NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Exercise 1 - LEDs & Buttons

Group 8:

Emil Braserud
Wesley Ryan Paintsil
Erblin Ujkani
Andreas Varntresk

September 20, 2017

Contents

1 Overview

This report is written for the first exercise in the course TDT4258 Low-Level Programming, throughout the course all the exercises are done on the EFM32GG prototype board. The requirements for this exercise is to do something with LED lights and buttons. Two different solutions will be implemented, one baseline solution and one improved solution. The baseline solution uses polling to control the LED's and the improved solution uses interrupts to control the LED's. Assembly is the required programming language, and a make-file has to be generated to compile and upload the program to the prototype board. The programs will implement a button to LED interface where a gamepad connected to the board's General Purpose Input Output ports (GPIO), is used as a peripheral for this interface. The main objective is to compare the power efficiency of the two versions, and if possible make further improvements to the improved solution. This report is structured so that the two solutions are described in their own chapter and a third chapter describes the power measurements done with the two different solutions.

1.1 Baseline Solution

The baseline solution consists of two "states", a reset state and a polling state. In the reset state, the GPIO clock in Clock Management Unit(CMU) was set up and activated. The clock for the GPIO controller has to be enabled to use the GPIO's.
To do this you have to set bit number 13 in the *CMU_HFPERCLKEN0_GPIO* to high. This is done by copying the *CMU_HFPERCLKEN0_GPIO* to a register, and using the "orr" function with another register which only has bit number 13 high (see section 11.5.18)[3]. This is done so there is no interference with the other bits in the register. The drive strength of the GPIO's is set in the reset section, and this decides the limit of current the board will send to the LED lights, in this case 20mA. The pins 0-7 is set as inputs for PORT C, and the pins 8-15 is set as outputs on PORT A in accordance with the connected gamepad. After the initialization phase, the program branches to the polling state, where the inputs on port C is constantly read. The input values are saved in a register and occupies bit 0-7.
The bits in the register is then left shifted eight times, because the bits that define the LED's occupies bit 8-15. The register is then loaded to the output GPIO, and the corresponding LED's will light up.

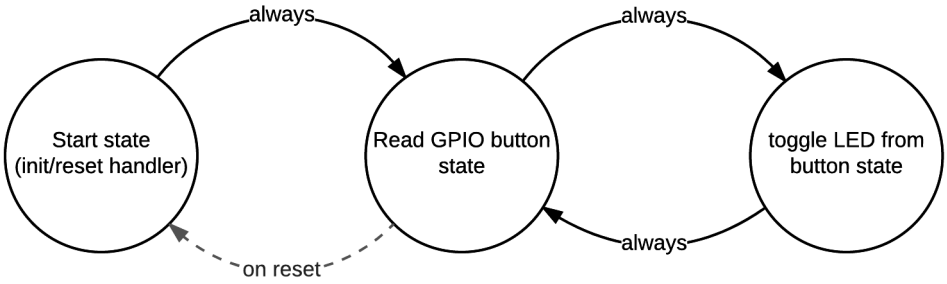


Figure 1.1: State machine of baseline sulution.

1.2 Improved Solution

The improved solution utilizes interrupts instead of polling. This allows the processor to for example sleep when inactive, thus drastically reducing the overall power consumption. The CPU would now also be free to do different tasks should it be necessary. As opposed to polling there the CPU constantly has to check and see whether a button has been pressed. The initialization phase is almost identical to the baseline approach with the exception of interrupt enabling. Enabling the external interrupts on GPIO pins on PORT C required the value *0x22222222* to be written to *GPIO_EXTIPSELL*(external interrupt select low register). According to the EFM32GG-RM datasheet section 32.5.10, this will select PORT C as the trigger for the interrupt flags. [3]

