


Arithmetic Logic Unit (ALU) Design and Implementation

By:

Martti Muzyka (0697280) : 

&

Obatosin Obat-Olowu (0680475) : 

Digital VLSI Circuit Design Project (EELE-4054-WA)
Department of Electrical Engineering
Lakehead University
Thunder Bay, Ontario, Canada

Abstract

An arithmetic logic unit, or ALU as it will be referred to from now on, is a digital device that performs arithmetic and logic operations. ALUs are heavily relied upon in the operation and function of computer processors (CPUs). The main objective of this project is to design an ALU program using Verilog hardware description language, that can be implemented on a DE10-Lite FPGA board. This ALU will take two 3-bit inputs (A and B) and depending on a 2-bit control signal (op), provide a 6-bit output (Dout). If the control signal is 2'b00, the ALU will perform an XNOR operation, if the control signal is 2'b01, the ALU will perform a shift-left operation, if the control signal is 2'b10, the ALU will perform an addition operation, and finally, if the control signal is 2'b11, the ALU will perform a multiplication operation. This ALU will also feature an enable signal (ena) and a synchronized reset (rst_n). The values of the inputs, control signal, and enable signal will be determined by the states of the DE10-Lites on-board switches (SW[9:1]), and the reset signal will be determined by the state of one of the on-board buttons (KEY[0]). Furthermore, the ALU output will be displayed on the DE10-Lites 7-segment display, for the XNOR and shift-left operations, the output will be displayed in binary, and for the addition and multiplication operations it will be displayed in decimal.

Table of Contents

Title Page	1
Abstract	2
Table of Contents	3
List of Figures	5
List of Tables	7
Objective	8
Design (1)	8
Design Theory (1.1)	8
Design Specifications (1.2)	9
Design (1.3)	10
-ALU Symbol and Interface Description	11
-Pin Assignment	11
-Block Diagrams	12
-Timing Diagrams	15
Implementation (2)	19
Verilog Files and Design Hierarchy (2.1)	19
Simulation Platform and FPGA Board Information (2.2)	19
Verification (3)	20
Modelsim RTL Simulation Results (3.1)	21
DE10-Lite FPGA Test Results (3.2)	27
Conclusion (4)	29
Appendix A (5) - Verilog Code	31
ALUlogic.v (5.1)	31
Display.v (5.2)	33
Sync.v (5.3)	42
LED.v (5.4)	43
ALU.v (5.5)	43

Top Module Shift-left Operation Test Bench (5.6)	45
Top Module Test Bench (5.7)	46
Appendix B (6) - Summary Reports	48
ALU.fit.summary (6.1)	48
ALU.map.summary (6.2)	49
ALU.sta.summary (6.3)	49
Appendix C (7) - References	50
Appendix D (8) - Index	51

List of Figures

Figure (1) - 3-bit ALU symbol.	11
Figure (2) - ALU top-module block diagram.	13
Figure (3) - ALU LED sub-module block diagram.	13
Figure (4) - ALU sync sub-module block diagram.	14
Figure (5) - ALU display sub-module block diagram.	14
Figure (6) - ALU logic sub-module block diagram.	14
Figure (7) - Timing diagram for ALUlogic.v sub-module.	16
Figure (8) - Timing diagram for Display.v sub-module.	17
Figure (9) - Timing diagram for Sync.v sub-module.	18
Figure (10) - Timing diagram for LED.v sub-module.	18
Figure (11) - ALU module hierarchy.	19
Figure (12) - DE10-Lite FPGA layout and components.	20
Figure (13) - ModelSim RTL simulation results depicting the XNOR operation functionality.	25
Figure (14) - ModelSim RTL simulation results depicting the shift-left operation functionality.	25
Figure (15) - ModelSim RTL simulation results depicting the addition operation functionality.	26
Figure (16) - ModelSim RTL simulation results depicting the multiplication operation functionality.	26

- Figure (17)** - DE10-Lite FPGA board displaying ALU bitwise XNOR operation results. 27
- Figure (18)** - DE10-Lite FPGA board displaying ALU shift-left operation results. 28
- Figure (19)** - DE10-Lite FPGA board displaying ALU addition operation results. 28
- Figure (20)** - DE10-Lite FPGA board displaying ALU multiplication operation results. 29

List of Tables

Table (1) - Tabulated ALU operations and their corresponding control signal values.	8
Table (2) - XNOR truth table.	8
Table (3) - Binary addition operation truth table.	9
Table (4) - DE10-Lite FPGA pin assignment.	11
Table (5) - Block diagram I/O with their corresponding size and signal.	15
Table (6) - Tabulated test bench test vectors, simulated outputs, and expected outputs.	21
Table (7) - DE10-Lite FPGA board operating conditions for photos.	27

Objective

The main objective of this VLSI project is to effectively design and implement an ALU (arithmetic logic unit) on a DE10-Lite FPGA board. This ALU will perform four different operations (XNOR, shift-left, addition, and multiplication) depending on the value of a control signal. There will be two input signals to the ALU (A and B), each will be 3-bits. Furthermore, the control signal will be 2-bits (op), and the output signal will be 6-bits (Dout). Lastly, there will also be an 1-bit enable signal (ena) to either disable or enable the ALU operation and a 1-bit synchronized reset signal (rst_n) to reset the ALU.

Design (1)

- Design Theory (1.1)
- Design Specifications (1.2)
- Design (1.3)

Design Theory (1.1)

There are four different operations that this ALU can perform, each can be selected by providing the corresponding control signal (op) value. The four different operations and their corresponding control signal values can be seen in table (1) below.

Operation	Control signal (op)
Bitwise XNOR ($Dout = A \sim B$)	2'b00
Shift-left ($Dout = \{A, B\} < i$)	2'b01
Addition ($Dout = A + B$)	2'b10
Multiplication ($Dout = A * B$)	2'b11

Table (1) - Tabulated ALU operations and their corresponding control signal values.

As can be seen in table (1) above, if the control signal is 2'b00, the ALU should perform a bitwise XNOR operation. The truth table describing the output of an XNOR operation can be found in table (2) below. An XNOR operation, in essence, returns a 1 if both inputs are the same (i.e. $A[0] = B[0] = 1$), but returns a 0 if both inputs have different values (i.e. $A[0] = 0, B[0] = 1$). The output of a bitwise XNOR operation performed on two 3-bit inputs will be 3-bits.

A (input)	B (input)	Dout (output)
0	0	1
0	1	0
1	0	0
1	1	1

Table (2) - XNOR truth table.

