Reykjavík University

Mechatronics I

T-411-MECH

Laboratory 3

06/09/2020

Students:
Giacomo Menchi
Silja Björk Axelsdóttir

Teacher:
Joseph T. Foley

Contents

1 Main Tasks

We started as usual by creating the Lab3 Github repo, where we then uploaded the needed program files. The repo is available at this link:

`https://github.com/ru-engineering/lab-3-group-10`

1.1 Main Task 1

Task: Create a circuit to the specific requirements given by BILL-Y:

- LEDs
- Buttons
- Off switch
- On light
- Colour coding
- Neat leads

Photo proof: The next image contains the circuit photo, which should comply with all the guidelines given in the assignment specification. The colour code we used is the following:

- **LED cables:** red, white, green (matching led colours)
- **Ground cables:** black
- **Turn off switch cable:** violet
- **Turn off switch led cable:** yellow
- **Two GPIO buttons cables:** blue

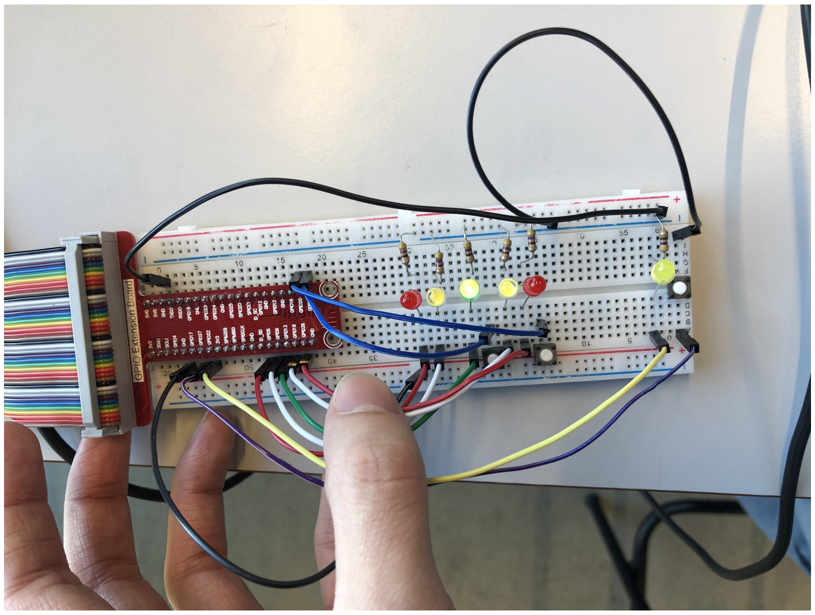


Figure 1: Task 1 circuit photo.

Circuit schematic: It represents the same info as the picture above, but in a nice schema (some ground links were also simplified since the breadboard in the schema is different from the real one).

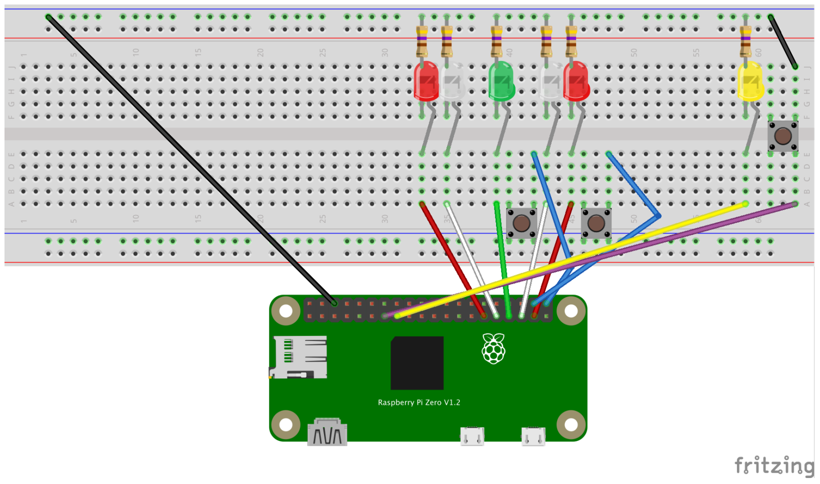


Figure 2: Main Task 1 circuit schematic.

