EFM32

... the world's most energy friendly microcontrollers

Interrupts and Energy Modes

TM0008 - Hands-on Exercise

Introduction

In this exercise we will extend the basic EFM32 programming concepts to include interrupts and energy modes.

This exercise includes:

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

ARM ZERO TINY GECKO LEOPARD GIANT WONDER

SILICON LABS

1 Introduction

1.1 How to use this document

Along with this document are several incomplete source code files, where the reader is required to fill in small pieces of code through out the exercises. If you get stuck, a solution file is provided for each exercise.

Projects for each of the following IDEs are provided:

- Simplicity IDE
- IAR Embedded Workbench
- Keil µVision
- Rowley CrossWorks

Since the projects are slightly different for the various kits, make sure that you open the project that is prefixed with the name of the kit you are using.

1.2 Prerequisites

The exercises require that you have an EFM32 kit at hand. Before you start working on this tutorial you should make sure you have installed one of the IDEs listed above. You also need Simplicity Studio installed. It can be downloaded here:

http://www.silabs.com/products/mcu/pages/simplicity-studio.aspx

In Simplicity Studio, make sure you have selected support for the EFM32 part you are using.

2 GPIO Interrupt

In this exercise you will extend the code from the previous exercises by adding interrupts

Task 1: Enable GPIO Interrupt

Open the emlib documentation and find the documentation for `GPIO_IntConfig()`. Use this function to set up a falling edge interrupt on the push button pin. This causes the GPIO peripheral to generate an IRQ request when the button is pressed.

To trigger the interrupt handler the corresponding interrupt must be enabled in the Nested Vectored Interrupt Controller (NVIC). You can find a list of functions for manipulating the NVIC in either the **Cortex-M3 Reference Manual** or **API Documentation -> ARM CMSIS Documentation** from Simplicity Studio. Find the function that enables an interrupt in NVIC.

The NVIC function takes one parameter of the enum type `IRQn_Type`. The values for this type is defined in the EFM32 header file for each device and follows a standard naming convention: the name of the IRQ request postfixed with `_IRQn`. E.g. the parameter for the GPIO_EVEN interrupt is called `GPIO_EVEN_IRQn`. Enable either the GPIO_EVEN or GPIO_ODD interrupt depending on which pin is used for the button.

Task 2: Implement Interrupt Handler

In the source code there is already an empty interrupt handler which you should complete. The interrupt handler should check which flags are set, and if the button has been pressed toggle the LED. You also have to clear the interrupt flags inside the interrupt handler. If not the handler will be triggered again and the program will be stuck in an endless loop.

Note

The empty handler is for the GPIO_ODD interrupt. If you are using an even-numbered pin for the button change this to GPIO_EVEN.

Task 3: Enter EM3 Between Events

Until now the application has always been in EM0 (active mode). To save power the application can enter a sleep mode when waiting for the next button press. Since the only event we are waiting for is an asynchronous event from GPIO it is safe to enter EM3. Change the endless loop to always enter EM3.

Task 4: Measure Power Consumption

Open the energyAware Profiler and measure the power consumption of your application. You should see a very noticable contribution from the LED (when it is on). Comment out what you did in Task 3 to see the difference when staying in active mode vs EM3.

Note

When the MCU is in debug mode it is prevented from going below EM1. To get an accurate representation the current consumption you have to reset the board after a debug session.

3 RTC Delay

In this exercise we will use the RTC to periodically toggle an LED.

Task 1: Include emlib RTC files

The project for this exercise is not set up to include the emlib files for RTC so the first thing we have to do is just that. In emlib there is a header and source file for each peripheral. The files we need for the RTC are: em_rtc.h and em_rtc.c.

In the top of 4_rtc_delay.c add the following line:

```
#include "em_rtc.h"
```

Include em_rtc.c as a source file for the project. This file is found in

<ss_install_dir>\emlib\src

Note

If you are unsure where <ss_install_dir> is you can click File -> Browse Installed Files... in Simplicity Studio, which will take you to this directory. The default path on Windows 7 is: C:\Users\<your_username>\AppData\Roaming\energymicro.

Task 1: Enable LFXO

The RTC is part of the low frequency domain, which uses different oscillators than the TIMER and CPU which we have used until now. In this exercise we will use the external low frequency crystal oscillator (LFXO) for the low frequency domain.

In the CMU section of the emlib documentation you should be find a function to start an oscillator. Use this function to start the LFXO and wait for it to be stable.

Task 2: Select Low Frequency Oscillator

The low frequency domain is split in two: the LFA and the LFB domain. In the CMU chapter of the Reference Manual you can find a overview over the clock domains. Find out which domain the RTC belongs to.

The emlib function `CMU_ClockSelectSet()` is used to choose an oscillator for a clock domain. Use this function to choose the LFXO as the clock source for the domain the RTC belongs to.

Task 3: Enable RTC Clock

As for all other peripherals you first need to enable the clock to the RTC. For all low frequency peripherals there is also an additional special clock that needs to be enabled. The clock CORELE provides synchronization between the high and low frequency domains and needs to be enabled before software can access the low frequency registers. Enable the clock to RTC and CORELE.

Task 4: Set Compare Value

A count value corresponding to x milliseconds is given by the following formula

x ms delay with RTC

$$n_x = \text{RTC\_FREQ} * x / 1000 \quad (3.1)$$

Set the Compare Register 0 (COMP0) to a value corresponding to 50 ms.

Task 5: Configure the RTC

The emlib API for RTC includes an initialization struct: `RTC_Init_TypeDef`. A pointer to this struct is passed to `RTC_Init()` to configure the RTC. Use this method to enable the RTC and make it wrap around when reaching the value in COMP0.

Description of the initialization structs are found in the **emlib documentation.**

Task 6: Enable RTC Interrupt

Enable the RTC COMP0 interrupt.

Task 7: Toggle LED in RTC Interrupt Handler

Implement the RTC interrupt handler. The handler should toggle the LED on every COMP0 interrupt. Remember to clear the interrupt flag before returning.

4 Revision History

4.1 Revision 1.01

2014-05-23

Moved to Silicon Labs license on source files

Added Simplicity IDE projects

4.2 Revision 1.00

2013-02-13

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories 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 Silicon Laboratories 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. Silicon Laboratories 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. Silicon Laboratories 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 Silicon Laboratories. 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. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories 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

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

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

List of Equations

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