Lab 6

October 5, 2021

Delivery date: October 5th, 2021

C Programming and I2C interfacing in Arduino Zero

Javier Mondragon Martin del Campo

A01365137

Prof. Matías Vázquez Piñón
Tecnológico de Monterrey

1 Introduction

1.1 Explain what you did in this laboratory.

During this lab I develop a program to communicate between two protocols, UART and I2C. To get to this point I had to understood the configuration and the structure of the micro-controller, the I2C protocol, RTC registers and the UART configuration for the FTDI. For this, there was a translation from assembly code to C and some test codes to know if both protocols were working correctly

1.2 Include a brief explanation of each .C file written for your project.

The first C code was a translation from assembly to C using structures and models given by C architecture and SAM library. The code generates a square signal and sends this signal to the PA14 pin. Using flags provided by the manufacturer, the code transfer from a long and illegible list of instructions to a brief simple code easier to read.
The second one was a testing file to prove the RTC works correctly and how you can connect via I2C protocol with this device.
The last one was a code developed with a custom library to communicate to the RTC. The code combines the second code with a third one provided by the practice and prints the entire date with a formatted string.

2 Results

2.1 Part I

2.1.1 Explanation of the code section where the polling operation takes place. Include an image of the generated waveform.

The probing operation is generated by the next part of the code:

```
while(1){
        if(TC3->COUNT8.INTFLAG.reg != TC_INTENCLR_OVF){
                PORT->Group[0].OUTTGL.reg = PORT_PA14;
                TC3->COUNT8.INTFLAG.reg = TC_INTENCLR_OVF;
                TC3->COUNT16.COUNT.reg = TIMER_INIT_COUNT;
        }
}
```

The loop is for keeping alive the signal. Inside it, a flag is checked in order to restore the settings if the counter have been overflow. If thats the case, the oscillator is mapped to port PA14, then the flag is reset and the init count of the integrated tick counter. The following image is taken from an oscilloscope at the lab:

