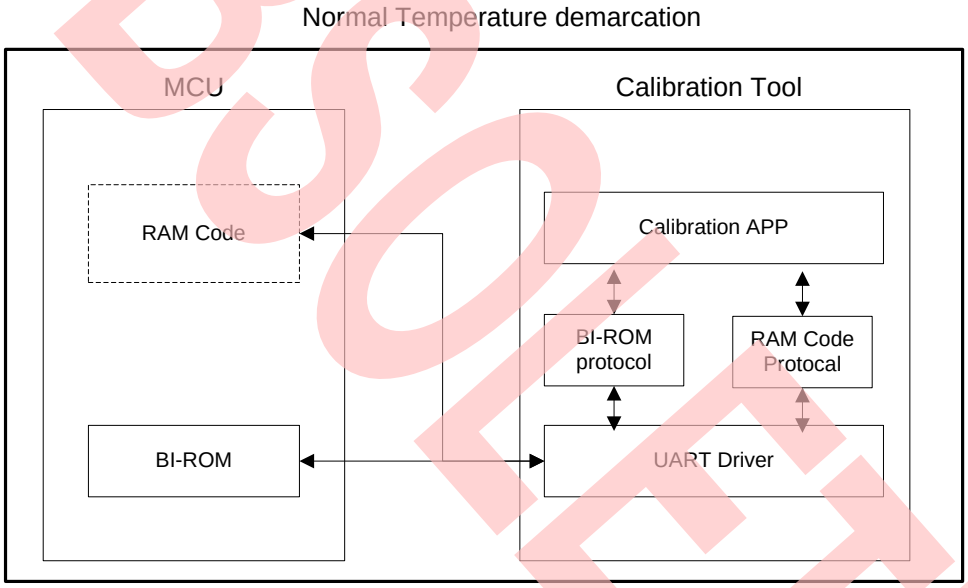
1. Normal temperature demarcation
2. Temperature compensation

2.6.2.1 Normal temperature demarcation

Normal temperature demarcation is used to get the Frequency tolerance (f_0). Usually f_0 is between -20ppm to 20ppm, but it is also affected by Load Capacitance[*1] of oscillator.

In the normal temperature demarcation process, the calibration tool[*2] will first download a RAM code to MCU by BI-ROM protocol. RAM code can produce a 1Hz signal from SUBOUT pin, the tool calculates the RTC clock frequency based on this signal and sends the normal temperature demarcation value to MCU via UART after calculation. Finally, the RAM code stores the calibration value received in a Flash area[*3].

Figure 10. Normal Temperature Demarcation Block Diagram



Notes:

1. Load Capacitance value is also specified in the oscillator's SPEC.
2. For the calibration tool usage, refer to "MB9AFA32N RTC Calibration Tool Simple User Manual".
3. In this solution, the Normal Temperature Demarcation value is stored at the Flash area (0x0001FFFC-0x0001FFFF)

2.6.2.2 Temperature Compensation

The MB9AFA32N RTC comes with a Frequency Correction Block that gives the user software control over the calibration process. The Frequency Collection Block removes 0 to 1023 cycles every 1 to 64 seconds. The number of pulses are blanked depends on the value loaded into the ten least significant bits of the Frequency Correction Value Setting Register (WTCAL), the calibration cycle in which pluses is masked depends on value loaded into the six least significant bits of the Frequency Correction Cycle Setting Register (WTCALPRD).

The Frequency Collection Block can only subtract clock cycles, which means that only higher frequency can be compensated whereas lower frequencies cannot.

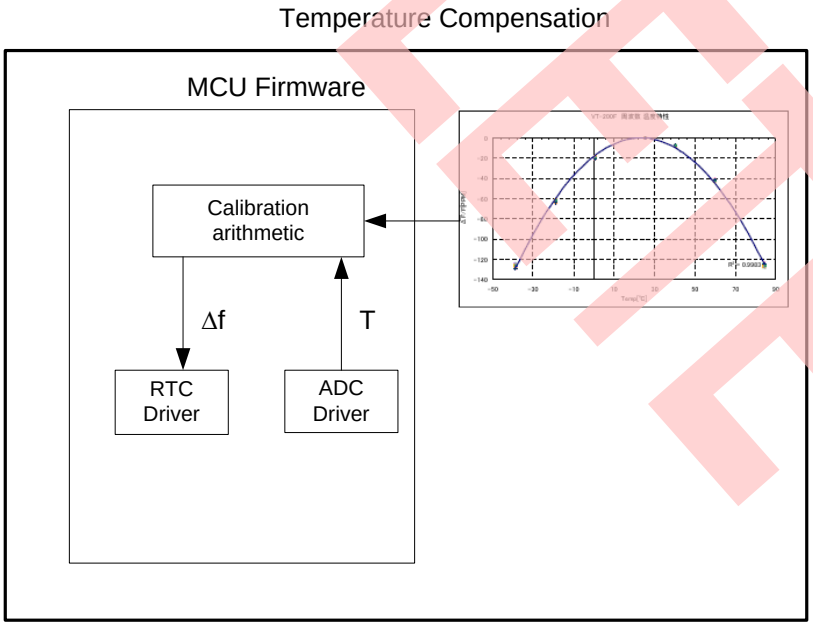
Assume in the 20s*1 calibration cycle If the prescaler of RTC clock is set to 32768*2, the frequency range that can be calibrated is [32768.00Hz, 32819.15Hz]*3. The workaround way is that the prescaler can be set to 32760*4 and the frequency range that can be calibrated is [32760.00Hz, 32811.13Hz]*5.

Notes:

1. To mask one pulse in 20s cycle means reducing the frequency by 1.526ppm (1/(32768*20))
2. Set Counter Cycle Setting Register (WTBR) to 8191 ((32768/4) -1)
3. 1.526ppm * 1023 = 1561.10ppm, (1Hz/32768Hz)*10^6 = 30.52ppm/Hz,
4. ΔF = 1561.10/30.52 =51.15Hz
5. Set Counter Cycle Setting Register (WTBR) to 8189 ((32760/4) -1)
6. 1.526ppm * 1023 = 1561.10ppm, (1Hz/32760Hz)*10^6 = 30.53ppm/Hz,
7. ΔF = 1561.10/30.53 =51.13Hz

In the temperature compensation process, MCU is in normal run mode, the firmware uses ADC to sample temperature via the thermistance, and calculates the frequency deviation by Frequency-Temperature formula, the value in additional of demarcation value are then written into the Frequency Correction Setting Register (WTCAL).

Figure 11. Temperature Compensation Block Diagram



2.7 Low Power Consumption Mode

To reduce the power consumption, the system provides low power consumption mode, which enables the use of the standby mode of SLEEP, TIMER, RTC and STOP modes and the deep standby mode of deep standby RTC and deep standby stop modes.

- Standby modes
  - ☐ Sleep modes
  - ☐ Timer modes
  - ☐ RTC mode
  - ☐ STOP mode
- Deep Standby modes
  - ☐ Deep Standby RTC mode
  - ☐ Deep Standby STOP mode

According to RTC applications, RTC mode and Deep Standby RTC Mode will be introduced here, if you want to learn other standby mode, please refer to "MB9Axxx / MB9Bxxx Series PERIPHERAL MANUAL"

2.7.1 RTC Mode

2.7.1.1 Clock operation state

In RTC mode, the CPU clock supplied to the CPU and AHB bus clock supplied to the on-chip memory and DMA controller are stopped. However, the contents of on-chip memory are retained. Also, the debug function is stopped.

In RTC mode, all APB clocks are stopped, and all resources, excluding the watch counter, RTC, and Low-voltage Detection Circuit, are stopped keeping the last state.

Table 1 .Clock Operation in RTC Mode

<table>
<tr><th></th><th>RTC mode</th><th>STOP mode</th></tr>
<tr><td>High speed CR clock</td><td rowspan="4">Stopped</td><td rowspan="10">Stopped</td></tr>
<tr><td>Main clock</td></tr>
<tr><td>Main PLL clock</td></tr>
<tr><td>Low speed CR clock</td></tr>
<tr><td>Sub clock</td><td>Operating</td></tr>
<tr><td>USB PLL clock</td><td rowspan="6">Stopped</td></tr>
<tr><td>CPU clock</td></tr>
<tr><td>AHB bus clock</td></tr>
<tr><td>APB0 bus clock</td></tr>
<tr><td>APB1 bus clock</td></tr>
<tr><td>APB2 bus clock</td></tr>
</table>

- The process to enter into RTC mode
  1. Write RTCE=1 to the PMD_CTL Register.
  2. Write KEY=0x1ACC, DSTM=0 and STM=0b10 to the STB_CTL Register together.
  3. Set SLEEPDEEP=1 in Cotex-M3 System Control Register (Addr:0xE000ED10)
  4. Execute the WFI or WFE instruction.

- RTC mode Return Cause

The RTC mode can be waken up by following factors:

Table 2. Return Cause of RTC Mode

Return factors by Reset	Return factors by Interrupt
INITX pin input reset Low-voltage detection reset	NMI interrupt External interrupt USB wake up Interrupt Watch counter interrupt RTC interrupt HDMI-CEC/ Remote Control Reception Low voltage detection interrupt

Notes:

1. The On-chip memory will retain after it enters into RTC mode, which means the global variable and peripheral registers will retains.
2. The code will continue to run after "WFI" or "WFE" instruction after it is waken up.
3. The clock mode of MCU will also keep same in this process.

2.7.2 Deep Standby RTC Mode

2.7.2.1 Clock operation state

In deep standby RTC mode, the CPU clock supplied to CPU and AHB bus clock supplied to on-chip memory and DMA controller are stopped. It turns off the CPU, on-chip Flash and on-chip SRAM. The contents of the CPU register and on-chip SRAM are not retained.

All APB bus clocks are stopped, and all resources, excluding RTC, HDMI-CEC/ Remote Control Reception, Low-voltage Detection Circuit, and GPIO, are turned off.

