NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Exercise 1

Group 17:

Marcus Bjørnå
Asgeir Hunshamar
Bendik Bogfjellmo

September 29, 2019

1 Overview

The goal of exercise 1 is to develop a simple input/output program for the Silicon Labs EFM32GG-DK3750 development board by using the ARM assembly language. By using assembly to manipulate different registers in the system this report describes software for controlling LEDs through simple input buttons on an external memory mapped I/O gamepad connected to the board.

Two solutions will be implemented; a simple polling solution and an interrupt-based solution. The energy efficiency of both solutions will then be compared.

1.1 Setup for button input and LED output

1.1.1 GPIO CMU

The EFM32GG developer kit has the capability to turn on and off the clock for individual I/O units. This means that operators have to manually enable the clock for the GPIO controller, the code below is written to perform this task.

```
// Enable GPIO clock
ldr r4, =CMU_BASE
ldr r5, [r4, #CMU_HFPERCLKEN0]

mov r6, #1
lsl r6, r6, #CMU_HFPERCLKEN0_GPIO
orr r5, r5, r6

str r5, [r4, #CMU_HFPERCLKEN0]
```

1.1.2 Drive strength

Drive strength of the GPIO-pins used to control the LEDs have to be configured so that the LEDs are provided with sufficient power/current to light up. GPIO pin configuration is covered on page 758 of the reference manual.[1]

```
// Set high drive strength
mov r1, #1
ldr r2, =GPIO_PA_BASE
str r1, [r2, #GPIO_CTRL]
```

1.1.3 Setup for LED output

In order to output a signal for turning on and off the LEDs, the code has to specify the pins the LEDs are connected to as output pins. The LED pins have to be set to push-pull output with drive-strength set by DRIVEMODE. This is done by writing 0x55555555 to the GPIO_MODEH register, as specified in the EFM32GG-RM manual on page 676.[1]

```
// sett output pins 8-15
ldr r3, =0x55555555
str r3, [r2, #GPIO_MODEH]
```

The LEDs responsiveness can then be tested by writing to the GPIO_DOUT register. Since the buttons are active lows writing 0 on the corresponding bits in the GPIO_DOUT register should illuminate the corresponding LEDS.

Since pins 8-15 are being used for the LEDs, the 8 least significant bits are redundant for controlling the LEDS. The following code successfully illuminated every other LED on the board.

The lsl command left shifts the content of the register by eight bits before storing it in the GPIO_DOUT register such that the code is manipulating pins 8-15.

```
ldr r6, =0b1010101
lsl r6, r6, #8 // left shift 8 bits
str r6, [r2, #GPIO_DOUT]
```

1.1.4 Setup for button input

The next step is to enable button input for the board.

The buttons are connected to pins 0-7 on the board.

By writing 0x33333333 to GPIO_PC_MODEL the input mode is enabled, as described in the reference manual on page 758.[1]

```
// setup for knapper
ldr r6, =0x33333333
ldr r3, =GPIO_PC_BASE
str r6, [r3, #GPIO_MODEL]
// sett internal pull-up
ldr r6, =0xff
str r6, [r3, #GPIO_DOUT]
```

1.2 Baseline Solution

The baseline solution is implemented through a simple polling teqnique in the while-loop where the values are loaded from the button register and written directly to the LEDs by left shifting them by 8 bits and storing the data to the GPIO_DOUT register. This will directly mirror the button input with the corresponding LED output.

```
#GPIO_DIN register and store

main_loop:

ldr r7, [r3, #GPIO_DIN]
lsl r7, r7, #8
str r7, [r2, #GPIO_DOUT]

b main_loop
```

