**PURPOSE**

The purpose of this lab was to design and build an Arithmetic Logic Unit (ALU) using the register file from Lab 11 as memory. The ALU was able to perform several logical and arithmetic functions using two 4-bit inputs. These functions would produce a 4-bit output. A control signal was used to read this output back into the register file when desired. A 4-bit latch was used as one of the inputs to the ALU, along with a selected register from the register file.

The Altera Quartus II software was used to design and test an initial implementation of the circuit, after which the circuit was designed on a protoboard and tested using the Intronix LogicPort Logic Analyzer software. Both implementations of the circuit were tested using a set of operations that were converted to pseudocode. Waveforms of these tests were generated in both the Altera Quartus software as well as the LogicPort software. These waveforms were compared to verify the ALU’s functionality.

**PROCEDURE**

1. A new project was created in the Altera Quartus II simulation software. A new schematic file was added to the project.

2. The 4-bit latch was designed using two 7474 packages, each containing two D flip-flops. The active-low Set and Reset inputs were tied to Vdd as the circuit was not to be driven by these inputs. A Latch Enable input (L\_EN) was connected to the clock signal of each flip-flop. Whenever the Latch Enable input was set high, data would be passed into the latch and held. The 4-bit latch sub-circuit is shown in Figure 1.

3. A schematic file was added to the project for the main ALU circuit. The following components were added to the circuit: one 74157 eight-to-one multiplexor, two 74193 counters (from Lab 11), one 74670 register file (from Lab 11), the 4-bit latch sub-circuit, one 74181 ALU, and two 74247 seven-segment display decoders.

4. Four of the inputs to the 4-bit ALU were connected to the outputs of the 74670 register file. The other four inputs were connected to the outputs of the 4-bit latch. The latch inputs were connected to the outputs of the register file. The multiplexor was added to control whether the ALU output would be read back into a register; this is controlled by the IN\_SEL signal. This circuit is shown in Figure 2.

5. A new vector waveform file was added to the project. The following signals were added to the waveform: the four data inputs, named DATA\_IN; the multiplexor input IN\_SEL, the Latch Enable input L\_EN; the latch contents, named A; the register contents, named B; the ALU outputs, named F; the ALU operation code, named F\_SEL; the ALU arithmetic/logical input, named M; the register read address and write address; and the Write Enable input.

6. Four operations were simulated on the circuit, two arithmetic and two logical. The two arithmetic operations were 3 + 4 and 10 – 5 – 1. The two logical operations were 3 AND 4’ and 10 OR 5. This waveform is shown in Figure 3.

7. A second vector waveform file was added to the project with the same signals as the first. The input signals were toggled in order to perform 2 + 3 + 4 + 5 and store the result in a register. This waveform is shown in Figure 4.

8. A third vector waveform file was added to the project with the same signals as the first two. On this waveform, the first partial product of 7 \* 5 was calculated. This waveform is shown in Figure 5. Figure 6 shows another multiplication waveform, this time of 6 \* 3.

9. The ALU circuit was constructed on a protoboard by reusing the register file circuit from Lab 11. SPDT switches were used for the L\_EN, IN\_SEL, and M inputs. DIP switches were used for the DATA\_IN and F\_SEL inputs.

10. A new project was created in the LogicPort Logic Analyzer software to observe the circuit. Since the circuit would be tested by manually setting the values of the data, L\_EN, IN\_SEL, M, F\_SEL, Read Address, Write Address, and Write Enable, some settings in the LogicPort software had to be adjusted so it could sample data from a longer time period. The Voltage Threshold value was decreased to 0.85 V. The clock frequency was decreased to 10 kHz. The pre-fill time limit was set to -0.5 seconds and the post-fill time limit was set to one minute.

11. The Read Address and Write Address inputs were set to be the clock signals of this waveform, as it was necessary for the other signals to change only in response to these two signals. The other signals were to trigger on the rising edge of the Read Address and Write Address inputs. The operations simulated in Steps 6-7 were tested on the physical circuit. These waveforms are shown in Figures 7, 8, and 9.

**RESULTS**

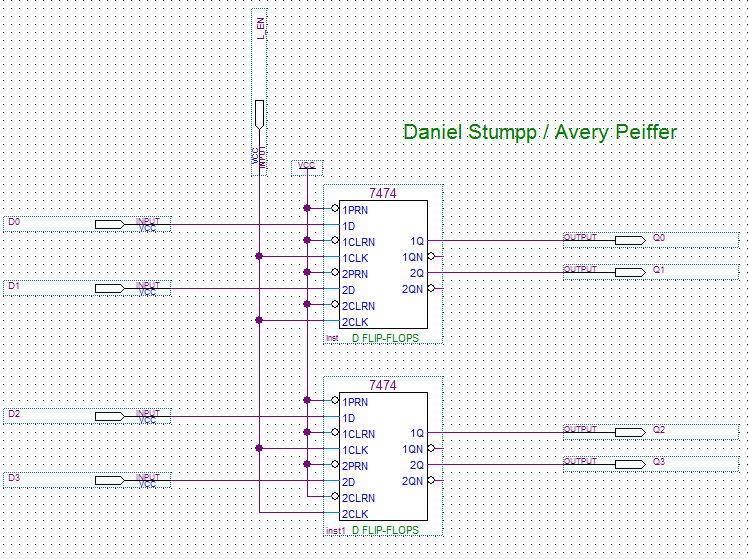


Figure : The latch sub-circuit designed using two 7474 packages.

Figure 1 above shows the latch sub-circuit designed for the overall ALU circuit, shown below in Figure 2. This latch holds the data passed in by inputs D0-D3 when the L\_EN signal is high. In the overall circuit, the latch acts as one of the inputs to the ALU. In order to perform an operation, a value had to first be written to the latch through the register file. The operation code for the ALU, shown by the signals S0-S3 and M, was then set and the operation was performed.

