

Lab#4 Report

LCD/I2C/Bit-Bang/EEPROM/Code Integration

Code Organization and Integration

The Code for Lab 4 is organized in 5 files:

1. Main4.c
Contains the main program and functions and the UI code and the function selections and top level functions which pre-process user inputs for easy integration with header and specific function files.
2. I2c.c
Contains all the implementation of the I2C functions the basic functions were obtained from the SDCC website (<http://sdccokr.dl9sec.de/resources.htm>) where I have chosen the Basic functions for I2C bus (msi2cbasic.zip) as a base for modifications to the header and .c files later.
3. Lcd.c
Contains the implementation of all the LCD related functions required for running the LCD. The header file and .c file on which I built my functions were obtained from the Paulmon2 website. (<http://home.iae.nl/users/pouwaha/lcd/lcd.shtml>)
4. I2c.h
Contains all the function declarations and hardware definitions of SCL and SDA
5. Lcd.h
Contains the definitions of various LCD commands and the PINS of the LCD interface according to the memory map.

GENERAL Q/A

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

Ans. I am using Windows 8 for my code development using codeblocks
Windows 8Pro Ver 6.2 (Build 9200)
Code::Blocks 12.11

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

Ans. SDCC 2.6.0 Rev 4289

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

Ans. Auto configured make.exe by Code::Blocks

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

Ans. Tera Term Pro Web Ver 3.1.3

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

Ans. Took a lot of time for me to figure out the settings on Code::blocks to start compiling on Windows 8

LCD Interfacing

Hardware

The LCD display provided has four rows and 16 columns. It can display 64 characters at once.

The Address map for the LCD display is as shown below

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Row0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Row1	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
Row2	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Row3	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F

The address for the first row is from 0x00 to 0x0F. The next row is from 0x40 to 0x4F and so on as shown in the figure.

The addressing for the LCD display is discontinues because they are manufactured keeping in consideration multiplexing of displays.

The DDRAM addresses are if 7 bits and the highest bit signify which row of memory is involved.

If we compare the address in the first row and the one just below it we see that there is only difference of one bit between the two addresses.

A potentiometer connected between Vcc and Vss pins can be adjusted to get a proper contrast on the LCD.

Generation of Control Signal for the LCD

The generation of Enable signal for the LCD is being done via the SPLD (Atmel TF16V8C) chip with the following logic.

$$E = \neg(WR_AL \ \& \ RD_AL) \ \& \ (A12 \ \& \ A13 \ \& \ A14 \ \& \ A15);$$

The LCD should be enabled when we are writing or reading data to/from the LCD. Hence the ANDing of the WR(write) and RD(read) signals. Since both are active low and the Enable signal (E) is supposed to be active high we logical NOT the result to get a complement signal.

The second portion is based on the memory map addressing used for the LCD which is discussed after this section.

The Enable only goes high when Address lines A12 to A15 are high.

Memory Mapping of LCD

The memory map is made such that The LCD should perform the following functions the corresponding four locations which can be accessed using the MOVX instruction.

LCD_COMMAND_WR	0xF000	#Writes the command to the LCD RS =0 /R/W =0
LCD_STATUS_RD	0xF001	#Reads the status of the LCD RS =0 /R/W =1
LCD_DATA_WR	0xF002	#Writes data to the LCD RS =1 /R/W =0
LCD_DATA_RD	0xF003	#Reads the data from the LCD RS =1 /R/W =1

The above mapping results in the following connections:

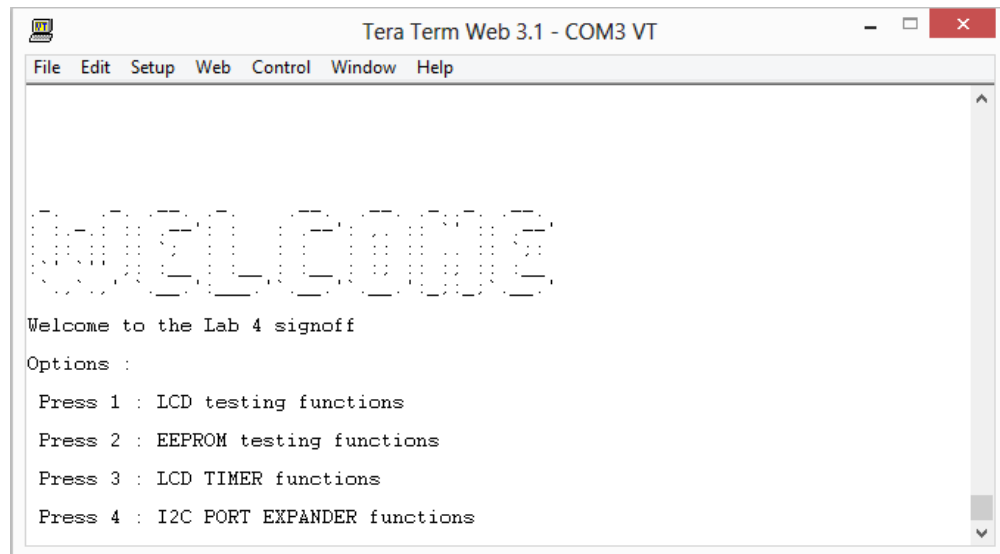
Latched pins A0 and A1 are connected to R/W and RS pins in the following manner

A0 =>R/W

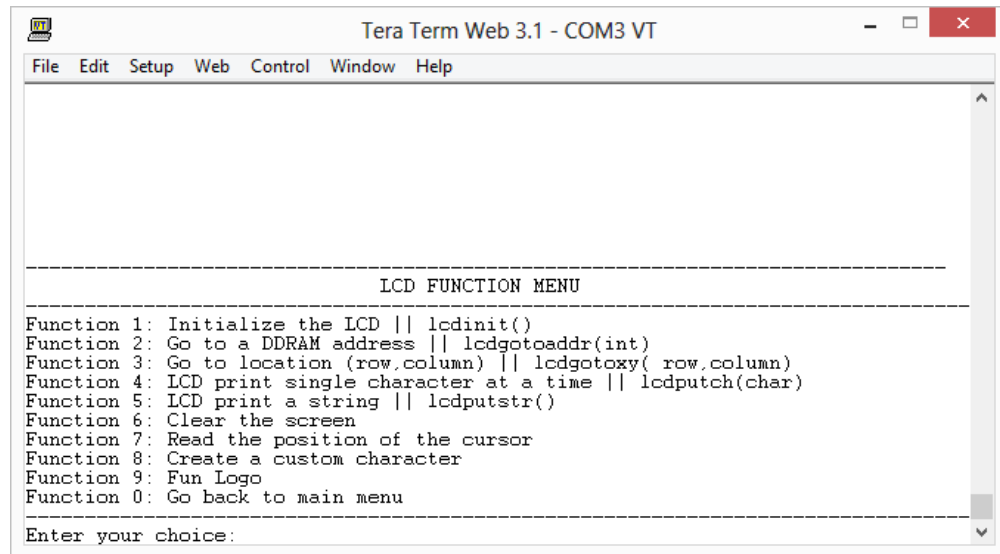
A1 =>RS

Drivers and Software for the LCD.

Below is the Welcome Screen with the Functional menu



To access the LCD Functions the User inputs "1" to go to the LCD Functions Menu which is as shown below



The LCD Function menu presents the user with the following options:

Function 1: Initialize the LCD || lcdinit()

Function 2: Go to a DDRAM address || lcdgotoaddr(int)

Function 3: Go to location (row,column) || lcdgotoxy(row,column)

Function 4: LCD print single character at a time || lcdputch(char)

Function 5: LCD print a string || lcdputstr()

Function 6: Clear the screen #(Additional Function Implemented)

Function 7: Read the position of the cursor #(Additional Function Implemented)

Function 8: Create a custom character

Function 9: Fun Logo

Function1: Initialize the LCD

This function initializes the LCD by using the "lcdbusywait()" function.

The lcdbusywait() function keeps polling the busy flag by masking the rest of the bit using the following logic

```
void lcdbusywait(void)
{
    while (lcd_status_rd & 0x80);
}
```

Here lcd_status_rd is defined as :

```
volatile xdata at LCD_STATUS_RD unsigned char lcd_status_rd;
```

Below is the result of initialization on the LCD as showing in the photo



The LCD gets initialized by the following set of commands

```
void lcdinit(void)
{
    lcd_command(LCD_CONFIG_CMD);
    lcd_command(LCD_ON_CMD);
    lcd_command(LCD_CLEAR_CMD);
}
```

Here the LCD_CONFIG_CMD & the LCD_ON_CMD and other definitions are provided in the "lcd.h" file .

