ECE 3544: Digital Design I
Project 5 – Datapath and Finite State Machine Design

Student Name: **Jonathan Lemarroy**

Honor Code Pledge: I have neither given nor received unauthorized assistance on this assignment.

Grading: The design project will be graded on a 100 point basis, as shown below:

Manner of Presentation (30 points)

_____ / 5 Completed cover sheet included with report

_____ / 15 Organization: Clear, concise presentation of content; Use of appropriate, well-organized sections

_____ / 10 Mechanics: Spelling and grammar

Technical Merit (70 points)

_____ / 5 General discussion: *Did you describe the objectives in your own words? Did you discuss your other conclusions and the lessons you learned from the assignment?*

_____ / 15 Design discussion: *Did you discuss the approach you took to designing and implementing the modules that make up your system, and how you synthesized the system from its components?*

_____ / 5 State diagrams: *Does your state diagram model a system that performs the required tasks? Connect this discussion to your design discussion*.

_____ / 5 Testing discussion: *What was your approach to formulating your test benches? How did you verify the correctness of the modules you designed?*

_____ / 10 Supporting figures: *Waveforms showing correct operation of the top-level module.*

_____ / 30 Validation of the final design on the DE1-SoC board

_____ **Project Grade**

ECE 3544: Digital Design I
Project 5 Validation Sheet – Page 1

GTA Validation Instructions:
Program the FPGA on the DE1-SoC Board with the student's implementation of the FSM-Datapath. When the programming has successfully completed, perform the tests described below. Apply the values to the switches and/or press the key as indicated, reading from left to right. In general, the first value represents the opcode, the second value represents operand A, and the third value represents operand B. For each step, record the value of HEX[3:0] as the **Result**.

HEX[3:0] should not change its value while the switches are being changed. If they do change, make a note in the comments section. (HEX[3:0] can change while KEY1 or KEY0 is being pressed. That is acceptable and there is no need to make a note of it.)

SW[3:0]		SW[7:4]	SW[3:0]		SW[7:4]	SW[3:0]		Result
							KEY0	____ ____ ____ ____
0000	KEY1	0110	0111	KEY1	1000	1100	KEY1	____ ____ ____ ____
0001	KEY1	0011	0101	KEY1	1110	1000	KEY1	____ ____ ____ ____
0010	KEY1	0001	0101	KEY1	1110	1000	KEY1	____ ____ ____ ____
0011	KEY1	0101	0110	KEY1				____ ____ ____ ____
0100	KEY1	0011	1010	KEY1	0101	0110	KEY1	____ ____ ____ ____
0101	KEY1	0110	1010	KEY1	1100	1001	KEY1	____ ____ ____ ____
0110	KEY1	0011	1100	KEY1	0110	1001	KEY1	____ ____ ____ ____
0111	KEY1	0101	0110	KEY1				____ ____ ____ ____
1000	KEY1	1101	0000	KEY1	0000	1010	KEY1	____ ____ ____ ____
1001	KEY1	0010	0111	KEY1	0000	0100	KEY1	____ ____ ____ ____
1010	KEY1	0011	0101	KEY1	0000	0101	KEY1	
1011	KEY1	1100	0010	KEY1	0000	0011	KEY1	

The validation continues on the next page.

ECE 3544: Digital Design I
Project 5 Validation Sheet – Page 2

SW[3:0]		SW[7:4]	SW[3:0]		SW[7:4]	SW[3:0]		Result
1100	KEY1	0101	1100	KEY1	0011	0111	KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____
							KEY1	____ ____ ____ ____

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

GTA Printed Name

GTA Signature

Purpose:

This project gives experience designing and implementing data-paths coupled with finite state machines that are interactive with each other.

Project Description:

Design and synthesize an ALU that performs the operations found in **Table 1** based on the given operation code. The arithmetic shifts and multiplication operations however cannot use the built in Verilog operators. The input values come from switches SW[7:0] and pushbuttons KEY[1:0]. The opcodes are set using SW[3:0] and operands come from SW[7:0]. KEY[0] is the active low reset while KEY[1] is "Enter" key progressing the ALU to the next state.

