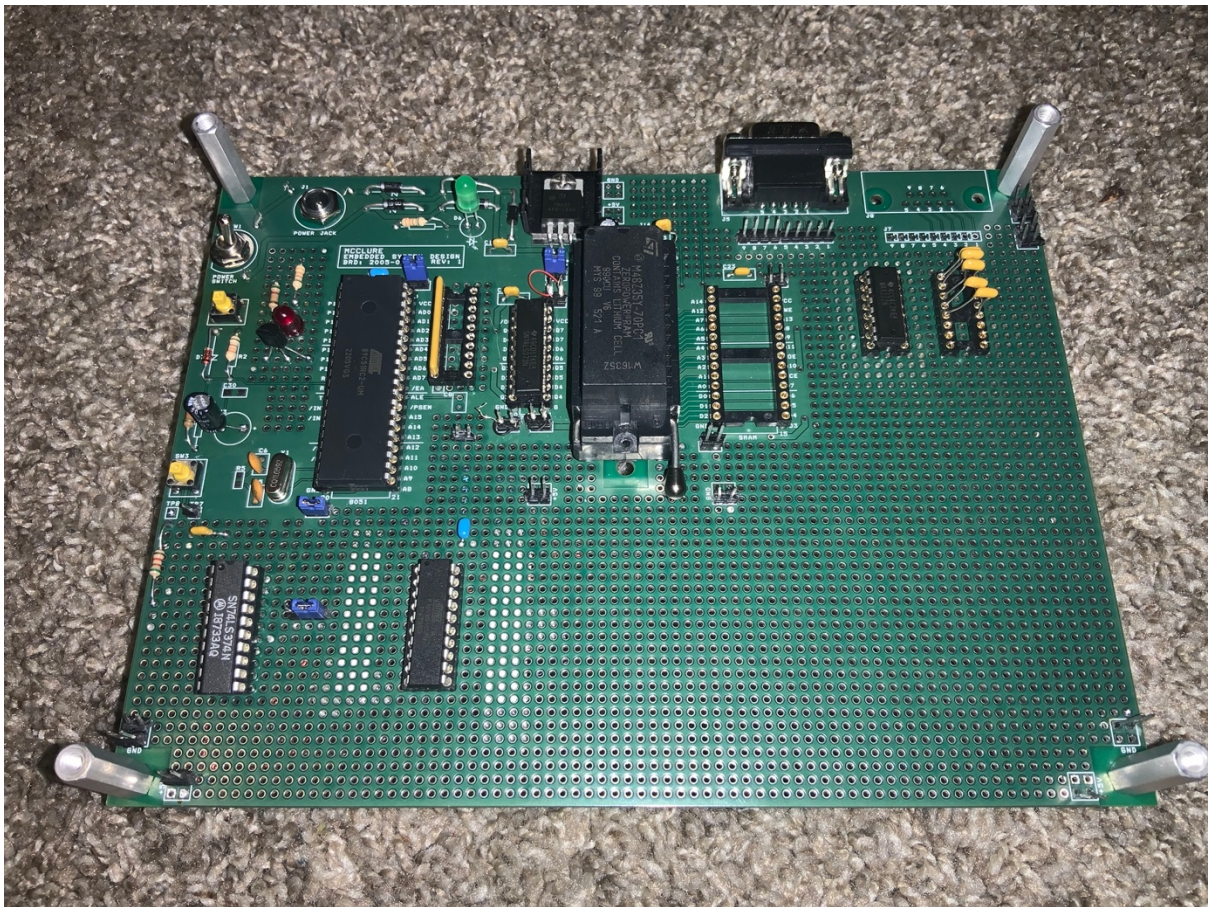
Lab 2 Writeup

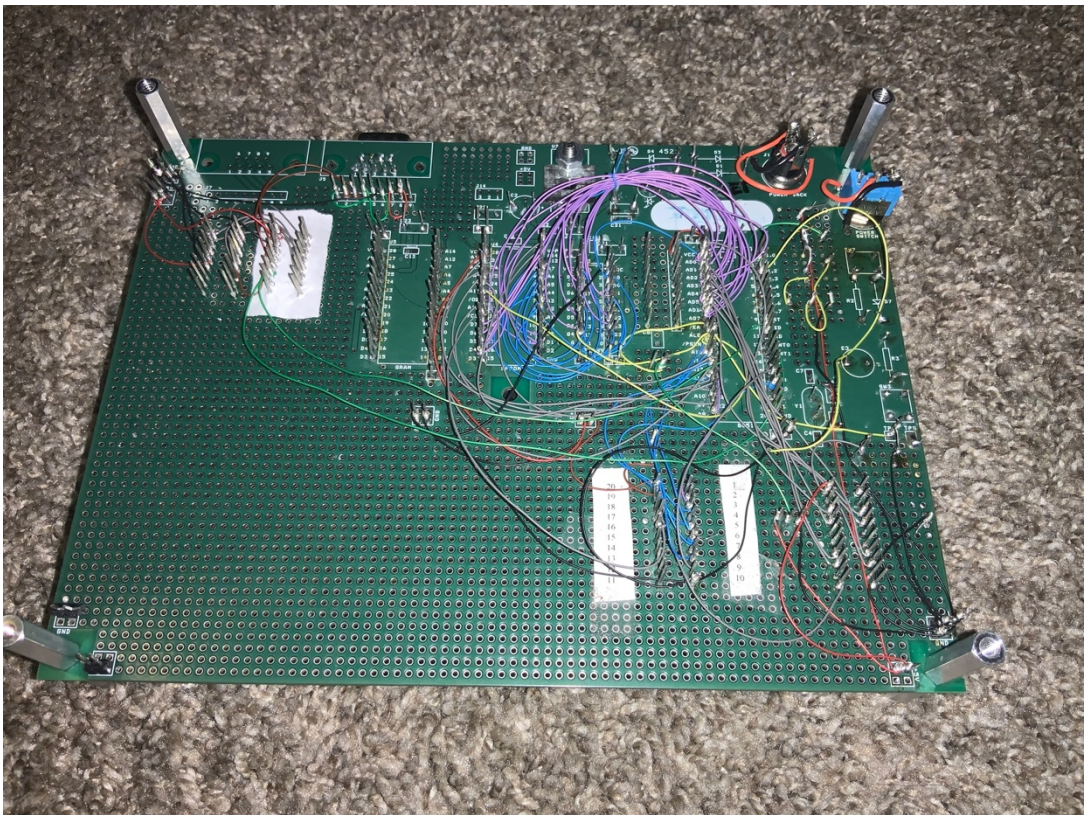
The following lab 2 write up goes through the details of work submitted by me before final submission of lab 2 part1 and part 2 elements. For the detailed folder structure please go through the following GitHub repository.
https://github.com/kiranj26/Embedded-System-Design-S22/tree/main/Labs/Lab2

Note : For both part 1 and part 2, required as well as supplemental parts has been implemented and verified.

