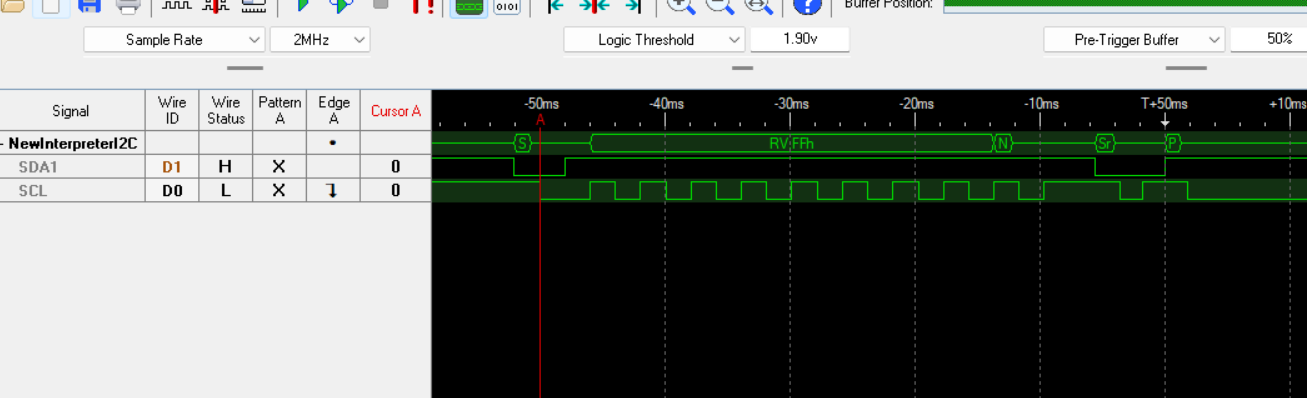
# LAB REPORT

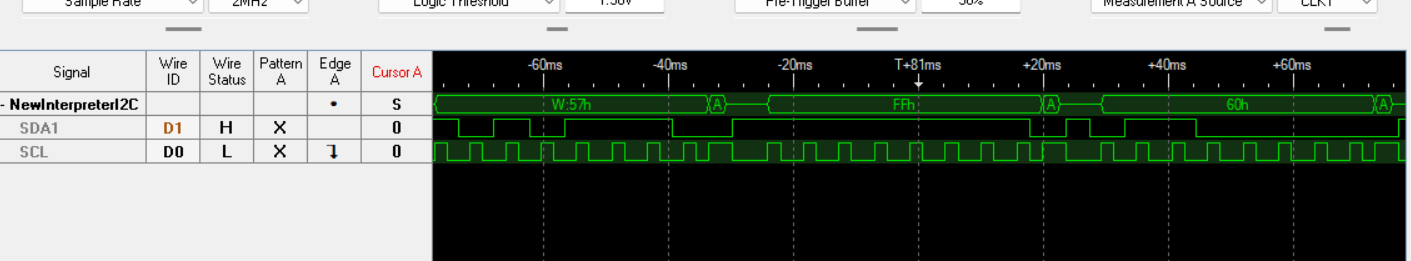
**Lab Objectives:**

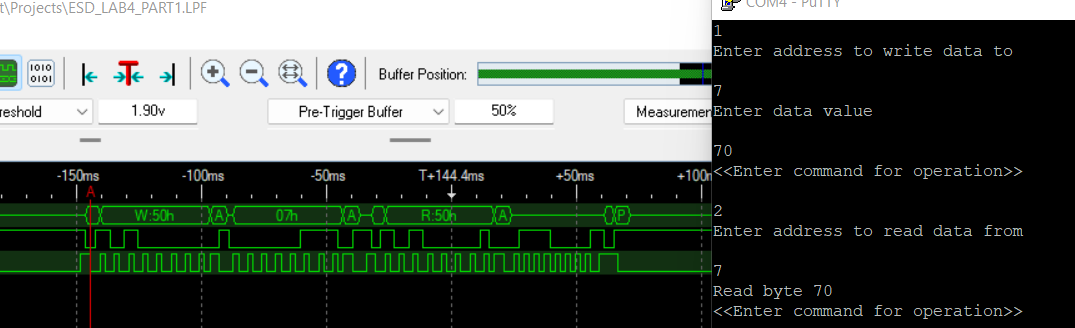
* Implement a bit-banged interface to the EEPROM.
* Add an LCD to your hardware. Implement a memory mapped I/O interface to the LCD and use C pointers to access the LCD as a memory-mapped peripheral.
* Write device drivers for the EEPROM and LCD.
* Write assembly and C programs to implement a user interface and perform user tasks.
* Gain experience in code integration and how to use embedded C including interrupts.
* Continue learning about the ARM architecture.