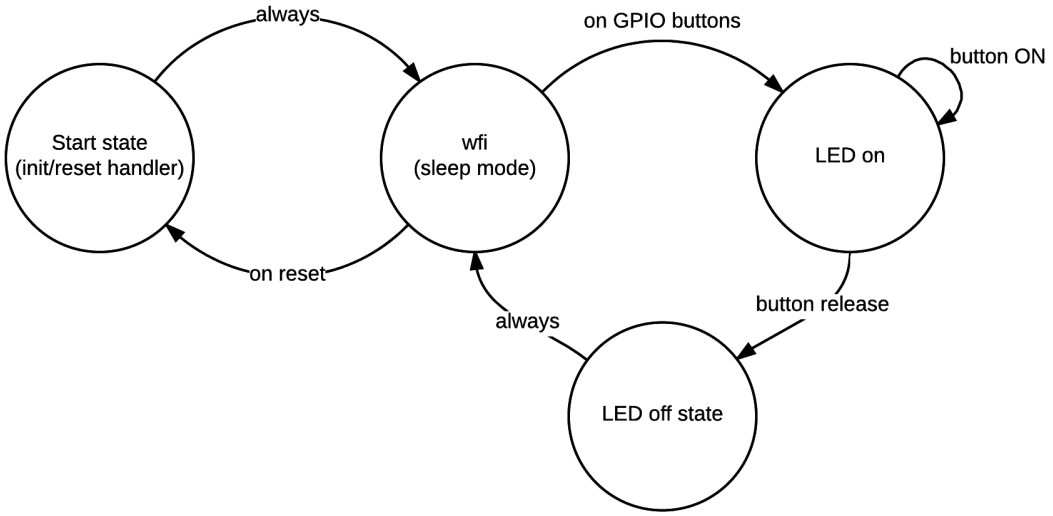


Figure 1.2: The state machine of the improved solution

The next step consists of setting the rising and the falling edge triggers by writing 0xff to the *GPIO_IEN*, *GPIO_EXTIFALL* and *GPIO_EXITRISE* registers. 0xff is written because only pin 0-7 are relevant. (see section 32.5.13,32.5.12,32,5,14) [3].
On the final step, the interrupt generation for Interrupt Request (IRQ) lines *GPIO_ODD* and *GPIO_EVEN* have to be enabled. This is done by writing 0x802 to the address *ISER0*, where the value 0x802 correspond to the IRQs lines of the *GPIO_EVEN* and *GPIO_ODD* interrupt sources. (see section 4.3.1) [2].
After enabling the interrupt, the CPU requires to know the source of the interrupt, which can be obtained by reading the *GPIO_IF* register, this flag also has to be cleared with the *GPIO_IFC* to ensure that the interrupt isn't repeatedly called.[3][1]
This is done inside the interrupt handler which will every time a interrupt is set, it will clear the interrupts and execute the GPIO handler instructions. The GPIO handler routine is the same as the function repeatedly called in the polling label in the baseline solution. It will read the GPIO states of PORT C and store it on the output PORT A for the LED's.
After the handler is finished executing all of its instructions, it needs to return to the next instruction which is the instruction invoked upon interrupt. This is done by the *bx LR* instruction and it is executed at the end of the handler.

1.2.1 Energy Management

The improved solution uses the power management features of the Cortex-M3 to ensure minimal power consumption when idle, the *SLEEPONEXIT* bit in the system control register is set, this ensures that the CPU sleeps when entering thread mode. Additionally deep sleep has been enabled by setting *SLEEPDEEP*. The microcontroller has 5 different energy modes ranging from EM0 - EM4, where EM0 is most energy consuming and EM4 is least energy consuming. The energy saving is done by disabling different peripherals. This program runs in energy mode 2. The *EMU_CTRL* register which contains *EM4CTRL*, *EMVREG*, *EM2BLOCK* are zero by default. (see section 10.5.1) [3] [2]
After configuring these registers, one has to run the Cortex-M3 instruction, 'Wait For Interrupt' (WFI), in order to transition to the low power mode. (section 10.3.2) [3]. After doing this the device can be "waken up" by the external interrupts from the GPIO buttons, and it can do its handler before returning back to sleep mode. Figure. 1.3 shows the different energy modes and its valid transitions from interrupts and exceptions.

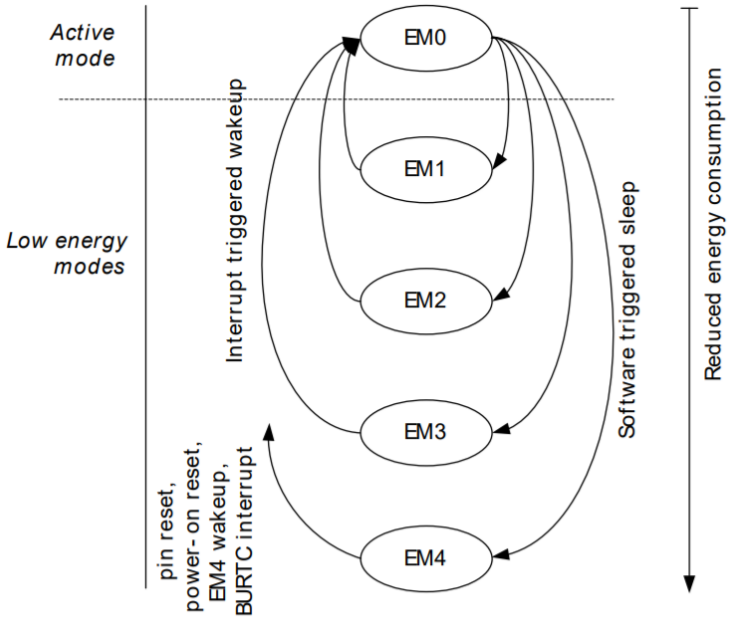


Figure 1.3: EMU energy modes transitions [3]

2 Results

2.1 Program

The final version of the program implements a simple button to LED interface. Button no. N on the game-pad turns on LED no. N as shown in figure. 2.1 , where N ranges from 0-7. This will as expected also have the possibility to turn on multiple LED's concurrently.