Table 3. Clock Operation in Deep Standby RTC Mode

	Deep standby RTC mode	Deep standby STOP mode
High speed CR clock	Stopped	Stopped
Main clock		
Main PLL clock		
Low speed CR clock		
Sub clock	Operating	
USB PLL clock	Stopped	
CPU clock		
AHB bus clock		
APB0 bus clock		
APB1 bus clock		
APB2 bus clock		

- The process to enter into Deep Standby RTC mode
  1. Write RTCE=1 to the PMD_CTL Register.
  2. Write KEY=0x1ACC, DSTM=1 and STM=0b10 to the STB_CTL Register together.
  3. Set SLEEPDEEP=1 in Cotex-M3 System Control Register (0xE000ED10)
  4. Execute the WFI or WFE instruction.
- Deep Standby RTC mode Return Cause

The Deep Standby RTC mode can be waken up by following factors:

Table 4. Return Cause of Deep Standby RTC Mode

Return factors by Reset	Return factors by Interrupt	Return factors by External pin
INITX pin input reset Low-voltage detection reset	Low-voltage detection interrupt RTC interrupt HDMI-CEC/ Remote Control Reception	WKUP input pin

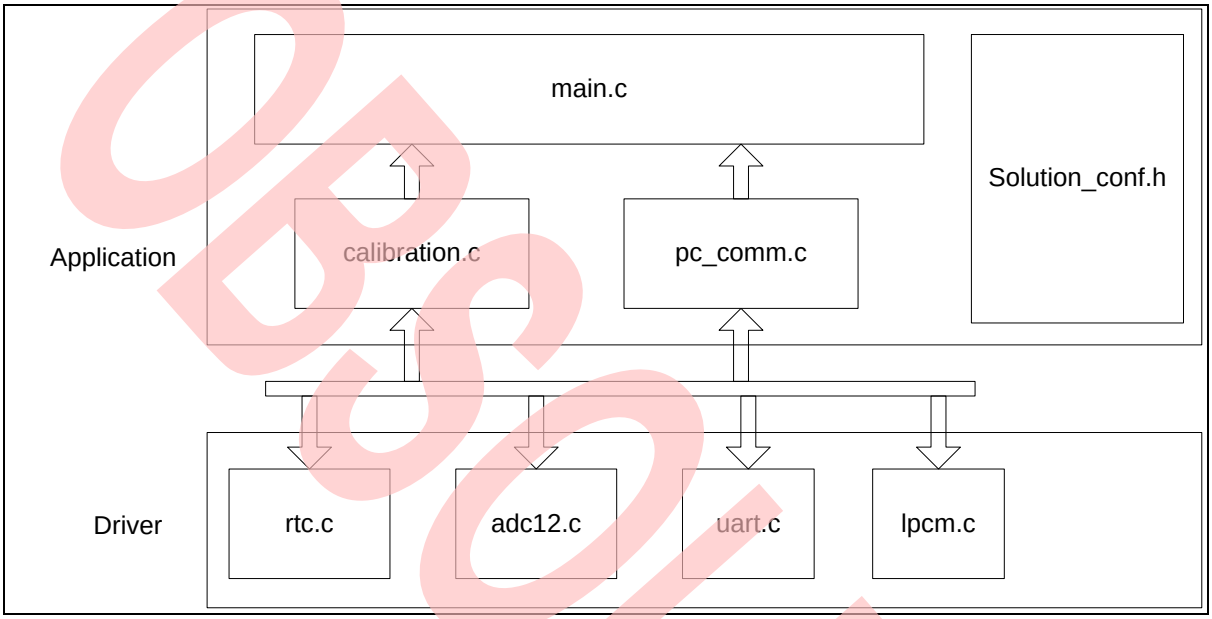
Notes:

1. The On-chip memory will not retain after it enters into Deep Standby RTC mode, which means the global variable and peripheral registers will not retains.
2. The code will continue to run from address stored at reset vector after it is waken up.
3. The clock mode of MCU will turn to the initial mode (high speed CR mode) as system reset.

3 Firmware Description

The RTC Development Kit integrated with RTC and low power consumption mode driver can demonstrate most RTC functions. This example makes use of UART communication with PC, user can test the example in the Hyper Terminal. The project structure of this example is shown as following figure.

Figure 12. RTC Development Kit Block Diagram.



In the calibration.c file, it implements the temperature sampling and calibration arithmetic.

In the pc_comm.c file, it includes all communication functions with PC via UART.

In the solution_conf.h file, it includes all definitions that can be configured by user when he applies this example program.

3.1 Program Flow Chart

3.1.1 Main Function

In the main program, user can input a character via Hyper Terminal to enter a certain function, which can be one of following items:

- Read Time
- Read Date
- Write Time
- Write Date
- Set Alarm
- Calibration Monitor
- Enter RTC Mode
- Enter Deep Standby RTC Mode

In parallel, the calibration process is on going that in a certain interval the system will sample the environment temperature, calculate the calibration value according to Frequency-Temperature formula, then make the calibration for RTC clock. The interval how long to implement this process is specified by the macro "RTC_CLOCK_CALIBRATION_PERIOD", defined at the configuration file "solution_conf.h".

