Electrical, Computer & Energy Engineering
UNIVERSITY OF COLORADO BOULDER

ECEN 5613 Lab-2 Report

Course Name: Embedded System Design
Student Name: ***RUSHI JAMES MACWAN***

This submission is created by Rushi James Macwan. Credits and courtesy to the TAs (Tristan and Kiran) for their immense help and support.

Lab-1 Part-1

1. The below section explains my code space utilization. The below explanation is based on the "LAB-2 Final Code (Previous Submission)" file associated with my Lab-2 submission in the "asm_code" folder. The newer version of the same code is saved under a different folder name: "LAB-2 Final Code". The only difference between the codes is the placement and calling of the Main and the ISR part. The changes were made while keeping in mind the good benefits of calling the Main and ISR parts that are essentially stored at different memory locations (separated by each other with a lot of gap) so that they do not overlap or intercept with the interrupt vector addresses or in the least, do not overlap with each other. This design was implemented keeping in view the coding styles that were discussed in the class.

   Bytes of code space my program requires in the NVRAM – 35 Bytes

   Explanation:

   Based on the code provided in this lab write-up and the listing file associated with the submission, the code space usage in the NVRAM looks as given below:

NVRAM Address Range	**Purpose for using it**	**Equivalent memory space used in terms of Bytes**
0000H to 0005H	Initial Accumulator initialization and the long jump performed to the MAIN loop	5 Bytes
000BH to 001AH	This code space was utilized for the ISR routine. I acknowledge that this is not the ideal design/usage of the code space since the given code space utilization strictly overlaps with the	15 Bytes

	address location for IE1 (external interrupt 1) interrupt vector address as given in the datasheet (please see the below image) which begins at 0013H.	
0040H to 004FH	This code space was utilized for the MAIN loop.	15 Bytes
Total code space utilization:		**35 Bytes**

Interrupt Sources and their Corresponding Interrupt Vectors

Source (Request Flags)	Vector	Vector Address
IE0	External interrupt 0	0003_H
TF0	Timer 0 interrupt	$000B_H$
IE1	External interrupt 1	0013_H
TF1	Timer 1 interrupt	$001B_H$
RI + TI	Serial port interrupt	0023_H
TF2 + EXF2	Timer 2 interrupt	$002B_H$

Credits: Siemens (C501 Datasheet)

2. The ISR routine execution took the following time durations (theoretically and practically):

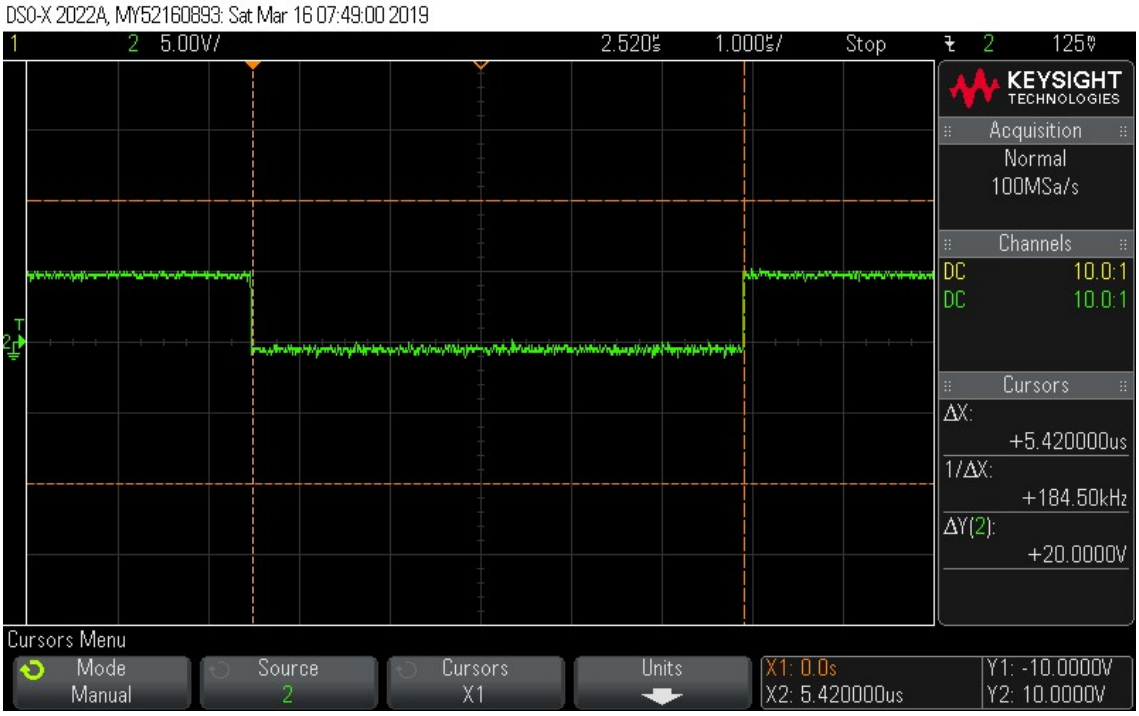
Code vs Oscillator Cycles breakup:

1. ISR: CPL P1.7 Osc Period: 12
2. CLR TF0 Osc Period: 12
3. INC A Osc Period: 12
4. CJNE A, #07H, REPEAT Osc Period: 24
5. MOV A, #00H Osc Period: 12
6. CPL P1.5 Osc Period: 12
7. REPEAT: PL P1.7 Osc Period: 12
8. RETI Osc Period: 24

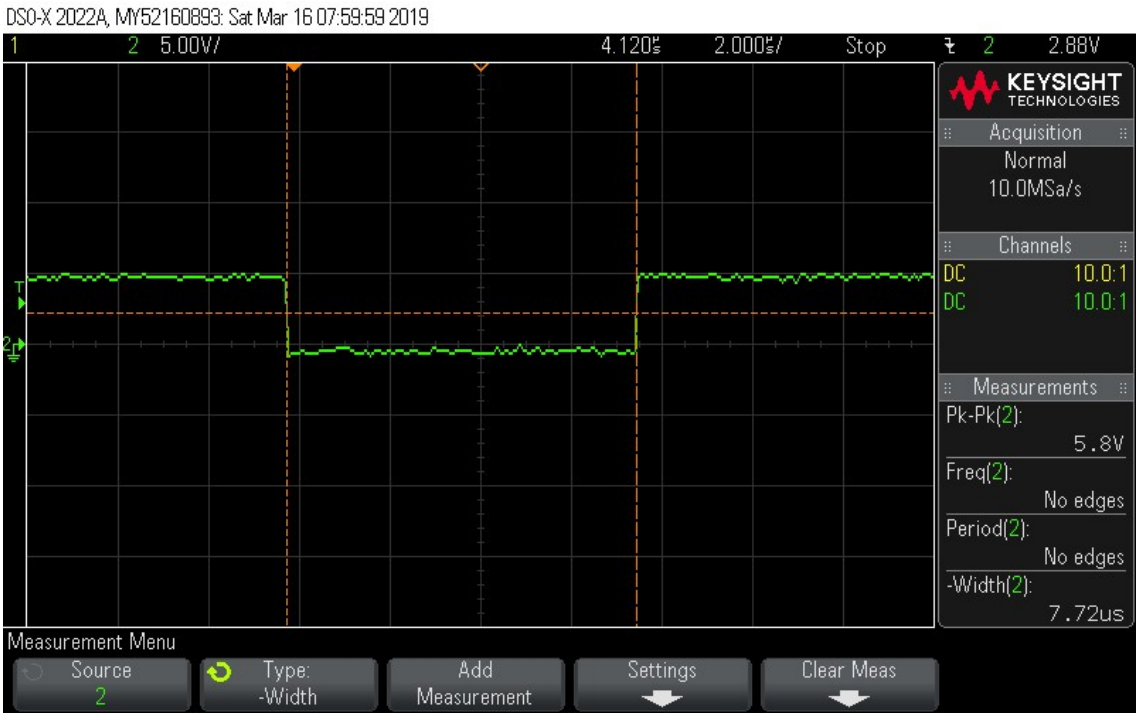
Note: Here, the red part is not practically present in the lab DSO output as the code will toggle the test pin (P1.7) only after the initial "CPL P1.7" instruction is successfully executed. Similarly, the test pin will be toggled before the RETI instruction is executed and so the practical calculation of the ISR timing based on the DSO output will exclude the first and last lines of the ISR code. For more, please refer to the entire assembly code attached with the submission. The submitted code file contains all the appropriate explanation for the code.