## **PART 1**

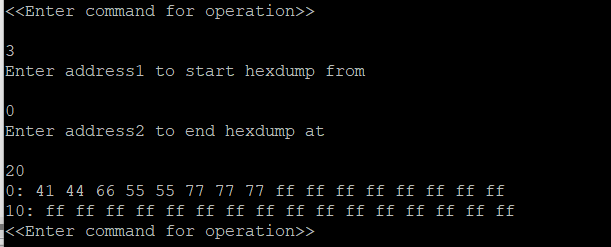
I have implemented bit-banged EEPROM circuit by connecting two port pins of the 8051 to the SDA and SCL of the EEPROM so it does not consume any processor space. Using the LogicPort analyzer’s I2C interpreter the following waveforms have been obtained.

**EEPROM reset sequence:** The reset sequence is important to prevent events such as microcontroller being set during I2C communication, brown-out conditions, excessive noise on the clock or data lines, and improper bus input levels during power up. With bi-directional nature of the bus in EEPROM, it is important to ensure the sender/receiver synchronization is maintained.

**EEPROM write sequence**: After receiving the start condition, the device code, block address and R/W bit is sent. On the 9th cycle, an acknowledge bit is received. The next byte transmitted is the word address (7FFh). After another ack bit, the 8051 will transfer the data into the addressed memory location (60h).

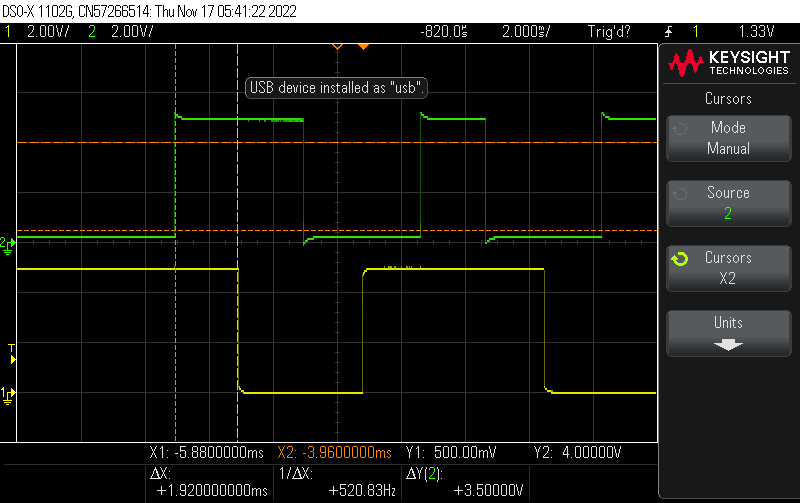
**EEPROM Read sequence:** In random read, the control byte is set with R/W=1 so read operation can be performed and the data is received.

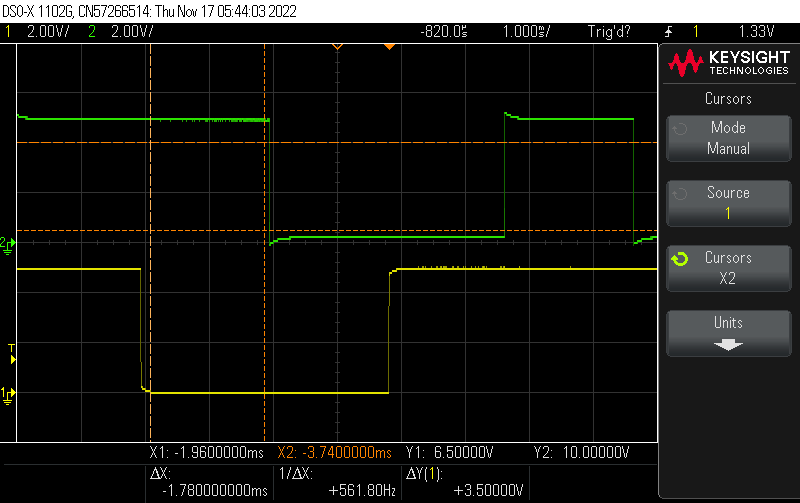
The hexdump of EEPROM values with starting address of 0 and ending address of 20h:



