Reykjavík University

Mechatronics I

T-411-MECH

Laboratory 2

05/09/2020

Students:
Giacomo Menchi
Silja Björk Axelsdóttir

Teacher:
Joseph T. Foley

Contents

1 Main Tasks

We started by creating the Lab2 Github repo, where we then uploaded the needed program files. The repo is available at this link:

`https://github.com/ru-engineering/lab-2-group-10`

1.1 Main Task 1

Task: Connect a LED to a GPIO pin and to the oscilloscope (if available).

Photo proof: These two photos below show the output we measured with the oscilloscope.

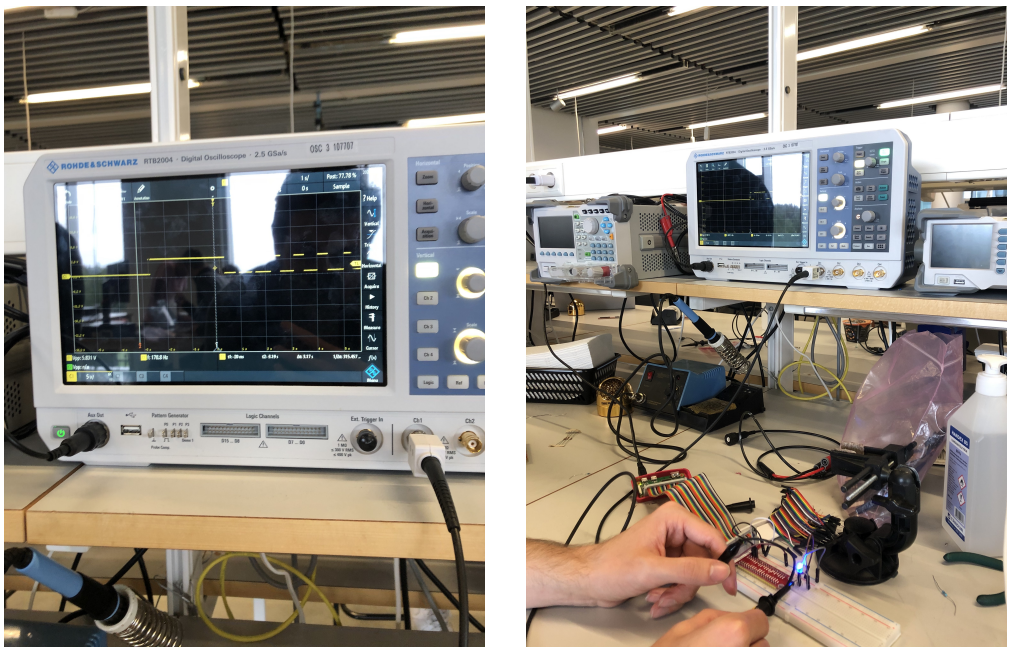

Figure 1: Connecting the led to an oscilloscope.

Circuit schematic: This picture below represents a circuit schematic, which has been simplified from the photo shown above but is completely equivalent.

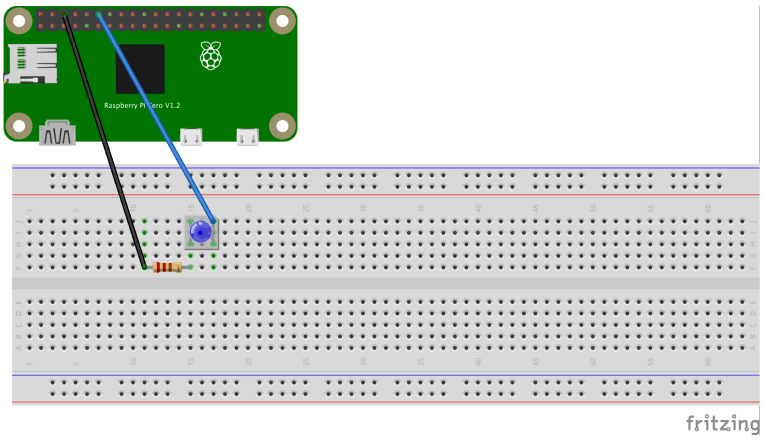

Figure 2: Task 1 circuit schematic.

1.2 Main Task 2

Task: Get the LED to blink with a fixed period.

Video proof: the next video shows how we got the led to blink, the code we used can also be found inside the Github repo, in the file named "Main Task 2.rs".