Board Layout

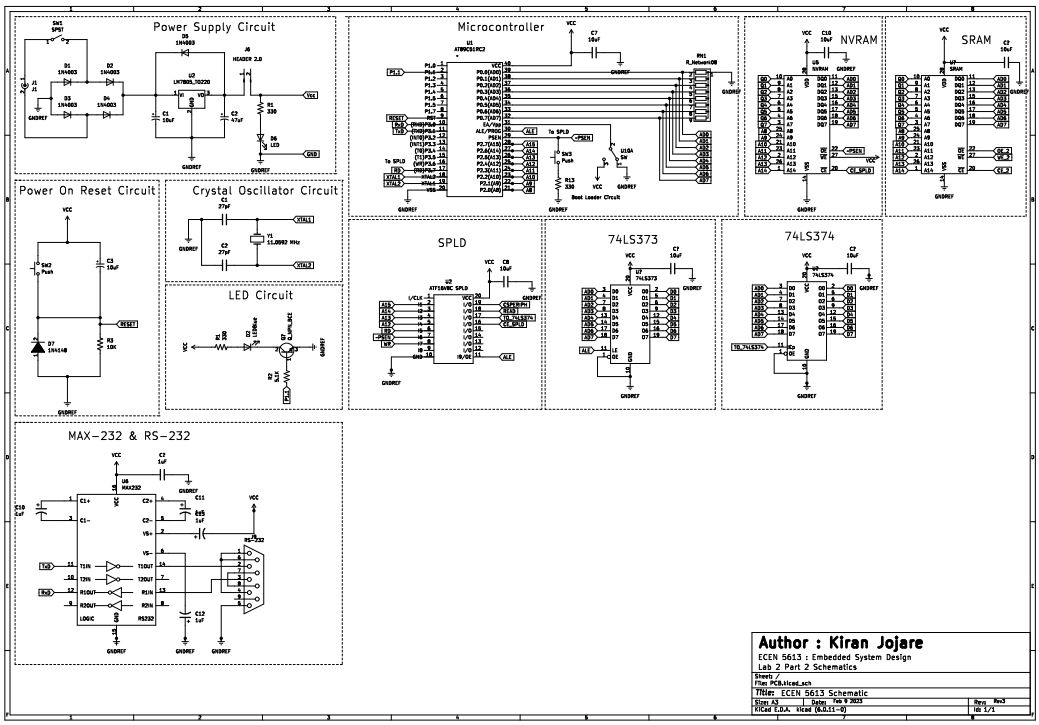
Both pictures below show the board layout from front and the back side. Please note that the EA pin on microcontroller is broken and as a workaround a header is added which is soldered to that broken EA pin.





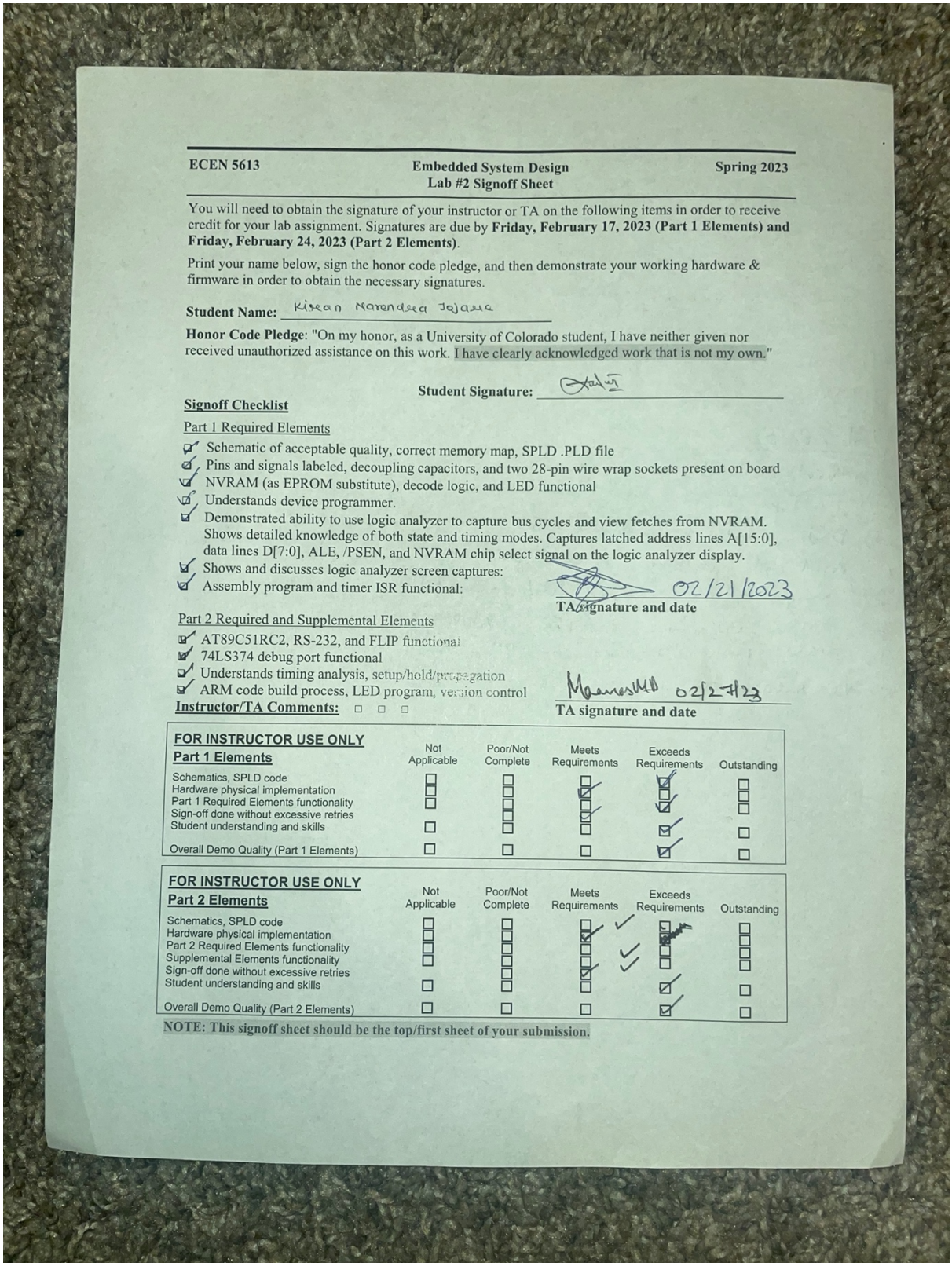
Schematic

Picture below shows the final schematic after lab 2 part1 and parts 2 submission.



Sign Off Sheets

Sign off sheet front and back page from lab 1 parts 1 submission is as follows :



ECEN 5613 **Embedded System Design** **Spring 2023**
Lab #2 Signoff Sheet

You will need to obtain the signature of your instructor or TA on the following items in order to receive credit for your lab assignment. Signatures are due by **Friday, February 17, 2023 (Part 1 Elements) and Friday, February 24, 2023 (Part 2 Elements)**.