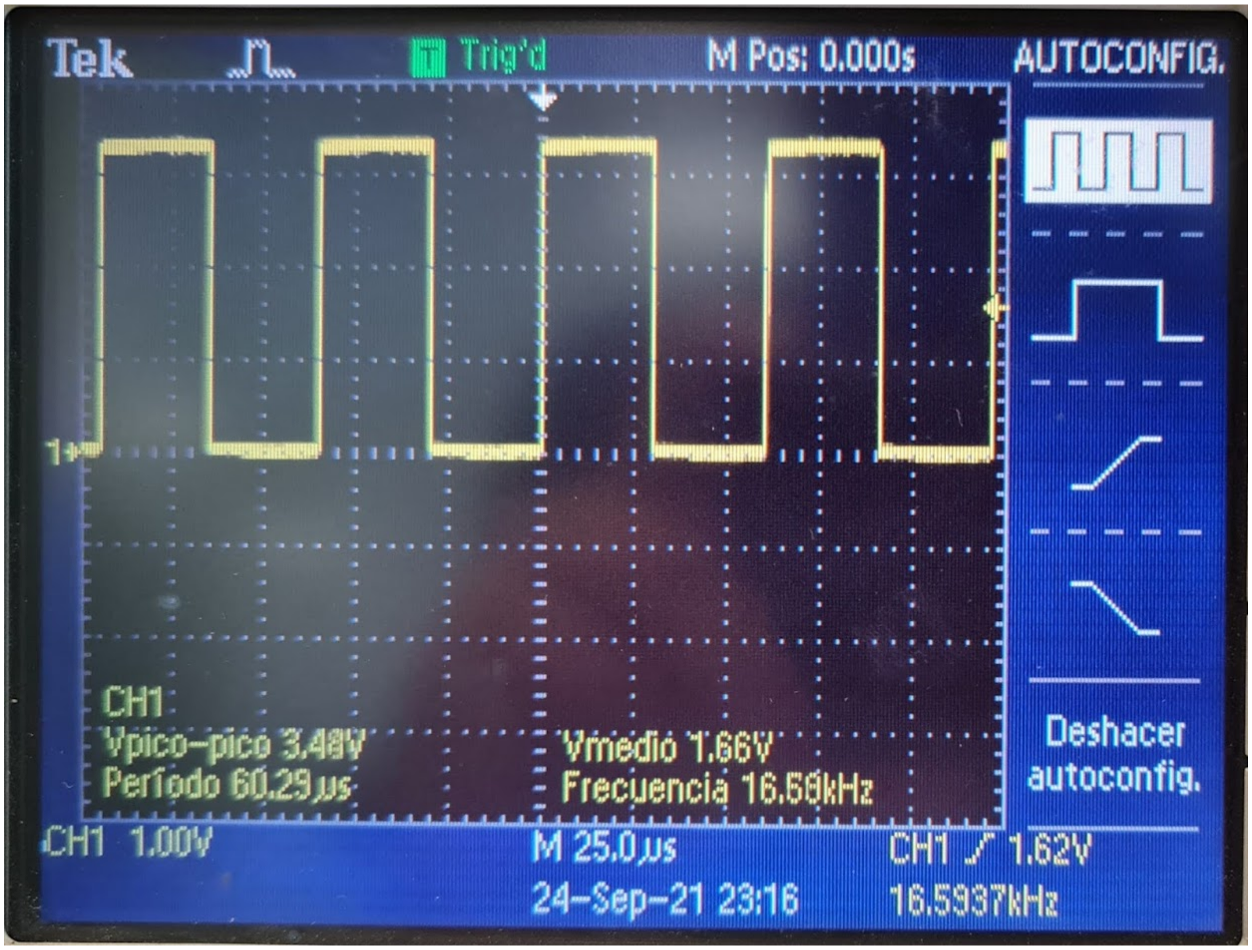


Figure 1: Generated waveform at Lab

2.2 Part II

2.2.1 Explanation of the procedure to configure I2C peripherals. List the different micro-controller registers, the values assigned to each register and the corresponding configuration. Draw an activity diagram with the configuration procedure.

First switch to 8MHz clock, then the port mux configuration takes place setting the pins PA22 and PA23 for SDA and SCL signals, then set a configuration for the PMUX and apply a mask to APBCMASK to enable certain bits. The GCLK configuration for sercom3 module is set and the SERCOM3 I2C module. Then calculate and set the BAUD rate to finally enable all modules to start transmission. The registers for the configuration are:

- OSC8M
- PIN_PA22
- PIN_PA23
- PMUX[11]
- APBCMASK
- CLKCTRL
- GENCTRL
- CTRLB
- BAUD
- CTRLA
- STATUS

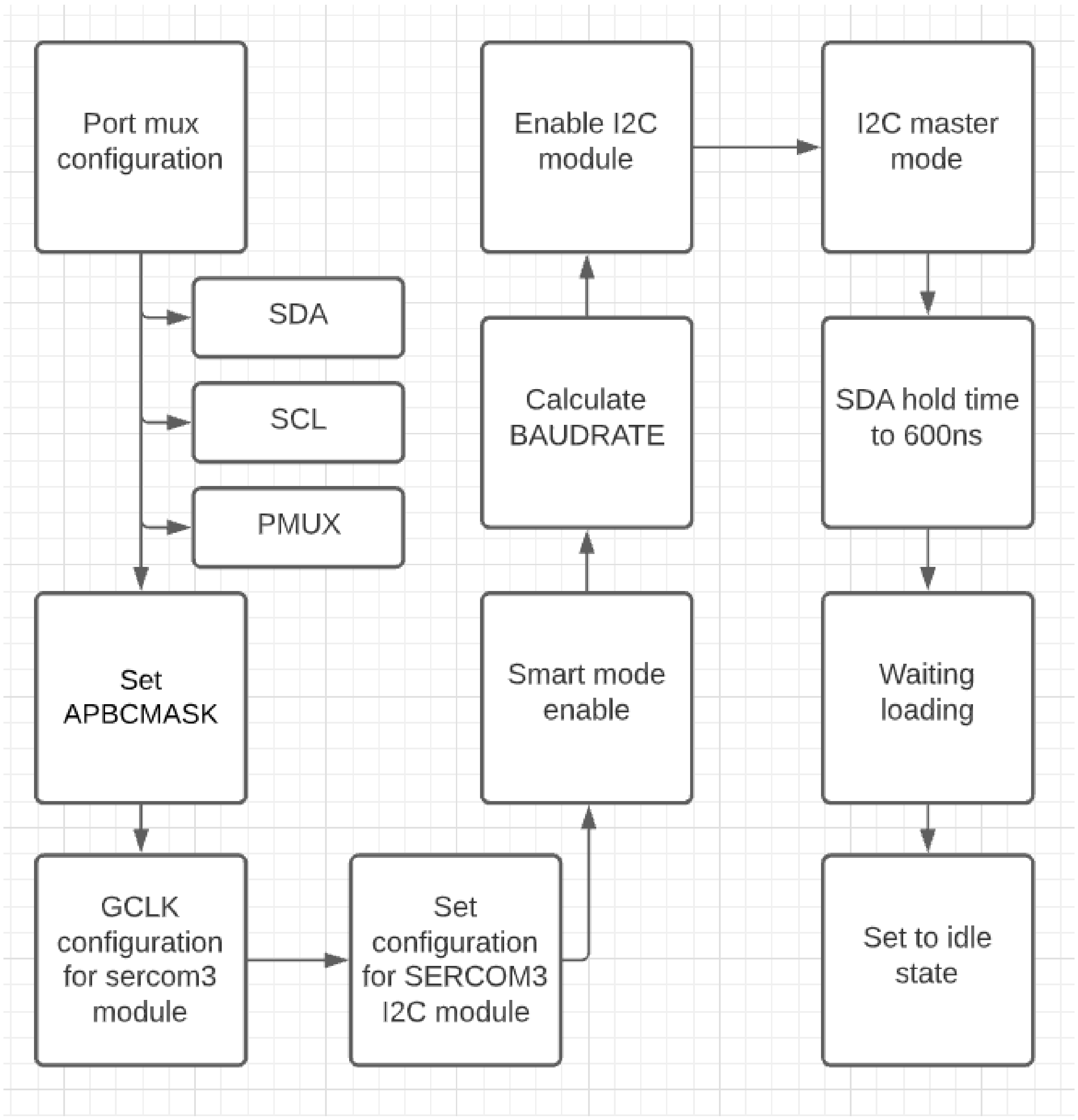


Figure 2: Diagram

2.2.2 Answer to the following questions

1. What is the ADDRESS of the RTC? The default address is 1101000b or 0x68h

2. Find in the datasheet the packet sequence to write to the slave. Report a drawing of this transfer packet. The following diagram shows the write sequence:

Figure 4. Data Write—Slave Receiver Mode

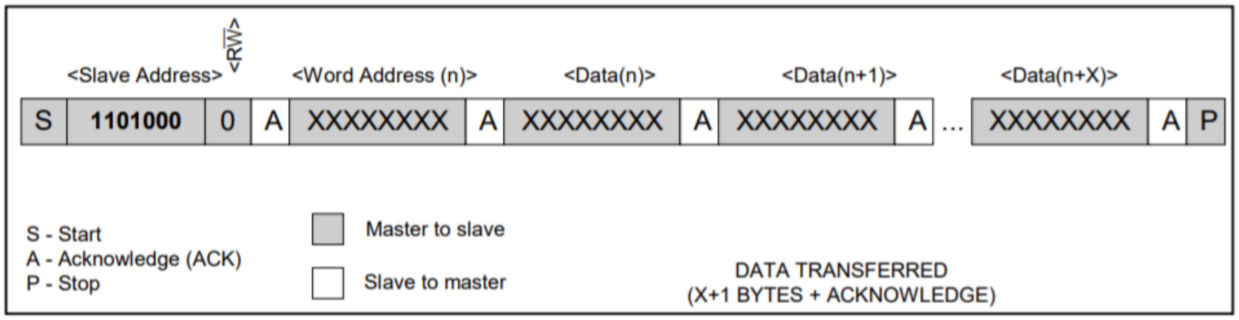


Figure 3: RTC Write sequence

3. Find in the datasheet the packet sequence to read from the slave. Report a drawing of this transfer packet. The following diagram shows the read sequence:

Figure 5. Data Read—Slave Transmitter Mode

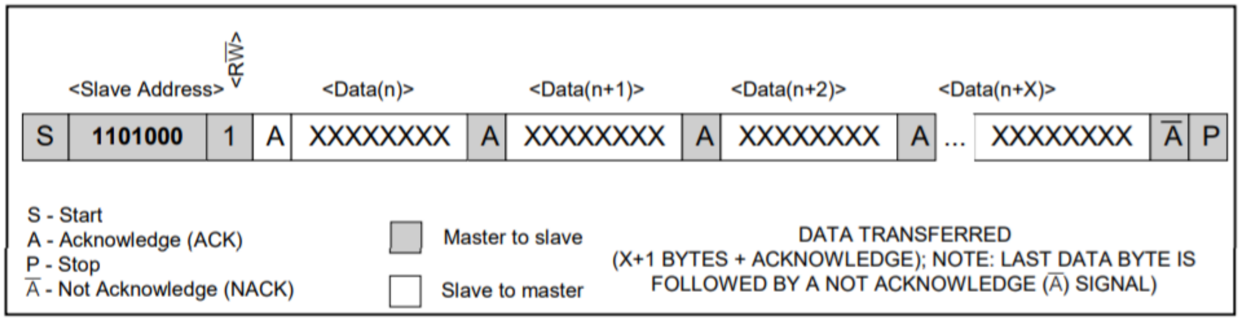
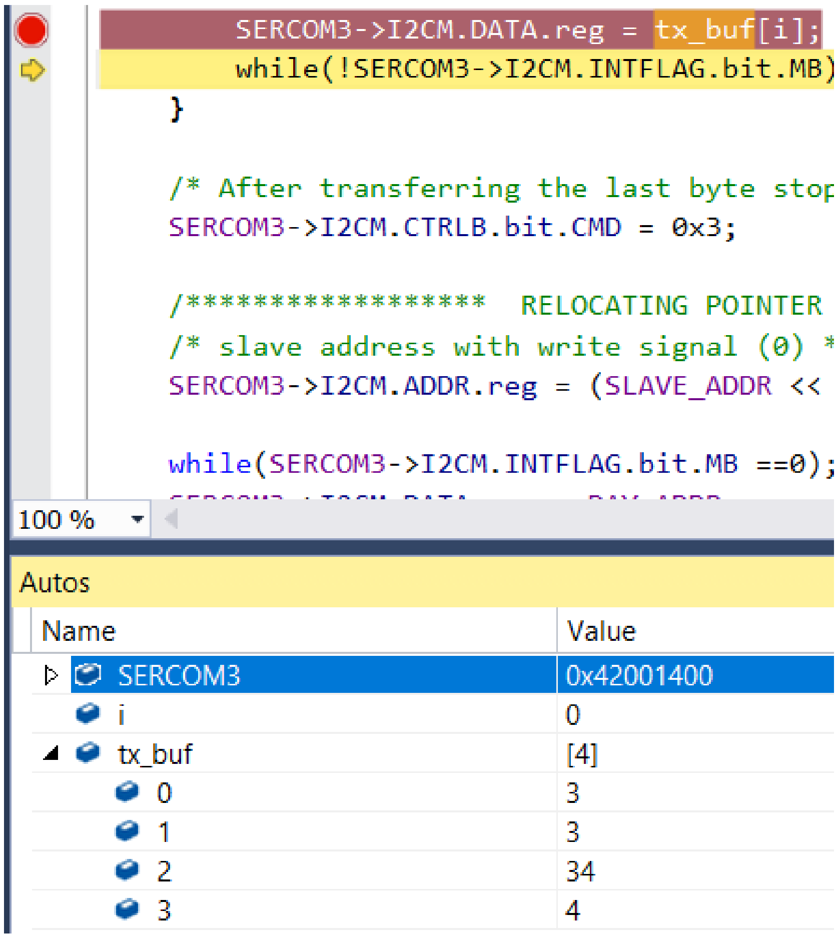