A different solution, also using a while loop, but polling through different button combinations in order to output different LED patterns was also implemented. This solution would also keep the LED states until a new button input was polled.

```
main_loop:
ldr r7, [r3, #GPIO_DIN] // button input

cmp r7, 0b11111110 // button 1 is pressed
beq ill_led_1

cmp r7, 0b11111101 // button 2 is presed
beq ill_led_2

cmp r7, 0b11111011 // button 3 is pressed
beq ill_led_3

cmp r7, 0b11110111 // button 4 is pressed
beq ill_led_4

cmp r7, 0b11101111 // button 5 is pressed
beq ill_led_5

cmp r7, 0b11011111 // button 6 is pressed
beq ill_led_6

cmp r7, 0b10111111 // button 7 is pressed
beq ill_led_7

cmp r7, 0b01111111 // button 8 is pressed
beq ill_led_8

b main_loop


ill_led_1:
    ldr r11, = 0xFE00
    str r11, [r2, #GPIO_DOUT]

    b main_loop

ill_led_2:
    ldr r11, = 0xFC00
    str r11, [r2, #GPIO_DOUT]

    b main_loop

ill_led_3:
    ldr r11, = 0xF800
    str r11, [r2, #GPIO_DOUT]

    b main_loop

ill_led_4:
    ldr r11, = 0xF000
    str r11, [r2, #GPIO_DOUT]

    b main_loop

ill_led_5:
    ldr r11, = 0xE000
    str r11, [r2, #GPIO_DOUT]

```

```
59      b main_loop
60
61  ill_led_6:
62      ldr r11, = 0xC000
63      str r11, [r2, #GPIO_DOUT]
64
65      b main_loop
66
67  ill_led_7:
68      ldr r11, = 0x8000
69      str r11, [r2, #GPIO_DOUT]
70
71      b main_loop
72
73  ill_led_8:
74      ldr r11, = 0x0000
75      str r11, [r2, #GPIO_DOUT]
76
77      b main_loop
```

1.3 Improved Solution

As is well known in embedded programming, constantly polling the input unit for updates is a big waste of system resources. In order to make a more energy efficient system we introduce a new solution to the problem using interrupts. When a button is pressed we want an interrupt signal to be generated, alerting the CPU when it needs to turn on the LEDs. This allows the CPU to sleep when there is nothing to do.

Most of the setup can be reused from the baseline solution, but in order to enable interrupts in our system some more setup is required.

We enable interrupts by writing 0x802 to ISER0

```
1  ldr r7, =0x802
2  ldr r8, =ISER0
3  str r7, [r8]
```

GPIO interrupts are enabled on port C by writing 0x2222222 to GPIO_EXTIPSELL

```
1  ldr r7, =0x22222222
2  ldr r8, =GPIO_BASE
3  str r7, [r8, #GPIO_EXTIPSELL]
```

We want interrupts signals to be generated when buttons are pressed and relased. This is done by writing 0xff to GPIO_EXTIFALL and 0xff to GPIO_EXTRISE. To enable interrupt generation 0xff is written to GPIO_IEN

```
1  // sett 1->0 transition
2  ldr r7, =0xff
3  str r7, [r8, #GPIO_EXTIFALL]
4
5  // sett 0->1 transtion
6  str r7, [r8, #GPIO_EXTIRISE]
```

```
7
8 // enable interupt generation
9 str r7, [r8, #GPIO_IEN]
```

In order to enable different energy modes, one can write to the SCR-register. In this example, 6 is written to the SCR, this enables deep sleep mode, and automatic sleep when the interrupt is dealt with.

```
1 // Write 6 to SCR register to enable low energy mode 2
2 ldr r11, =0x6
3 ldr r12, =SCR
4 str r11, [r12]
5
6 //Run wfi function
7 wfi
```

On interrupts, one can find the source of the interrupt by reading the GPIO_IF register. After the source is determined, the value of the GPIO_IF-register is written to the GPIO_IF-register. This resets the interrupt, and enables us to correctly read the next interrupt.

