Software Real-Time Clock

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	Introduce implementing Software Real-Time Clock (RTC) use 38.4 kHz Low Speed Internal Oscillator (LIRC)
BSP Version	M031_Series_BSP_CMSIS_V3.01.000
Hardware	NuMaker-M031

1 Function Description

1.1 Introduction

In this example code, the NuMicro Cortex®-M0 M031 chip is used as an example. The software Real Time Clock with timing and alarm function can be realized through the low-speed RC oscillator circuit (LIRC) inside the MCU and combined with the software algorithm. In addition, the timer works in power-down mode to reduce system power consumption and extend system life.

In this application, Software Real Time Clock provides information on date (year, month, day) and time (hour, minute, second). Users can freely choose 24-hour and 12-hour output according to their needs. It can also be customized for the alarm in the interrupt handling service to meet the user's application requirements.

1.2 Real Time Clock System Architecture

Real Time Clock, which requires timing, date calculation, and record seconds function. In general, the hardware Real Time Clock has the following basic units:

1. Timer Unit: set for a period of time, the counter is accumulated when the set time is reached that is used to calculate the time
2. Calendar Calculate Unit: Calculates the updated value of the year, month, day, hour, minute, and second as the Timer is accumulated.
3. Time Information Storage Unit: store information such as year, month, day, hour, minute, second, etc.

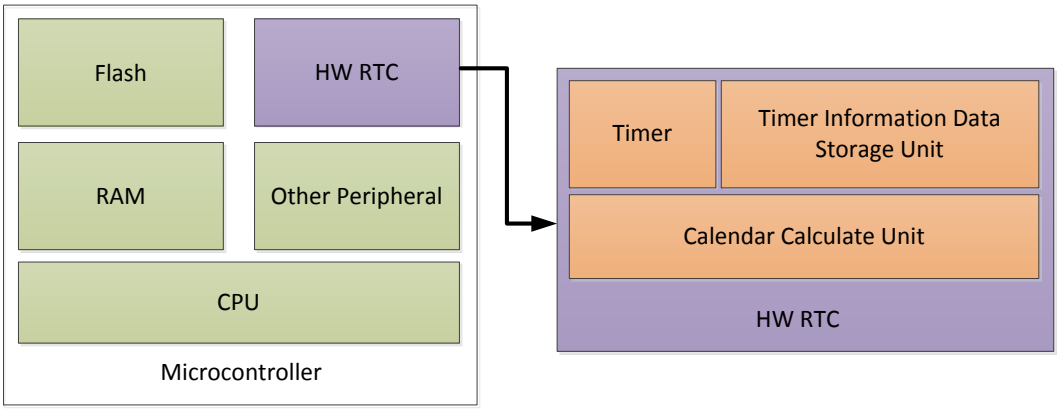


Figure 1-1 Hardware Real Time Clock System Architecture Block Diagram

This Application Note uses software to implement Real Time Clock. For the above hardware, uses the following methods:

1. Using the timer inside the M031 chip to implement the timer unit in the hardware Real Time Clock
2. Using the CPU of the M031 to implement the time calculation unit through the time update algorithm
3. Using the RAM of the M031 to implement the Time Information Storage Unit

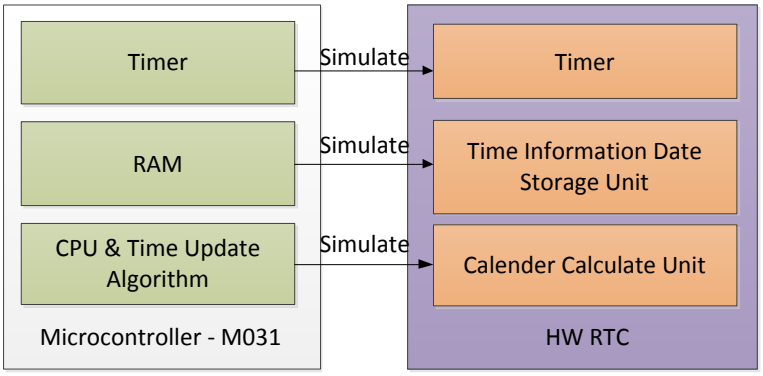


Figure 1-2 M031 Software Real Time Clock System Map

The architecture of the system is shown in Figure 1-3 Software Real Time Clock Architecture Diagram. The low-speed internal RC oscillator circuit provides the clock to the timer. The timer sends an interrupt signal to the nested vector interrupt controller. The nested vector interrupt controller informs the CPU. The interrupt event is generated, and the CPU executes the corresponding interrupt subroutine to update the time information. When the time information is updated, the current time information is first retrieved from the random access memory, and the new time is calculated after the extraction, and the random access memory is stored and the cycle is completed.