If the control signal is 2'b01, the ALU should perform a shift-left operation. This shift-left operation will continuously shift the concatenated inputs ({A,B}) to the left at a rate of 0.5 seconds, providing enough time for the user to view the change in output on the 7-segment display. The shift-left operation will restart once the reset signal (rst_n) is applied and it will stop and hold its current value if the enable signal goes low (ena). Similarly, if the inputs are changed, the shift-left operation will restart with the new inputs concatenated together. The output of the shift-left operation will be 6-bits as both 3-bit inputs are being concatenated together.

If the control signal is 2'b10, the ALU will perform an addition operation on the inputs. The addition of two 3-bit inputs results in a 4-bit output, a 3-bit summation output (Sum) and a 1-bit carry output (Cout), both of which are concatenated together ({Cout,Sum}). The truth table describing the output of an addition operation can be seen in table (3) below.

A (input)	B (input)	Sum (output)	Cout (output)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table (3) - Binary addition operation truth table.

If the control signal is 2'b11, the ALU will perform a multiplication operation. For this project, the multiplication operation is implemented using an addition/shift-left approach, and the provided output is 6-bits. This unique approach works by selecting one of the inputs, either A or B, as a reference of sorts, the position of 1's in this input determines the position of A in the concatenated output. If B is 3b'010, A will be shifted 1-bit to the left in the 6-bit output and then concatenated with 0's to fill the remaining bits (Dout = {3'b0,A,1'b0}). Furthermore, if B contains multiple 1's, there will be multiple concatenated outputs added together, for each 1 there will be the same number of addition operations. To view this relationship, refer to equation (1) below.

$$\text{Equation (1) : } \text{Dout} = A * B = 3'b010 * 3'b011 = \{2'b0,A,1'b0\} + \{3'b0,A\} = 6'b000110$$

Lastly, this ALU will also possess an enable signal (ena) and a synchronized reset (rst_n). When the enable signal is "on" the ALU will be able to carry out its varying operations and display them on the 7-segment display, but when the enable signal is "off" the ALU will cease operation and display the last output value on the 7-segment display. The reset signal will reset the ALU to 7'b0, when selected.

Design Specifications (1.2)

When designing the ALU for this project, there were certain specifications that needed to be implemented for it to operate as desired. As stated previously, this ALU has two 3-bit inputs

(A and B), and a 6-bit output (Dout), furthermore, the operation that is performed on the inputs and output from the ALU is determined through a 2-bit control signal (op). When the control signal is set to 2'b00, the ALU performs a bitwise XNOR operation. To implement this operation in the Verilog program, the bitwise XNOR operator was utilized (~^). The value returned from this operation was then concatenated with 3'b0 MSB because the output signal is 6-bits. When displaying this output on the 7-segment display, only the first three segments are required to show the complete output in binary, as the result of a 3-bit XNOR operation will always be 3-bits. A requirement of this project is to display the output of the XNOR operation in binary on the 7-segment display.

When the control signal is set to 2'b01, the ALU performs a shift-left operation on the concatenated inputs ({A,B}). To implement this operation in the Verilog program, the shift-left operator (<<) was employed, with an integer (i) on the right determining how many bits to shift the inputs over. Furthermore, as long as the control signal is set to 2'b01 there is a 24-bit variable (toggle) which is incremented by 1 for every positive clock pulse. Once, toggle reaches 12500000 (DEC), the concatenated inputs are shifted over to the left, it takes 0.5 seconds for toggle to reach this value. The amount of bits that the inputs are shifted over depends on the value of the integer "i", "i" is initialized to 0 at the start of the program, whenever the inputs change, and whenever the control signal changes, but it is incremented by 1 everytime toggle reaches 12500000 (DEC). Therefore, if the control signal remains set to 2'b01, the concatenated inputs will be visibly shifted over to the left on the 7-segment display. If the reset signal (rst_n) is selected then the shift-left operation will restart, and if the enable signal goes low then the program will stop shifting the bits and the 7-segment display will simply show the last received output. Also, if the inputs are adjusted, the shift-left operation will restart with those new inputs concatenated together. When displaying the shift-left operation on the 7-segment display, all six segments are required as the output is 6-bits. A requirement of this project is to display the output of the shift-left operation in binary on the 7-segment display.

When the control signal is set to 2'b10, the ALU will perform an addition operation. To implement this operation a 3-bit summation (Sum) variable and a 1-bit carryout (Cout) variable were initialized to store the value of the added inputs (A and B). When concatenated together ({Cout,Sum}), Sum and Cout can store up to 4-bits, which is required when adding two 3-bit values. Once this operation was complete, the output was then set equal to Cout and Sum concatenated together with 2'b0 MSB, providing a 6-bit output. A requirement of this project is to display the output of the addition operation in decimal on the 7-segment display. Therefore, only the first two segments of the 7-segment display are needed as the maximum output value of the addition of two 3-bit numbers is 14 (DEC).

Lastly, when the control signal is set to 2'b11 the ALU performs a multiplication operation on the inputs. This multiplication operation was implemented using the addition/shift-left approach covered in section 1.1 (design theory) of the report. A requirement of this project is to display the output of the multiplication operation in decimal on the 7-segment display. Seeing as the maximum output value of the multiplication of two 3-bit numbers is 49 (DEC), only the first two segments of the 7-segment display are required to display the output.

Design (1.3)

ALU Symbol and Interface Description:

Refer to figure (1) below for the ALU symbol displaying the input and output requirements. The inputs, A and B, are 3-bit binary numbers, their values will be determined by the state of 6 of the DE10-Lites 10 physical on-board switches. Switches [6:4] will be used for B, and switches [3:1] for A. Furthermore, the control signal (op) is another input, that is a 2-bit binary number. Similarly, to the inputs A and B, the value of the control signal will also be determined by the state of two on-board switches, switches [8:7]. Lastly, the enable signal is a 1-bit binary number, its value will be determined by the state of switch [9]. Switch [0] will remain unused for this project. When the switches are activated they will provide a 1 to their corresponding signal, if they are deactivated they will provide a 0.

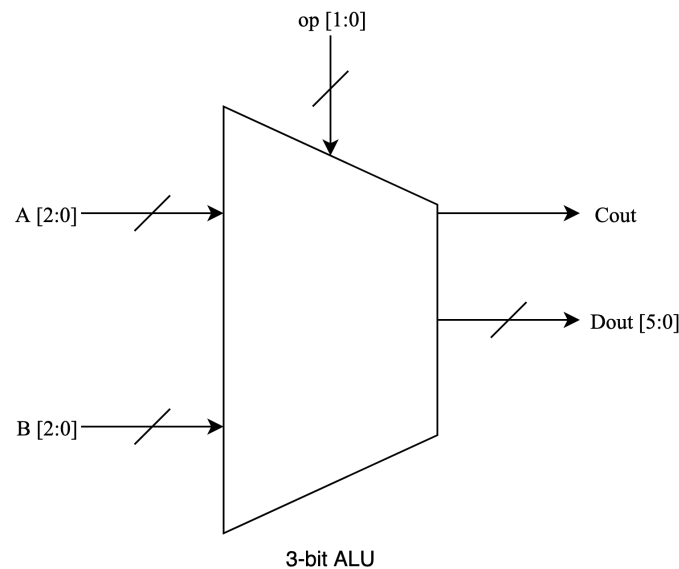


Figure (1) - 3-bit ALU symbol.

