EFM®32

... the world's most energy friendly microcontrollers

Tickless Calendar with Temperature Compensation

AN0006 - Application Note

This application note describes how a tickless calendar based on the Real Time Counter for the EFM32 can be implemented in software. The calendar uses the standard C library time.h where an implementation of the time() function calculates the calendar time based on the RTC counter value.

Software examples for the EFM32TG_STK3300, EFM32G_STK3700 and EFM32_Gxxx_STK Starter Kits are included.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

1 Tickless Calendar

1.1 General

Many microcontroller applications need to keep track of time and date with a minimum current consumption. This application note describes a possible software implementation of a low current tickless calendar on the EFM32.

2 Software

2.1 C Date and Time

Standard C contains date and time library functions that are defined in *time.h*.

These functions are based on Unix time, which is defined as a count of the number of seconds passed since the Unix epoch (midnight UTC on January 1, 1970). More on Wikipedia [http://en.wikipedia.org/wiki/Unix_time].

A key component in this library is the **time()** function that returns the system time as Unix time. The library also contains functions to convert Unix time to human readable strings, and to convert a Unix time to a calendar time and date structure and the other way around.

For embedded systems, some functions in the C standard library are not implemented. Usually, the *time.h* functions **time()** and **clock()** are among the functions that are declared, but not implemented.

In this application note, *clock.c* contains an almost *time.h* compliant implementation of the **time()** function, using the EFM32 RTC as a basis for a tickless calendar.

More information on the C standard library functions can be found on Wikipedia [http://en.wikipedia.org/wiki/Time.h] or the web site of ISO/IEC JTC1/SC22/WG14-C [http://www.open-std.org/JTC1/SC22/WG14/], the standardization group on the C programming language, where a link to the latest C specification can be found.

2.2 Clock.c

Clock.c contains the functions that are needed for the tickless calendar.

A prerequisite before using the functionality of *clock.c* , is that the RTC must be configured and running with the overflow interrupt enabled. The RTC must be clocked by the LFXO, or another high precision external clock, to give the required precision for time keeping.

Clock.c must be told how fast the RTC clock is running. This is done when initializing the software calendar.

To initialize the software calendar, **clockInit()** is used. When the calendar is initialized, a *Clock_Init_Typedef* structure that contains the RTC frequency and start time must be given.

Clock.c also keeps track of how many times the RTC has overflowed since the start time. This is required for time keeping to be correct when the RTC overflows. To keep track of the overflow count, the user application must call **clockOverflow()** every time an overflow occurs. Preferably by enabling the RTC overflow interrupt and calling the overflow function in the RTC interrupt handler.

Clock.c contains the following functions and constants:

2.2.1 time()

```
time_t time( time_t * timer )
```

A *time.h* compliant implementation of the standard **time()** function for the EFM32. The RTC is used to keep track of calendar time.

Before **time()** can be used, the clock must be initialized with the **clockInit()** function.

The current calendar time is evaluated by the following expression:

Current calendar time

$$t = t_0 + N_{of} * T_{of} + (N_{rtc} / f_{rtc}) \tag{2.1}$$

where t is the current Unix time, t_0 is the initial Unix time, N_{of} is the overflow counter value, T_{of} is the overflow interval in seconds, N_{rtc} is the RTC counter value, and f_{rtc} is the RTC count frequency in Hz.

Parameters: timer - NULL, or a pointer to a variable to store current time.

Returns: system time

2.2.2 clockInit()

```
void clockInit( struct tm * timeptr )
```

Initializes the system clock. The input time and date structure is converted to Unix time and set as the initial Unix time, t_0. The overflow counter is reset.

Parameters: timeptr - a calendar time and date structure that contains the initial Unix time.

2.2.3 clockSetStartCalendar()

```
void clockSetStartCalendar( struct tm * timeptr )
```

Sets the initial Unix time of the system clock, t_0.

Parameters: timeptr - a calendar time and date structure that contains the initial Unix time.

2.2.4 clockSetStartTime()

```
void clockSetStartTime( time_t offset )
```

Sets the initial Unix time of the system clock, t_0.

Parameters: offset - Initial Unix time.

2.2.5 clockGetStartTime()

```
time_t clockGetStartTime( void )
```

Returns the initial Unix time of the system clock, t_0.