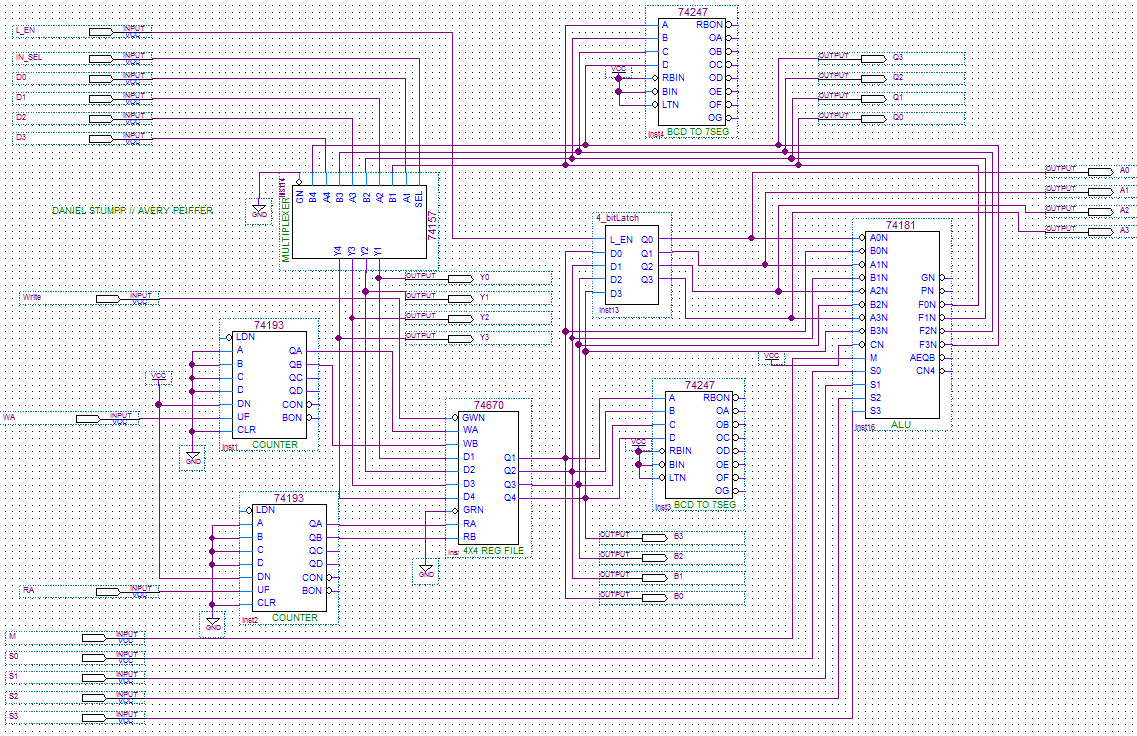


Figure : The Altera Quartus schematic for the ALU circuit.

Figure 3 below shows the Altera Quartus simulation of the four operations discussed in Step 6 of the Procedure section. The first operation (3 + 4) occurs from 0 to 150 ns; likewise, the second operation (10 – 5 – 1) occurs from 150 to 300 ns, the third operation (10 OR 5) occurs from 300-450 ns, and the fourth operation (3 AND 4’) occurs from 450-640 ns. The results for each operation are shown in the F output in that order. There are some intermediate values that are present at the F output due to changes to the inputs. For example, when the read address input (RA) is set high, the F output momentarily reads the output of that register, regardless of whether that is the final answer.

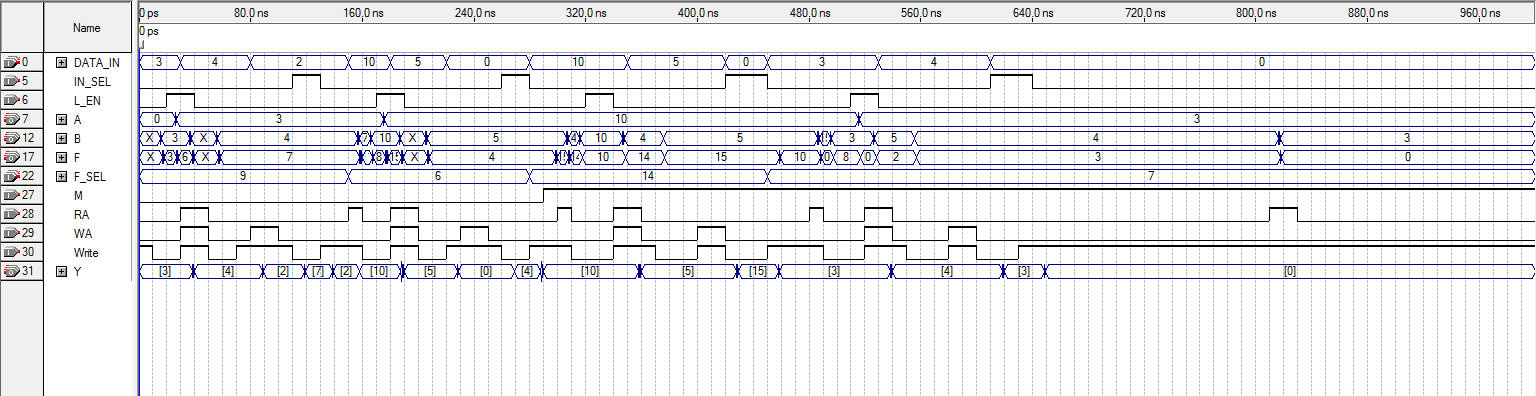


Figure : Simulation of two arithmetic and two logical operations on the ALU circuit.

Figure 4 shows the addition of four hex digits (in this case, 2 + 3 + 4 + 5) in an Altera Quartus waveform. 2, 3, and 4 were loaded into registers, while 5 was loaded into the latch. Then, each register was methodically added to the latch value and the result was written back into the latch. The final result was then stored back into one of the registers.

