NTNU – Trondheim
Norwegian University of
Science and Technology

TDT4258 Low-Level Programming
Laboratory Report

Exercise 1

Group 23:

Maciej Ambrozek
Matej Mnoucek
Jonas Peis
Emanuel Roos

September 29, 2019

1 Overview

The EFM32 has an extension board with buttons and LEDs on it. The goal in this exercise is to control the LEDs with the buttons in some way. To do this, knowledge about the development board with its functions and features and its microcontroller, the ARM Cortex M3, is necessary. This includes the various registers, what they do and how to program them. This information is not given and must be gathered from the documentation and reference manuals.

We are asked to make two solutions: one which uses the simpler method of polling the state of the buttons and responding accordingly while the other one uses interrupts to respond to button presses. Using interrupts enables the processor to sleep while no button is pressed, making it more energy efficient.

For the exercise it is also relevant how much energy the program consumes and we are asked to minimize the energy consumption. The board offers advanced possibilities to monitor the current energy consumption of the microcontroller and therefore makes it possible to compare the energy efficiency of different programs. We optimized both versions to use as little energy as possible.

To compile the code just type `make` to compile both solutions or `make ex1_basic` or `make ex1_improved` to just compile either version. Typing `make upload_basic` or `make upload_improved` will flash the respective version. A working .bin file is also included.

1.1 Baseline Solution

In the baseline solution the microcontroller uses polling to respond to button presses. It continuously reads the state of the buttons and lights up the corresponding LED as long as the button is pressed, creating a one to one mapping of button SWx to LED Dx.

In Figure 1.1 a simple finite state machine of a single button/LED combination is shown. Every button/LED combination behaves in this same way.

Even without interrupts some energy saving methods can be applied. Since the microcontroller is constantly active, we need to reduce the active power consumption as much as possible. We did that by disabling unused RAM blocks and changing the clock source to the Low Frequency RC Oscillator (LFRCO), which runs a at low 32768 Hz. This reduces energy consumption through the lower clock rate as well as a more energy

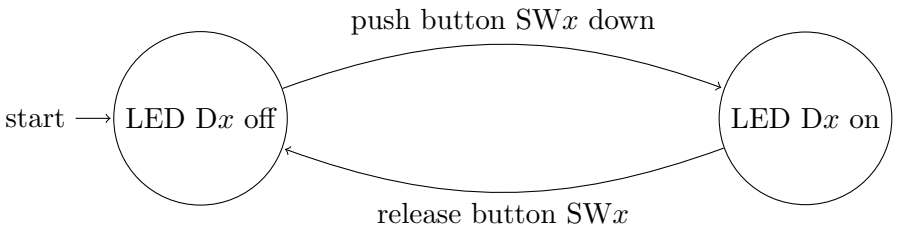

Figure 1.1: Finite state machine of each button/LED combination

efficient oscillator. The frequency is however still high enough to respond to button presses quickly enough such that no lag is noticeable. By these means we were able to improve the current drawn from 3.5 mA to 42 µA.

The program can be divided in two parts or phases. At power up or reset the initialization code is run first (_reset label) and after that the main loop is repeatedly executed (main_loop label).

The initialization phase starts with configuring the microcontroller for lower energy consumption. First, RAM blocks 1-3 are disabled. Consequently, the linker script needs to be changed accordingly as the RAM length is now just 32768 bytes, down from 131072 bytes. Second, the clock source is switched to the Low Frequency RC Oscillator (LFRCO).

After that the clock is configured to be propagated to the General Purpose Input/Output (GPIO) since by default all the periphery is clock gated to save energy. Then, the in- and outputs of the extensions board are configured (buttons as inputs, LEDs as outputs) and initialized.

Afterwards the main loop is entered. Every time it is executed the state of the buttons is read and then directly written to the LED controlling register, lighting up the LED with the same number as the button.

1.2 Improved Solution

The improved solution uses interrupts to handle button presses and has some advanced functionality. It is possible to switch every LED on or off permanently and the intensity of the LEDs can be adjusted.

The way you control the LEDs is shown in part in the finite state machine on Figure 1.2. At any point during the program a specific LED is selected internally. At the start position 1 is selected, meaning LED 1 can be toggled on or off. To toggle the currently selected LED press the down button of either D-pad (SW4/SW8). The currently selected position can be changed by pressing the left and right button of either D-pad (SW1/SW5 left, SW3/SW7 right). Note that you cannot see which position/LED is currently selected – this is only an internal state.

