Design Project 2: Design and Implementation of a Function Unit on the DE0 Nano board

---

Jamahl Savage

ECE 2504: Introduction to Computer Engineering

October 27, 2018

Purpose and Project Description

The purpose for this design project was to create a function unit that could be used to perform several logical, shift, and arithmetic operations. This function unit utilizes two 8-bit operands called A and B to perform certain operations, which are decided based on a given 4-bit opcode. The function unit will also generate an 8-bit result of the chosen operation as well as four status bits. These outputted status bits represented any of the operation carry-out, any overflow as a result of the operation being performed, if the result of the operation was a negative value, or if the result of the was simply zero. The switches on the DE0 circuit board will be used to control a 16:1 multiplexer, which controls what is displayed on the LEDs, while the KEY[0] and KEY[1] control which address's operation is computed in the rom.txt file. According to the specifications of the project, this design must only utilize combinational logic and only using structural and dataflow Verilog constructs. By completing this project, I believe I will enrich my understanding of combinational circuits and learn how certain structural tasks can be implemented easier with the use of Verilog.

Technical Approach and Implementation

Before I decided to settle on a design to meet the project's requirements, I did a little research on function units and the operations they could contain. This research allowed to think of several ways I could approach creating the function unit and the ways I could implement that operations inside. One idea that interested me was the fact that I could create 3 separate multiplexers to handle the logical, shift, and arithmetic operations separately. However, I felt that would have been more complex to implement for this project, so I decided to have them all be controlled together in one compact function unit instead. As an alternative to grouping the operations using multiplexers, I decided to group the operations as if they were separate units based on the opcodes I assigned them.

For the opcodes 0000, 0001, and 0010 I assigned the operations not, nand, and xnor respectively and I thought of this group of opcodes as my logic unit. This unit was mainly represented by the two MSBs (Most Significant Bit) being 00. I assigned the not operation 0000 because of its ability to invert all the bits provided to it. The nand and xnor operations were assigned their opcodes based on the number of zeros in their respective opcodes, which represents the number of cases in which there would be an output of 1 in their truth table.

Similarly, the opcodes 0011, 0100, 0101, 0110, and 0111 I assigned the operations mult4, div16, mod16, csr, and csl respectively and I thought of this group of opcodes as my shift unit. This shift unit was grouped based on the two MSBs being 01 except for the special case of the operation mult4. I decided to assign 0011 to operation mult4 because it was the only shift in this group that utilized an arithmetic left shift. I gave operation div16 the opcode 0100 because it would utilize the four MSBs while operation mod16 was given the opcode 0101 they have a similar design but instead

utilizes the four LSBs (Least Significant Bit) based on my design for both operations. The operation csr was assigned 0110 while csl was assigned 0111 is because I personally associate “0” with right the direction while I associate “1” with the left direction.

Lastly, the opcodes 1000, 1001, 1010, and 1011 were assigned to the operations add, sub, inc, and neg respectively and had this group of opcodes to be known as my arithmetic unit. This unit was grouped based on the two MSBs being 10. The operation add was given the opcode 1000 while sub was assigned 1001 because in an adder-subtractor circuit, addition is associated with a carry-in of 0 while subtraction is associated with a carry-in of 1. I assigned 1010 to inc operation because it only increments the given operand by 1 while the neg operation was assigned 1011 because it has to invert all the bits of the operand before incrementing by 1.

To implement the logic unit, I used dataflow Verilog since I already knew about how the specific logic gates operated and the amount I would need to use on each bit. Thus, for the not, nand, and xnor operations I essentially just used their respective dataflow operators to fully implement them in the function unit. To create the operations associated with the shift unit, I used dataflow Verilog to decrease the complexity involving the shift related operations I wanted to create. Since all the operations in this shift unit utilized arithmetic shifts I simply used the shift operator to accomplish the type of shifts I needed to be done. The mult4 operation utilized a left arithmetic shift of 2, div16 and mod16 operations used a right arithmetic shift of 4; however, div16 only kept the four MSBs and padded the appropriate sign while mod16 only kept the four LSBs and did the same type of padding. The operations csr and csl both used arithmetic shifts of 1 bit respective of their operation names while retaining the bit that was casted out by the shift and placing at either the MSB or LSB location.

To implement the arithmetic unit, I decide to use structural Verilog mainly for purpose of practicing both types of Verilog coding. Additionally, I have previously created most of the structural Verilog needed in all the arithmetic operations for full functionality. The operations add and sub both used an adder-subtractor circuit I created using full adders and xor gates; however, for the add operation the cin was wired to ground and vice versa for the sub operation. The inc and neg operations both utilized half adders since there was no need for a second operand, additionally this design decision would lower the total gate count for the function unit. The only difference between the inc and neg operation’s structural Verilog was the fact that in order to negate a given operand, the operand would need to be inverted before incrementing it by one, thus requiring the use of not gates. As can be seen in Figures 1 – 12 of Appendix A, the operations developed utilizing dataflow and structural Verilog simulate well with sample inputs.