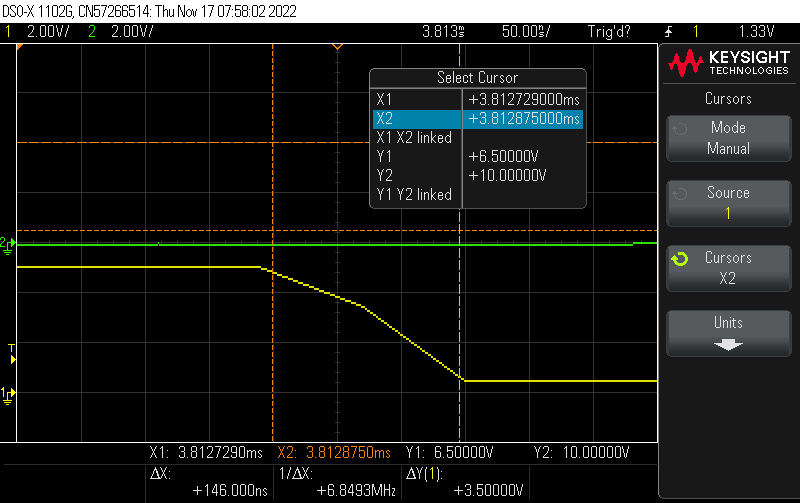
**TIMING ANALYSIS OF EEPROM AC CHARACTERISTICS:**

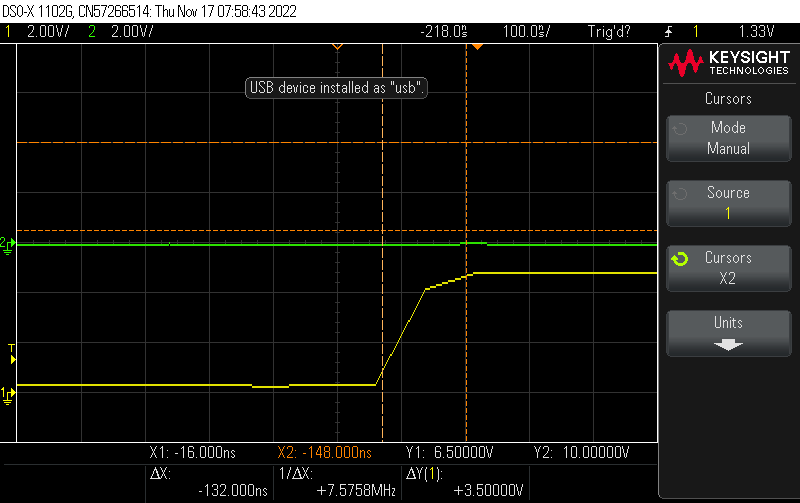
**Start condition hold time** **(1.92 ms):**