What's missing from the FSM is the change of intensity of the LEDs since it is a global switch that applies to every LED and in the same way in every state. By pressing the up-button of either D-pad (SW2/SW6) the intensity is toggled between the lowest and highest drive strength of 0.1 mA and 20 mA respectively.

The initialization phase of the improved solution includes everything from the basic

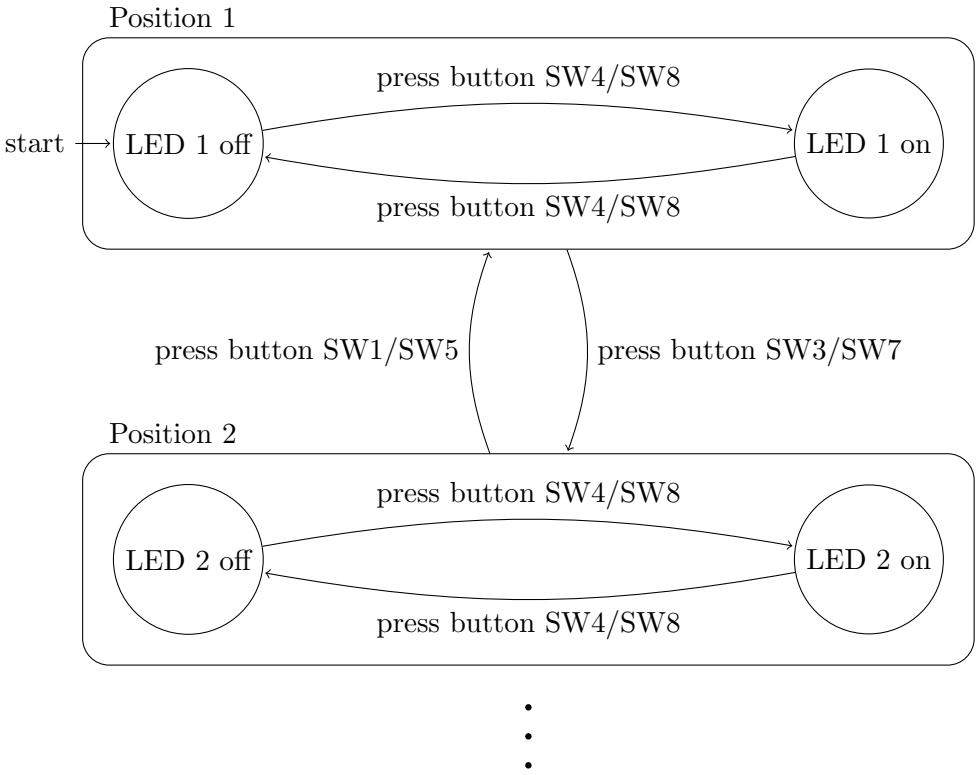

Figure 1.2: Partial FSM showing LED position 1 and 2 out of 8

solution except for the switch to the low frequency clock. We are using interrupts now and the microcontroller can stay in deep sleep as long as the button state doesn't change. Reducing the clock rate will reduce the dynamic power consumption but the static power consumption stays the same as long as the microcontroller is active. Therefore, it is more effective to minimize the time the microcontroller is active by clocking it as high as possible and making it finish it's job faster rather than keeping it active for longer at a lower power consumption.

Additional steps in the initialization phase include configuring interrupts and deep sleep. The exception vector table is changed to link to the GPIO handler for all GPIO interrupts. The GPIO interrupts are configured to trigger on a rising edge which, because of the structure of the buttons, happens whenever a button is released. Finally, the microcontroller is configured to enter EM2 Deep Sleep Mode when going to sleep and to go back to sleep immediately after finishing handling an interrupt. This eliminates the need for any main program as the microcontroller is sent to sleep after the initialization and after handling interrupts.

When an interrupt is triggered the microcontroller jumps to the gpio_handler label. It evaluates which button caused the interrupt and jumps to the according subroutine to execute its function. Afterwards the interrupt is cleared, the microcontroller exits the interrupt handler and returns to sleep.

While testing we noticed the EFM32 board doesn't always behave as expected: first, after reset the microcontroller might not enter deep sleep. A reset of the microcontroller itself is needed to fix this. Second, when resetting of the microcontroller a GPIO interrupt seems to be triggered once at the start, causing LED 1 to be lit immediately. Strangely, we found that if you press the reset button very quickly or lightly, the interrupt does not get triggered.