Table 1: Datapath Operations

Operation	**Result of the Operation**	**Operation code**
Add[1]	Add A and B. Take A and B as 8-bit signed 2's complement operands, sign-extended to 16 bits.	0000
Subtract B from A[1]	Subtract B from A. Take A and B as 8-bit signed 2's complement operands, sign-extended to 16 bits.	0001
Subtract A from B[1]	Subtract A from B. Take A and B as 8-bit signed 2's complement operands, sign-extended to 16 bits.	0010
Negate[1]	Take the 2's complement of A. Take A as an 8-bit signed 2's complement operand, sign-extended to 16 bits.	0011
AND2	Bitwise-AND the values of A and B.	0100
OR2	Bitwise-OR the values of A and B.	0101
XOR2	Bitwise-XOR the values of A and B.	0110
(1's) Complement2	Bitwise-complement the value of A.	0111
Rotate Right3	Rotate A to the right by B positions. Take B as a 4-bit unsigned operand.	1000
Rotate Left3	Rotate A to the left by B positions. Take B as a 4-bit unsigned operand.	1001
Arithmetic Shift Right4	Arithmetically shift A to the right by B positions. Take B as a 4-bit unsigned operand.	1010
Arithmetic Shift Left4	Arithmetically shift A to the left by B positions. Take B as a 4-bit unsigned operand.	1011
Multiply5	Multiply A and B. Take A as the multiplicand, and B as the multiplier.	1100
Debug6	Display the values of internal signals in your design. You may choose these values as needed.	Any unused opcode

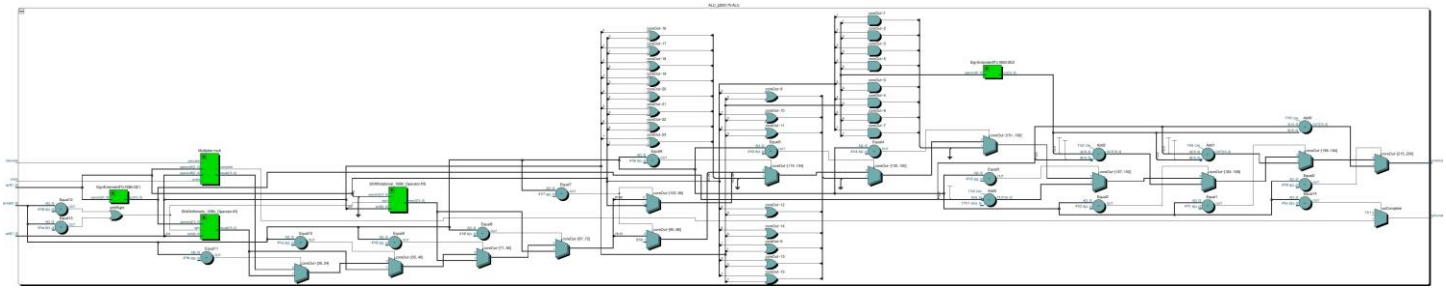
The ALU should operate in the following manner of steps:

0. Between operations, the system is waiting for the user to enter an operation code.
1. Set SW[3:0] to the desired operation code. Press and release Enter.
2. Set SW[7:0] to the value of operand A. Press and release Enter. *For operations requiring only operand A, go to Step 4.*
3. For operations requiring operand B, set SW[7:0] to the value of operand B. Press and release Enter.
4. For all operations except the multiply operation, HEX[3:0] should now display the result. *Return to Step 0.* For the multiply operation, the displays should now show {8'b0, B}. *This is the initial value of the product register in the multiplier described in the lecture notes. Continue to Step 5.*
5. For the multiply operation, press and release Enter eight more times to show the result. After each press and release, HEX[3:0] should show the value the product register. After the eighth press, HEX[3:0] should display the result of the multiply operation. *Return to Step 0.*

