ECE 501 Digital Systems Laboratory
Experiment 12 – A 4-bit ALU

Seth So
Lab Partner: Martin Klena
Station 2

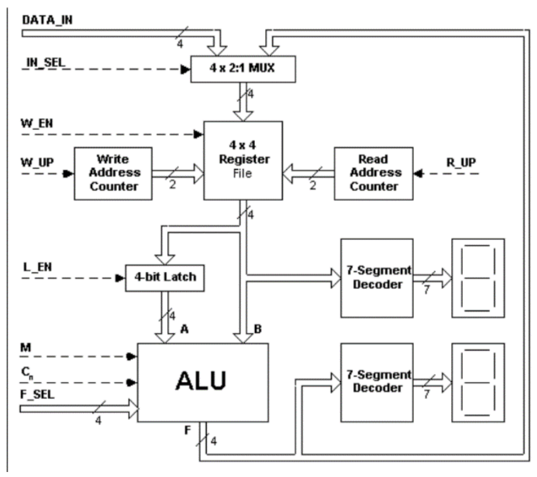
Date Performed: 4-2-18 to 4-10-18
Date Written: 4-11-18

....

Introduction:

The Arithmetic Logic Unit (ALU) is integral to any computer system. It is the unit responsible for processing the mathematical and logic operations necessary to run the system. In this lab, an ALU will be designed incorporating the register file from the previous lab as “on-board” memory. It will be able to perform several of each operation, arithmetic and logical. A block diagram is shown below in in *Figure 1*, outlining a the basic set up and interactions necessary to make the ALU function properly:

Figure 1: 4x4 Register File Skeleton Design



Control Signals	
W_EN:	Write Enable
W_UP:	Write Counter UP
R_UP:	Read Counter UP
L_EN:	Latch Enable
IN_SEL:	MUX Input Select
M:	Mode Control
C_n:	Carry Input

n-Bit Data Signals	
F_SEL:	Function Select
DATA_IN:	Data Input
A:	ALU Input A
B:	ALU Input B
F:	ALU Output

Figure 1 also acts as design specifications: the ALU should be able to take two 4-bit inputs as operands and output a single 4-bit output. Six input lines should control the ALU’s function, with the remaining circuitry supporting the ALU and register file.

After design in Altera Quartus II, the circuit was physically made on the proto-boards, with the LED outputs being 7-segment LED displays. See the *References* section to see how the LED outputs were translated to display form (note that the “old set” of LEDs was used).

Part I: Preliminary Considerations

Purpose:
The purpose of part 1 was to become familiarized ALU circuit on a conceptual level. After grasping the concept, the physical implementation (i.e. the inputs, outputs, and pins) of the 74LS181 ALU chip and surrounding circuitry was considered to develop a robust plan before implementing in either Altera Quartus II or on the protoboard.

Procedure:
1. The first step in designing the ALU was to examine the block diagram for the ALU in *Figure 1* (re-presented below in the results section). To achieve full understanding, each component was described individually, and a mock signal was traced throughout the whole circuit to see the interactions of each component.

2. A 4-bit latch using D flipflops and no gates was designed on paper to serve as the latch shown on the block diagram, shown in *Figure 2* of the results section.

3. The pins (inputs and outputs) of the 74LS181 were examined and checked with the data sheet develop a thorough understanding of how it is implemented. After discussion of the pins and function, a pseudo-code of adding two numbers, utilizing the register to save operands and the result, is shown in *Figure 3*.

Results:

Figure 1: ALU Circuit Block Diagram

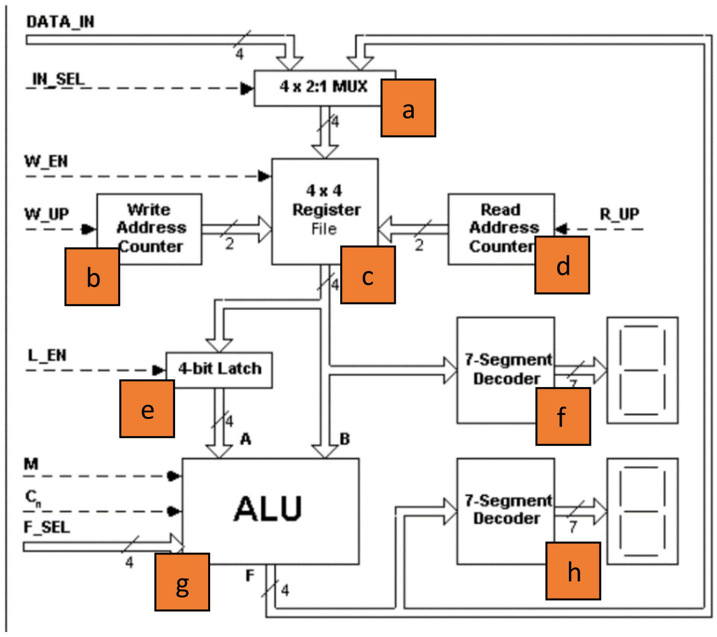


Figure 1 above shows the skeleton design for a fully functioning ALU circuit. Here is a brief description of each part (see corresponding letterings):

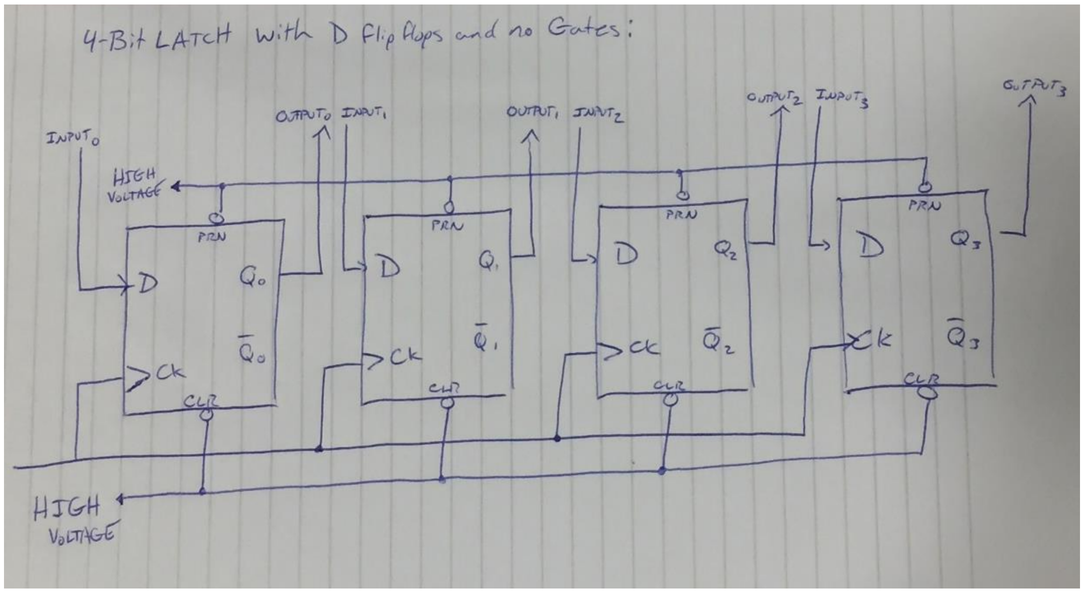
a. *4x2:1 MUX* – based on the select line bit, this multiplexer chooses whether to read out the output of the ALU or manually set data, both of which are 4 bits.
b. *Write Address Counter* – from lab 11, this component acts as the select lines for choosing which address will be written to on the 4x4 Register File.
c. *4x4 Register File* – this component was built in the last lab, along with its peripherals. It can store up to four 4-bit numbers for later reading and outputting.
d. *Read Address Counter* – from lab 11, this component acts as the select lines for choosing which address will be outputted from on the 4x4 Register File.
e. *4-Bit Latch* – this component will serve as a temporary storage to save an operand to for outputting to the ALU.
f. *7-Segment Decoder* – this component will display the current register read value.
g. *ALU* – this chip will perform the logic or arithmetic operation on its operands.
h. *7-Segment Decoder* - this component will display the current ALU output value.

Tracing the circuit from the top, a signal can either come from a manual input or the ALU output. The MUX will select which one is passed on to the register file, which can choose to save that signal for later. The signal, displayed on the first 7-segment display, passes through the register file directly to the input of the 4-Bit Latch and operand B of the ALU. The latch cock may be toggled so that the signal is saved in the latch and constantly outputted to operand A. Once the ALU has signals inputted to operands A and B, it will perform the operation on them specified by its select inputs M, C_n, and F_SEL. The output of the ALU is displayed and piped back to the multiplexer at the top of the diagram, and the cycle is complete.

For the ALU inputs, M stands for Mode and controls whether the ALU is performing arithmetic (0) or logic (1) operations. C_n stands for Carry-In and is necessary for arithmetic functions and unnecessary in logic operations. Therefore, C_n will be set HIGH since it is a DON'T CARE for the logic operation. F_SEL stands for Function_Select and determines which specific arithmetic or logic function is performed. There are eight possibilities for each, so it is controlled by three bits S[3..0].

The complement to the 181, the 182 chip, may be combined with the 181 chip to produce up an up to 16-bit ALU. It has inverted inputs and outputs, so when combined with its compliment, the signal destination does not matter, as one chips logic inversions will always be undone by its complement. See the *References* section for data sheets and tables for the 181 and 182 chips.

Figure 2: Sketch of 4-Bit Latch



Show above in *Figure 2* is the 4-Bit Latch, which has several important features. First, it is essentially four individual D flipflops, with their own inputs and outputs, which serve as Data in and Data out. Next, they all run on the same, so it a synchronous latch. Finally, as tested before with the 7474 D flipflop chips, CLEAR and Preset are active-LOW, and therefore must be set HIGH for the latch to function correctly.

