NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Energy Efficient Computer Design
Laboratory Report

TDT4258 Exercise 1

Group 27:

Eirik Rognø
Eirik Fosse
Michael McMillan

September 21, 2015

Abstract

This report describes how we programmed a microcontroller to turn on LEDs on a gamepad whenever a button was held down. Additionally the report explains how we altered the energy mode of the microcontroller in order to keep the energy consumption as low as possible. We accomplished this by disabling unnecessary RAM blocks, enabling deep sleep and utilizing interrupt handlers in favour of polling the buttons.

1 Introduction

In the first exercise we were to write a program for an EFM32GG DK3750 development board equipped with an ARM Cortex-M3 CPU.

One of the imposed requirements was to program in assembly. The program had to handle input from buttons on a custom made gamepad and output signal to a row of LEDs.

Additionally, power consumption had to be kept to an absolute minimal while the program was running. To accomplish this the exercise strongly encouraged the use of interrupts as well as static and dynamic power reduction.

It was left up to the group to decide what the program was supposed to do. We decided to make the program behave such that pressing the leftmost button on the gamepad would enable the first LED. Pressing the next button in a clockwise manner would trigger the next LED and so on.

2 Background and Theory

The hardware platform consists of an EFM32GG-DK3750 development board from Silicon Labs/Energy Micro. It features an "[...] energy-effective 32-bit ARM Cortex M3-based microcontroller [...]" [4], a Thin-Film-Transistor (TFT) LCD display and a gamepad.

A power measurement subsystem was also provided in order for us to measure the power consumption of our program. Furthermore, a gamepad with 8 buttons, a jumper and a row of 8 LED lights was also a part of the hardware platform.

We can distinguish power consumption into two parts: dynamic power and static power. The latter is more or less a constant and can be determined by setting the clock frequency of the CPU close to 0 Hz. Dynamic power is correlated with the clock frequency due to the transistors being turned on or off. [1, MCU Power Consumption]

EFM32 devices usually have five different energy modes allowing the designer to scale and customize the resources needed in order to minimize power consumption. The following table summarizes how the different energy modes alters the static resources on an EFM32TG840F32 device.

Static power consumption in power domains						
Power domain	EM0	EM1	EM2	EM3	EM4	Static power TG840F32
Core domain	On	On	Retained	Retained	Off	34 µA
Low Energy domain	On	On	On	On	Off	590 nA
Backup domain	On	On	On	On	Off	100 nA

[3, page 2] In the methodology chapter we will explain how we altered our device to use EM3.

3 Methodology

We first started setting up all the relevant registers as described in the provided compendium [4, page 25]. We enabled the GPIO clock, set the pins for the lights to output and the pins for the buttons to input as well as setting the high-drive strength and internal pull-up.

After we verified that we could turn the lights on by writing 0 to bit 8-15 in GPIO_PA_DOUT we started setting up interrupts. We decided to skip the polling method as we knew this would not be an optimal solution and because all of us already had experience with a similar microcontroller, the EFM32 Gecko Starter Kit from last year. `https://www.silabs.com/products/mcu/lowpower/Pages/efm32-g8xx-stk.aspx`

We decided to make a simplistic program, and focus on low energy consumption. To achieve this we set up the GPIO registers to enable interrupts [4, page 27] whenever a button on the gamepad was pressed. In the GPIO interrupt handler we read the button inputs from GPIO_PC_DIN and proceeded to left shift this value by 8 bits and write it to GPIO_PA_DOUT to turn on the corresponding LEDs. The following code shows how we linked the buttons to the LEDs.

```
1 LDR R0, =GPIO_PC_BASE      // Load address of GPIO port C base to R0
2 LDR R1, [R0, #GPIO_DIN]    // Load address of GPIO DIN to R1
3 LSL R1, R1, #8             // Leftshifting value of R1
4 LDR R0, =GPIO_PA_BASE      // Load address of GPIO port A base to R0
5 STR R1, [R0, #GPIO_DOUT]   // Store led values in R1 to GPIO DOUT on port A.
```

Listing 3.1: GPIO handler

3.1 Energy saving

Since the name of the course is Energy Efficient Computer Design, we wanted to make the system as energy efficient as possible. To do this, we focused on the most effective ways to reduce energy consumption in the EFM32 Energy Optimization Application Note [3].

