Application Report
SLAA290A–January 2006–Revised January 2011

Using the Real-Time Clock Library

Lane Westlund *MSP430 Applications*

ABSTRACT

This document serves as an overview describing the use of the RTC library within an existing C or assembly project. The RTC library encapsulates commonly used routines for keeping track of time. These functions are written in assembly to be optimized for the MSP430, but can be called from any C program that includes the RTC.h header file. To be easily displayable on an LCD, all variables are encoded in BCD. All variables, except the year, use one byte. The year variable uses two bytes.

Related software is available for download from http://www.ti.com/lit/zip/slaa290.

Contents

1 Introduction

It is often the case that, along side its main purpose, an MSP430 application also keeps track of and displays the current time. The purpose of this library is to encapsulate the features commonly associated with keeping track of time and to make it easy to display on an LCD screen. This encapsulation means the user only needs to make simple function calls with well defined parameters and not worry about all the computational steps needed to ensure correct time keeping.

The RTC library includes all files necessary to setup a periodic interrupt, and keep real-time. It includes functions to handle the periodic-time interrupt and increment all time-keeping variables. Time-keeping variables include seconds, minutes, hours, days of the week, months, years, and even leap-years.

The RTC is structured to use one-second time intervals by calling the incrementSeconds function. Inside this function, the variable TI_second is updated. When it increments to a count of 60, it is set to zero and the TI_minute variable is incremented. When TI_minute reaches 60, it is zeroed, and the TI_hour variable is incremented, and so on. All clock and calendar variables update automatically with the one function call to incrementSeconds.

There are two files included in the library for time-keeping: RTC.s43 and RTC_Calendar.s43. The first is for use only when time is desired but not calendar functions like day-of-the-week, month, etc. The second implements the full calendar including leap-year adjustments. To implement an RTC in an application, one of these files, but not both, is simply included in the project and the appropriate function is called at one-second intervals.

The library also includes functions to setup the required one-second interval. Several timers can be used to generate a one-second interrupt, including Timer_A, the Watchdog timer, and the Basic Timer. Functions to setup each for a one-second interrupt are included in the files named RTC_TA.s43, RTC_WDT.s43, and RTC_BT.s43.

Also included in this application report are a series of code examples. The purpose of these code examples is to illustrate the setup needed to use the library in various ways.

2 Usage From C

A typical clock type application that uses the RTC library would look like this:

```
#include "RTC.h"
void main ( void )
{
   WDTCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
   setTime( 0x12, 0, 0, 0);     // initialize time to 12:00:00 AM
   P1DIR |= 0x01;               // Set P1.0 to output direction
   CCR0 = 32768-1;
   TACTL = TASSEL_1+MC_1;       // ACLK, upmode
   CCTL0 |= CCIE;               // enable CCRO interrupt
   _EINT();
   while( 1 )
   {
      LPM3;                     // enter LPM3, clock will be updated
      P1OUT ^= 0x01;            // do any other needed items in loop
          _NOP();                   // set breakpoint here to see 1 second int.
   }
}
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
   incrementSeconds();
   LPM3_EXIT;
}
```

The file RTC.h must be included to gain access to both the variables and functions when using the RTC library from a C program.

In the example above, the writer chose to implement their own Timer_A Interrupt Service Routine (ISR), and set up Timer_A for a 1 second interrupt. If they did not want to write their own ISR and were using a 32.768-kHz crystal, they could have used the RTC_TA library:

```
#include "RTC.h"
#include "RTC_TA.h"

void main ( void )
{
   WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
   setTime( 0x12, 0, 0, 0);    // initialize time to 12:00:00 AM
   TA_1sec_wake();             // configure TA for 1 second update
   LPM3;                       // enter LPM3, clock will be updated
}
```

By calling the function TA_1sec_wake(), Timer_A is set up to run in continuous mode with 32768 in CCR1. For this reason a 32.768-kHz crystal must be supplied on LFXT1. The ISR supplied in RTC_TA simply calls the function incrementSeconds and returns. After exiting the ISR, the device returns to its previous state. This means it returns back into LPM3 or any other low-power mode it was previously in. To return from the ISR into active mode (to complete a loop) the user must write a custom ISR as shown previously.

In addition to incrementSeconds, the library also supports several other increment functions such as incrementMinutes, incrementHours, incrementDays, and incrementYears. These functions are provided so that they may be called in situations where the time is to be set through user interaction (for example, by pushing a button).