Figure 3: Pseudo Code – Adding Two Numbers

Initialize: M=0, C_n=1, F_SEL=1001, MUX_SEL=0	
1. Input 0010 to Data In	6. Save register output (0010) to Latch
2. Write to address 00	7. Read from address 01
3. Input 0011 to Data IN	8. Change MUX_SEL=1
4. Write to address 01	9. Write ALU output to address 10
5. Read from address 00	END

The table in *Figure 3* above shows the steps necessary perform "2 plus 3 = 5" on the ALU circuit. First the ALU inputs are initialized to the correct settings. Then, 2 (0010 in binary) is written to the first address. Next, 3 (0011 in binary) is written to the second address. After the operands are stored in the register file, the first operand 2 is read from the first address and saved to the latch. After, the second operand 3 is read from and inputted directly into the ALU, which then performs addition as it was initialized to do. Finally, the multiplexer select line is switched so that the register reads the ALU output instead of the data in, and the ALU output is saved to a third address.

Conclusion:
A strong understanding of the ALU design and function was developed. Initial planning was completed, and preparations were complete to begin digital design in Altera Quartus II.

Part II: Quartus Simulation

Purpose:
The purpose of part 2 was to implement the block diagram in Altera Quartus II software on the Graphic Editor so that functionality of the design could be confirmed in the Waveform Editor before actual construction of the circuit.

Procedure:

1. First the 4-bit latch design was tested. After implementation in the graphic editor, it was simulated in the waveform editor to confirm functionality. *Figures 4 and 5* below in the results section show the schematic design and simulated waveform results, respectively.

2. With latch design successfully completed, it was incorporated into the register file designed in lab 11. The selection MUX (74LS157) and ALU (74LS181) chip were also added to complete the design. *Figure 6* in the results section presents the final schematic, along with a comprehensive description of each component and its signals.

3. To test the functionality of the schematic before building on the protoboard, several operations were tested, including: **1**. Four different operations (two arithmetic and two logic) with unique data on one waveform, shown in *Figure 7*. **2**. The addition of four hex numbers, shown in *Figure 8*. **3**. Prepare a full pseudo code for and simulate the multiplication of two values using bit-masking, shown in *Figure 9*.

Results:

Figure 4: Latch Schematic

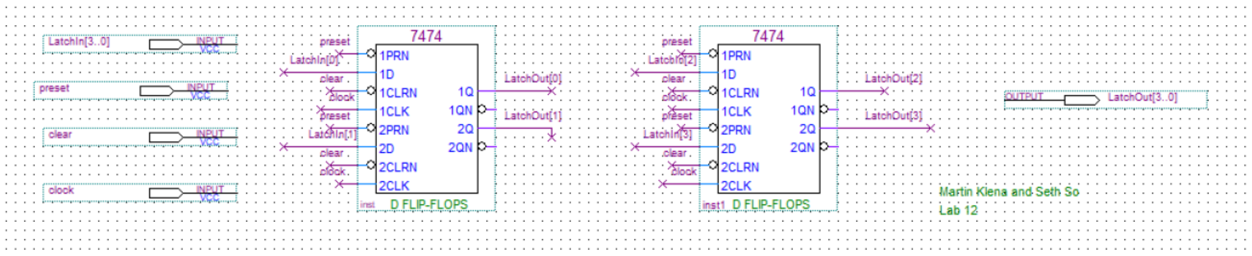


Figure 4 shows the design for the 4-bit latch. It utilizes two 74LS74 chips, which each contain two D flipflops. They all run on the same clock, and CLEAR and PRN inputs are set HIGH as they are active-LOW signals, so that the latch can function correctly. Again, the purpose of the latch is the save and output one of the operands inputted to the ALU.

Figure 5: Latch Test-Waveform

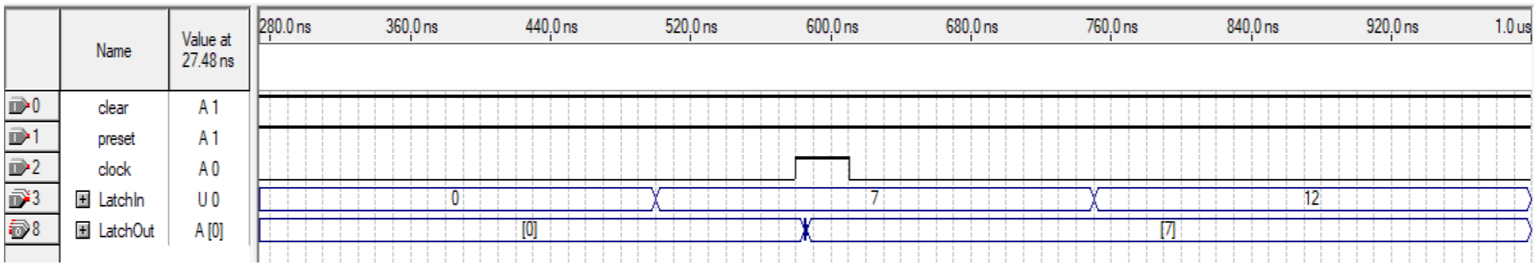


Figure 5 above shows the test-waveform for the latch to confirm functionality. As stated above, CLEAR and PRN (“preset”) are set HIGH to avoid interfering with the function. Initially, the latch is empty, shown by LatchIn and LatchOut both being 0. When LatchIn is set to 7 (inputted as a 4-bit binary signal and displayed here in decimal), the value is not saved until the “clock” is triggered. On the rising edge, the value 7 (inputted as 0111), is saved an then constantly outputted. When a new value, 12, is inputted to the latch, the latch does not save it as this doe not occur on a clock edge. This test confirms proper functionality y of the 4-bit latch.

(Figure 6 on next page)

Figure 6: 4-bit ALU Schematic

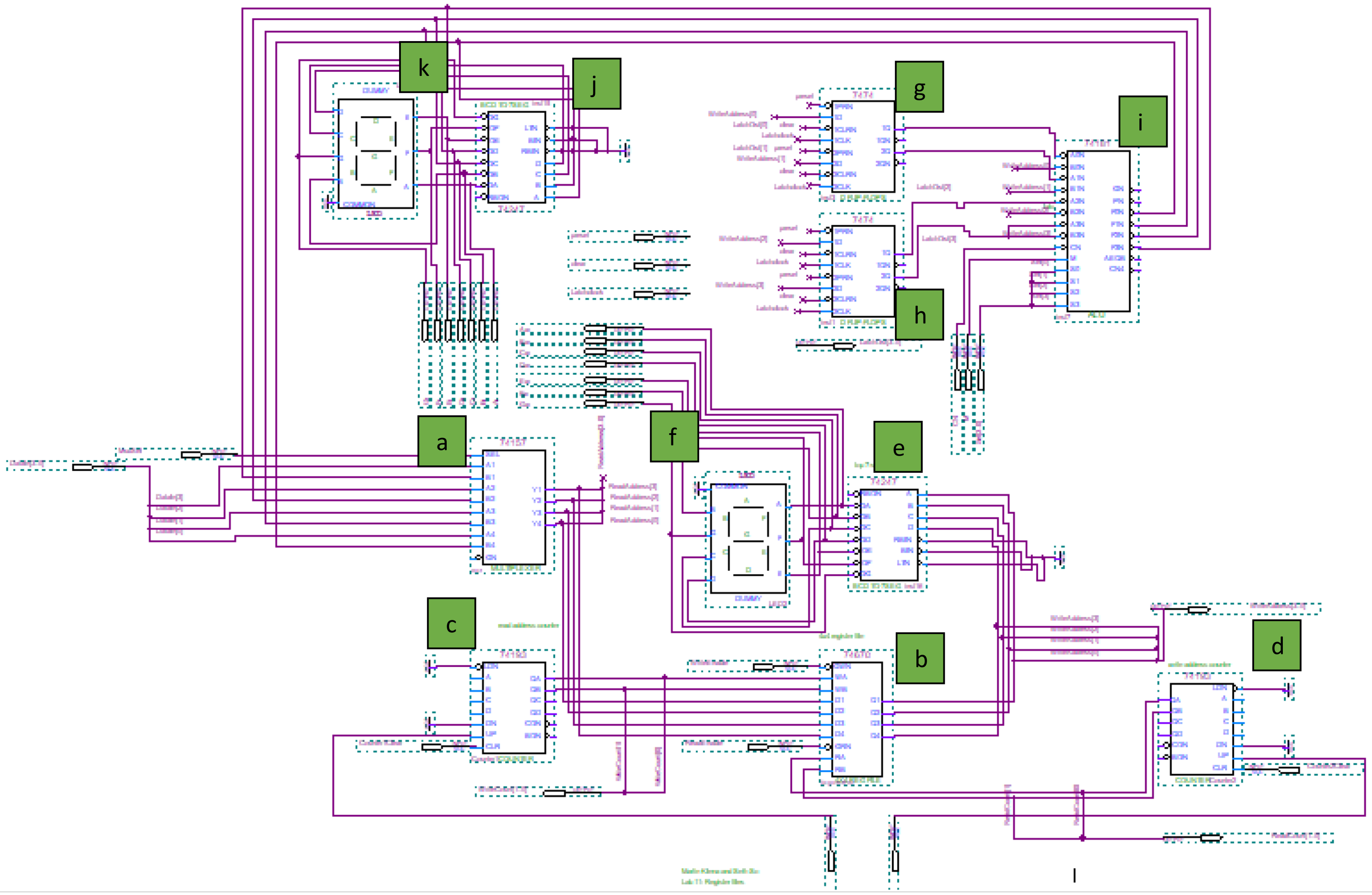


Figure 6 above shows the design for the 4-bit ALU schematic. Below is a list of all parts and data signals. Such initializations such as chip power, ground, unused pins, clear, and reset, and preset are not included. Only data signals are included here. These may be found in the *References* section in each respective components datasheet.

a. *4x2:1 MUX* – based on the select line bit, this multiplexer chooses whether to read out the output of the ALU or manually set data, both of which are 4 bits. (**Inputs**: DataIn[3..0], ALUOutput[3..0], and SelectLine[1..0]. **Outputs**: DataIn[3..0] or ALUOutput[3..0])

b. *4x4 Register File* – this component was built in the last lab, along with its peripherals. It can store up to four 4-bit numbers for later reading and outputting. (**Inputs**: DataIn[3..0] or ALUOutput[3..0], ReadAddress[1..0], and WriteAddress[1..0]. **Outputs**: RegisterOut[3..0])