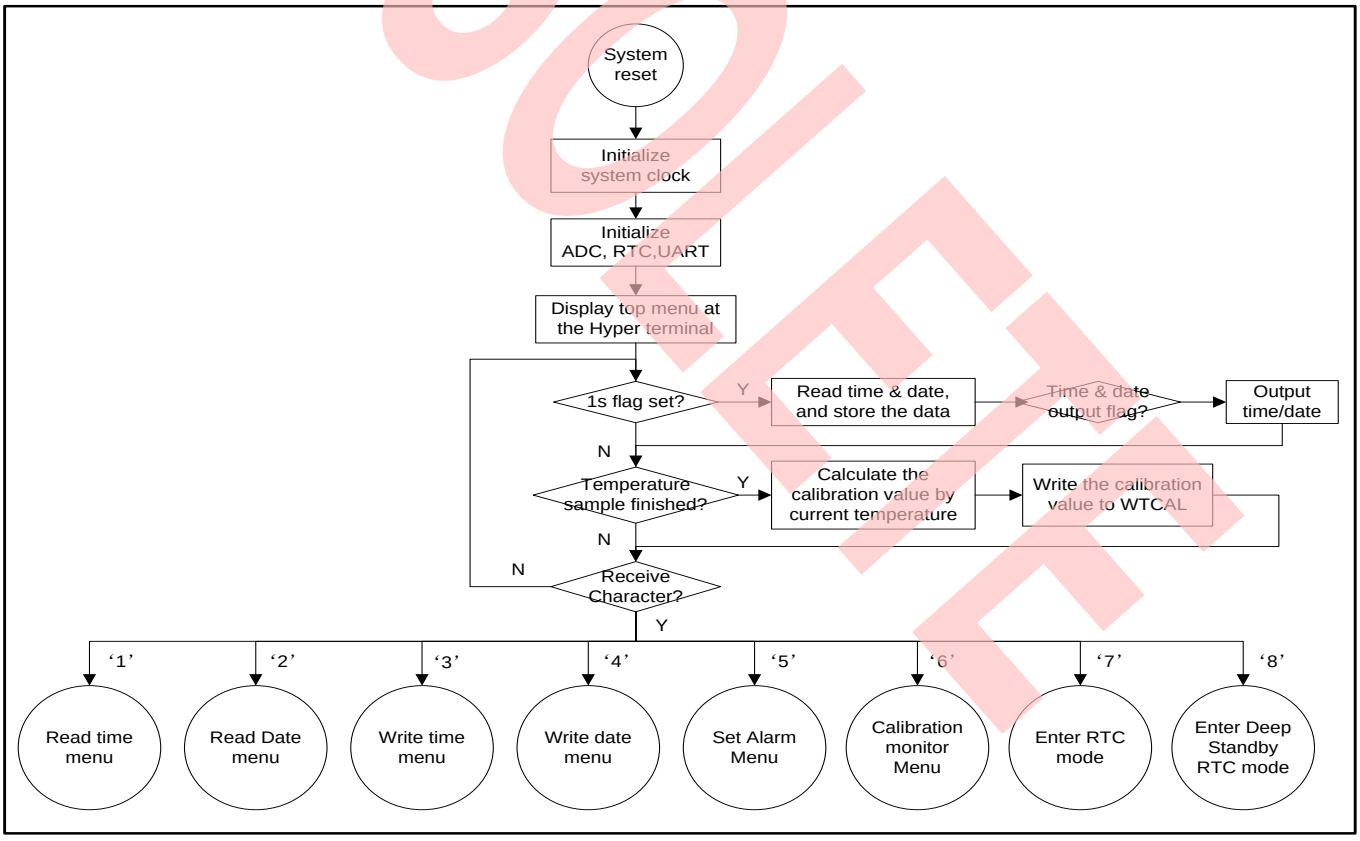


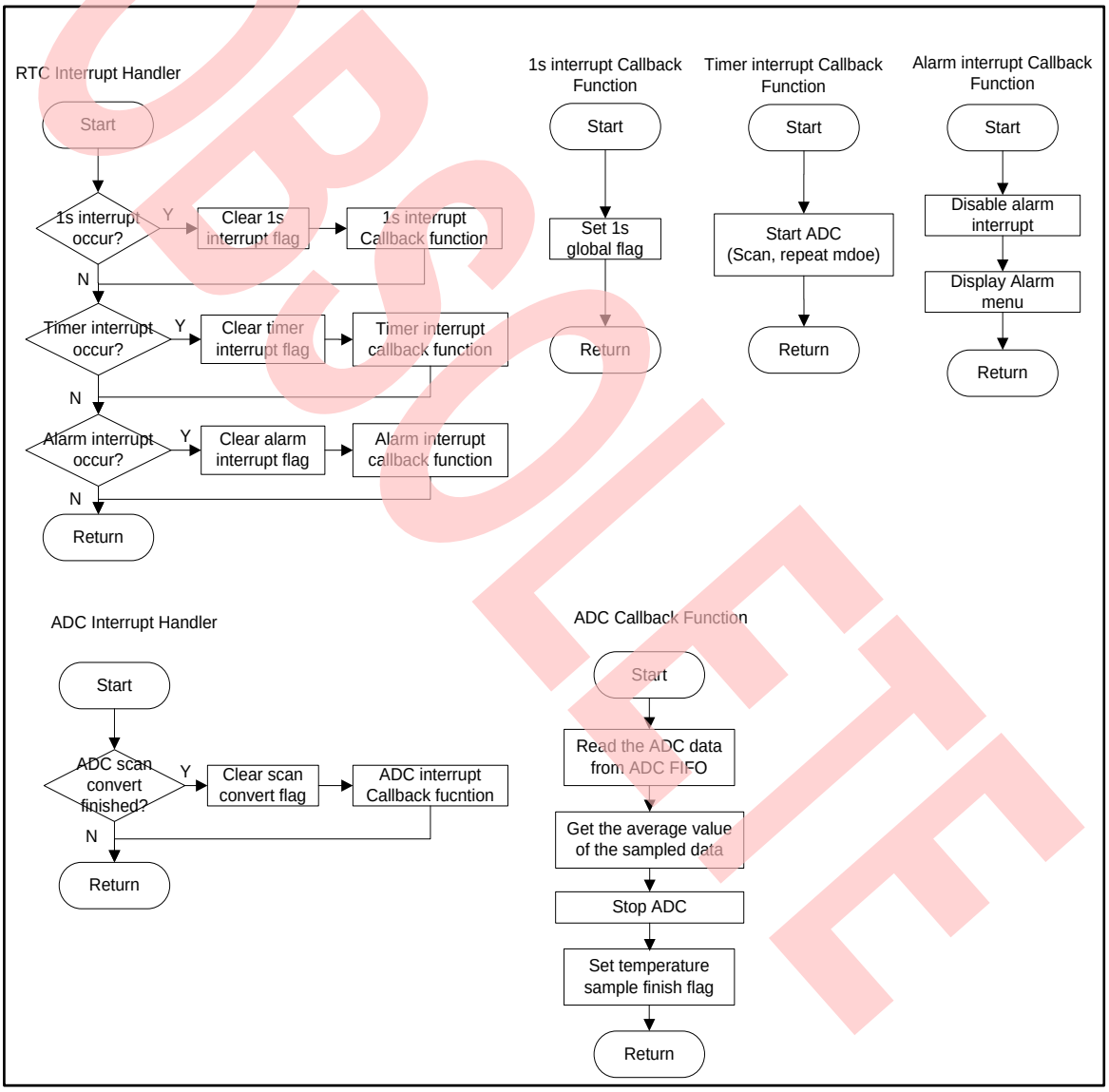
Figure 3-2: Main Program Flow of RTC Development Kit

3.1.2 Interrupt Handler

In the RTC interrupt handler, there are three RTC interrupt causes that are handled in the system, which are 1s interrupt, timer interrupt and Alarm interrupt. In 1s interrupt callback function, the 1s global flag is set, which is used to read the system time; in timer interrupt callback function, ADC is started to sample the environment temperature; in alarm interrupt, the alarm occurrence menu will display.

In the ADC interrupt handler, the system will store the sample data and set a flag which indicates the temperature sampling completion.

Figure 13. Interrupt Handler of RTC Development Kit



3.2 Configuration File

The configuration file (solution_conf.h) provides some macros which can be configured by user when he integrates this example into own applications.

- RTC_OSC_TWICE_PARA

The parabolic coefficient of 32768Hz oscillator, this value is specified by oscillator's specification.

- RTC_OSC_TURN_TEMP

The turnover temperature of 32768Hz oscillator, this value is specified by oscillator's specification.

- RTC_CALI_PPM_PER_STEP

The frequency deviation when 1 pulse is masked. [unit：ppm]

E.g. if Calibration cycle = 20s, this definition should set to 1.526 ((1/(20*32768))*10^6)

- RTC_NORMAL_DEC_DEFAULT_VAL

The default normal temperature demarcation value, in this system, this value will be stored at the Flash area (0x0001FFFC~0x0001FFFF). If the data at this area is 0xFFFFFFFF, this value will be treated as normal temperature demarcation value,.

- RTC_NORMAL_DEC_VAL_ADDR

This macro defines the store address of normal temperature demarcation value.

- RTC_CLOCK_PRECCALER_VALUE

This macro defines the prescaler of RTC clock, which also indicates the standard clock of RTC.

E.g. If this value is set to 32760, this means the 32760Hz will be standard frequency. Assume the affection of temperature is ignored, if the external clock is 32760Hz, it will not be compensated, if the external clock is 32768Hz, 244.2ppm (((32768-32760)/32760)*10^6) will be compensated.

- RTC_CLOCK_CALIBRATION_CYCLE

The macro defines calibration cycle. If it is set to 20, the Frequency Correct Block will make calibration every 20 second. [unit：s]

- RTC_CLOCK_CALIBRATION_PERIOD

The macro defines calibration period, which means the interval to make a temperature sample and modify the calibration value. [unit：s]

E.g. if set the calibration period to 15 minutes, set this macro to 900.

- ADC_TEMP_SMPL_CH

This micro defines the temperature sample channel of ADC.

E.g. If user wants to use channel 14 to make temperature sample, set this channel to AN14

- ADC_TEMP_SAMPLE_TIMES_PER_TIME

This macro defines temperature sample times of each time. The final sample value will be the average of these data. This value can set to 1~16.

- UART0_PCLK_FREQ

This macro defines the peripheral clock of UART0, this depends on the external clock and clock setting. [unit: Hz]

- UART0_BAUD_RATE

This macro defines UART0 communication baud rate. [unit: bps]

3.3 RTC Driver

RTC driver implements both RTC calendar function and RTC clock (calibration) function.

- RTC Clock API:
  - void RTC_ClkInSel(uint8_t InClk);
  - void RTC_SetCaliVal(uint16_t CaliValue);
  - void RTC_EnableCali(void);
  - void RTC_DisableCali(void);
  - void RTC_SetCaliCycle(uint8_t CaliCycle);
  - void RTC_EnableSuboutDiv(void);
  - void RTC_DisableSuboutDiv(void);
  - void RTC_SetSuboutDivVal(uint8_t Div);
  - void RTC_COSel(uint8_t COSel);
- RTC Calendar API:
  - void RTC_RegInit(void);
  - void RTC_SetCntCycle(uint32_t CntCycle);
  - void RTC_Stop(void);
  - void RTC_Start(void);
  - void RTC_Reset(void);
  - void RTC_InitTime(RTC_TimeInfoT* pTime);
  - void RTC_InitDate(RTC_DateInfoT* pDate);
  - void RTC_InitDay(RTC_DayInfoT Day);
  - void RTC_WriteTime(RTC_TimeInfoT* pTime,uint8_t CntRstFlag);
  - void RTC_WriteDate(RTC_DateInfoT* pDate,uint8_t CntRstFlag);
  - void RTC_WriteDay(RTC_DayInfoT Day, uint8_t CntRstFlag);
  - void RTC_ReadTime(RTC_TimeInfoT* pTime);
  - void RTC_ReadDate(RTC_DateInfoT* pDate);
  - void RTC_ReadDay(RTC_DayInfoT *pDay);
  - void RTC_EnableAlarmMinCmp(void);
  - void RTC_DisableAlarmMinCmp(void);
  - void RTC_EnableAlarmHourCmp(void);
  - void RTC_DisableAlarmHourCmp(void);
  - void RTC_EnableAlarmDateCmp(void);
  - void RTC_DisableAlarmDateCmp(void);
  - void RTC_EnableAlarmMonCmp(void);
  - void RTC_DisableAlarmMonCmp(void);
  - void RTC_EnableAlarmYearCmp(void);
  - void RTC_DisableAlarmYearCmp(void);
  - void RTC_ConfigAlarm(RTC_AlarmInfoT *pAlarmInfo);

- void RTC_EnableAlarmInt(RTC_AlarmIntCallback* RTCAlarmIntCallback);
- void RTC_DisableAlarmInt(void);
- void RTC_EnableSubSecInt(RTC_SubSecIntCallback* RTCSubSecIntCallBack);
- void RTC_DisableSubSecInt(void);
- void RTC_EnableSecInt(RTC_SecIntCallback* RTCSecIntCallBack);
- void RTC_DisableSecInt(void);
- void RTC_EnableMinInt(RTC_MinIntCallback* RTCMinIntCallBack);
- void RTC_DisableMinInt(void);
- void RTC_EnableHourInt(RTC_HourIntCallback* RTCHourIntCallback);
- void RTC_DisableHourInt(void);
- void RTC_EnableTimer(void);
- void RTC_DisableTimer(void);
- void RTC_EnableTimerInt(RTC_TimerIntCallback* RTCTimerIntCallback);
- void RTC_DisableTimerInt(void);
- void RTC_ConfigTimer(RTC_TimerInfoT* pTimerInfo);
- void RTC_ClrAllIntFlag(void);

3.3.1 API Functions

3.3.1.1 RTC_ClkInSel

Select the source clock of RTC

Prototype:

void

RTC_ClkInSel(uint8_t InClk)

Parameter:

InClk **:** The source clock of RTC

Return:

None

Description:

This function is used to select the source clock of RTC, which can be main clock or sub clock.

The parameter *InClk* can be one of following value:

RTC_CLOCK_IN_SUB_CLOCK - Sourced by sub clock

RTC_CLOCK_IN_MAIN_CLOCK - Source by main clock

Remark:

Please note that the RTC count cycle (prescaler) of main clock is different with that of sub clock to generate 0.5s sub second. The RTC count cycle is set by RTC_SetCntCycle.

3.3.1.2 RTC_SetCaliVal

Set the calibration value of RTC

Prototype:

void

RTC_SetCaliVal(uint16_t CaliValue)

Parameter:

CaliValue**:** Calibration value, it can be set between 0 with1023.

Return:

None

Description:

This function is used to set the calibration value of RTC, which means how many pulses will be masked in a calibration cycle (Set by SetCaliCycle).

3.3.1.3 RTC_EnableCali

Enable the calibration function of RTC

Prototype:

void

RTC_EnableCali(void)

Parameter:

None

Return:

None

Description:

This function is used to enable the calibration function of RTC.

3.3.1.4 RTC_DisableCali

Disable the calibration function of RTC

Prototype:

void

RTC_DisableCali(void)

Parameter:

None

Return:

None

Description:

This function is used to disable the calibration function of RTC.

3.3.1.5 RTC_SetCaliCycle

Set the calibration cycle of RTC

Prototype:

void

RTC_SetCaliCycle(uint8_t CaliCycle)

Parameter:

CaliCycle : the calibration cycle, it can be set between 1 to 64

Return:

None

Description:

This function is used to set the calibration cycle. 1 to 64 second can be set. When count of this period elapses to 0, the calibration will be implemented automatically by Frequency Correction Block.