**Start condition set-up time (1.78ms):**

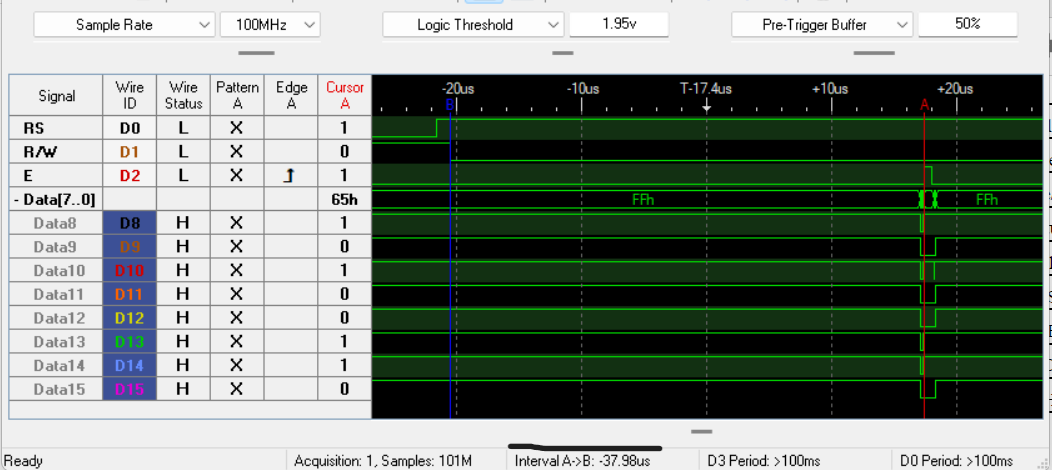
**SCL fall time (146ns):**

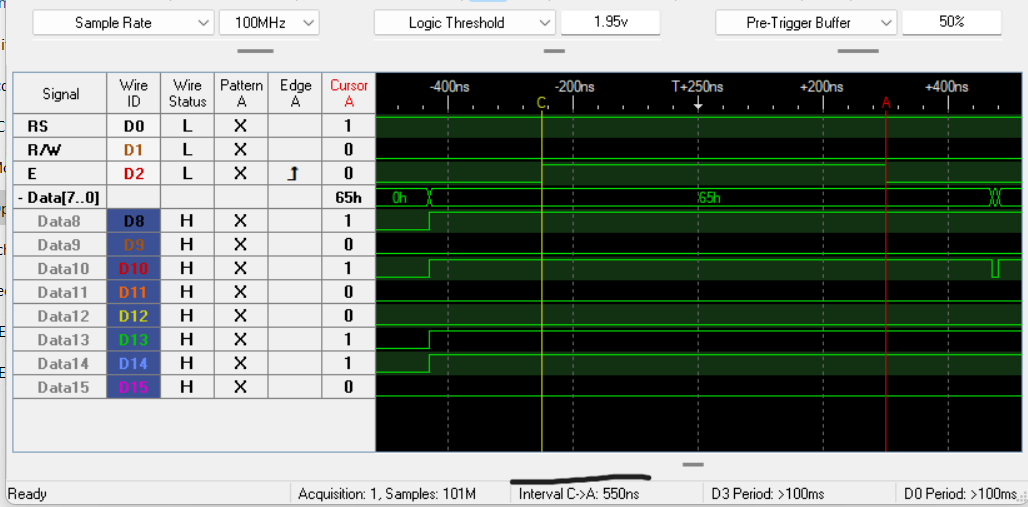


**SCL rise time (132ns):**

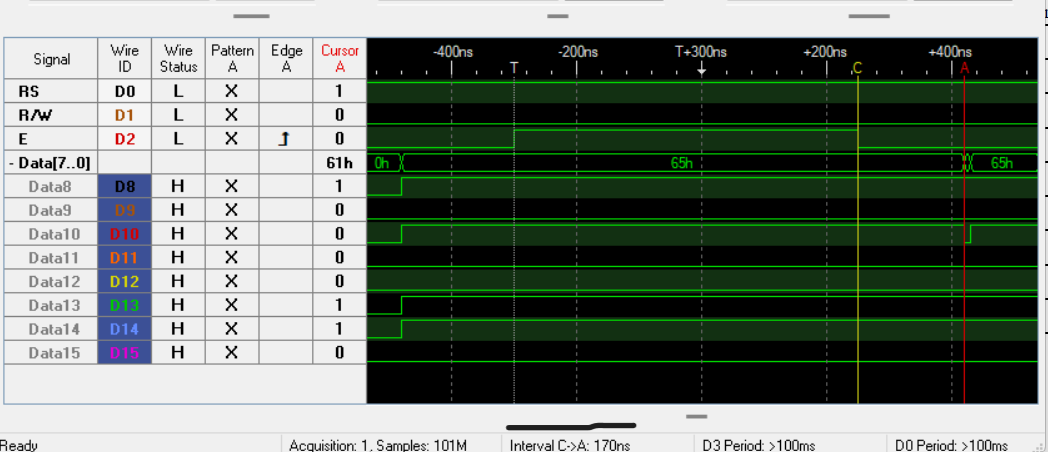
**Part 2**

**TIMING ANALYSIS OF LCD AC CHARACTERISTICS:**

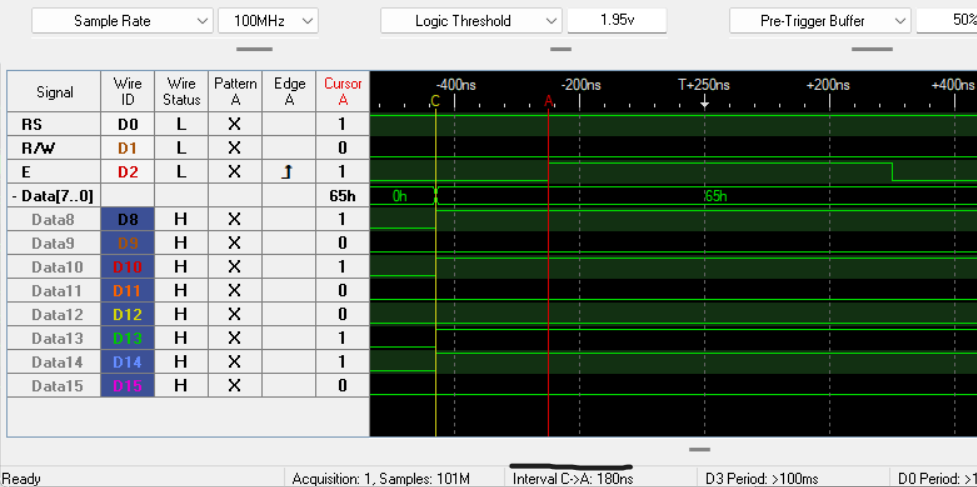
1. **Address set-up time (37.98 us): The time between change of R/W bit and E bit**



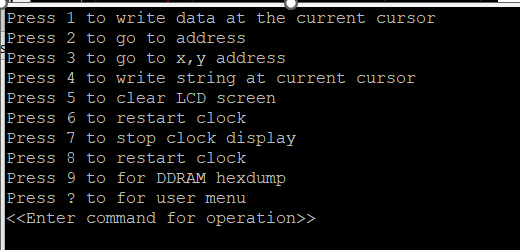
1. **Enable pulse width (550ns): The on time period of Enable (E) pin**