(continued on next page)

c. *Write Address Counter* – from lab 11, this component acts as the select lines for choosing which address will be written to on the 4x4 Register File. (**Inputs**: none. **Outputs**: WriteAddress[1..0])

d. *Read Address Counter* – from lab 11, this component acts as the select lines for choosing which address will be outputted from on the 4x4 Register File. (**Inputs**: none. **Outputs**: ReadAddress[1..0])

e. *Register LED Decoder* – this decoder takes the data outputted by the 4x4 register file and decodes the information into signals sent to the 7-segment LED display. (**Inputs**: RegisterOut[3..0]. **Outputs**: A_R-G_R.)

f. *Register 7-Segment Display* – this component displays the ReadAdress signal sent to it by the decoder in hexadecimal. (**Inputs**: A_R-G_R. **Outputs**: display)

g. *1st Half of 4-Bit Latch* – this chip saves the two least significant bits of the 4-bit latch, with its two flip flops. When triggered by a switch, it saves the two bits sent to it from the register and outputs those 2 bits to the ALU operand A. It operates on the same clock trigger as the second half (**Inputs**: RegisterOut[1..0]. **Outputs**: LatchOut[1..0])

h. *2nt Half of 4-Bit Latch* – this chip saves the two most significant bits of the 4-bit latch, with its two flip flops. When triggered by a switch, it saves the two bits sent to it from the register and outputs those 2 bits to the ALU operand A. It operates on the same clock trigger as the first half (**Inputs**: RegisterOut[3..2]. **Outputs**: LatchOut[3..2])

i. *ALU* – this chip will take the two operands and perform the logic or arithmetic operation to output. (**Inputs**: M[1..0], C_{in}[1..0], F_SEL[3..0], RegisterOut[3..0], LatchOut[3..0]. **Outputs**: ALUOut[3..0]) Note that pins 13-18 were unused.

j. *ALU LED Decoder* – this decoder takes the data outputted by the ALU and decodes the information into signals sent to the 7-segment LED display. (**Inputs**: ALUOut[3..0]. **Outputs**: A_{W}-G_W.)

k. *ALU 7-Segment Display* – this component displays the signal sent to it by the decoder in hexadecimal. (**Inputs**: A_W-G_W. **Outputs**: display)

Though unlisted, the WriteCounter, ReadCounter, and Latch are triggered and via inverted debounced latch switches (from lab 7) that serve as enable signals. Again, recall that initializations such as power and ground for each component were not listed.

(Figure 7 on next page)

Figure 7: Quartus 4 Different Operations Test

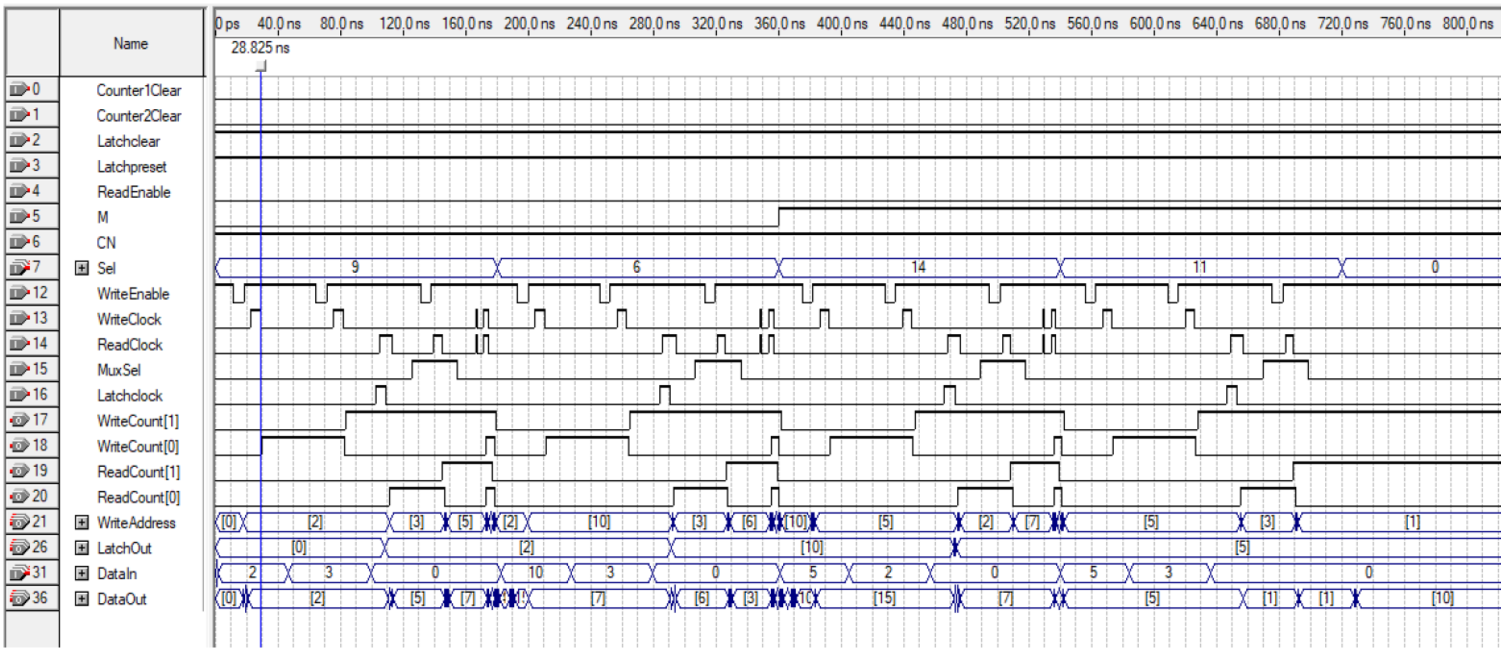


Figure 7 above shows four consecutive, unrelated operations. Counter Clears are set LOW, Latchclear and Latch preset are set HIGH, and ReadEnable is set LOW, as it is active-LOW. M, CN, and Sel are used for ALU initialization, WriteEnable and the Write and Read Clocks allow for address selection, which is displayed in in Write and Read Counts [1..0]. MuxSel is used to select the ALU output to be saved to an address. WriteAddress and LatchOut show what value is being contained and outputted by their respective components. DataIn is the data manually inputted, and DataOut is the ALU output.

The basic procedure and structure of each waveform lasts 180ns and is as follows: **1**. The ALU is initialized for the proper operation. **2**. Operand A is saved to an address. **3**. Operand B is saved to an address. **4**. Operand A is saved to and outputted from the latch to the ALU. **5**. Operand B is read from the register and outputted to the ALU. **6.** The result is saved to a third address. Here are the four operations:

1. [0-180ns, M=0, C=1, Sel=9] ***A PLUS B***: *2 plus 3 = 5*
2. [181-360ns, M=0, C=1, Sel=6] ***A MINUS B MINUS 1***: *10 minus 3 minus 1 = 6*
3. [361-540ns, M=1, C=1, Sel=14] ***A OR B***: *5 OR 2 = 7* (bit-wise OR on the binary values)
4. [541-720ns, M=1, C=1, Sel=11] ***A AND B***: *5 AND 3 = 1* (bit-wise AND on the binary values)

All operations and address saving were successful. Any intermediate values outputted by the ALU to DataOut are a result of the ALU not having a clock and constantly outputting the result of its current operation set up on whatever two operands it has, regardless of whether or not set-up for the next operation is complete. The circuit behaved exactly as expected.

Note that even though the 7-segment display bits are not show here, they do output the correct displays. They will be included for the other two waveforms.