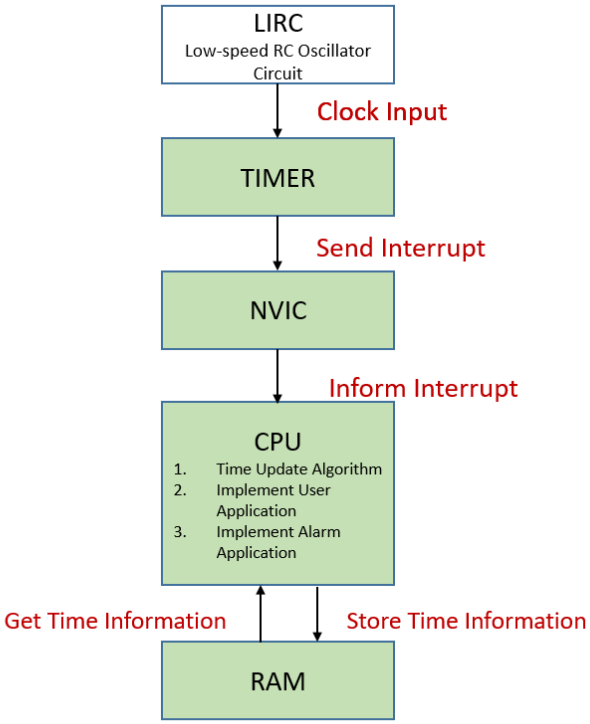


Figure 1-3 Software Real Time Clock Architecture Diagram

1.3 Software RTC Algorithm

1.3.1 Time Update Algorithm

Assuming that the timer wakes up the chip and has passed the previous update for 1 second, in the timer interrupt service routine RTC_IRQ_Process(), _RTC_UpdateTime() is called to update the time information. The following is the calculation process for this function:

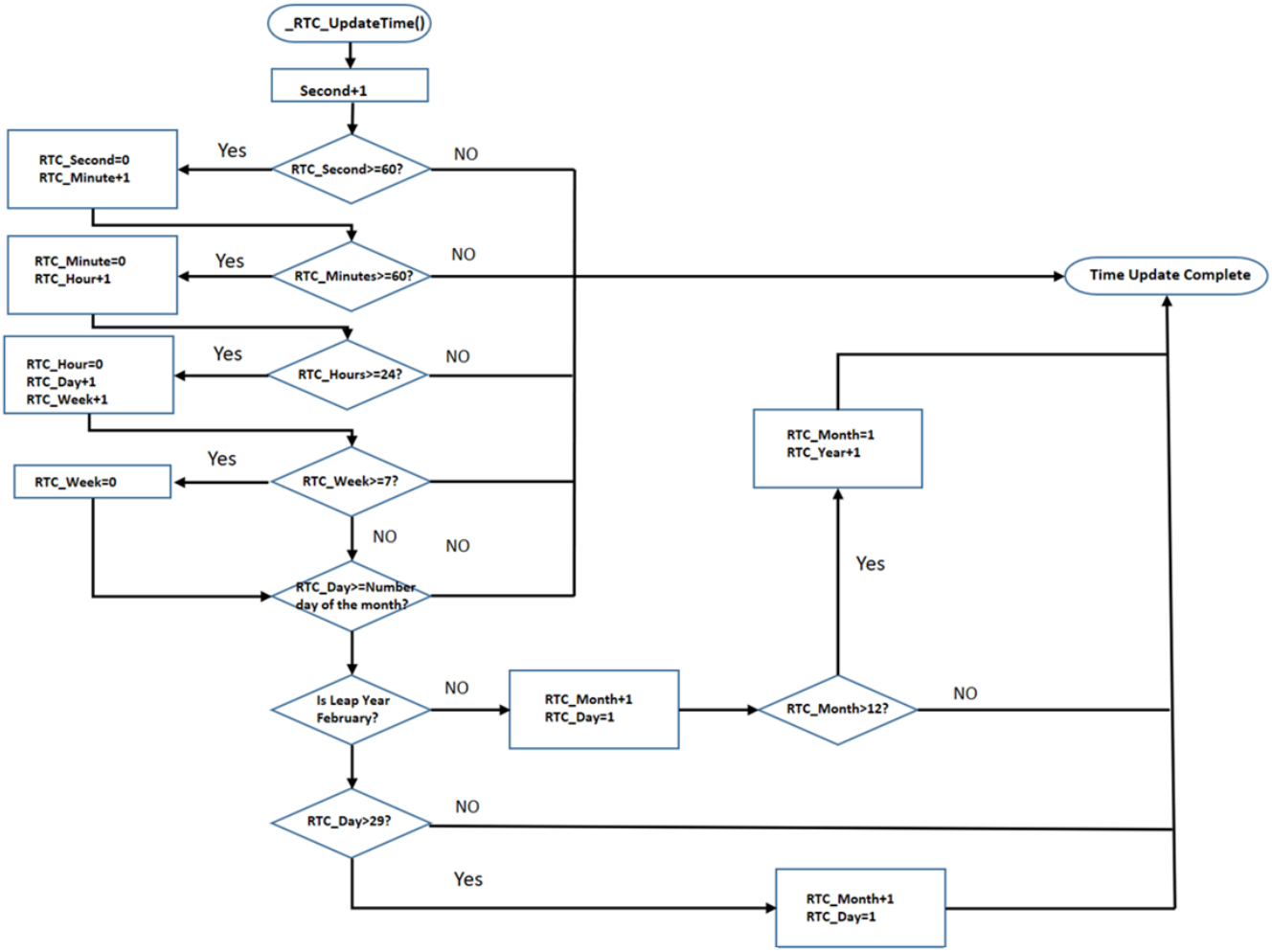


Figure 1-4 Time Update Algorithm Flow Chart

1.3.2 Leap Year Judgement Algorithm

There is an error between the Gregorian calendar and the Earth revolution used today, and this error will be corrected by the following year. Every leap year, February will be one more day than normal, that is, 29 days in February.

The Gregorian calendar defines the following year as follows:

1. DM divided by 4 is not divisible, is an ordinary year (non-leap year)
2. years divided by 4 can be divisible but divided by 100 is not divisible, is a leap year
3. years divided by 100 can be divisible but divided by 400 is not divisible, is an ordinary year (non-leap year)

The above conditions can be simplified in one sentence. "Every four years is a leap year, every one hundred years is a non-leap year, and every four hundred years is a leap year."

