NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Exercise 1 - Using the gamepad buttons on the EFM32GG-DK3750

Group 20:

Trude Jostad
Simen Tjøtta Vie
Maren Orvedal

September 19, 2016

Contents

1 Overview of the Solution

In this exercise, we have made two binary files, one containing a baseline solution and one containing an improved solution. The baseline solution uses polling for its functionality to read values from the button registers, and store them to the LED registers. In the improved solution we try to improve the power consumption of the system by taking advantage of interrupt calls instead of constantly polling the registers to look for changes.

1.1 Compiling

The assembly of our code into object files, which was then linked into an executable, was done using a supplied Makefile and its contained rules. These rules executed the arm-none-eabi-as and arm-none-eabi-ld commands to achieve this. Finally, to get a clean binary file, the command arm-none-eabi-objcopy -j .text -O binary is executed. Without the make command, we would have had to manually execute those commands every time we wanted to upload a file, and we have realized the immense value of such a feature.

1.2 Debugging

To debug our program, we used the GNU Debugger, or simply GDB. We uploaded our binary file to the development board, and then started the debug server by issuing the command JLinkGDBServer. This allowed us to place breakpoints and verify register values triggered by our program, which proved essential for our understanding of the assembly code.

1.3 Setting up the GPIO pins

Before we could actually use the GPIO pins, they had to be programmatically set up. This is done by setting different I/O registers for the GPIO clock, the LED lights and the buttons. First, we activated the GPIO clock. This is done by writing 1 to the 13. bit of HFPERCLKEN0. To activate the LED GPIO pins, we have to specify that they are output pins, which is done by writing 0x55555555 to the GPIO_MODEH of port A. Similarly, we have to activate the buttons by specifying that they are output by writing 0x33333333 to GPIO_MODEL of port C.

1.4 Baseline Solution

The goal of the solution is to activate different LED lights when different buttons are pressed. For the baseline solution we used polling strategy, which constantly read values from the button input register, and store them to the LED output register.

The polling loop was made by using a label. The label serves as a function that calls itself at the end of its execution, making it a loop. In the loop values are read from the button input register, and stored to the LED output register.

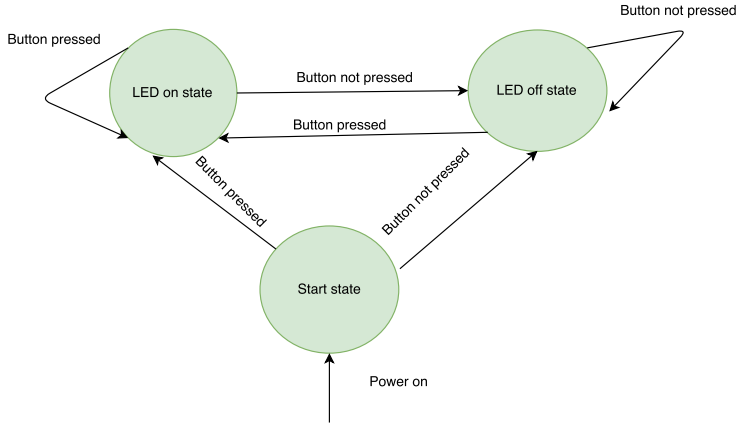

Figure 1.1: simple state machine of polling.

1.5 Improved Solution

The process of continuously polling the GPIOs, as seen in figure 1.1, to check if a change has a occurred is obviously very energy demanding. Luckily, it is unnecessarily demanding, and we can improve the design by putting the CPU to sleep while nothing is happening, and then waking it up, using an interrupt signal when a change actually occurs, see figure 1.2. In our improved solution, we issued the SCR command to put the CPU to deep sleep and wait for an interrupt. Changes were drastic and whilst the development board is a minor device, we realize how the cost must scale when systems are not implemented with an energy efficient design.

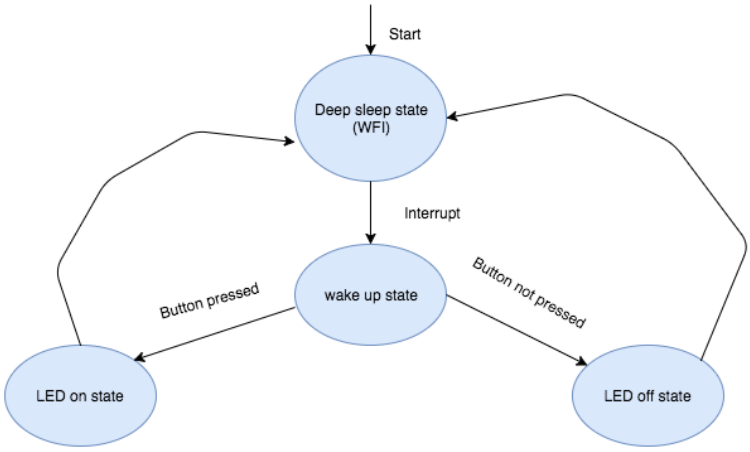

Figure 1.2: simple state machine of interrupt handling.

2 Energy Measurements

To measure the power consumption, we used EnergyAware Profiler (eAProfiler) software, which was preinstalled on the virtual machine we used. We also moved the jumper on the gamepad to its lower position to measure only the power consumption used by the CPU (and not the LEDs), to get a clearer result directly related to the active polling and deep sleep mode. The power consumption itself was measured during active polling and deep sleep state. The results can be seen in figure 2.1 and 2.2.