`https://shorturl.at/ayJN0`

1.3 Main Task 3

Task: Connect the button to a GPIO pin and reduce the signal noise. Observe signal with oscilloscope or AD (if available).

Explanation: We used software debouncing by implementing a double check on the button press, the first immediately and the second after 500 microseconds, which is enough time for the button to get stable and hence fixes the problem (the code for that can be found once again in the Github repo, in the file "Main Task 3.rs".

Circuit schematic: The next picture represents the circuit schematic, once again some connections may be different, but only in cable positioning and ground connections, while the pins are correct.

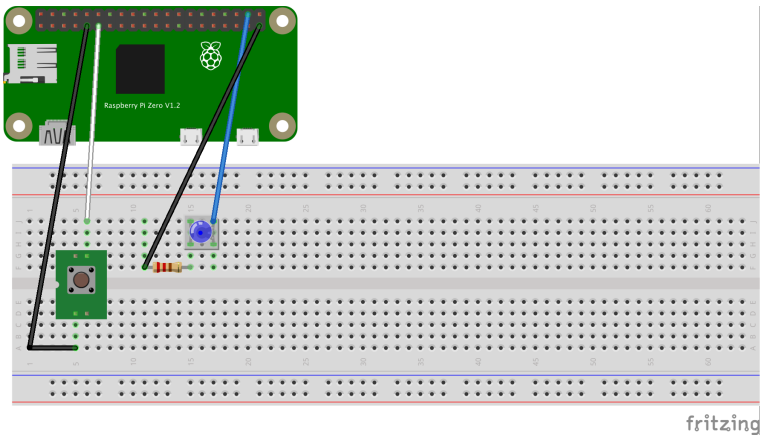

Figure 3: Task 3 circuit schematic.

Video proof: We included a video proof of this task, which is combined with the next two tasks (it can be found below, in the Main Task 5 section).

1.4 Main Task 4

Task: Get the LED to blink once you press the button from 3. (Note, we did not make it clear if it should blink once or many times, so it is up to you).

Video proof: Once again, we merged this video proof with the one from the Task 5, so it can be found in the link below.

1.5 Main Task 5

Task: Write out on the terminal when the button is pressed and when it is depressed again.

Video proof: The next video shows all the requests from task 3, 4 and 5 (which were very similar one another and so we decided to put them together).

`https://shorturl.at/mnsK1`

2 Hard Tasks

2.1 Hard Task 1

Task: Combine Main#1 and Main#4 so they are 2 separate LEDs and work as defined simultaneously (Note: We meant #2 and #4, but if you just used the code from #1, we have to accept it.).

Video proof: Once again, here is the video proof for this task, in this case we merged Main#2 and Main#4, as the correction said.

`https://shorturl.at/tALY4`

2.2 Hard Task 2

Task: Using Rust Enums have the code mark if the pin is input, output or unused, have the enum also store the pin number.

Explanation: As specified by the assignment itself, the code for this task is in the "main.rs" file in the repo. We used the generic type <T> in the enum to make it save every type of input/output (leds, buttons, etc.) and then if let statements to get back data contained inside the enums declared and use it (to check if button was pressed and turn on/off leds).

2.3 Hard Task 3

Task: Use the enum described in Hard#2 to check for errors in using the code. Have it print an error message to the terminal if:

- You tried to write to an Input pin
- You tried to read from an output pin
- You tried to read or write from an uninitialized pin

Video proof: In the link below we posted the video which shows the requests in action. We first tried to read/write from/to some pins to show the checking mechanism in action, then proceeded showing the classical program from previous tasks, this time modified to also include the checking mechanism.

`https://shorturl.at/CJTV0`

3 Advanced Tasks

3.1 Advanced Task 1

Task: Add a STOP button and have the program panic if an error occurred or the STOP button is pressed, have it write out on the terminal what caused the panic.

Video proof: This time, we first created the crash on purpose by calling the read() function on an unused pin, and then removed the call, hence letting the program work correctly up until the pression of the STOP button, which crashed the program again (with a different panic).

`https://shorturl.at/yCUY0`

3.2 Advanced Task 2

Task: Allow the program to continue running if it panics, however, make it print out the error messages to the terminal and bypass the forbidden thing. Have the STOP button still stop the code.

Video proof: This time, we used the panic library functions to bypass the panics when they occurred, so the program still prints the panic and its cause, but keeps going bypassing the crash. The STOP button, instead, still panics and make the program crash, as requested in the task.

`https://shorturl.at/noBKZ`