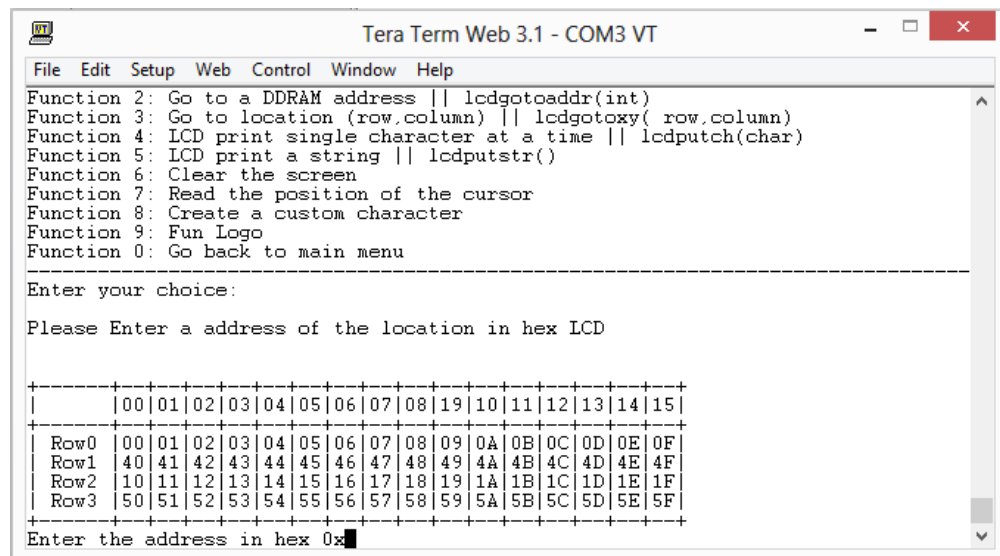
The `lcd_command()` functions calls an `lcdbusywait()` function every time and then writes the command to the LCD as shown below.

```
void lcd_command(unsigned char cmd)
{
    lcdbusywait();
    lcd_command_wr = cmd;
}
```

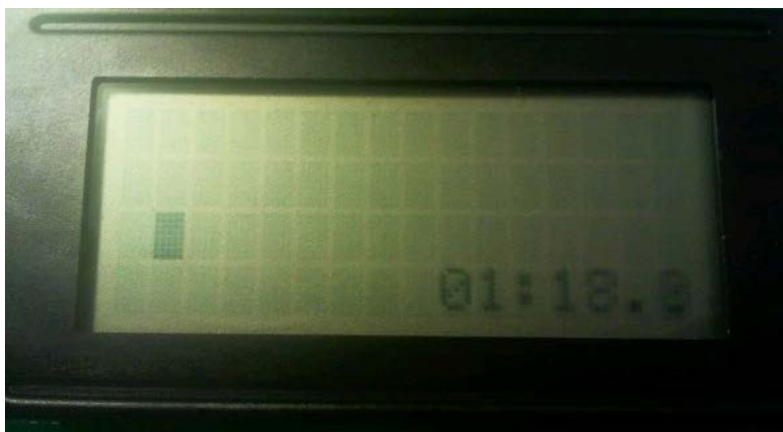
Function2: Go to DDRAM address (Hex input)

This function accepts prompts the user to input the address in hex in a valid range and then sets the cursor to the corresponding position.

The prompt Screen for the function is as shown below:

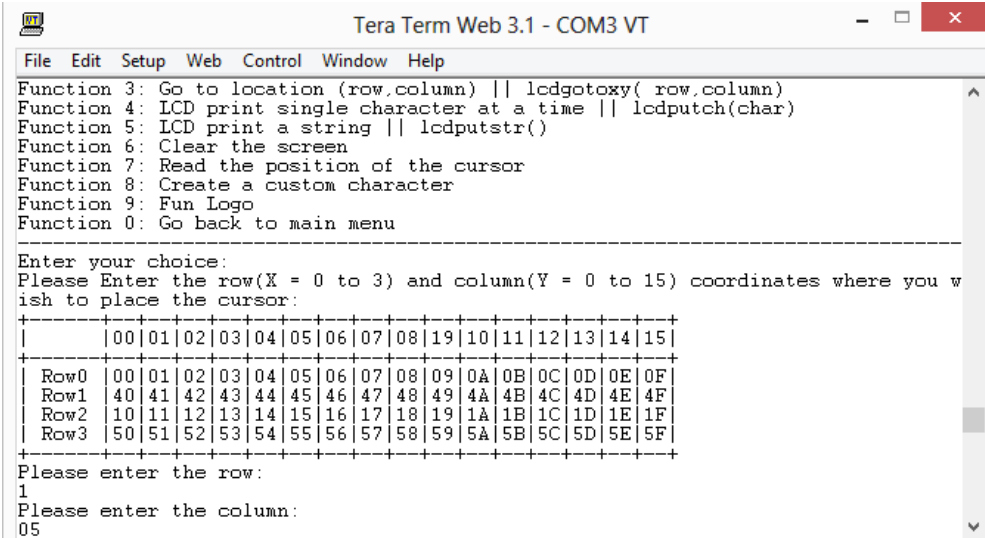


For the input "0x11" the cursor shifts to the corresponding location as shown below :



Function3: Go to location (row ,column)

The lcdgotoxy(row, column) function accepts arguments row and column and converts the row/column coordinates to addresses and calls the lcdgotoaddr() function.

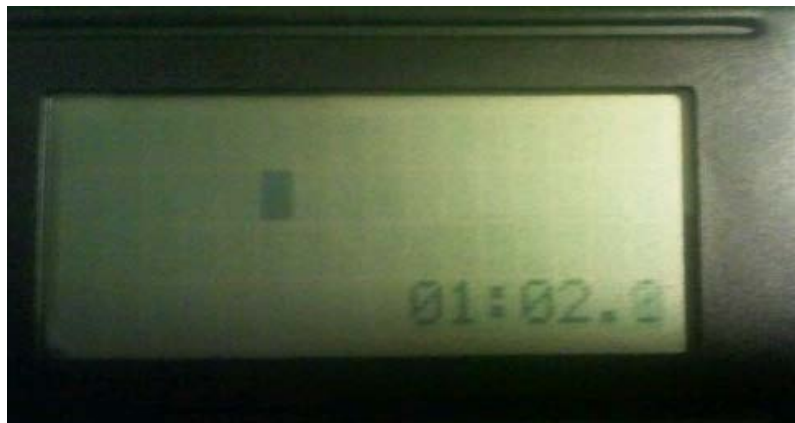


```

Tera Term Web 3.1 - COM3 VT
File Edit Setup Web Control Window Help
Function 3: Go to location (row,column) || lcdgotoxy( row,column)
Function 4: LCD print single character at a time || lcdputch(char)
Function 5: LCD print a string || lcdputstr()
Function 6: Clear the screen
Function 7: Read the position of the cursor
Function 8: Create a custom character
Function 9: Fun Logo
Function 0: Go back to main menu
-----
Enter your choice:
Please Enter the row(X = 0 to 3) and column(Y = 0 to 15) coordinates where you wish to place the cursor:
+-----+
| 00|01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|
+-----+
| Row0|00|01|02|03|04|05|06|07|08|09|0A|0B|0C|0D|0E|0F|
| Row1|40|41|42|43|44|45|46|47|48|49|4A|4B|4C|4D|4E|4F|
| Row2|10|11|12|13|14|15|16|17|18|19|1A|1B|1C|1D|1E|1F|
| Row3|50|51|52|53|54|55|56|57|58|59|5A|5B|5C|5D|5E|5F|
+-----+
Please enter the row:
1
Please enter the column:
05
  
```

Once the option for the lcdgotoxy() is selected it displays the Row Column Locations for the available addresses of the LCD.

The result of entering row =1 and column =5 is as follows;



Function 4: Print character to LCD

This function prints accepts ASCII characters and prints them to the screen.

Below is the screen for the user prompt and the output for the LCD.

```
Tera Term Web 3.1 - COM3 VT
File Edit Setup Web Control Window Help
+---+ Row3 |50|51|52|53|54|55|56|57|58|59|5A|5B|5C|5D|5E|5F|
+---+-----+
Please enter the row:
1
Please enter the column:
05

-----
LCD FUNCTION MENU
-----
Function 1: Initialize the LCD || lcdinit()
Function 2: Go to a DDRAM address || lcdgotoaddr(int)
Function 3: Go to location (row,column) || lcdgotoxy( row,column)
Function 4: LCD print single character at a time || lcdputch(char)
Function 5: LCD print a string || lcdputstr()
Function 6: Clear the screen
Function 7: Read the position of the cursor
Function 8: Create a custom character
Function 9: Fun Logo
Function 0: Go back to main menu
-----
Enter your choice:Invalid Input
Please enter a character to be printed to the LCD:█
```

A character input 'c' to the prompt is given which yields the following output



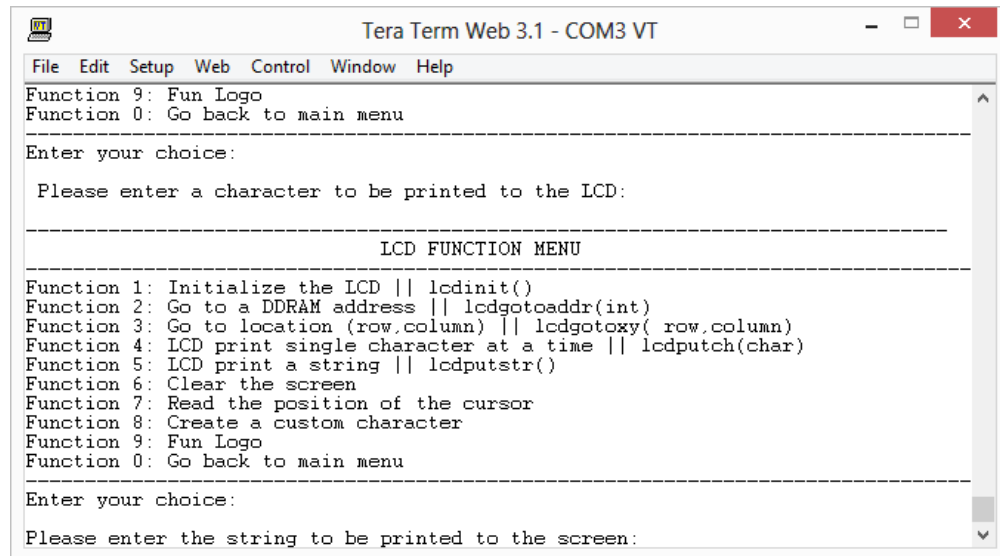
Result of printing different ASCII symbols on the screen. The code is built in such a way that the characters get printed in a row1 to row 2.