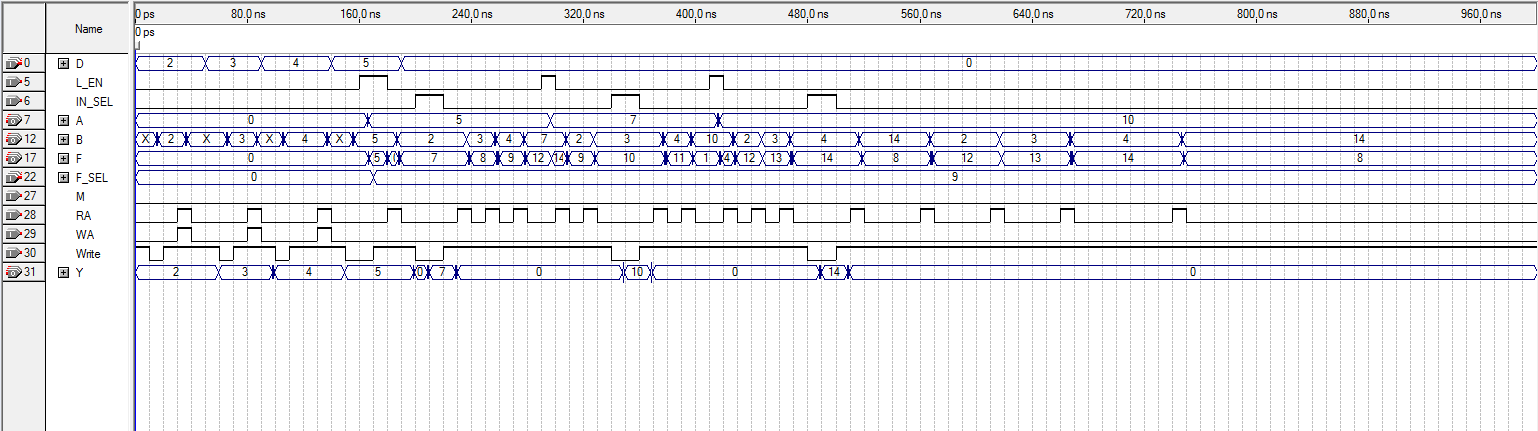


Figure 4: Simulation of 2 + 3 + 4 + 5. Note that after the operation has finished at 560 ns, each register is read in order. This yields three of the original operands and the final result, 14.

Figure 5 (split into two images for clarity) shows the first partial product of 7 \* 5. Since the first partial product is 0111 \* 0001, the result will be 0111. A lengthy set of pseudocode instructions was developed to implement this operation. These instructions, shown in Table 1 have been annotated to show where on the waveform each operation occurs. The instructions essentially found the least significant bit of B and multiplied it by each bit of A. This was accomplished by doing the AND operation of B0 and A. This value was added to a running sum through the OR operation. B0 was shifted to the left, aligning itself with the next most significant bit of A, and the process was repeated until each bit in A was multiplied by B0. This waveform was also used to find the first partial product of 6 \* 3 using the same set of instructions, shown further below in Figure 6. Again, there are many intermediate values that appear at the output throughout the operation, though this does not affect the final result.

