Reykjavík University

Mechatronics I

T-411-MECH

Laboratory 4

08/09/2020

Students:
Giacomo Menchi
Silja Björk Axelsdóttir

Teacher:
Joseph T. Foley

Contents

1 Main Tasks

As usual, the code for this assignment is available at the following Github repo:

`https://github.com/ru-engineering/lab-4-group-10.git`

1.1 Main Task 1

Task: Control a servo using a PWM pin on the RPi, have the servo go from 0° to 180° with a couple of intermittent steps.

Video proof: After a seemingly infinite amount of tries, we managed to find correct values and timings which allowed us to turn the servo by 90 degrees and then by 90 again. This part was done using the PWM library, though it could have been easier to just use the servo library, which has been invented just to deal with servomotors, and which we wrongly assumed we couldn't use.

`https://shorturl.at/fiwFW`

1.2 Main Task 2

Task: Create your own function that moves the servo to a certain position slowly, parse into it the destination and some speed variable.
HINT slow_move(angle : u8, speed : time::Duration)

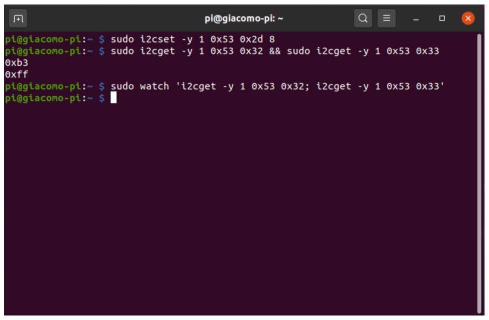
Video proof: This time, we used the servo library which allowed us to make things enormously easier and to keep code more compact. The video demonstration of how this worked out is available below.

`https://shorturl.at/lwH48`

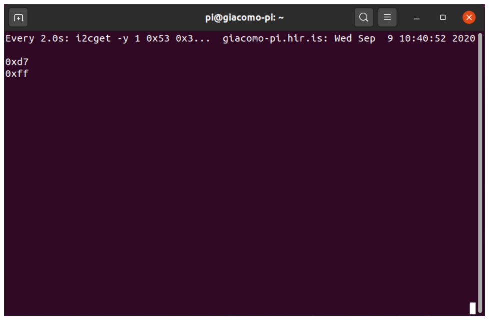
1.3 Main Task 3

Task: Get the values of the x-axis from the accelerometer (ADXL345) using Linux commands.

Screenshot proof: In order to get values from the accelerometer we can just use the Linux terminal commands "i2cset" and "i2cget". The first one is used to set the accelerometer as awake and operational and the second one retrieves data from the accelerometer itself by reading the correct addresses. Furthermore, by using the "watch" command, we can observe how the values change from time to time, without needing to execute the command again multiple times.



```
pi@giacomo-pi:~ $ sudo i2cset -y 1 0x53 0x2d 8
pi@giacomo-pi:~ $ sudo i2cget -y 1 0x53 0x32 && sudo i2cget -y 1 0x53 0x33
0xb3
0xff
pi@giacomo-pi:~ $ sudo watch 'i2cget -y 1 0x53 0x32; i2cget -y 1 0x53 0x33'
pi@giacomo-pi:~ $ 
```