3 Usage From Assembly

The library can also be used from assembly. In this case, no imports are required, but the desired functions must be listed using the **EXTERN** keyword. Care must also be taken to make sure function calls are in immediate mode.

```
#include "msp430x14x.h"
            EXTERN  TA_1sec_wake
;-------------------------------------------------------------------------------
            ORG     01100h                  ; Program Start
;-------------------------------------------------------------------------------
RESET       mov.w   #0A00h,SP               ; Initialize 'x1x9 stack pointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
            call    #TA_1sec_wake
            bis.w   #CPUOFF,SR              ; CPU off, interrupts enabled
```

4 Leap Years

When using the calendar functions, leap years are correctly calculated except for all years that are evenly divisible by 400. As of this writing, the next year that will meet this criterion is 2400 A.D., which was deemed as an acceptable trade-off to reduce code size.

Leap years are calculated any time an increment or setDate function is called. Should the year variable be changed outside of the RTC_Calendar library, the function testLeap should be called to set the correct number of days in February for that year.

For convenience, the macro LEAP_YEAR has been included in RTC_Calendar.h. After the current years leap year calculation has been done, C programs can put in the LEAP_YEAR conditional to test whether the current year is a leap year or not. LEAP_YEAR is defined as (FebDays == 0x29)

5 Daylight Savings

Both EU and US daylight savings is accounted for. By setting the dayLightZone variable to either US_DAYLIGHT_SAVINGS, EU_DAYLIGHT_SAVINGS, or NO_DAYLIGHT_SAVINGS, the library automatically adjust the clock on the following times:

	Begins (hour +1)	Ends (hour -1)
US	Second Sunday in March at 2 am	First Sunday in November at 2 am
EU	Last Sunday in March at 1 am	Last Sunday in October at 1 am

NOTE: The variable TI_dayLightSavings is meant to be used internally for the library. It is designed to prevent the library from being stuck in an endless loop, rolling back the hour each time 2 or 1 am is reached in the last Sunday in October. It is set when daylight savings begins, and cleared the first time 2 or 1 am is reached on the last Sunday in October. **It is always set to 1 when the setDate() routine is called, regardless of the date given**. If the code is calling setDate() to a static date, it is possible to set this variable to a static value in the code, which is updated correctly

5.1 Cycle Count

Function	Cycle Count	
	RTC	RTC_CALENDAR
incrementSeconds	14	14
incrementMinutes	14	14
incrementHours	22	35(1)
get24Hour	23	23
incrementDays	NA	37(1)
incrementMonths	NA	14(1)
incrementYears	NA	36(1)
setDate	NA	681(2)

(1) When incrementing from 1:00 PM April 29, 2005. This represents the most common cycle count. Daylight savings days and incrementing into a leap year require more cycles.
(2) When setting date to April 29, 2005.

5.2 Code Size

	Size (bytes)
RTC	126
RTC_Calendar	714

NOTE: The default state of the variables is 12:00:00 AM (Tuesday, January 1, 2004 when the Calendar library is used). In IAR, these variables are initially set to this default state when any global C variables are used. If this is not the case, these variables must be explicitly initialized in the application code using the setTime(h,m,s,pm) and setDate(y,m,d) functions.

6 Example Includes

The following is a list of hypothetical example projects, which describe which code files should be included for each one.

1. RTC application with Calendar (non-library based 1 second interrupt):
   - RTC_Calendar.s43
   - RTC_Calendar.h
2. RTC application with Calendar using Timer_A from Library
   - RTC_Calendar.s43
   - RTC_Calendar.h
   - RTC_TA.s43
   - RTC_TA.h
3. RTC application without Calendar (non-library based 1 second interrupt):
   - RTC.s43
   - RTC.h
4. RTC application without Calendar using Timer_A from Library
   - RTC.s43
   - RTC.h
   - RTC_TA.s43
   - RTC_TA.h

6.1 Included Files

In the zip file accompanying this application report (http://www.ti.com/lit/zip/slaa290), there are two directories: source_CCS and source_IAR. The files in these directories are functionally equivalent, and only contain minor changes to allow for compiling using CCS or IAR respectively.

6.1.1 RTC.s43

This file includes all variables and functions needed for a "time only" RTC. Measurements beyond hours of the day are not accounted for. If greater functionality is desired, RTC_Calendar should be included and not this file

6.1.2 RTC_Calendar.s43

This file includes all variables and functions needed for a full RTC. Calendar based functions such as day, month, year, day of week, and daylight savings are included.

6.1.3 RTC_BT.s43

This file includes the function BT_1sec_wake(). Calling this function initializes the Basic Timer to wake every 1 second and call the function incrementSeconds().

NOTE: An external 32-kHz crystal on LFXT1 is assumed.

