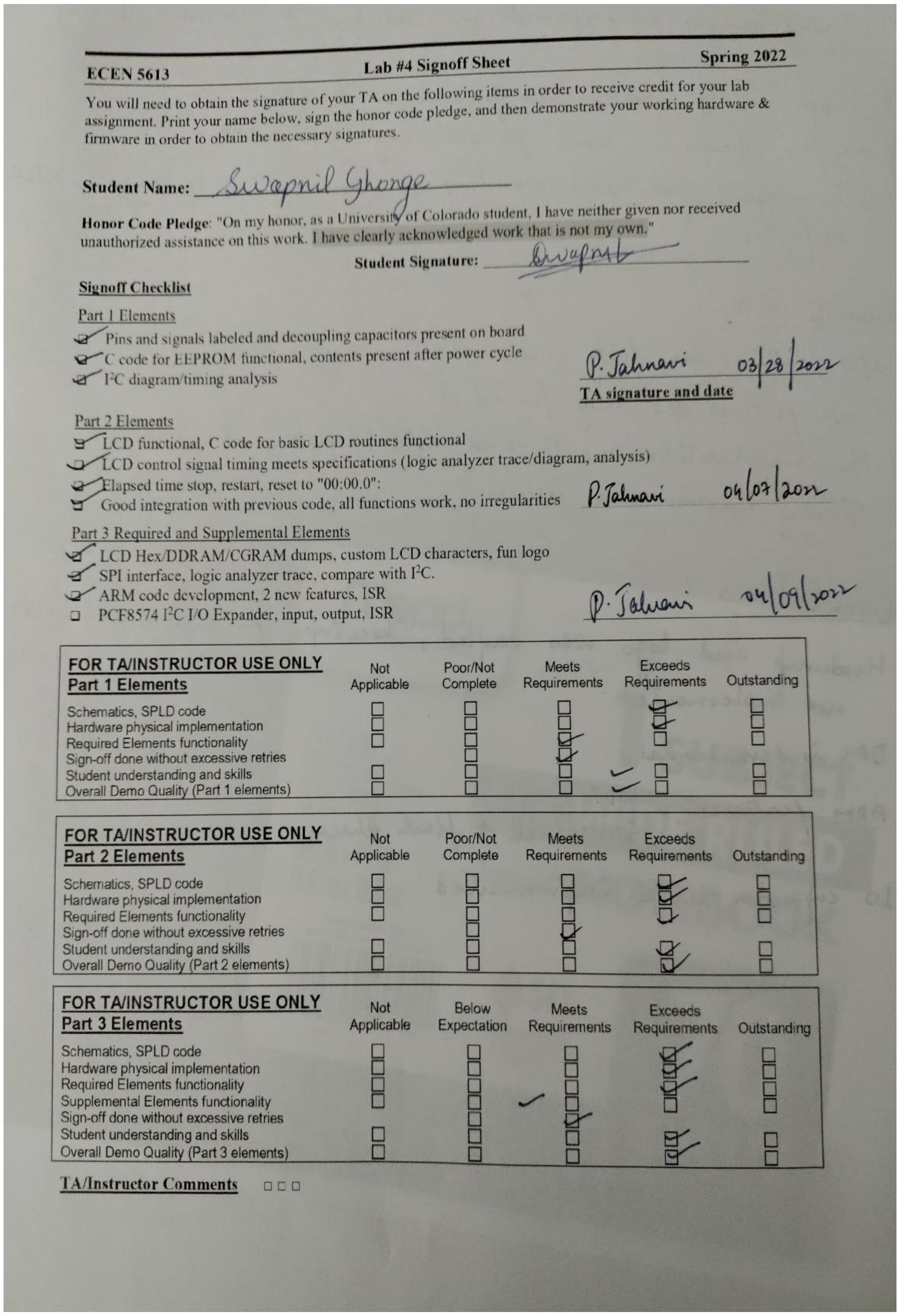
ESD LAB 4 REPORT

Sign-off Sheet:



ECEN 5613 — Lab #4 Signoff Sheet — Spring 2022

You will need to obtain the signature of your TA on the following items in order to receive credit for your lab assignment. Print your name below, sign the honor code pledge, and then demonstrate your working hardware & firmware in order to obtain the necessary signatures.

Student Name: Swapnil Ghonge

Honor Code Pledge: "On my honor, as a University of Colorado student, I have neither given nor received unauthorized assistance on this work. I have clearly acknowledged work that is not my own."

Student Signature: [illegible]

Signoff Checklist

Part 1 Elements

- ✓ Pins and signals labeled and decoupling capacitors present on board
- ✓ C code for EEPROM functional, contents present after power cycle
- ✓ I²C diagram/timing analysis

[illegible] 03/28/2022
TA signature and date

Part 2 Elements

- ✓ LCD functional, C code for basic LCD routines functional
- ✓ LCD control signal timing meets specifications (logic analyzer trace/diagram, analysis)
- ✓ Elapsed time stop, restart, reset to "00:00.0":
- ✓ Good integration with previous code, all functions work, no irregularities

[illegible] 04/07/2022

Part 3 Required and Supplemental Elements

- ✓ LCD Hex/DDRAM/CGRAM dumps, custom LCD characters, fun logo
- ✓ SPI interface, logic analyzer trace, compare with I²C.
- ✓ ARM code development, 2 new features, ISR
- ☐ PCF8574 I²C I/O Expander, input, output, ISR