Remark:

It is available to use this API to set the calibration cycle only when RTC calibration function is disabled.

3.3.1.6 RTC_EnableSuboutDiv

Enable the divider of RTC source clock

Prototype:

void

EnableSuboutDiv (void)

Parameter:

None

Return:

None

Description:

This function is used to enable the divider of RTC source clock.

3.3.1.7 RTC_DisableSuboutDiv

Disable the divider of RTC source clock

Prototype:

void

DisableSuboutDiv (void)

Parameter:

None

Return:

None

Description:

This function is used to disable the divider of RTC source clock.

3.3.1.8 RTC_SetSuboutDivVal

Set division value of RTC source clock

Prototype:

void

RTC_SetSuboutDivVal(uint8_t Div)

Parameter:

Div : Division value

Return:

None

Description:

The division value can be set between 1 with 32768, this function can only be used when divider is disable. Enable the divider after this value is set and the divided clock is output from SUBOUT pin.

The parameter *DIV* can be one of following value:

- RTC_DIV_RATIO_SETTING_1
- RTC_DIV_RATIO_SETTING_2
- RTC_DIV_RATIO_SETTING_4
- RTC_DIV_RATIO_SETTING_8
- RTC_DIV_RATIO_SETTING_16
- RTC_DIV_RATIO_SETTING_32
- RTC_DIV_RATIO_SETTING_64
- RTC_DIV_RATIO_SETTING_128
- RTC_DIV_RATIO_SETTING_256
- RTC_DIV_RATIO_SETTING_512
- RTC_DIV_RATIO_SETTING_1024
- RTC_DIV_RATIO_SETTING_2048
- RTC_DIV_RATIO_SETTING_4096
- RTC_DIV_RATIO_SETTING_8192
- RTC_DIV_RATIO_SETTING_16384
- RTC_DIV_RATIO_SETTING_32768

Remark:

The SUBOUT pin can be selected from SUBOUT_0, SUBOUT_1, SUBOUT_2. Configure the EPFR00 register to make this selection. (See GPIO chapter in MB9Axxx / MB9Bxxx Series PERIPHERAL MANUAL")

3.3.1.9 RTC_COSel

Select RTCCO output frequency

Prototype:

void

RTC_COSel(uint8_t COSel)

Parameter:

COSel : Frequency selection

Return:

None

Description:

This function is used to select the RTCCO output frequency. CO (always 2Hz signal) is standard clock of calendar module, which is also output to RTCCO pin. By divided by 2, a 1Hz signal can also be gotten from RTCCO.

The parameter *COSel* can be one of following value:

- RTC_CO_SEL_2Hz - Divided by 1
- RTC_CO_SEL_1Hz - Divided by 2

Remark:

The clock outputting from RTCCO has been calibrated if the calibration function is enabled. So this signal can be used to test the clock precision after calibration.

The RTCCO output is available after RTC starts (See RTC_Start).

The RTCCO pin can be selected from RTCCO _0, RTCCO_1, RTCCO_2. Configure the EPFR00 register to make this selection. (See GPIO chapter in MB9Axxx / MB9Bxxx Series PERIPHERAL MANUAL")

3.3.1.10 RTC_RegInit

Initialize RTC control registers

Prototype:

void

RTC_RegInit(void)

Parameter:

None

Return:

None

Description:

This function is used to initialize the control registers of RTC.

3.3.1.11 RTC_SetCntCycle

Set RTC control registers

Prototype:

void

RTC_SetCntCycle(uint32_t CntCycle)

Parameter:

CntCycle : Counter cycle

Return:

None

Description:

This function is used to set the counter cycle (prescaler) of RTC source clock. This value is highly critical, as it determines standard clock of calendar module. To make sure 0.5s cycle is output from CO, obtain this value by following formula:

CntCycle = RTC Source Clock/4 - 1.

Remark:

If RTC is sourced by 32768Hz sub clock, 8191 should be set to CntCycle, but if clock calibration is involved, a value less than 8191 should be set, so as to the actual frequency seems to be faster than normal, and the tiny deviation of 32768Hz can be calibrated in this way.

3.3.1.12 RTC_Stop

Stop RTC counter

Prototype:

void

RTC_Stop(void);

Parameter:

None

Return:

None

Description:

Stop the RTC counter.

Remark:

After implementing this function, the time count and RTCCO output will stop.

3.3.1.13 RTC_Start

Start RTC counter

Prototype:

void

RTC_Start(void);

Parameter:

None

Return:

None

Description:

Start the RTC counter.

Remark:

After implementing this function, the time count and RTCCO output will start.

3.3.1.14 RTC_Reset

Reset RTC counter

Prototype:

void

RTC_Reset(void);

Parameter:

None

Return:

None

Description:

Reset RTC counter.

Remark:

After implementing this function, all RTC register is reset to initial value.

3.3.1.15 RTC_InitTime

Initialize RTC time

Prototype:

void

RTC_InitTime(RTC_DateInfoT* pDate);

Parameter:

***pTime*:** Pointer to time information structure

- ***Sec:*** Second value (decimal input)

- ***Min:*** Minute value (decimal input)

- ***Hour*:** Hour value (decimal input)

Return:

None

Description:

Initialize the RTC time. The 24-hour format is supported.

Remark:

Make sure the RTC stops before using this function.

3.3.1.16 RTC_InitDate

Initialize RTC date

Prototype:

void

RTC_InitDate(RTC_DateInfoT* pDate);

Parameter:

***pDate*:** Pointer to time information structure

- ***Year:*** Year value (decimal input)

- ***Mon:*** Month value (decimal input)

- ***Date*:** Date value (decimal input)

Return:

None

Description:

Initialize the RTC date. The Year value can be set from 0 to 99. When set the date information, mind that the input date is available, or uncertain date will be set. The leap year is also compliant in the system.

Remark:

Make sure the RTC stops before using this function.

3.3.1.17 RTC_InitDay

Initialize RTC day

Prototype:

void

RTC_InitDay(RTC_DayInfoT Day)

Parameter:

***Day*:** Day enumeration data

Return:

None

Description:

Initialize the RTC day.

The parameter *Day* can be one of following value:

- RTC_DAY_SUN - Sunday
- RTC_DAY_MON - Monday
- RTC_DAY_TUE - Tuesday
- RTC_DAY_WED - Wednesday
- RTC_DAY_THU - Thursday
- RTC_DAY_FRI - Friday
- RTC_DAY_SAT - Saturday

Remark:
Make sure the RTC stops before using this function.

3.3.1.18 RTC_WriteTime

Write the time of RTC

Prototype:

void

RTC_WriteTime(RTC_TimeInfoT* pTime,

uint8_t CntRstFlag);

Parameter:

***pTime*:** Pointer to time information structure

- ***Sec:*** Second value (decimal input)

- ***Min:*** Minute value (decimal input)

- ***Hour*:** Hour value (decimal input)

CntRstFlag: the flag of time count reset or continued

Return:

None

Description:

This function is used to rewrite the time of RTC when RTC count is in run status.

The parameter *CntRstFlag* can be one of following value:

- RTC_CNT_CONTINUE - time count continue mode

- RTC_CNT_RESET - time count reset mod

3.3.1.19 RTC_WriteDate

Write the date of RTC

Prototype:

void

RTC_WriteDate(RTC_DateInfoT* pDate,

uint8_t CntRstFlag);

Parameter:

***pDate*:** Pointer to time information structure

- ***Year:*** Year value (decimal input)

- ***Mon:*** Month value (decimal input)

- ***Date*:** Date value (decimal input)

CntRstFlag: the flag of time count reset or continued

Return:

None

Description:

This function is used to rewrite the date of RTC when RTC count is in run status. The Year value can be set from 0 to 99. When set the date information, mind that the input date is available, or uncertain date will be set. The leap year is also compliant in the system.

The parameter *CntRstFlag* can be one of following value:

- RTC_CNT_CONTINUE - time count continue mode
- RTC_CNT_RESET - time count reset mod

3.3.1.20 RTC_WriteDay

Write the day of RTC

Prototype:

void RTC_WriteDay(RTC_DayInfoT Day,

uint8_t CntRstFlag);

Parameter:

***Day*:** Day enumeration data

CntRstFlag: the flag of time count reset or continued

Return:

None

Description:

This function is used to rewrite the day of RTC when RTC count is in run status.

The parameter *Day* can be one of following value:

- RTC_DAY_SUN - Sunday
- RTC_DAY_MON - Monday
- RTC_DAY_TUE - Tuesday
- RTC_DAY_WED - Wednesday

- RTC_DAY_THU - Thursday
- RTC_DAY_FRI - Friday
- RTC_DAY_SAT - Saturday

The parameter *CntRstFlag* can be one of following value:

- RTC_CNT_CONTINUE - time count continue mode
- RTC_CNT_RESET - time count reset mod

3.3.1.21 RTC_ReadTime

Read the time of RTC

Prototype:

void

RTC_ReadTime(RTC_TimeInfoT* pTime);

Parameter:

***pTime*:** Pointer to time information structure

- ***Sec:*** Second value (decimal output)

- ***Min:*** Minute value (decimal output)

- ***Hour*:** Hour value (decimal output)

Return:

None

Description:

This function is used to read the time of RTC when RTC count is in run status.

3.3.1.22 RTC_ReadDate

Read the date of RTC

Prototype:

void

RTC_ReadDate(RTC_DateInfoT* pDate)

Parameter:

***pDate*: Pointer to time information structure**

- ***Year:*** Year value (decimal output)

- ***Mon:*** Month value (decimal output)

- ***Date*:** Date value (decimal output)

Return:

None

Description:

This function is used to read the date of RTC when RTC count is in run status.

3.3.1.23 RTC_ReadDay

Read the day of RTC

Prototype:

void

RTC_ReadDay(RTC_DayInfoT *pDay)

Parameter:

***pDay*:** Pointer to day enumeration data

Return:

None

Description:

This function is used to read the day of RTC when RTC count is in run status.