Technical Design:

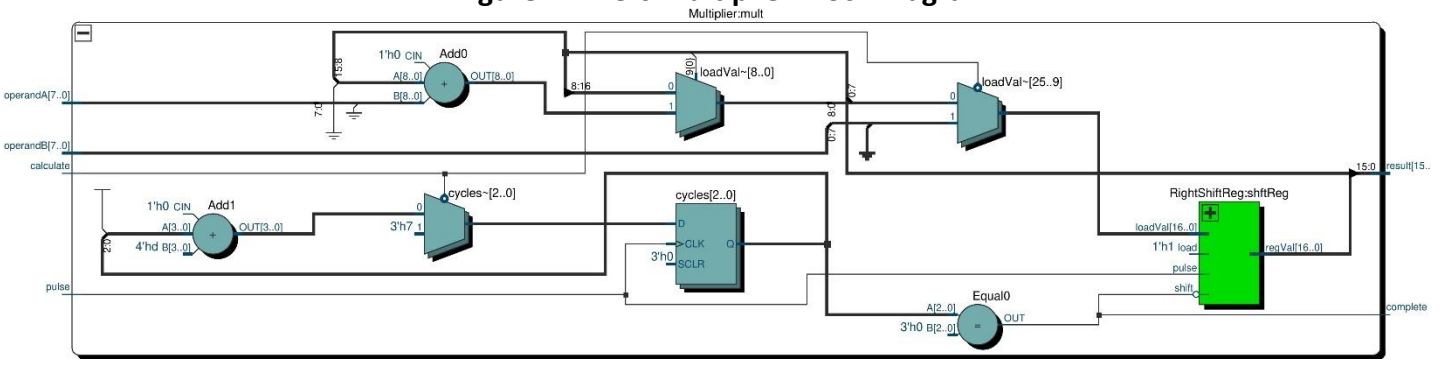
The first module designed was the ALU's core. This was a very easy task since it could be done using the conditional operator along with the allowed Verilog operators to simulate a mux like output that switches based on opcodes. This amounted to the circuit found in **Figure 1** being designed.

Figure 1: ALU Core Block Diagram



Creating the shift operators was as simple as using the concatenation operator in conjunction with the conditional operator. The Multiplier module however was the most challenging part of this project. Using the add and shift sequential design from lecture, it formed the circuit found in **Figure 2**.

Figure 2: ALU's Multiplier Block Diagram



The last step was creating a controller for the ALU to operate in the manner described above which means it needs to operate with the state space as found in **Figure 3**. And using this state diagram to design the circuit it led to the design found in **Figure 4**.