Pin assignment:

Refer to table (4) below for the pin assignments of the DE10-Lite FPGA signals utilized in the ALU program. This table also includes a brief description of the functionality of the pins and their corresponding I/O standards.

Signal Name	FPGA Pin No.	Description	I/O Standard
MAX10_CLK1_50	PIN_P11	50 MHz clock input	3.3-V LVTTL
KEY0	PIN_B8	Push-button [0]	3.3 V SCHMITT TRIGGER
SW0	PIN_C10	Slide Switch [0]	3.3-V LVTTL
SW1	PIN_C11	Slide Switch [1]	3.3-V LVTTL
SW2	PIN_D12	Slide Switch [2]	3.3-V LVTTL

SW3	PIN_C12	Slide Switch [3]	3.3-V LVTTL
SW4	PIN_A12	Slide Switch [4]	3.3-V LVTTL
SW5	PIN_B12	Slide Switch [5]	3.3-V LVTTL
SW6	PIN_A13	Slide Switch [6]	3.3-V LVTTL
SW7	PIN_A14	Slide Switch [7]	3.3-V LVTTL
SW8	PIN_B14	Slide Switch [8]	3.3-V LVTTL
SW9	PIN_F15	Slide Switch [9]	3.3-V LVTTL
LEDR0	PIN_A8	LED [0]	3.3-V LVTTL
LEDR1	PIN_A9	LED [1]	3.3-V LVTTL
LEDR2	PIN_A10	LED [2]	3.3-V LVTTL
LEDR3	PIN_B10	LED [3]	3.3-V LVTTL
LEDR4	PIN_D13	LED [4]	3.3-V LVTTL
LEDR5	PIN_C13	LED [5]	3.3-V LVTTL
LEDR6	PIN_E14	LED [6]	3.3-V LVTTL
LEDR7	PIN_D14	LED [7]	3.3-V LVTTL
LEDR8	PIN_A11	LED [8]	3.3-V LVTTL
LEDR9	PIN_B11	LED [9]	3.3-V LVTTL
HEX0	PIN_C14	Seven Segment Digit [0]	3.3-V LVTTL
HEX1	PIN_E15	Seven Segment Digit [1]	3.3-V LVTTL
HEX2	PIN_C15	Seven Segment Digit [2]	3.3-V LVTTL
HEX3	PIN_C16	Seven Segment Digit [3]	3.3-V LVTTL
HEX4	PIN_E16	Seven Segment Digit [4]	3.3-V LVTTL
HEX5	PIN_D17	Seven Segment Digit [5]	3.3-V LVTTL
HEX6	PIN_C17	Seven Segment Digit [6]	3.3-V LVTTL

Table (4) - DE10-Lite FPGA pin assignment.

Block diagrams:

Figure (2) below depicts the block diagram for the top-module ALU.v Verilog program. Figures (3-6) below depict the block diagrams for each of the four different sub-modules, LED.v, Sync.v, Display.v, and ALUlogic.v. Refer to table (5) below for a legend showing the

inputs/outputs of the block diagrams below, and their corresponding DE10-Lite FPGA signal names.

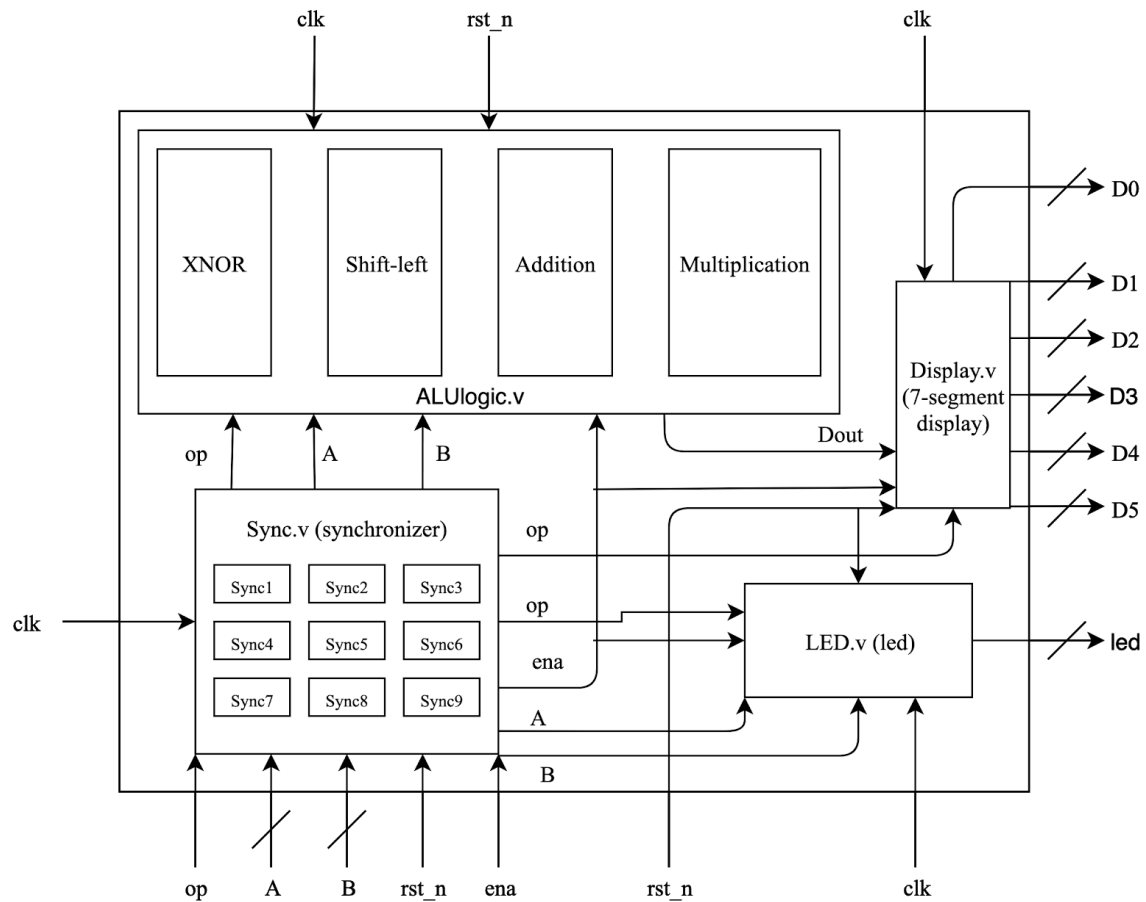


Figure (2) - ALU top-module block diagram.