ISR Execution Time (Theoretical)	ISR Execution Time (Practical)

For full ISR execution (i.e. when the Timer 0 has overflowed for 7 times in a row which equals to a period of 0.5 seconds and passes the execution through the CJNE instruction), the time consumed will be from line 1 to 8 of the ISR code which sums up to a total oscillator period of 120 cycles. For, one oscillator cycle, the time period will be 0.0904 usec and therefore for 120 cycles, the time duration will be **10.850 usec**.	For full ISR execution, based on the practical DSO output, it took **7.72 usec**. This is because the DSO will only see the code execution of lines 2-7 which theoretically amounts into 84 oscillator cycles which corresponds to 7.595 usec. The practical value differs a little from the theoretical value due to the inaccuracies of the DSO, the inaccuracies in the measurement and the measuring probes.
For incomplete ISR execution (cases where Timer 0 has not overflowed for 7 times and the CJNE jump is performed), the oscillator periods will be equal to that of 96 cycles (code lines 1 to 4 and 7 to 8) which corresponds to **8.680 usec**.	For incomplete ISR execution, the practical DSO output was **5.42 usec**. This is because the DSO will only see the code execution of lines 2-4 and 7 which amounts to period of 60 oscillator cycles which theoretically corresponds to 5.425 usec. Again the slight variance might be due to the above mentioned reasons.



Short ISR timing based on P1.7 toggling



Long ISR timing based on P1.7 toggling

SPLD WinSim Code Comments:

The SPLD code takes the following parameters as inputs and outputs for the chip select logic:

Inputs	Outputs
A15 – the address pin on 8051. As this pin goes low, the NVRAM is activated because of the reason that the first 32kB in the memory map starting from 0000H to 7FFFH should correspond to the NVRAM address space. This directly requires that the A15 pin must be held low in order for the NVRAM to be selected.	**/OE** – The output enable pin of the NVRAM is held low (meaning that the NVRAM can send code data to the 8051) only when the chip is active and /PSEN goes low (meaning only when 8051 is actually reading the code from NVRAM).
/PSEN – the program store enable active-low pin. As this pin goes low, 8051 will be reading code from the internal/external code space (whichever is specified through hardware or software means). In our case, since we are holding the /EA line low, it means that 8051 will read code from the external code space only (in our case NVRAM) as it happens to be	**/CE** – The chip enable signal goes low (which means that NVRAM becomes active for usage) only when all the input signals go low. The reason for this is already mentioned on the left: to consume as less power as possible.

the external code storage device. Also, the NVRAM will be activated only when all of the three signals: A15, /PSEN and /EA go low.	
/EA – Although the /EA line is externally grounded so that it always remains low and has no decision making in the SPLD output, it is kept in order for future labs. While /EA line is not low, the external storage NVRAM will not be turned on to save power while it will be turned on only and only when all of the three input signals are low to ensure that the board consumes as less power as possible.	**CP** – This is the clock pulse signal associated with the debug port latch (74LS374). When the CP signal goes from high to low (as the latch is edge-triggered), the latch outputs are updated based on the inputs read from the AD0-AD7 pins from the 8051. The logic is designed in the SPLD such that CP signal goes from high to low only and only while the 8051 is performing a write operation to an external memory location and while it does, AD0-AD7 is actually D0-D7 because data flow takes place during that stage.
/WR – The /WR signal is taken as an input by the SPLD based on which it generates the CP (clock pulse) signal for the debug port latch (74LS374) so that it can drive the D0-D7 data bus to the debug port output.	