Figure 8: Quartus 4 Consecutive Addition Test

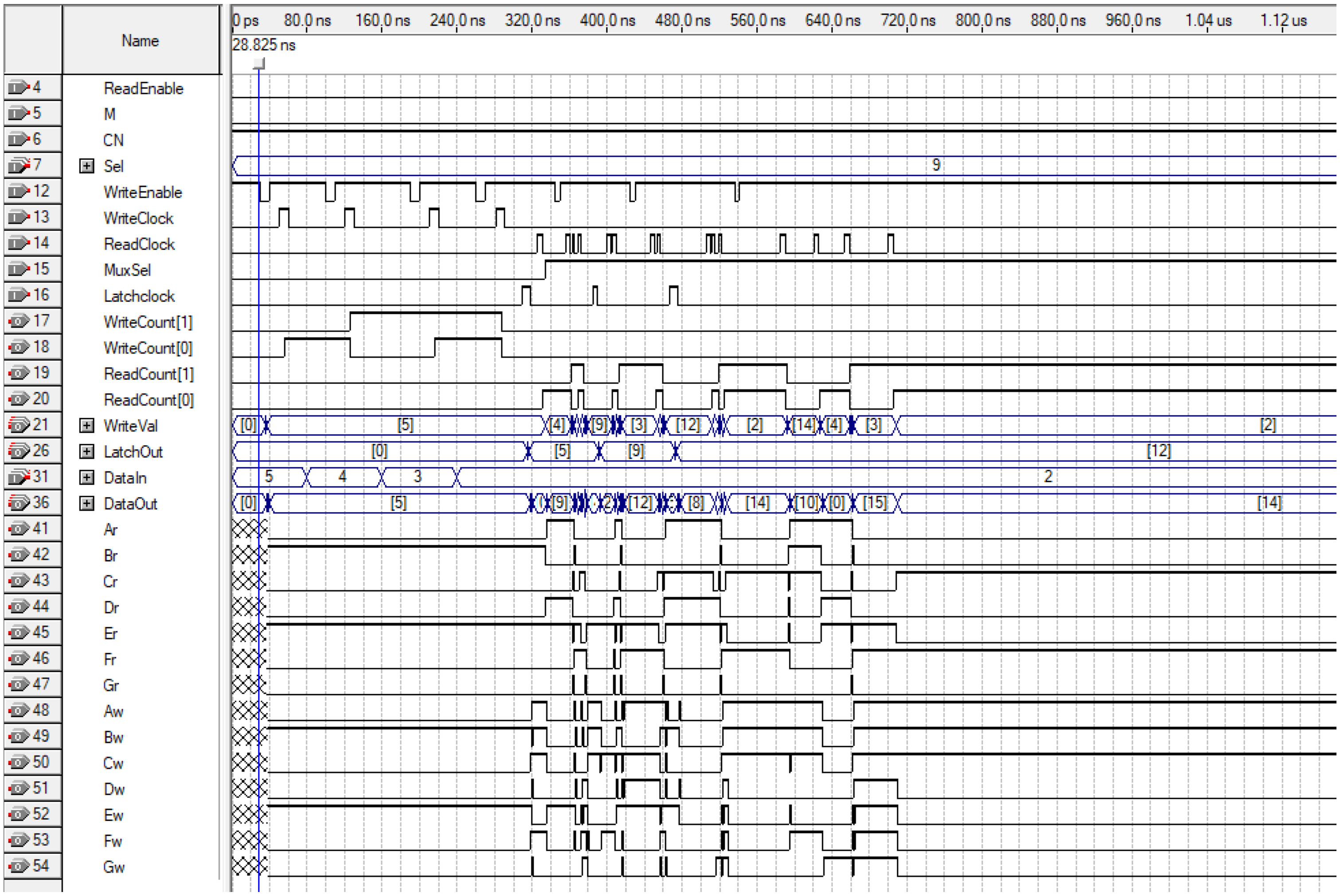


Figure 8 above shows the addition of four hexadecimal numbers: *5 plus 4 plus 3 plus 2 = 14*. All of the signals here were in the previous test and described there. The outputs of the 7-segment displays are also included to confirm that the connections its connections with the ALU output and WriteAddress are correct.

The procedure of the operation is as follows: **1**. Initialize ALU to proper operation. **2**. Save operand 1 to first address. **3.** Save operand 2 to second address. **4.** Save operand 3 to third address. **5**. Save operand 4 to fourth address. **6.** Read first address and save operand 1 to latch to output to ALU input A. **7**. Read second address to output operand 2 to ALU input B. **8**. Switch MuxSelect to pass ALU output to register. **9**. Save ALU output Sum$_1$ to first address. **10**. Read first address and store Sum$_1$ to latch and output to ALU input A. **11**. Read third address to output operand 3 to ALU input B. **12**. Save ALU output Sum$_2$ to first address. **13**. Read first address and store Sum$_2$ to latch and output to ALU input A. **14**. Read fourth address to output operand 3 to ALU input B. 15. Save ALU output Sum$_3$ to first address.

The A PLUS B settings were used for the ALU to test the sum. All outputs and values are correct and expected, including the 7-segment displays Note that any sum over 15 would have overflowed and displayed that overflowed value.

Figure 9: Quartus Multiplication Test

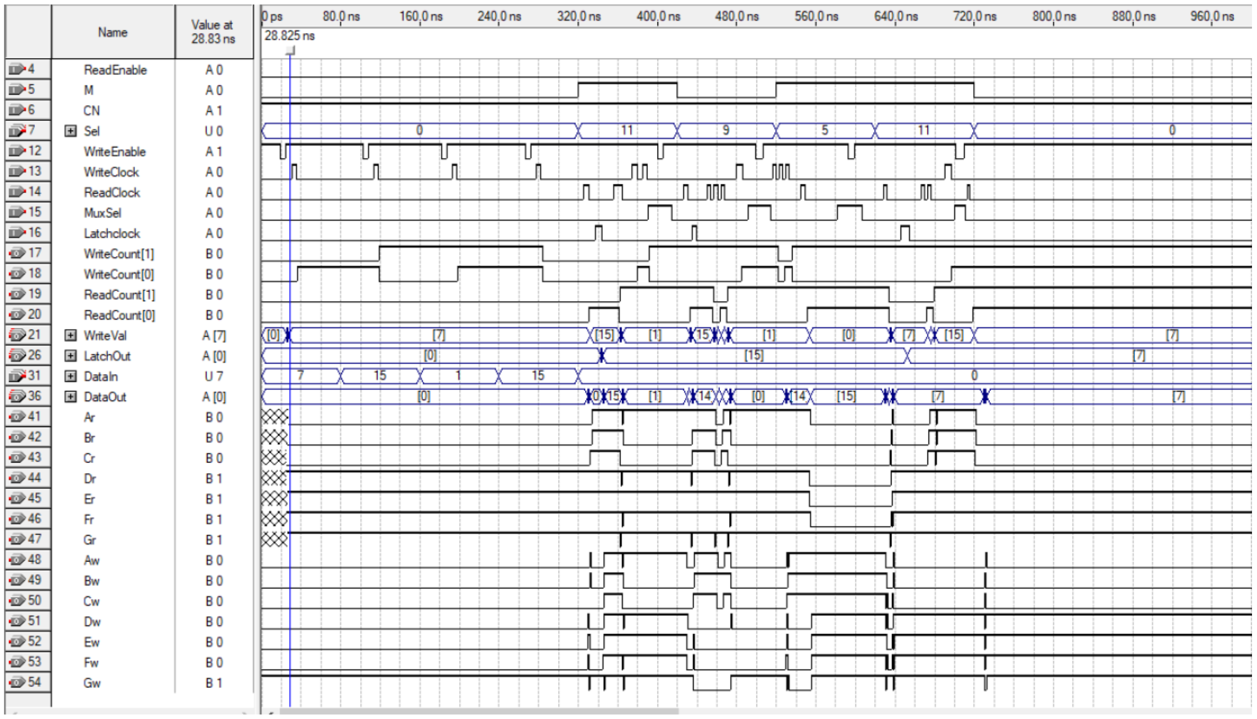


Figure 9 above shows the first partial sum of multiplication of two numbers: *7 multiplied by 15 = 7*. The signals in this waveform are the same as the signals in the previous test.

The following is pseudo code for calculating the first partial sum B_0 in a 4-bit x 4-bit multiplication. The sequence may be repeated (n-1)bits times until completion:

1. Save A to first address. **2**. Save B to second address. **3**. Save "0001" to third address. **4**. Save second address value to latch. **5**. AND first address value and latch value. **7.** Save ALU output to latch. **7**. AND first address value and latch value, and save ALU output to fourth address. **8**. Shift third address value to second address. **9**. Save second address value to latch. **10**. AND first address value and latch value, and save to third address. **11**. Save fourth address value to to latch. **12**. ADD third address value and latch value, and save to fourth address. **13**. Shift second address value to third address. **14**. Save third address value to latch. **15**. AND first address value and latch value, and save ALU output to second address. **16**. Save fourth address value to latch. **17**. ADD second address value and latch value, and save ALU output t fourth address. **18**. Shift third address value to second address. **19**. Save second address value to latch. **20**. AND first address value and latch value, and save ALU output to third address. **21**. Save fourth address value to latch. **22**. ADD third address value and latch value, and save ALU output to fourth address.

Again, this process must be repeated 3 more times to get the full sum. This is such a lengthy process manually, as the multiplication must be carried at as a series of logic and arithmetic operations and shifts, all while working within the space of four register spaces. For ease of observation, the simulation above simply calculates the first partial sum of two simpler numbers: *7 x 15 = 7*. The circuit behave exactly as expected and outputted all correct values and displays.

Conclusion:
The Altera Quartus II design of the 4-bit ALU was a success. It was able to generate the correct outputs for each of the three given functionality tests, which were very rigorous. Through this process, a very firm understanding of the ALU operation and design was developed. There was now enough confidence in the design to start physical implementation on the protoboard.

Part III: Proto-Board Construction

Purpose:

The purpose of part 3 was to realize the digital schematic made in Quartus II on the proto-boards. The design would again be tested twice for functionality – both manually and with the Logic Analyzer. The same tests would be used as were used in the Quartus simulations to confirm results.

Procedure:

1. The schematic in *Figure 6* was constructed on the protoboards with the addition of several extra components. Rather than have clock signals, inverted debounced buttons from laboratory 7 (see *References*) were constructed to control the counters, as well Read and Write Enable. Additionally, eleven DIP switches with 1kΩ pull-up resistors were used to control the DataIn inputs Q[3..0], Latch Enable, Mode, Function Select S[3..0] for the ALU, and Multiplexer Select. Two LEDs with 150Ω current limiting resistors were added onto each of the Counter outputs (address select lines) so that the Read and Write addresses could be visibly kept track of. Finally, and additionally 7-segment display + decoder was added to view what was on Write Address for ease of use. These were wired appropriately. See *Figures 10 and 11* below in the results section below for a picture of the ALU circuit and a labelling key, respectively.

2. The completed circuit board was run through the functionality tests simulated in Quartus. It passed the 4-operations test, but failed the 4-addition test, revealing a major bug. After debugging for an hour, it was discovered that the output of the ALU was connected to the output of the register, rather than the register's input. This caused a continuous summing issue when trying to use the ALU's output as an operand. Once this was fixed, the circuit was able to pass the remainder of the tests. It was then successfully demonstrated to Dr. Jones, who signed off on its functionality.

3. Once the manual test of the circuit was complete, all relevant circuit outputs and inputs were connected to the logic analyzer. Using the logic analyzer for more accurate and reliable measurements, the circuit was once again run through the functionality tests, with images of the waveforms taken.

   Because there were 31 channels being used, the Logic analyzer filled up with data too quickly from live feed rates. To circumvent this problem, the sample rate was dropped from 200MHz to 1kHz to allow for longer sampling time. Appropriate post-fill times were chosen to ensure completion of the given test.

Another quick fix to make the logic analyzer output the correct values was adjusting the logic threshold to 1.25V down from 2. For this design, this allowed the logic analyzer to distinguish between HIGHs and LOWs much more accurately. *Figures 12, 13, and 14* below show the waveform results of each test.