The return data which *pDay* points to can be one of following value:

- RTC_DAY_SUN - Sunday
- RTC_DAY_MON - Monday
- RTC_DAY_TUE - Tuesday
- RTC_DAY_WED - Wednesday
- RTC_DAY_THU - Thursday
- RTC_DAY_FRI - Friday
- RTC_DAY_SAT - Saturday

3.3.1.24 RTC_EnableAlarmMinCmp

Enable the minute comparison of alarm

Prototype:

void

RTC_EnableAlarmMinCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to enable the minute comparison of alarm, which means the comparison of alarm minute register and minute counter will be implemented, if the value matches, the alarm interrupt flag (INTALI) will be set.

3.3.1.25 RTC_DisableAlarmMinCmp

Disable the minute comparison of alarm

Prototype:

void

RTC_DisableAlarmMinCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to disable the minute comparison of alarm. See also RTC_EnableAlarmMinCmp.

3.3.1.26 RTC_EnableAlarmHourCmp

Enable the hour comparison of alarm

Prototype:

void

RTC_EnableAlarmHourCmp(void)

Parameter:

None

Return:

None

Description:

This function is used to enable the hour comparison of alarm, which means the comparison of alarm hour register and hour counter will be implemented, if the value matches, the alarm interrupt flag (INTALI) will be set.

3.3.1.27 RTC_DisableAlarmHourCmp

Disable the hour comparison of alarm

Prototype:

void

RTC_DisableAlarmHourCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to disable the hour comparison of alarm. See also RTC_EnableAlarmHourCmp.

3.3.1.28 RTC_EnableAlarmDateCmp

Enable the date comparison of alarm

Prototype:

void

RTC_EnableAlarmDateCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to enable the date comparison of alarm, which means the comparison of alarm date register and date counter will be implemented, if the value matches, the alarm interrupt flag (INTALI) will be set.

3.3.1.29 RTC_DisableAlarmDateCmp

Disable the date comparison of alarm

Prototype:

void

RTC_DisableAlarmDateCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to disable the date comparison of alarm. See also RTC_EnableAlarmDateCmp.

3.3.1.30 RTC_EnableAlarmMonCmp

Enable the month comparison of alarm

Prototype:

void

RTC_EnableAlarmMonCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to enable the month comparison of alarm, which means the comparison of alarm month register and month counter will be implemented, if the value matches, the alarm interrupt flag (INTALI) will be set.

3.3.1.31 RTC_DisableAlarmMonCmp

Disable the month comparison of alarm

Prototype:

void

RTC_DisableAlarmMonCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to disable the month comparison of alarm. See also RTC_EnableAlarmMonCmp.

3.3.1.32 RTC_EnableAlarmYearCmp

Enable the year comparison of alarm

Prototype:

void

RTC_EnableAlarmYearCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to enable the month comparison of alarm, which means the comparison of alarm year register and year counter will be implemented, if the value matches, the alarm interrupt flag (INTALI) will be set.

3.3.1.33 RTC_DisableAlarmYearCmp

Disable the year comparison of alarm

Prototype:

void

RTC_DisableAlarmYearCmp(void);

Parameter:

None

Return:

None

Description:

This function is used to disable the year comparison of alarm. See also RTC_EnableAlarmYearCmp.

3.3.1.34 RTC_ConfigAlarm

Configure the timer of RTC

Prototype:

void

RTC_ConfigAlarm(RTC_AlarmInfoT *pAlarmInfo)

Parameter:

pAlarmInfo : Pointer to alarm inforamtion structure

- ***Time :*** Time structure

- ***Date :*** Time structure

- ***AlarmMode :*** The mode of alarm setting

- ***RTCAlarmIntCallback:*** Pointer to callback function of alarm interrupt

Return:

None

Description:

This function is used to configure alarm time/date of RTC, the elements which will be taken comparison depends on the element comparison enable/disable API (see RTC_EnableAlarmxxxCmp functions). Both once mode and cycle mode are supported.

The parameter *AlarmMode* can be one of following value:

- RTC_ALARM_ONCE - once mode
- RTC_ALARM_CYCLE - cycle mode

3.3.1.35 RTC_EnableAlarmInt

Enable alarm interrupt

Prototype:

void

RTC_EnableAlarmInt(void)

Parameter:

None

Return:

None

Description:

This function is used to enable alarm interrupt. If the RTC alarm setting time matches the RTC counter's time (assume the elements to be comparison are enabled) and the RTC alarm interrupt is enable, RTC Alarm interrupt request will generate.

3.3.1.36 RTC_DisableAlarmInt

Disable alarm interrupt

Prototype:

void

RTC_DisableAlarmInt(void)

Parameter:

None

Return:

None

Description:

This function is used to disable alarm interrupt. See also RTC_EnableAlarmInt.

3.3.1.37 RTC_EnableSubSecInt

Enable RTC sub second interrupt

Prototype:

void

RTC_EnableSubSecInt(RTC_SubSecIntCallback* RTCSubSecIntCallBack)

Parameter:

RTCSubSecIntCallBack: Pointer to callback function of RTC sub second interrupt

Return:

None

Description:

This function is used to enable sub second interrupt, if it is enable, every 0.5 second, the sub second will generate. User can make a callback function as parameter for this API, in which something can be done when the interrupt occurs.

3.3.1.38 RTC_DisableSubSecInt

Disable RTC sub second interrupt

Prototype:

void

RTC_DisableSubSecInt(RTC_SubSecIntCallback* RTCSubSecIntCallBack)

Parameter:

None

Return:

None

Description:

This function is used to disable sub second interrupt, see also RTC_EnableSubSecInt.

3.3.1.39 RTC_EnableSecInt

Enable RTC second interrupt

Prototype:

void

RTC_EnableSecInt(RTC_SecIntCallback* RTCSecIntCallBack)

Parameter:

RTCSecIntCallBack: Pointer to callback function of RTC second interrupt

Return:

None

Description:

This function is used to enable second interrupt, if it is enable, every second, the second interrupt will generate. User can make a callback function as parameter for this API, in which something can be done when the interrupt occurs.

3.3.1.40 RTC_DisableSecInt

Disable RTC second interrupt

Prototype:

void

RTC_DisableSecInt(void)

Parameter:

None

Return:

None

Description:

This function is used to disable second interrupt. See also RTC_EnableSecInt.

3.3.1.41 RTC_EnableMinInt

Enable RTC minute interrupt

Prototype:

void

RTC_EnableMinInt(RTC_MinIntCallback* RTCMinIntCallBack)

Parameter:

RTCMinIntCallBack: Pointer to callback function of RTC minute interrupt

Return:

None

Description:

This function is used to enable minute interrupt, if it is enable, every minute, the minute interrupt will generate. User can make a callback function as parameter for this API, in which something can be done when the interrupt occurs.

3.3.1.42 RTC_DisableMinInt

Disable RTC minute interrupt

Prototype:

void

RTC_DisableMinInt(void)

Parameter:

None

Return:

None

Description:

This function is used to disable minute interrupt. See also RTC_EnableMinInt.

3.3.1.43 RTC_EnableHourInt

Enable RTC hour interrupt

Prototype:

void

RTC_EnableHourInt(RTC_HourIntCallback* RTCHourIntCallBack)

Parameter:

RTCHourIntCallBack: Pointer to callback function of RTC hour interrupt

Return:

None

Description:

This function is used to enable hour interrupt, if it is enable, every hour, the hour interrupt will generate. User can make a callback function as parameter for this API, in which something can be done when the interrupt occurs.

3.3.1.44 RTC_DisableHourInt

Disable RTC hour interrupt

Prototype:

void

RTC_DisableHourInt(void)

Parameter:

None

Return:

None

Description:

This function is used to disable hour interrupt. See also RTC_EnableHourInt

3.3.1.45 RTC_EnableTimer

Enable timer operation

Prototype:

void

RTC_EnableTimer(void)

Parameter:

None

Return:

None

Description:

This function is used to enable timer operation. See also RTC_DisableTimer

Remark:

The RTC timer is independent from RTC counter. Even if RTC counter stops, the timer operation starts after it is enabled.

3.3.1.46 RTC_DisableTimer

Disable timer operation

Prototype:

void

RTC_DisableTimer(void)

Parameter:

None

Return:

None

Description:

This function is used to disable timer operation. See also RTC_EnableTimer

3.3.1.47 RTC_EnableTimerInt

Enable timer interrupt

Prototype:

void

RTC_EnableTimerInt(RTC_TimerIntCallback* RTCTimerIntCallback);

Parameter:

RTCTimerIntCallback Pointer to callback function of RTC timer interrupt

Return:

None

Description:

This function is used to enable hour interrupt, if it is enable, when the timer count elapses to 0, the timer interrupt will generate. User can make a callback function as parameter for this API, in which something can be done when the interrupt occurs.

3.3.1.48 RTC_DisableTimerInt

Disable timer interrupt

Prototype:

void

RTC_DisableTimerInt(void)

Parameter:

None

Return:

None

Description:

This function is used to disable the timer interrupt. See also RTC_EnableTimerInt.

3.3.1.49 RTC_ConfigTimer

Configure timer of RTC

Prototype:

void

RTC_ConfigTimer(RTC_TimerInfoT* pTimerInfo);

Parameter:

pTimerInfo: Pointer to timer configuration structure.

- Count: the count value of RTC timer

- TimerMode: timer mode

Return:

None

Description:

This function is used to configure RTC timer, both one shot mode and period mode are supported.

The parameter *TimerMode* can be one of following values:

- RTC_TIMER_ONCE

- RTC_TIMER_CYCLE

Remark:

Make sure RTC timer stops before calling this function.

3.3.1.50 RTC_ClrAllIntFlag

Clear all RTC interrupt flags

Prototype:

void

RTC_ClrAllIntFlag(void)

Parameter:

None

Return:

None

Description:

This function is used to clear all RTC interrupt flags.

