Universidade de Aveiro
Mestrado em Cibersegurança
Código: 41794 - Engenharia Reversa
Responsible: João Paulo Silva Barraca

Pic32 Reversing

Friday 14th June, 2024
Ricardo Covelo (102668) - Telmo Sauce (104428)

Contents

1 Introduction

It was proposed to us by the professors of the course to extract the maximum amount of information about the inner workings of a Pic32 controller.

First, the program needs to capture the signals at the right time. To avoid missing any signal changes, we decided through trial and error to measure at 1 MHz for 10 seconds, resulting in 10 MHz.

Figure 1: Samples

2 I2c

We began by measuring the I2C pins to determine which one was the clock and which one was the data. As shown below, we concluded that the left sensor has the address ”0x4D” and the right sensor has the address ”0x48”, this was accomplished by seeing which temperature was being read on the respective address.

Additionally, we observed that each sensor’s address was first written with ”00” to reset the value at that address before reading the temperature value.

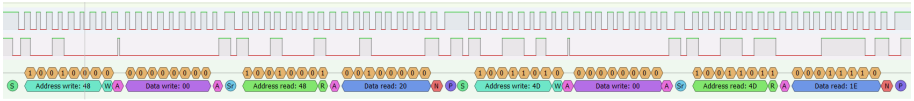

Figure 2: i2c Capture (communication of the controller with the câmeras