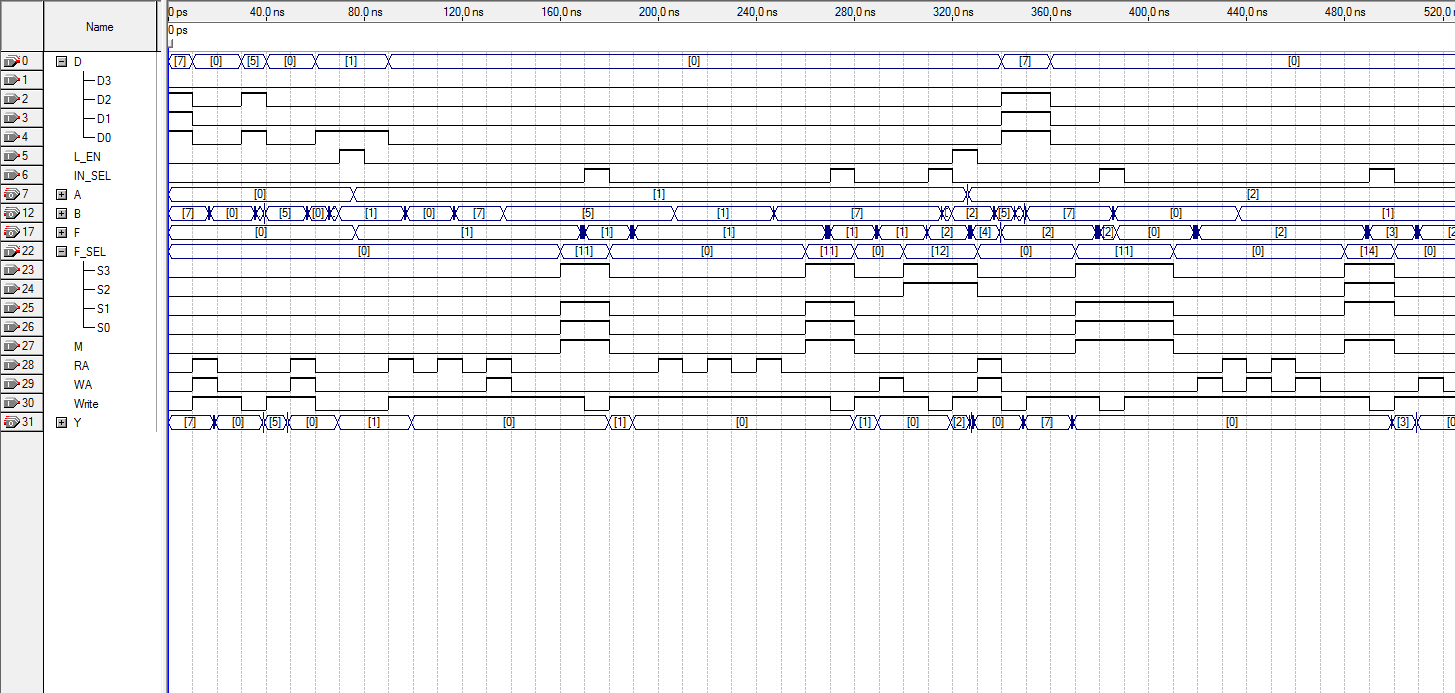
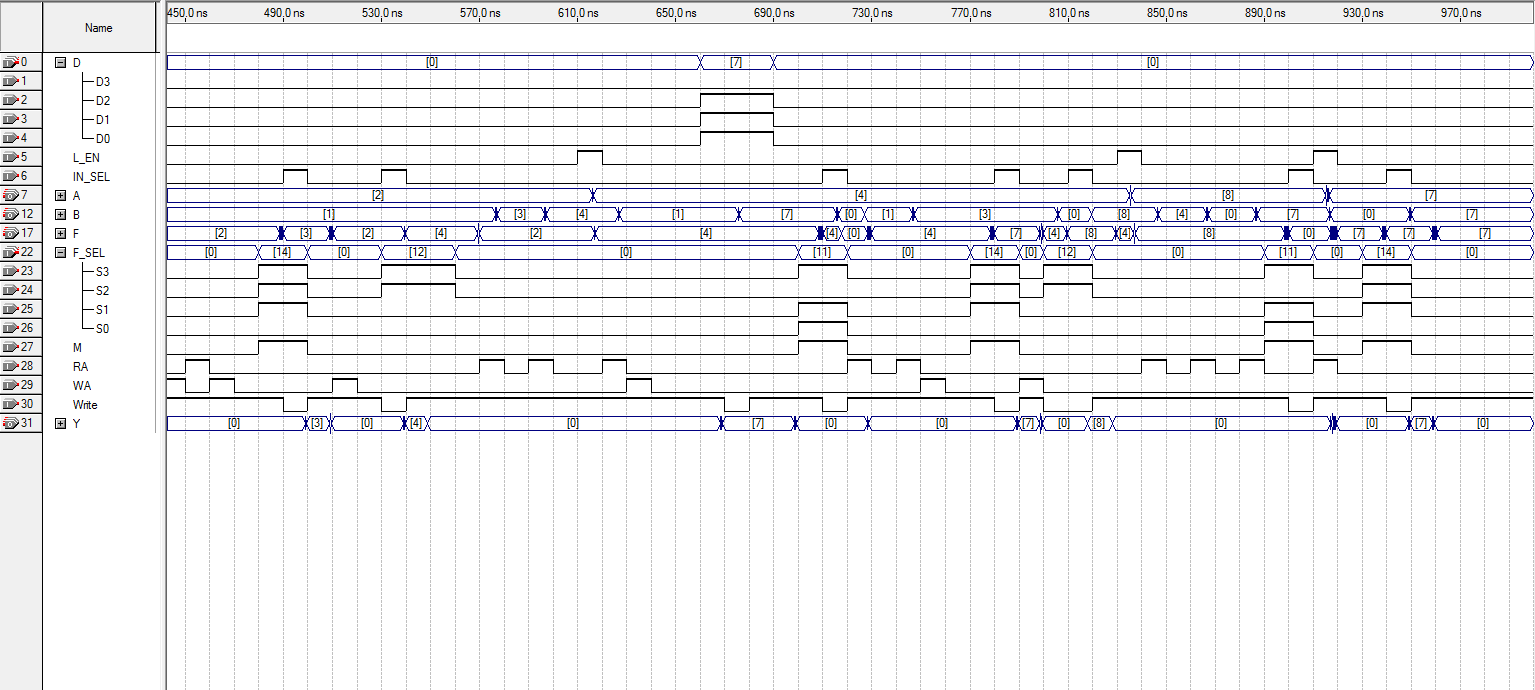
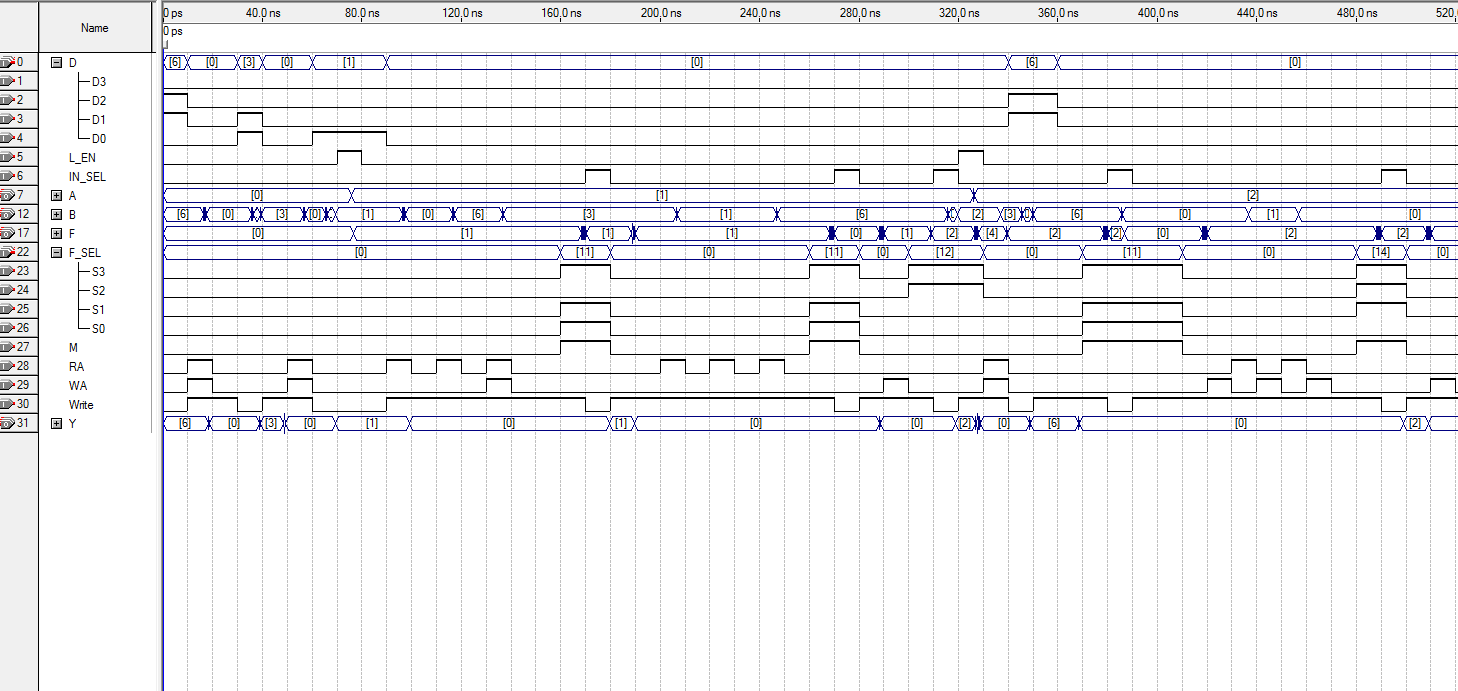