After the source is of the interrupt is determined, we find out which buttons is pressed by reading the value of GPIO_DIN. This value is then copied to the GPIO_DOUT-register in order to have a 1/1 mapping of button to LED.

```
1 gpio_handler:
2
3 // clear the GPIO_IFC Register
4 ldr r9, =GPIO_BASE
5 ldr r10, [r9, #GPIO_IF]
6 str r10, [r9, #GPIO_IFC]
7
8 // read buttonstatus and copy to led output register
9
10 ldr r10, [r3, #GPIO_DIN]
11 lsl r10, r10, #8
12 str r10, [r2, #GPIO_DOUT]
13
14 bx r14
```

2 Energy Measurements

2.1 Polling based solution

The system was tested for energy consumption using the polling method with one lit LED, two lit LEDs, 4 lit LEDs, and 8 lit LEDs.
As the figures 2.1 2.2, 2.3, 2.4, 2.5 illustrates; When no LEDs was lit the system drew 3.78mA of current at 3.28 volts, yielding the effect of 12.40mW. With one lit LED the system drew 4.31 mA of current, yielding the effect of 14.14mW. With two lit LEDs the system drew 4.92mA of current yielding the effect of 16.14mW. With four lit LEDs the system drew 6.06mA, yielding the effect of 19.88mW. With 8 lit LEDs the system drew 8.32mA, yielding the effect of 27.29mW.