Altering energy mode

As explained earlier the EMF32 has several energy modes (0-4). To enter a low energy mode we first had to configure the desired energy mode through the EMU_CTRL register and the SLEEPDEEP bit in the Cortex-M3 System Control Register. By setting the first three bits in EMU_CTRL to 0 we enabled energy mode 2 and lower, and reduced voltage regulator drive strength in EM2 and EM3.

```
1 LDR R0, =EMU_BASE          // Load address of EMU Base
2 MOV R1, #0                 // Put value 0 directly in R1
3 STR R1, [R0, #EMU_CTRL] // Store 0 back to memory at EMU CTRL.
```

Listing 3.2: Configuring EMU Control Register

Enable deep sleep

In the system control register we did three configurations. We enabled deep sleep by writing 1 to the SLEEPDEEP bit. We also wrote 1 to the SLEEPONEXIT bit to enter sleep mode when returning from handler mode to thread mode. Lastly we wrote 0 to the SEVONPEND bit such that only enabled interrups can wake up the processor.

```
1 LDR R0, =SCR     // Load System control block
2 MOV R1, #0b110   // Put binary 110 directly in R1
3 STR R1, [R0]     // Store R1 in SCR to enable deep sleep mode.
```

Listing 3.3: Configuring System Control Register

Disable RAM blocks

The device has a total of 4 RAM blocks. Since our program did not require more RAM

than one block of 32 kB we decided to disable RAM blocks 1 to 3 in order to reduce power consumption. Current is decreased by approximately 170 nA per 32 KB block that is turned off [3, p. 4].

```
1 LDR R0, =EMU_BASE              // Load address of EMU Base
2 MOV R1, #0b111                 // Put binary 111 (7) in R1
3 STR R1, [R0, #EMU_MEMCTRL]     // Store R1 to EMU Memory control
```

Listing 3.4: Disabling RAM blocks

Additionally we had to adjust the linker file to account for the reduced size of available RAM. Originally it was set to 128 kB.

Clock optimization

Lastly we optimized the clock setup for our application. We set the frequency band to 1.2MHz for the high frequency clock. We tried to set the correct tuning for 1.2MHz, but we were unable to retrieve the value from HFRCO_CALIB_BAND_1. We tried various values and set it to 0 since it gave us the result we wanted. We also lowered the frequency of the low frequency clock.

```
1 LDR R0, =CMU_BASE              // Load address of CMU Base
2 MOV R1, #0                     // Put 0 in R1
3 STR R1, [R0, #CMU_LFRCOCTRL]   // Store 0 in LFRCOCTRL (tuning)
4 STR R1, [R0, #CMU_HFRCOCTRL]   // Store 0 in HFRCOCTRL, (set band and tuning)
```

Listing 3.5: Clock optimizaiton

3.2 Testing

Due to our eagerness we chose not to use the debugger. Since the first exercise was relatively basic in terms of logic we did not expect to save any time on setting up the gdb and properly debugging. Instead we simply carried out the testing by flashing the device and making sure the correct LED were turned on whenever a button was held down.

4 Results

Since we already had experience programming in assembly with both polling and interrupts, we decided to write the program using interrupts directly. We knew from experience that this is the most efficient solution in regards to power consumption. Therefore we cannot compare polling to interrupts.

In the following sections we present the measured power consumption for the microcontroller. The power measurements we describe were carried out while the microcontroller was asleep. We define asleep as waiting for an interrupt. This implies that the microcontroller was in energy mode 3.

4.1 Without energy saving

Prior to optimizing for energy efficiency the power consumption was approximately 1.8 mA. See figure 4.1

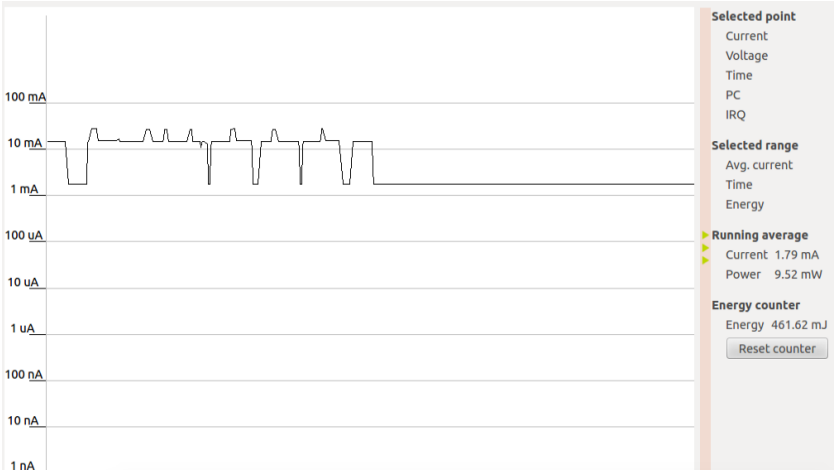

Figure 4.1: Lights turned off without energy saving.

4.2 Energy saving

Below we present results from incrementally optimizing the energy efficiency. In figure 4.2 we present the final result after all the optimizations had been applied when the microcontroller is asleep. Figure 4.3 shows the power consumption while the lights are turned on.