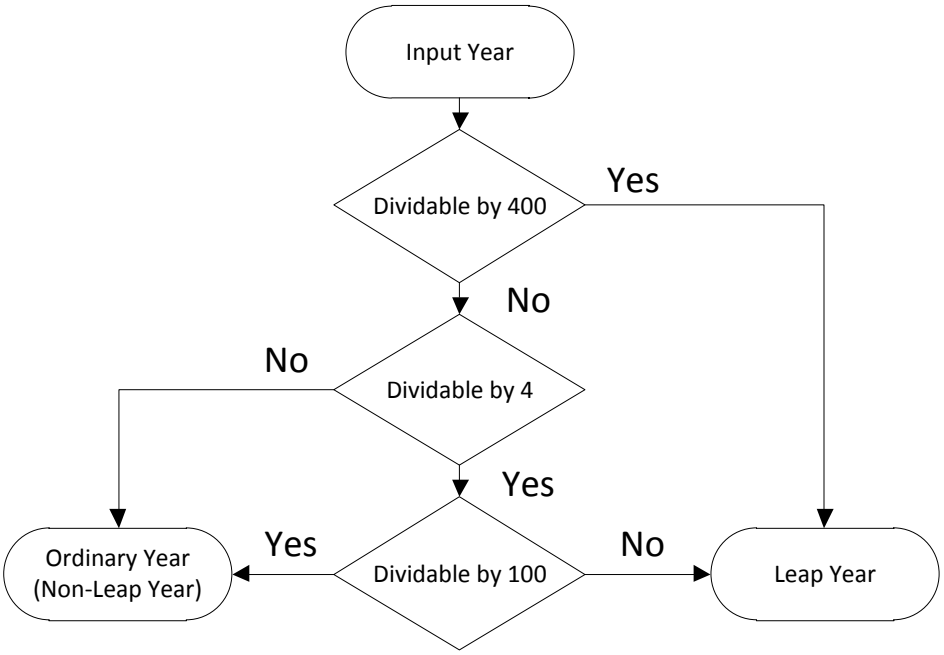


Figure 1-5 Leap Year Judgement Flow Chart

1.3.3 The Day of the Week Calculation Algorithm

In this Application Note, the Zeller formula derived from the German mathematician Christian Zeller is used in the calculation of the week. The formula can calculate the date of the week as the day of the week.

Scope of application: Any day after October 15, 1582

Note: The Roman Pope, I am the 13th, promulgating the new calendar (Gregorian calendar), changing the day after October 4, 1582 to October 15, 1582, so there will be a fault on the date before and after this day.

Zeller's congruence：

$$w = \left(y + \left[\frac{y}{4} \right] + \left[\frac{c}{4} \right] - 2c + \left[\frac{26(m+1)}{10} \right] + d - 1 \right) \bmod 7$$

w: the day of the week. The calculated values have the following correspondences: 0-Sunday; 1-Monday; 2-Tuesday; 3-Wednesday; 4-Thursday; 5-Friday; 6-Saturday

c: Year's first 2 digits, for example 2018, c value is 20

y: Year's last 2 digits, for example 2018, y value is 18

m: Month, In the Zeller formula, January and February are to be counted as 13 and 14 months of the previous year, and January 1, 2018, for example, to be counted as January 1, 2017.

d: Day

[ ]: Gaussian symbol, taking the largest integer not greater than the original number

Mod: The congruence symbol, divided by a number, takes the remaining number, using % in the programming language.

Since the result of the Zeller formula may be negative, the results can be corrected using the following methods to ensure that the range of results is limited to 0-Sunday ~ 6-Saturday:

$$\text{if } (w < 0) \;\; w += 7;$$

1.4 Reduce System Power Consumption

In order to save power, the system must be in the Power-down Mode. During the power-down mode, the user should turn off all unnecessary devices and external energy-consuming components.
In power-down mode, the internal low-speed RC oscillator circuit of 38.4 kHz does not stop working, so the timer that uses the pulse input at this time will continue to operate. The timer will wake up the MCU while sending the interrupt signal, and return the MCU to the normal mode. In the normal mode, when the time of the Real Time Clock is updated, the MCU should return to the power-down mode as soon as possible to reduce the power consumption of the system.

1.5 Real Time Clock Alarm

The user needs to set an alarm time and turn on the alarm function. When the time after the Real Time Clock is updated is equal to the time set by the alarm, the user can jump to the corresponding alarm program, and the user can edit the program of the alarm subprogram to meet the application requirements. The user can set and read the set value of the alarm through the instructions RTC_SetRTCAlarmTime() and RTC_GetRTCInfo() in the library.

1.6 Software Real Time Clock System Process

When implementing Real Time Clock through software, user needs to set the timer frequency, timer interrupt, timer wake-up function, etc. In addition, in order to reduce system power consumption, user need to turn on the power-down mode. After the MCU is initialized, the program flow can be divided into three states: chip sleep (power-down mode), interrupt service routine, and main program loop. The following is a brief description of these processes.

Initialize the hardware and software:

1. Initialize the timer, turn on the timer interrupt and timer wake-up function
2. Initialize Real Time Clock time, Real Time Clock alarm time
3. Select whether to enable the Real Time Clock alarm function.
4. Tell Real Time Clock that the timer generates several interrupts (interrupt frequency) in one second.