2 Energy Measurements

The two different approaches to the problem also result in differing power consumption which will be further discussed in this chapter. When a button is pressed, the current flowing over the internal pull up resistors is higher than the current flowing through the microcontroller. Therefore, the spikes in the following figures are caused by the pressed buttons and not by an increased power consumption of the chip.

2.1 Baseline Solution

In the baseline solution using polling the microcontroller consumes about **42 µA**. That is already considerably lower than without any energy saving methods. To achieve that the low frequency oscillator is used as a clock source instead of the high frequency oscillator. Without that the microcontroller would consume a current of about **3.5 mA**. This shows that using a lower clock rate results in a power consumption that is only **1 to 5%** as high as the original one.

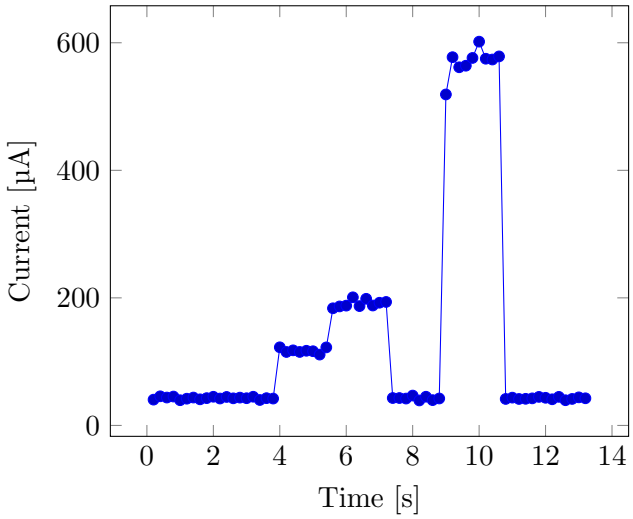

Figure 2.1: Power consumption of the basic solution **without** LEDs.

In Figure 2.1 the current draw excluding the LED power is displayed. The first increase is caused by pressing and holding a single button and the second one by pressing and holding two at the same time. During the high peak at the end all eight buttons were held

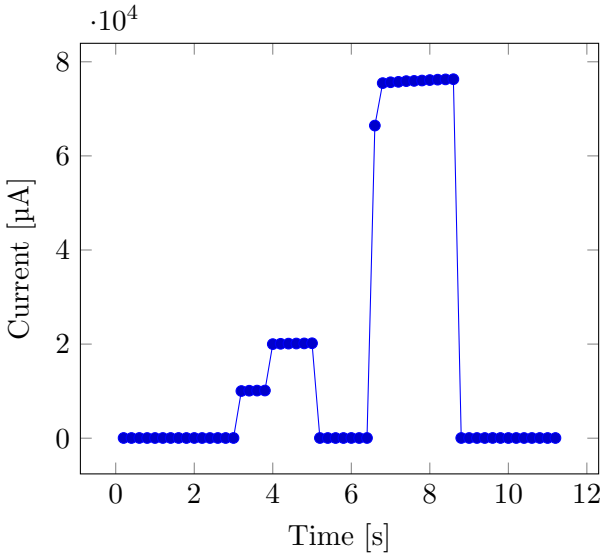

Figure 2.2: Power consumption of the basic solution **with** LEDs.

simultaneously which results in a rather high peak in current consumption because all pull up resistors are connected in parallel and therefore reduce the resistance to ground by a factor of 8.

In Figure 2.2 power consumed by the LEDs is included. The power consumption of the LEDs is much higher than the one of the microcontroller and is therefore the dominating factor in this figure.

2.2 Improved Solution

In the improved version deep sleep is used to save even more power. A lower clock rate would not result in further improvements and is therefore not used. In deep sleep the microcontroller only consumes about **1 µA** of current.

Figure 2.3 shows the power consumption of the improved solution without the LEDs. The three spikes around t = 5s are caused by short presses of a single button. The first turns on LED 1, the second moves to position 2 and the third turns on LED 2. The plateau is caused by changing the intensity from high to low. The increase in power consumption is unexpected and we speculated that it is caused by the fact that all the power drawn by the LEDs is excluded but power drawn by resistors or other components is not. So it seems like when only a low current is provided to the LEDs other parts use the power instead. The spike on the plateau is caused by pressing the button to toggle the intensity back to high.