Returns: the initial Unix time of the system clock.

2.2.6 clockSetOverflowCounter()

```
void clockSetOverflowCounter( uint32_t of )
```

Sets the RTC overflow counter, N_{of}. A count of how many times the RTC has overflowed since it was started.

Parameters: of - RTC overflow counter.

2.2.7 clockGetOverflowCounter()

```
uint32_t clockGetOverflowCounter( void )
```

Returns the RTC overflow counter, N_{of}. A count of how many times the RTC has overflowd since it was started.

Returns: the RTC overflow counter.

2.2.8 clockOverflow()

```
uint32_t clockOverflow( void )
```

Increments the RTC overflow counter, N_{of}. Must be called every time the RTC overflows. This is most conveniently implemented by enabling the RTC overflow interrupt and calling this function in the interrupt handler.

Returns: the RTC overflow counter after the increment.

2.3 Clock_tc.c

Clock_tc.c is a temperature compensated version of clock.c. It basically contains the same functionality as clock.c, but with added functions for temperature compensation of the RTC.

To do temperature compensation, the *clockDoTemperatureCompensation* function must be called at a periodic interval. This interval must be input to *clockInit()*.

2.4 Calendar_ui.c

Calendar_ui.c contains the functions that handles the user interface.

Functionality to adjust the calendar is included in this file. The calendar is adjusted by modifying the initial Unix time in clock.c. I.e. when the user wants to set the clock one hour forward, one hour is added to the initial Unix time.

3 Temperature Compensation

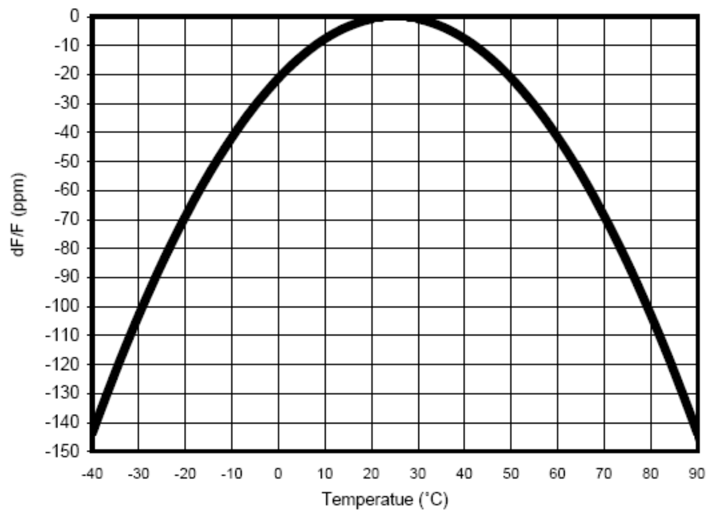
3.1 Crystal Frequency vs Temperature

Low frequency crystals (e.g. 32.768 kHz) are commonly cut such that the frequency over temperature is a parabolic curve centered at 25°C. The behaviour can be modelled by the equation below in which ß is the frequency coefficient with a typical value of -0.04 [ppm/°C²]. A typical curve is shown in Figure 3.1 (p. 6) .

Tuning Fork Crystal Frequency

$$f = f_0[1 + ß(T-T_0)^2] \tag{3.1}$$

Figure 3.1. Typical Tuning Fork Crystal Frequency vs Temperature



3.2 Compensation Approach

In order to compensate the frequency deviation, the temperature is measured regularly and the deviation from the nominal temperature (typically 25°C) is calculated. A look-up table containing data similar to the curve in Figure 3.1 (p. 6) is accessed to find the frequency deviation.

Two observations are made; firstly the curve is symmetrical and hence only one side of the look-up table is needed. Secondly, it can be seen that the frequency will always be less than or equal to f_0. The time error due to frequency error is accumulated and when this above the equivalent of 1 second, the clock is incremented with an extra second. Each time an extra second is added, the equivalent of 1 second is subtracted from the accumulated value.

Note

The temperature is measured using the internal temperature sensor so the crystal might not be at the same exact temperature.

4 Software examples

4.1 Tickless System Clock

The attached software example consists of a clock application that demonstrates how to use the EFM32 RTC as a basis for a tickless software calendar.