After entering the main program, it will repeat the cycle in the three states of chip sleep (power-down mode), interrupt service routine and main program loop:

1. MCU sleep (power-down mode): The MCU is maintained in a low-power state waiting for an interrupt event to occur
2. Interrupt service routine: When the timer interrupt is generated, the system jumps to the timer interrupt service routine. At this time, first increment the register value of the wake-up count by one, and then judge whether the value of the wake-up register is greater than or equal to The number of interrupts generated by the timer in one second. When the condition is established, it means that the time passes one second. At this time, the time update will be started, and the wake-up number register will be cleared to 0 after the update is completed. If the condition is not established, the representative time has not accumulated to 1 second, so the time will not be performed the update.

   If there is a time update, the user application will be executed after the update is completed. If there is an alarm function, it will continue to judge whether the alarm is generated and execute the corresponding alarm application. After the end, leave the interrupt service program and return to the main program. In the loop of the program; if there is no time update, it will leave the interrupt service program and return to the main program loop.
3. The main program loop: the user's program should enter the MCU sleep state after the execution

The relationship between the timer interrupt frequency and the number of wakeups: Interrupt frequency means that the timer generates several interrupt signals every second. The interrupt frequency of 2 Hz means that the chip is awake twice per second and the interrupt signal is sent twice. Each time the interrupt wakes up, the number of wakeups is accumulated. Therefore, when the

number of wakeups is greater than or equal to the number of wakeups per second of the timer, the time is already represented. After 1 second, user needs to update the time immediately.