Figure 5: The first partial product of 1111 \* 0101, the result of which is shown at the F output beginning at 960 ns.

|  |  |
| --- | --- |
| **Pseudocode Instruction** | **Time of occurrence in Figure 5** |
| Load A to r0 | 0-10 ns |
| Load B to r1 | 30-40 ns |
| Load 0001 to latch | 60-90 ns |
| AND r1 & latch, store in latch | 90-180 ns |
| AND r0 & latch (Partial product of A0B0), move latch to r3 | 200-280 ns |
| Shift B0 | 290-320 ns |
| Move r0 to latch | 320-330 ns |
| Load A to r1 | 330-360 ns |
| AND latch & r1 (Partial product of A1B0) | 370-390 ns |
| OR r3 and latch, store in r0 (Result is 00[A1B0][A0B0]) | 420-500 ns |
| Shift B0 in latch | 510-560 ns |
| AND A & latch (Partial product of A2B0) | 570-720 ns |
| Move r0 to latch, OR A2B0 & latch (Result is 0[A2B0][A1B0][A0B0]) | 730-790 ns |
| Shift B0 in latch | 800-840 ns |
| Load A to r0, AND r0 & latch (Partial product of A3B0) | 840-910 ns |
| OR r3 & latch (Result is [A3B0][A2B0][A1B0][A0B0]) | 910-950 ns |



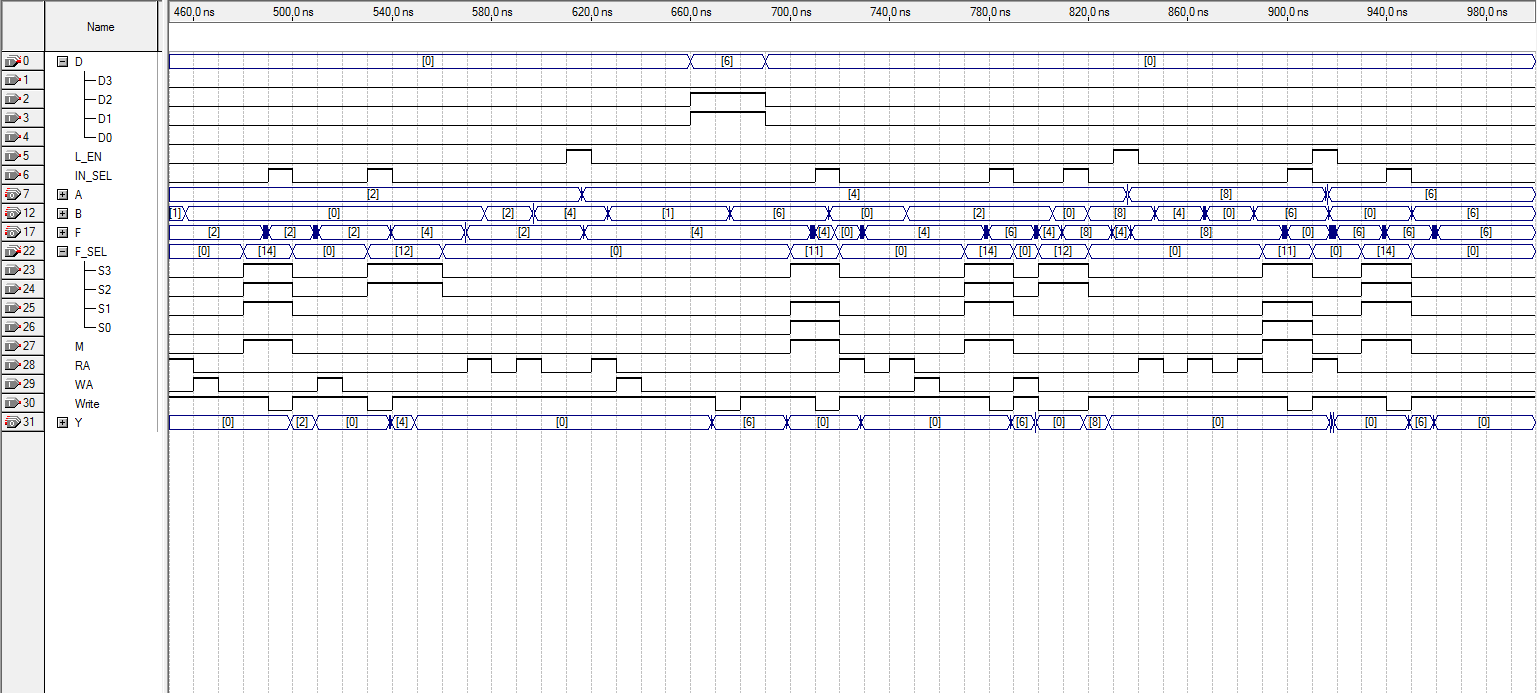


Figure 6: The first partial product of 0110 \* 0011, the result of which is shown at the F output beginning at 960 ns.

The physical circuit was constructed on a proto-board using the register file circuit from Lab 11. A seven-segment display was used to physically observe the F output. Once the circuit was built, the LogicPort Logic Analyzer was connected to test its behavior. The same operations tested in Figures 3, 4, and 5 were implemented on the physical circuit, using SPDT and DIP switches to control the inputs. These operations are shown in Figures 7a-d, 8, and 9, respectively.