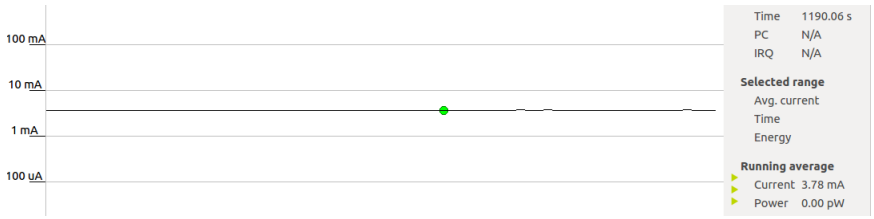

Figure 2.1: Power usage with polling method, no lit LEDs

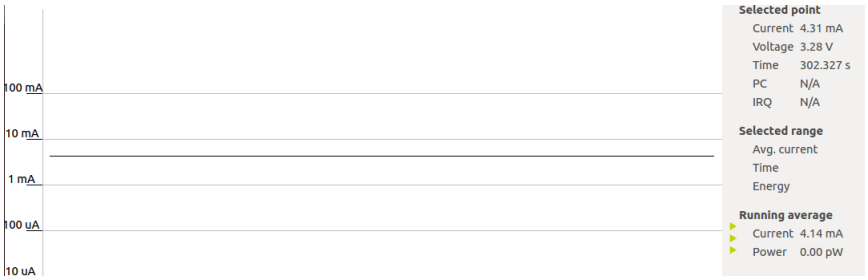

Figure 2.2: Power usage with polling method, 1 lit LED

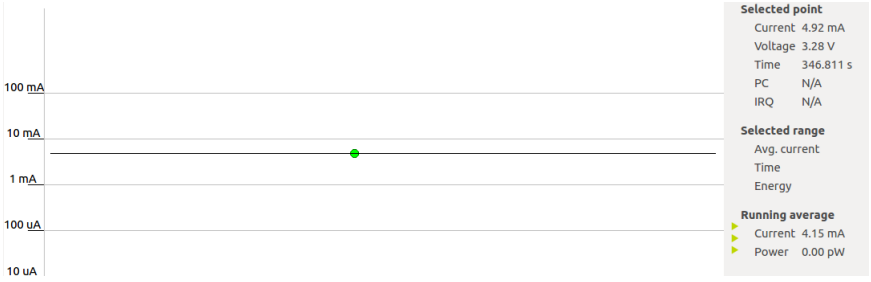

Figure 2.3: Power usage with polling method, 2 lit LEDs

Figure 2.4: Power usage with polling method, 4 lit LEDs

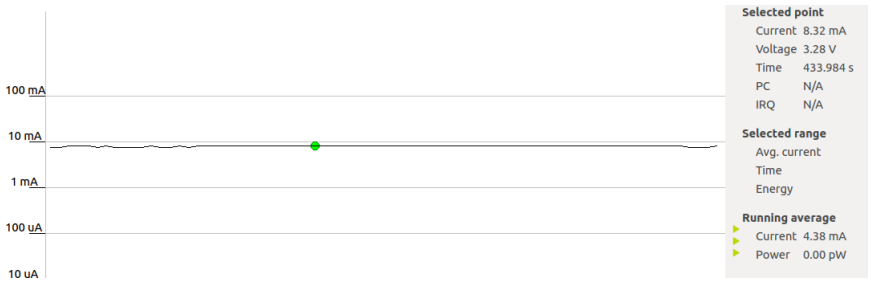

Figure 2.5: Power usage with polling method, 8 lit LEDs

2.2 Interrupt based solution

The system was tested for energy consumption using the interrupt based approach with no lit LEDs one lit LED, two lit LEDs, four lit LEDs, and eight lit LEDs.

As the figures 2.6, 2.7, 2.8, 2.9, 2.10 illustrates; When no LEDs are lit the system drew 1.24µA at 3.28 volts, yielding the effect of 4.06µW. With one lit LED the system drew 642.6 µA of current, yielding the effect of 2.11mW. With two lit LEDs the system drew 1.25mA of current yielding the effect of 4,1mW. With four lit LEDs the system drew 2.50mA, yielding the effect of 8.2mW. With 8 lit LEDs the system drew 4.94mA, yielding the effect of 16.20mW.