Function1: Print a string to the LCD

This function utilizes the `lcdputch()` function to print string input by the user. Similar to `lcdputchar()` the wraparound for the text goes from row 0,1,2,3 and back to 0.

Below is the prompt screen for the string function is as shown below



```
Tera Term Web 3.1 - COM3 VT
File Edit Setup Web Control Window Help
Function 9: Fun Logo
Function 0: Go back to main menu
-----
Enter your choice:
Please enter a character to be printed to the LCD:
-----
LCD FUNCTION MENU
-----
Function 1: Initialize the LCD || lcdinit()
Function 2: Go to a DDRAM address || lcdgotoaddr(int)
Function 3: Go to location (row,column) || lcdgotoxy( row,column)
Function 4: LCD print single character at a time || lcdputch(char)
Function 5: LCD print a string || lcdputstr()
Function 6: Clear the screen
Function 7: Read the position of the cursor
Function 8: Create a custom character
Function 9: Fun Logo
Function 0: Go back to main menu
-----
Enter your choice:
Please enter the string to be printed to the screen:
```

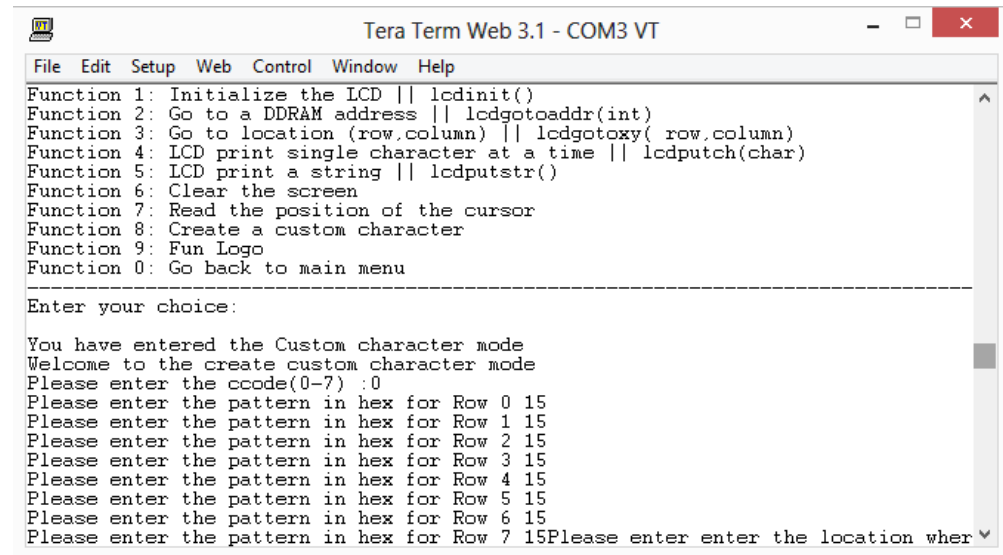
The result of entering the sting , “Hello my name isTEJAS JOSHI and this is the LAB4 report” is as show below.



Function2: Create a custom Character

This function prompts the user to provide hex input for patterns for different CGRAM locations and then asks the user to input the cursor position where the character is to be printed.

Below is a screenshot of the prompt screen and a sample custom character being created using the function.



```

Tera Term Web 3.1 - COM3 VT
File Edit Setup Web Control Window Help
Function 1: Initialize the LCD || lcdinit()
Function 2: Go to a DDRAM address || lcdgotoaddr(int)
Function 3: Go to location (row,column) || lcdgotoxy( row,column)
Function 4: LCD print single character at a time || lcdputch(char)
Function 5: LCD print a string || lcdputstr()
Function 6: Clear the screen
Function 7: Read the position of the cursor
Function 8: Create a custom character
Function 9: Fun Logo
Function 0: Go back to main menu
-----
Enter your choice:
You have entered the Custom character mode
Welcome to the create custom character mode
Please enter the ccode(0-7) :0
Please enter the pattern in hex for Row 0 15
Please enter the pattern in hex for Row 1 15
Please enter the pattern in hex for Row 2 15
Please enter the pattern in hex for Row 3 15
Please enter the pattern in hex for Row 4 15
Please enter the pattern in hex for Row 5 15
Please enter the pattern in hex for Row 6 15
Please enter the pattern in hex for Row 7 15Please enter enter the location wher

```

Pattern entered is 0x15 printed at location row 0 and column 0 as shown below



Function2: Fun Logo

The fun logo I have designed is that of a car with the following pattern configuration.

	1	2	3	4	5		6	7	8	9	10	C1	C2
0												0	0
1												7	18
2												0D	0C
3												19	0E
4												7F	1F
5												6B	1B
6												14	14
7												08	08

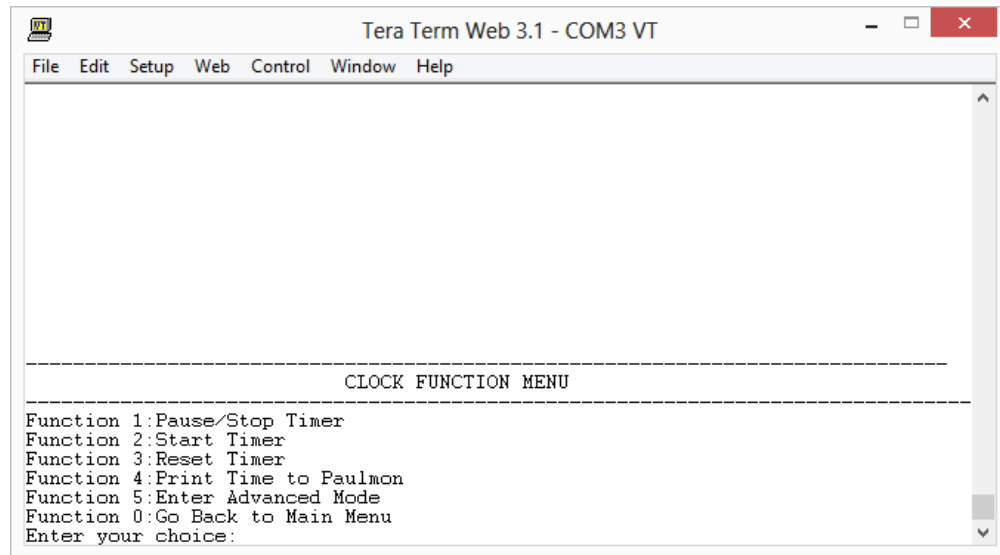
The values in column C1 and C2 are the patterns in Hex.

The output of the fun logo is as shown below:



Timer/Clock on the LCD

The timer clock functions are provided through as separate menu after selection from the main menu.



The timer functions provide the user ability to start, stop and reset the timer. The advanced mode timer keeps count with 1/100th of a second resolution. For this function I am utilizing Timer 1 in 16 bit mode "2".

```
SCON = 0x50;          //Serial mode 8-bit UART variable frequency
    TMOD |= 0x20;       //Timer mode 2 auto reload 8-bit
    TH1=0xFD;          //9600 Baud rate
    TL1=0x00;
    TR1=1;             //Generate baud rate for serial comm
    TI=1;              //Set TI=1 so program doesn't wait to
clear it for first time

    //timer0 initialization
    TMOD |= 0x01;      // timer 0 , Mode 1
    TH0 = 0xFC;        // time for 1 ms will be from count incrementing from
FC65H to FFFFh
    TL0 = 0x70;
    //TR0 = 0;
    IE|= 0x82;
```

Initially the timer count was 0xFC68 which caused the timer to run slower than usual but was replaced to 0xFC70 so it was more precise.

The timer operates in two modes . Normal Mode and Enhanced Mode

The timers print the time with different resolution . The normal mode maintains an accuracy of 1/10 of a second and the Enhanced mode maintains accuracy of 1/100th of a second.