Results:

Figure 10: Proto-Board Realization

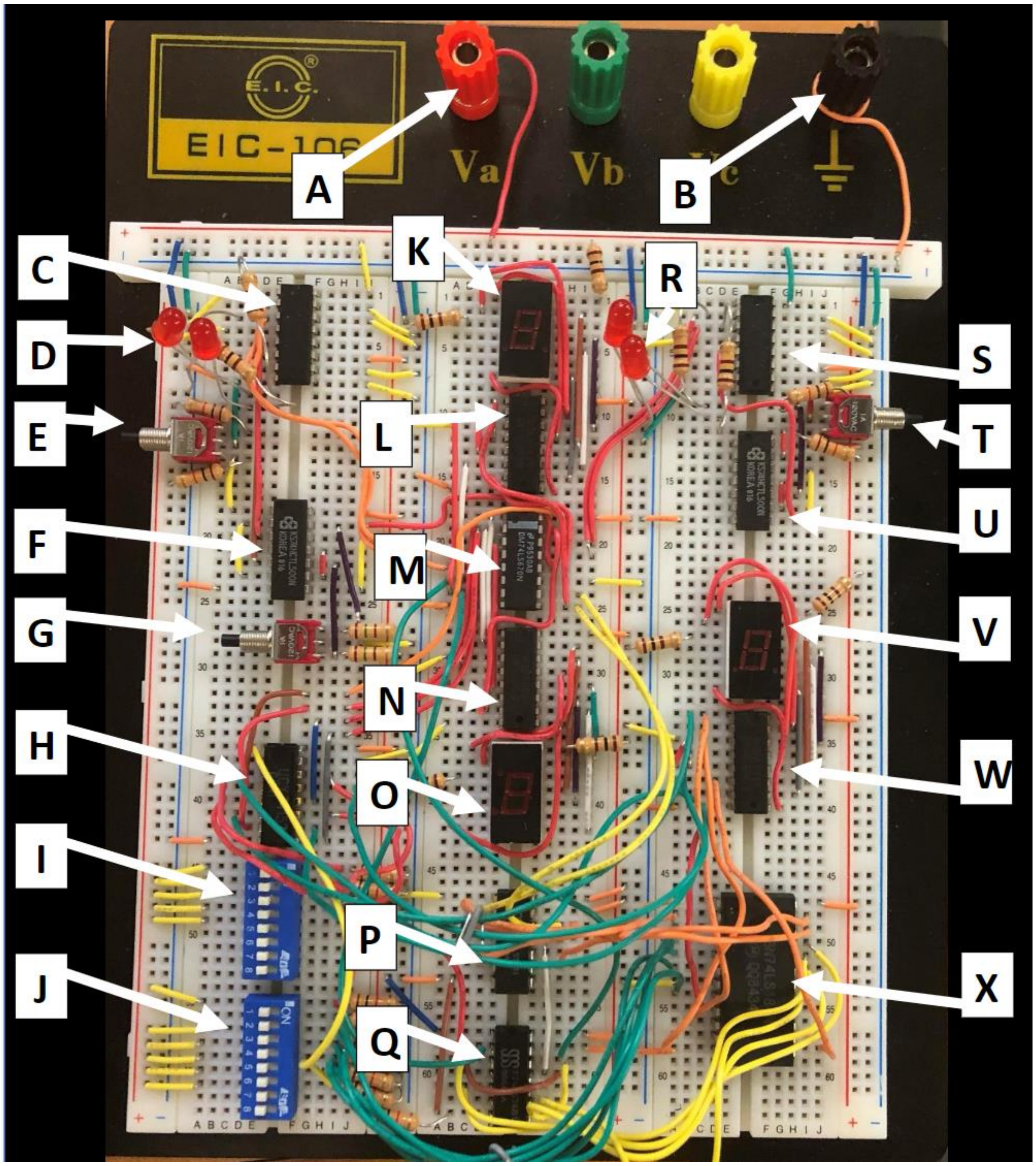
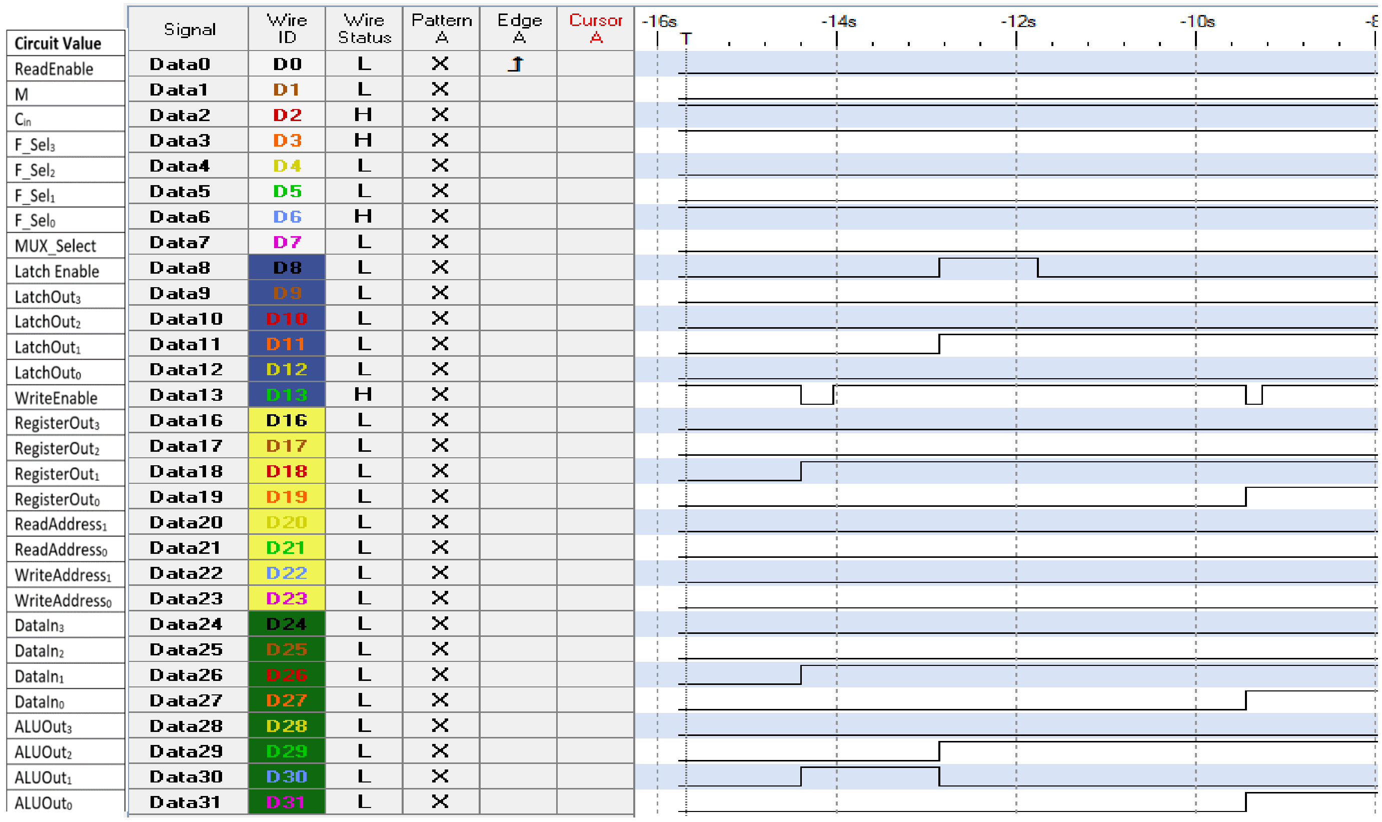


Figure 11: Proto-Board Realization Label Key

Label	**Component**	**Function**
A	Power Input	Deliver V_{DD}=5 volts to the circuit
B	Ground Input	Ground the circuit
C	74193 Chip	A two-bit counter used to control read address
D	LEDs	Visual representation of two-bit read address (top LED is most significant bit; bottom LED is least significant bit)
E	SPDT Switch	Controls logic source used as read counter clock (toggles read address)
F	7400 Chip	Contains NAND gates used for logic source in E
G	SPDT Switch	Controls WriteEnable input to register file
H	74157 Chip	Four-bit 2:1 multiplexer to change register file input between DataIn and ALU Output
I	DIP Switch	Input Control containing the following values: 1. $DataIn_0$ 2. $DataIn_1$ 3. $DataIn_2$ 4. $DataIn_3$
J	DIP Switch	Input Control containing the following values: 1. $LatchEnable$ 2. M (Mode select for ALU) 3. $Select_3$ (for ALU) 4. $Select_2$ (for ALU) 5. $Select_1$ (for ALU) 6. $Select_0$ (for ALU) 7. $MultiplexerSelect$
K	7-Segment LED Display	Display value stored at current read address
L	74247 Chip	Decode four-bit read address value for display in K
M	74670 Chip	4x4 Register File at the center of the lab
N	74247 Chip	Decode four-bit register file input value for display in O
O	7-Segnment LED Display	Display value being inputted to register file
P	7474 Chip	D flip flops used for the two least significant Latch bits
Q	7474 Chip	D flip flops used for the two most significant Latch bits
R	LEDs	Visual representation of two-bit read address (top LED is most significant bit; bottom LED is least significant bit)
S	74193 Chip	A two-bit counter used to control write address
T	SPDT Switch	Controls logic source used as read counter clock (toggles read address)
U	7400 Chip	Contains NAND gates used for logic source in E
V	7-Segnment LED Display	Display value being outputted from ALU
W	74247 Chip	Decode four-bit ALU output value for display in V
X	74181 Chip	Arithmetic Logic Unit (ALU) used for lab operations