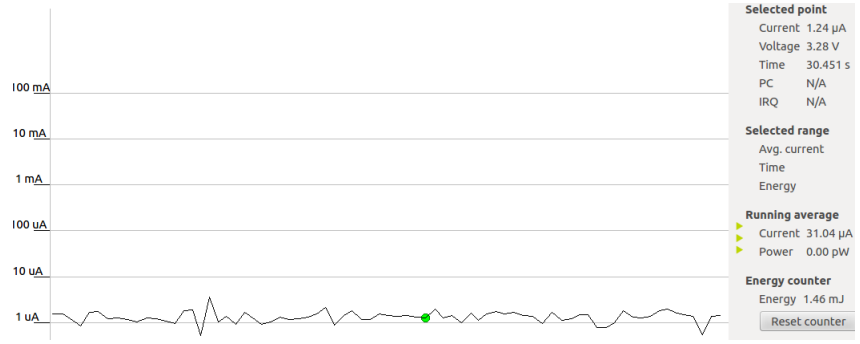

Figure 2.6: Power usage with interrupt based solution, 0 lit LEDs

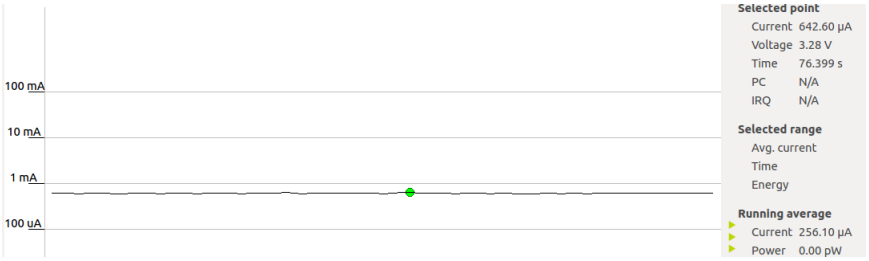

Figure 2.7: Power usage with interrupt based solution, 1 lit LEDs

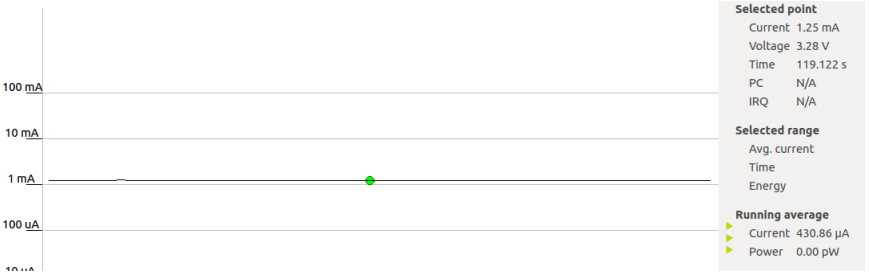

Figure 2.8: Power usage with interrupt based solution, 2 lit LEDs

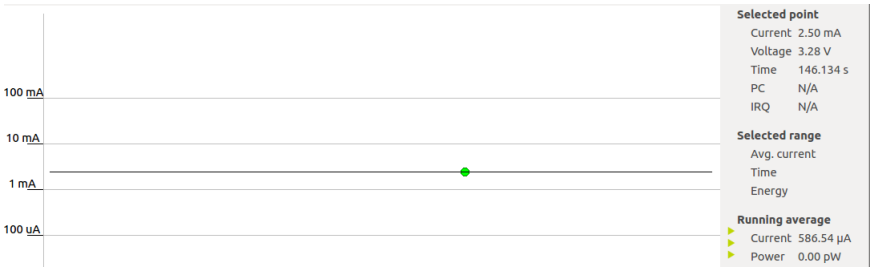

Figure 2.9: Power usage with interrupt based solution, 4 lit LEDs

Figure 2.10: Power usage with interrupt based solution, 8 lit LEDs

2.3 Comparison

As the chart in figure 2.11 shows, there are huge advantages to using the interrupt based method with concerns to power consumption, especially when no LEDs are lit

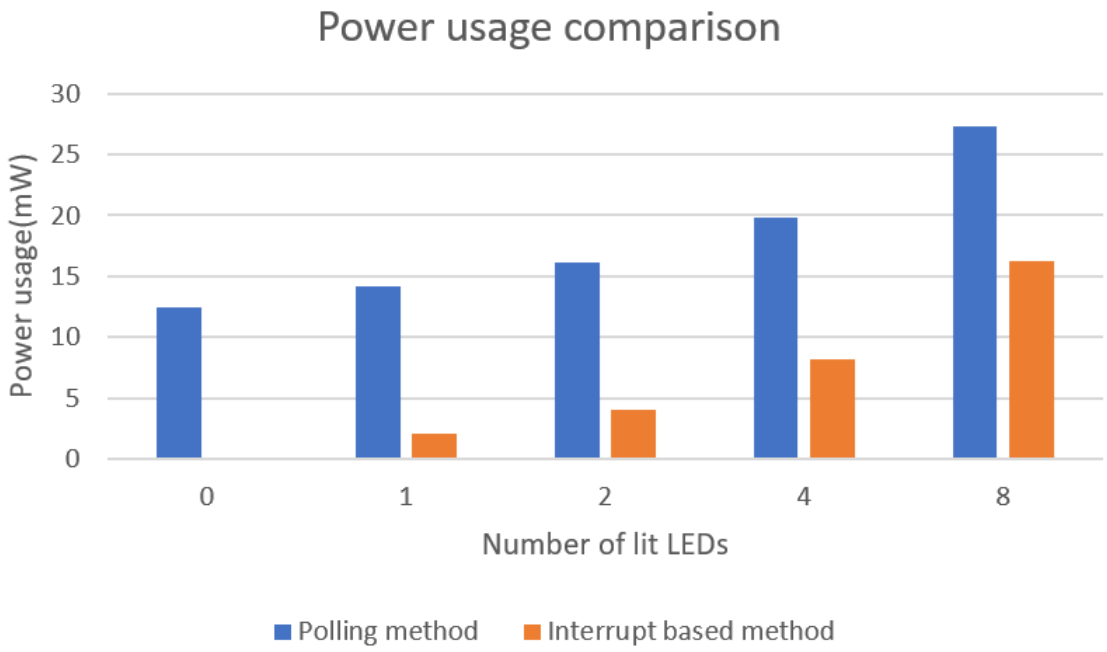

Figure 2.11: Chart comparing the power usage of the interrupt based method and the polling method.

2.4 Conclusions

As proved through comparing the power usage of the two solutions interrupts are more energy efficient than polling and should be used in systems where it is possible and/or feasible to implement with concerns to the system requirements and main function. Enabling the CPU to go to a low energy mode when not interrupted drastically reduced its power consumption, proving it as the superior solution.

Bibliography

[1] Silicon Labs. *EFM32GG reference manual*. Silicon Labs, 2016.