Altering energy mode and enabling deep sleep

After altering the energy mode and enabling deep sleep we managed to lower the power consumption to 1.2 μA.

Disabling unused RAM blocks

By disabling the unused RAM blocks the power consumption fluctuated between 400 nA and 900 nA.

Clock optimization

Due to the power fluctuations we could not determine the improvement in energy efficiency after optimizing the clocks of the microcontroller. Because of this we have decided to comment out the lines that try to optimize the clocks in the attached source code.

The provided images (figure 4.1, 4.2 and 4.3) are from the eAProfiler program, which monitors the microcontrollers energy consumption.

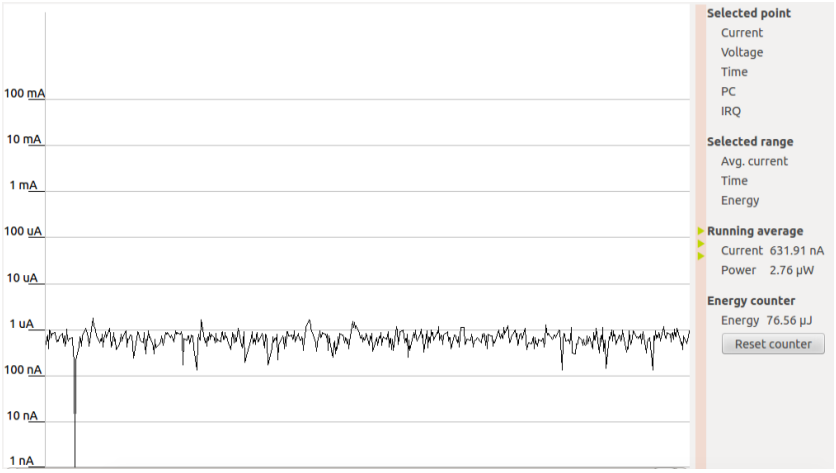

Figure 4.2: Lights turned off with energy saving.

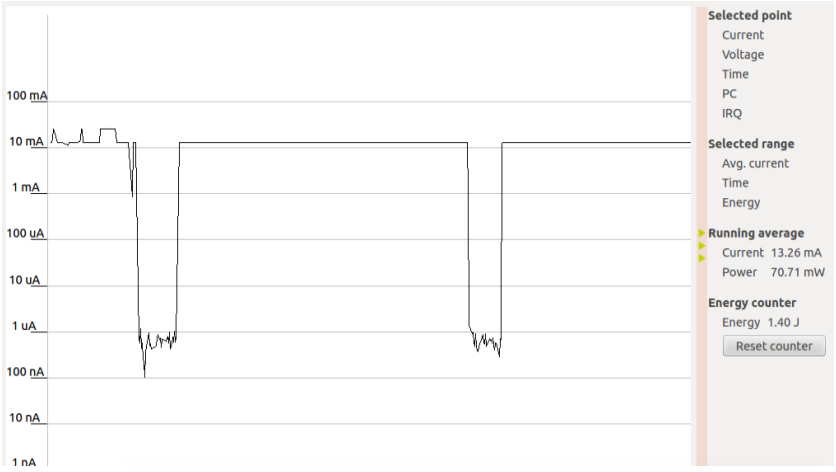

Figure 4.3: Lights turned on with energy saving.

5 Conclusion

Creating an energy efficient program relies heavily on the ability to scale down hardware resources that is not needed. Luckily the EFM32GG-DK3750 makes it very convenient for the designer to easily customize what should be enabled or disabled.

The EFM32 MCU family "[...] is the world's most energy friendly microcontroller and is specially suited for use in low-power and energy sensitive applications, including energy, water, and gas metering, building automation, alarm and security, and portable medical/fitness equipment" [2].

Being able to create energy efficient systems is imperative in industries that rely on durable devices. Conclusively we found that the power consumption could be greatly lowered when optimizing for energy efficiency.

5.1 Evaluation of the Assignment

We found the assignment to be a great learning experience on how to dig into manuals and application notes in order to write energy efficient software.

As mentioned earlier in the report, the entire group had participated in a pilot program for the TDT4160 course in 2014. TDT4160 had already given us some insight into how to write assembly and use GPIO to interface with peripherals. Building on this with energy efficiency was interesting, albeit frustrating at times (due to mistakes on our end).

Bibliography

[1] *Introducing the EFM32 Wonder Gecko.* Silabs.

[2] Ultra low power 32-bit microcontroller technology.

[3] *EFM32 Energy Optimization Application Note.* Silabs, 2015.

[4] *TDT4258 Compendium - Lab Exercises in TDT4258 Low-Level Programming.* NTNU, 2015.