As previously stated in my last report, if the output of a logic gate is used to drive many other gates, then propagation delay could increase substantially. Unfortunately, while implementing the operations I would use the output of a logic gate to drive other

gates to lower the complexity and the number of gates used in the function unit. For the completely implemented function unit, there was a total gate count of 340. If given the appropriate propagation delay for each gate, the formula used to calculate the total propagation delay of the function unit would be $144t_1 + 56t_2 + 56t_3 + 8t_4 + 0t_5 + 68t_6 + 8t_7$ (Appendix D, Table 3). It's also important to note that in real world scenario, the propagation delay in the circuit can also be affected by the surrounding's temperature and internal temperature of the circuit. The resistance of the wire could increase the propagation delay of the circuit as well.

Before simulating the arithmetic logic unit (ALU), the switches used to operate the 16:1 multiplexer that controls what is displayed on the LEDs needed to be assigned values. According to the project specifications, the top 8 values were already defined by the specifications while the lower 8 were up to me to use for debugging purposes. As can be seen in Table 2 of Appendix C, I assigned the switch values of 0000, 0001, 0010, and 0011 to display either and only the status bit carry-out, overflow, negative, or zero. The rest of the switches from 0100 to 0111 were set to only show the location in the multiplexer.

Results

After creating the functioning operations for the function unit, assigning specific opcodes to them, and assigning values to the switches, I proceeded to simulating the ALU and programming it onto the DE0 Nano board. The results of simulating the completed ALU with different switch settings when the clock is set to 50 MHz can be seen in Figure 13 of Appendix E. The sample opcodes and operands for A and B used in the simulation can be found in Table 1 of Appendix B. When programming the DE0 Nano board, the results did not fail to meet my expectations because of its success with previous simulations conducted. The LEDs correctly turned on and off in addition to matching the desired output based on the switch setting at the time.

Conclusion

There were multiple ways this design project could have been approached, one way could implement a more complex design with potential for using less gates while the other utilizes a simpler design but also increases the gate count. In the end I decided to implement some of the ideas behind a more complex design into what I believed was to be a simpler design approach. This design project has provided me with the opportunity to test my understanding of combinational logic and my ability to implement them using different types of Verilog. This design project has also given me the chance to think about the type of decisions made when designing a larger system that controls and implements various features. Aside from that, after conducting several simulations on the implemented operations and the ALU itself, I believe it possible to say that I have designed and implemented a sufficient ALU that meets the project specifications.