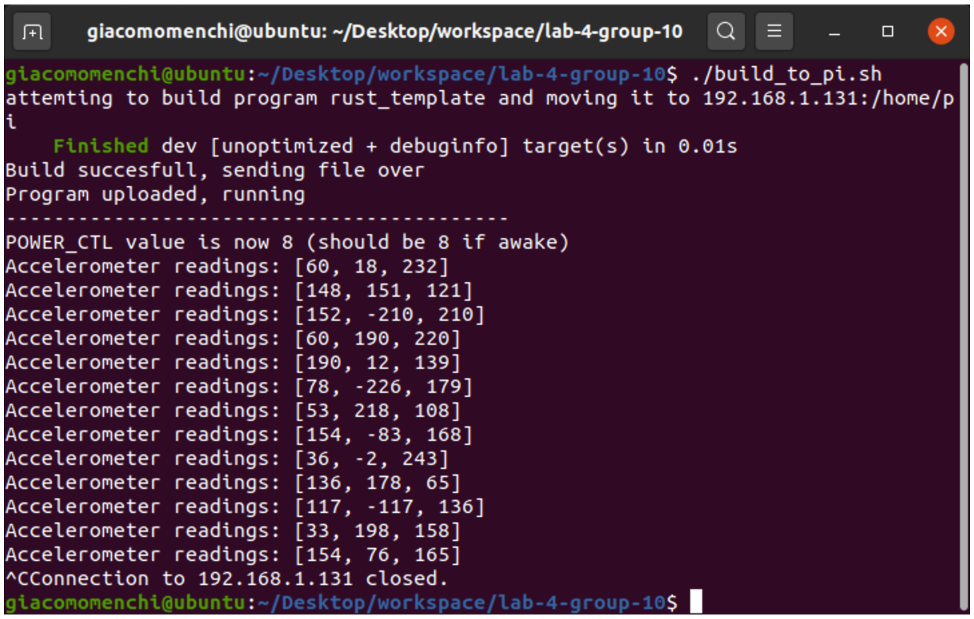
```
Every 2.0s: i2cget -y 1 0x53 0x3...  giacomo-pi.hir.is: Wed Sep  9 10:40:52 2020

0xd7
0xff
```

Figure 1: Retrieving x-asis data once and continuously.

1.4 Main Task 4

Task: Print the axis data onto the terminal.

Photo proof: The following screenshot demonstrates getting the same data we got beforehand, plus the other two axes, using rust integrated libraries and code; data has also been elaborated to return a single value for each axis, instead of a couple.



```
giacomomenchi@ubuntu:~/Desktop/workspace/lab-4-group-10$ ./build_to_pi.sh
attemting to build program rust_template and moving it to 192.168.1.131:/home/p
i
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
Build succesfull, sending file over
Program uploaded, running
-----------------------------------------
POWER_CTL value is now 8 (should be 8 if awake)
Accelerometer readings: [60, 18, 232]
Accelerometer readings: [148, 151, 121]
Accelerometer readings: [152, -210, 210]
Accelerometer readings: [60, 190, 220]
Accelerometer readings: [190, 12, 139]
Accelerometer readings: [78, -226, 179]
Accelerometer readings: [53, 218, 108]
Accelerometer readings: [154, -83, 168]
Accelerometer readings: [36, -2, 243]
Accelerometer readings: [136, 178, 65]
Accelerometer readings: [117, -117, 136]
Accelerometer readings: [33, 198, 158]
Accelerometer readings: [154, 76, 165]
^CConnection to 192.168.1.131 closed.
giacomomenchi@ubuntu:~/Desktop/workspace/lab-4-group-10$
```

Figure 2: Printing x/y/z axes data on terminal.

1.5 Main Task 5

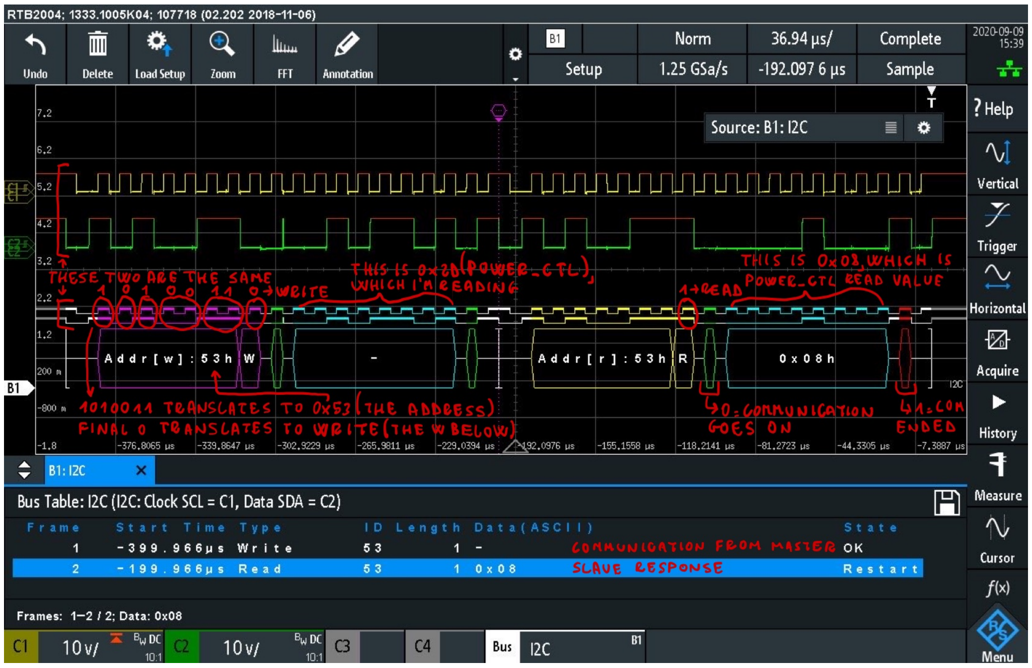
Task: Use an Oscilloscope or AD2 to examine the I2C signal. In the case of unavailability of oscilloscope use the following picture. Mark with a short description text how each byte being read into value, how it starts and ends. Try to describe what each of these bytes do.

