Reykjavík University

Mechatronics I

T-411-MECH

Individual Assignment 4

07/09/2020

Students:
Giacomo Menchi

Teacher:
Joseph T. Foley

Contents

1 Tasks

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

`https://shorturl.at/jkuDR`

1.1 Task 1

Task: What is I2C? Compare it to SPI. Mention following parameters: wires, data rate, power, supported devices.

I2C: It's a serial communication protocol (Inter integrated circuit) which can allow us to create a master-slave relationship between devices with different speed: it uses just two wires, one for the data (SDA) and one for the clock synchronization (SCL), can handle more than one master but is limited in slaves number, is flexible but also slow (slower devices slow down the faster ones, so data rate is almost always very limited 400 kbit/s), implements error handling via ACK/NACK (message acknowledged or not acknowledged) but consumes a lot of power (it has open-drain topology) and operates only close distance.

SPI: It's a full-duplex communication protocol (Serial peripheral interface) that manages master-slave relationship between one master and more than one slave: it uses four wires, one for the clock (SCLK, shared by the master), one from master to slave (MOSI), one from slave to master (MISO) and one to select the line to use to communicate with the slave (CS). SPI allows continue stream of data at faster speed (almost doubled if compared to I2C), simultaneous communication on both ends (full-duplex), simple implementation and low power usage, but also complex wiring and no error checking.

1.2 Task 2

Task: What is PWM and what is it used for.

PWM: Stands for pulse-width modulation, thus a technique which reduces the average values of voltage and current by separating the power signal in discrete sections, thus creating more optimized duty cycles (a duty cycle is a percentage indicating how much time the power signal is on and how much is off). This can be used in a lot of different ways, such as:

- **Power control**, since limiting the power delivery means avoiding wasting power due to cables internal resistance.
- **Voltage control**, because creating a more appropriate duty cycle means leveling voltage to a better level.
- **LEDs brightness control**, since by applying a discrete power impulse, we can make the LED start from an off status, slowly raise brightness to its maximum level and then dim again until it's completely off.

1.3 Task 3

Task: What is threading, when should it be used and when should it not be used?

Threading: A thread is a set of instructions to be executed in a computer as part of a software. When said software is executed, a process representing it is created, which uses a certain amount of memory and contains all the variables/objects/data declared and used inside the program itself: this process has the useful ability that it can split itself in different threads, which can execute different portions of code simultaneously and by also sharing aforementioned variables and other data. Threads mechanism should be used whenever different parts of the same software can run independently without interfering with each other: in this case, we benefit from executing them in concurrency (in parallel, using multiple threads), since running all the instruction sequences one after the other would take significant more time. Threads are not suggested in practically any case in which they are not necessary for implementation or execution speed purposes: bugs can be very difficult to track since threads execution is chaotic (depends on a lot of factors, like operating system state, CPU workload, memory availability, etc.) and hence the result obtained can change every time; furthermore, they

are surely useless in single core processors, since one core can execute only one thread at a time.

1.4 Task 4

Task: Create a code that demonstrates PWM by making a LED slowly increase in brightness and slowly fade out

Video proof: I made a video demonstrating how the circuit works and acts, the code is also available on Github in "main.rs" (it's present under "//Task 4" comment).

`https://shorturl.at/coIKL`

1.5 Task 5

Task: Create a code that demonstrates reading I2C by getting the ID of the accelerometer

NOTE; read the datasheet to find out how

Screenshot proof: The screenshot below includes the code I used (also available on Github) and the result I got (the accelerometer ID is 229).

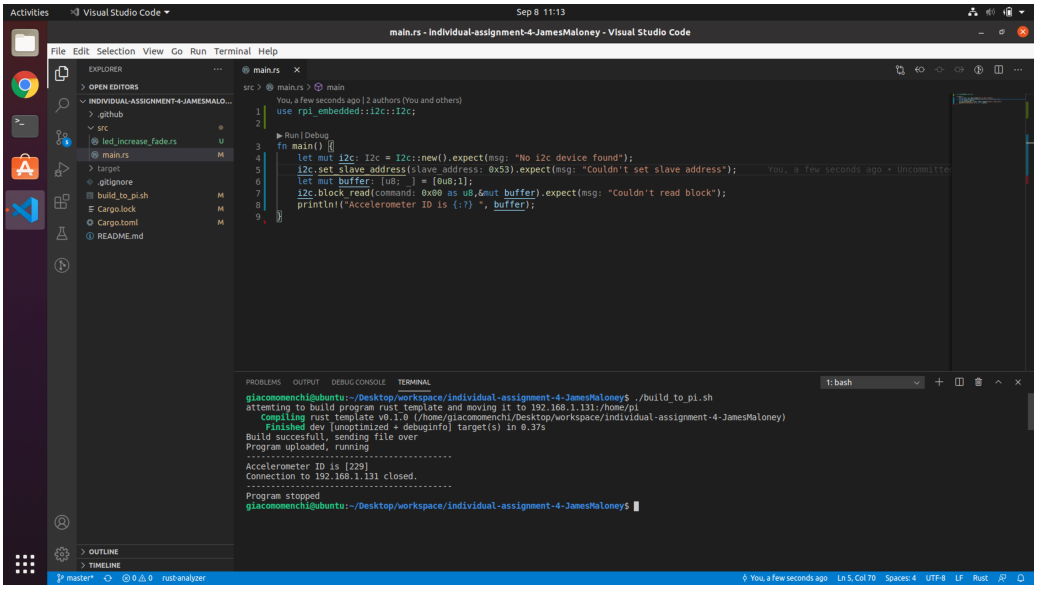

Figure 1: Task 5 code and execution.

1.6 Task 6

Task: Create a code that demonstrates writing I2C by setting the POWER_CTL register on the accelerometer to Wakeup

Video proof: As before, the code is in the "main.rs" file on Github, while here is a video proof of the task completed.

`https://shorturl.at/ikH48`

1.7 Task 7

Task: Create a thread and have the LED code in it. Meanwhile have the accelerometer code in another thread read data and print it out on the terminal

Explanation: I merged this task with the next one since they were very similar one another and asked almost the same thing, so check below for a video proof.

1.8 Task 8

Task: Have the LED react to the I2C data. Use communication between threads

Video proof: The link below demonstrates both task 7 and 8. I got data from accelerometer and used it to dynamically turn on the led: the code used to turn on the led is the same as in step 4, so the more the accelerometer is tilted, the brighter the led will be.

`https://shorturl.at/ginGU`

1.9 Task 9

Task: Have the LED thread panic after 20 seconds and have the I2C thread stop after 30 seconds, have the main code detect which thread panicked and which thread exited successfully, have the main code also print the runtime of each thread.

Video proof: Once again, this video proofs the completion of task 9, with the led thread panicking after 20 seconds and the other thread, the one containing the i2c code, stopping after 30 seconds.

`https://shorturl.at/svzPW`