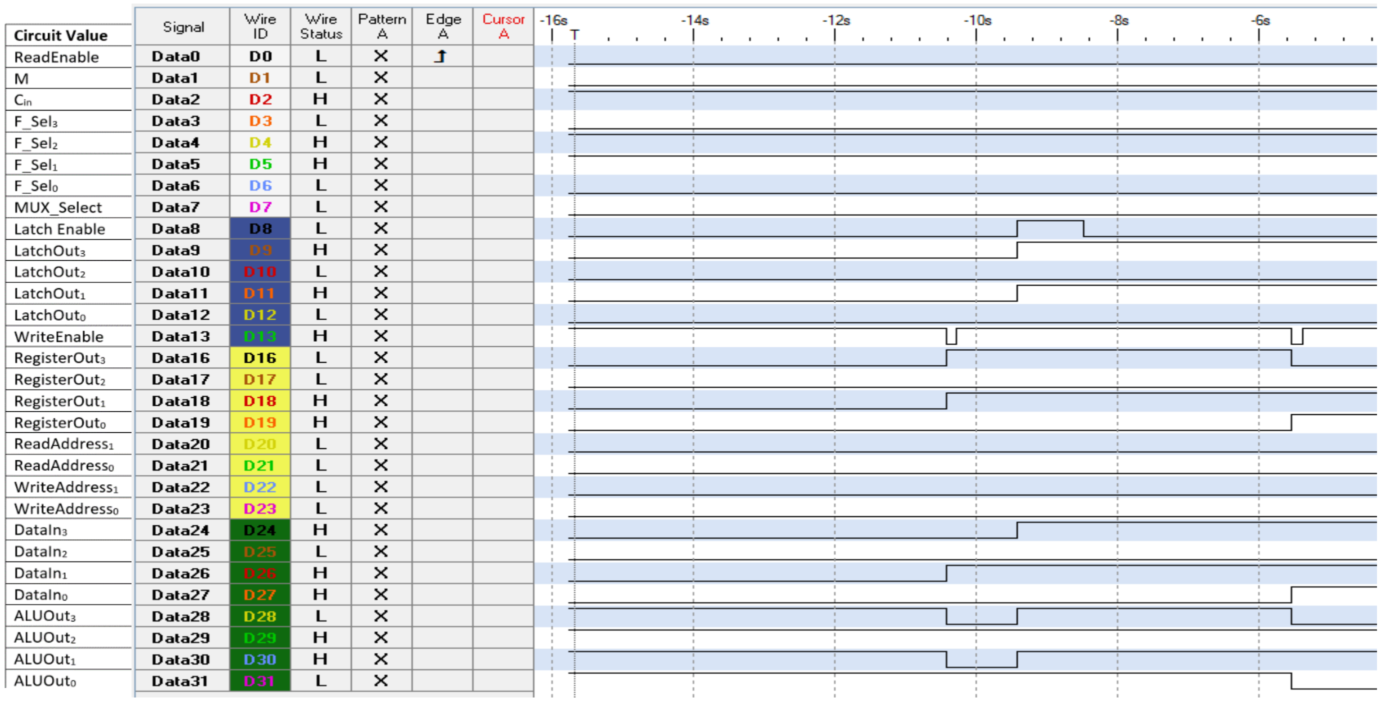
Figure 12: Logic Analyzer 4 Different Operations Test

A. <u>A PLUS B</u>: *2 plus 3 = 5*



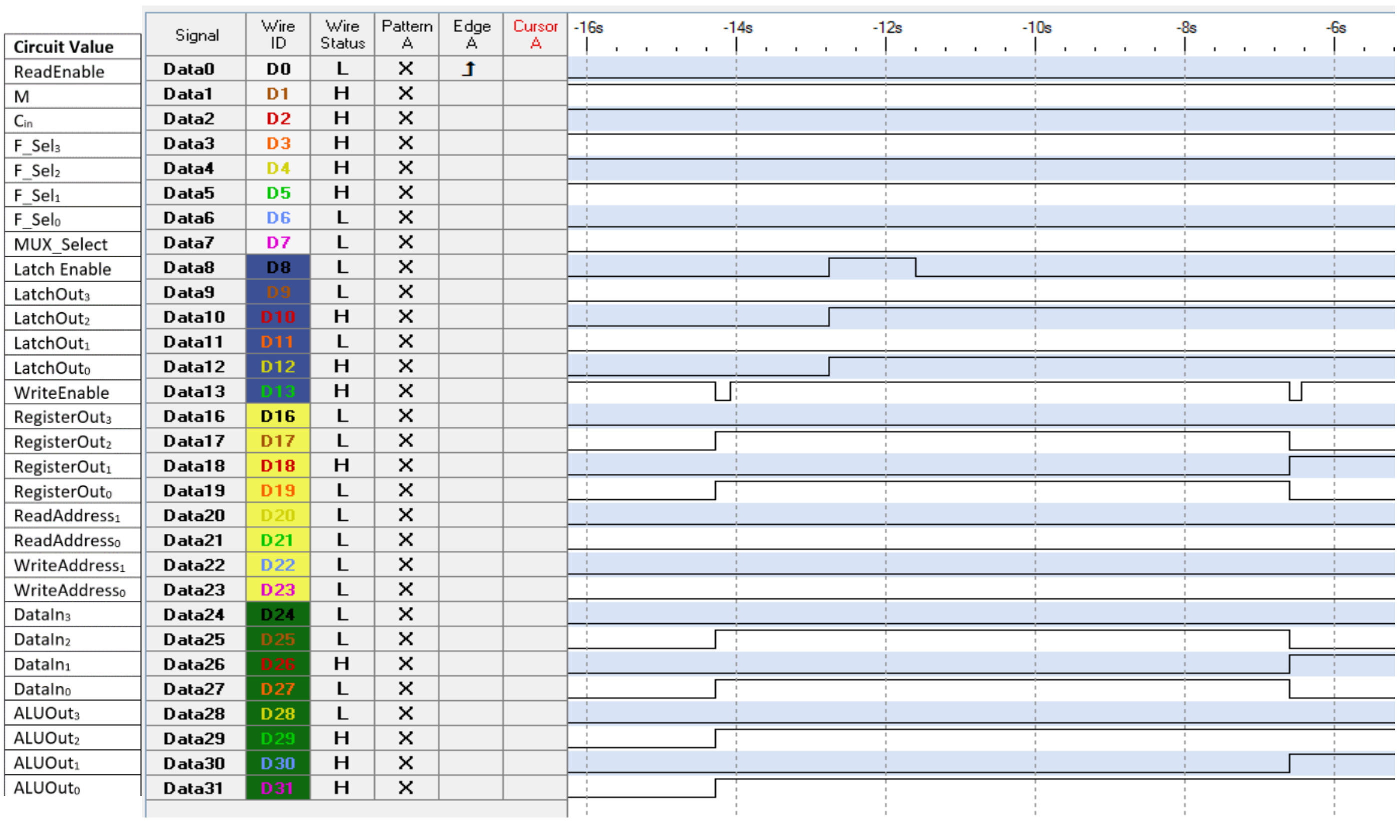
Circuit Value	Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
ReadEnable	Data0	D0	L	X	↥	
M	Data1	D1	L	X		
C_{in}	Data2	D2	H	X		
F_Sel$_3$	Data3	D3	H	X		
F_Sel$_2$	Data4	D4	L	X		
F_Sel$_1$	Data5	D5	L	X		
F_Sel$_0$	Data6	D6	H	X		
MUX_Select	Data7	D7	L	X		
Latch Enable	Data8	D8	L	X		
LatchOut$_3$	Data9	D9	L	X		
LatchOut$_2$	Data10	D10	L	X		
LatchOut$_1$	Data11	D11	L	X		
LatchOut$_0$	Data12	D12	L	X		
WriteEnable	Data13	D13	H	X		
RegisterOut$_3$	Data16	D16	L	X		
RegisterOut$_2$	Data17	D17	L	X		
RegisterOut$_1$	Data18	D18	L	X		
RegisterOut$_0$	Data19	D19	L	X		
ReadAddress$_1$	Data20	D20	L	X		
ReadAddress$_0$	Data21	D21	L	X		
WriteAddress$_1$	Data22	D22	L	X		
WriteAddress$_0$	Data23	D23	L	X		
DataIn$_3$	Data24	D24	L	X		
DataIn$_2$	Data25	D25	L	X		
DataIn$_1$	Data26	D26	L	X		
DataIn$_0$	Data27	D27	L	X		
ALUOut$_3$	Data28	D28	L	X		
ALUOut$_2$	Data29	D29	L	X		
ALUOut$_1$	Data30	D30	L	X		
ALUOut$_0$	Data31	D31	L	X		