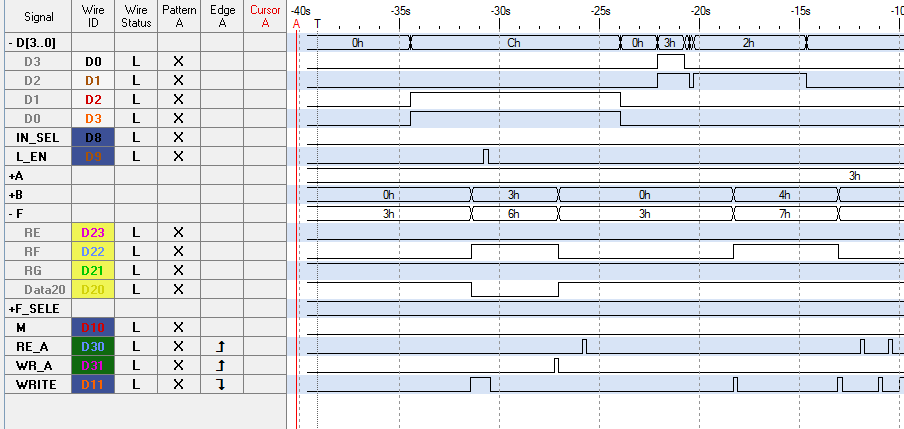


Figure 7a: Simulation of 3 + 4 in the LogicPort software. The F output contains the result, 7, when A is 3 and B is 4.

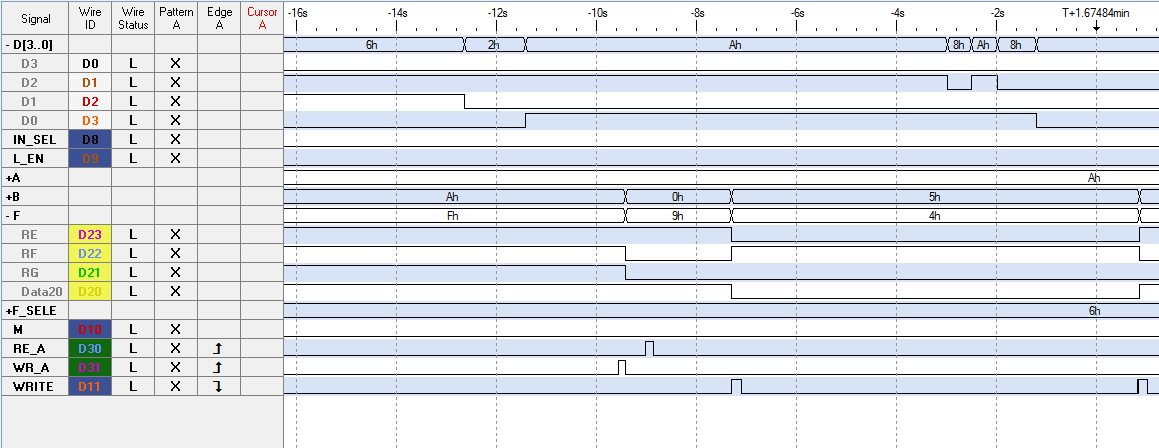


Figure 7b: Simulation of 10 – 5 – 1 on the LogicPort software. The F output contains the result, 4, when A is 10 and B is 5.

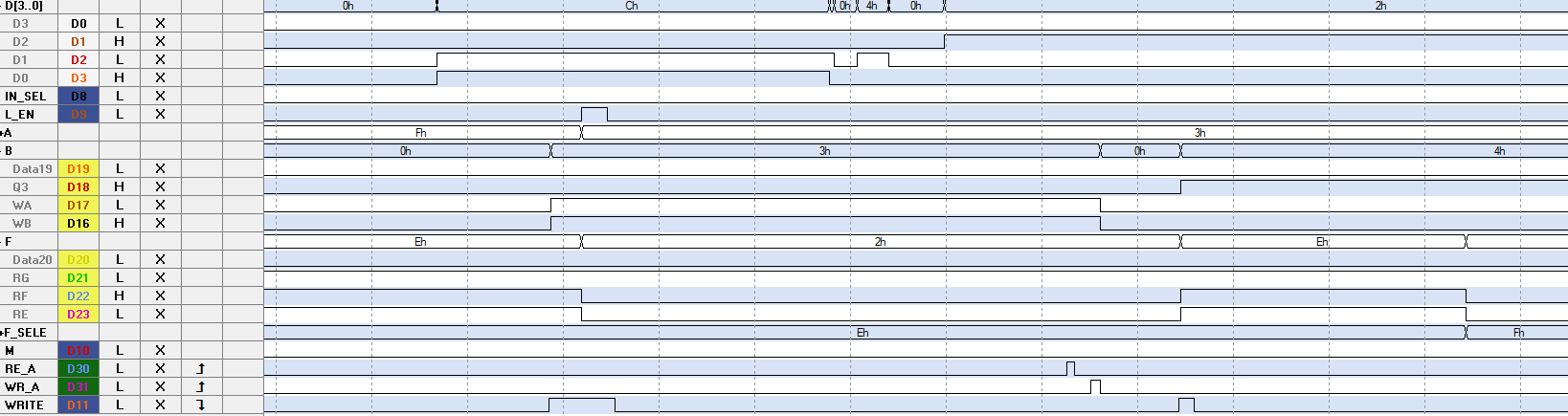


Figure 7c: Simulation of 3 AND 4’ on the LogicPort software. When A is 3 and B is 4, the output F contains the result, 3.