Then we measured the period, it measured 5.50us

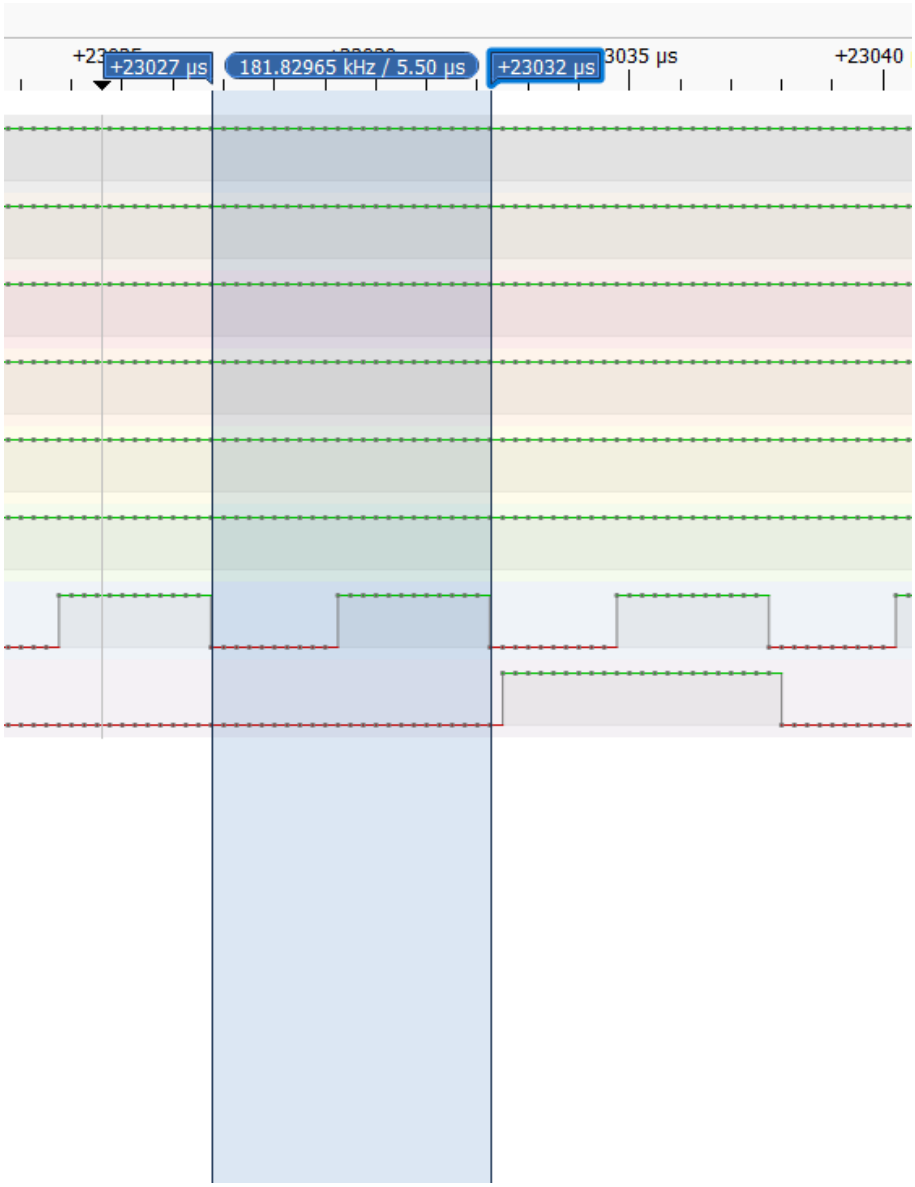

Figure 3: i2c period

After we noticed that the program was measuring the temperature for both sensors 7 times in a row with a sampling rate of 41 ms and then it waited for 874 ms to repeat this process.

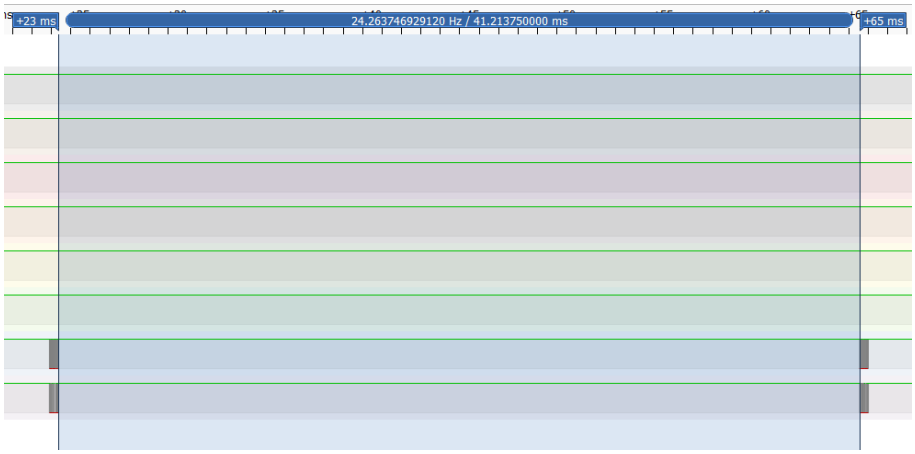

Figure 4: i2c Sampling Rate in groups

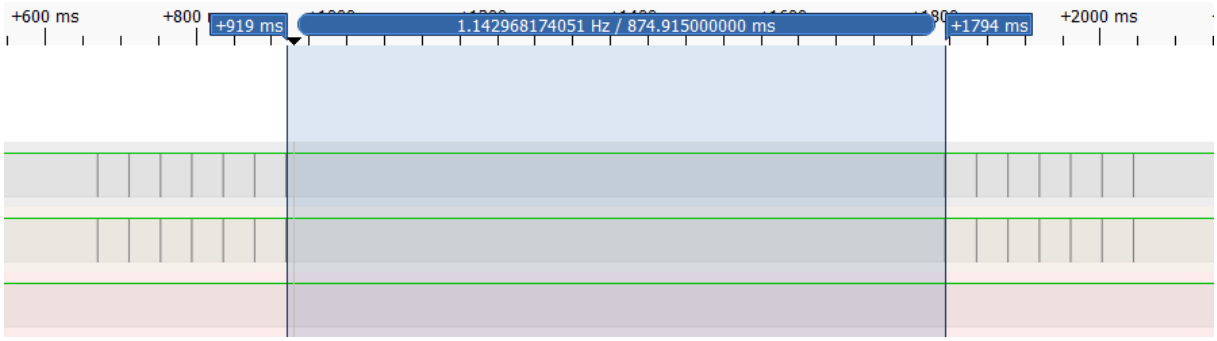

Figure 5: i2c Sampling Rate of groups

3 OCs

These are the outputs to control the energy provided to increase or decrease the temperature. Observing the behavior of each one(increasing and decreasing the temperature of a single one and watching one change behavior) we concluded that the OC1 is responsible for the left sensor and the OC2 for the right one. We measured the change on duty cycle according to the difference between the setpoint and the measured temperature and this is the result.

The figure below shows our results. The column labeled ”DutyCycle” represents the actual duty cycle obtained, while the column ”3.7 * TempDif” shows the duty cycle calculated using our formula. Although this formula approximates the actual duty cycle fairly well, there is a slight error of approximately 2.