Appendix A: Individual Operation Simulations

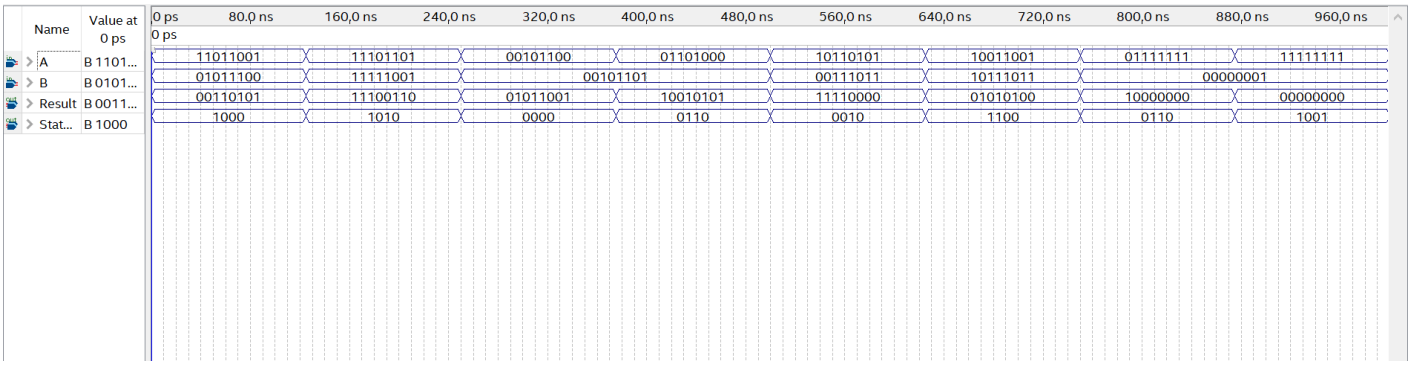

Figure 1: Addition Operation Simulation

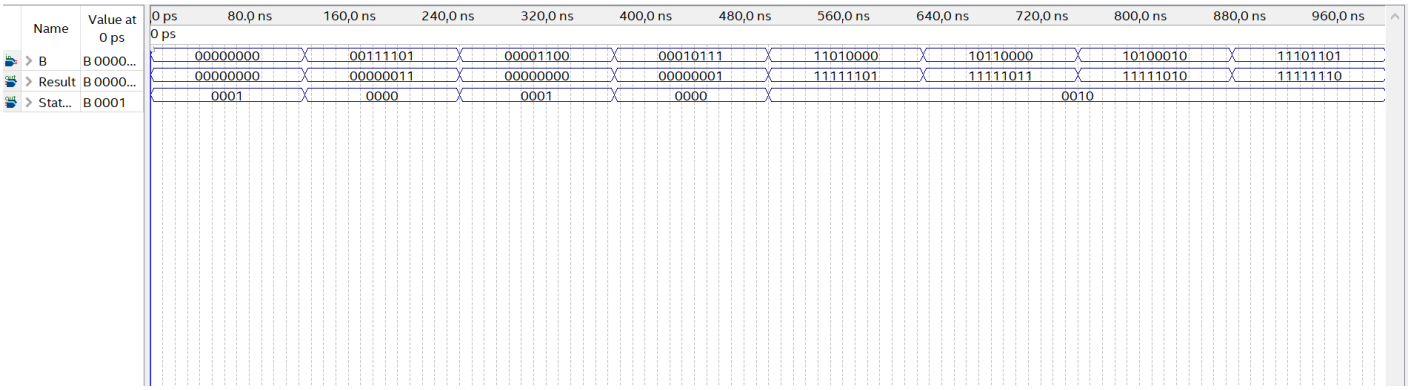

Figure 2: Division by 16 Operation Simulation

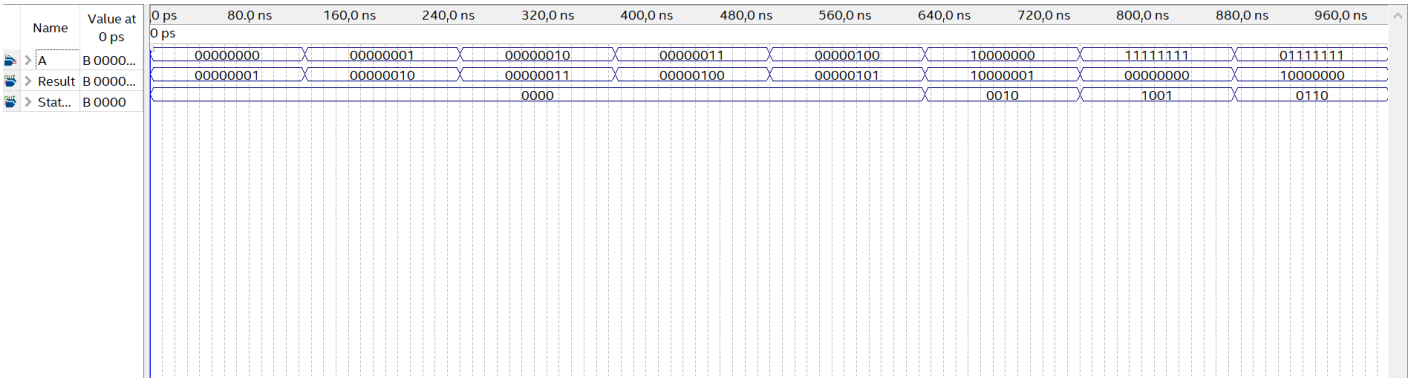

Figure 3: Increment Operation Simulation

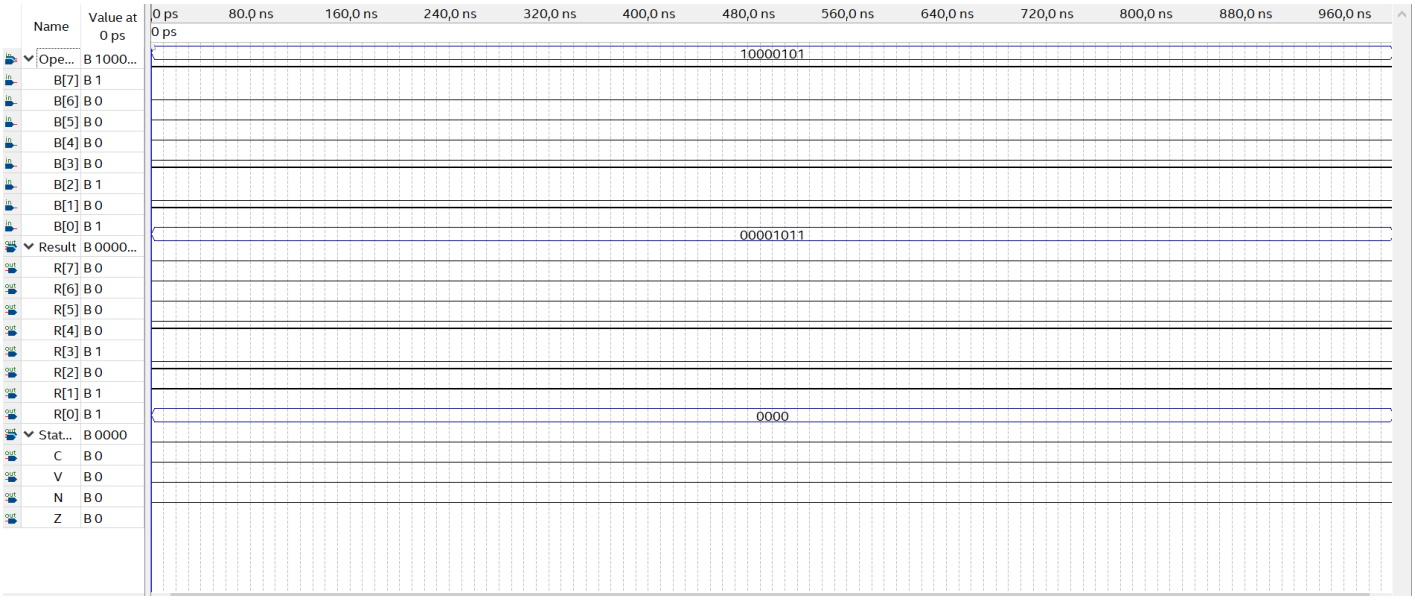

Figure 4: Left Circular Shift Operation Simulation

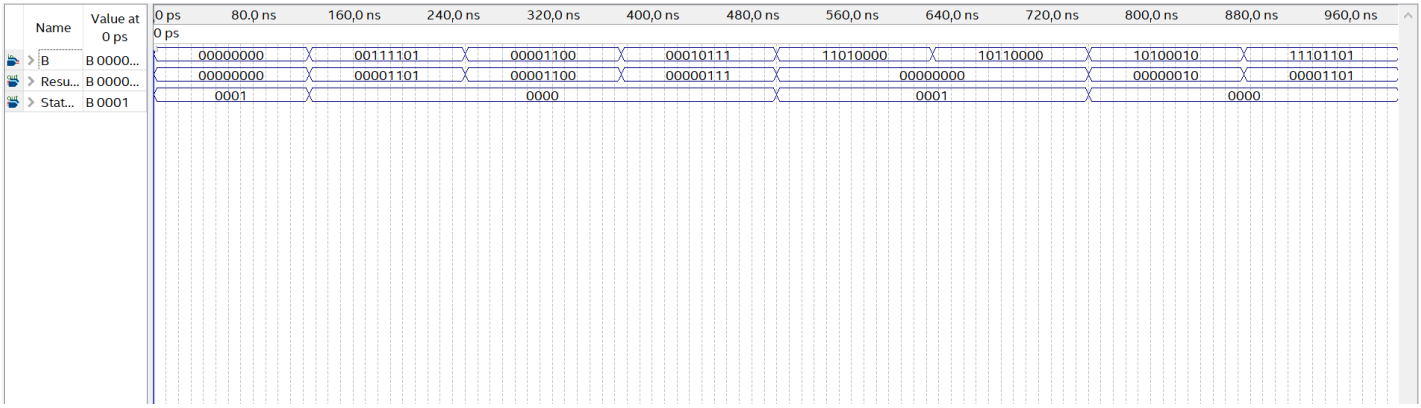

Figure 5: Modulus of 16 Operation Simulation

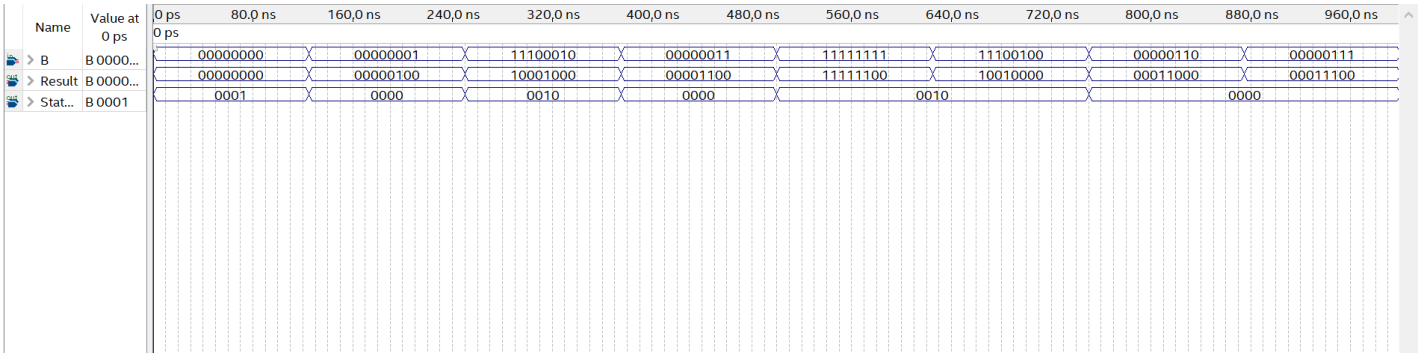

Figure 6: Multiply by 4 Operation Simulation

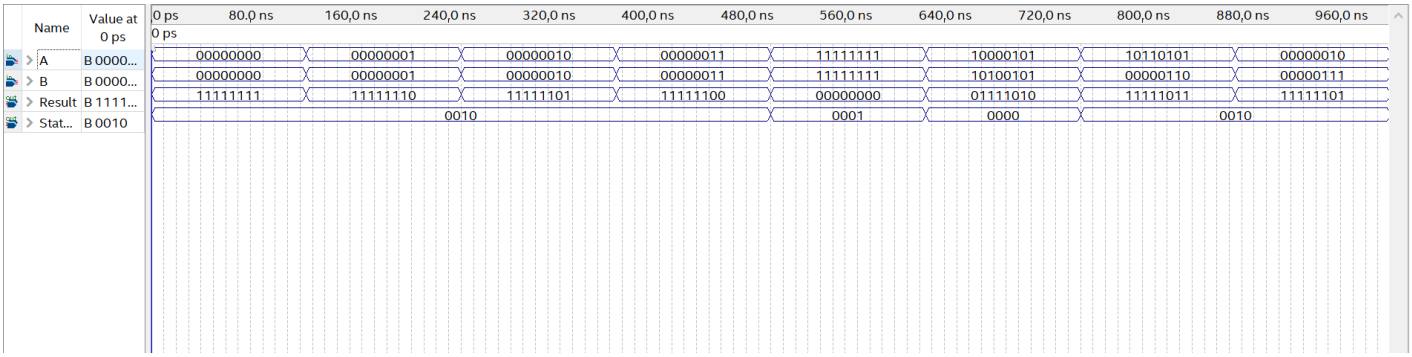