3.3.2 Example Program

3.3.2.1 RTC Initialization

The following program gives example to use the APIs to initialize RTC. It implements following functions:

- Make the reference clock to calendar as 2Hz (0.5s cycle)
  - Set the count cycle to 8189 (prescaler: 32760, standard frequency: 32760Hz)
- Output 1Hz signal from RTCCO_1
- Initialize RTC calendar (08:30, 5/4/2012)
- Enable calibration function
  - Set calibration cycle to 20s
  - Initialize calibration value to 168

Figure 14. RTC Initialization Example Program

```
RTC_TimeInfoT   Time = {0};
RTC_DateInfoT   Date = {0};
/* Config RTCCO 1 */
FM3_GPIO->ADE &= ~(1ul<<3);
FM3_GPIO->PFR1 |= (1ul<<3);
FM3_GPIO->EPFR00 |= (2ul<<4);
/* Clear all RTC interrupt flag */
RTC_ClrAllIntFlag();
/* Stop RTC */
RTC_Stop();
/* Select source clock of RTC */
RTC_ClkInSel(RTC_CLOCK_IN_SUB_CLOCK);
/* Stop Subout clock divider */
RTC_DisableSuboutDiv();
/* Output 1Hz signal from RTCCO */
RTC_COSel(RTC_CO_SEL_1Hz);
/* Set the count cycle of RTC */
RTC_SetCntCycle(8189);
/* Disable RTC calibration function */
RTC_DisableCali();
```

```
/* Set RTC calibration cycle */
RTC_SetCaliCycle(20);
/* Enable RTC calibration function */
RTC_EnableCali();
/* Initialize calibration value */
RTC_SetCaliVal(168);
/* initial RTC time */
Time.Sec = 0;
Time.Min = 30;
Time.Hour = 8;
RTC_InitTime(&Time);
/* initial RTC Date */
Date.Date = 5;
Date.Mon = 4;
Date.Year = 12;
RTC_InitDate(&Date);
/* Start RTC */
RTC_Start();
/* Enable RTC IRQ */
NVIC_EnableIRQ(OSC_PLL_RTC_IRQn);
```

3.3.2.2 RTC Timer Setting

The following program demonstrates how to use the RTC timer to generate interrupt every 60 second. User can do something in the timer interrupt callback function.

Figure 15. RTC Timer Setting Example Program

```
RTC_TimerInfoT TimerInfo = {0};
/* Config RTC timer */
RTC_DisableTimer();
TimerInfo.Count = 119;
TimerInfo.TimerMode = (uint32_t)RTC_TIMER_CYCLE;
RTC_ConfigTimer(&TimerInfo);
RTC_EnableTimer();
RTC_EnableTimerInt(RTCTimerIntCallback);
```

3.3.2.3 RTC Alarm Setting

The following program demonstrates how to use the RTC alarm:

- Set alarm time (08:31)
- Once mode

User can do something in the alarm callback function

Figure 16. RTC Alarm Setting Example Program

```
RTC_AlarmInfoT AlarmInfo = {0};
AlarmInfo.Time.Hour = 8;
AlarmInfo.Time.Min = 31;
AlarmInfo.AlarmMode = RTC_ALARM_ONCE;
RTC_ConfigAlarm(&AlarmInfo);
RTC_EnableAlarmInt(COMM_AlarmCallback);
RTC_EnableAlarmHourCmp();
RTC_EnableAlarmMinCmp();
```

3.3.2.4 RTC Second Interrupt Setting

The following program demonstrates how to generate the second interrupt

User can do something in the callback function of second interrupt

Figure 17. RTC Second Interrupt Setting Example Program

```
/* Enable 1s interrupt */
RTC_EnableSecInt(RTCSecIntCallback);
```

3.3.2.5 Update RTC Time

The following program gives an example to set time to “14:35:22” with continuous mode

Figure 18. RTC Time Update Example Program

```
RTC_TimeInfoT gMon_CurTime;
gMon_CurTime.Hour = 14;
gMon_CurTime.Min = 35;
gMon_CurTime.Sec = 22;
RTC_WriteTime(&gMon_CurTime, RTC_CNT_CONTINUE);
```

3.3.2.6 Update RTC Date

The following program gives an example to set date to “2012-6-19” with continuous mode

Figure 19. RTC Date Update Example Program

```
RTC_DateInfoT gMon_CurDate;
gMon_CurDate.Year = 12;
gMon_CurDate.Mon = 6;
gMon_CurDate.Date = 19;
RTC_WriteDate(&gMon_CurDate, RTC_CNT_CONTINUE);
```

3.3.2.7 Read RTC Time

The following program gives an example to read the current time.

Figure 20. RTC Time Read Example Program

```
RTC_TimeInfoT gMon_CurTime;
RTC_ReadTime(&gMon_CurTime);
```

3.3.2.8 Read RTC Date

The following program gives an example to read the current date

Figure 21. RTC Date Read Example Program.

```
RTC_DateInfoT gMon_CurDate;
RTC_ReadDate(&gMon_CurDate);
```

3.3.2.9 Update Calibration value

Figure 22. RTC Calibration Value Update Example Program

```
/* Set RTC calibration cycle */
uint8_t NewCali = 140 ;
RTC_SetCaliVal(NewCali);
```

3.4 Low Power Consumption Driver

MB9AFA32N supports both standby mode and deep standby mode. The diver of low power consumption mode includes following APIs:

- void LowPwrCon_GoToStandByMode(LowPwrModeT Type, uint8_t IOStat);
- void LowPwrCon_EnableRetCause(RetCauseEnT Type);
- void LowPwrCon_DisableRetCause(RetCauseEnT Type);
- void LowPwrCon_SetWkupPinLevel(WkupPinIndexT PinIndex, uint8_t Level);
- RetCauseFlagT LowPwrCon_ReadRetCause(void);
- void LowPwrCon_WriteBackupReg(BackupRegT RegIndex, uint8_t Data);
- void LowPwrCon_ReadBackupReg(BackupRegT RegIndex, uint8_t *Data);

3.4.1 API Functions

3.4.1.1 LowPwrCon_GoToStandByMode

Enter into the low power consumption mode

Prototype:

void

LowPwrCon_GoToStandByMode(LowPwrModeT Type,

uint8_t IOStat)

Parameter:

***Type*:** the enumeration of lower power consumption mode

IOStat: I/O status when entering into lower power consumption mode

(0: Retain, 1: High impendence)

Return:

None

Description:

This function is used to enter into low power consumption mode. User should prepare the return cause (wakeup ways) before entering into the standby mode. Set the return cause of deep standby mode with LowPwrCon_EnableRetCause

The parameter *Type* can be one of following values:

STB_SlpMode - standby sleep mode

- STB_TimerMode - standby timer mode
- STB_RTCMode - standby RTC mode
- STB_StopMode - standby stop mode
- DPSTB_RTCMode - deep standby RTC mode
- DPSTB_StopMode - deep standby stop mode

3.4.1.2 LowPwrCon_EnableRetCause

Enable the return cause of deep standby mode.

Prototype:

void

LowPwrCon_EnableRetCause(RetCauseEnT Type)

Parameter:

***Type*:** the enumeration of deep standby mode return cause

Return:

None

Description:

This function is used to enable the return cause of deep standby mode.

The parameter *Type* can be one of following values:

- DSTB_RetCause_RTCInt - return from RTC interrupt
- DSTB_RetCause_LVD - return from LVD interrupt
- DSTB_RetCause_WkupPin1 - return from Wkup pin1 detection
- DSTB_RetCause_WkupPin2 - return from Wkup pin2 detection
- DSTB_RetCause_WkupPin3 - return from Wkup pin3 detection
- DSTB_RetCause_WkupPin4 - return from Wkup pin4 detection
- DSTB_RetCause_WkupPin5 - return from Wkup pin5 detection
- DSTB_RetCause_CEC0 - return from Wkup CEC1 reception interrupt
- DSTB_RetCause_CEC1 - return from Wkup CEC2 reception interrupt

Remark:

The return from WKUP0 is always enabled. Before enable wakeup from wakeup pin, make sure that the related wakeup pin exists. E.g. MB9AFA32N only has WKUP0 - WKUP3.

3.4.1.3 LowPwrCon_DisableRetCause

Disable the return cause of deep standby mode.

Prototype:

void

LowPwrCon_DisableRetCause(RetCauseEnT Type)

Parameter:

***Type*:** the enumeration of deep standby mode return cause

Return:

None

Description:

This function is used to disable the return cause of deep standby mode.

The parameter *Type* can be one of following values:

- DSTB_RetCause_RTCInt - return from RTC interrupt
- DSTB_RetCause_LVD - return from LVD interrupt
- DSTB_RetCause_WkupPin1 - return from Wkup pin1 detection

- DSTB_RetCause_WkupPin2 - return from Wkup pin2 detection
- DSTB_RetCause_WkupPin3 - return from Wkup pin3 detection
- DSTB_RetCause_WkupPin4 - return from Wkup pin4 detection
- DSTB_RetCause_WkupPin5 - return from Wkup pin5 detection
- DSTB_RetCause_CEC0 - return from Wkup CEC1 reception interrupt
- DSTB_RetCause_CEC1 - return from Wkup CEC2 reception interrupt

Remark:

See also LowPwrCon_EnableRetCause.

3.4.1.4 LowPwrCon_SetWkupPinLevel

Set the valid level of wakeup pin

Prototype:

void

LowPwrCon_SetWkupPinLevel(WkupPinIndexT PinIndex,

uint8_t Level);

Parameter:

***PinIndex*:** The pin index of wakeup pin

***Level*:** The valid level (High or Low)

Return:

None

Description:

This function is used to set the valid level of wakeup pin.

The parameter *PinIndex* can be one of following values:

- WKUP_PIN1 - index of wkup pin 1
- WKUP_PIN2 - index of wkup pin 2
- WKUP_PIN3 - index of wkup pin 3
- WKUP_PIN4 - index of wkup pin 4
- WKUP_PIN5 - index of wkup pin 5

Remark:

The valid level of wkup pin 0 is always low.