Schematic Diagram:

The schematic diagram has been included in the submission.

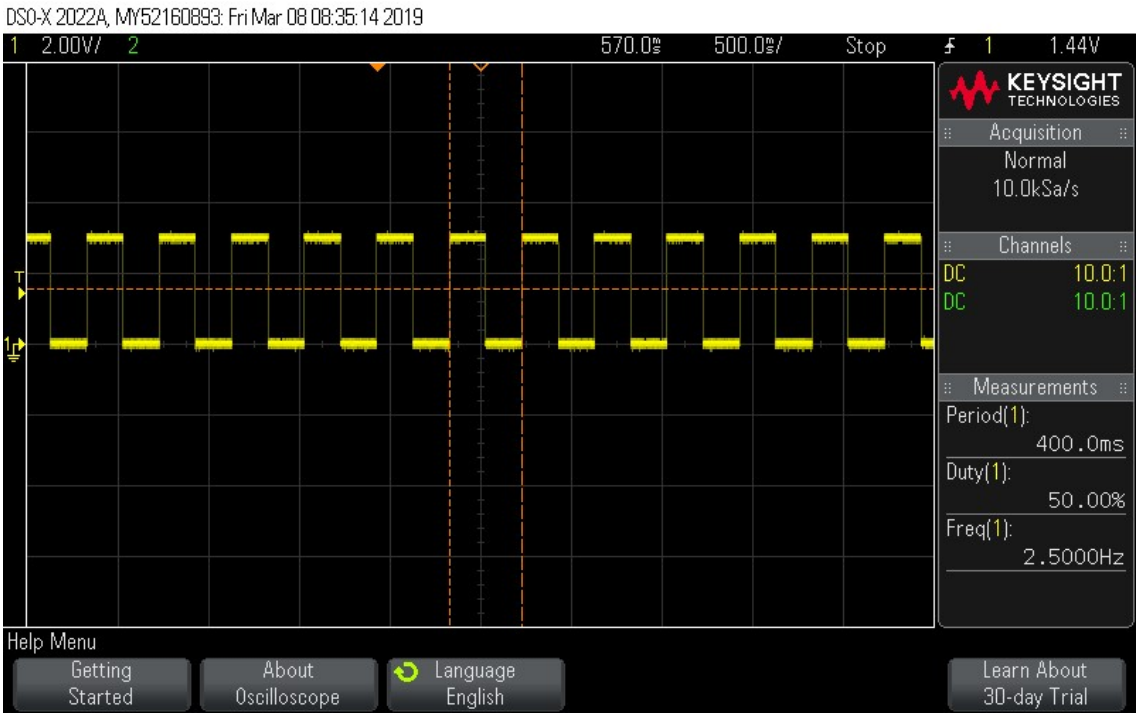
Wiring Description:

Wire Colour	Purpose
RED	5V Power (Vcc)
BLACK	Ground (GND)
BLUE	Control Signals (e.g. ALE)
YELLOW	Address Bus (A0-A7 and A8-A15)
GREEN	Multiplexed Address and Data Bus (AD0-AD7)
ORANGE	Buffered Data (D0-D7)
GREY	Other necessary connections across the board

Lab-1 Part-2

MSP432 Codes:

The MSP432 Codes have been successfully modified based on the example codes. The screenshots folder contains the LED on-off timing of 200 ms as per the requirement (which is included here). The submitted code files include all the appropriate explanation for the code.