6.1.4 RTC_TA.s43

This file includes the function TA_1sec_wake(). Calling this function initializes Timer_A to wake up every one second and call the function incrementSeconds(). CCR1 is used and loaded with 32768. The timer runs in continuous mode so each during each interrupt an additional 32768 is added to CCR1. It is possible to use CCR0 for additional interrupt timing, however the TAR should not be modified and the mode should not be changed, to avoid mistiming.

NOTE: An external 32-kHz crystal on LFXT1 is assumed.

6.1.5 RTC_WDT.s43

This file includes the function WDT_1sec_wake(). Calling this function initializes the Watchdog Timer to wake every 1 second and call the function incrementSeconds().

NOTE: An external 32-kHz crystal on LFXT1 is assumed.

6.1.6 Test_Suite.c

This file is included to illustrate all tests used on the RTC library. In this file, all use cases for the Library are tested

6.2 Variable Description

All RTC library variables begin with the TI_ prefix to avoid collision with any other variable names used in an end application.

6.2.1 TI-second

The current second count: 0x00 to 0x59

6.2.2 TI_minute

The current minute count: 0x00 to 0x59

6.2.3 TI_hour

The current hour count: 0x00 to 0x12

6.2.4 TI_day

The current day of the month: 0x01 to 0x31

6.2.5 TI_dayOfWeek

The current day of the week: 0 to 6 (integer), Sunday == 0. See headers for defined day values.

6.2.6 TI_month

The current month of the year 0x00 to 0x11, January == 0. See headers for define month values.

6.2.7 TI_year

The current year 0x0000 to 0x2399. Leap years are not computed properly for 0x2400 (see earlier note).

6.2.8 TI_PM

The AM/PM flag. AM = 0, PM = 1

6.2.9 TI_FebDays

The number of days in this current year's February. Either 0x28 or 0x29.

6.2.10 TI_DayLightZone

The current daylight savings time method being used. Can be either "no daylight savings", "US daylight savings", or "EU daylight savings" (use header file definitions).

6.2.11 TI_DayLightSavings

Used to test whether the daylight savings hour has already been rolled back to avoid an infinite loop. See the earlier note for more information on this variable and its usage.

6.3 Function Description

6.3.1 incrementSeconds(void)

This function adds one to the BCD variable second. If second increases to 0x60, incrementMinutes() is called and second reverts to 0x00.

6.3.2 incrementMinutes(void)

This function adds one to the BCD variable minute. If minute increases to 0x60, incrementHours() is called and minute reverts to 0x00.

6.3.3 incrementHours(void)

This function adds one to the BCD variable hour. If hour increases to 0x12, the variable PM is toggled. If hour increases to 0x13 it reverts to 1. If hour increases to 0x13 and PM is 0x01, incrementDays() is called.

6.3.4 setTime(char hour, char minute, char second, char pm)

This function sets the current time to the values specified in the values passed in. This function is strictly a macro function so the parameters must be passed in BCD format using 12 hour notation.

6.3.5 Get24Hour()

This function returns a char containing the current hour in BCD 24 hour format. For example 12:00 AM returns as 0x00 and 12:00 PM returns as 0x12, 11:00 PM returns as 0x23.

6.3.6 incrementDays(void) – Calendar Only

This function adds one to the BCD variable day. If day is greater than the number of days in the current month, it reverts back to 0x01, and incrementMonths() is called.

6.3.7 incrementMonths(void) – Calendar Only

This function adds one to the BCD variable month. If month increases to 0x12, incrementYears() is called and month reverts to 0x00.

NOTE: This function does not correctly set the dayOfWeek variable when called from a source external to the library.

6.3.8 incrementYears(void) – Calendar Only

This function adds one to the BCD variable year.

NOTE: This function does not correctly set the dayOfWeek variable when called from a source external to the library.

6.3.9 testLeap(void) – Calendar Only

This function correctly sets the FebDays variable based on the current year variable. It computes whether February has 28 or 29 days in the current year. This function is typically only called by setDate(), but could be called after anytime the TI_year variable is changed.

6.3.10 setDate(int year, char month, char day) – Calendar Only

This function sets the current date to the values specified in the variables year, month, and day. It computes whether the year is a leap year and correctly set the FebDays variable. It also correctly sets the dayOfWeek variable. The variables should be passed in decimal (not BCD) format. Months should be passed with January == 1 and day starting at 1 as well.

Revision History

Changes from Original (January 2006) to A Revision	Page
• Editorial changes throughout	1
• Changed CCE references to CCS	5

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265