EFM32
... the world's most energy friendly microcontrollers

Introduction to EFM32 API

TM0007 - Hands-on Exercise

Introduction

In this exercise you will learn how to write some basic programs for the EFM32 and how to find the relevant documentation.

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 CMSIS Register Definitions

The peripheral registers are documented at the end of each chapter in the reference manual. All peripheral registers are 32 bit wide, but not all bits are used. Unused bits are marked as reserved and should not be modified.

Figure 2.1. DAC0_CTRL Register

26.5.1 DACn_CTRL - Control Register

Offset	Bit Position																											
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset											0x0			0x0			0x0			0			0x0		0	0	0x1		0x0		0	0
Access											RW			RW			RW			RW			RW		RW	RW	RW		RW		RW	RW
Name											REFRSEL			PRESC			LPFFREQ			LPFEN			REFSEL		CH0PRESCRST	OUTENPRS	OUTMODE		CONVMODE		SINEMODE	DIFF

Bit	Name	Reset	Access	Description
31:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0.*		
21:20	REFRSEL	0x0	RW	**Refresh Interval Select**

Select refresh counter timeout value. A channel x will be refreshed with the interval set in this register if the REFREN bit in DACn_CHxCTRL is set.

Value	Mode	Description
0	8CYCLES	All channels with enabled refresh are refreshed every 8 prescaled cycles
1	16CYCLES	All channels with enabled refresh are refreshed every 16 prescaled cycles
2	32CYCLES	All channels with enabled refresh are refreshed every 32 prescaled cycles
3	64CYCLES	All channels with enabled refresh are refreshed every 64 prescaled cycles

2.1 Reading and Writing

The EFM32 header files defines the names for each register according to the reference manual and groupes together all registers for each peripheral in one struct. Writing to a register can be done as follows:

```
DAC0->CH0DATA = 100;
```

Note that this register is defined in the reference manual as `DACn_CH0DATA`. In the C-code we have replaced the underscore with the dereferencing arrow and substituted 0 for the n.

Reading from a register is done in the same way:

```
myVar = DAC0->STATUS;
```

2.2 Bit Fields

Often when modifying the registers we are interested in setting specific bits. The header files defines constants for each bit field in the registers. In `DACn_CTRL` the `SINEMODE` bit field corresponds to bit 1 in this register, see Figure 2.1 (p. 3). This means that in the header files there will be a definition like this:

```
#define DAC_CTRL_SINEMODE (0x1UL << 1)
```

When setting a bit in a control register it is important to make sure you do not unintentionally clear other bits in the register. To ensure this it is common or OR the new bits with the existing content.

```
DAC0->CTRL |= DAC_CTRL_SINEMODE;
```

Clearing a bit is done by ANDing the register with a value with all bits set except for the bit to be cleared:

```
DAC0->CTRL &= ~DAC_CTRL_SINEMODE;
```

When setting a new value to a bit field containing multiple bits, a simple OR function will not do, since you will risk that the original bit field contents OR'ed with the mask will give a wrong result. Instead you should make sure to clear the entire bit field (and only the bit field) before you OR in the new value like shown below:

```
DAC0->CTRL = (DAC0->CTRL & ~_DAC_CTRL_REFRSEL_MASK) | DAC_CTRL_REFRSEL_16CYCLES;
```

Note that for bit fields with multiple bits there is an additional mask with all bits in that field set to 1. The same naming convention is followed but this mask is prefixed with an underscore and postfixed with _MASK. Each possible value for the bit field also has its own defined constant.

2.3 Grouped registers

Some registers are grouped together within each peripheral. An example of such a group is the registers associated with each GPIO port, like the Data Out Registers (DOUT). Each GPIO port (A, B, C, ...) contains a DOUT register and the description below is common for all of these. In the reference manual, this registers is simply specified as `GPIO_Px_DOUT`, where the x indicates the port wild card.

In the CMSIS defines the port registers are grouped in an array P[x]. When using this array, we must index it using numbers instead of the port letters (A=0, B=1, C=2, ...). Accessing the DOUT register for port C can be done like this:

```
GPIO->P[2].DOUT = 0x000F;
```

3 Exercise 1: Register Operation

This example will show how to write and read registers using the CMSIS structs. In the main function in the **1_register_operation.c** there is a marker space where you can fill in your code.

Figure 3.1. CMU HFPERCLKEN0 Register

11.5.18 CMU_HFPERCLKEN0 - High Frequency Peripheral Clock Enable Register 0

Offset	Bit Position																												
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access															RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name															DAC0	ADC0	PRS	VCMP	GPIO	I2C1	I2C0	ACMP1	ACMP0	TIMER3	TIMER2	TIMER1	TIMER0	UART1	UART0	USART2	USART1	USART0

Task 1: Enable timer clock

In this example we are going to use TIMER0. By default all peripheral clocks are disabled, hence we must turn on the clock for TIMER0 before we use it. If we look in the CMU chapter of the reference manual, we see that the clock to TIMER0 can be switched on by setting the TIMER0 bit in the HFPERCLKEN0 register in the CMU peripheral. This step is already filled in.

Task 2: Start timer

Start the timer by writing a 1 to the START bit in the CMD register in TIMER0.

Task 3: Wait for threshold

Create a while-loop that waits until counter is 1000 before proceeding.

Task 4: Stop the timer

Stop the timer by writing a 1 to the STOP bit in th CMD register in TIMER0.