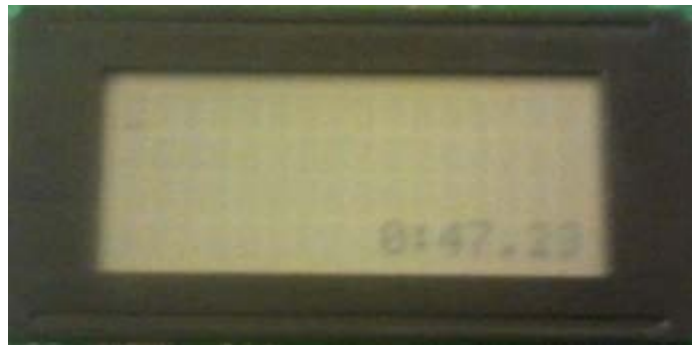
The time is printed to the LCD screen in the following formats as shown below:

Minute:Seconds:100 th /sec MM:SS.SS
--

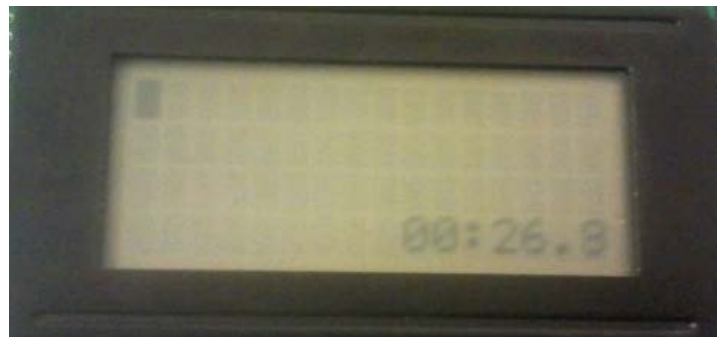
Minute:Seconds:10 th /sec MM:SS.S
--

Below are screenshots of the timer in both modes

Enhanced Mode:



Normal Mode:



Additional Functions Implemented

There are certain additional functions I have implemented for the LCD and the time/clock

Function 6: Clear the screen function

This function send the command to the LCD to clear the screen. The function is implemented using the blow code:

```
#define LCD_CLEAR_CMD 0x01

void lcdclear(void)
{
    lcd_command(LCD_CLEAR_CMD);
}
```

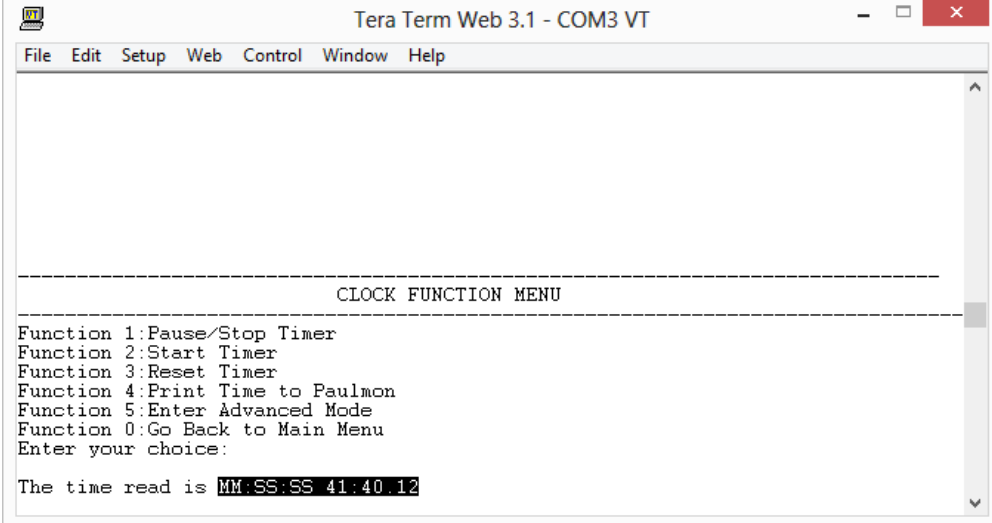
Function 7: Read cursor position

Sometimes it is necessary to obtain the cursor position. I have implemented this function because it is required to read the cursor position before going into the timer ISR.

The current cursor position is displayed in decimal format of the address.

Timer/ Clock Function 7: Print time to Terminal/Paulmon

This function prints the LCD time to the terminal:

A screenshot of a terminal window titled "Tera Term Web 3.1 - COM3 VT". The window has a menu bar with "File", "Edit", "Setup", "Web", "Control", "Window", and "Help". The terminal content shows a menu titled "CLOCK FUNCTION MENU" with several options: "Function 1:Pause/Stop Timer", "Function 2:Start Timer", "Function 3:Reset Timer", "Function 4:Print Time to Paulmon", "Function 5:Enter Advanced Mode", and "Function 0:Go Back to Main Menu". Below the menu, it says "Enter your choice:". At the bottom, it displays "The time read is MM:SS:SS 41.40.12" where the time is highlighted in a dark box.

```
Tera Term Web 3.1 - COM3 VT
File Edit Setup Web Control Window Help

-----
                        CLOCK FUNCTION MENU
-----
Function 1:Pause/Stop Timer
Function 2:Start Timer
Function 3:Reset Timer
Function 4:Print Time to Paulmon
Function 5:Enter Advanced Mode
Function 0:Go Back to Main Menu
Enter your choice:

The time read is MM:SS:SS 41.40.12
```

Timing Analysis of the LCD

The above screenshot shows the timing analysis of the LCD signals RS, R/W and the E signals along with the Data bus.

All the timings of the signals meet the specifications given in the datasheet

Bus Timing Characteristics

Write Operation

Item	Symbol	Min	Typ	Max	Unit	Test Condition
Enable cycle time	t_{cycE}	1000	—	—	ns	Figure 27
Enable pulse width (high level)	PW_{EH}	450	—	—		
Enable rise/fall time	t_{Er}, t_{Ef}	—	—	25		
Address set-up time (RS, R/W to E)	t_{AS}	60	—	—		
Address hold time	t_{AH}	20	—	—		
Data set-up time	t_{DSW}	195	—	—		
Data hold time	t_H	10	—	—		

Timing Characteristics

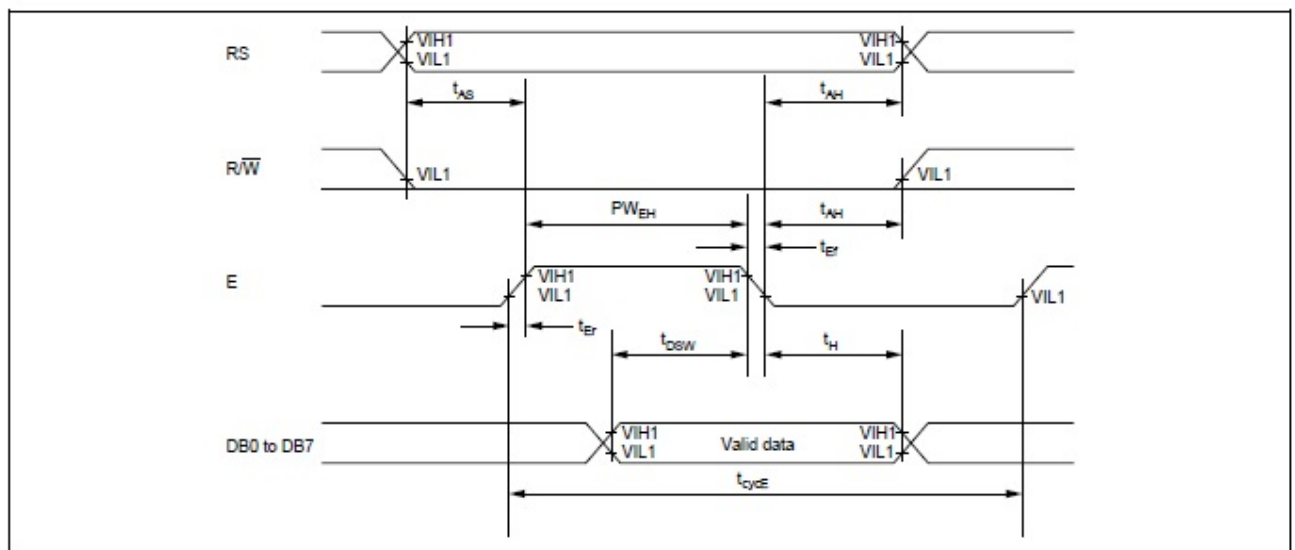
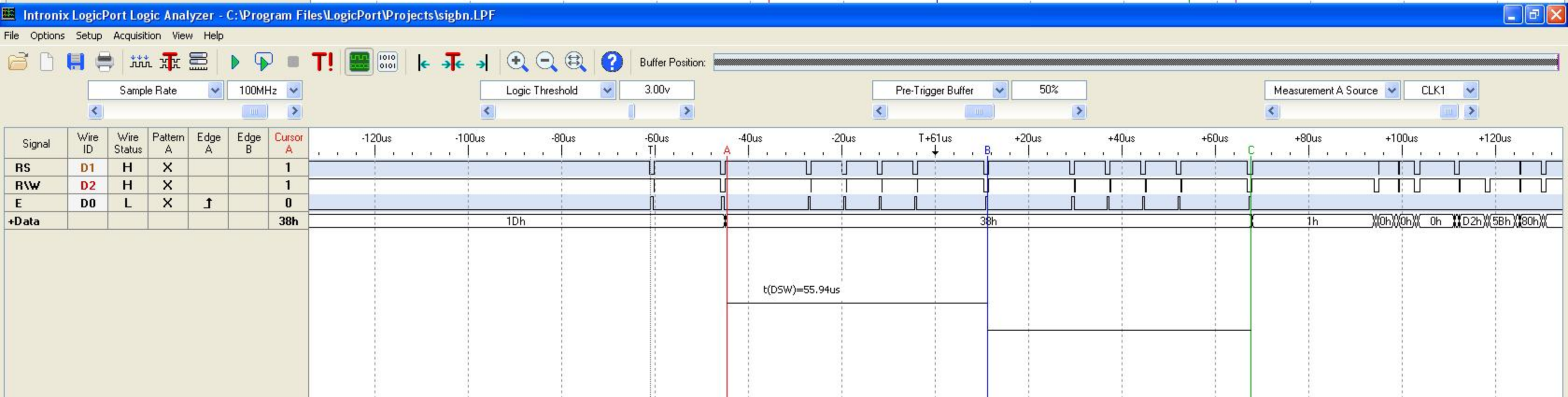
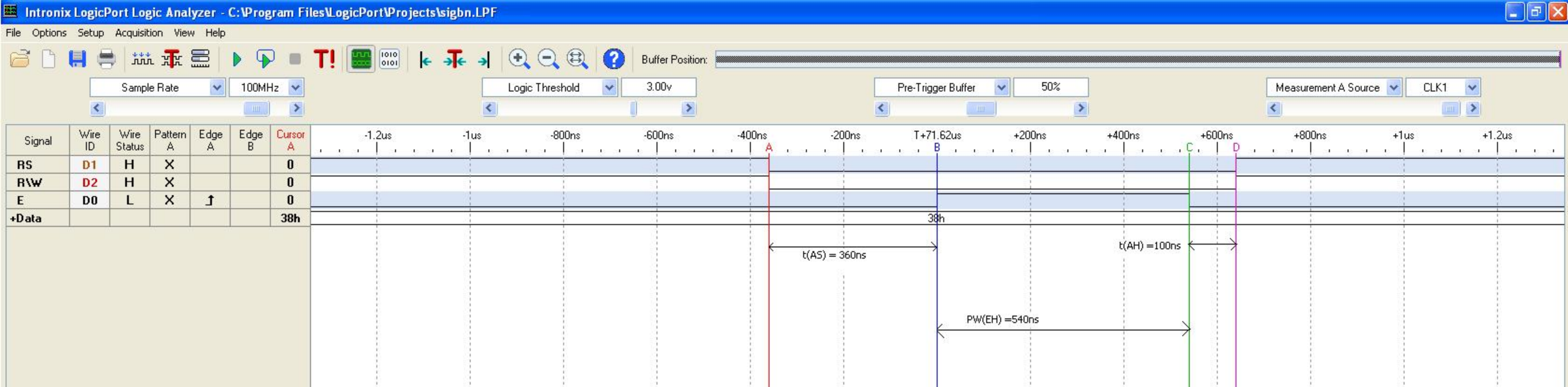


