

EFM[®]32

... the world's most energy friendly microcontrollers

Tickless Calendar

AN0006 - Application Note

A decorative graphic on the right side of the page consists of several concentric circles. The outermost circle is light green, followed by a medium green ring, then a blue ring, and finally a dark blue inner circle. The numbers 0, 1, 2, 3, and 4 are arranged in a horizontal line across the middle of these rings, with 0 in the light green ring, 1 in the medium green ring, 2 in the blue ring, 3 in the dark blue ring, and 4 in the innermost dark blue circle.

Abstract

This application note describes how a tickless calendar based on the Real Time Counter for the EFM32G 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.

A software example for the EFM32TG_STK3300 Tiny Gecko Starter Kit is 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 while 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 by defining the constant **COUNTS_PER_SEC**.

To initialize the software calendar, **clockInit()** is used. When the calendar is initialized, a calendar time and date structure that contains the RTC 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 the time keeping to be correct also after the RTC overflows. To keep track of the overflow count, the user application must call the **clockOverflow()** function 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 and **COUNTS_PER_SEC** must be changed to reflect the RTC count frequency. As the clock functions rely on a running RTC, the RTC must also be enabled and configured. The RTC overflow interrupt must be enabled, and the interrupt handler must include a call to **clockOverflow()**.

The current calendar time is evaluated by the following expression:

Current calendar time

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

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 overflowed 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.

2.2.9 COUNTS_PER_SEC

```
#define COUNTS_PER_SEC 32768
```

Number of RTC counts per second, f_{rtc} . This constant must reflect the RTC count frequency.

2.2.10 OVERFLOW_INTERVAL

```
#define OVERFLOW_INTERVAL ((0x00FFFFFF+1) / COUNTS_PER_SEC)  
#define OVERFLOW_INTERVAL_R ((0x00FFFFFF+1) % COUNTS_PER_SEC)
```

The number of seconds between each RTC overflow, T_{of} . This constant is calculated from `COUNTS_PER_SEC` and the RTC counter width.

The overflow interval also includes a remainder that is used in case the number of seconds per RTC overflow is not an integer.

2.3 ClockApp.c

`ClockApp.c` contains the functions that handles the user interface, such as display control and interrupt handling. The application uses three interrupts: One GPIO interrupt that is triggered by the STK pushbuttons that is used to adjust the clock, a RTC compare match interrupt is used to periodically wake the MCU to display the clock every second and the RTC overflow interrupt which use is described in the previous section.

The RTC overflow interrupt is the only interrupt that is really needed for the tickless calendar. The other two interrupts are used for the application's 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 Software example

3.1 Tickless Calendar 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 epoch (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 TG STK are used to set up the LCD and to display the clock.

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

3.2 Instructions

Compile the example, connect the Tiny Gecko Starter Kit with USB and download the program to the kit.

Press the Reset button on the kit. The clock will now start at 12:00:00.

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

4 Revision History

4.1 Revision 2.01

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

4.2 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.

4.3 Revision 1.12

2011-10-21

Updated IDE project paths with new kits directory.

4.4 Revision 1.11

2011-05-18

Updated project to align with new bsp version.

4.5 Revision 1.10

2011-04-24

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

4.6 Revision 1.03

2010-12-30

Changed SegmentLCD_Init() position inside calendarInit().

4.7 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

4.8 Revision 1.00

2010-09-20

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Energy Micro AS intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Energy Micro products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Energy Micro reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Energy Micro shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Energy Micro. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Energy Micro products are generally not intended for military applications. Energy Micro products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

Energy Micro, EFM32, EFR, logo and combinations thereof, and others are the registered trademarks or trademarks of Energy Micro AS. ARM, CORTEX, THUMB are the registered trademarks of ARM Limited. Other terms and product names may be trademarks of others.

B Contact Information

B.1 Energy Micro Corporate Headquarters

Postal Address	Visitor Address	Technical Support
Energy Micro AS P.O. Box 4633 Nydalen N-0405 Oslo NORWAY	Energy Micro AS Sandakerveien 118 N-0484 Oslo NORWAY	support.energymicro.com Phone: +47 40 10 03 01

www.energymicro.com

Phone: +47 23 00 98 00

Fax: + 47 23 00 98 01

B.2 Global Contacts

Visit **www.energymicro.com** for information on global distributors and representatives or contact **sales@energymicro.com** for additional information.

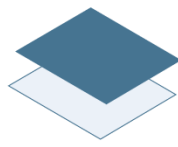
Americas	Europe, Middle East and Africa	Asia and Pacific
www.energymicro.com/americas	www.energymicro.com/emea	www.energymicro.com/asia

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. ClockApp.c 5
- 3. Software example 6
 - 3.1. Tickless Calendar Clock 6
 - 3.2. Instructions 6
- 4. Revision History 7
 - 4.1. Revision 2.01 7
 - 4.2. Revision 2.00 7
 - 4.3. Revision 1.12 7
 - 4.4. Revision 1.11 7
 - 4.5. Revision 1.10 7
 - 4.6. Revision 1.03 7
 - 4.7. Revision 1.02 7
 - 4.8. Revision 1.00 8
- A. Disclaimer and Trademarks 9
 - A.1. Disclaimer 9
 - A.2. Trademark Information 9
- B. Contact Information 10
 - B.1. Energy Micro Corporate Headquarters 10
 - B.2. Global Contacts 10

List of Equations

2.1. Current calendar time 4



ENERGY[®]
micro

*Energy Micro AS
Sandakerveien 118
P.O. Box 4633 Nydalen
N-0405 Oslo
Norway*

www.energymicro.com