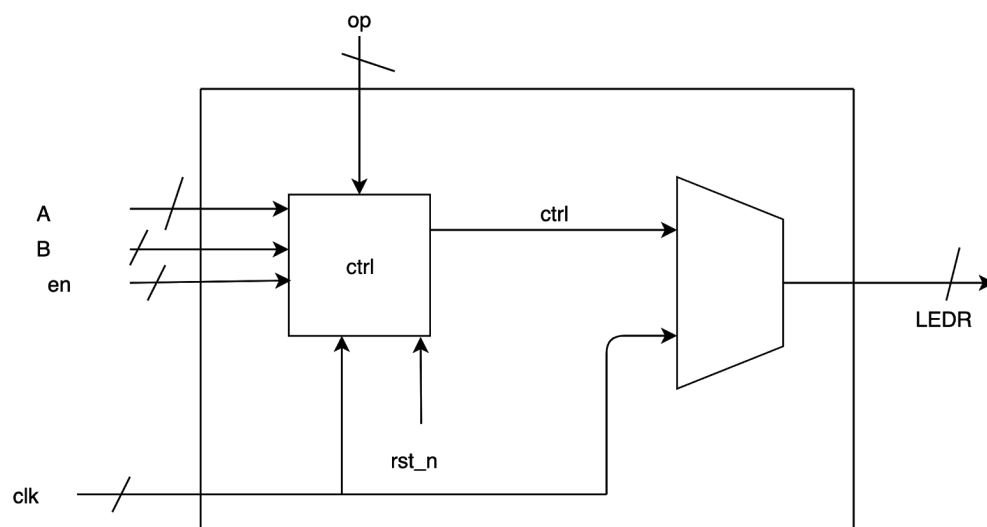


Figure (3) - ALU LED.v sub-module block diagram.

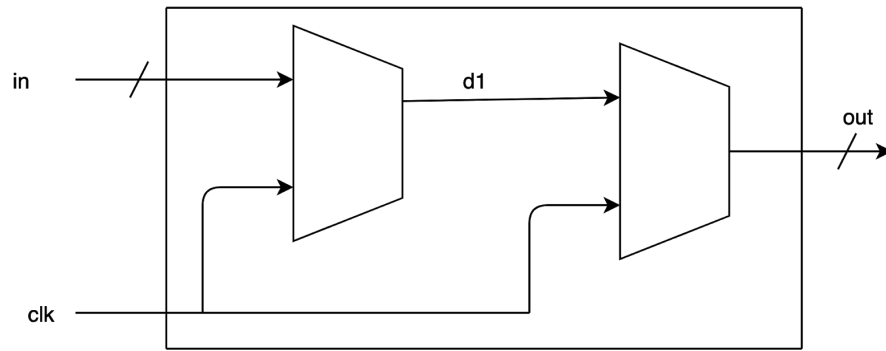


Figure (4) - ALU sync.v sub-module block diagram.

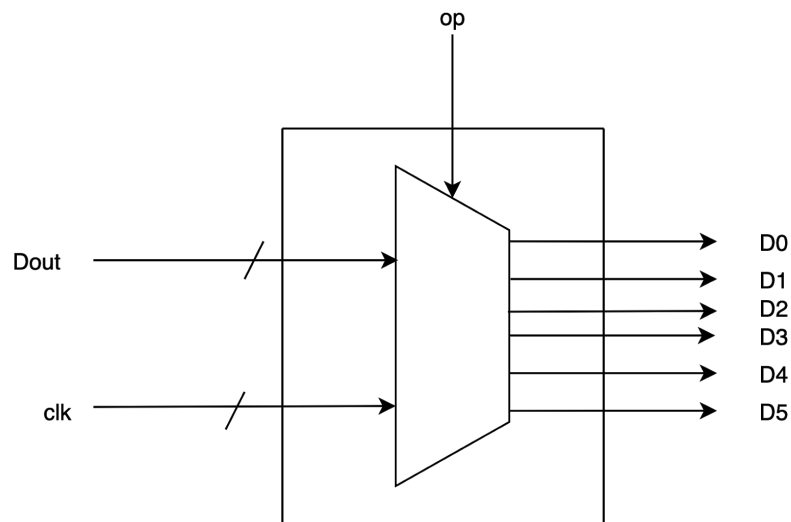


Figure (5) - ALU display.v sub-module block diagram.

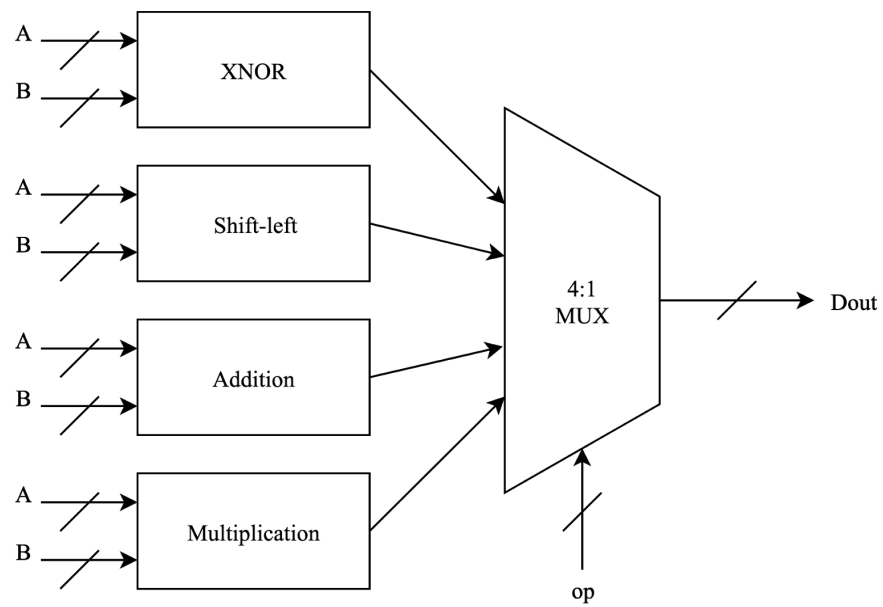


Figure (6) - ALUlogic.v sub-module block diagram.