Temp	SetPoint	Temp Dif	Period(ms)	UpTime(ms)	DutyCycle	3.7×Temp Dif
40	40	0	8.3	0	0	0
38	40	2	8.3	0.55	6.626506	7.4
35	40	5	8.3	1.6	19.27711	18.5
31	40	9	8.3	2.8	33.73494	33.3
30	40	10	8.3	3.2	38.55422	37
25	40	15	8.3	4.7	56.62651	55.5
20	40	20	8.3	6.3	75.90361	74
15	40	25	8.3	7.9	95.18072	92.5
14	40	26	8.3	8.15	98.19277	96.2
13	40	27	8.3	8.3	100	99.9

Figure 6: Enter Caption

4 SPI

SPI is used for communication between the memory and the controller. It has four pins: the clock is the top right pin, the select is the bottom left pin, the MISO is the top left pin, and the MOSI is the bottom right pin. Then we measured the frequency of the clock, which was 222 MHz.

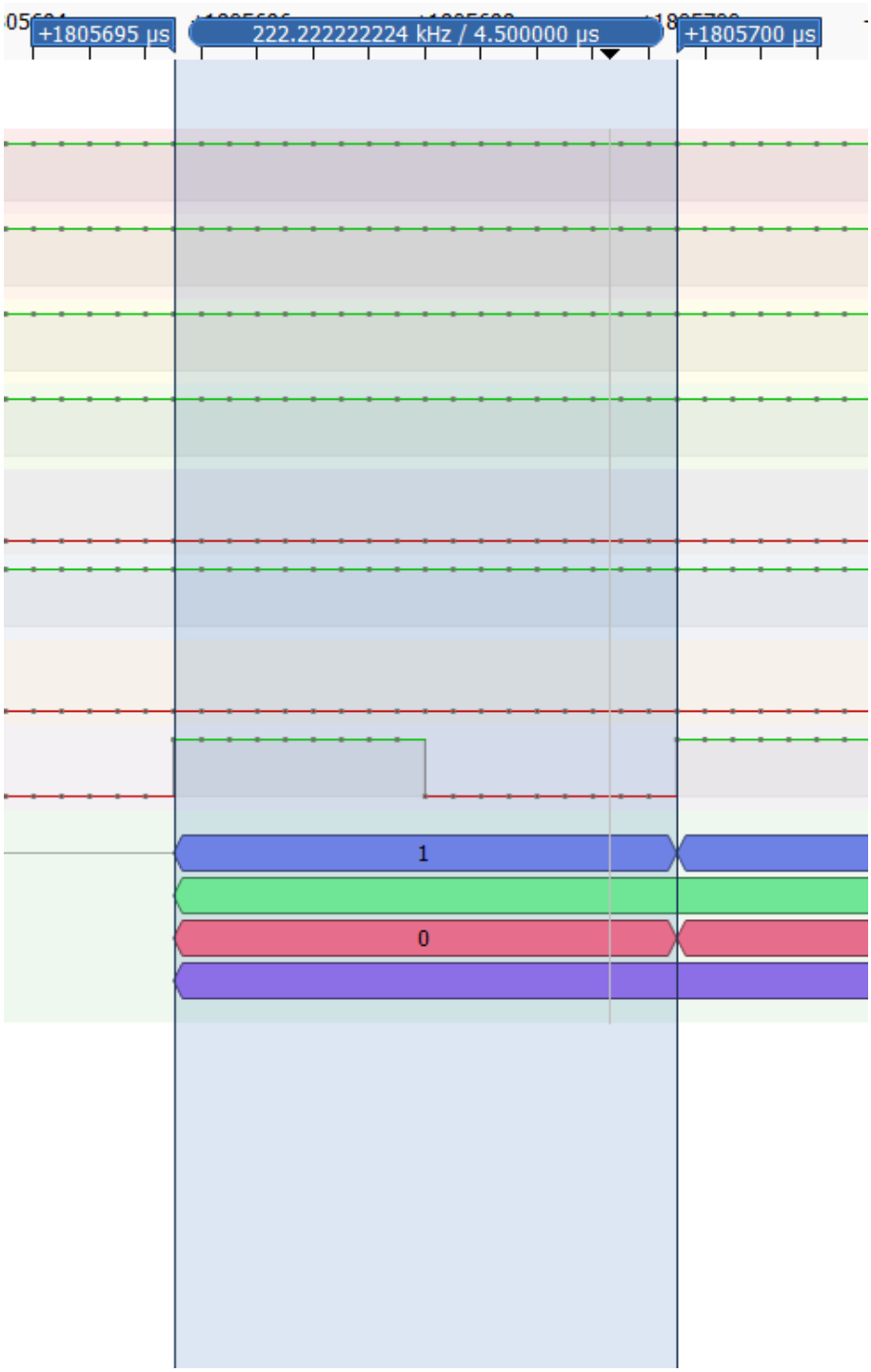

Figure 7: SPI Clock Frequency

4.1 Information Flow

4.1.1 Temperature storage

In the flow of information, the first message sent to the memory is the read status, in this figure it responds with 0, which means that the memory is not busy and can read and write information.