2.1 Polling Power Consumption

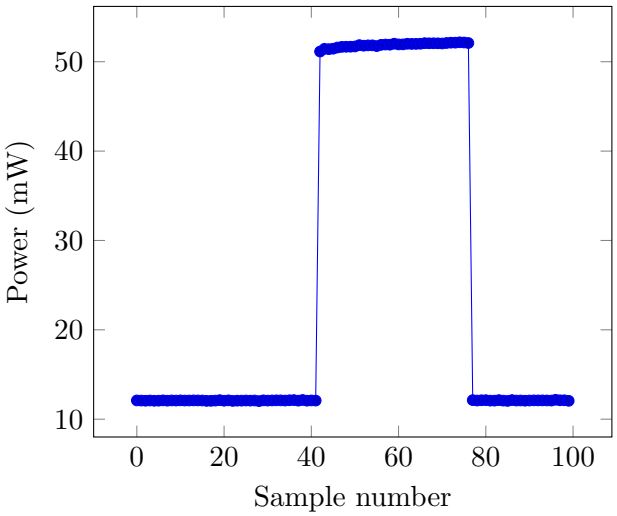

Figure 2.1: Power consumption plot of the baseline solution, generated from eAProfiler data. Note that the y-axis starts at 10mW

Sample number	Current (mA)	Voltage (V)	Power (mW)
1	3.68054	3.28712	12.09837
2	3.67567	3.28682	12.08127
3	3.6708	3.28667	12.06471

Table 2.1: Example consumption values taken from the data/polling_one_light.csv file. The values are from when the processor is continuously polling button values, and no LED lights are on.

Values from table 2.1 shows that the power consumption while the processor is in run state and while actively polling is about 12mW.

Figure 2.1 shows the power consumption while one button is pressed, and power consumption when waiting for a button to be pressed. Notice how the polling state consumes just above 10 mW

2.2 Interrupt Power Consumption

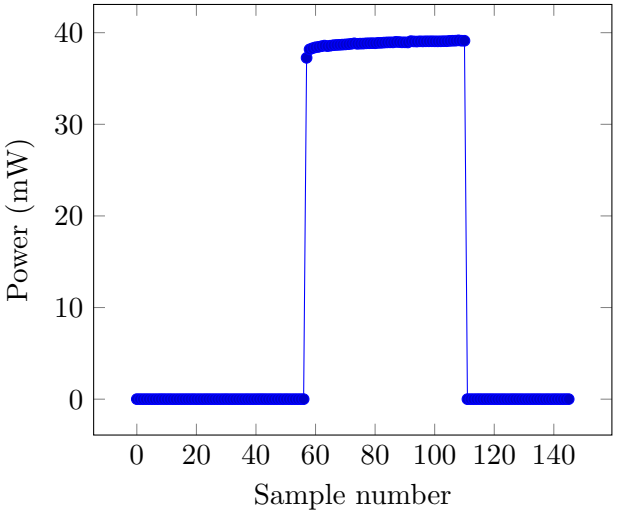

Figure 2.2: Power consumption plot of the improved solution, generated from eAProfiler data. The state changes at sample 60 when the state changes from LED off to LED on, and on 110 when the state changes back to LED off.

Sample number	Current (mA)	Voltage (V)	Power (µW)
1	0.0014728	3.28682	4.841
2	0.000755638	3.28926	2.485
3	0.000938744	3.28995	3.088
4	0.00091967	3.28812	3.024
5	0.000587792	3.28705	1.932

Table 2.2: Example consumption values taken from the data/interrupt_one_light.csv file. The values are from when the processor is in deepsleep mode between interrupts, and no LED lights are on.

Table 2.2 shows the power consumption from when the processor is in deep sleep mode. The values range from about 2 µW to 5 µW. These jumps are however expected when measuring values as low as these. Figure 2.2 shows the power consumption when a LED

light is off and on. When the LED is on, the system draws about 40 mW, making the processor's power usage of 2-5 µW negligible.

2.3 Polling Measurements compared to Deep Sleep Measurements

In tables 2.1 and 2.3, the power consumption when waiting for a button to be pressed is about 12 mW and 3 µW. This implies that the power consumption is reduced about 4000 times in the improved solution. This is a great amelioration, and something we should strive to achieve in the future. This difference is less significant when LED lights are on, as LED lights consume a great deal of power compared to the processor. By looking at the figures 2.1 and 2.2, we can see that the power consumption while polling and a LED light is on is about 38 mW, and about 50 mW when the processor goes to deep sleep mode between interrupts. This is a difference of 32% in power, which is still a great improvement. This difference originates from the polling process consuming about 10 mW, like we saw when the jumper was placed in its lower position.

State	Power Consumption (mW)
Polling and LED off	12
Polling and LED on	50
Interrupt and LED off	~ 0.003
Interrupt and LED on	38

Table 2.3: Consumption values taken from the data/interrupt_one_light.csv file. The values are from when the processor is in deepsleep mode between interrupts, and no LED lights are on.

3 Conclusion

3.1 Conclusions on the exercise

In this exercise we started our learning process of how microcontroller programming is done in Assembly. We achieved both goals of programming the gamepad to control the output LEDs, and observing and analysing how interrupt handling implemented in the system reduces the power consumption compared to using active polling. Another goal of the exercise was to familiarize ourselves with the manuals associated with the development board and its processor. Especially the EFM32GG Reference Manual and the Cortex-M3 Reference Manual proved highly utilitarian for this. The group feels like the exercise yielded valuable knowledge on introduction to microcontroller programming and good practice for future exercises.