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: Kiran Narendra Jojare

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:

Signoff Checklist

Part 1 Required Elements

- ☑ Schematic of acceptable quality, correct memory map, SPLD .PLD file
- ☑ Pins and signals labeled, decoupling capacitors, and two 28-pin wire wrap sockets present on board
- ☑ NVRAM (as EPROM substitute), decode logic, and LED functional
- ☑ Understands device programmer.
- ☑ Demonstrated ability to use logic analyzer to capture bus cycles and view fetches from NVRAM. Shows detailed knowledge of both state and timing modes. Captures latched address lines A[15:0], data lines D[7:0], ALE, /PSEN, and NVRAM chip select signal on the logic analyzer display.
- ☑ Shows and discusses logic analyzer screen captures:
- ☑ Assembly program and timer ISR functional:

02/21/2023
TA signature and date

Part 2 Required and Supplemental Elements

- ☑ AT89C51RC2, RS-232, and FLIP functional
- ☑ 74LS374 debug port functional
- ☑ Understands timing analysis, setup/hold/propagation
- ☑ ARM code build process, LED program, version control

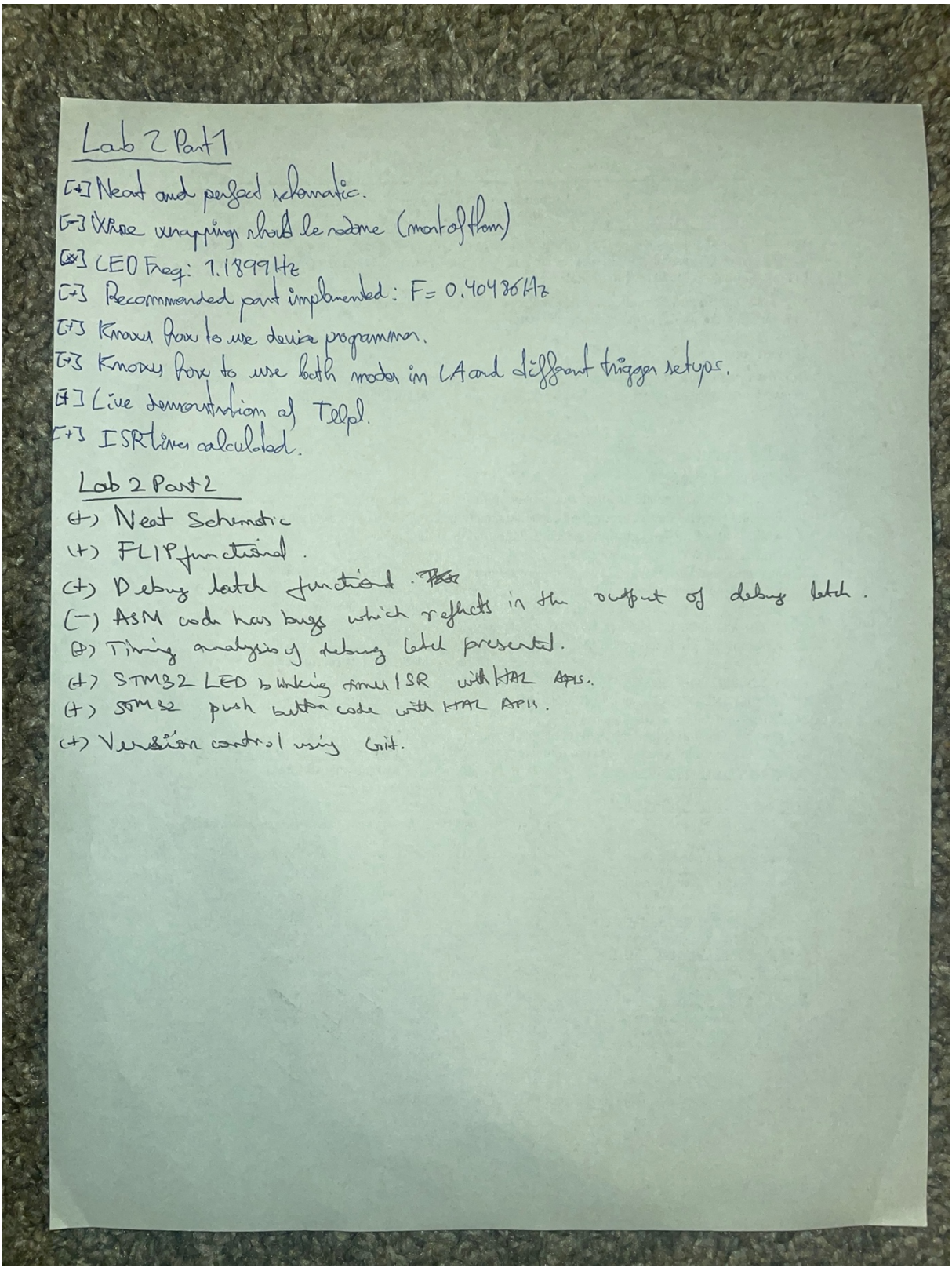
02/27/23
TA signature and date

Instructor/TA Comments: □ □ □

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

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

NOTE: This signoff sheet should be the top/first sheet of your submission.



Lab 2 Part 1

[+] Neat and perfect schematic.
[-] Wire wrapping should be same (most of them)
[x] LED Freq: 1.1899Hz
[+] Recommended part implemented: F= 0.40486Hz
[+] Knows how to use device programmer.
[+] Knows how to use both modes in LA and different trigger setups.
[+] Live demonstration of Telpl.
[+] ISR times calculated.

Lab 2 Part 2

(+) Neat Schematic
(+) FLIP functional.
(+) Debug latch functional.
(-) ASM code has bugs which reflects in the output of debug latch.
(+) Timing analysis of debug latch presented.
(+) STM32 LED blinking timer ISR with HAL APIs.
(+) STM32 push button code with HAL APIs.
(+) Version control using Git.

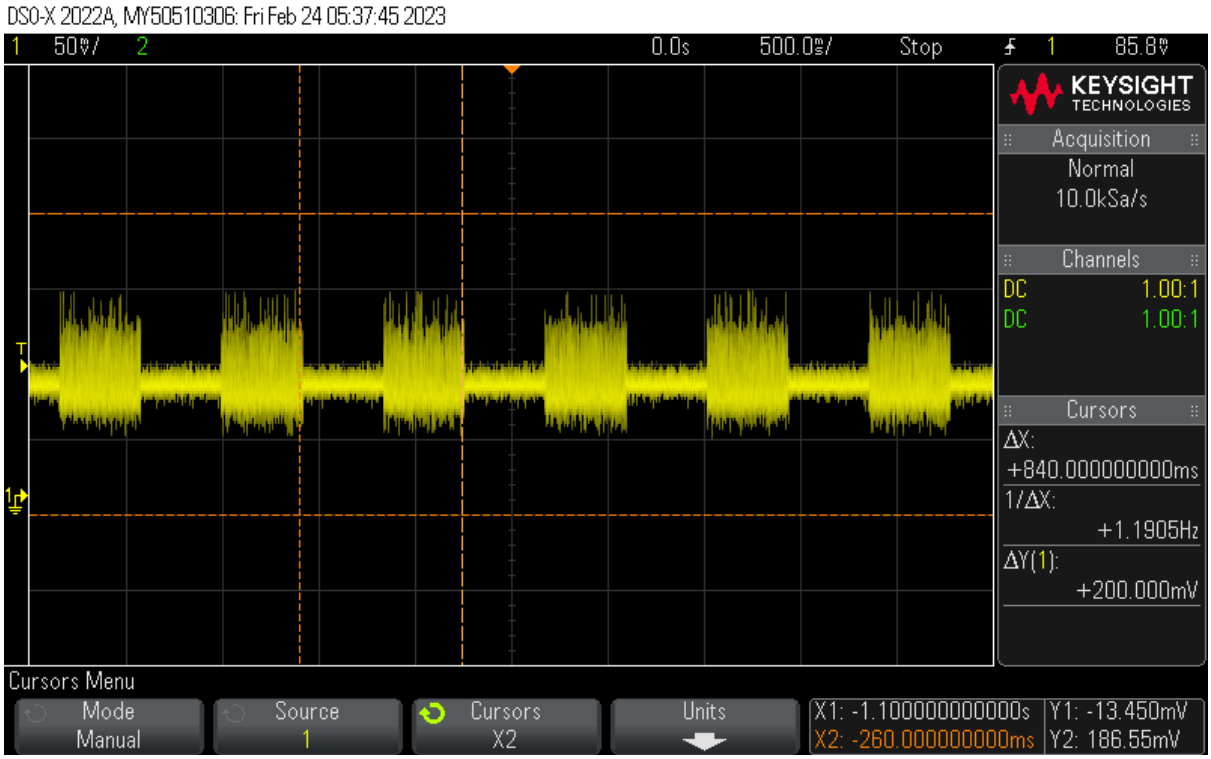
Lab 2 Part 1:

LED Blinking

C501 processor for Part 1 Elements has been chosen for this part. The assembly code has been developed which first initialize the 8051 registers and then enter an infinite loop. An ISR triggered by Timer 0 blinks an LED (by toggling a port pin) at about 1.19 Hz +/- 0.2% (on for ~0.42 seconds and off for ~0.42 seconds). A second unused port pin toggles each time the ISR executes (set the port pin to a logic high as the first instruction in your ISR and clear the port pin to a logic low immediately before you execute the RETI instruction).