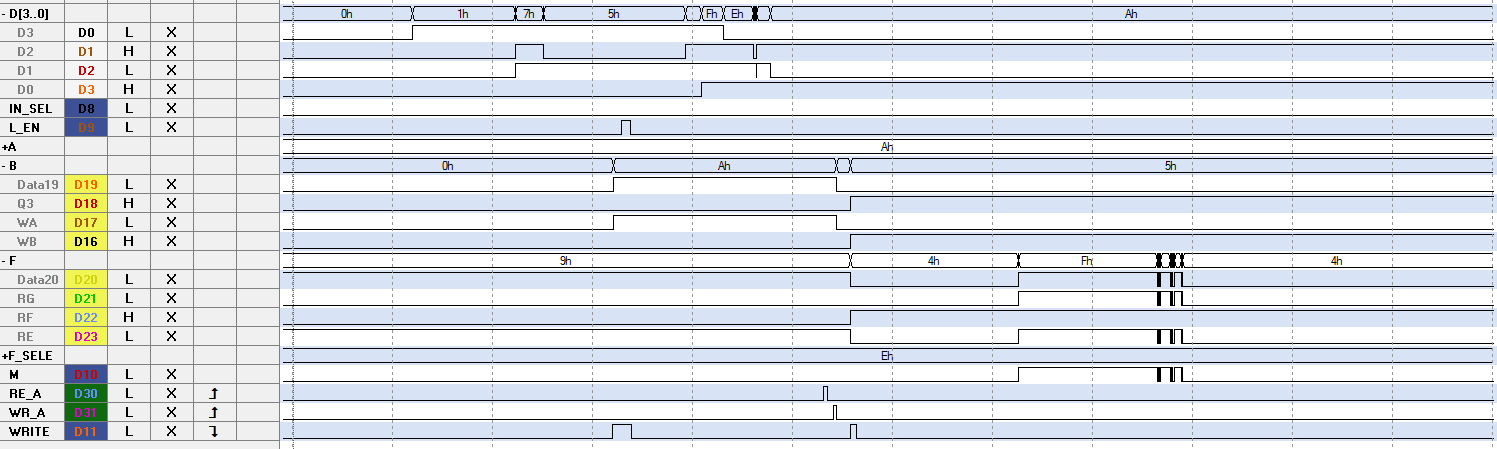
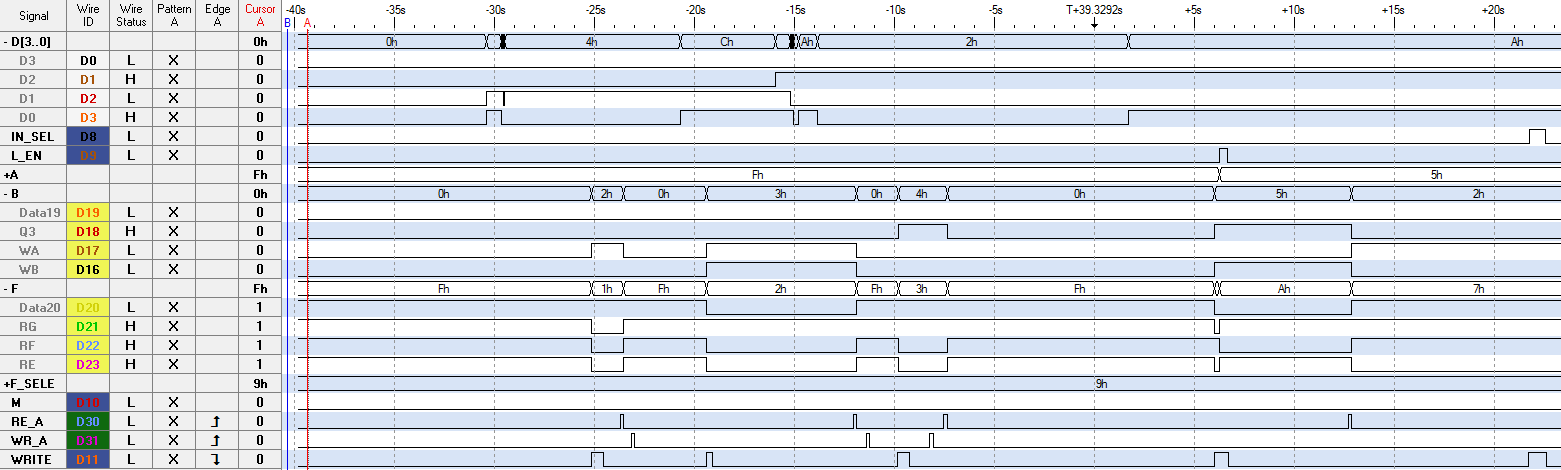


Figure 7d: Simulation of 10 OR 5 on the LogicPort software. When A is 10 and B is 5, the output F contains the result, 15 (F in hex), before unexpectedly changing its value.

It should be noted that in Figure 7d, the result F changes from 15 to 4 with no change to any of the input signals. This can likely be attributed to some accidental jostling of the circuit during the observation period. It is possible that one or more of the wires connected from the ALU outputs to the Logic Analyzer came loose and resulted in the incorrect value being displayed.

Figure 8 below shows the testing of 2 + 3 + 4 + 5 using the Logic Analyzer, again split into two images for ease of viewing. The addition is done in the same manner as the Altera Quartus simulation; 2, 3, and 4 were loaded into registers while 5 was loaded to the latch. One by one, each value was added to the latch, effectively acting as a running total.