Figure 11: Between the writings

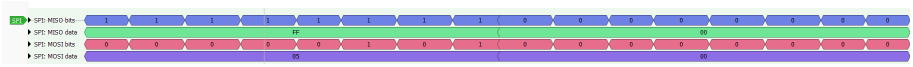

Figure 8: Read Status

Then the master reads the address 3D and 3E, in this case, it shows 37 and 1C respectively. 3D will always show the size of the buffer minus 2 and 3E store the address where the temperatures will be stored minus 4.

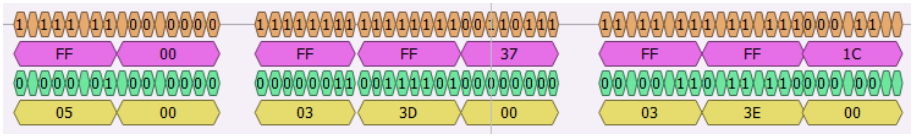

Figure 9: 3D and 3E reading

Next, the write is enabled and the two temperatures are written in the memory, being the even address of the SetPoint1 and the odd SetPoint2, the address for each sensor is equivalent to the 3E plus 4 for the even and 5 for the odd.

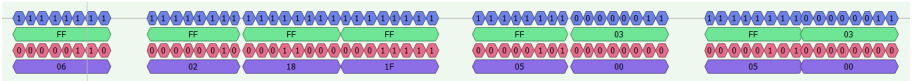

Figure 10: Write sensor values in memory

Between the writings, the status is read in a loop waiting to read the value 00 once again meaning the write is available again.

After the write is once again enabled the temperature is stored on its respective address.

Finally, the last write is to to the address 3E, which contains the previous 3E value plus 2.

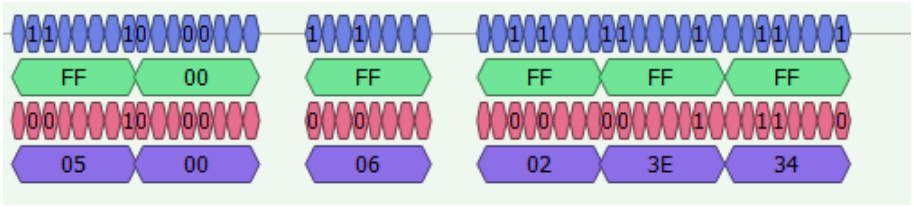

Figure 12: 3E address

We also noticed that once the the 3D value is less than the address used to store the temperature in this case 37 is lower than 38, the 3E address in the end will store 00.