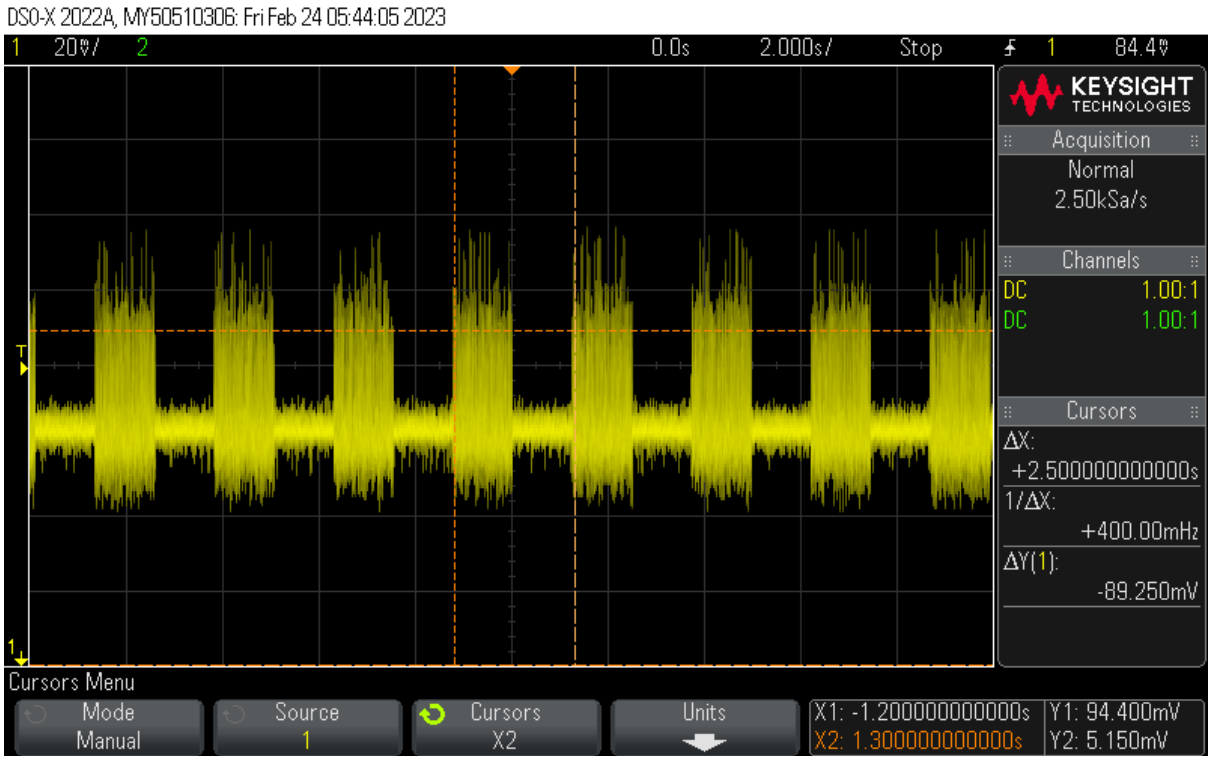
Implemented third unused port pin required to use as an input and regularly/periodically retrieved the state of the pin in your assembly code. If the pin is at a HIGH state, keep the LED blinking at the rate shown above. However, whenever the pin is detected at a LOW state, the LED blinks at a slower rate of ~0.40 Hz.

The scope output based on state of unsed input pin (P1.2 in my case) responds as follows:



The above diagram matches the requirement where the frequency of the signal is 1.19 Hz as required and LED ON and OFF time is in total 840 ms (420 ms OFF and 420 ms OFF). Note that unsed pin P1.2 was HIGH in this case requiring to keep the requirement of 1.19 Hz as it is.

The below scope output shows the behaviors of LED with unused input pin P1.3 set to LOW resulting in the LED blinking with frequency of 40 Hz.



The scope screenshot shows that LED was toggling at a frequency of 0.40 Hz with (1.25 sec ON and 1.25 Sec OFF) which matches the requirements if the unsed pin is set to LOW.

ISR calculations

To calculate the time, it takes the ISR to execute, we need to determine the number of clocks cycles each instruction takes and how many times each instruction is executed in the ISR. We'll assume that the clock frequency is 11.0592 MHz.

First, let's look at the shortest path through the ISR code, which occurs when P1.3 is low:

```
ISR_TMR:
    CPL P1.2                ; 1 cycle
    JNB P1.3, LOWER_FRQ     ; 2 cycles if not taken, 1 cycle if taken
NORMAL:
    DJNZ B,LOOP             ; 2 cycles if not taken, 3 cycles if taken
    MOV TH0,#0X17           ; 2 cycles
    MOV TL0,#0X46           ; 2 cycles
    MOV B,#6                ; 1 cycleā
    CPL P1.0                ; 1 cycle
    CPL P1.2                ; 1 cycle
    RETI                    ; 4 cycles
```

The shortest path through the ISR takes 1 + 1 + 2 + 2 + 2 + 1 + 1 + 4 = 14 cycles.
Now, let's look at the longest path through the ISR code, which occurs when P1.3 is high:

```
ISR_TMR:
    CPL P1.2                ; 1 cycle
    JNB P1.3, LOWER_FRQ     ; 2 cycles if not taken, 1 cycle if taken
NORMAL:
    DJNZ B,LOOP             ; 2 cycles if not taken, 3 cycles if taken
    MOV TH0,#0X17           ; 2 cycles
```