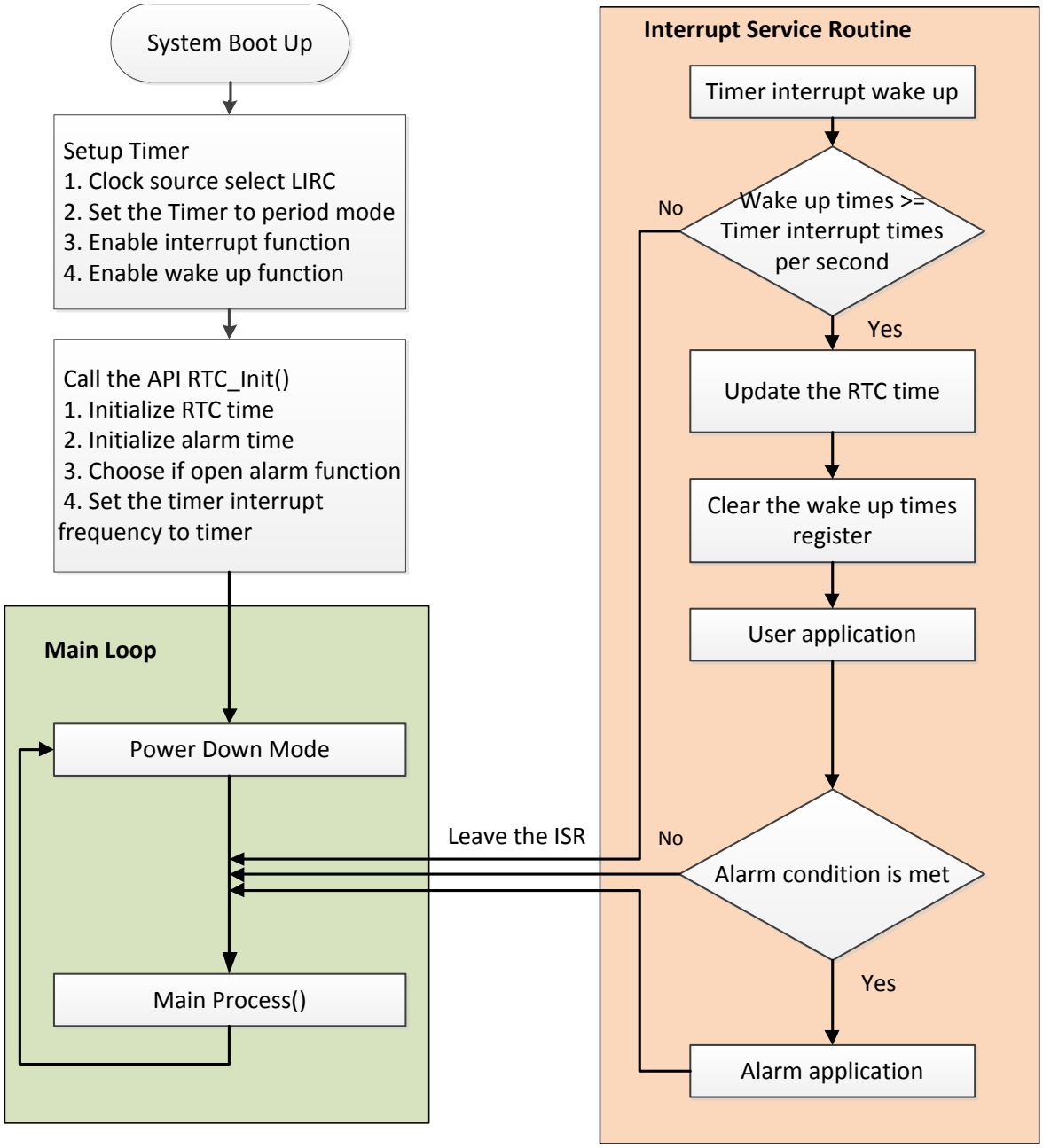


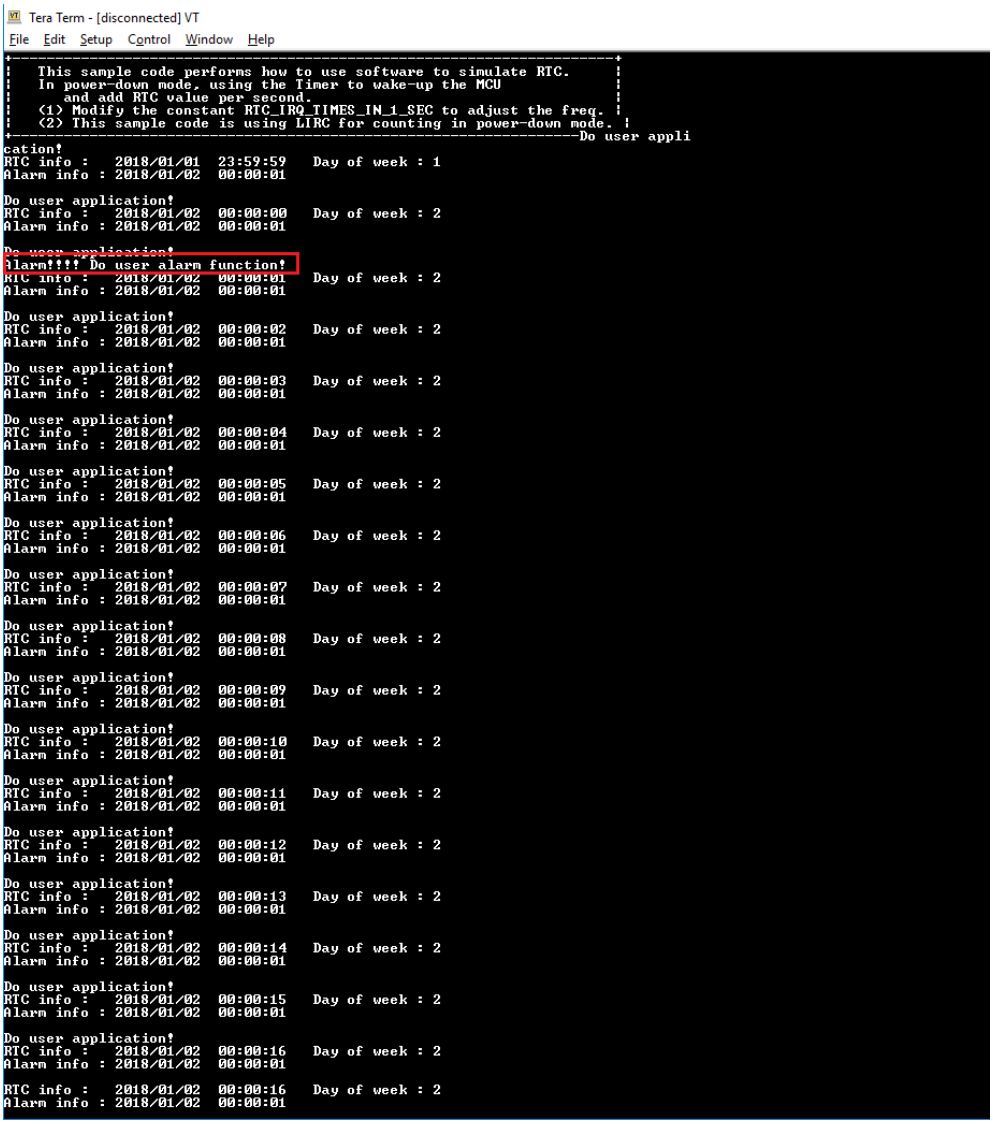
Figure 1-6 System Software Flow Chart

1.7 Example Code Execution Result

Real Time Clock Time: 2018/01/01 23:59:58

Real Time Clock Alarm Time: 2018/01/02 00:00:01

When the Real Time Clock alarm is triggered, the alarm subroutine is executed and the number of day increases as the Day of the week increases.



2 Code Description

2.1 main()

```
/*---------------------------------------------------------------------------*/
/*  Main Function                                                            */
/*---------------------------------------------------------------------------*/
int32_t main(void)
{
    /* Init System, IP clock and multi-function I/O. */
    SYS_Init();

    /* Init UART0 for printf */
    UART0_Init();

    /* Init SW RTC */
    SWRTC_Init();

    /* Init TIMER */
    TIMER_Init();

    /* Start Timer0 counting */
    TIMER_Start(TIMER0);

    printf("+-------------------------------------------------------------------+\n");
    printf("|   This sample code performs how to use software to simulate RTC.  |\n");
    printf("|   In power-down mode, using the Timer to wake-up the MCU          |\n");
    printf("|      and add RTC value per second.                                |\n");
    printf("|   (1) Modify the constant RTC_IRQ_TIMES_IN_1_SEC to adjust the freq. |\n");