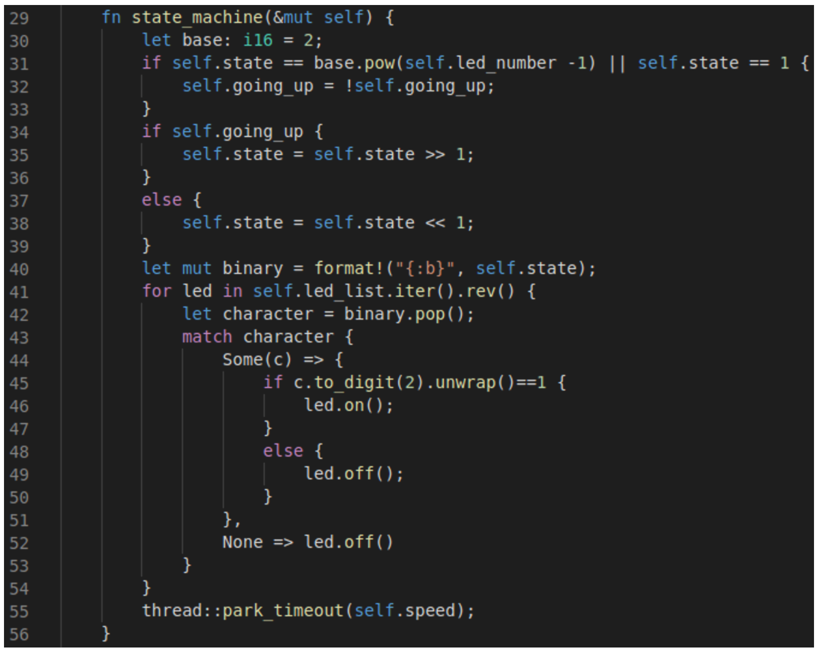
1.2 Main Task 2

Task: Create a function that cycles between the LEDs like a Larson scanner and have it be in a finite state machine.

HINT: have the function need to be passed a number that is the current state and have it return the new state.

Try using bitwise operations if you can, it helps.

Explanation: The screenshot below contains the state machine implementation using bitwise operators. Each state is represented by an integer value which binary equivalent is a 1 followed by some 0s: the 1 represents the lit led, while each zero is an unlit led: counting the 0s we can always understand which led is lit (if the binary value is 1000, we know that the fourth led is lit, because there are three 0s and one 1). Furthermore, the going_up variable is used to keep track of the direction where the leds are moving, so that the state can change accordingly (we either use bitwise left shift or bitwise right shift to add/remove 0s, thus to change state).



```rust
    fn state_machine(&mut self) {
        let base: i16 = 2;
        if self.state == base.pow(self.led_number -1) || self.state == 1 {
            self.going_up = !self.going_up;
        }
        if self.going_up {
            self.state = self.state >> 1;
        }
        else {
            self.state = self.state << 1;
        }
        let mut binary = format!("{:b}", self.state);
        for led in self.led_list.iter().rev() {
            let character = binary.pop();
            match character {
                Some(c) => {
                    if c.to_digit(2).unwrap()==1 {
                        led.on();
                    }
                    else {
                        led.off();
                    }
                },
                None => led.off()
            }
        }
        thread::park_timeout(self.speed);
    }
```

Figure 3: Finite state machine implementation.

Video proof: Since the Main Task 2 and 3 are almost identical, we just made one video for both of them, which can be found in the paragraph below.

1.3 Main Task 3

Task: Have the main() function run this function periodically with a fixed timer.

Video proof: As also stated above, this video summarizes both task 2 and 3.

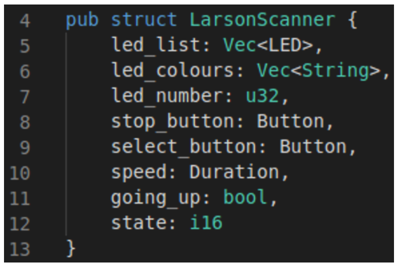
`https://shorturl.at/ervJK`

1.4 Main Task 4

Task: Create a Struct for the lights,

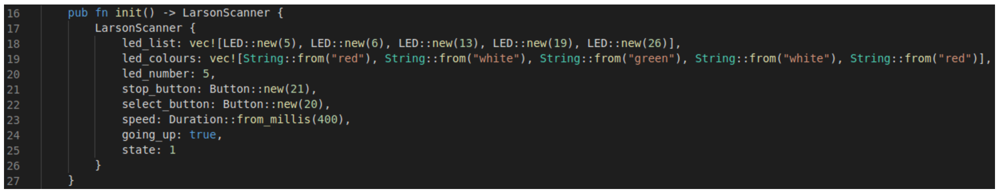
- Have parameters for the state, the pins for the LEDs and number of lights connected
- Create an initialiser function that sets the values of the struct
- Have the function from Main#2 be a method implemented in the struct.
- Modify the function from Main#2 so when it is called using the struct it uses all the included values to run it.