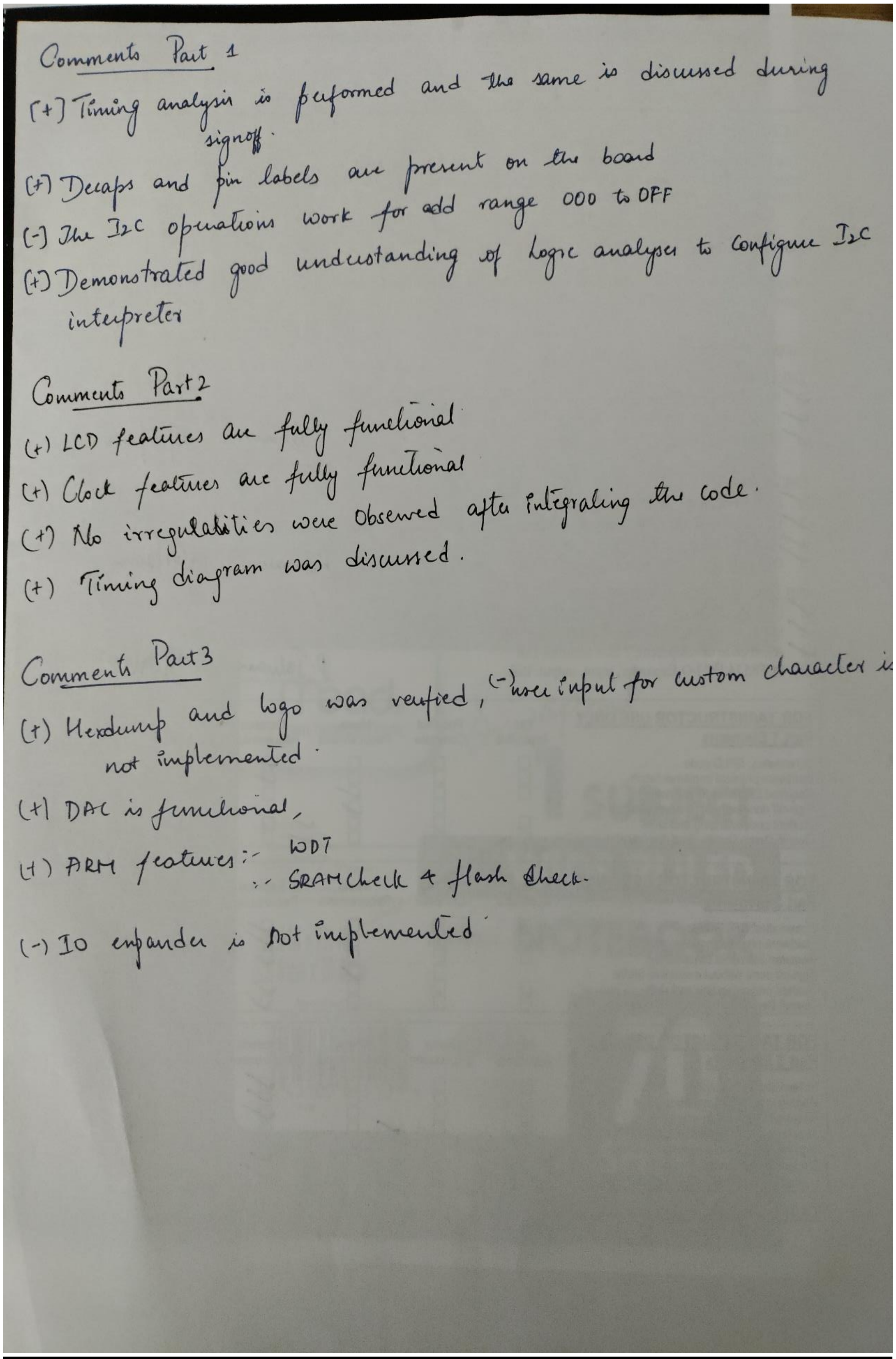
[illegible] 04/09/2022

FOR TA/INSTRUCTOR USE ONLY **Part 1 Elements**	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	✓	☐
Hardware physical implementation	☐	☐	☐	✓	☐
Required Elements functionality	☐	☐	✓	☐	☐
Sign-off done without excessive retries		☐	✓		
Student understanding and skills	☐	☐	☐	✓ ☐	☐
Overall Demo Quality (Part 1 elements)	☐	☐	☐	✓ ☐	☐

FOR TA/INSTRUCTOR USE ONLY **Part 2 Elements**	Not Applicable	Poor/Not Complete	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	✓	☐
Hardware physical implementation	☐	☐	☐	✓	☐
Required Elements functionality	☐	☐	☐	✓	☐
Sign-off done without excessive retries		☐	✓		
Student understanding and skills	☐	☐	☐	✓	☐
Overall Demo Quality (Part 2 elements)	☐	☐	☐	✓	☐

FOR TA/INSTRUCTOR USE ONLY **Part 3 Elements**	Not Applicable	Below Expectation	Meets Requirements	Exceeds Requirements	Outstanding
Schematics, SPLD code	☐	☐	☐	✓	☐
Hardware physical implementation	☐	☐	☐	✓	☐
Required Elements functionality	☐	☐	☐	✓	☐
Supplemental Elements functionality	☐	☐	✓ ☐	☐	☐
Sign-off done without excessive retries		☐	✓		
Student understanding and skills	☐	☐	☐	✓	☐
Overall Demo Quality (Part 3 elements)	☐	☐	☐	✓	☐

TA/Instructor Comments ☐ ☐ ☐



Comments Part 1

(+) Timing analysis is performed and the same is discussed during signoff.

(+) Decaps and pin labels are present on the board

(-) The I2C operations work for add range 000 to 0FF

(+) Demonstrated good understanding of logic analyser to configure I2C interpreter

Comments Part 2

(+) LCD features are fully functional.

(+) Clock features are fully functional

(+) No irregularities were observed after integrating the code.

(+) Timing diagram was discussed.

Comments Part 3

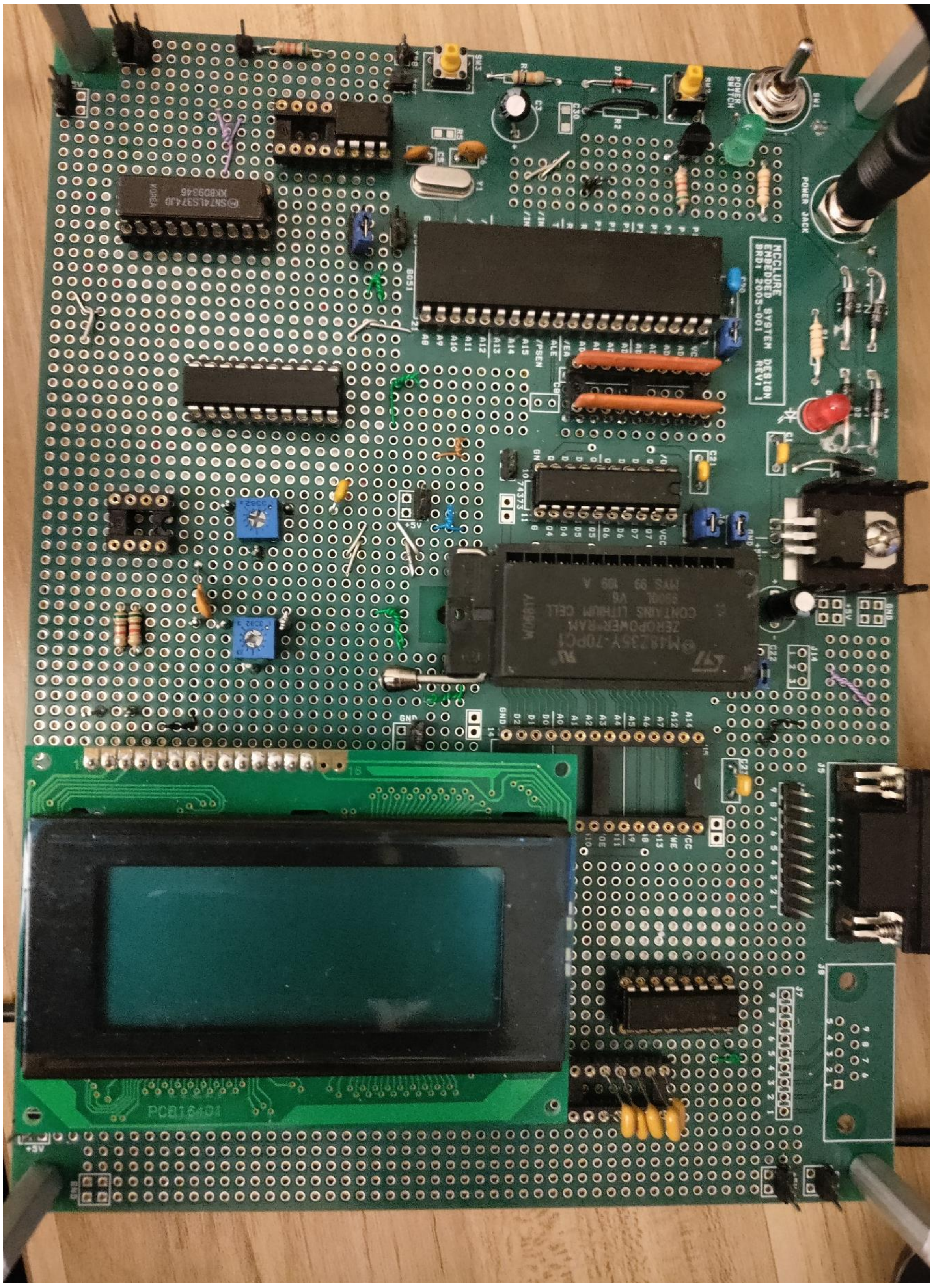
(+) Hexdump and logo was verfied, (-) user input for custom character i not implemented.