B. <u>A MINUS B MINUS 1</u>: *10 – 3 – 1 = 6*



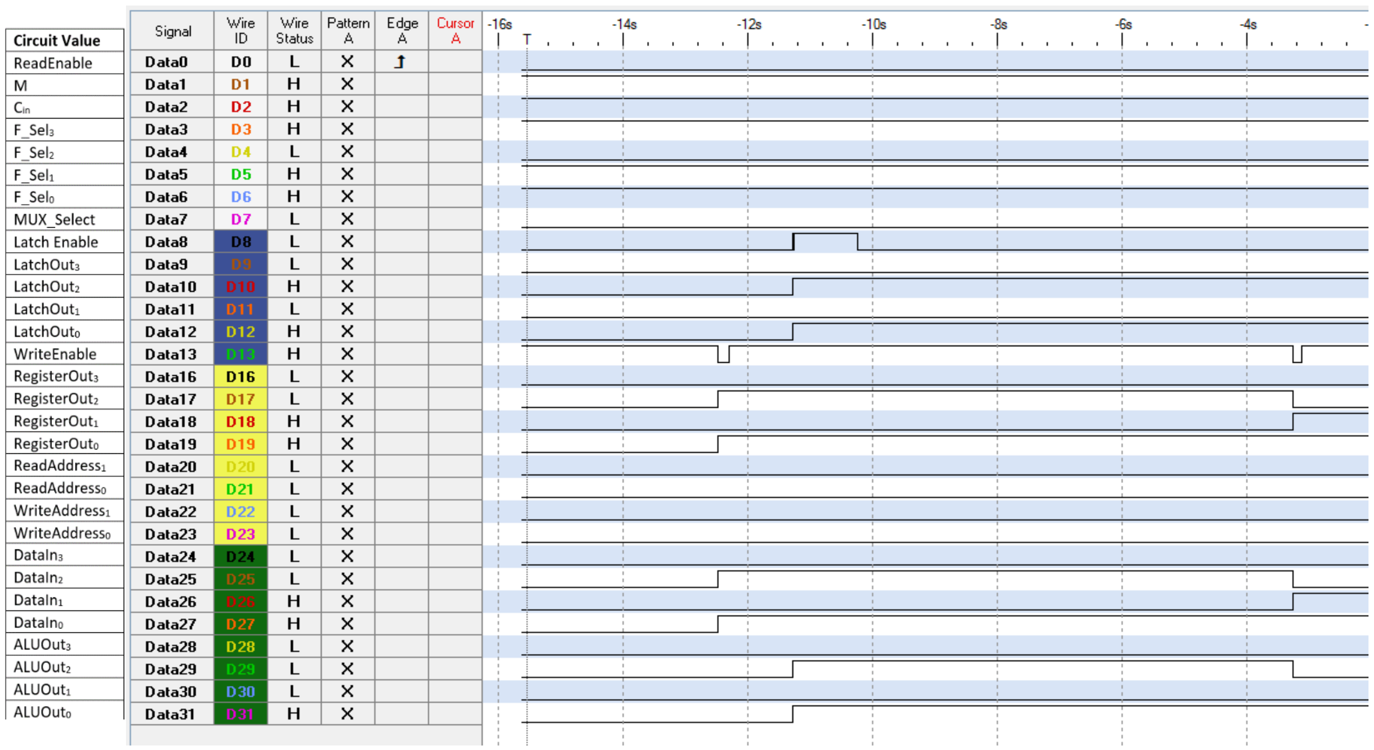
Circuit Value	Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
ReadEnable	Data0	D0	L	X	↥	
M	Data1	D1	L	X		
C_{in}	Data2	D2	H	X		
F_Sel$_3$	Data3	D3	L	X		
F_Sel$_2$	Data4	D4	H	X		
F_Sel$_1$	Data5	D5	H	X		
F_Sel$_0$	Data6	D6	L	X		
MUX_Select	Data7	D7	L	X		
Latch Enable	Data8	D8	L	X		
LatchOut$_3$	Data9	D9	H	X		
LatchOut$_2$	Data10	D10	L	X		
LatchOut$_1$	Data11	D11	H	X		
LatchOut$_0$	Data12	D12	L	X		
WriteEnable	Data13	D13	H	X		
RegisterOut$_3$	Data16	D16	L	X		
RegisterOut$_2$	Data17	D17	L	X		
RegisterOut$_1$	Data18	D18	H	X		
RegisterOut$_0$	Data19	D19	H	X		
ReadAddress$_1$	Data20	D20	L	X		
ReadAddress$_0$	Data21	D21	L	X		
WriteAddress$_1$	Data22	D22	L	X		
WriteAddress$_0$	Data23	D23	L	X		
DataIn$_3$	Data24	D24	H	X		
DataIn$_2$	Data25	D25	L	X		
DataIn$_1$	Data26	D26	H	X		
DataIn$_0$	Data27	D27	H	X		
ALUOut$_3$	Data28	D28	L	X		
ALUOut$_2$	Data29	D29	H	X		
ALUOut$_1$	Data30	D30	H	X		
ALUOut$_0$	Data31	D31	L	X		

C. A OR B: *5 OR 2 = 7* (bitwise OR binary values)



Circuit Value	Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
ReadEnable	Data0	D0	L	×	↑	
M	Data1	D1	H	×		
C$_{in}$	Data2	D2	H	×		
F_Sel$_3$	Data3	D3	H	×		
F_Sel$_2$	Data4	D4	H	×		
F_Sel$_1$	Data5	D5	H	×		
F_Sel$_0$	Data6	D6	L	×		
MUX_Select	Data7	D7	L	×		
Latch Enable	Data8	D8	L	×		
LatchOut$_3$	Data9	D9	L	×		
LatchOut$_2$	Data10	D10	H	×		
LatchOut$_1$	Data11	D11	L	×		
LatchOut$_0$	Data12	D12	H	×		
WriteEnable	Data13	D13	H	×		
RegisterOut$_3$	Data16	D16	L	×		
RegisterOut$_2$	Data17	D17	L	×		
RegisterOut$_1$	Data18	D18	H	×		
RegisterOut$_0$	Data19	D19	L	×		
ReadAddress$_1$	Data20	D20	L	×		
ReadAddress$_0$	Data21	D21	L	×		
WriteAddress$_1$	Data22	D22	L	×		
WriteAddress$_0$	Data23	D23	L	×		
DataIn$_3$	Data24	D24	L	×		
DataIn$_2$	Data25	D25	L	×		
DataIn$_1$	Data26	D26	H	×		
DataIn$_0$	Data27	D27	L	×		
ALUOut$_3$	Data28	D28	L	×		
ALUOut$_2$	Data29	D29	H	×		
ALUOut$_1$	Data30	D30	H	×		
ALUOut$_0$	Data31	D31	H	×		

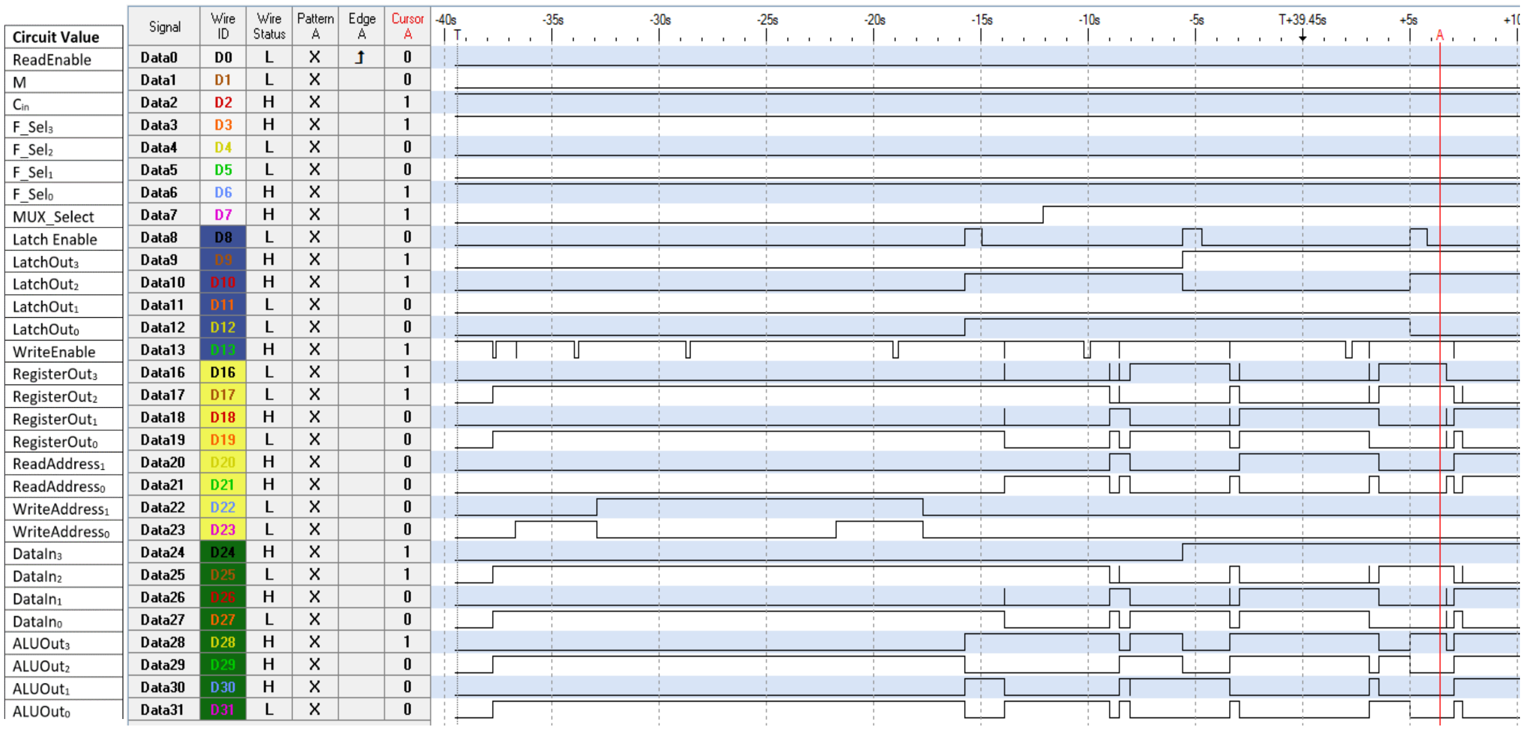
D. A AND B: *5 AND 3 = 1* (bitwise AND binary values)



Circuit Value	Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
ReadEnable	Data0	D0	L	×	↑	
M	Data1	D1	H	×		
C$_{in}$	Data2	D2	H	×		
F_Sel$_3$	Data3	D3	H	×		
F_Sel$_2$	Data4	D4	L	×		
F_Sel$_1$	Data5	D5	H	×		
F_Sel$_0$	Data6	D6	H	×		
MUX_Select	Data7	D7	L	×		
Latch Enable	Data8	D8	L	×		
LatchOut$_3$	Data9	D9	L	×		
LatchOut$_2$	Data10	D10	H	×		
LatchOut$_1$	Data11	D11	L	×		
LatchOut$_0$	Data12	D12	H	×		
WriteEnable	Data13	D13	H	×		
RegisterOut$_3$	Data16	D16	L	×		
RegisterOut$_2$	Data17	D17	L	×		
RegisterOut$_1$	Data18	D18	H	×		
RegisterOut$_0$	Data19	D19	H	×		
ReadAddress$_1$	Data20	D20	L	×		
ReadAddress$_0$	Data21	D21	L	×		
WriteAddress$_1$	Data22	D22	L	×		
WriteAddress$_0$	Data23	D23	L	×		
DataIn$_3$	Data24	D24	L	×		
DataIn$_2$	Data25	D25	L	×		
DataIn$_1$	Data26	D26	H	×		
DataIn$_0$	Data27	D27	H	×		
ALUOut$_3$	Data28	D28	L	×		
ALUOut$_2$	Data29	D29	L	×		
ALUOut$_1$	Data30	D30	L	×		
ALUOut$_0$	Data31	D31	H	×		