Figure 4: RTC Read sequence

2.2.3 RTC I2C Program

- After executing the rtc_12c.c program step by step and before starting the "sending sequence", what are the values "to be sent" to the RTC?. These values are located at the tx_buf[] buffer. The values located in tx_buf[] are:

  1. 3 (POINTER)
  2. 3 (DAY OF WEEK)
  3. 34 (22th DAY OF MONTH)
  4. 4 (NUMBER OF MONTH)

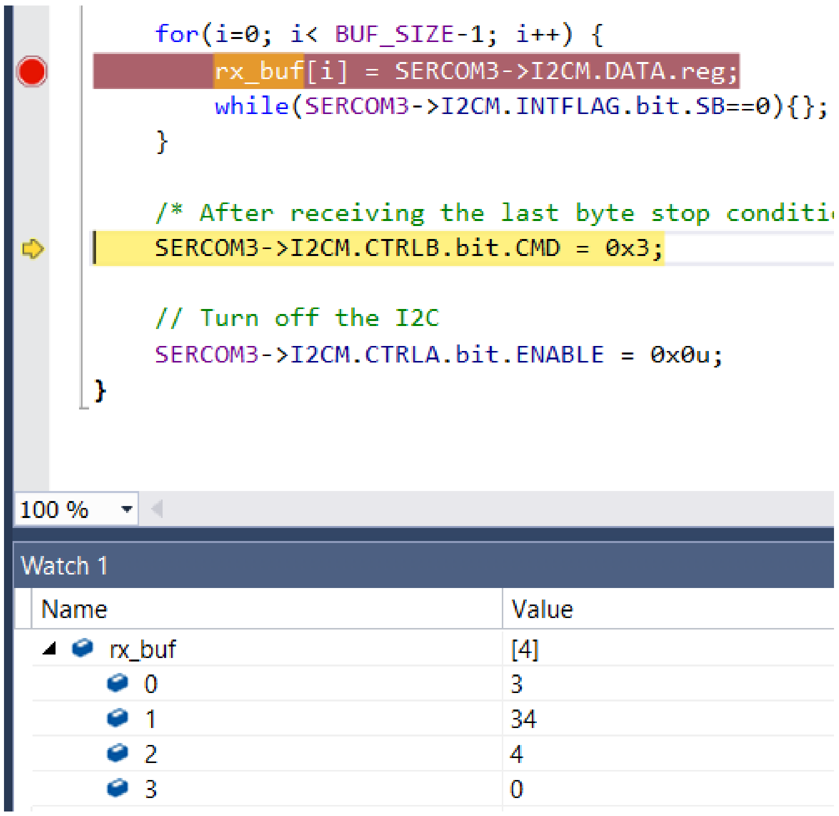


```
        SERCOM3->I2CM.DATA.reg = tx_buf[i];
        while(!SERCOM3->I2CM.INTFLAG.bit.MB)
    }

    /* After transferring the last byte stop
    SERCOM3->I2CM.CTRLB.bit.CMD = 0x3;

    /******************  RELOCATING POINTER
    /* slave address with write signal (0) *
    SERCOM3->I2CM.ADDR.reg = (SLAVE_ADDR <<

    while(SERCOM3->I2CM.INTFLAG.bit.MB ==0);
```