Figure 27 Write Operation



SERIAL EEPROM (I2C)

Hardware

For the 24LC16B 16K-bit Serial EEPROM which is implemented using the I2C bus is implemented using the port 1 pins 0 and 1 connected in the following fashion.

P1_0 => SCL

P1_1 => SDA

Since the i2c is asynchronous serial communication the timing and sequence of the clock and data signals is important.

EEPROM addresses:

The eeprom has 16 k of memory which is divided into 8 "page blocks" Each page block contains 16"pages" and there are 16 "words" in each page

Hence $8 \text{ page blocks} \times 16 \text{ pages} \times 16 \text{ words} \times 8 \text{ bytes} = 16384 \text{ bits} = 16 \text{ kb}$

EEPROM Read/Write Modes:

The EEPROM can be used to serially read or write in two modes.

1. Random Read/Write

This mode is used for a single "word" read or write operation. It involves the following sequence of signals on the SCL and SDA lines.

The sequence of signals for the Read/write is as follows

- a. Start
- b. Slave Address
- c. Page Block
- d. Sending Read or Write
- e. Checking for Acknowledge
- f. Sending the 8 bit address and checking for Acknowledge
- g. Reading or Writing the Data and checking for Acknowledge
- h. Sending a Stop

2. The Sequential Write/Read

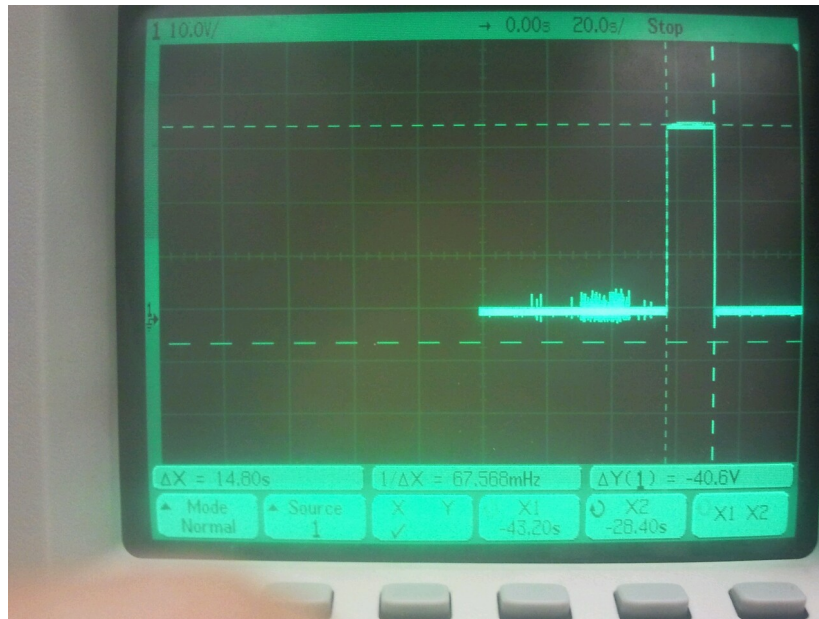
a. The Sequential Write

This mode is similar in all steps except after the first set of the data write/ & ACK frame 15 more frames of the same can be written before sending the stop bit

b. The Sequential Read

Steps a. through c. are the same then a Write signal is sent . Then the same frame is sent with a read signal and 16 frames are sent by the EEPROM over the bus for the read operation.

Block Write Timing Observed on Toggled Port Pin P1_7



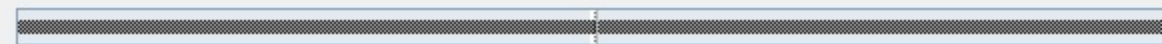
in X1 mode
14.80sec
improved to
13.25sec

Timing analysis for EEPROM:

Below is the analysis of the i2c signals for the EEPROM



Buffer Position:

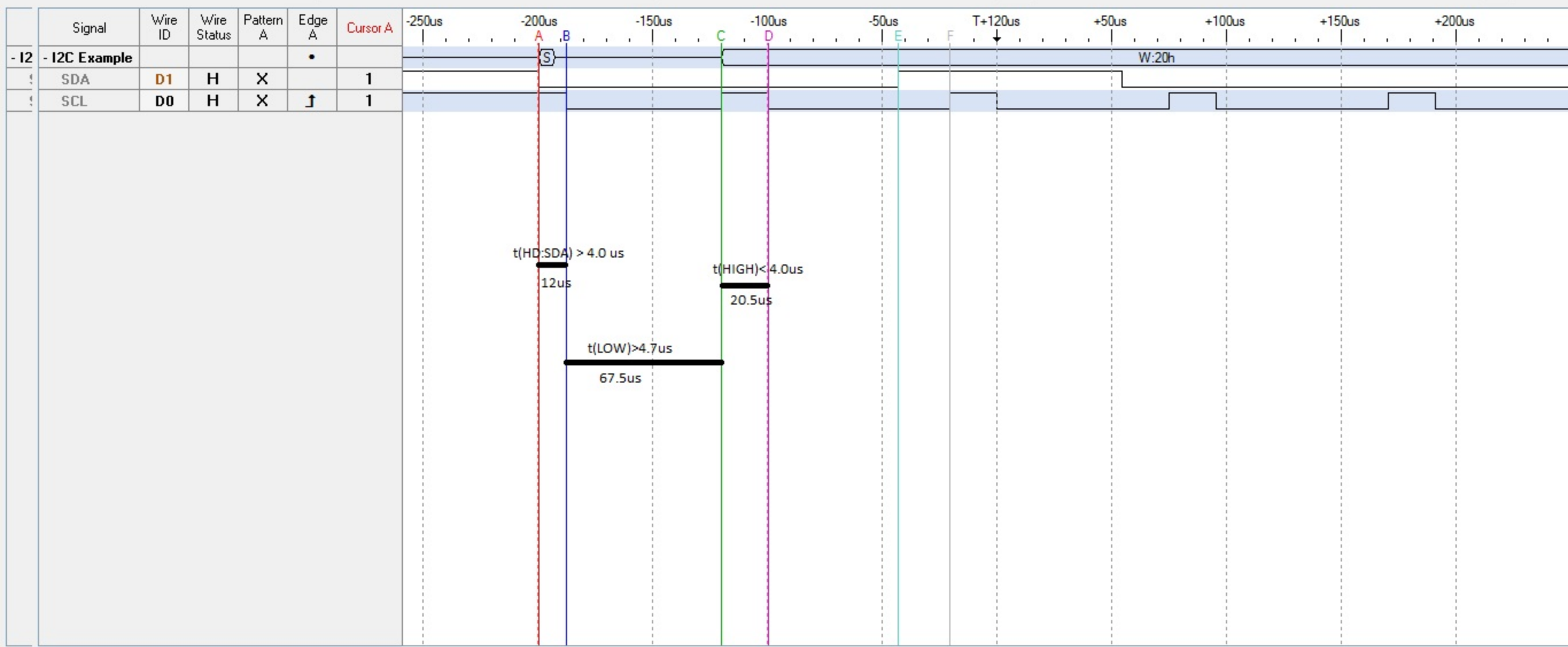


Sample Rate 100MHz

Logic Threshold 1.40v

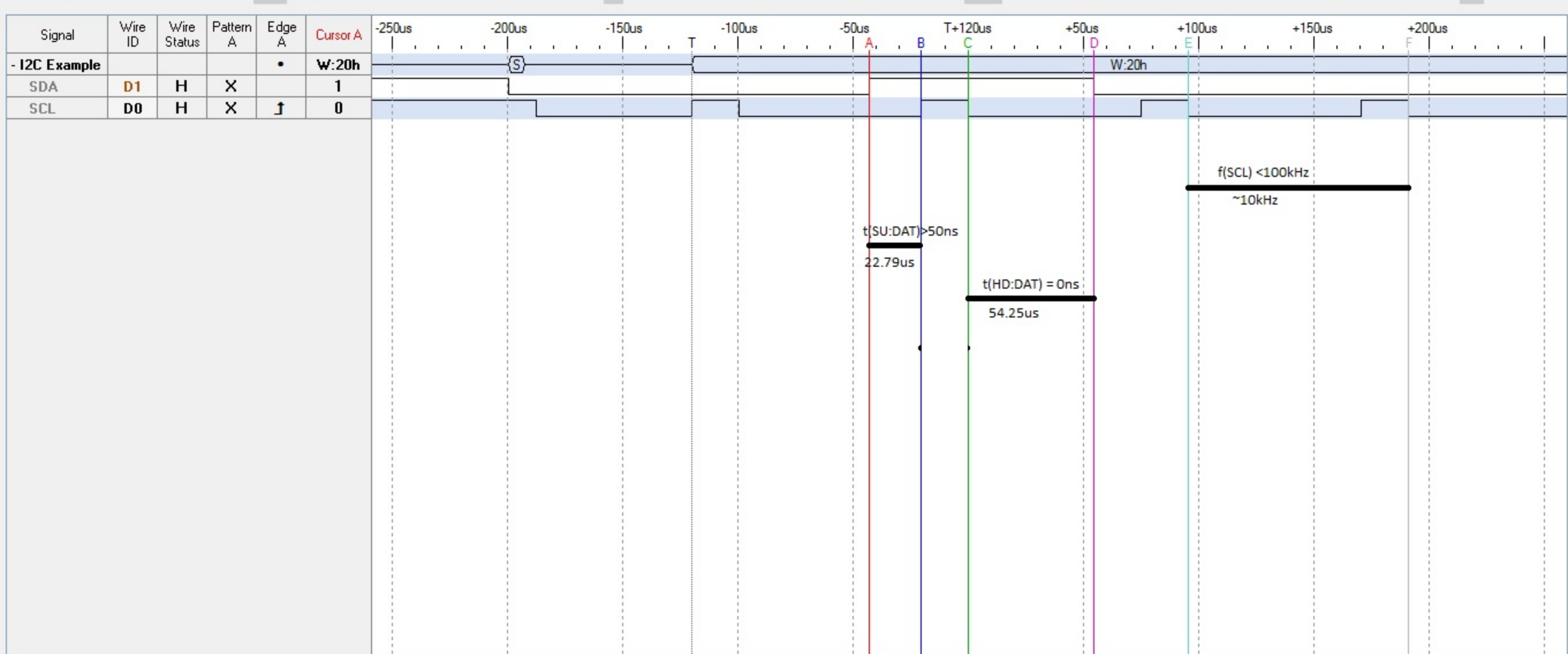
Pre-Trigger Buffer 50%

Measurement A Source CLK1



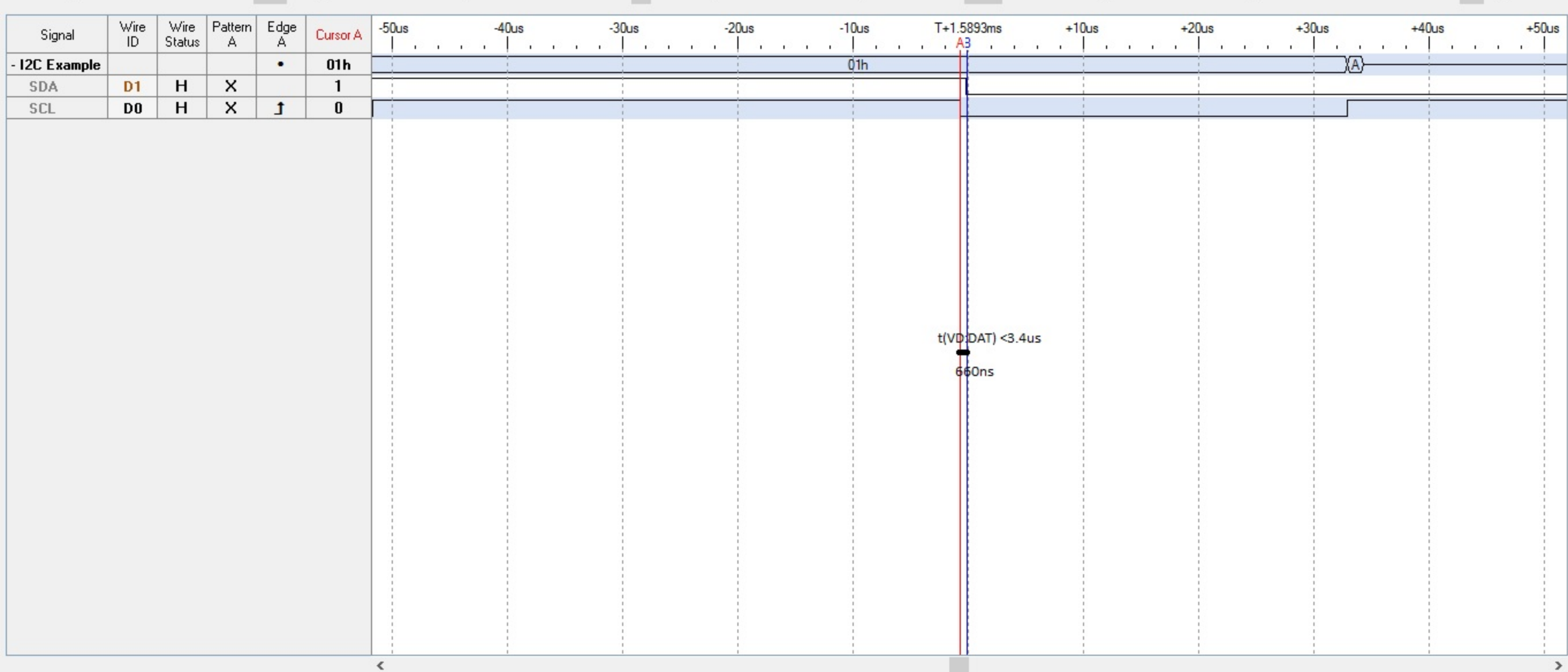
Buffer Position:

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



Buffer Position:

Sample Rate: 100MHz Logic Threshold: 3.00v Pre-Trigger Buffer: 50% Measurement A Source: CLK1