Input/Output	Signal Name	Size (bits)
clk	MAX10_CLK1_50	1
rst_n	KEY[0]	1
A	SW[3:1]	3
B	SW[6:4]	3
op	SW[8:7]	2
ena	SW[9]	1
led	LEDR[9:0]	10
D0	HEX0[7:0]	8
D1	HEX1[7:0]	8
D2	HEX2[7:0]	8
D3	HEX3[7:0]	8
D4	HEX4[7:0]	8
D5	HEX5[7:0]	8

Table (5) - Block diagram I/O with their corresponding size and signal.

Timing diagrams:

Refer to figure (7) below for an image of the timing diagram associated with the ALUlogic.v sub-module. The control signal (op) begins set to 2'b00 and is adjusted by 1-bit every 200 ns, until reaching 2'b11. The inputs (A and B) are initialized and adjusted to various values as to show the changing output (Dout). Every 200 ns, the enable signal (ena) is set low, when this occurs, the output does not change regardless of the inputs. Similarly, every 200 ns, the reset signal (rst_n) is set high momentarily, when this occurs, the output is set to 6'b0. For theory pertaining to this specific sub-module and its operation, see section 1.1 (Design Theory).

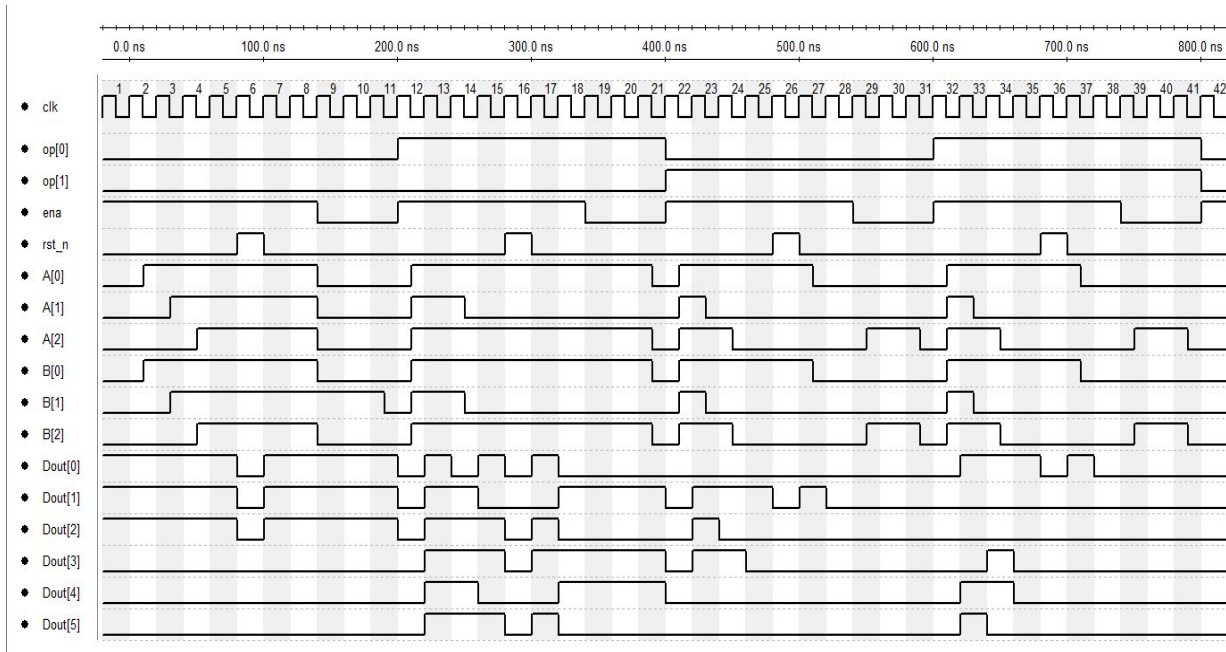


Figure (7) - Timing diagram for ALUlogic.v sub-module.

Refer to figure (8) below for an image of the timing diagram associated with the Display.v sub-module. The control signal begins set to 2'b10 and at the 200 ns mark, is increased to 2'b11, as per the design specifications, when the control signal is set to these two values, the output will be displayed in decimal on the first two 7-segment displays. The enable signal is set to 1 indefinitely and the reset signal is set to 0 indefinitely. The output signal is initially set to 14 (DEC) and increased to 49 (DEC) at the 200 ns mark. When the output signal is set to 14 (DEC) the first 7-segment display signal (D0) is set to 8'b10011001 which displays a "4" on the 7-segment display, similarly, the second 7-segment display signal (D1) is set to 8'b11111001 as to display a "1" on the 7-segment display. When the output signal changes to 49 (DEC), the 7-segment display outputs change to 8'b10011000 (D0) and 8'b10011001 (D1) respectively.