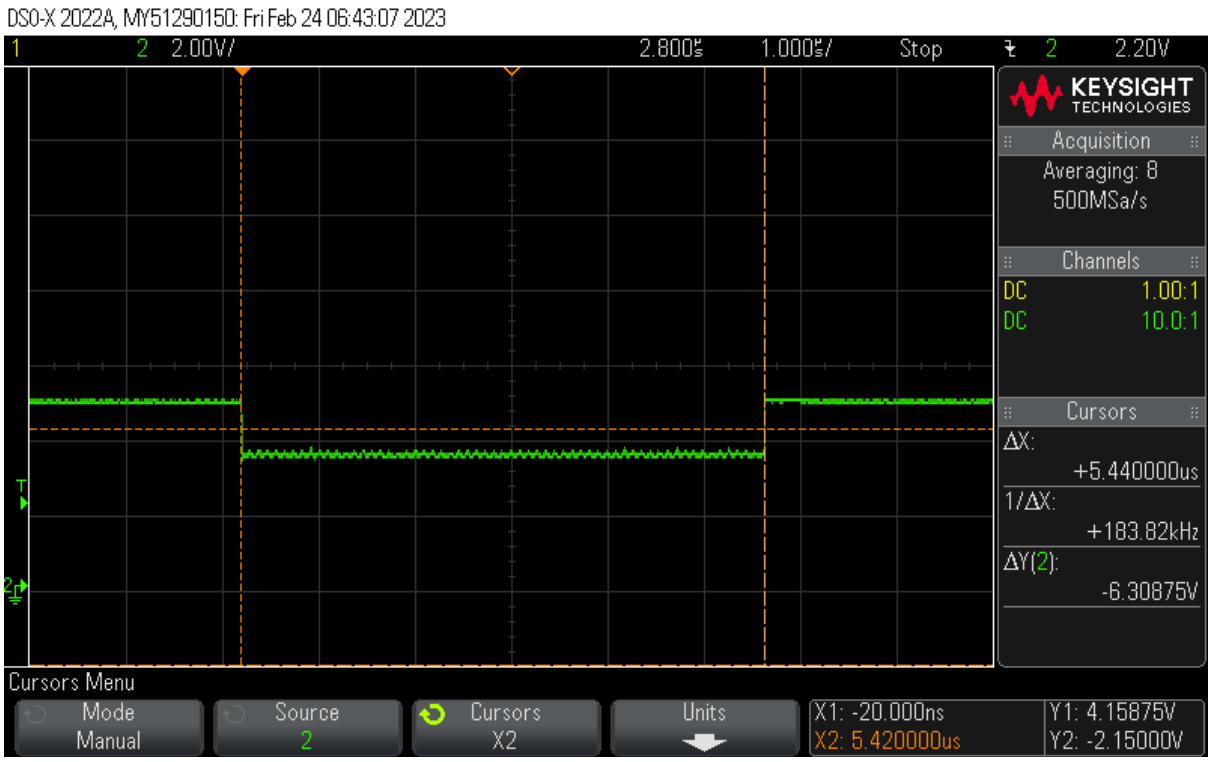
```
    MOV TL0,#0X46                ; 2 cycles
    MOV B,#6                     ; 1 cycle
    CPL P1.0                     ; 1 cycle
    CPL P1.2                     ; 1 cycle
    RETI                         ; 4 cycles
LOWER_FRQ:
    DJNZ R0,LOOP                 ; 2 cycles if not taken, 3 cycles if taken
    MOV TH0,#0X94                ; 2 cycles
    MOV TL0,#0X5B                ; 2 cycles
    MOV R0,#18                   ; 1 cycle
    CPL P1.0                     ; 1 cycle
    CPL P1.2                     ; 1 cycle
    RETI                         ; 4 cycles
```

The longest path through the ISR takes 1 + 2 + 3 + 2 + 2 + 1 + 1 + 4 + 3 + 2 + 1 + 1 + 4 = 25 cycles.

Therefore, the ISR takes between 14 and 25 clock cycles to execute once, depending on the path taken through the code. To convert this to time, we divide the number of clock cycles by the clock frequency:

Shortest time: (14 cycles * 12)/ 11.0592 MHz = 1.264 us
Longest time: (25 cycles * 12) / 11.0592 MHz = 5.261 us

These calculation does matches to most of the requirement when found out on oscilloscope.

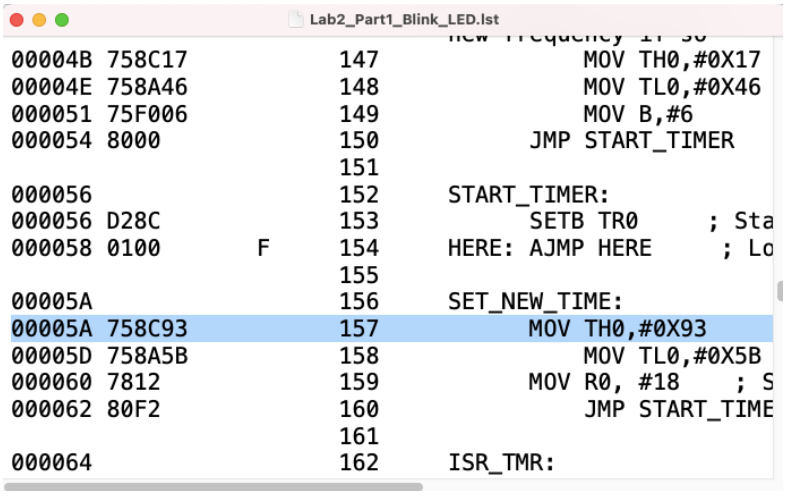


NVRAM Instruction Fetching

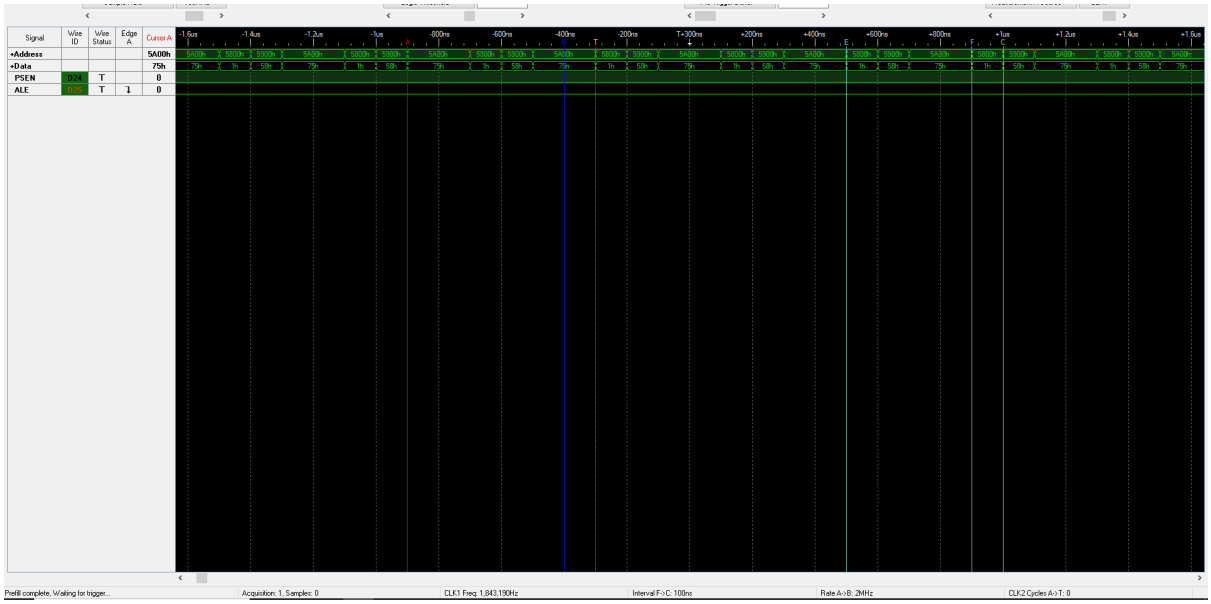
State Mode

After hooking up the logic analyzer I was able to able to decode the data and see the fetched instruction. ALE was primarily used as a state clock and PSEN also to some extent. The result of using the state mode to capture the sequence of instruction and has been verified with the .LST file for the same.