Figure 7: Nand of A and B Operation Simulation

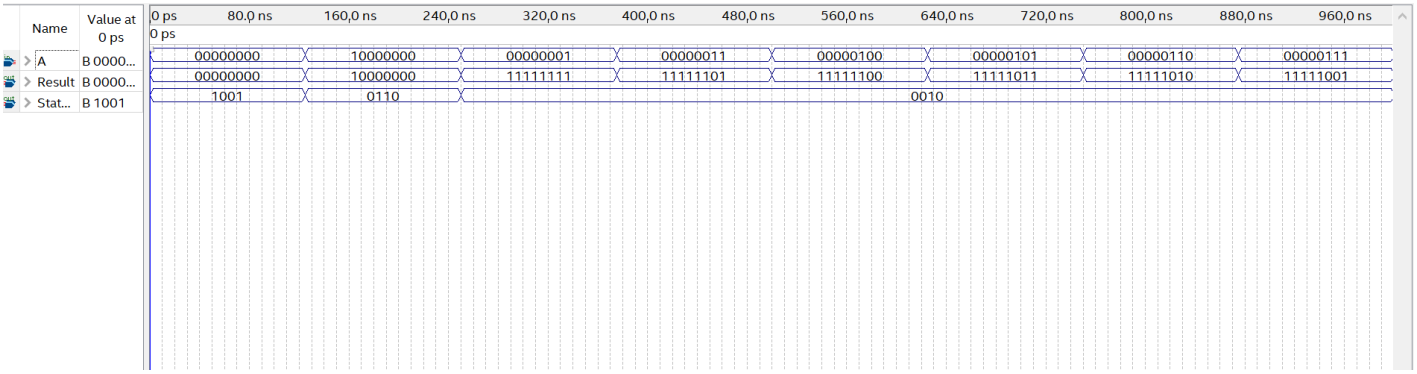

Figure 8: Negatation of A Operation Simulation

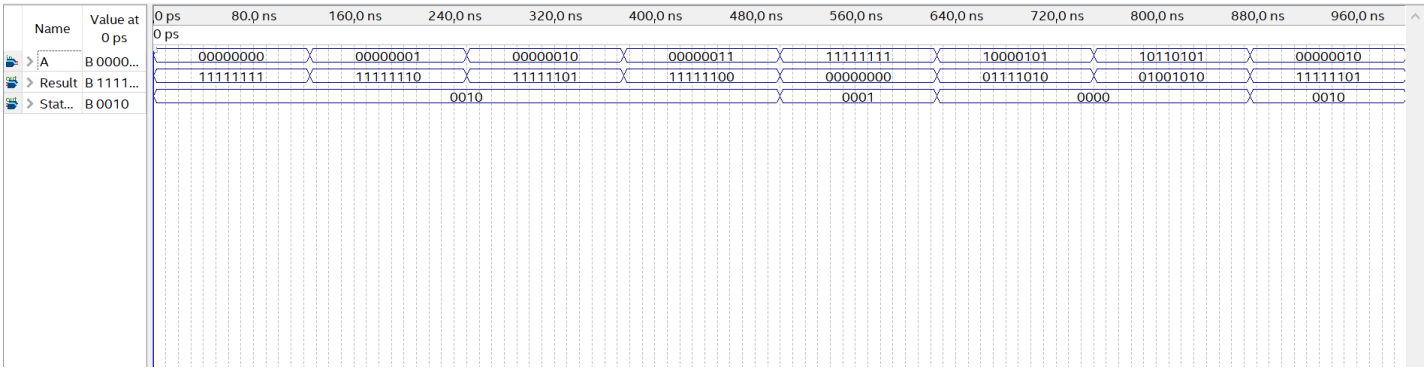

Figure 9: Not of A Operation Simulation

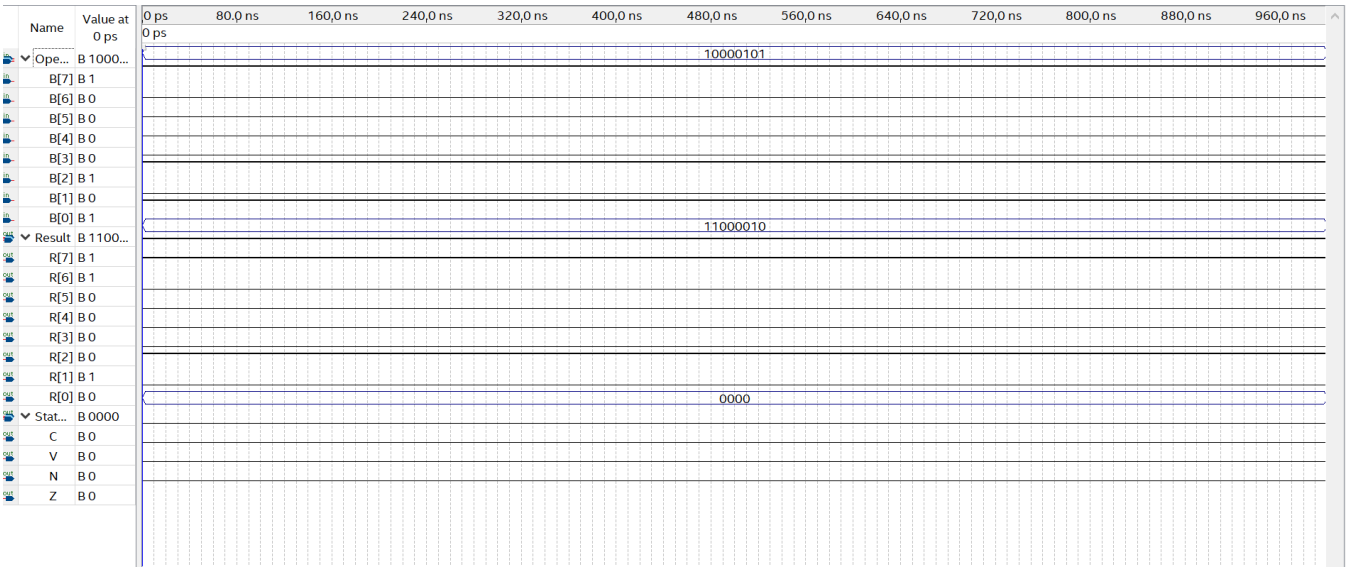

Figure 10: Right Circular Shift Operation

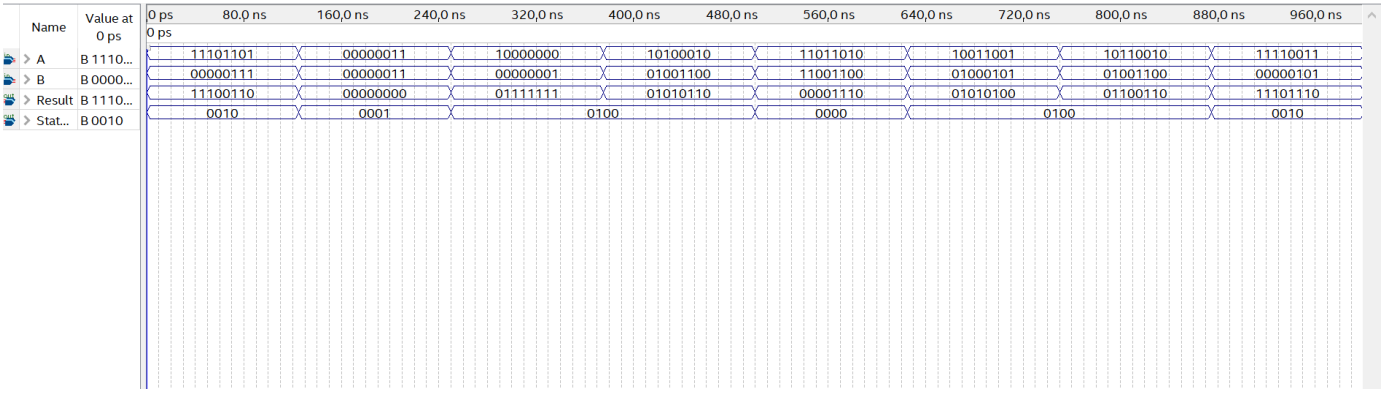

Figure 11: Subtraction of A by B Operation Simulation

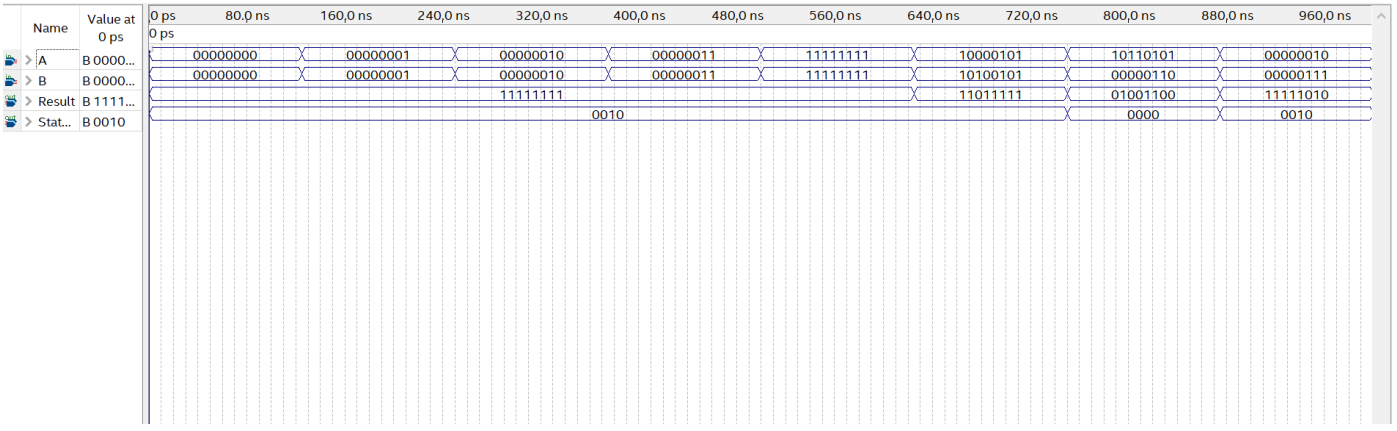

Figure 12: Xnor of A and B Operation Simulation

Appendix B: ROM Address Table

ROM address	operation	operand A	operand B
0	not	-19	-39
1	nand	92	7