2. **Write data hold time (170ns):**

****

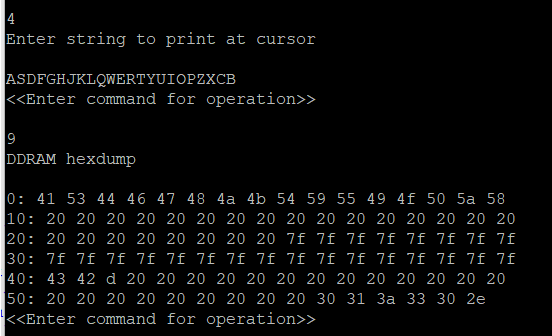
1. **Write data set-up time (180ns):**

****

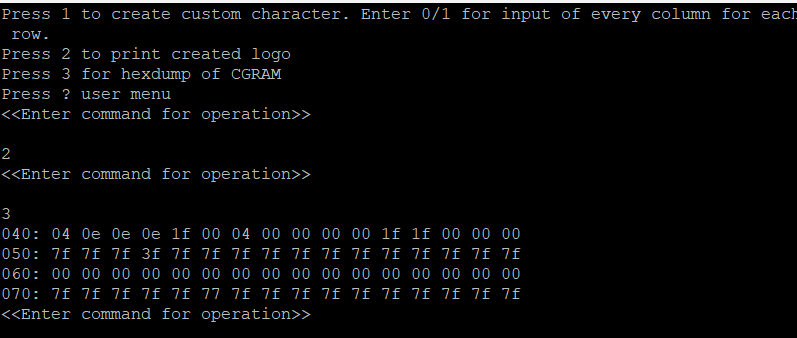
**LCD User Interface:**

****

The user interface for the LCD commands is shown in the image below. Various options are presented to write data at present cursor, go to a location on the LCD, go to (x,y) row and column on the LCD, write a string, clear the LCD, start/stop/reset the clock, print the DDRAM hexdump and get help on commands.