In Figure 2.4 the power consumption of the LEDs is included. For this graph the same

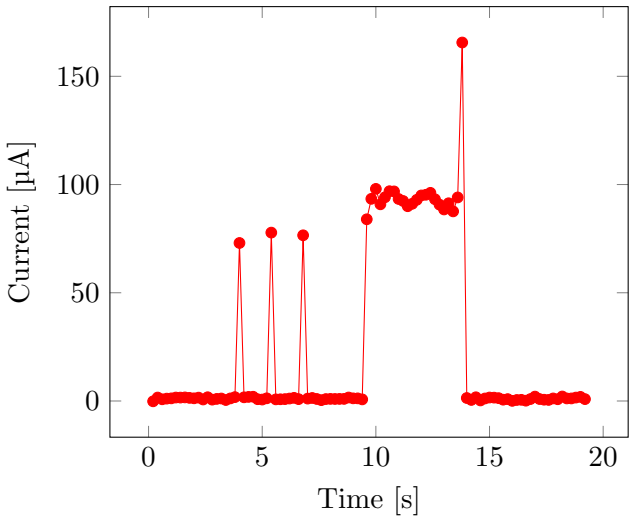

Figure 2.3: Power consumption of the improved solution **without** LEDs.

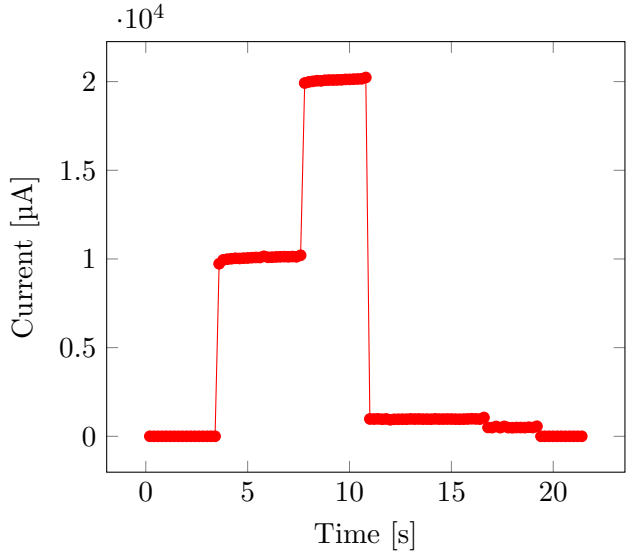

Figure 2.4: Power consumption of the improved solution **with** LEDs.

actions were performed as for Figure 2.3 which is hidden by the high power consumption of the LEDs compared to everything else. It is clearly visible when one (t ≈ 4s) and two LEDs (t ≈ 8s) are lit up. The drop afterwards is again caused by changing the intensity to low. If you look closely you can see that the power consumption is higher than in the beginning. Then follow two small declines which are caused by turning the LEDs off again.

2.3 Solution comparison

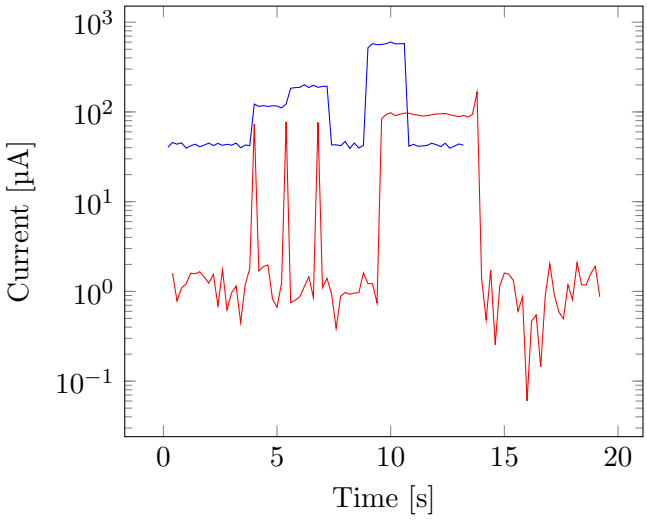

Figure 2.5: Power consumption of both solutions **without** LEDs.

In Figure 2.5 the current consumption of the two solutions is compared. The basic solution (red) has a consumption of about **40 µA**. The improved version (blue) only consumes about **1 µA**. If you consider that the base solution already uses some power saving methods the improved version is about **1000 times more efficient** than using no such options at all. The spikes in the graphs are once again caused by one or more buttons being pressed, except for the plateau caused by the lower intensity LEDs.

If you look at the the isolated number of power consumption, a small standard CR2354 3V battery with a capacity of about 560 mAh could power the microcontroller in deep sleep for more than 50 years and the base version with polling for about 1 to 2 years. Without any power saving techniques the battery would only last about 500 hours or 20 days.