(+) DAC is functional,

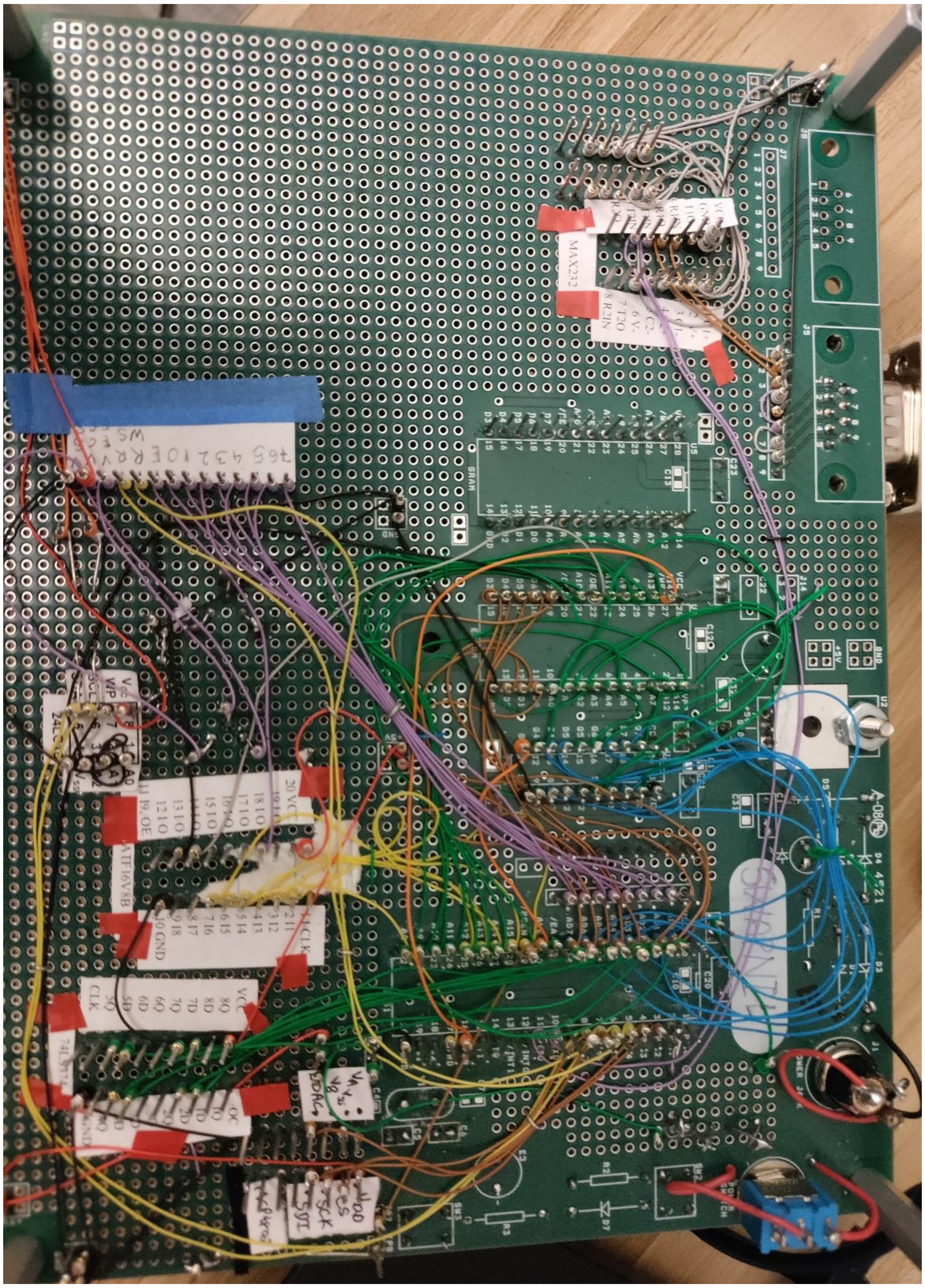
(+) ARM features :- WDT
:- SRAM check + flash check.

(-) IO expander is not implemented.

Board Top



Board Bottom



Part-1

Brief: The of Lab 4 Part 1 was to write I2C driver with the ability to bit-bang write and read a byte at any EEPROM I2C address using function calls from C.

In this part of lab I implemented an I2C device driver EEPROM IC NM24C04. Where I connected the SCL and SDA to the pins of AT89C51RC2 microcontroller IC.

To implement bit bang technique I deviced 2 function, so that I could read, write the data void weeprom (char page, char address, char datum); char reeprom (char page, char address);

Then I programmed about the acknowledge bit after we send the address and the data.

Key Learning:

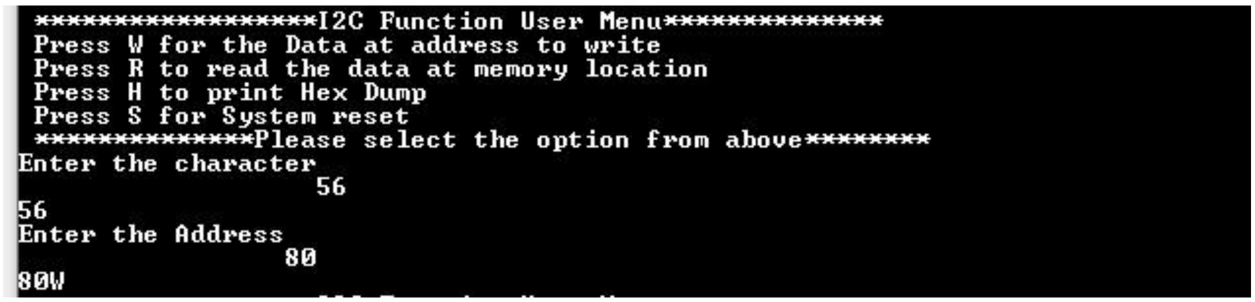
Was able to Write I2C driver and perform the bit bang technique.