Photo proof: The photos below show how we implemented all this task requests, and each caption contains what the screenshot represents.



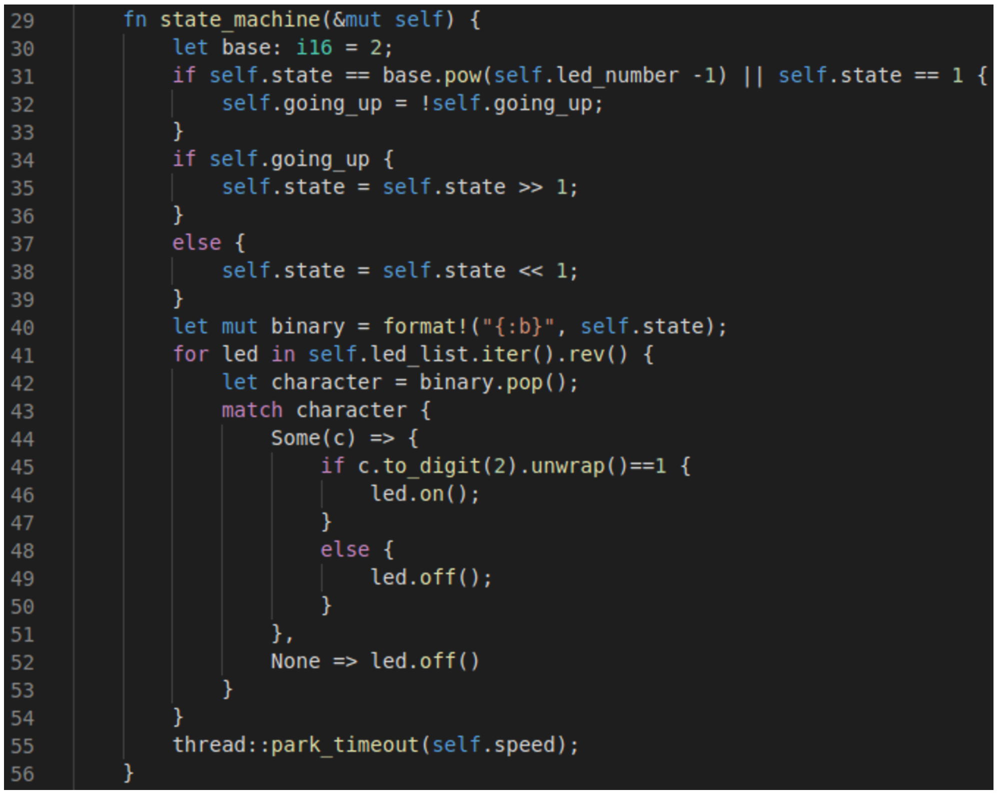
```rust
pub struct LarsonScanner {
    led_list: Vec<LED>,
    led_colours: Vec<String>,
    led_number: u32,
    stop_button: Button,
    select_button: Button,
    speed: Duration,
    going_up: bool,
    state: i16
}
```

Figure 4: Struct declaration with requested parameters.



```rust
    pub fn init() -> LarsonScanner {
        LarsonScanner {
            led_list: vec![LED::new(5), LED::new(6), LED::new(13), LED::new(19), LED::new(26)],
            led_colours: vec![String::from("red"), String::from("white"), String::from("green"), String::from("white"), String::from("red")],
            led_number: 5,
            stop_button: Button::new(21),
            select_button: Button::new(20),
            speed: Duration::from_millis(400),
            going_up: true,
            state: 1
        }
    }
```

Figure 5: Initialiser function.



```rust
    fn state_machine(&mut self) {
        let base: i16 = 2;
        if self.state == base.pow(self.led_number -1) || self.state == 1 {
            self.going_up = !self.going_up;
        }
        if self.going_up {
            self.state = self.state >> 1;
        }
        else {
            self.state = self.state << 1;
        }
        let mut binary = format!("{:b}", self.state);
        for led in self.led_list.iter().rev() {
            let character = binary.pop();
            match character {
                Some(c) => {
                    if c.to_digit(2).unwrap()==1 {
                        led.on();
                    }
                    else {
                        led.off();
                    }
                },
                None => led.off()
            }
        }
        thread::park_timeout(self.speed);
    }
```

Figure 6: Finite state machine implementation (which remained the same from task 2).