The below LST file shows that Address 5Ah is assigned data 75H followed by



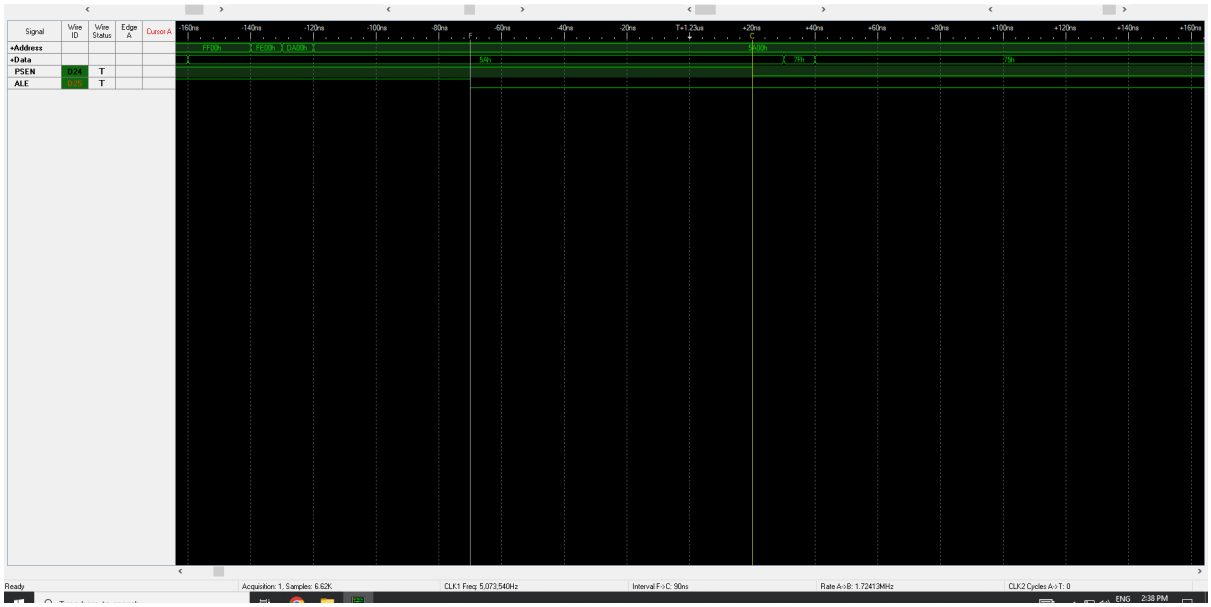
The diagram below shows that for the address of 5Ah the data is 75H which does matches the LST file configuration when seen using logic analyzer.



Timing Mode

t_{LLPL} is the time it takes for the 8051 microcontrollers to latch the address supplied by the 74LS373 latch and activate the PSEN signal during an instruction fetch. To measure t_{LLPL}, we need to use a timing analyzer tool that can capture and display the signals from the 8051 and the 74LS373 latch.

The below screenshot from logic analyzer shows the exact behaviors as seen below :



The measurement of interval between F and C shows t_{LLPL} as 90 ns which is found to be acceptable which is greater than the t_{LLP} minimum acceptable time of 40 ns as per the data sheet of 8051 processor used.

Lab 2 Part 2 :

UART

Atmel AT89C51RC2 processor for Part 2 Elements has been used. For lab 2 first requirement CS501 is replaced with ATMEL processor (Atmel AT89C51RC2). Verified that the oscillator circuit and run-time reset circuit work correctly. ALE signal does comes up at the correct frequency after every power cycle and reset cycle. Verified that the Blinking LED code works with the Atmel processor too.

RS-232 has been integrated into the circuit as seen in board layout picture. This is done using MAX 232 and RS 232 with capacitor dumps as seen in the schematic picture.

Optional Assembly Code :

An assembly code has been implemented to test the UART logic both with internal as well as external RAM modes and code was developed to transmit ‘U’ continuously.

```
; MIT License
;
; Copyright (c) [2022] [Kiran Jojare]
;
; Permission is hereby granted, free of charge, to any person obtaining a copy
; of this software and associated documentation files (the "Software"), to deal
; in the Software without restriction, including without limitation the rights
; to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
; copies of the Software, and to permit persons to whom the Software is
; furnished to do so, subject to the following conditions:
;
; The above copyright notice and this permission notice shall be included in
; all copies or substantial portions of the Software.
;
; THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
; IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
; FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
; AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
; LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
; OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
; THE SOFTWARE.
;
;
; @ brief  : assembly program which initializes the 8051 serial port
;          and then continuously (in an infinite loop) transmits
;          the character 'U'.
; @ author : Kiran Jojare
; @ date   : 23 Feb 2022
; @ referenses :
;         1) Offcial website from Edsim
;            http://www.edsim51.com/examples.html

#include<reg51.h>

CLR SM0             ; Clear the Stack Pointer for this program.

SETB SM1            ; Enable all Interrupts.

MOV A, PCON         ; Move contents of PCON register to accumulator A.
```

```
SETB ACC.7          ; Set the 7th bit of the accumulator to 1.
MOV PCON, A         ; Move the contents of accumulator A back to PCON register.
; This code sets the SMOD bit of PCON, which is used to set the baud rate of the serial port.

MOV TMOD, #20H  ; Initialize the Timer 1 in Mode 2 (auto-reload) for 9600 baud rate
MOV TH1, #-12
MOV SCON, #50H   ; Set the Serial Control Register (SCON) to 8-bit data, 1 stop bit, and enable Reception (REN)

SETB TR1 ; Start Timer 1


; Loop to transmit the character 'U' continuously
again:
MOV 30h, #'U'       ; Move the ASCII code of 'U' to memory address 30h.
MOV 31h, #0         ; Move the value 0 to memory address 31h.
MOV R0, #30h        ; Move the memory address of 'U' to register R0.
MOV A, @R0          ; Move the contents of memory location pointed to by register R0 to accumulator A.
JZ finish   ; Jump to label 'finish' if accumulator A is 0 (i.e. end of string is reached)
MOV SBUF, A         ; Move the contents of accumulator A to Serial Buffer Register (SBUF) to transmit data
INC R0              ; Increment the value in register R0 to point to the next memory location.
JNB TI, $  ; Jump to current address if the TI flag is not set. (TI flag is set when the SBUF is transmitted)
CLR TI              ; Clear the TI flag to indicate that the data in SBUF has been transmitted.
JMP again           ; Jump to label 'again' to continue sending the next character in the string.

finish:
JMP $               ; Jump to the current address to halt the program.
```

Above code tests the UART with 9600 baud, 8 data bits, 1 stop bit, no parity, and no flow control. Same code is available in the in the github as well as canvas. Also note that the code works in double baud rate mode.

Boot-Loader

Implemented the required circuit features for /EA and /PSEN to enable the Atmel UART bootloader to execute when coming out of reset. Used a momentary pushbutton and pull-down resistor for /PSEN, and hold that button when coming out of reset to force.

Bootloader operation: then, after the bootloader has started, release that button to eliminate drive fight issues on the /PSEN line. Used a header/jumper for the /EA input. Note that if you use these hardware conditions to enter the bootloader when you come out of reset, then the Atmel bootloader is entered regardless of the values of BLJB, BSB, and SBV.

Verified that Lab #2 blinking LED code runs correctly from the flash memory (/EA high) on the new processor. Verified that FLIP can communicate with the Microchip/Atmel bootloader. Used FLIP to download the code to internal flash memory on the processor.