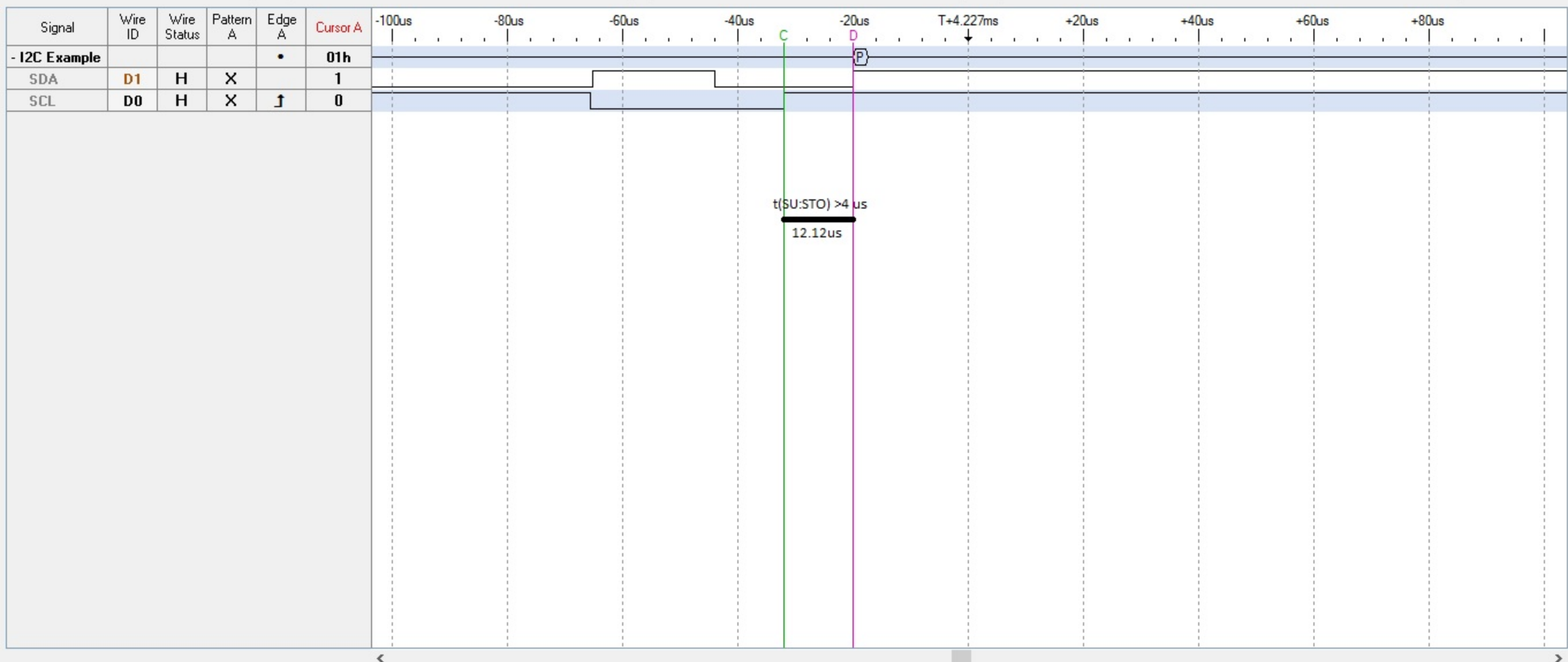




File Options Setup Acquisition View Help

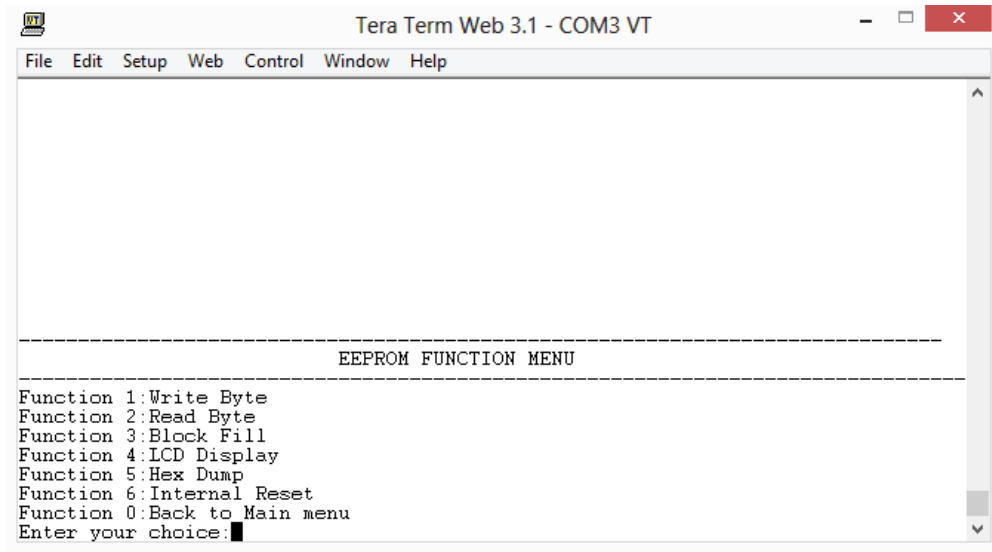
Sample Rate: 100MHz Logic Threshold: 3.00v Pre-Trigger Buffer: 50% Measurement A Source: CLK1

Buffer Position:



EEPROM functions and Code:

The EEPROM menu provides the user with Functions as shown in the screenshot below:



Function 1: Write to EEPROM

This function prompts the user to enter an address where the data is to be written and then prompts for the data in hex.

It uses the `eebytew()` function to write to the EEPROM

Function 2: Read from EEPROM

This function prompts the user to enter an address from where the data is to be read and then displays the data to the terminal.

This function uses the `eebytew()` function to read from the eeprom.

Function 3: Block Fill

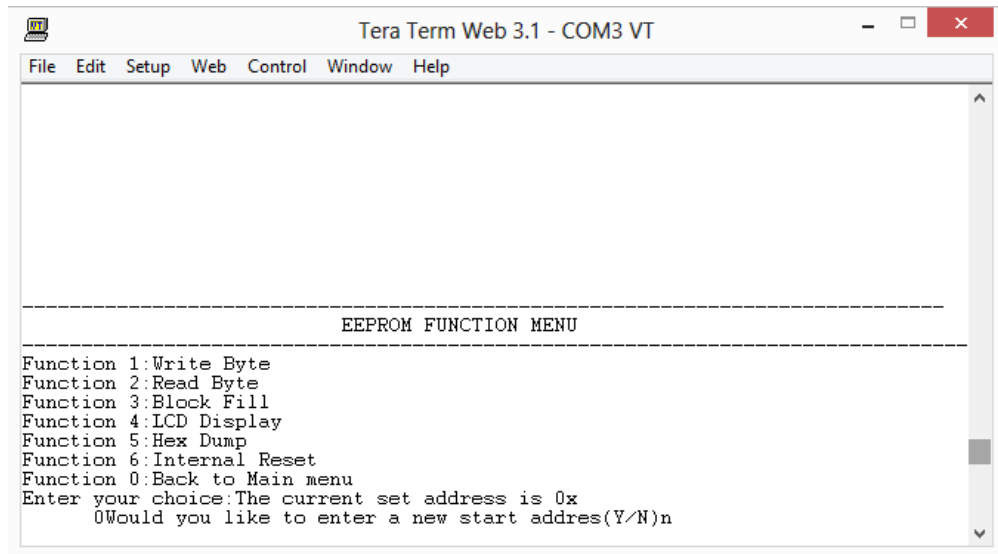
This function accepts a start address and an end address and the data to be filled in that particular block of EEPROM memory.

The start address has to be smaller than the end address or the program displays an error to the terminal.

Function 4: Print Data to the LCD

This function prompts the user for an address and/or prints the address and data from a particular location in the format `AAAA:DD` where "AAAA" is the data and "DD" is the data in hex.

The prompt screen for the PRINT to LCD data is as shown below



Below is what is displayed on the screen.

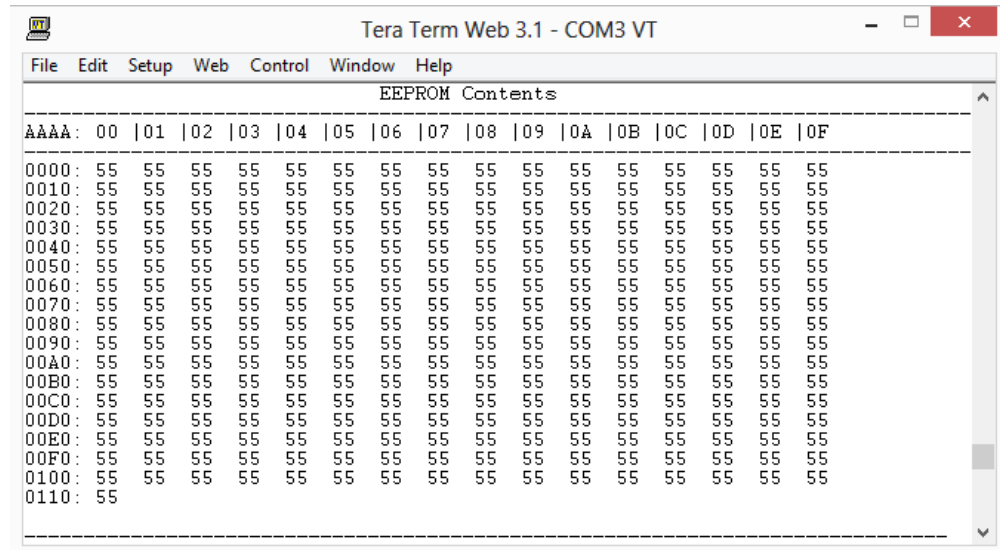


The display prints 4 data and address sets and performs an offset to print the set of next 4 data/address sets every time this function is called.

Function 5: Hex Dump

This function accepts the start and end address from the user and then dumps the content of the EEPROM to the terminal screen as specified in the requirement that the address be printed with a set of sixteen data values.

Below is a screenshot of the hex dump from the terminal for location 0x000 to 0x110



Function 6: Internal Reset

This function is responsible for bringing the EEPROM back to a valid state. This is required as a part of the system startup sequence so the EEPRO does not start in a faulty state.

The sequential starts and data (of 9 bits of 1's) and start and stop sequences bring the EEPROM to a valid state and is ignored by any other device in on the I2C bus.

Below is the screen shot of the Reset frame?