2 Hard Tasks

2.1 Hard Task 1

Task: Create a function named stop_lights implemented in the struct that when called stops the finite state machine and prints out on-screen which light it stopped on.

Have the function ask if it wants the user to continue and wait for input (your choice).

Video Proof: The next link contains a little demonstration of the requested function (which implementation can also be found on Github, as usual). In this case, we made the program call the stop_lights function automatically after a little bit that the Larson scanner was running, and used the classical Linux terminal format ([Y/n]) to ask user for input.

`https://shorturl.at/bQ149`

2.2 Hard Task 2

Task: Have the middle button call stop_lights once pressed.

Video proof: Once again, we have a video demonstration. This time, the function is not called after a while, but only when we press the button.

`https://shorturl.at/fpqU3`

2.3 Hard Task 3

Task: Have the program speed up if the user hits the button on the green light and slow down if the user hits the edge lights

Have it ask each time if the user wants to continue or quit.

Video proof: Final video! This shows how the Larson scanner doubles its speed each time we press the button when the green led is on, do nothing when we press it on the white, ask us to continue and in case halve the speed if we press it on the red (that's what we understood from the task, we hope that we got it right). Should we choose not to continue when prompted, of course, the program stops.

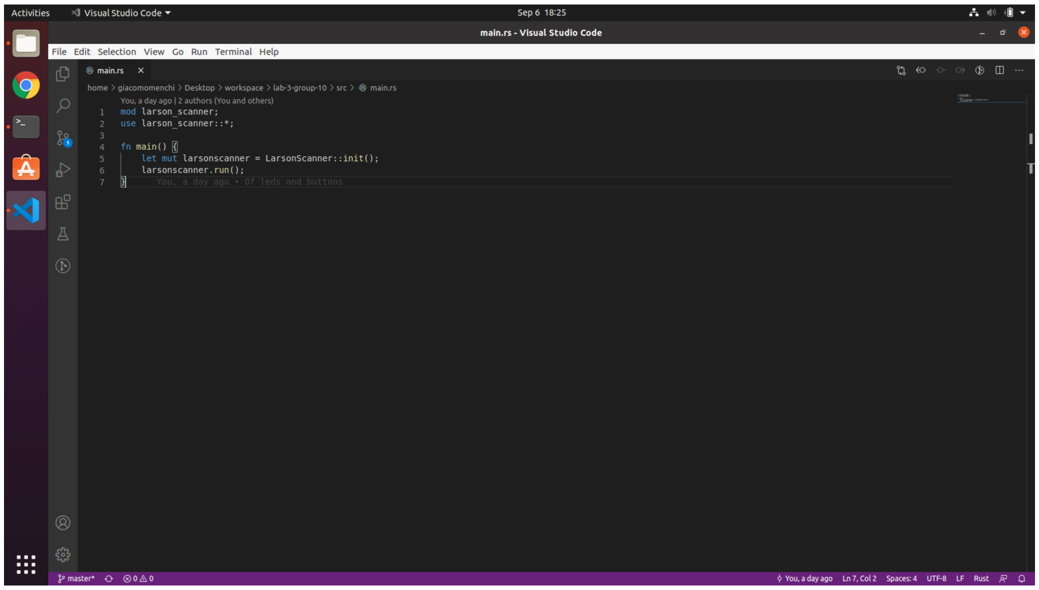
`https://shorturl.at/uxAP1`

3 Advanced Task

Task: Make the code be less than or equal to 5 lines after the "fn main(){" is called in the main file. After a semicolon, a new line is required.

HINT use another file to store all your code

Photo proof: As also suggested by the hint, we stored everything in the "larson_scanner.rs" file and left only the main function inside the main file, then used "mod" to import the external file and make it all work again.


Activities
Visual Studio Code
Sep 6 18:25
main.rs - Visual Studio Code
File Edit Selection View Go Run Terminal Help
main.rs
home > giacomomenchi > Desktop > workspace > lab-3-group-10 > src > main.rs
You, a day ago | 2 authors (You and others)
master*
You, a day ago Ln 7, Col 2 Spaces: 4 UTF-8 LF Rust

```rust
mod larson_scanner;
use larson_scanner::*;

fn main() {
    let mut larsonscanner = LarsonScanner::init();
    larsonscanner.run();
}       You, a day ago • Of leds and buttons
```

Figure 7: Main file after shifting everything to a separate one.