Name	Value
SERCOM3	0x42001400
i	0
tx_buf	[4]
0	3
1	3
2	34
3	4

Figure 5: Debugging tx_buf

- Review the code and explain using an activity diagram the steps needed to write values to the RTC and the steps needed to read information from the RTC. Report these diagrams.
- What memory locations of the RTC were written? and what are the values at these locations after completing the "sending sequence"? Report this findings. The locations written are 0x03h, 0x04h and 0x05h, being these locations for day of the week, day of the month and number of the month respectively. The values written were 0x03h, 0x22h and 0x04h. These values represent Tuesday (supposing the first day of the week is sunday) April 22th.
- After executing the "receiving sequence", were the values correctly recovered?. Report the recovered values. Yes, they were:



```
        for(i=0; i< BUF_SIZE-1; i++) {
            rx_buf[i] = SERCOM3->I2CM.DATA.reg;
            while(SERCOM3->I2CM.INTFLAG.bit.SB==0){};
        }

        /* After receiving the last byte stop conditi
        SERCOM3->I2CM.CTRLB.bit.CMD = 0x3;

        // Turn off the I2C
        SERCOM3->I2CM.CTRLA.bit.ENABLE = 0x0u;
    }
```

```
Name                Value
◢ rx_buf            [4]
    0               3
    1               34
    2               4
    3               0
```

Figure 6: Debugging rx_buf

- What is the purpose of the code "RELOCATING POINTER BEFORE RECEIVING"? If you write comments to this code, is the code working as expected? Report your explanation and findings.
  The comments do not affect the operation. About the pointer, in order to start reading at the 3rd address, a written instruction of the address following to the reading instruction is required for doing that operation. The next figure obtained from the datasheet shows the diagram operation:

Figure 6. Data Read (Write Pointer, Then Read)—Slave Receive and Transmit

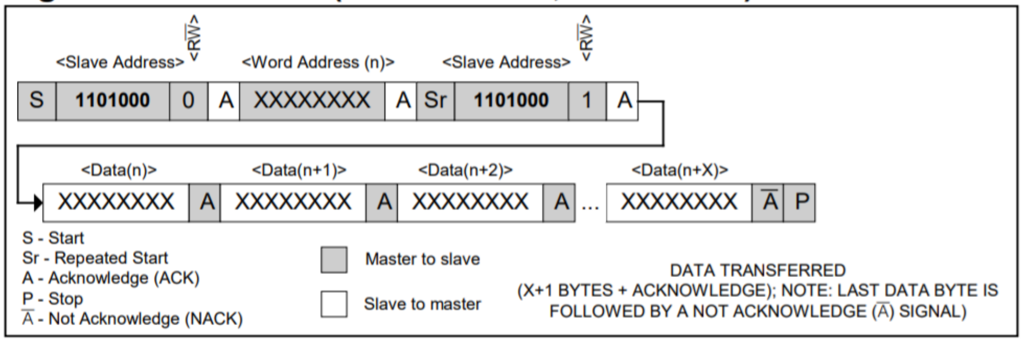
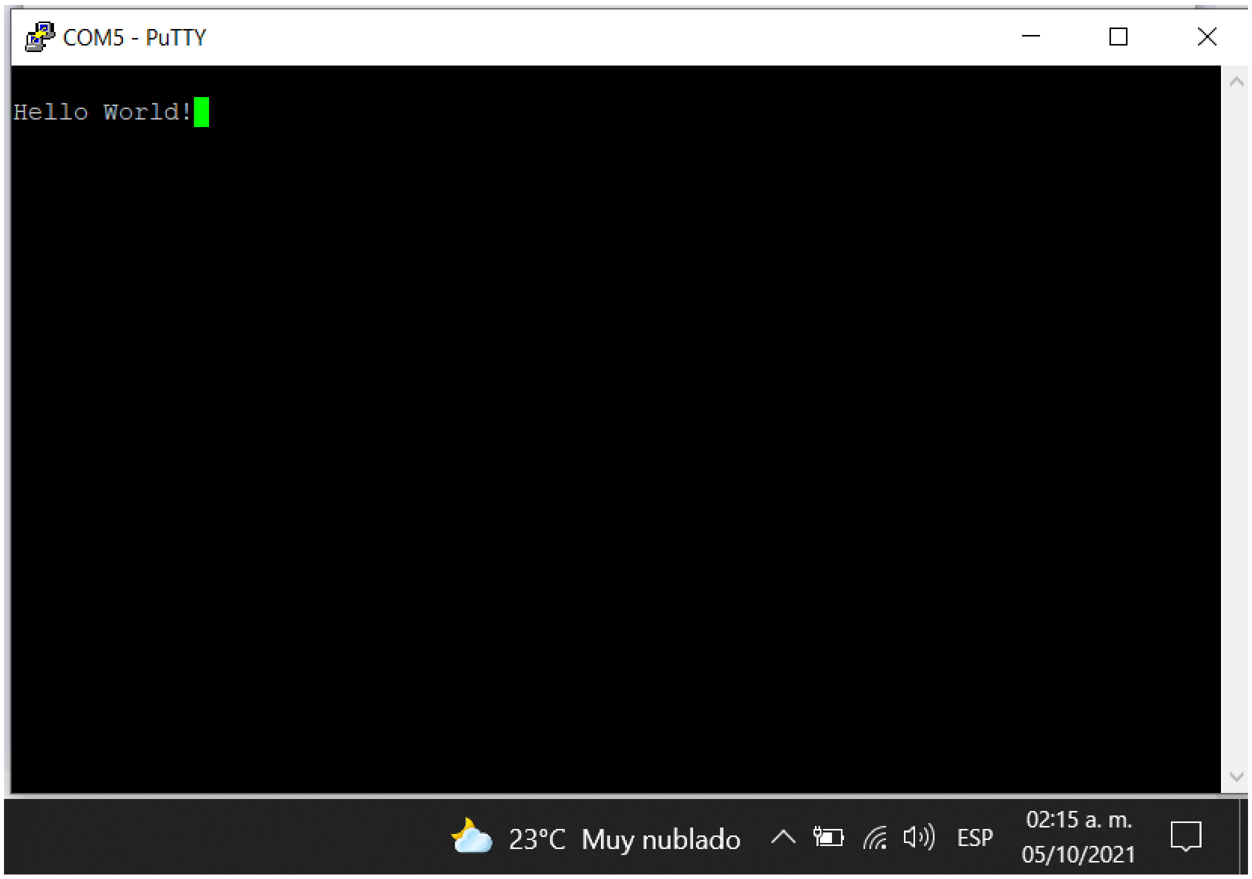