2	xnor	45	89
3	mult4	44	-26
4	div16	-3	-48
5	mod16	-19	-48
6	csr	-69	89
7	csl	127	53
8	addition	127	44
9	subtraction	-16	59
A	increment	-1	59
B	negate	-39	92
C	subtraction	37	44
D	mod16	92	89
E	mult4	59	0
F	addition	127	-39

Table 1: My ROM addresses and their assigned operations as well operands that were used in the complete circuit simulation **(Appendix E)**

Appendix C: Switch Assignment Values

SW[3:0]	Value displayed on LEDs
0000	Displays the single status bit C
0001	Displays the single status bit V
0010	Displays the single status bit N
0011	Displays the single status bit Z
0100-0111	Gives back the location inside of the mux (Essentially no purpose)
1000	Right two digits of the last four digits of my student ID, in BCD
1001	Left two digits of the last four digits of my student ID, in BCD
1010	The result
1011	The status bits with padded leading 0's
1100	The address
1101	The opcode
1110	Operand B
1111	Operand A

Table 2: Shows the opcode of the switches and the value they represent

Appendix D: Function Unit Gate Count

Propagation Delay of the Operations							
Operation Name	**# AND Gates**	**# OR Gates**	**# NOT Gates**	**# NAND Gates**	**# NOR Gates**	**# XOR Gates**	**# XNOR Gates**
Addition	24	8				26	
Subtraction	24	8				26	
Increment	8					8	
Negate	8		8			8	
Circular Shift Left	16	8	8				
Circular Shift Right	16	8	8				
Modulus 16	16	8	8				
Division 16	16	8	8				
Multiplication 4	16	8	8				
Xnor							8
Nand				8			
Not			8				
Totals	144	56	56	8		68	8
Propagation Delay = $144t_1 + 56t_2 + 56t_3 + 8t_4 + 0t_5 + 68t_6 + 8t_7$						Total FU Gates	340

Table 3: Shows the approximate total of logic gates used for each operation and the total amount of logic gates used in the function unit. (t stands for the propagation time related to that specific gate)