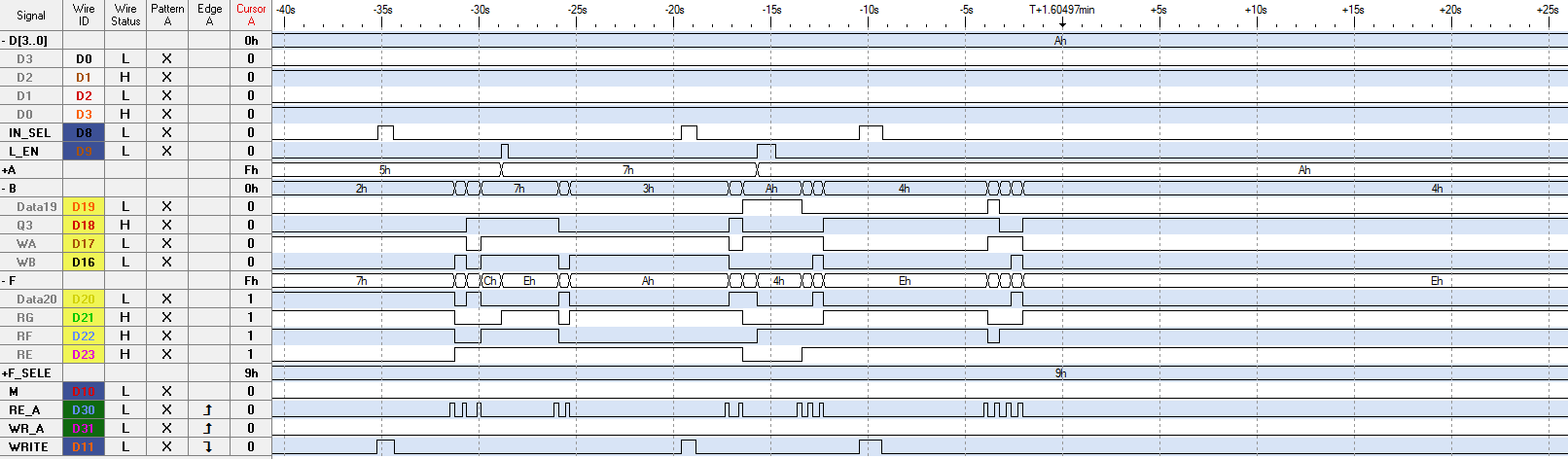
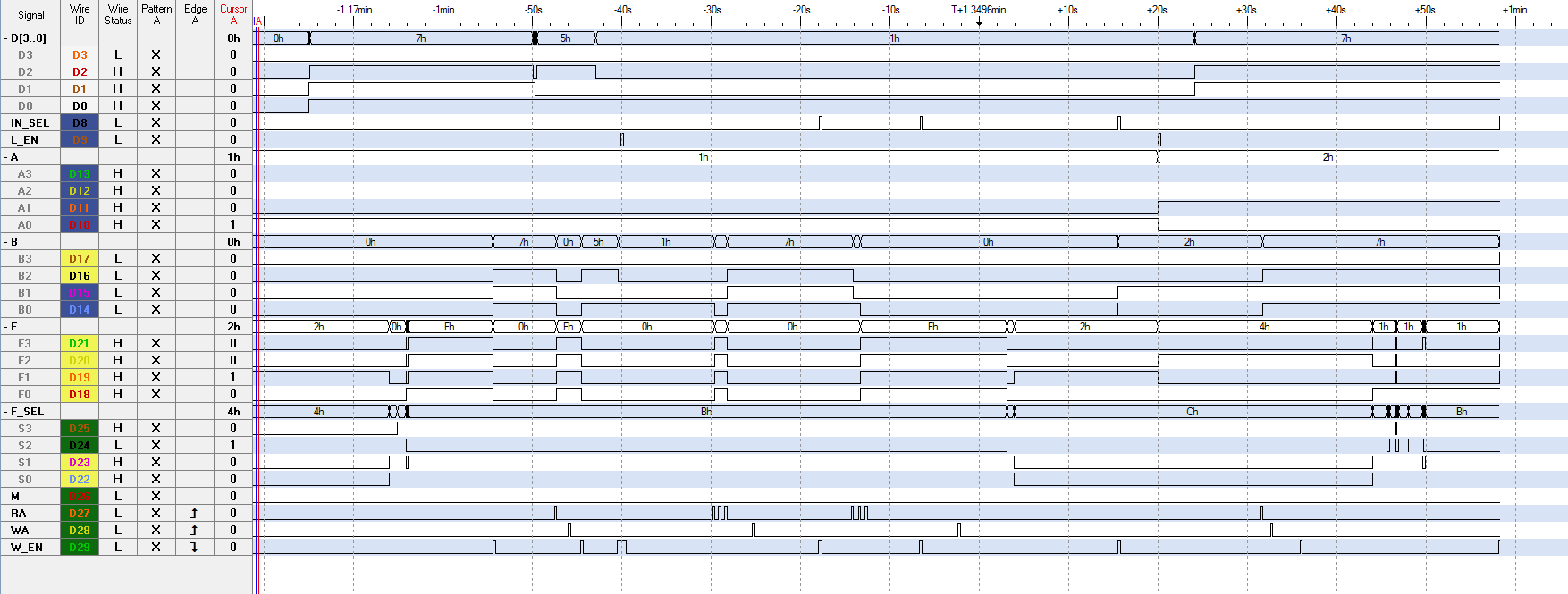


Figure 8: Simulation of 2 + 3 + 4 + 5 using the Logic Analyzer. After all values have been added, the F output contains the result, 14 (E in hex).

Figure 9 below shows the first partial product of 7 \* 5 in the LogicPort software, split into two images for clarity. This multiplication was done in the same manner as the Altera Quartus simulation. After the least significant bit of B was isolated, it was compared to the least significant bit of A using the AND operation. This value was stored, to be used later as a running total. B0 was then shifted, and the process was repeated. Following each subsequent AND operation, the result was effectively added to the running total using the OR operation.



****

Figure 9: The first partial product of 7 \* 5 using the Logic Analyzer. After the first partial product has been computed, F contains the result (7).

**CONCLUSIONS**

In this lab, an arithmetic logic unit was designed and built using the register file circuit from Lab 11 as memory. A 4-bit latch sub-circuit was designed to read data from the register file and serve as the second input to the ALU. Two seven-segment displays were used to observe the contents of the latch and the outputs of the ALU. Data from the ALU could be written back into a register depending on the select input to a multiplexor. The register file circuit and the 4-bit latch circuit were used to design the overall circuit in the Altera Quartus software before being implemented on a protoboard.

Several arithmetic and logical operations were generated to verify that the ALU could correctly perform both types of operations and that the outputs could be stored back into registers for later use. These operations were each divided into a set of pseudocode instructions that could be easily executed by toggling the circuit inputs. These operations were used to simulate the circuit schematic and to later test the physical circuit with the LogicPort Logic Analyzer. It was somewhat difficult to test the physical circuit using the Logic Analyzer, as the high number of delicate switch inputs meant that the circuit could easily display the incorrect values, as was the case in Figure 7d. More attention will have to be paid to any SPDT switches, especially, when testing a similar circuit in the future to ensure that no components are moved out of position.

**REFERENCES**

[1] Dr. Alex Jones’ Laboratory Manual

[2] Data Sheets for the 74LS193, 74LS247, 74670 and 74181 devices

[3] Seven Segment LED Display Layout Sheet

[4] Partner Daniel Stumpp