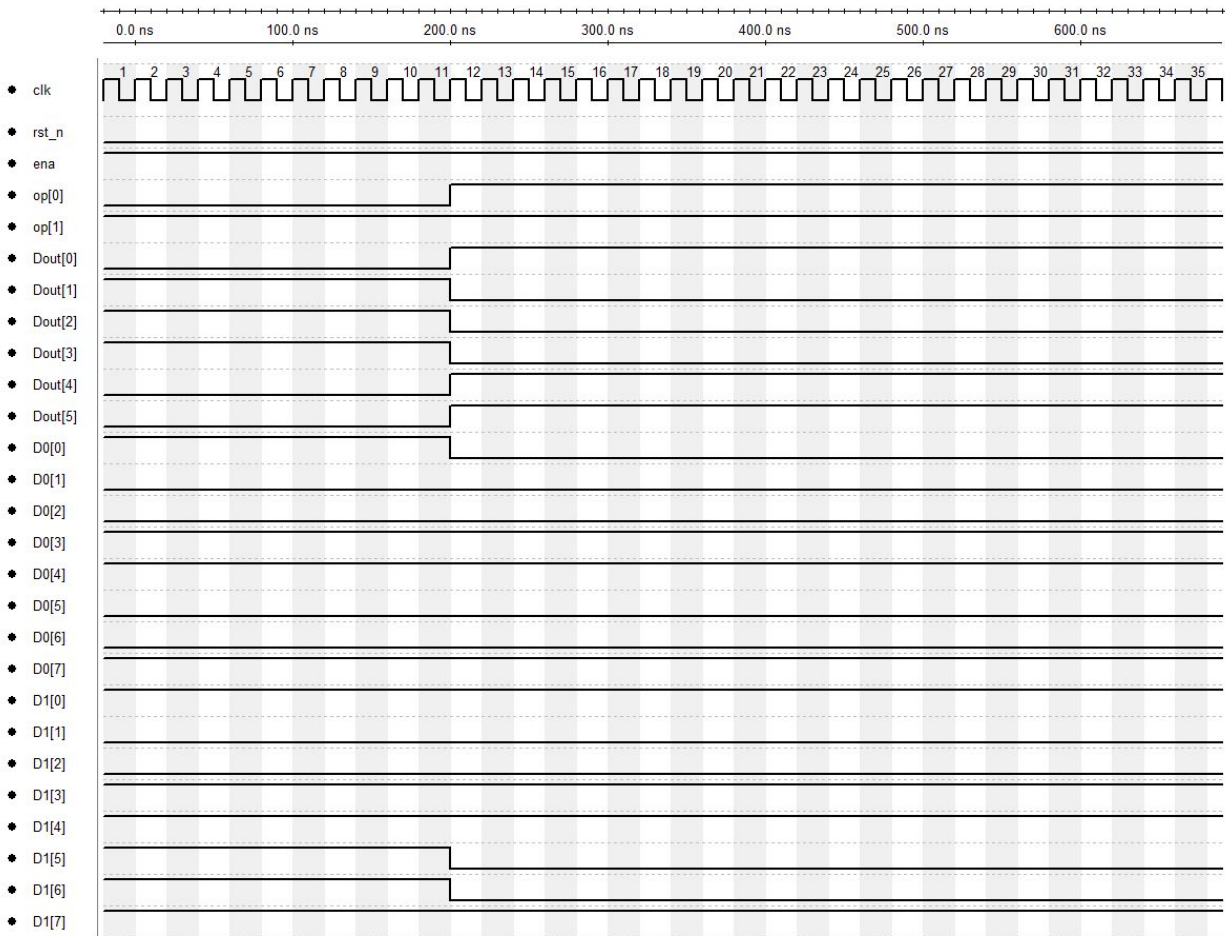


Figure (8) - Timing diagram for Display.v sub-module.

The sync.v sub-module models two D flip-flops (DFF) and is used to synchronize the inputs and outputs with respect to the DE10-Lites 50MHz clock. Each of the DE10-Lites on-board switches and buttons are input into this module. This means that if a switch changes state when the clock is low, the sync.v sub-module will store the value until the positive edge of the clock, where the output is then set equal to the input. This can be viewed in figure (9) below which depicts the timing diagram associated with the sync.v sub-module. So long as the reset signal is low, the output (out) will equal the output of the first DFF (d1), which is equal to the input (in). However, if the reset signal is set high, both DFF outputs are set low indefinitely, regardless of the input.

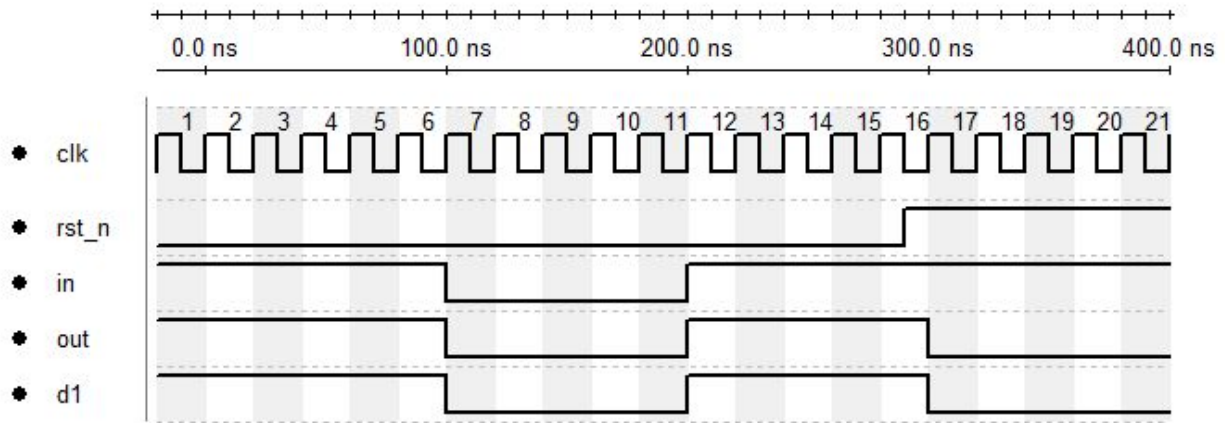


Figure (9) - Timing diagram for Sync.v sub-module.

Refer to figure (10) below for an image of the timing diagram associated with the LED.v sub-module. The values of the control signal (op), the enable signal (ena) and the inputs (A and B) are determined through physical switches located on the DE10-Lite FPGA board. The role of the LED.v sub-module is to illuminate the LEDs located above each of the switches when the switches are high, and to turn them off when the switches are low. Furthermore, if the reset signal is set high the LEDs are turned off, regardless of the switch states. The enable (ena) signal corresponds to led [9], the control signal (op) to led [8:7], and the inputs A and B correspond to led [1:3] and led [4:6] respectively. Therefore, if the enable signal is high, led [9] will also be high, and the DE10-Lite LED above the enable switch will illuminate.