    printf("|   (2) This sample code is using LIRC for counting in power-down mode. |\n");
    printf("+-------------------------------------------------------------------
+\n\n");

    /* Wait uart transfer message */
    while(!UART_IS_TX_EMPTY(UART0));

    while(1)
    {
        /* Unlock protected registers */
        SYS_UnlockReg();
        /* Make MCU power-down */
```

```
        CLK_PowerDown();
        /* lock protected registers */
        SYS_LockReg();

        #if 0 //Set new RTC Time
        TIMER_Stop(TIMER0);
        RTC_SetRTCTime(&g_sRTCInfo);
        TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, RTC_IRQ_TIMES_IN_1_SEC);
        #endif

        /* Dump the RTC information and alarm information. */
        RTC_GetRTCInfo(&g_sRTCInfo);
        RTC_GetAlarmInfo(&g_sAlarmInfo);
        printf("RTC info :   %04d/%02d/%02d  %02d:%02d:%02d   Day of week : %d \n",
g_sRTCInfo.date.u32year, g_sRTCInfo.date.u8month, g_sRTCInfo.date.u8day,
g_sRTCInfo.time.u8hour, g_sRTCInfo.time.u8minutes, g_sRTCInfo.time.u8seconds,
g_sRTCInfo.date.u8dayofweek);
        printf("Alarm info : %04d/%02d/%02d  %02d:%02d:%02d \n\n",
g_sAlarmInfo.date.u32year, g_sAlarmInfo.date.u8month, g_sAlarmInfo.date.u8day,
g_sAlarmInfo.time.u8hour, g_sAlarmInfo.time.u8minutes, g_sAlarmInfo.time.u8seconds);

        /* Wait uart transfer message */
        while(!UART_IS_TX_EMPTY(UART0));
    }
}
```