Photo proof: The next picture is a screenshot we took of the oscilloscope while connected to the accelerometer circuit. As also written in the image, the C1 and C2 waves in the top half are equal to the waves represented just above the B1 representation, in the bottom half. Each segment of the I2C communication is composed by three parts:

- The address, which contains the binary value for the address to read or write.
- The read/write bit, which is 0 if we want to write or 1 if we want to read.
- The actual data to read/write, again in binary.

In our case, the screenshot represents an attempt to read the i2c device status via POWER_CTL value (which, as explained in other assignments, is 0x08 if the device is in "Wakeup" mode), so we first write 0x2D on 0x53 address, to make it understand that we would like to read that status, and then receive a "Restart" (a slave response) with the same address and the read value, which is 0x08.

Finally, each section of the i2c communication is separated using one bit, which is 0 if there is more to the communication, or 1 if the communication is complete (notice: the separator is not there between the address and the read/write bit).



Bus Table: I2C (I2C: Clock SCL = C1, Data SDA = C2)

Frame	Start Time	Type	ID	Length	Data(ASCII)		State
1	-399.966µs	Write	53	1	-	COMMUNICATION FROM MASTER	OK
2	-199.966µs	Read	53	1	0x08	SLAVE RESPONSE	Restart

Frames: 1–2 / 2; Data: 0x08

Figure 3: Oscilloscope screenshot with written explanation.

2 Hard Tasks

2.1 Hard Task 1

Task: Control the servo motor with data from the accelerometer of your own choice.

Video proof: The video link below shows the servo controlled using the accelerometer data. The code was not perfect at the time, so the servo ignores some inputs in the video, but this was fixed in the hard task 2 (which is an evolution of this one), so check out that other video too.

`https://shorturl.at/kDW08`

2.2 Hard Task 2

Task: Make a haptic control scheme using the accelerometer to control the servo motor

- Have the servo motor respond to you tilting your hand left to right so when you tilt your hand one direction the servo follows
- Have the servo motor respond to you tilting your hand forward and backwards so the motor moves slower to the final value when you tilt it backwards and faster when you tilt it forwards.

HINT: Roll and Pitch

Video proof: After completing hard task 1, we rewrote the servo motor code to make it work better, and then added the features requested.

`https://shorturl.at/dezJ8`

3 Advanced Tasks

3.1 Advanced Task 1

Task: Attach another servo and another accelerometer and have the 2 sets be useable at the same time.

Video proof: Again, the video below shows the two sets of accelerometers and servos working together.

`https://shorturl.at/wEPY0`

3.2 Advanced Task 2

Task: Create a physical crane arm that you can attach to the servos to be able to grab something and demonstrate that it works.

NOTE: This can be made from non-fancy things like cardboard and foam-core. Get creative.

Video proof: For this last step, we tried using everything we found inside the lab to create the crane arm (which were very limited resources): cardboard for the actual arm, a magnet to grab objects, a screw screwed to a servo to make the arm move left and right, another servo bound to a string to make the crane magnet go up and down. In order to make it work, we used a trimmer with an a2d converter (it's the only way to make a trimmer work with Raspberry Pi) to move the crane left and right (first "screwed" servo) and a button to make the second servo (the one connected to a string) turn by 180 degrees, thus raising and lowering the magnet.

`https://shorturl.at/cknE2`