Task 5: Download and debug

Once your program compiles, download the program and enter a debug session. Use the register viewer and find the TIMER0 registers. While the program is still halted at the begginning of `main()`, try to modify any of the TIMER0 registers. This is not possible, because the clock to TIMER0 has not been enabled yet.

Locate the RUNNING bit in the TIMER0_STATUS register. Watch this bit as you single step over the code that starts the timer.

Extra task: Implement a delay function

Using what you have learned implement a function `void delay(uint32_t ms)` that will wait for a number of milliseconds.

The function should do the following:

- Reset the counter value

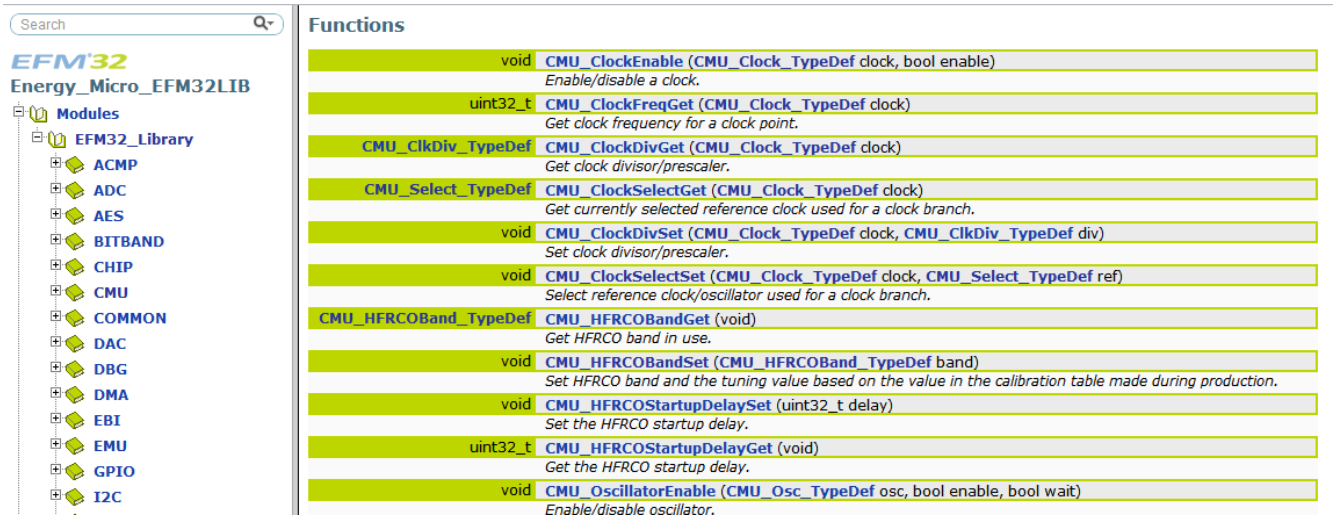
- Start the timer
- Wait the given number of milliseconds
- Stop the timer

Make sure the clock to the timer is enabled and use the PRESC field of TIMER0->CTRL to slow down the timer. By default it is running from the HFRCO oscillator at 14 MHz.

4 Exercise 2: GPIO

In this exercise we will move on to use emlib functions. It is very useful to have the documentation open when doing the next tasks. To open the emlib documentation click **API Documentation** in **Simplicity Studio** and select **emlib Peripheral API**. Then, expand **Modules->EFM32_Library**.

Figure 4.1. Documentation for the CMU-specific emlib functions



Task 1: Turn on GPIO clock

In the list of CMU functions we find the following function to turn on the clock to the GPIO:

```
void CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)
```

If we click on the function, we are shown a description of how to use the function. If we also click on the CMU_Clock_TypeDef link we are taken to a list of the allowed enumerators for the *clock* argument. Use this function to turn on the GPIO clock.

Task 2: Configure GPIO pin for LED

Table 4.1. STK Pin Configurations

STK	LED	Button
EFM32GG-STK3700	PE2	PB9
EFM32TG-STK3300	PD7	PD8
EFM32_G8xxx_STK	PC0	PB9

In **Simplicity Studio** under **Kit Documentation** we find the User Manual for the STK that we are using, which describes which pins are routed to LEDs and buttons. Table 4.1 (p. 7) lists suitable values for different STKs.

Note

If you are using a Development Kit, the User LEDs and buttons are not routed directly to any MCU Pin and must be controlled by board control functions. To follow this exercise it is recommended that you use the prototyping board to connect a LED and and button.

Looking into the available functions for the GPIO we find the following function to configure the mode of the GPIO pins:

```
void GPIO_PinModeSet(...)
```

Use this function to configure the LED pin as Push-Pull output with the initial DOUT value set to 0.

Task 3: Configure GPIO pin for button

Refer to Table 4.1 (p. 7) and configure the pin for the button as input to be able to detect the button state. The push buttons are pulled high by external resistors.

Task 4: Toggle LED when button is pressed

Write a loop that toggles the LED every time the button is pressed. Make sure that you do not only check that the button is pressed, but also that it is released, so that you only toggle the LED once for each time you press the button.

HINT: To wait for a button event you can use a busy loop that continuously checks the state of the button.

Extra Task: LED Animation

Use the delay function you wrote in Exercise 1 make the LED blink with a 100 ms period.

5 Revision History

5.1 Revision 1.01

2014-05-23

Moved to Silicon Labs license on source files

Added Simplicity IDE projects

5.2 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 Figures

List of Tables

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