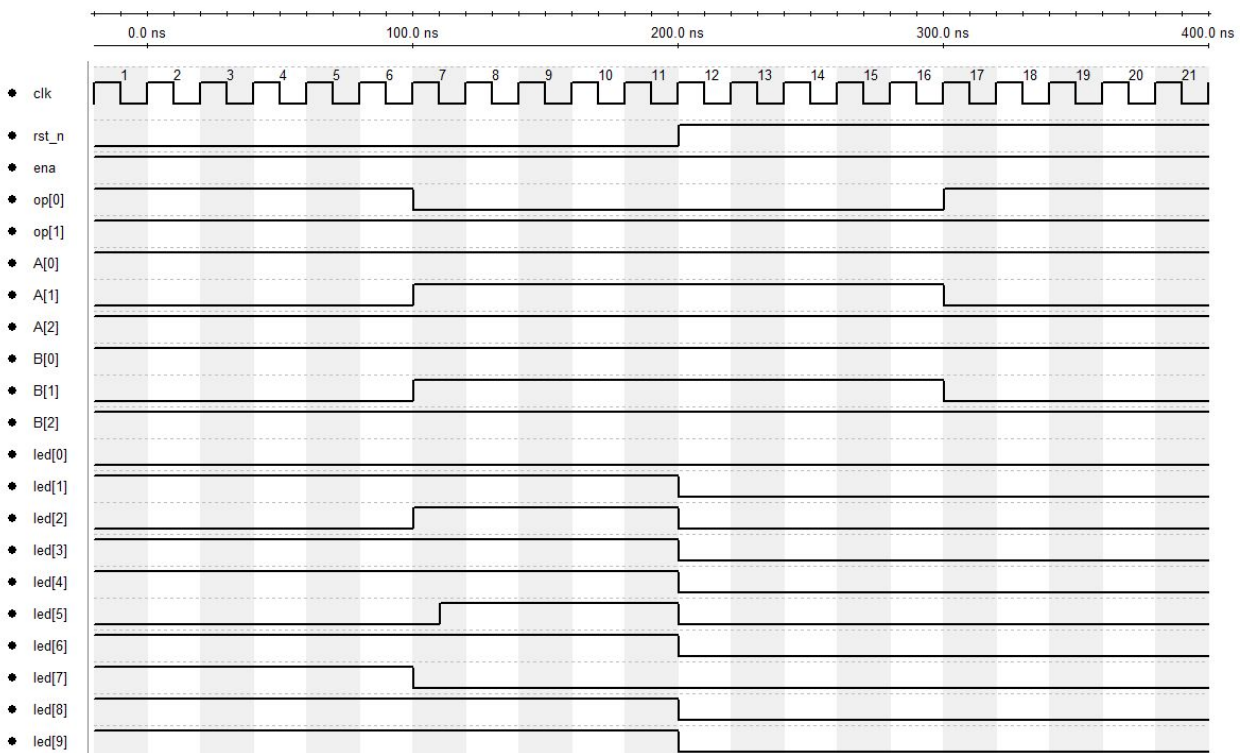


Figure (10) - Timing diagram for LED.v sub-module.

Implementation (2)

- Verilog Files and Design Hierarchy (2.1)
- Simulation Platform and FPGA Board Information (2.2)

Verilog Files and Design Hierarchy (2.1)

The designed ALU Verilog program consists of four sub-modules (ALUlogic.v, Display.v, Sync.v, LED.v) and one top-module (ALU.v), they can be seen listed below, along with a brief summary of their functionality. The modules themselves can be seen in Appendix A. The module hierarchy can be seen in figure (11) below as well.

- ALUlogic.v: A sub-module responsible for carrying out the arithmetic and logic operations of the ALU program depending on the value of the control signal (op).
- Display.v: A sub-module responsible for displaying the output (Dout) on the DE10-Lite's 7-segment display.
- Sync.v: A sub-module responsible for synchronizing the inputs and outputs with respect to the DE10-Lite's on-board 50MHz clock.
- LED.v: A sub-module responsible for activating and deactivating the DE10-Lite's on-board LEDs.
- ALU.v: The top-module that connects the various sub-modules and initializes the inputs and outputs as physical I/O.

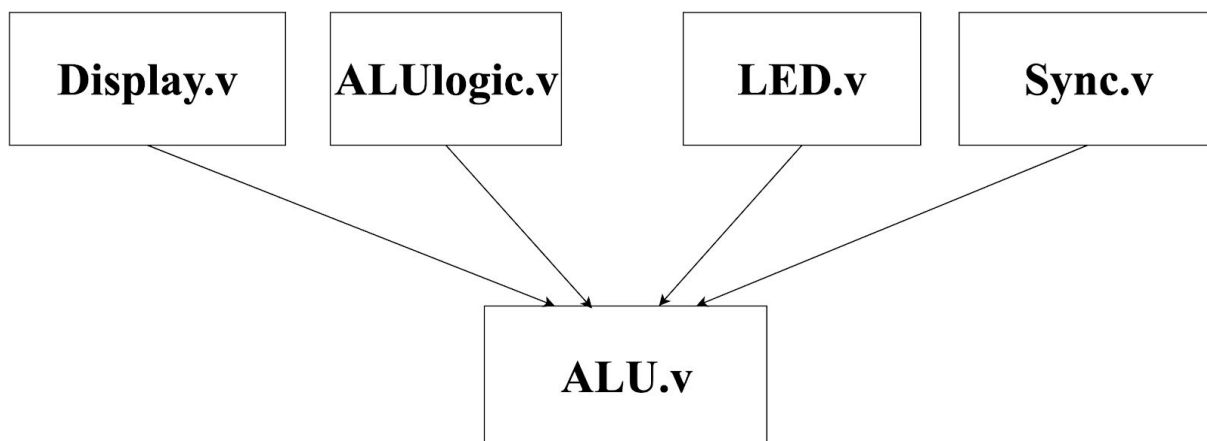


Figure (11) - ALU module hierarchy.

Simulation Platform and FPGA Board Information (2.2)

ModelSim HDL simulator which is a part of the Quartus Prime software is used to verify the design modules using RTL simulations. RTL simulations are done on each module using a testbench with sample input values to test the functionality of the unit under test. All modules are compiled and debugged for errors with the aid of the transcript provided. The simulation results are shown in section 3 (verification) of the report.

After the design has been verified and proven free of any errors, all the necessary files needed to proceed with a Quartus project can be generated using the DE10-Lite System Builder, an assistant tool that automatically creates the files required to start implementing the design on to the FPGA board.

Quartus Prime generates every file required to implement a design onto an FPGA board, it also provides some other tools that aid in optimal design implementation such as power mapping, timing analyzers and place and route fitters. Once a design is compiled, it can then be implemented onto the board through the programmer.

The FPGA board used, the Terasic DE10-Lite, is a cost effective Altera MAX 10 based FPGA board. It includes a 3 Clocks, 10 switches, 10 LEDs, an SDRAM and six 7-segment displays. Logic functions can be implemented to the board using the above software. FPGA dedicates specific blocks to each logic function making it process more efficiently as it only performs one task at a time. Refer to figure (12) below for a top-down view of the DE10-Lite FPGA board and its various I/O.