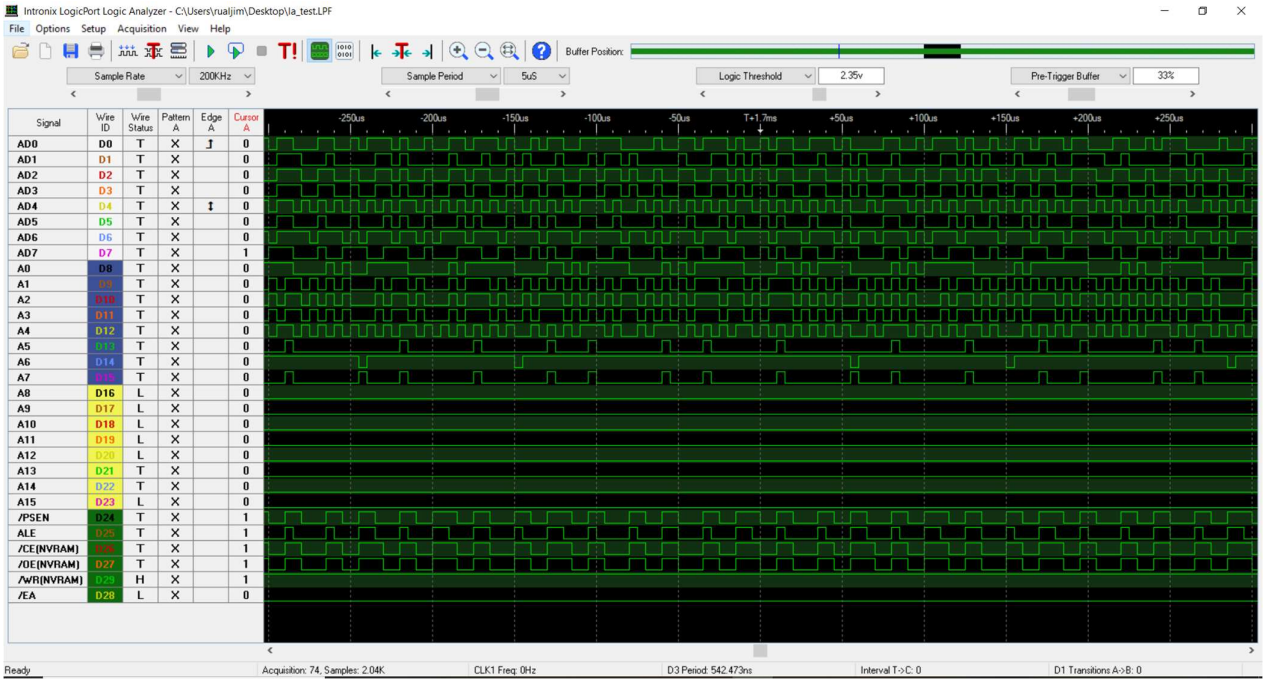
MSP432 LED blinking timing (200 ms on/off timing)

Logic Analyzer Screenshots

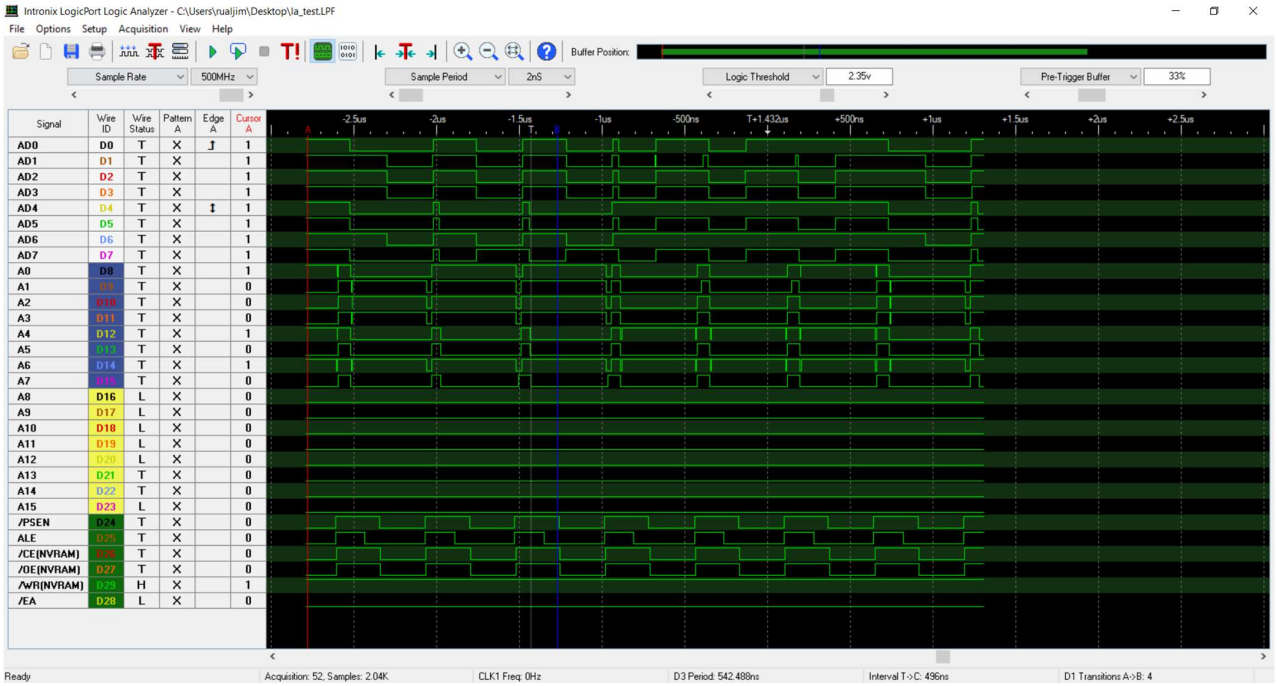
The Logic Analyzer has been a special tool to detect the tiniest of signal variations in terms of digital logic. The Logic Analyzer can take samples as soon as 2nS and that is a decent sampling rate to examine all the setup and hold time qualities and effects for an embedded system design course like this one.