At every clock cycle which is connected to GPIO, 1 bit of data was sent.

Challenges faced:

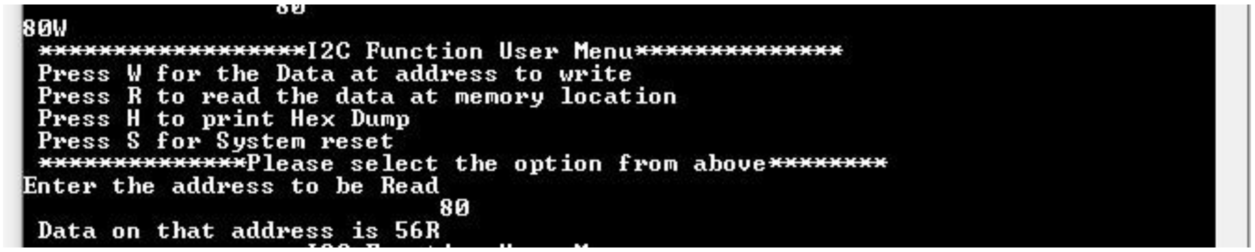
To debug the master and slave was most difficult task, for which I used a logic analyzer to debug my code. I also faced a lot of difficulty in receiving the acknowledge bit which was hard to debug, for which I made some changes in my code which led to accurate value of acknowledge from the slave.

Below are the screenshots of the implementation of I2C using bit banging technique.



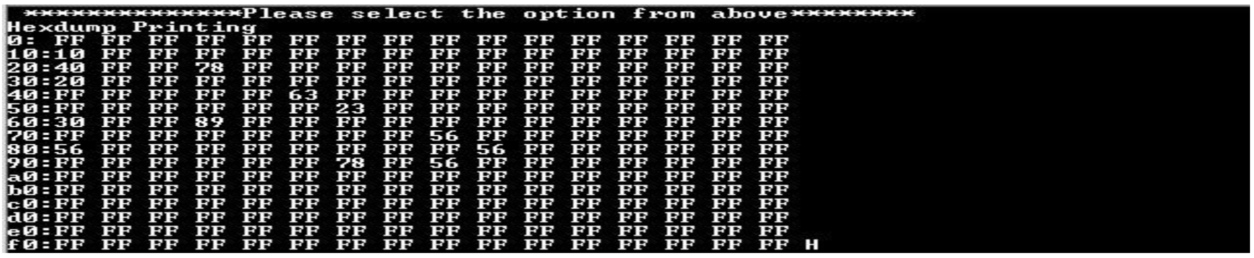
```
 *******************I2C Function User Menu***************
 Press W for the Data at address to write
 Press R to read the data at memory location
 Press H to print Hex Dump
 Press S for System reset
 **************Please select the option from above********
Enter the character
                    56
56
Enter the Address
                  80
80W
```

User menu on Serial Communication to write a character at specific location.



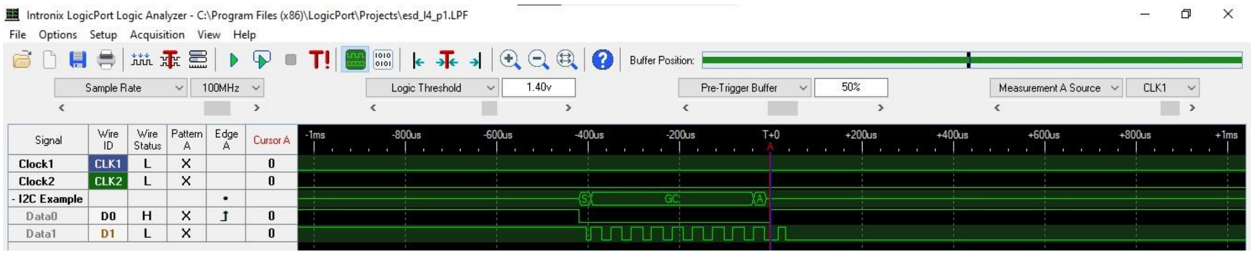
```
80W
 *******************I2C Function User Menu***************
 Press W for the Data at address to write
 Press R to read the data at memory location
 Press H to print Hex Dump
 Press S for System reset
 **************Please select the option from above********
Enter the address to be Read
                            80
 Data on that address is 56R
```

Reading the same written value on address 0x80h

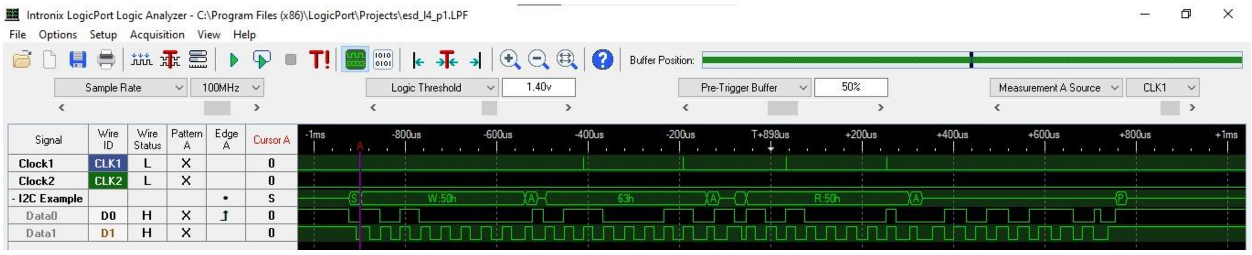


```
 ***************Please select the option from above*********
Hexdump Printing
0:  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
10:10 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
20:40 FF FF 78 FF FF FF FF FF FF FF FF FF FF FF FF
30:20 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
40:FF FF FF FF FF 63 FF FF FF FF FF FF FF FF FF FF
50:FF FF FF FF FF FF 23 FF FF FF FF FF FF FF FF FF
60:30 FF FF 89 FF FF FF FF FF FF FF FF FF FF FF FF
70:FF FF FF FF FF FF FF FF 56 FF FF FF FF FF FF FF
80:56 FF FF FF FF FF FF FF FF 56 FF FF FF FF FF FF
90:FF FF FF FF FF FF 78 FF 56 FF FF FF FF FF FF FF
a0:FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
b0:FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
c0:FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
d0:FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
e0:FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
f0:FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF H
```

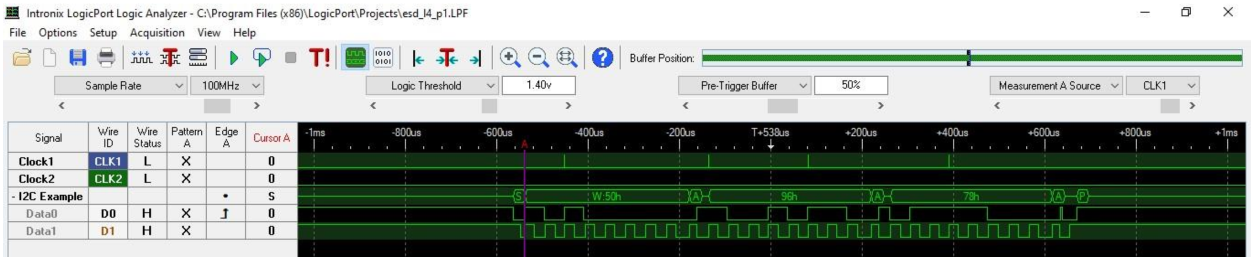
Printing the dump of values of memory location using Hex dump function.