Figure 13: Logic Analyzer 4 Consecutive Addition Test

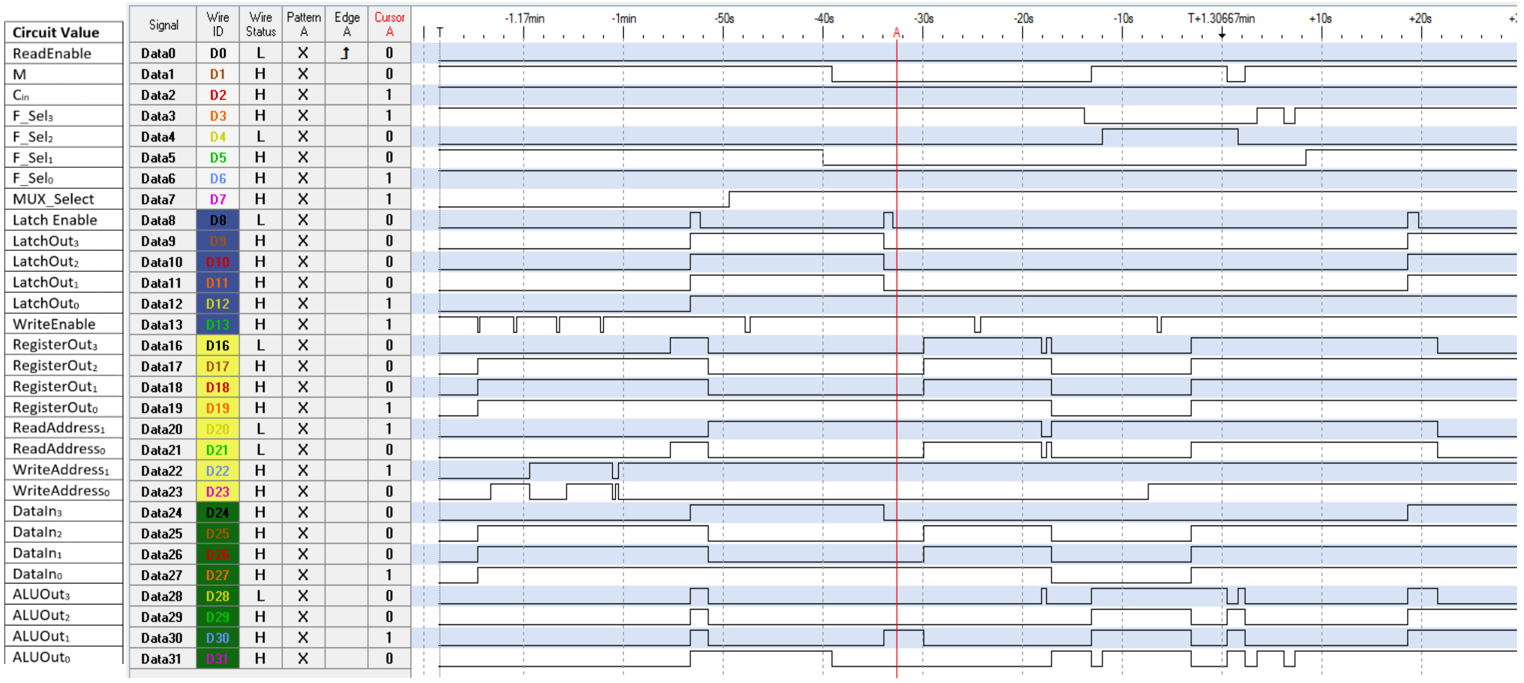
A. <u>A PLUS B</u>: *5 plus 4 plus 3 plus 2 = 14*



Circuit Value	Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
ReadEnable	Data0	D0	L	X	↑	0
M	Data1	D1	L	X		0
C_{in}	Data2	D2	H	X		1
F_Sel_3	Data3	D3	H	X		1
F_Sel_2	Data4	D4	L	X		0
F_Sel_1	Data5	D5	L	X		0
F_Sel_0	Data6	D6	H	X		1
MUX_Select	Data7	D7	H	X		1
Latch Enable	Data8	D8	L	X		0
$LatchOut_3$	Data9	D9	H	X		1
$LatchOut_2$	Data10	D10	H	X		1
$LatchOut_1$	Data11	D11	L	X		0
$LatchOut_0$	Data12	D12	L	X		0
WriteEnable	Data13	D13	H	X		1
$RegisterOut_3$	Data16	D16	L	X		1
$RegisterOut_2$	Data17	D17	L	X		1
$RegisterOut_1$	Data18	D18	H	X		0
$RegisterOut_0$	Data19	D19	L	X		0
$ReadAddress_1$	Data20	D20	H	X		0
$ReadAddress_0$	Data21	D21	H	X		0
$WriteAddress_1$	Data22	D22	L	X		0
$WriteAddress_0$	Data23	D23	L	X		0
$DataIn_3$	Data24	D24	H	X		1
$DataIn_2$	Data25	D25	L	X		1
$DataIn_1$	Data26	D26	H	X		0
$DataIn_0$	Data27	D27	L	X		0
$ALUOut_3$	Data28	D28	H	X		1
$ALUOut_2$	Data29	D29	H	X		0
$ALUOut_1$	Data30	D30	H	X		0
$ALUOut_0$	Data31	D31	L	X		0

Figure 14: Logic Analyzer Multiplication Test

A. <u>Mix of Arithmetic and Logic:</u> *7 multiplied with 15 = 7* (first partial sum only)



Circuit Value	Signal	Wire ID	Wire Status	Pattern A	Edge A	Cursor A
ReadEnable	Data0	D0	L	X	↑	0
M	Data1	D1	H	X		0
C_{in}	Data2	D2	H	X		1
F_Sel_3	Data3	D3	H	X		1
F_Sel_2	Data4	D4	L	X		0
F_Sel_1	Data5	D5	H	X		0
F_Sel_0	Data6	D6	H	X		1
MUX_Select	Data7	D7	H	X		1
Latch Enable	Data8	D8	L	X		0
$LatchOut_3$	Data9	D9	H	X		0
$LatchOut_2$	Data10	D10	H	X		0
$LatchOut_1$	Data11	D11	H	X		0
$LatchOut_0$	Data12	D12	H	X		1
WriteEnable	Data13	D13	H	X		1
$RegisterOut_3$	Data16	D16	L	X		0
$RegisterOut_2$	Data17	D17	H	X		0
$RegisterOut_1$	Data18	D18	H	X		0
$RegisterOut_0$	Data19	D19	H	X		1
$ReadAddress_1$	Data20	D20	L	X		1
$ReadAddress_0$	Data21	D21	L	X		0
$WriteAddress_1$	Data22	D22	H	X		1
$WriteAddress_0$	Data23	D23	H	X		0
$DataIn_3$	Data24	D24	H	X		0
$DataIn_2$	Data25	D25	H	X		0
$DataIn_1$	Data26	D26	H	X		0
$DataIn_0$	Data27	D27	H	X		1
$ALUOut_3$	Data28	D28	L	X		0
$ALUOut_2$	Data29	D29	H	X		0
$ALUOut_1$	Data30	D30	H	X		1
$ALUOut_0$	Data31	D31	H	X		0

Upon close inspection, all the waveforms matched nearly identically with those produced in the Quartus simulations, save timing differences. However, there was one minor discrepancy found. The in the subtraction operation and 4 consecutive addition tests, signal Data24, or the most significant bit of DataIn, should be low, as it was physically switched off. Connections and voltage threshold were checked, but none were the issue, so the cause of the problem is unknown. However, this did not affect the final value of the circuit, so the tests were still all successfully completed.

Conclusion:
All results were as expected. The built circuit was able to perform as it was intended, producing the same outputs as those generated in the Quartus II simulation. The one major advantage of the Logic Analyzer was that it displayed all of the logic values of any relevant signal. Even though the results were all accurate, this would have made debugging much simpler, as the particular signal that was incorrect could be singled out and troubleshot.

Summary:

The goal of Laboratory 12 was to design and build a 4-bit ALU circuit, incorporating the 4x4 register file built from lab 11. The design was once again first planned conceptually, then built and simulated digitally, and then built and tested physically. Rigorous testing was conducted at each stage to prevent build up of error and find problems at their root cause. In the end, the circuit was able to successfully perform all functions intended.

(References on next page)

References:

- 74193 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464854_2/courses/2184_UPITT_ECE_0501_SEC1010/74193.pdf

- 74247 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464855_2/courses/2184_UPITT_ECE_0501_SEC1010/74247.pdf

- 74670 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464856_2/courses/2184_UPITT_ECE_0501_SEC1010/74670.pdf

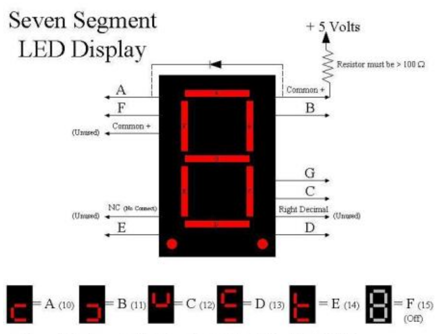
- 7400 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464846_2/courses/2184_UPITT_ECE_0501_SEC1010/7400.pdf

- 74157 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464843_2/courses/2184_UPITT_ECE_0501_SEC1010/74AC157.pdf

- 7474 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464850_2/courses/2184_UPITT_ECE_0501_SEC1010/7474.pdf

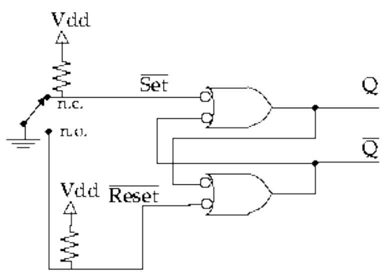
- 74181 datasheet: https://courseweb.pitt.edu/bbcswebdav/pid-24291878-dt-content-rid-23464853_2/courses/2184_UPITT_ECE_0501_SEC1010/74181.pdf

- LED outputs to Display Conversion:



7 Segment Display symbols for Hexadecimal values A-F in old set of LEDs.

- Debounce Switch Circuit:


Vdd
Set
Q
n.c.
n.o.
Q
Vdd
Reset