3.4.1.5 LowPwrCon_ReadRetCause

Read the return cause of deep standby mode

Prototype:

RetCauseFlagT

LowPwrCon_ReadRetCause(void);

Parameter:

None

Return:

Return cause

Description:

This function is used to read the return cause of deep standby mode

The return value can be one of following values:

- DSTB_RetCauseFlag_RTCInt - return from RTC interrupt
- DSTB_RetCauseFlag_LVD - return from LVD interrupt
- DSTB_RetCauseFlag_WkupPin0 - return from Wkup pin0 detection
- DSTB_RetCauseFlag_WkupPin1 - return from Wkup pin1 detection
- DSTB_RetCauseFlag_WkupPin2 - return from Wkup pin2 detection
- DSTB_RetCauseFlag_WkupPin3 - return from Wkup pin3 detection
- DSTB_RetCauseFlag_WkupPin4 - return from Wkup pin4 detection
- DSTB_RetCauseFlag_WkupPin5 - return from Wkup pin5 detection
- DSTB_RetCauseFlag_CEC0 - return from Wkup CEC1 reception interrupt
- DSTB_RetCauseFlag_CEC1 - return from Wkup CEC2 reception interrupt

Remark:

Only the return cause of deep standby mode can be identified, the return cause of standby mode can not be identified

3.4.1.6 LowPwrCon_WriteBackupReg

Write data into a backup register

Prototype:

void

LowPwrCon_WriteBackupReg(BackupRegT RegIndex,

uint8_t Data);

Parameter:

***RegIndex*:** The index of backup register

Data: The date to be written

Return:

None

Description:

This function is used to write data into a backup register.

The parameter *Type* can be one of following values:

- BACKUP_REG1 - backup register 1
- BACKUP_REG2 - backup register 2
- BACKUP_REG16 - backup register 3

Remark:

In MB9AFA32N, there are 16 bytes backup registers, which can retain the data after system enters into deep standby mode.

3.4.1.7 LowPwrCon_ReadBackupReg

Read data from the backup register

Prototype:

void

LowPwrCon_ReadBackupReg(BackupRegT RegIndex,

uint8_t *Data);

Parameter:

RegIndex: The index of backup register

****Data***: Pointer to the read data

Return:

None

Description:

This function is used to read data from a backup register.

The parameter *Type* can be one of following values:

- BACKUP_REG1 - backup register 1
- BACKUP_REG2 - backup register 2
- BACKUP_REG16 - backup register 3

Remark:

In MB9AFA32N, there are 16 bytes backup registers, which can retain the data after system enters into deep standby mode.

3.4.2 Example Program

3.4.2.1 Enter into RTC Mode

The following program demonstrates how to enter into RTC mode and wakeup by RTC timer interrupt after 1 minute.

Figure 23. Example Program of Entering into RTC Mode

```
/* Config RTC timer */
RTC_DisableTimer();
TimerInfo.Count = 119;
TimerInfo.TimerMode = (uint32_t)RTC_ALARM_CYCLE;
RTC_ConfigTimer(&TimerInfo);
RTC_EnableTimer();
/* Enter RTC mode */
LowPwrCon_GoToStandByMode(STB_RTCMode)
```

3.4.2.2 Enter into Deep Standby RTC Mode

The following program demonstrates how to enter into deep standby RTC mode and wakeup by wakeup pin 1 (WKUP1).

Figure 24. Example Program of Entering into Deep Standby RTC Mode

```
/* Enable return cause of WKUP1 */
LowPwrCon_EnableRetCause(DSTB_RetCause_WkupPin1);
    /* Low level validity */
LowPwrCon_SetWkupPinLevel(WKUP_PIN1, VALID_LEVEL_LOW);
/* Enter into deep standby RTC mode */
LowPwrCon_GoToStandByMode(DPSTB_RTCMode, 0);
```

4 Firmware Demonstration

This example can demonstrate following functions:

- Read Time

24 hours format [hh:mm:ss]

- Read Date

[yy-mm-dd]

- Write Time

24 hours format [hh:mm:ss]

- Write Date

[yy-mm-dd]

- Set Alarm

Once mode

The hour and minute of alarm can be set

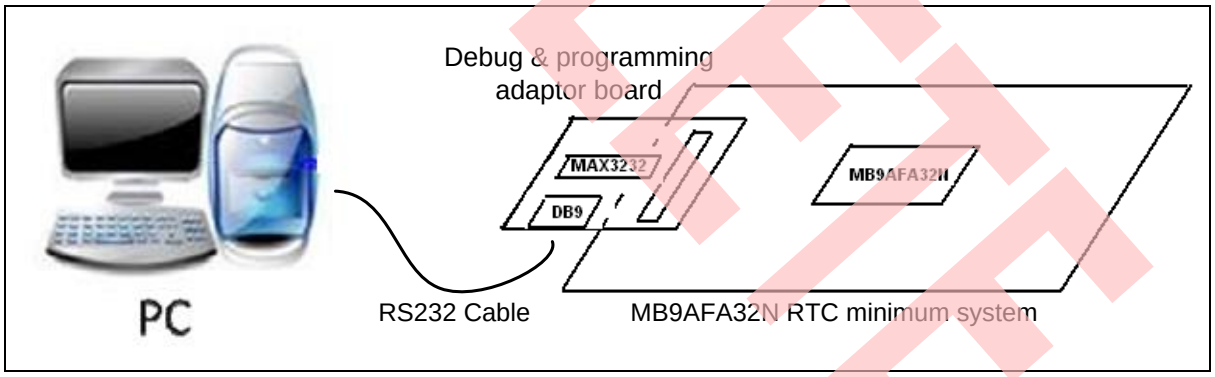
- Calibration Process Monitor

Time, calibration cycle, Fo, temperature, calibration cycle can be monitored

- Enter into RTC mode
- Enter into deep standby RTC Mode

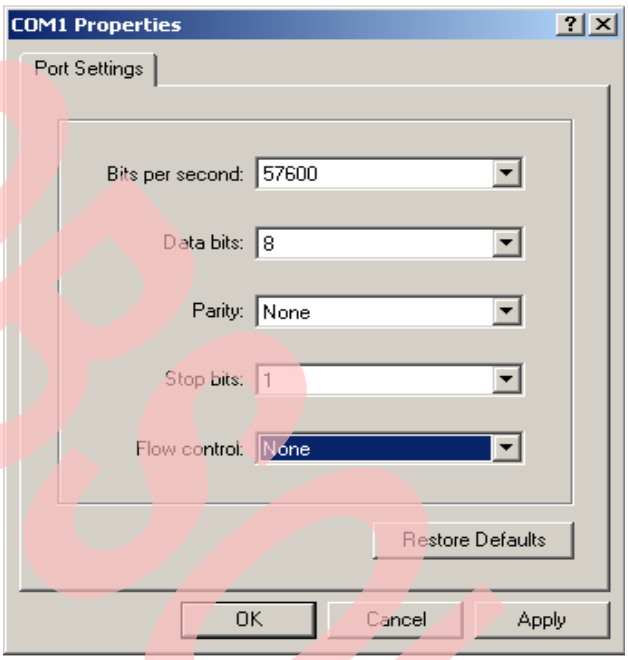
Before using this example, make sure the program has been programmed into MCU Flash and prepare the hardware as following:

Figure 25. Hardware Preparation for RTC Development Kit



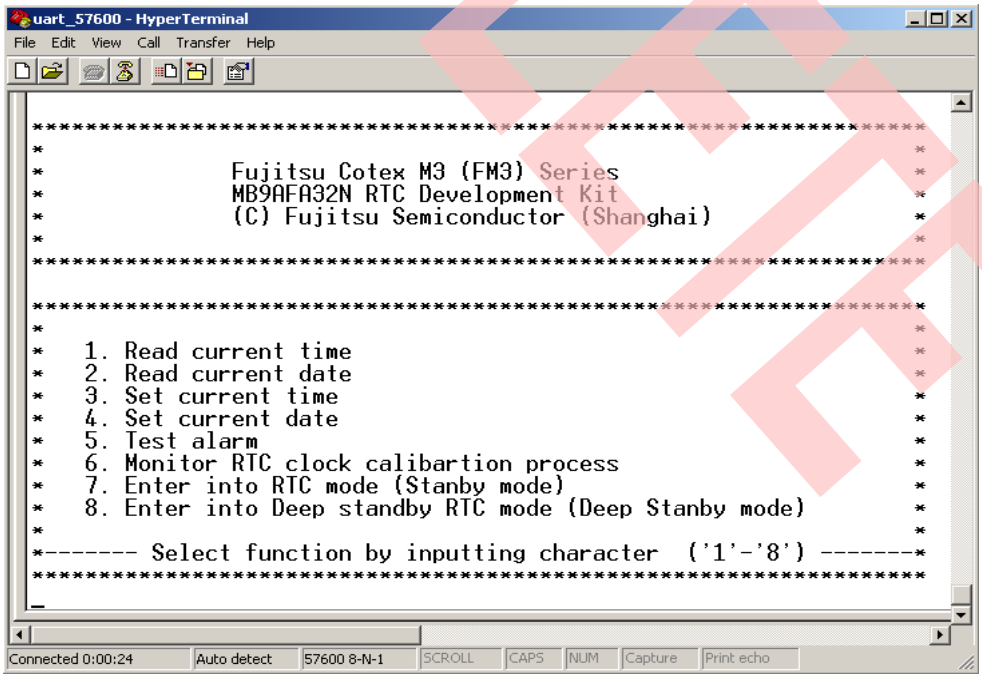
Configure the Hyper terminal as following figure, the baud rate of COM port should be same with the macro (UART0_BAUD_RATE) in solution_conf.h.

Figure 26. Hyper Terminal Setting



Power on the target board, the top menu of RTC development kit displays in the Hyper Terminal:

Figure 27. Top Menu of RTC Development Kit



4.1 Read Time

Input '1' to read RTC time. Input '*' to get back to the top menu.

4.2 Read Date

Input ‘2’ to read the RTC time. Input ‘*’ to get back to the top menu.

4.3 Write Time

Input ‘3’ to write RTC time.

4.4 Write Date

Input ‘4’ to write RTC Date.

4.5 Set Alarm

Input ‘5’ to enter into alarm setting menu.

After alarm time is set, the system will wait for alarm occurs.

During the waiting time, input ‘*’ to cancel the alarm and get back to the top menu.

4.6 Calibration Process Monitor

Input ‘6’ to monitor the calibration process. Following information can be monitored. The value of some information depends on the macro setting in the configuration file.

- Standard Frequency
  - Depend on RTC_CLOCK_PRECCALER_VALUE
- Normal temperature demarcation value
  - Depend on the value in the Flash address (0x0001FFFC~0x0001FFFF)
- Calibration Cycle
  - Depend on RTC_CLOCK_CALIBRATION_CYCLE
- Temperature
- Calibration Value

The calibration interval time depends on RTC_CLOCK_CALIBRATION_PERIOD.