The below screenshots, contain the following signals:

AD0-AD7	Multiplexed Address Bus
A0-A7	Latched Address Bus (lower byte)
A8-A15	Address bus (higher byte)
D0-D7	Buffered Data Bus (available at debug port)
/PSEN	Program store enable control signal from 8051
/EA	External access enable control signal from 8051
/CE (NVRAM)	NVRAM chip select signal
/OE (NVRAM)	NVRAM output enable signal
/WR (NVRAM)	NVRAM write enable signal (connected to Vcc)
ALE	Address latch enable signal from 8051



Logic Analyzer (1)



Logic Analyzer (2) – larger version

As can be found in the 74LS374 edge-triggered latch datasheet, the setup and hold time requirements and definitions are as follow:

AC SETUP REQUIREMENTS (T_A = 25°C, V_{CC} = 5.0 V)

Symbol	Parameter	Limits				Unit
		LS373		LS374		
		Min	Max	Min	Max	
t_W	Clock Pulse Width	15		15		ns
t_s	Setup Time	5.0		20		ns
t_h	Hold Time	20		0		ns

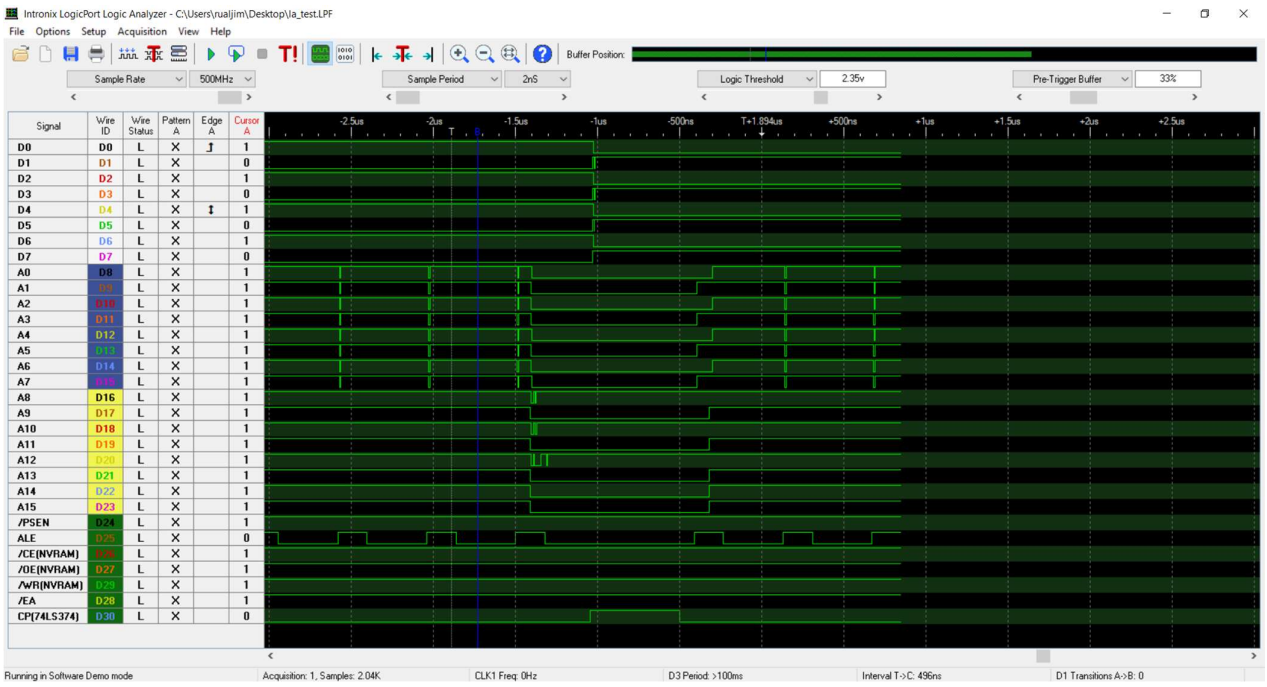
DEFINITION OF TERMS

SETUP TIME (t_s) — is defined as the minimum time required for the correct logic level to be present at the logic input prior to LE transition from HIGH-to-LOW in order to be recognized and transferred to the outputs.

HOLD TIME (t_h) — is defined as the minimum time following the LE transition from HIGH-to-LOW that the logic level must be maintained at the input in order to ensure continued recognition.

As is clear from the above information, the setup time required for the LS374 is at least 20ns. The hold time is 0ns and therefore we will neglect it. What this means is that while the clock signal (CP) goes from high to low, the inputs should have remained stable for at least 20ns prior to the CP transition. As can be seen below, the input signal remains stable for a little over **500ns** before the CP transition which sufficiently meets the LS374 setup time requirement (since it only requires a minimum setup time of 20ns). Also, as required in the supplemental element, the D0-D7 debug port output keeps on toggling between AAH and 55H as I have set these values for debug port testing in the asm code file named "LAB-2 Debug Port Code".