Intronix LogicPort Logic Analyzer - C:\Program Files (x86)\LogicPort\Projects\esd_l4_p1.LPF
File Options Setup Acquisition View Help
Buffer Position:
Sample Rate 100MHz
Logic Threshold 1.40v
Pre-Trigger Buffer 50%
Measurement A Source CLK1
Signal
Clock1 CLK1 L X 0
Clock2 CLK2 L X 0
- I2C Example
Data0 D0 H X 0
Data1 D1 L X 0
-1ms -800us -600us -400us -200us T+0 +200us +400us +600us +800us +1ms
S GC A

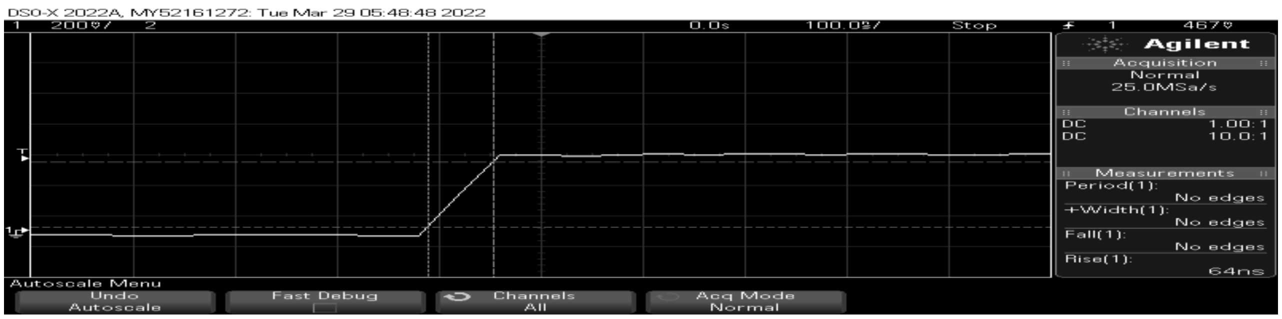
Resetting the EEPROM and data seen on Logic Analyzer


Intronix LogicPort Logic Analyzer - C:\Program Files (x86)\LogicPort\Projects\esd_l4_p1.LPF
Sample Rate 100MHz
Logic Threshold 1.40v
Pre-Trigger Buffer 50%
Measurement A Source CLK1
Clock1 CLK1 L X 0
Clock2 CLK2 L X 0
- I2C Example S
Data0 D0 H X 0
Data1 D1 H X 0
-1ms -800us -600us -400us -200us T+898us +200us +400us +600us +800us +1ms
S W:50h A 63h A R:50h A P

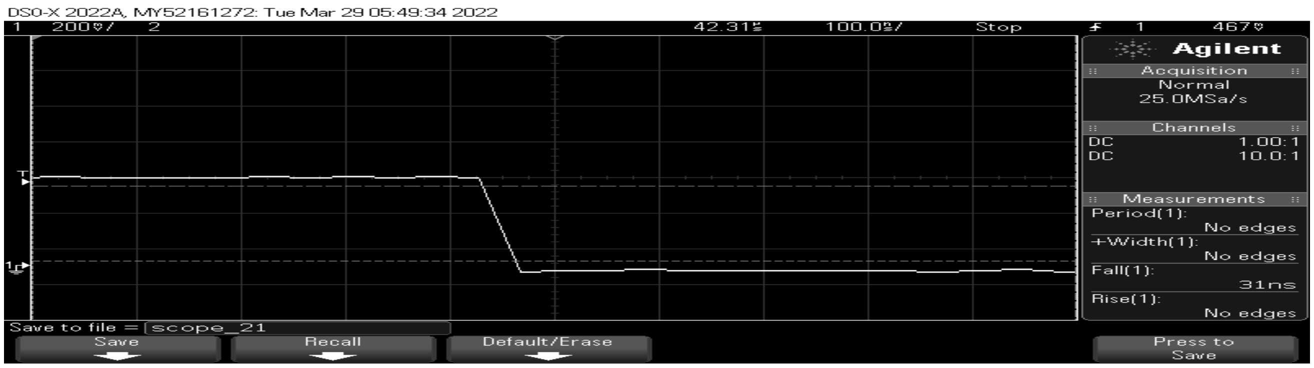
Writing value 60h at 50h memory location.


Intronix LogicPort Logic Analyzer - C:\Program Files (x86)\LogicPort\Projects\esd_l4_p1.LPF
Sample Rate 100MHz
Logic Threshold 1.40v
Pre-Trigger Buffer 50%
Measurement A Source CLK1
Clock1 CLK1 L X 0
Clock2 CLK2 L X 0
- I2C Example S
Data0 D0 H X 0
Data1 D1 H X 0
-1ms -800us -600us -400us -200us T+538us +200us +400us +600us +800us +1ms
S W:50h A 96h A 78h A P

Reading the same 60h value at 50h memory location.



Rising time of the EEPROM IC using Oscillloscope: 64ns



Falling time of the EEPROM IC using Oscillloscope: 31ns

PART-2:

Brief: The objective of this part was to Interface LCD HD44780U, to integrate the LCD with AT89C51RC2 microcontroller and display string, character, time elapsed, restart time, stop time.

To integrate the LCD with AT89C51RC2 I wire wrapped the DB0- DB7 pins of LCD to AD0-AD7 pin of the microcontroller. The Register select pin was connected to P1.4 and Register Write P1.5 GPIO pins. To initialize the LCD the address pointer at 8000h. The lcd init function uses 0x30h value with a delay of 150ms while continuously checking the LCD is busy or not. Now here the RS and RW pins are set as low. To turn the cursor-on the LCD is checked of Buffer flag is zero.