In the example, the RTC is used for two separate purposes. At a regular interval (by default 1 second), the RTC comparator 0 generates an interrupt that triggers an update of the LCD. Separate from display update, the RTC is also used to keep track of system time. A time() function is implemented. This function calculates the system time based on the RTC counter value, and outputs it as a Unix time number. The time number is the number of seconds passed since the Unix epoc (midnight UTC on January 1, 1970). C Standard library functions defined in time.h are used to convert unix time to a human readable calendar format.

Energy Micro library functions for the STKs are used to set up the LCD and to display the clock.

GPIO interrupts on the STK pushbuttons are used to adjust the clock.

4.2 Temperature Compensated Tickless System Clock

A temperature compensated version of the Tickless System Clock is included. This example uses the internal temperature sensor to measure temperature, and adjusts for temperature induced frequency devitation based on information from crystal data sheet. The example code uses the data sheet parameters for the 32.768 kHz crystal used on the EFM32GG-STK3700.

4.3 Instructions

The clock starts at January 1 2012 12:00:00. Due to limited display size, only the time of day is shown.

Press PB0 and PB1 to set the clock. PB0 adds 1 hour while PB1 adds 1 minute.

5 Revision History

5.1 Revision 2.05

2013-11-15

Added software example with temperature compensation

Minor bug-fix in clock.c

5.2 Revision 2.04

2013-10-14

New cover layout

5.3 Revision 2.03

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

5.4 Revision 2.02

2012-11-12

Added software projects for the Gecko and the Giant Gecko STKs.

Adapted software projects to new kit-driver and bsp structure.

5.5 Revision 2.01

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

5.6 Revision 2.00

2012-03-26

Rewrite of the calendar function. Calendar is now tickless and is based on an implementation of the time() function where the RTC is used to keep track of system time.

Temperature compensation is no longer a part of the example.

Example project is now written for the Tiny Gecko Starter Kit, EFM32TG_STK3300.

5.7 Revision 1.12

2011-10-21

Updated IDE project paths with new kits directory.

5.8 Revision 1.11

2011-05-18

Updated project to align with new bsp version.

5.9 Revision 1.10

2011-04-24

Updated software project and document with a calendar implementation using time.h libary.

5.10 Revision 1.03

2010-12-30

Changed SegmentLCD_Init() position inside calendarInit().

5.11 Revision 1.02

2010-11-16

Changed TEMP_GRAD_CODES constant in code example to latest datasheet value.

Added (double) cast to integers in temperature calculation equation.

Changed example folder structure, removed build and src folders.

Added chip-init function.

Changed software to use the segmentlcd.c functions, lcdcontroller.c is deprecated

5.12 Revision 1.00

2010-09-20

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

A.2 Trademark Information

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

1. Tickless Calendar 2
    1.1. General 2
2. Software 3
    2.1. C Date and Time 3
    2.2. Clock.c 3
    2.3. Clock_tc.c 5
    2.4. Calendar_ui.c 5
3. Temperature Compensation 6
    3.1. Crystal Frequency vs Temperature 6
    3.2. Compensation Approach 6
4. Software examples 7
    4.1. Tickless System Clock 7
    4.2. Temperature Compensated Tickless System Clock 7
    4.3. Instructions 7
5. Revision History 8
    5.1. Revision 2.05 8
    5.2. Revision 2.04 8
    5.3. Revision 2.03 8
    5.4. Revision 2.02 8
    5.5. Revision 2.01 8
    5.6. Revision 2.00 8
    5.7. Revision 1.12 8
    5.8. Revision 1.11 8
    5.9. Revision 1.10 9
    5.10. Revision 1.03 9
    5.11. Revision 1.02 9
    5.12. Revision 1.00 9
A. Disclaimer and Trademarks 10
    A.1. Disclaimer 10
    A.2. Trademark Information 10
B. Contact Information 11
    B.1. 11

List of Figures

3.1. Typical Tuning Fork Crystal Frequency vs Temperature 6

List of Equations

2.1. Current calendar time 4
3.1. Tuning Fork Crystal Frequency 6

silabs.com

ZERO
ARM Cortex-M0+

TINY
ARM Cortex-M3

GECKO
ARM Cortex-M3

LEOPARD
ARM Cortex-M3

GIANT
ARM Cortex-M3

WONDER
ARM Cortex-M4