Appendix E: Complete Simulation of the Function Unit

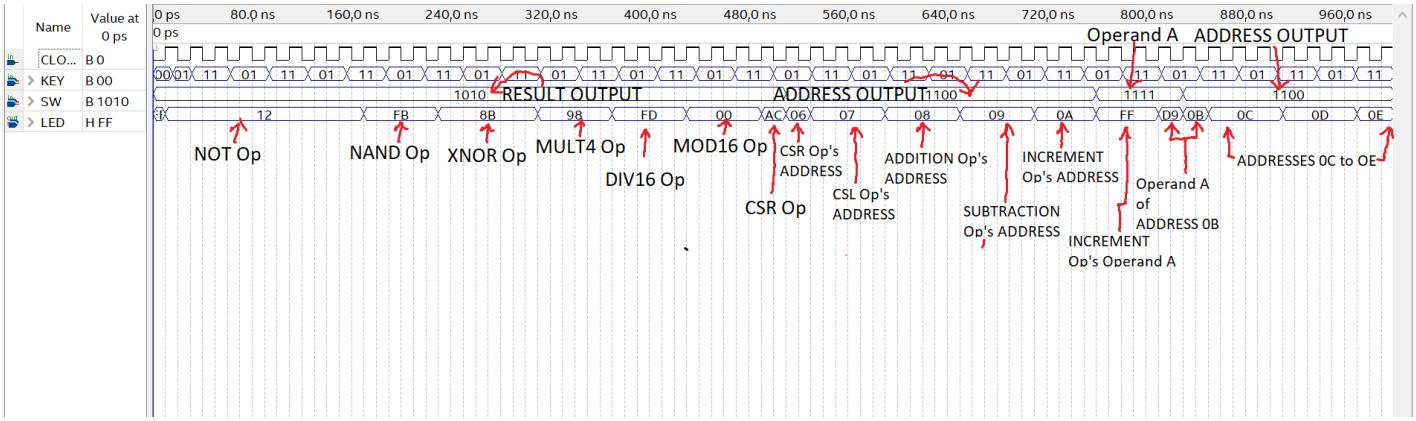

Figure 13: The simulation of the whole function unit while testing the switches and their outputs with my own supplied operands for A and B.

Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board
Validation Sheet – Page 1

The Validation Sheet for Project 2 is two pages long. Make sure that you include both pages as the last two pages of your project report. You should complete page 1 and leave page 2 blank for the TA to complete.

No *GTA* or *student* should discuss any aspect of a student's design with another student. Among other things, this includes the manner in which a student implements specific operations.

Student Name: Jamahl Savage
Last four Digits of your student ID:3855

GTA Validation Instructions

1. Program the FPGA on the DE0 Nano board using the Start button on the programmer window.
2. When the programming has successfully completed, reset the design by pressing and releasing the KEY0 pushbutton.
3. Set the DIP switches to 1001 and record the value of the LEDs as two hex digits in Table 1 on the next page. (The DIP switches are 0 on the side labeled "ON".)
4. Set the DIP switches to 1000 and record the value of the LEDs as two hex digits in Table 1.
5. Compare the four digits from steps 3 and 4 to the last four digits of the student's ID number. **If the four digits do not match the last four digits of the student's ID number on their ID card, STOP THE VALIDATION. DO NOT CONTINUE.**

For the remaining steps of the validation, the values of the switch settings in Gray code order will allow the requested items to be checked in the order in the table more quickly. **All values should be recorded in the table as two hexadecimal digits.** For each address, verify that the opcode, operand A and operand B digits match the value in the corresponding address in the rom.txt file on the student's computer.

6. Fill in the row of Table 2 for the "mult4" operation. Verify that the address is 0.
7. Press and release KEY1.
8. Fill in the row of Table 2 for the "inc" operation. Verify that the address is 1.
9. Press and release KEY1.
10. Fill in the row of Table 2 for the "neg" operation. Verify that the address is 2.
11. Press and release KEY1.
12. Fill in the row of Table 2 for the "nand" operation. Verify that the address is 3.
13. Press and release KEY1.
14. Fill in the row of Table 2 for the "xnor" operation. Verify that the address is 4.
15. Press and release KEY1.
16. Fill in the row of Table 2 for the "not" operation. Verify that the address is 5.

17. Press and release KEY1.
18. Fill in the row of Table 2 for the “csl” operation. Verify that the address is 6.
19. Press and release KEY1.
20. Fill in the row of Table 2 for the “csr” operation. Verify that the address is 7.
21. Press and release KEY1.
22. Fill in the row of Table 2 for the “add” operation. Verify that the address is 8.
23. Press and release KEY1.
24. Fill in the row of Table 2 for the “sub” operation. Verify that the address is 9.
25. Press and release KEY1.
26. Fill in the row of Table 2 for the “div16” operation. Verify that the address is A.
27. Press and release KEY1.
28. Fill in the row of Table 2 for the “mod16” operation. Verify that the address is B.

Design Project 2: Design and Implementation of an Arithmetic Logic Unit on the DE0 Nano board

Validation Sheet – Page 2

Table 1: Checking BCD equivalent to student ID

	Switch setting, SW[3:0]	
	1001	**1000**
LED value in hexadecimal	____ ____	____ ____

Table 2. Checking the operation of the ALU. All values of the LEDs should be recorded as two digit hexadecimal numbers.

SW[3:0] →	1100	1101	1111	1110	1010	1011
operation	**address**	**opcode**	**operandA**	**operandB**	**result**	**status**
mult4	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
inc	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
neg	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
nand	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
xnor	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
not	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
csl	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
csr	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
add	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
sub	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
div16	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___
mod16	___ ___	___ ___	___ ___	___ ___	___ ___	___ ___

Comments (only required if something is unusual or wrong):