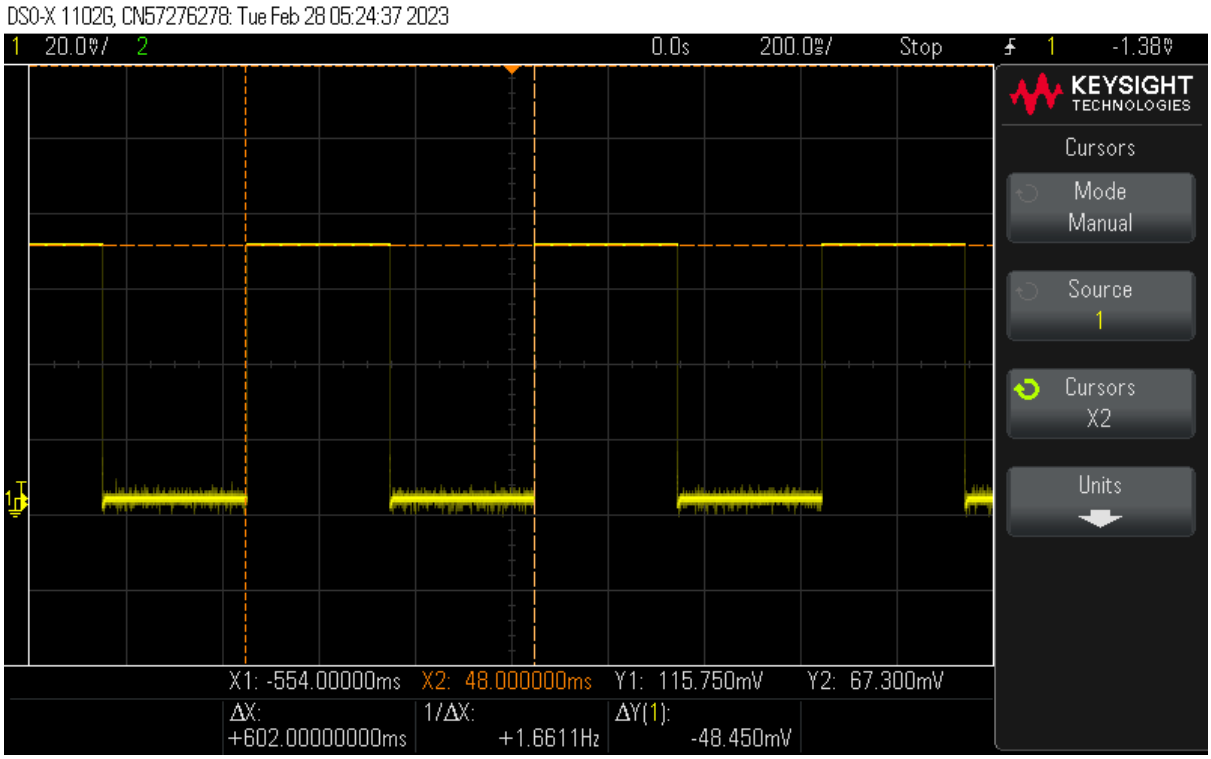
ARM Codes (STM32-F4)

STM32F4 has been used to develop for

1. First program is to toggle a red LED using timer interrupt. Timer 2 has been used and initialized as interrupt when timer overflows. The code toggle LED red every 600 ms (300 ms ON and 300 ms OFF)
2. Second program is to control on board using push button. Initially the LED's red and blue toggle as soon as the button is pressed the LED running at the instant will be ON unless push button is again pressed. Timer 2 is initialized for this code. GPIO pin 12 and 15 has been initialized.

Note : Code is not a bare metal programming. Code is developed using HAL.

The scope output from program one indicating toggling of red LED is shows below where LED is toggling every 300 ms.



Debug Circuit using 74LS374

As per schematic below SPLD ode has been implemented to interact with new latch 74LS374.

```
Name     jojare_esd_lab2 ;
PartNo   00 ;
Date     2/24/2022 ;
Revision 01 ;
```

```
Designer Kiran Jojare ;
Company  University of Colorado ;
Assembly None ;
Location Boulder ;
Device   g16v8a ;

/* *************** INPUT PINS *********************/
PIN 2 = A15 ;         /* INPUT PIN 2: connected to A15 */
PIN 3 = A14 ;         /* INPUT PIN 3: connected to A14 */
PIN 4 = A13 ;         /* INPUT PIN 4: connected to A13 */
PIN 5 = A12 ;         /* INPUT PIN 5: connected to A12 */
PIN 6 = RD ;          /* INPUT PIN 6: connected to RD */
PIN 7 = PSEN ;        /* INPUT PIN 7: connected to PSEN */
PIN 8 = WR ;          /* INPUT PIN 8: connected to WR */

/* ************* OUTPUT PINS *********************/

PIN 16 = CE ;                   /* OUTPUT PIN 16: connected to CE */
PIN 17 = TO_74LS374 ;           /* OUTPUT PIN 17: connected to WE of TO_74LS374 */
PIN 18 = READ ;       /* OUTPUT PIN 18: connected to READ */
PIN 19 = CSPERIPH ;             /* OUTPUT PIN 19: connected to CSPERIPH */

/* ******* Logic Operations *********************/

TO_74LS374 = (!A15 & !WR);              /* WRITE_ENABLE is high when A15 and WR are low */
READ = RD & PSEN;                               /* READ is high when RD and PSEN are both high */
CSPERIPH = !(A15 & A14 & A13 & A12);            /* CSPERIPH is high when A15, A14, A13, and A12 are all low */
CE = A15;                                       /* CE is high when A15 is high */
```

Output TO_74LS374 is going to Cp of the new latch 74LS374.

The logic analyzer screenshot explaining the mapping of correct data to mememory si as follows.

The figure below shows that the memoery is started with 00h, O1h, 02h and keeps on incrementing indicating that system is looping in Main starting from address 00h to 7Fh.