Here, the AD0-AD7 signals are replaced by D0-D7 signals which are read from the debug port.



Logic Analyzer (3)

Conclusion:

As part of this lab, I have pointed out some of the key learnings that I had:

1. Ability to write interrupt service routines for low-level hardware systems
2. Ability to understand the importance of setup and hold timing for components
3. Ability to understand effect of propagation delays when signals are driven through various elements before it reaches a final component where the signal acts as an input signal (e.g. The CP signal in the circuit diagram is actually driven by the 8051 /WR and A15 signals. The propagation delays in the /WR and A15 signals coming out of the 8051 adds to the propagation delay as in the case of the SPLD after which it provides the CP output which is then fed to the 74LS374 debug port. However, this propagation delay comes into picture when we compare the input signals (AD0-AD7) and the CP signal. The AD0-AD7 signals arrive a little before the CP signal arrives at the debug port because of all the propagation delays mentioned above. That brings an error of a few tens of nano-seconds which can be ignored in our lab designs but is critical and very fatal when designing highly sophisticated systems like aircraft control systems.
4. Ability to effectively use the Logic Analyzer.
5. Ability to write fluent register-level access codes in Embedded C for a Cortex M4F 32-bit ARM controller like the MSP432.
6. Ability to work on Eclipse based IDE (i.e. the CCS)
7. Key learnings about interrupts, polling, UART and RS232 communication, timers and especially error elimination while efficiently increasing the baud rate as much as allowed by the embedded system.
8. Flip – on board programmer
9. Debug port design
10. So much more that I can recollect but I have experienced while completing the lab elements.