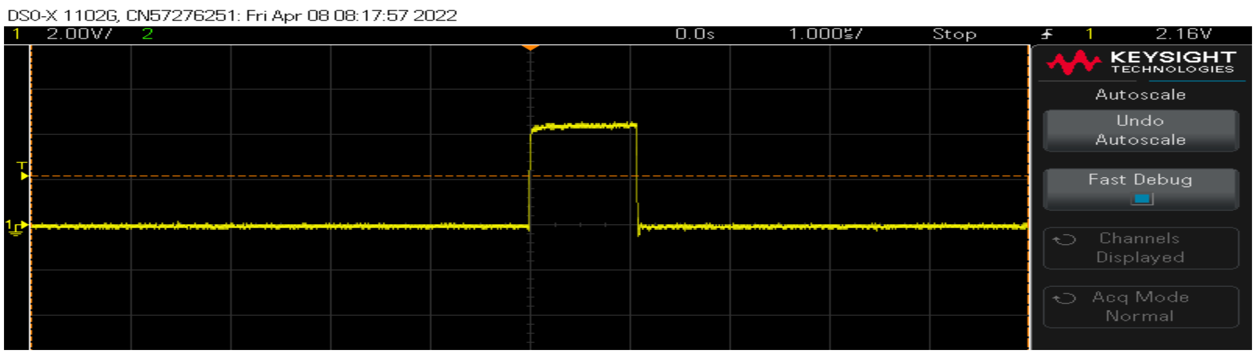
Key Learning:

I was able to successfully to initialize the LCD with writing characters on LCD, writing strings on LCD at any coordinates. I was able to initialize the timer in LCD and by the help of putchar() function I was able to print string on LCD. I was able to print to timer with millisecond and second and minute.

Challenges:

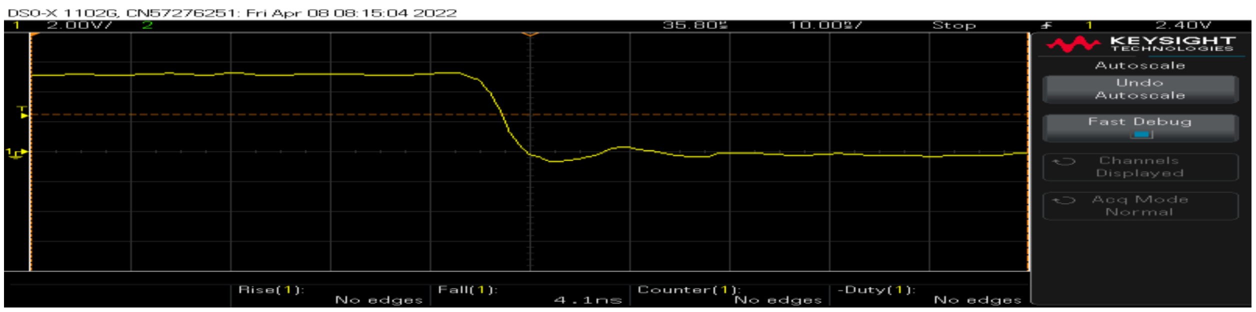
The biggest challenges I faced was contrast of the LCD was not optimum. I started debugging the code, but I found out that the program was correct and according to datasheet. I checked connection according to datasheet which were proper, Then I shifted my attention to working of potentiometer. The potentiometer when powered on the resistance was not changing, to I took help for TA Maanas and I changed the potentiometer. After tweaking the potentiometer a bit I was able to some brightness on LCD.

Below are the Screenshots of the reading I took


DSO-X 1102G, CN57276251: Fri Apr 08 08:17:57 2022

1 2.00V/ 2 0.0s 1.000µs/ Stop 1 2.16V

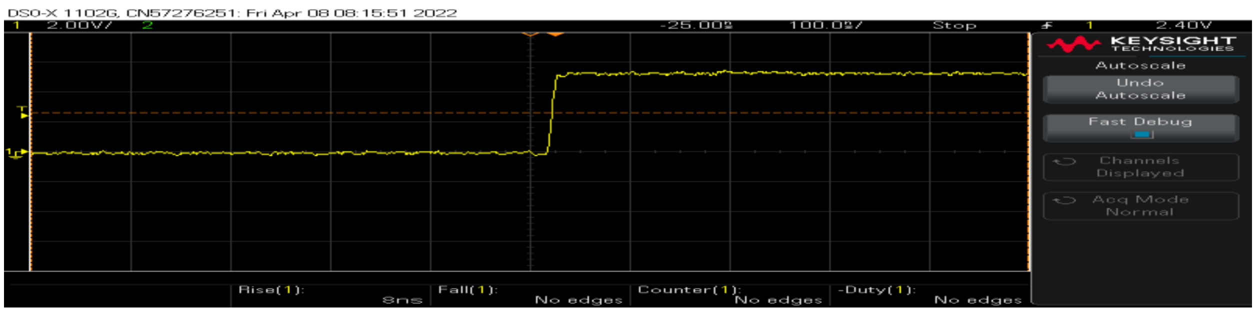
Rise time and Fall time of the Enable.


DSO-X 1102G, CN57276251: Fri Apr 08 08:15:04 2022

1 2.00V/ 2 35.80ns 10.00ns/ Stop 1 2.40V

Rise(1): No edges Fall(1): 4.1ns Counter(1): No edges -Duty(1): No edges

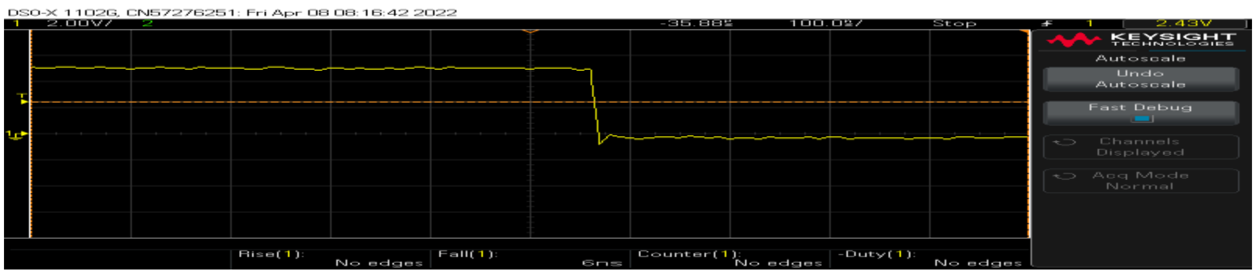
Register Select RS fall time: 4.1ns


DSO-X 1102G, CN57276251: Fri Apr 08 08:15:51 2022

1 2.00V/ 2 -25.00ns 100.0ns/ Stop 1 2.40V

Rise(1): 8ns Fall(1): No edges Counter(1): No edges -Duty(1): No edges

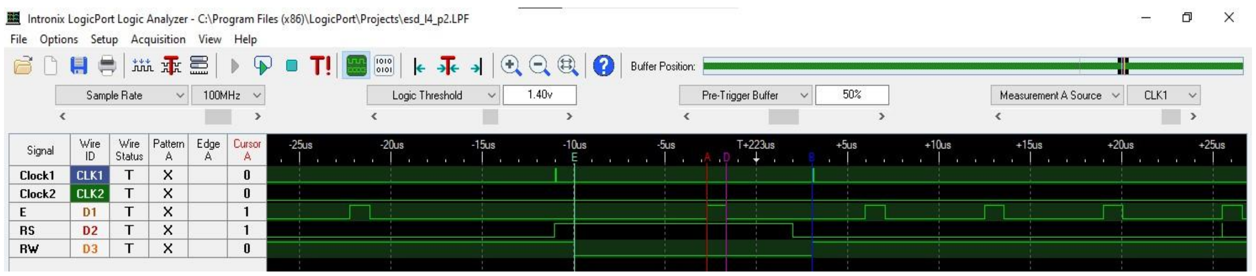
Register Select RS rise time: 4.1ns


DSO-X 1102G, CN57276251: Fri Apr 08 08:16:42 2022

1 2.00V/ 2 -35.88ns 100.0ns/ Stop 1 2.43V

Rise(1): No edges Fall(1): 6ns Counter(1): No edges -Duty(1): No edges

Register Write RW fall time: 6ns


Intronix LogicPort Logic Analyzer - C:\Program Files (x86)\LogicPort\Projects\esd_l4_p2.LPF
File Options Setup Acquisition View Help
Buffer Position:
-25us -20us -15us -10us -5us T+223us +5us +10us +15us +20us +25us