Figure 28. Calibration Process Monitor Menu of RTC Development Kit

```
************** Calibartion Monitor ************************
Time:  08:44:00
Standard Freq:  32760Hz
Fo:  12.2ppm [Default value]
Calibration Cycle: 20
-------------------------------------------------------------
Temperature: 28'C
Calibration Value: 168
*************************************************************
```

4.7 Enter into RTC mode

Input ‘7’ to enter into RTC mode. In this system, the RTC interrupt is enabled before enter into RTC mode, so it is waken up immediately by RTC 1 second interrupt.

User can make following try:

1. Disable RTC interrupt
2. Try to use external interrupt, NMI to wake up from RTC mode.

When the system wakes up, the RTC register and some global variable are read. We can find that the value of RTC register retains and the value of global variable losses.

4.8 Enter into Deep Standby RTC mode

Input ‘8’ to enter into deep standby RTC mode. In this system, user can select to use RTC interrupt or WKUP1 as return cause.

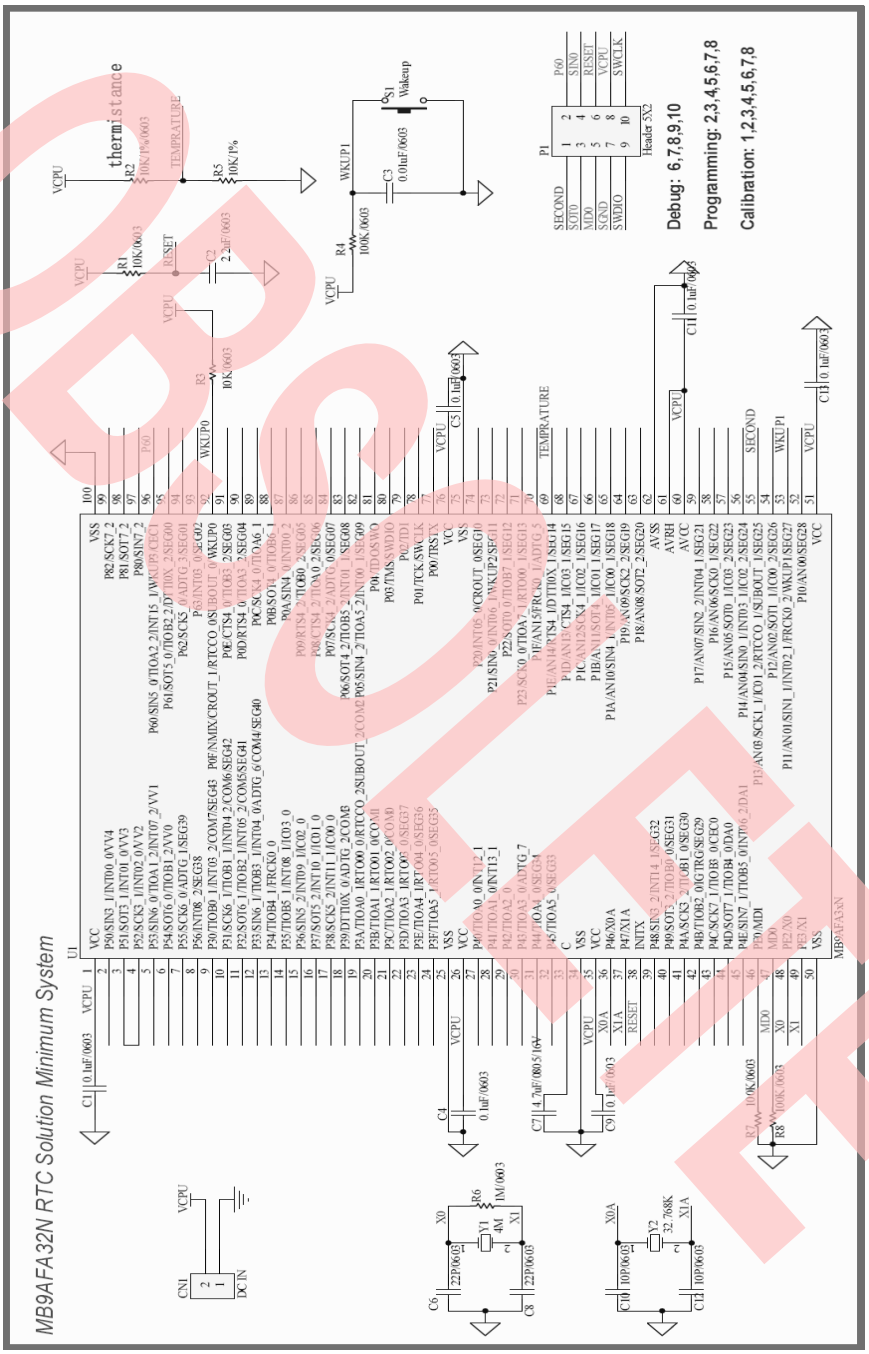
Input ‘1’ to return from deep standby RTC mode by RTC interrupt

Input ‘2’ to return from deep standby RTC mode by WKUP1

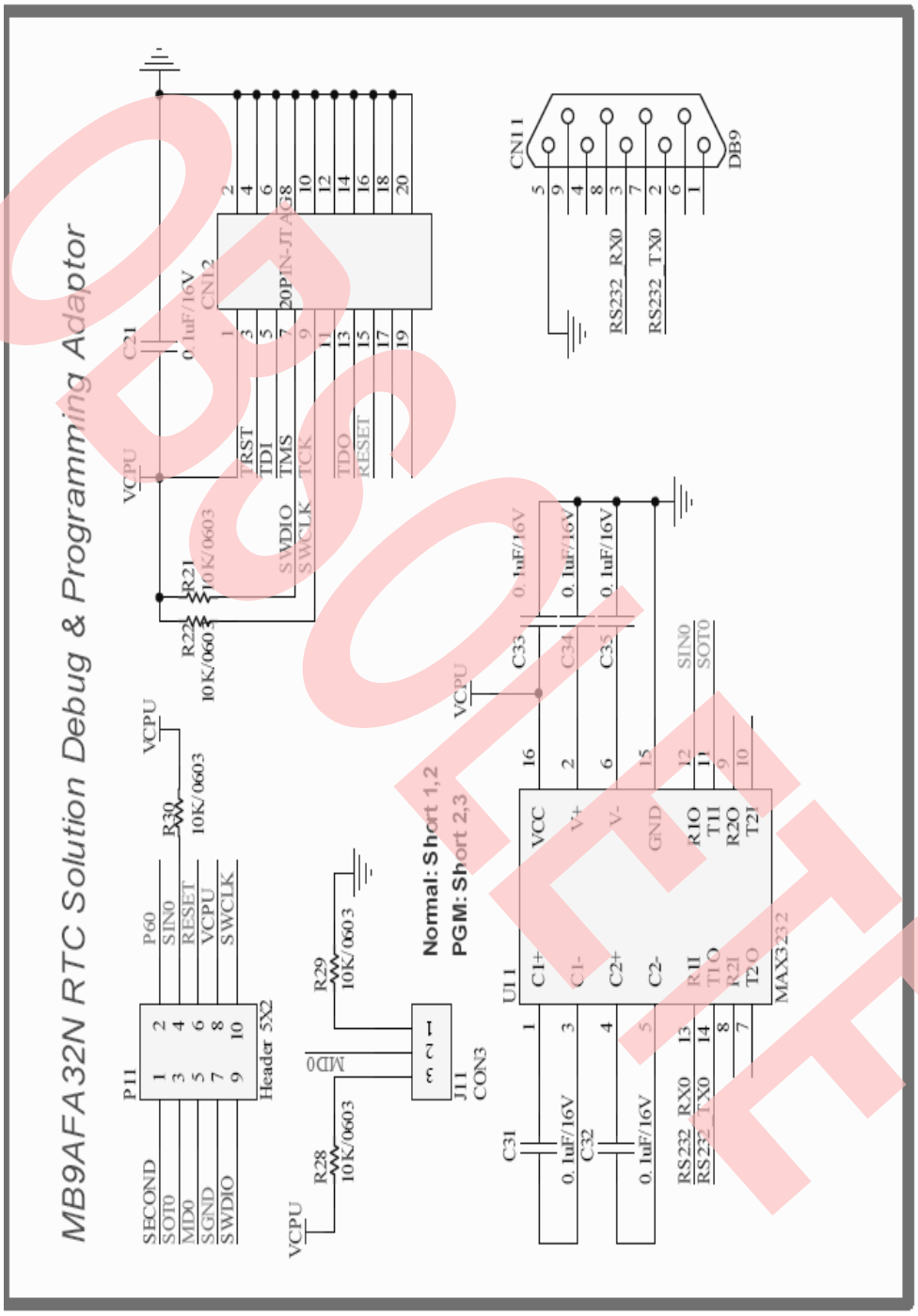
When the system wakes up, the RTC register and some global variable are read. We can find that the value of RTC register retains and the value of global variables are all lost.

5 Annex

5.1 Schematic of MB9AFA32N RTC Solution Minimum System:



5.2 Schematic of RTC solution Debug & Programming Adaptor:



5.3 RTC Clock Precision [After calibration]

Sample 1		Sample 2		Sample 3	
T [℃]	Δf [ppm]	T [℃]	Δf [ppm]	T [℃]	Δf [ppm]
-10	2.633	-10	4.953	-10	3.689
0	1.752	0	2.842	0	1.684
10	-0.522	10	-0.487	10	0.336
20	-0.487	20	-0.522	20	-0.731
30	-0.592	30	-0.998	30	-1.578
40	-1.612	40	-2.448	40	-2.181
50	-2.343	50	-3.666	50	-4.037
60	-3.816	60	-4.976	60	-4.826

5.4 Key Component Selection

	P/N	Value	Vender
Oscillator	VT-200	32768Hz	Seiko
Thermistance	NCP18XH103F03RB	10k±1%	Murata

Document History

Document Title: AN205285 – FM3, MB9AA30 Series, MB9AFA32N RTC Solution Development Kit

Document Number: 002-05285

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	Edison Zhang	06/15/2012	V1.0
*A	5281371	HUAL	05/23/2016	Migrated Spansion Application Note “MCU-AN-510045-E-10” to Cypress format.
*B	6268708	XITO	08/02/2018	Obsoleted.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Video | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709