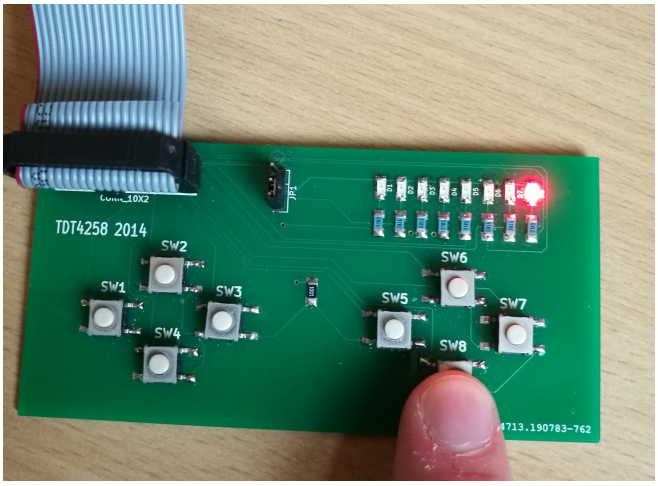
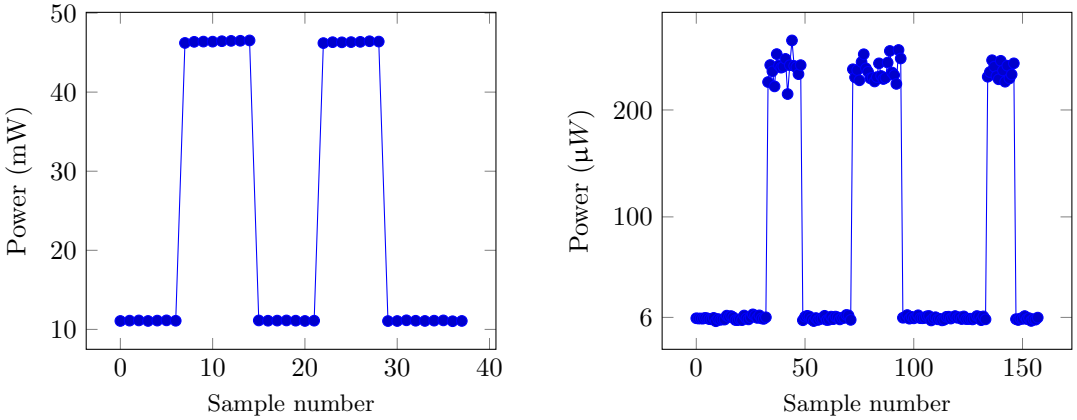


Figure 2.1: Button press

2.2 Energy Measurements

Power consumption measurements were taken using the EnergyAware Profiler application developed by Silicon Labs. Figure 2.2b shows the power consumption of the improved baseline solution during sleep mode. While figure. 2.2a shows the power consumption of the baseline solution using polling. Compared to the polling version as shown in figure 2.2a, it has drastically improved the power efficiency, even though the programs serve the same functionality. During idle mode the improved solution had current peaks of around $1.9\mu A$ corresponding to $5.6\mu W$, while the polling version always stayed around 3.6 mA corresponding to approximately 10.9 mW.
This gives us a total improvement of 1900x times of power efficiency during idle mode. In figure 2.2b, the power consumption during a button press or during the interrupt handler without the LED's current draw is also shown. This reached peaks of around



(a) Polling version

(b) Interrupts and sleep mode version

Figure 2.2: Comparison of power consumption

80 μA or 250 μW. For the polling version the power consumption stays the same at all time, with exception if the LED's are taken into account as shown in the large peaks in figure 2.2a.

Bibliography

[1] Computer Architecture, Department of Computer Design Group, and Information Science NTNU. Compendium: Lab exercises in tdt4258 low-level programming, August 2017. (Accessed on 16/09/2017).

[2] Silicon Labs. Cortex-m3 reference manual, feb 2011. (Available at https://www.silabs.com/documents/public/reference-manuals/EFM32-Cortex-M3-RM.pdf, Accessed on 10/09/2017).

[3] Silicon Labs. Efm32gg reference manual, apr 2016. (Available at https://www.silabs.com/documents/public/reference-manuals/EFM32GG-RM.pdf, (Accessed on 10/09/2017).