Figure 3: ALU Controller State Diagram

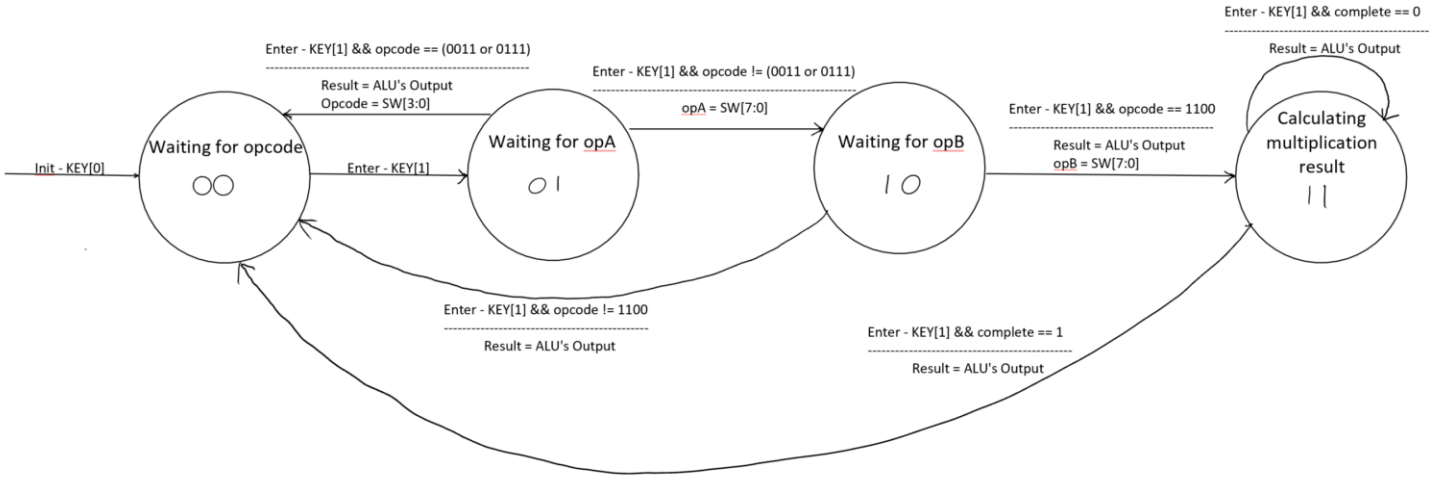
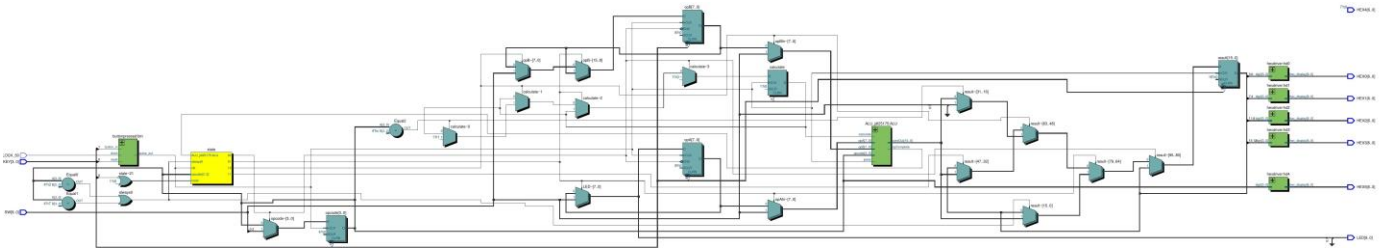
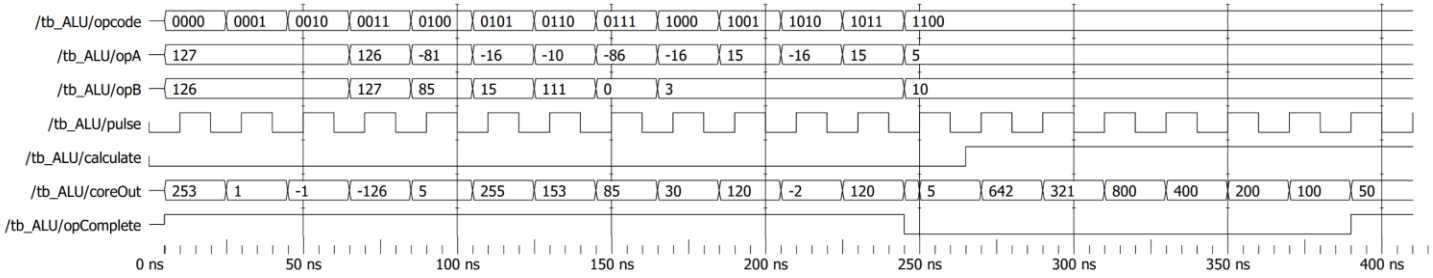


Figure 4: ALU Controller Block Diagram



And after testing this ALU under each opcode it produced the **Waveform 5** below.

Waveform 5: ALU's Sample Testbench



Analysis & Conclusion:

With the current implementation the circuit follows the spec requirements. The one design decision that I would have changed if the spec did not require each iteration of the multiplier be displayed is the calculate bit. Although it currently is there just to inform the module that the operands have been set, I would have instead been set when the operands were modified, and in the event they weren't modified the result would be currently sitting in the shift registers output.