Figure 7: RTC Read sequence

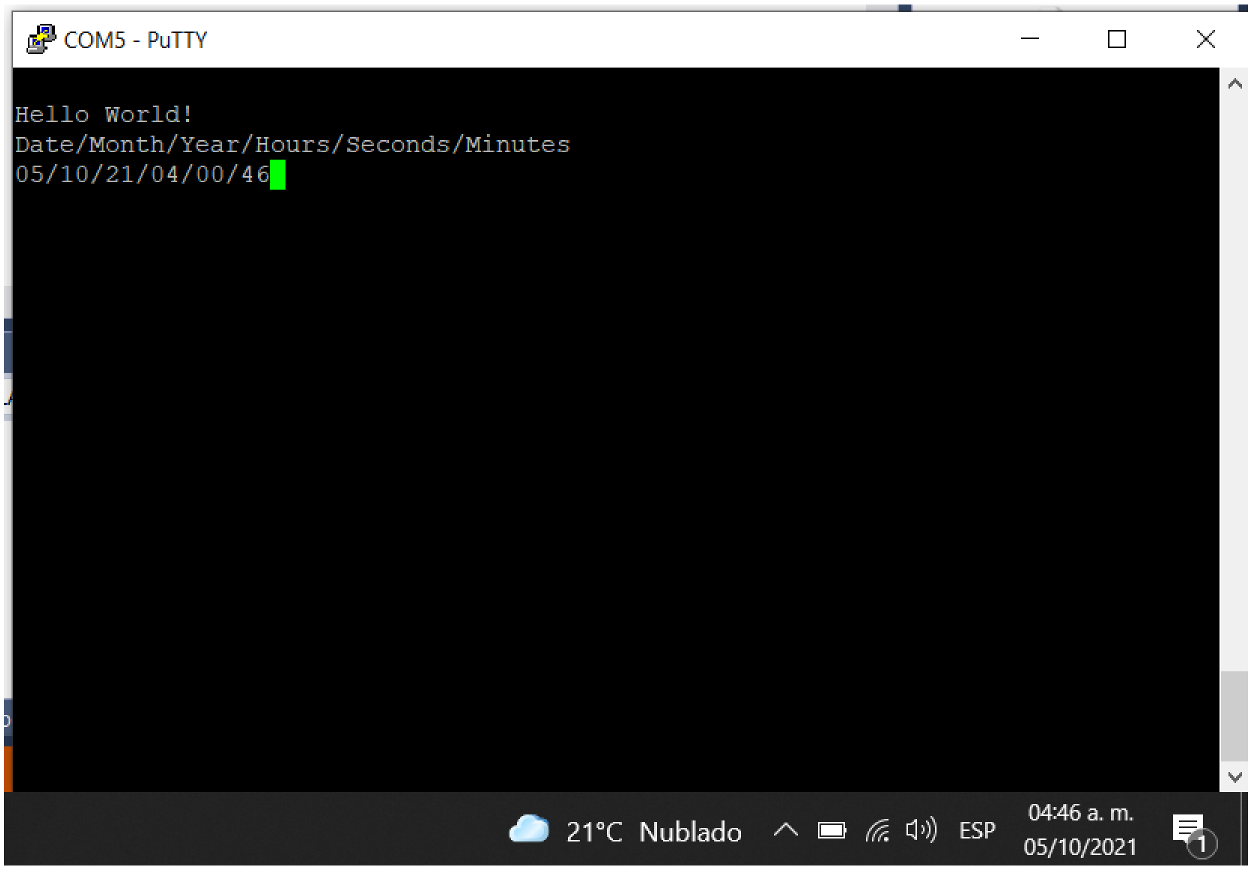
2.2.4 Screenshot of the ”Hello World” line displayed on the Terminal.



```
Hello World!
```

Figure 8: RTC Read sequence

2.2.5 Screenshot of the myprintf function printing the initialized RTC with the current time and date, at the PuTTY terminal.



```
Hello World!
Date/Month/Year/Hours/Seconds/Minutes
05/10/21/04/00/46
```

Figure 9: RTC read from UART

3 Conclusions

The configuration and understanding of the code is essential for developing embedded systems. Since the I/O sets the configuration with memory registers, the data-sheet is required to know such registers. It is tedious to translate from one language to another, but the knowledge acquired to do it is a more practical way to understand how the micro-controller works. The combination of FTDI and the I2C protocol is a great way to debug and the RTC communication is a practice from the last lab.

4 Appendix A

- Link to GCC C Executable project for Part I with the translation from Assembly to C of the square wave generation rutine: `https://github.com/javiermomc/Sistemas_Embebidos/tree/main/L06/Timer`
- Link to GCC C Executable project for Part II with code for serial communication and RTC initialization and control: `https://github.com/javiermomc/Sistemas_Embebidos/tree/main/L06/RTC_COM`

5 Bibliography

1. https://github.com/matias-vazquez/SistemasEmbebidos