Sample Rate: 100MHz
Logic Threshold: 1.40v
Pre-Trigger Buffer: 50%
Measurement A Source: CLK1

Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
Clock1	CLK1	T	X		0
Clock2	CLK2	T	X		0
E	D1	T	X		1
RS	D2	T	X		1
RW	D3	T	X		0

Logic Analyzer Enable/RS/RW following the timing analysis according to datasheet

PART-3

Brief: LCD DDRAM and displays it in hex on the PC screen in a clean and logical format. Also, LCD characters and print fun logo. LCD HexDump the values of DDRAM, CGRAM. Interface SPI with MCP4802(DAC) and compare with I2C. Develop ARM code of two features, the new I implemented was watch-dog timer, SRAM check and Flash Check and blink the LED.

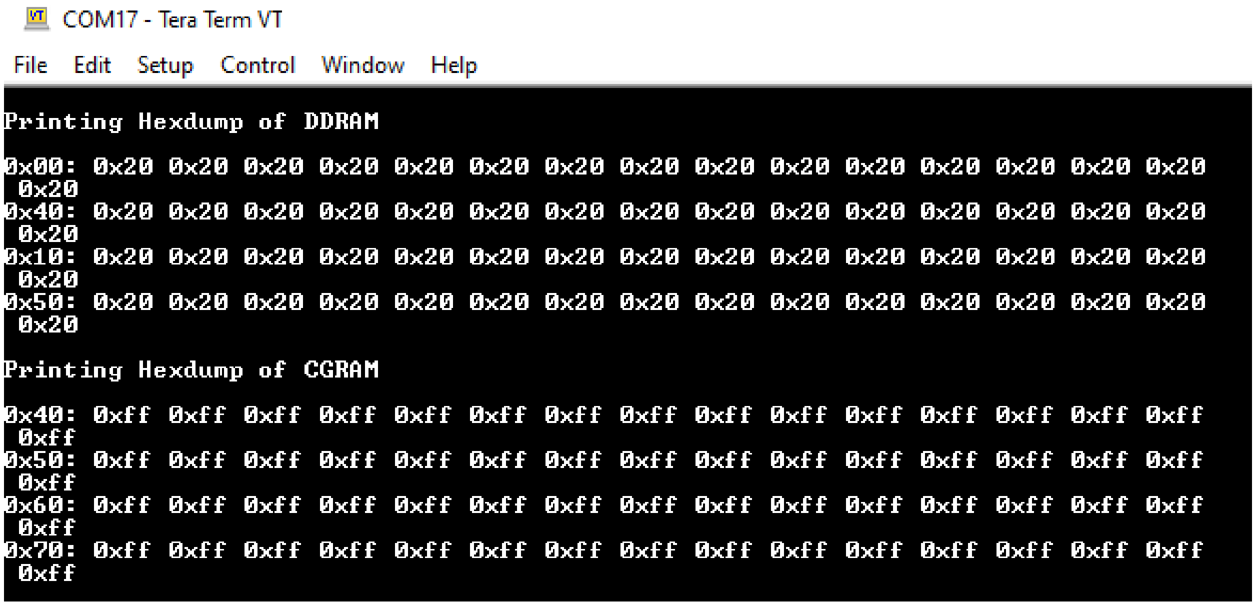
Key Learning:

I successfully created the Fun logo of a character of 'CU' by writing the values on CGRAM at address starting from 0x40h. I learnt how to write the values in CG RAM and DDRAM and integrating with putchar function. I also learn to integrate timer with LCD and successfully integrated the timer interrupt with LCD. I also learnt about how to configure the DAC through SPCON register and sending message and converting to voltage, I sent the valye 0x7450h and converted voltage I got was 0.81 volts according to the calculation Vo = Vin/2^N. For the ARM code development I chose to learn more about watchdog timer and its operation. Secondly in ARM code, I also learnt how to check the code spaces in SRAM and Flash drive and blink an LED when one of the memory is accessed while checking.

Challenges:

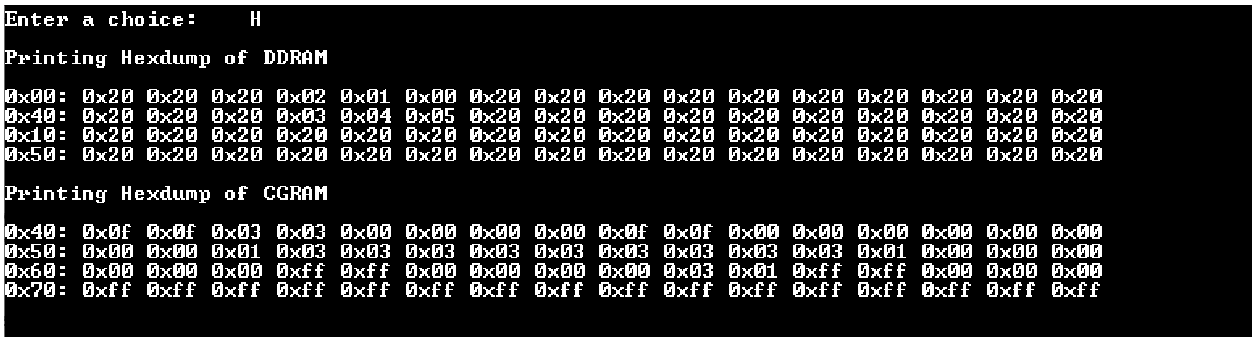
The challenges I faced configuring the correct value to initialize the DAC by SPCON register. I took help from classmates and learnt how to configure the DAC correctly. The next challenge I faced was I was not receiving data on Logic Analyzer, to which I checked my code how the value progressed. Another challenge I faced was printing the value of Hexdump of CGRAM, I was not able to configure it correctly, I took help from my PES lecture and tried to follow the steps to print Hexdump and I successfully implemented the hexdump.