2.2 SWRTC_Init()

```
void SWRTC_Init(void)
{
    /* Set RTC time and RTC alarm time. */
    g_sRTCInfo.date.u32year = 2018;
    g_sRTCInfo.date.u8month = 1;
    g_sRTCInfo.date.u8day = 1;
    g_sRTCInfo.time.u8hour = 23;
    g_sRTCInfo.time.u8minutes = 59;
    g_sRTCInfo.time.u8seconds = 58;

    g_sAlarmInfo.date.u32year = 2018;
    g_sAlarmInfo.date.u8month = 1;
    g_sAlarmInfo.date.u8day = 2;
```

```
    g_sAlarmInfo.time.u8hour = 0;
    g_sAlarmInfo.time.u8minutes = 0;
    g_sAlarmInfo.time.u8seconds = 1;

    /* Init SW RTC Setting */
    RTC_Init();
    /* Set RTC time */
    RTC_SetRTCTime(&sRTCInfo);
    /* Set Alarm time */
    RTC_SetAlarmTime(&sAlarmInfo);
    /* Enable Alarm function */
    RTC_SetAlarmFunc(eRTC_ALARM_ENABLE);
    /* Set interrupt frequency for calculating if it is passing 1 second and update RTC
time. */
    RTC_SetIRQFreq(WAKEUPFREQ);
}
```

2.3 Timer Initialize

```
void TIMER_Init(void)
{
    /* Initial Timer0 to periodic mode with WakeUpFreq Hz.
    User can modify the marco "WakeUpFreq" to change wake-up frequency */
    TIMER_Open(TIMER0, TIMER_PERIODIC_MODE, RTC_IRQ_TIMES_IN_1_SEC);

    /* Enable timer wake up system */
    TIMER_EnableWakeup(TIMER0);

    /* Enable Timer0 interrupt */
    TIMER_EnableInt(TIMER0);
    NVIC_EnableIRQ(TMR0_IRQn);
}
```

2.4 User Application

```
void User_Application(void)
{
    printf("Do user application!\n");
}
```

2.5 Alarm Application

```
void User_AlarmFunction(void)
{
    printf("Alarm!!!! Do user alarm function!\n");
}
```

2.6 Interrupt of Timer

```
void TMR0_IRQHandler(void)
{
    if(TIMER_GetIntFlag(TIMER0)==TRUE)
    {
        /* Clear wake up flag */
        TIMER_ClearWakeupFlag(TIMER0);
        /* Clear interrupt flag */
        TIMER_ClearIntFlag(TIMER0);
        /* Import callback function and run RTC process */
        RTC_IRQ_Process(User_Application, User_AlarmFunction);
    }
}
```

3 Software and Hardware Environment

Software Environment

- BSP version
  - M031_Series_BSP_CMSIS_V3.01.000
- IDE version
  - Keil uVersion 5.26

Hardware Environment

- Circuit components
  - NuMaker-M031
  - Nu-Bridge
- Diagram

  For printing the debug message, user needs to connect two wire, the UART_RX0(PB.12), UART_TX0(PB.13) of M031 to Nu-Bridge. Then set the COM Port number and Baudrate in terminal tool. User can find the COM Port number in Device Manager of PC and it will display “NuBridge Virtual Com Port (COMX)”. The Baudrate should be 115200.

UART_RX0 (PB.12)	↔	PB5/ TX1 (Orange wire)
UART_TX0 (PB.13)	↔	PB4/RX1 (Brown wire)
GND	↔	GND (Black wire)
M031		Nu-Bridge

4 Directory Information

- EC_M031_SW_RTC_V1.00
  - Library — Sample code header and source files
    - CMSIS — Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
    - Device — CMSIS compliant device header file
    - StdDriver — All peripheral driver header and source files
  - SampleCode
    - ExampleCode — Source file of example code

5 How to Execute Example Code

1. This project supports Keil uVersion 5.26 and higher version
2. According to the chapter 4 Directory Information, enter the ExampleCode/SWRTC_V1.00/KEIL folder and double click the SW_RTC.uvproj
3. Enter the Keil compile mode window
    a. Build the project
    b. Download
    c. Start / Stop the debug mode
4. Enter debug mode
    a. Run

6 Revision History

Date	Revision	Description
Oct. 8, 2019	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.