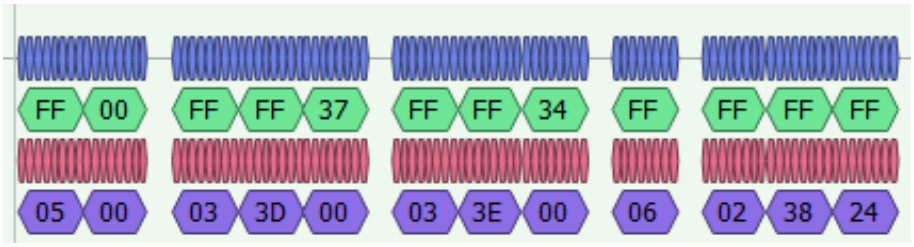

Figure 13: Last Iteraion

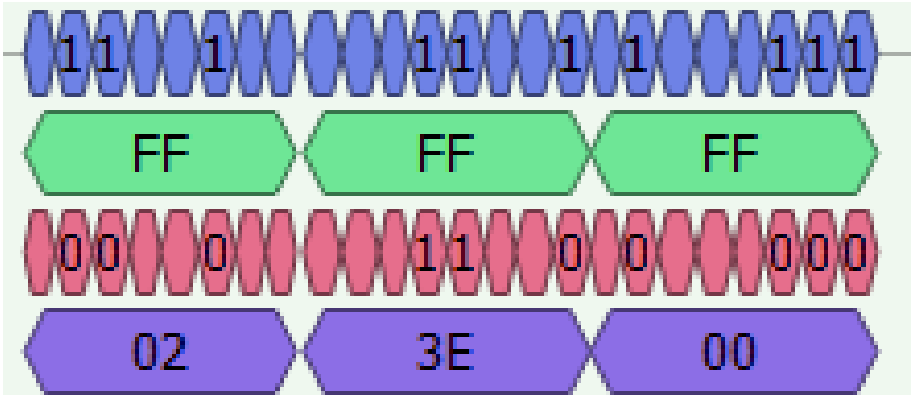

Figure 14: Reset Loop

4.1.2 Setpoint Storage

In this phase, we just observed the flows that happened when we clicked the ”storing” button. With this tactic, we can conclude that SetPoint 1 is stored in ”0x3B” with value ”5E” and SetPoint 2 is stored in ”0x3C” with value ”4A”, and the average is stored in ”3A” with value ”36”.

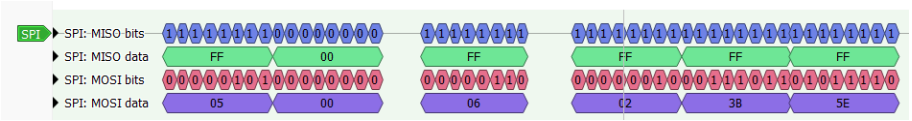

Figure 15: Setpoint1

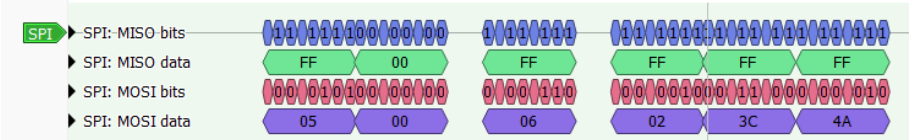

Figure 16: Setpoint2

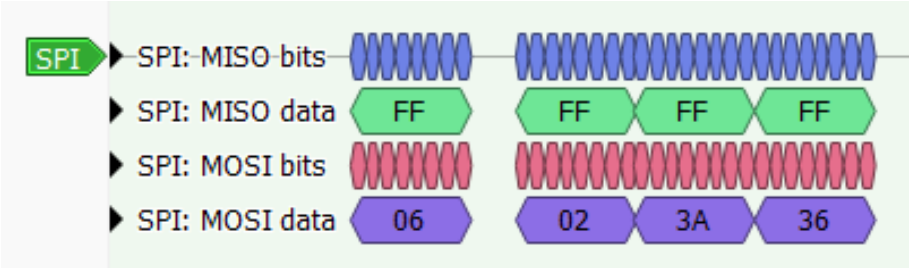

Figure 17: Average

4.1.3 Reset the program

As seen above after resetting the SPI reads from EEPROM the data from the 2 setpoints which are 3B and 3C and from the address 3A which stores the averageSetpoint.

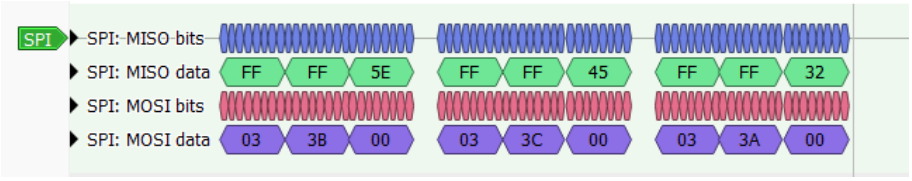

Figure 18: Reset SPI

4.2 Conclusions

After some more observing, we concluded that the addresses containing the temperatures measured are the ones from ”0x04” to ”0x39”. To find out what address to write the next temperature, the register ”0x3E” is used, so that AddrTemp1=0x4+”0x3E” and AddrTemp2=AddrTemp1+1. We couldn’t quite discover the usage of ”0x3D”(During testing it changed once from 36 to 37, and we couldn’t reproduce the change or figure out why it occurred) but we suspect that it could be to indicate the range of valid readings in memory (if the whole memory wasn’t only looped yet and the pointer is to 0x20 the readings in 0x30 arent’ valid but if already looped once the value 0x30 is valid).

5 RS232

Firstly we discovered which pin is the RX and which one is TX (RX-OC3, TX-OC4), we checked that one of them is not transmitting any information and that one is the receiver, and then we discovered the communication format (Baudrate=57600, Parity=Even, Stop Bits=1) by trial and error.