****

When the character ‘9’ is entered, the DDRAM hexdump is presented as shown in the image.

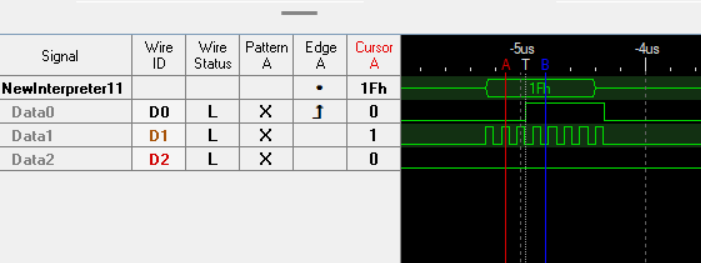


In the next image, the user interface for custom character program is shown. When the character ‘3’ is pressed, the CGRAM hexdump is presented.

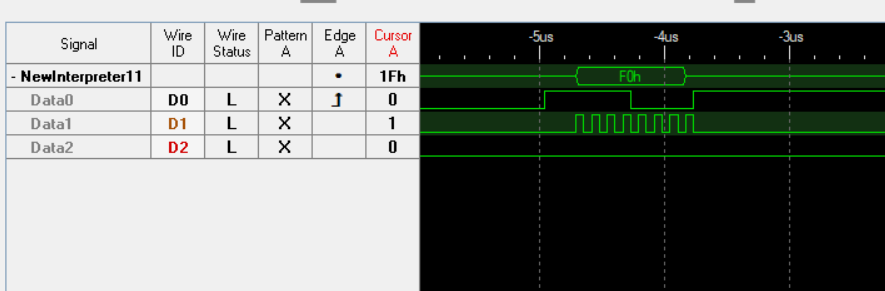
## Part 3

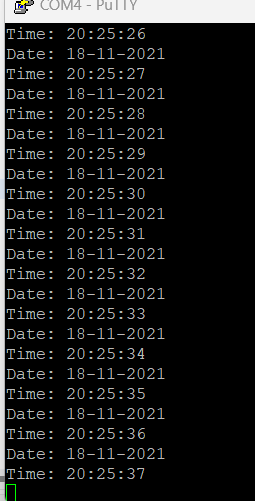
**DAC Implementation with the use of SPI controller:**

The DAC-MCP4802 is implemented on the STM32 board to get a square wave of 1Hz. The write command register is of 2 bytes where bit 15 is set to 0 to select DACa output, bit 13 is set to 0 to get (VOUT = 2 \* VREF \* D/4096), bit 12 is set to 1 for active mode operation. The data (FFh) is written into from bits 11-4. Using the SPI interpreter on the LogicPort with the pins- chip select, SDI and SCK, the following waveforms have been obtained.



First byte of register

Second byte

**Real time clock implementation:**

The real-time clock on the STM32 board is implemented using the Wakeup **interrupt** to get more precise timing. The date and time values are transmitted to PUTTY using the UART in blocking mode.

* Another feature presented during the sign-off is Independent Watchdog Timer.

**SUBMISSION QUESTIONS**

a) What operating system (including revision) did you use for your code development?

Windows 11- 64-bit operating system, x64-based processor

b) What compiler(s) (including revision) did you use?

 Small Device C Compiler (SDCC)

c) What exactly (include name/revision if appropriate) did you use to build your code (an IDE, make/makefile, or command line)?

STM32CubeIDE by ST

d) Did you install and use any other software tools to complete your lab assignment?

No.

e) Did you experience any problems with any of the software tools? If so, describe the problems.

No.