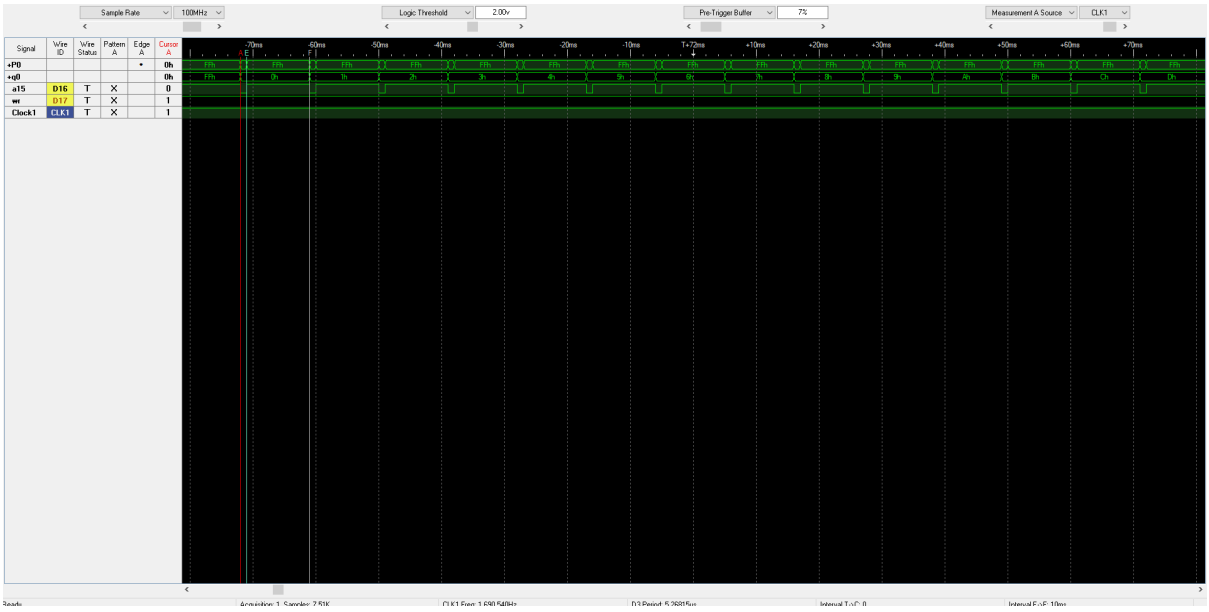
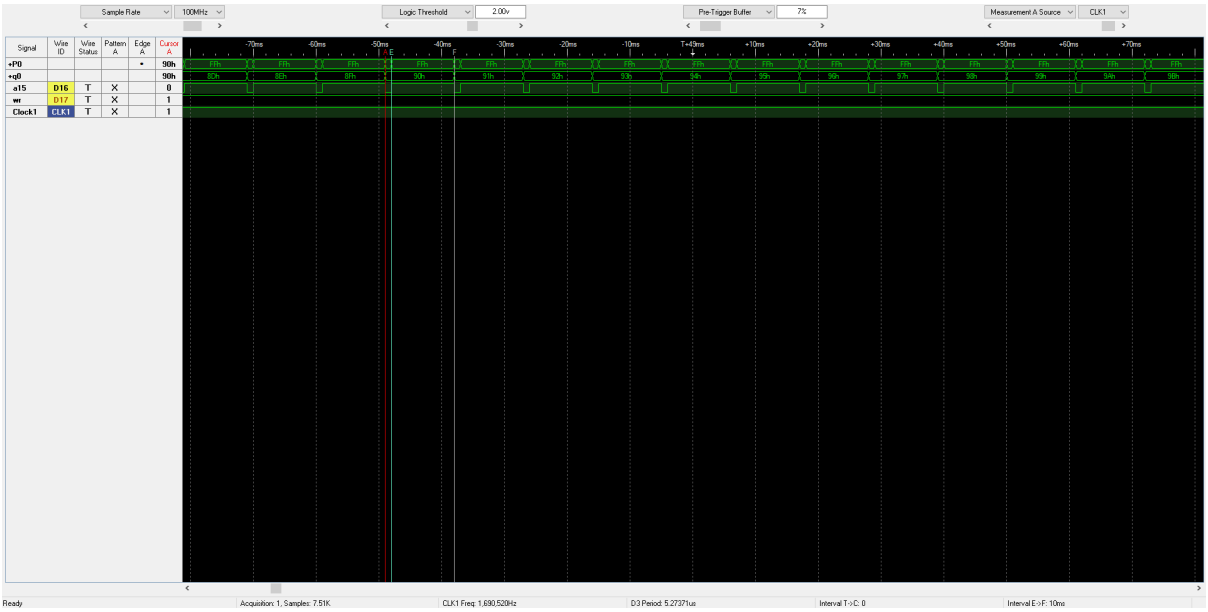
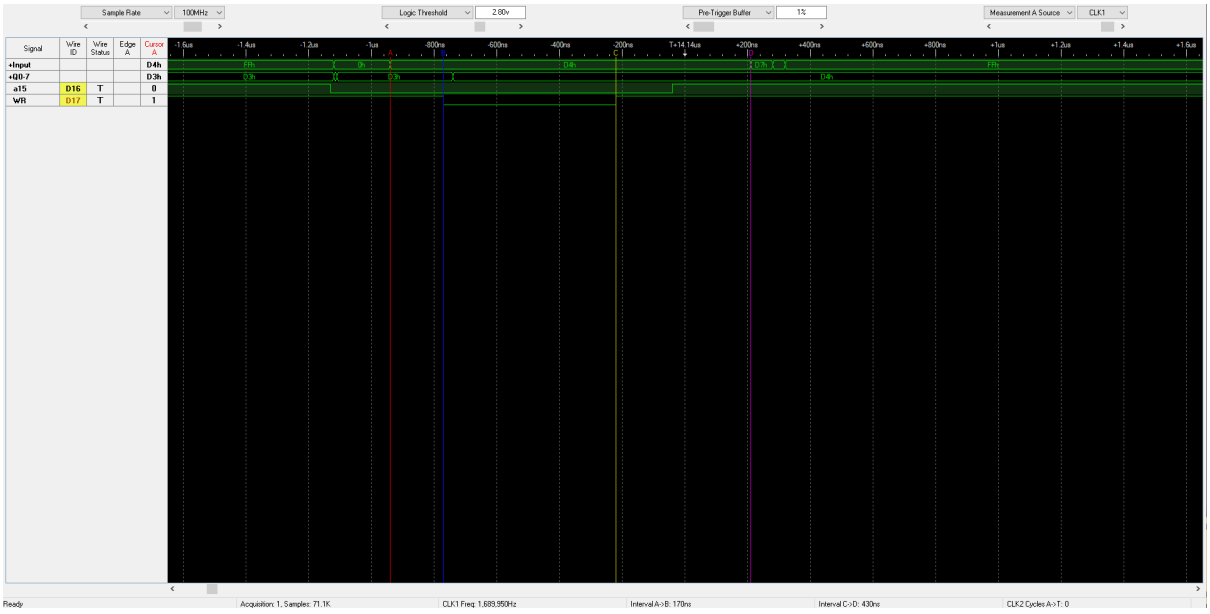


Figure below shows how the ISR starts with address 80h and increments till FFh.



Timing Analysis using 74LS374 Latch

The figure below calculates the setup and hold time observed using logic analyzer.



The interval A->B shows that **setup time observed was exactly 170 ns**
The interval C-> D shows that **hold time is observed to be 430 ns**.

Submission Questions

a) What operating system (including revision) did you use for your 8051-code development?

I used MAC-OS and most of the assembly code development is done and simulated using EdSim and project is crated using Simplicity Studios. All the project and files are available in “asm” and “efm8” folders for reference.

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

A51 micro assembler is used for asm files.

c) What ARM development tools did you use?

All the ARM code has been developed using STMCube IDE using inbuild HAL libraries. Note the ARM code is not a bare metal code programming.

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

Yes, I tried most of the ASM file and simulated the behaviour using Edsim before simulating and building same ASM files using EFM8 board and Simplicity studios.

e) Did you experience any problems or challenges with this lab assignment or any of the software tools? If so, describe the issues.

Yes, my hardware board was behaving strangly as during bootloader mode my PSEN was expected to go from +5V to)volts but it was going from +5 Volts to +2.2 Volts nd not 0 Volts as expected.

Along with TA’s contribution I was able to debug the issue where my MCU’s Vcc was not properly wired wrap. After fixing the wire wrap, PSEN went back to) volts during bootloader mode.

f) If you have any suggestions for changes to this lab assignment for the future, please include those ideas in your submission.

I believed that due to time restrictions, I could not fully contribute to the STM part. Additionally, I thought the addition of STM code generation marked a very abrupt transition from working with hardware to developing software with the STM32F4. I believe I would have contributed more if the STM part had been a separate task for the Lab 2 part 3 elements submission.

Source Code Control

Submitted data is available on git at below link :
https://github.com/kiranj26/Embedded-System-Design-S22/tree/main/Labs/Lab2