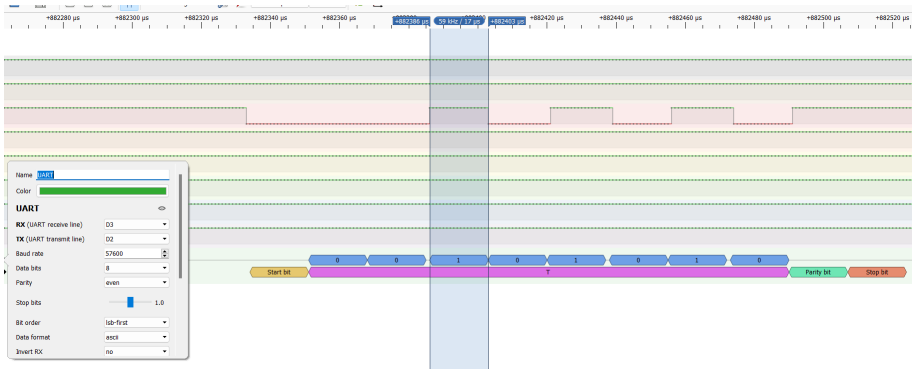

Figure 19: Uart

Firstly on reset the transmitter starts sending a various number of bytes that encode to a starting hello message ”Thermal Camera Control V 4.1”

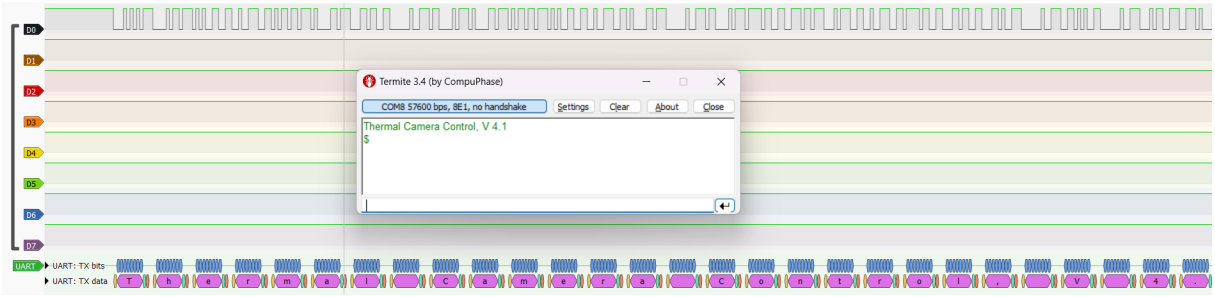

Figure 20: Hello mensage

6 LEDS

The LEDs turn on and off according to the mode the controller is and its values. There are 9 LEDs, 0-7 work together and are either on or off. The last LED is the led 11 which may or may not be blinking according to the mode.

- Normal Mode:
  - Leds 5 and 2 ON.
- Set Point 1:
  - Led 0 ON / 11 Blinking
- Set Point 2:
  - Led 7 ON / 11 Blinking
- Set Average:
  - Leds 7 e 0 ON / 11 Blinking

As for the just functionalities we observed that the Ajust left temperature activated the LEDs 4 to 7 and increasing its temperature depending on the setpoint would increase the number of LEDs shown. The right temperature sensor had the same behavior, however, it activated the LEDs 3 to 0.

The Average just point activated, activated all the LEDs if we increased the temperature of the left sensor the LEDs would light just like the mode ”Adjust left temperature” and the LEDs on the right would work similarly to the ”Adjust right temperature” if we increased the temperature the right side sensor.

Below we can see our findings, with all these results we were still unable to figure out the pattern to light the LEDs.

StartTemp	EndTemp	SetPoint	Leds
0	50	99	1
51	66	99	2
67	82	99	3
83	99	99	4
0	44	75	1
45	53	75	2
54	64	75	3
65	99	75	4
0	38	50	1
39	42	50	2
43	46	50	3
47	99	50	4
0	35	35	1
35	99	35	4
0	35	40	1
36	36	40	2
37	37	40	3
38	99	40	4

Figure 21: LED temperatures

The LED 11 also blinked before showing the current temperature, this can indicate that it blinked every time it would fetch the value of the current temperature.

7 Odities

During the testing, we found out that when going below the temperature 00, it reset to 35. However going above 99 it would stay at 99.

8 Conclusion

We conducted a thorough internal investigation of the Pic32 controller in this study, paying particular attention to its various communication protocols. We were able to learn a lot about the controller's operation thanks to this examination.