Below are screenshots of the part 3 of Lab 4



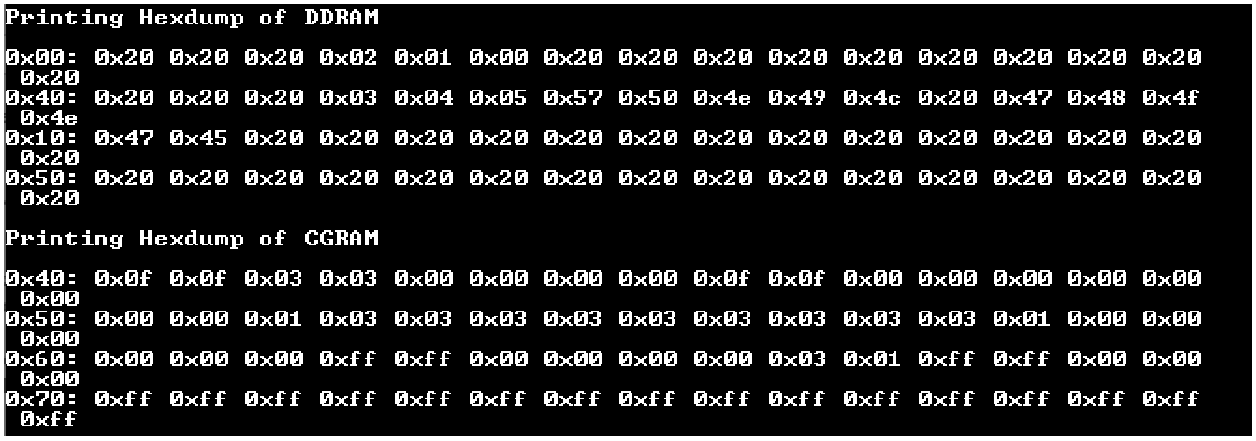
```
Printing Hexdump of DDRAM

0x00: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20
0x40: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20
0x10: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20
0x50: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20

Printing Hexdump of CGRAM

0x40: 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
 0xff
0x50: 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
 0xff
0x60: 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
 0xff
0x70: 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
 0xff
```

Printing of HexDump when NO value is written.



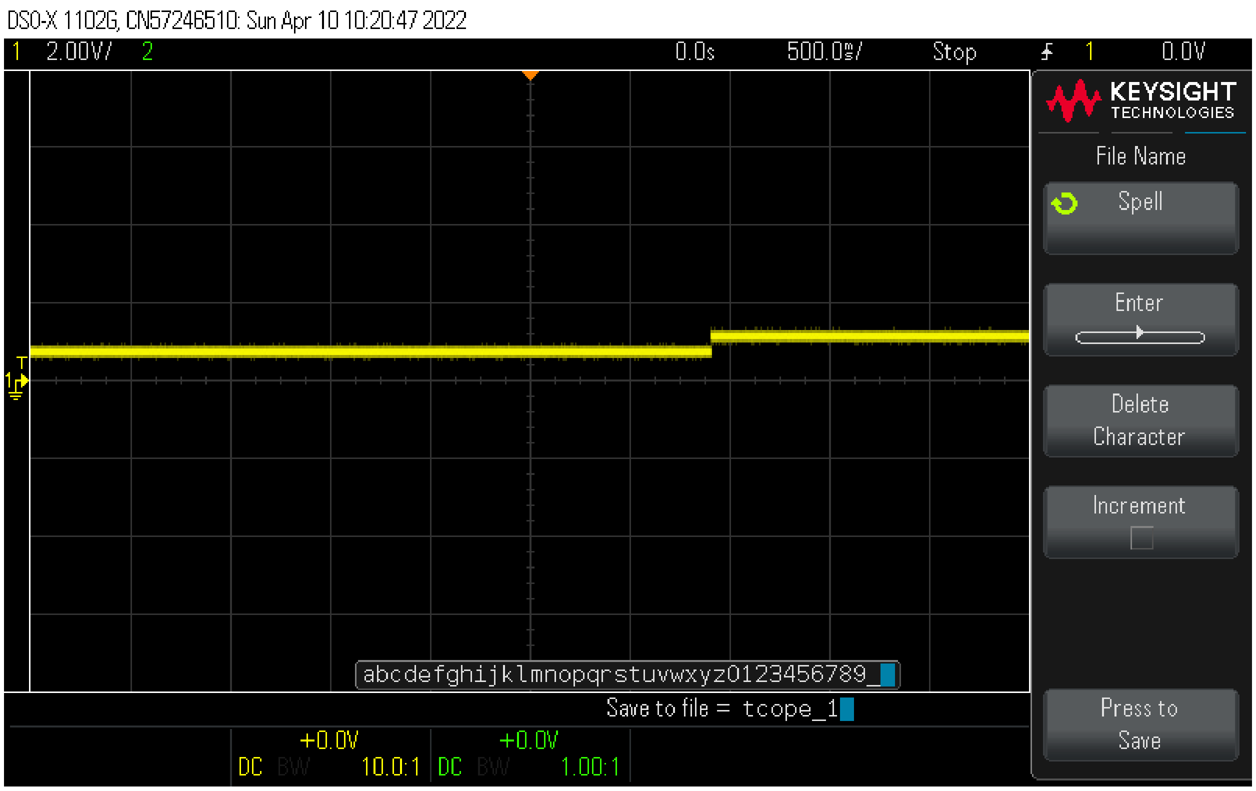
```
Enter a choice:    H

Printing Hexdump of DDRAM

0x00: 0x20 0x20 0x20 0x02 0x01 0x00 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
0x40: 0x20 0x20 0x20 0x03 0x04 0x05 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
0x10: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
0x50: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20

Printing Hexdump of CGRAM

0x40: 0x0f 0x0f 0x03 0x03 0x00 0x00 0x00 0x00 0x0f 0x0f 0x00 0x00 0x00 0x00 0x00 0x00
0x50: 0x00 0x00 0x01 0x03 0x03 0x03 0x03 0x03 0x03 0x03 0x03 0x03 0x01 0x00 0x00 0x00
0x60: 0x00 0x00 0x00 0xff 0xff 0x00 0x00 0x00 0x00 0x03 0x01 0xff 0xff 0x00 0x00 0x00
0x70: 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
```

Printing the HexDump when after printing the CU Logo.



```
Printing Hexdump of DDRAM

0x00: 0x20 0x20 0x20 0x02 0x01 0x00 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20
0x40: 0x20 0x20 0x20 0x03 0x04 0x05 0x57 0x50 0x4e 0x49 0x4c 0x20 0x47 0x48 0x4f
 0x4e
0x10: 0x47 0x45 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20
0x50: 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
 0x20

Printing Hexdump of CGRAM

0x40: 0x0f 0x0f 0x03 0x03 0x00 0x00 0x00 0x00 0x0f 0x0f 0x00 0x00 0x00 0x00 0x00
 0x00
0x50: 0x00 0x00 0x01 0x03 0x03 0x03 0x03 0x03 0x03 0x03 0x03 0x03 0x01 0x00 0x00
 0x00
0x60: 0x00 0x00 0x00 0xff 0xff 0x00 0x00 0x00 0x00 0x03 0x01 0xff 0xff 0x00 0x00
 0x00
0x70: 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
 0xff
```

Printing HexDump values when I pass String in the DDRAM



Output of DAC as 0.8 volts as discussed above