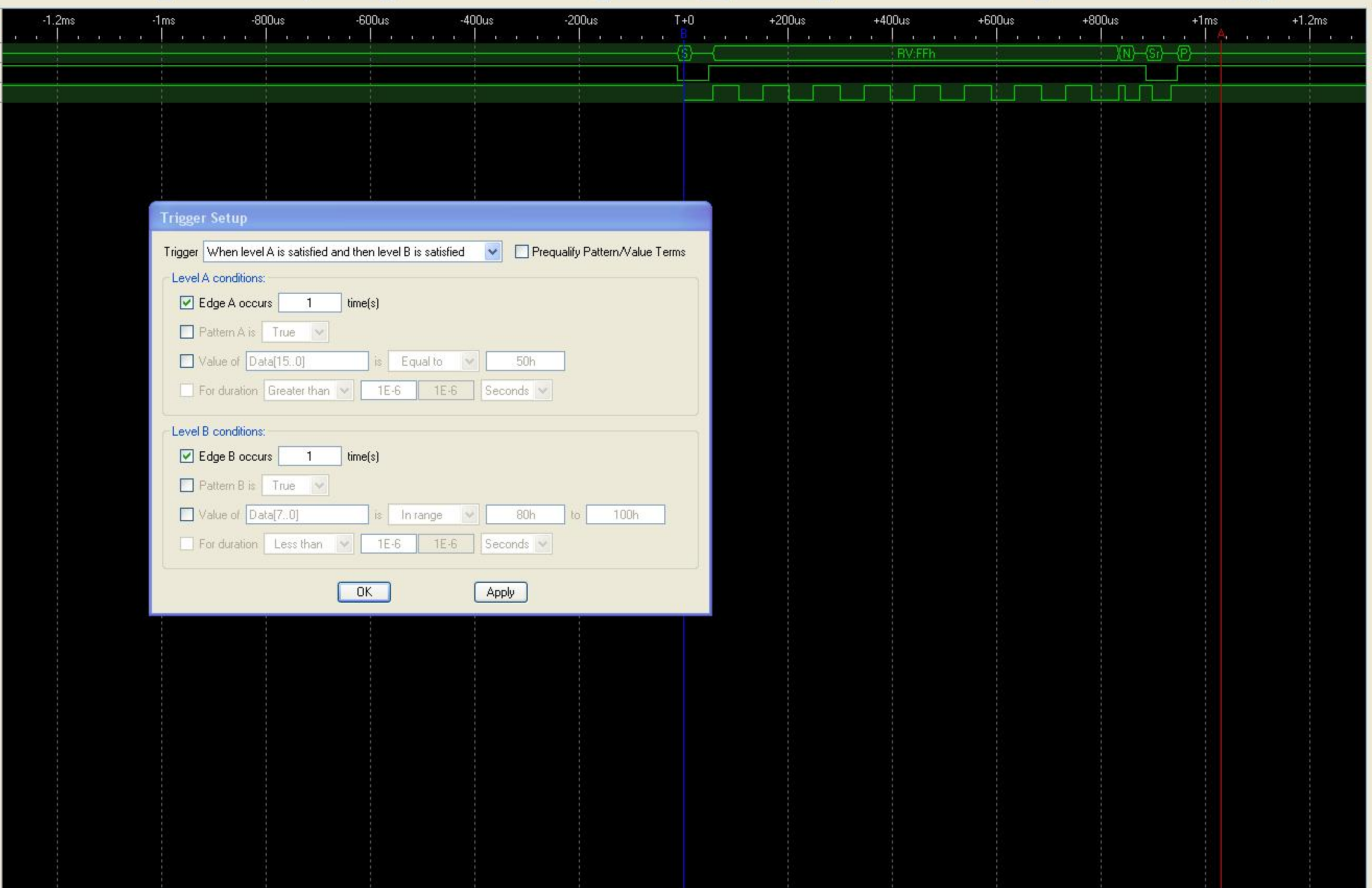
Sample Rate: 200MHz

Logic Threshold: 1.85v

Pre-Trigger Buffer: 50%

Measurement A Source: CLK1

Signal	Wire ID	Wire Status	Pattern A	Edge B	Edge A	Cursor A
- I2C Example				•	•	
SDA	D1	H	X		↓	1
SCL	D0	H	X	↓		1



Trigger Setup

Trigger: When level A is satisfied and then level B is satisfied

☐ Prequalify Pattern/Value Terms

Level A conditions:

☒ Edge A occurs 1 time(s)

☐ Pattern A is True

☐ Value of Data[15..0] is Equal to 50h

☐ For duration: Greater than 1E-6 1E-6 Seconds

Level B conditions:

☒ Edge B occurs 1 time(s)

☐ Pattern B is True

☐ Value of Data[7..0] is In range 80h to 100h

☐ For duration: Less than 1E-6 1E-6 Seconds

OK Apply

Serial I2C I/O Expander

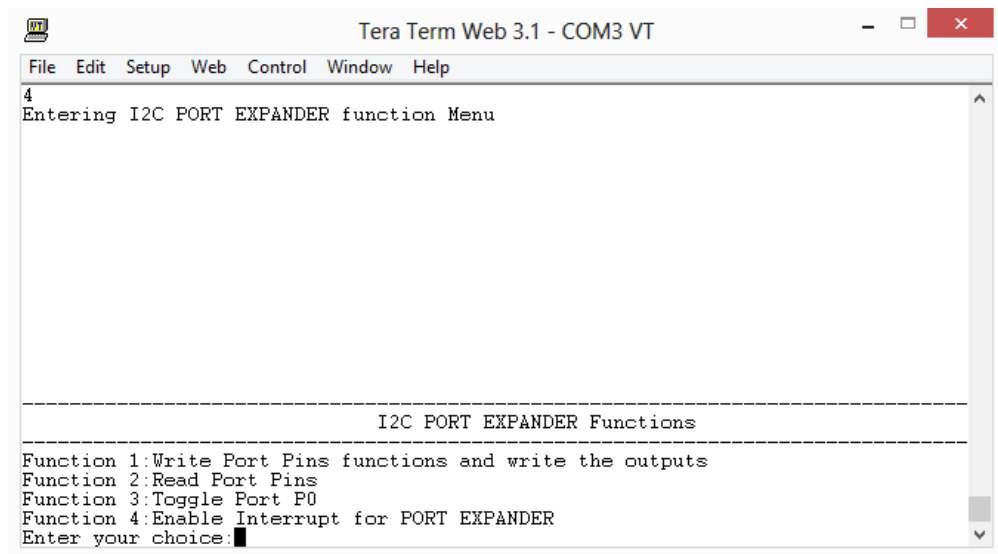
Hardware

Similar to the EEPROM, the I/O expander has a slave address of “**0100**” for identification on the I2C bus. The ports are bit-banged to provide the particular sequence for the I2C bus.

A Led is connected to Output port pin P0 and a pushbutton to the P7 pin for interrupt generation.

Code and Functions:

The I2C header and .c file is modified to include functions for the I/O expander. The code is implemented to provide the user with a Port Expander function menu as shown below:



```
Tera Term Web 3.1 - COM3 VT
File Edit Setup Web Control Window Help
4
Entering I2C PORT EXPANDER function Menu

-----
I2C PORT EXPANDER Functions
-----
Function 1:Write Port Pins functions and write the outputs
Function 2:Read Port Pins
Function 3:Toggle Port P0
Function 4:Enable Interrupt for PORT EXPANDER
Enter your choice:
```

Functions 1: Assign function to port pins

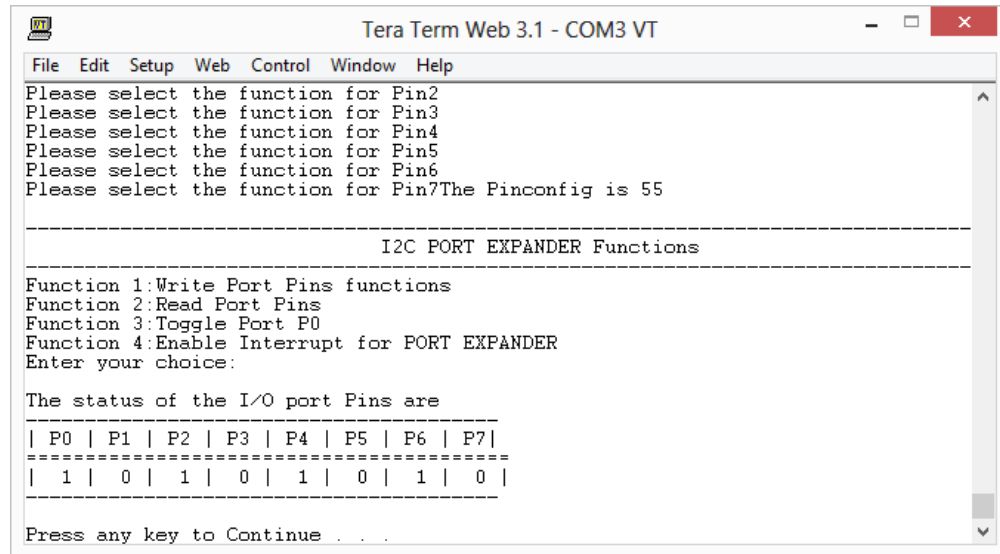
This function puts writes to the Port Expander . First sends a frame with salve address and the write function indicator.

The acknowledge then triggers the sendbyte() function to send the port pin functions and followed by a ACK and then writing to the port pins using bit masking.

Functions 2: Read Pins

This function sends a read frame to the port expander. This is followed by the status of the port pin and then the frame which contains the data. Bit-masking is used for the same function to read only pins that are configured as input.

The port pin status is displayed to the terminal in the following fashion to the terminal screen



```
File Edit Setup Web Control Window Help
Please select the function for Pin2
Please select the function for Pin3
Please select the function for Pin4
Please select the function for Pin5
Please select the function for Pin6
Please select the function for Pin7The Pinconfig is 55

-----
I2C PORT EXPANDER Functions
-----
Function 1:Write Port Pins functions
Function 2:Read Port Pins
Function 3:Toggle Port P0
Function 4:Enable Interrupt for PORT EXPANDER
Enter your choice:

The status of the I/O port Pins are
-----
| P0 | P1 | P2 | P3 | P4 | P5 | P6 | P7 |
-----
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
-----

Press any key to Continue . . .
```

Functions 3: Toggle port Pin p0

This function requires initialization of the port pin P0 as an output pin and then whenever this function is called toggles the value of the pin, which also causes the LED to toggle.

Functions 4: Interrupt Enable for Interrupt detection on INTO pin

This function utilizes the interrupt generation capability of the I/O Expander on the INT pin. It enables the Interrupt for Hardware Interrupt INTO and a pushbutton attached to the port pin P7 on the I/O expander pulls the pin high , causing a transition from 0 => 1 .

Hence generating an interrupt on the INTO pin whenever the pushbutton is pressed. The ISR is designed for the interrupt such that it causes port pin P1_7 of the processor to toggle an already connected LED for previous experiments.