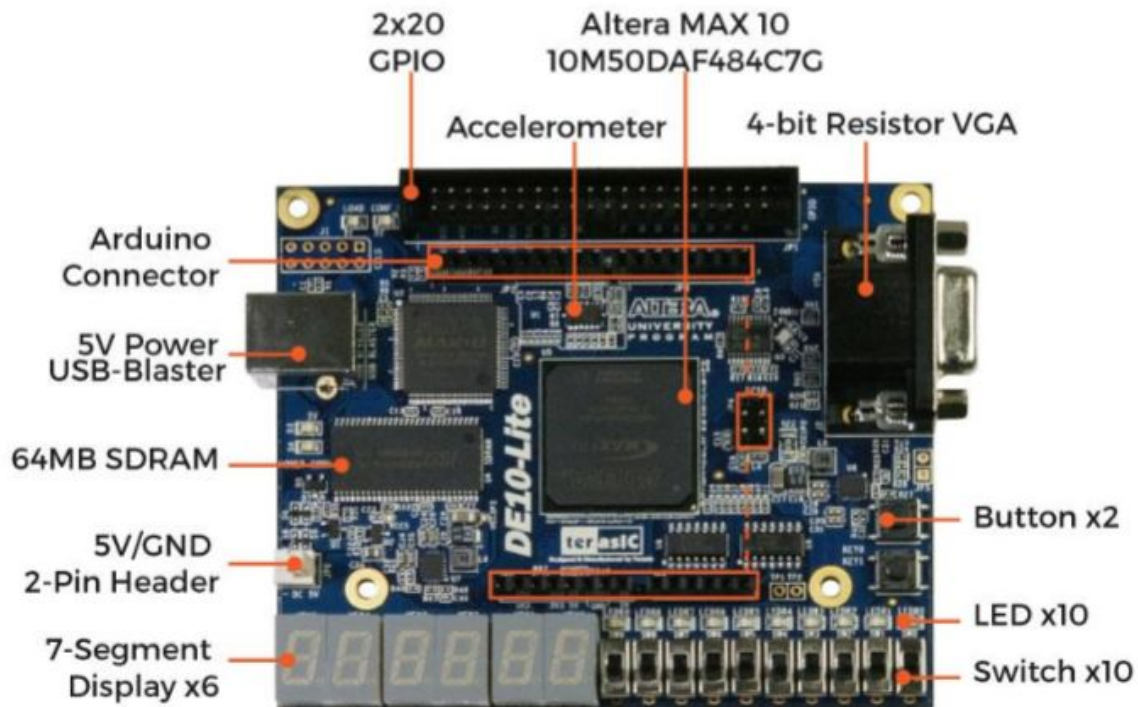


Figure (12) - DE10-Lite FPGA layout and components.

Verification (3)

- Modelsim RTL Simulation Results (3.1)
- DE10-Lite FPGA Test Results (3.2)

Modelsim RTL Simulation Results (3.1)

Figures (13-16) below depict Modelsim RTL simulations performed on the top-module verilog file using the two test benches attached in Appendix A (5). In the test bench used to simulate the multiplication, addition and XNOR operations, the control signal (op) was adjusted between 2'b00, 2'b10, and 2'b11. Each operation was tested individually for simplicity, which is why the switch (SW[9:1]) values that would cause the ALU to perform multiplication and addition operations are temporarily commented out. A separate test bench was used for the shift-left operation as its best to test its functionality with consistent input values. This is done to ensure that the concatenated inputs are effectively being shifted to the left all the way. To test the shift-left operation the 0.5 second delay, used to allow the user to view the output on the 7-segment display, was removed. Furthermore, the reset signal on the RTL simulations behaves opposite as it would in real world situations. In reality, the reset signal is initialized to KEY[0] on the DE10-Lite board. KEY[0] is a button that signals the reset when its pushed down (high). Therefore, when KEY[0] is pushed the ALU resets, and when it remains stationary (low) the ALU performs its regular operations. The opposite is visible in the RTL simulations, when the reset signal (rst_n) goes low, the ALU resets, and when the reset signal is high, the ALU continues to operate. This is a minor inconvenience though, as the program functions as designed when tested on the DE10-Lite FPGA board. The unknown values (red lines) at the beginning of the RTL simulations can be contributed to the fact that these variables are not initialized in the program, they are dependent on other inputs/outputs. This can be ignored as it will not occur in real world situations. To elaborate, the inputs to the ALU program are determined through the states of switches, which will either be high or low, not unknown. The test vectors and their corresponding outputs associated with the Modelsim RTL simulations seen in figures (13-16) below, can be seen in table (6) below. Note that the following RTL simulations were performed with the output (Dout) initialized to 7-bits, not 6, this was done to ensure that the output remained in the 6-bit range and that the arithmetic and logic operations were providing the proper outputs.

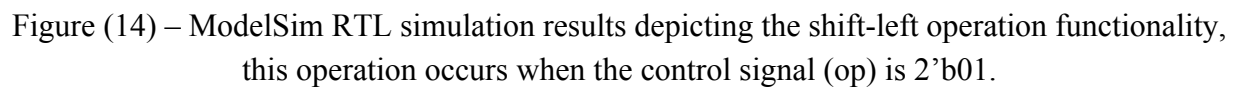
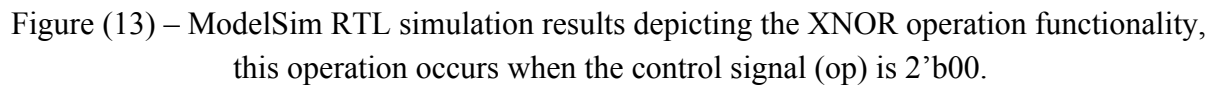
Control Signal (op)	Test Vectors	Expected Output (Dout)	Simulated Output (Dout)	Corresponding RTL Simulation Figure
SW[8:7] = 2'b00 (XNOR)	SW[3:1] = 3'b000 SW[6:4] = 3'b000 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000111	6'b000111	Figure (8)
	SW[3:1] = 3'b000 SW[6:4] = 3'b001 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000110	6'b000110	

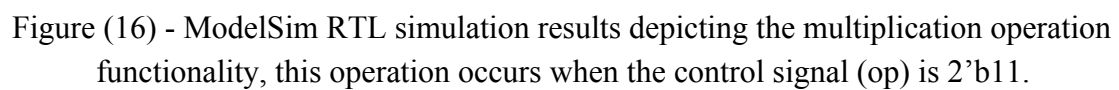
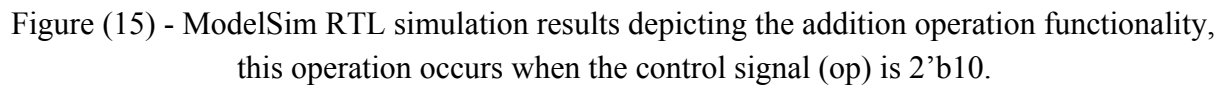
	SW[3:1] = 3'b000 SW[6:4] = 3'b010 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000101	6'b000101	
	SW[3:1] = 3'b000 SW[6:4] = 3'b011 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000100	6'b000100	
	SW[3:1] = 3'b000 SW[6:4] = 3'b100 KEY[0] = 1'b0 SW[9] = 1'b1	6'b000000	6'b000000	
	SW[3:1] = 3'b000 SW[6:4] = 3'b101 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000010	6'b000010	
	SW[3:1] = 3'b000 SW[6:4] = 3'b110 KEY[0] = 1'b1 SW[9] = 1'b0	6'b000010	6'b000010	
	SW[3:1] = 3'b000 SW[6:4] = 3'b111 KEY[0] = 1'b1 SW[9] = 1'b0	6'b000010	6'b000010	
SW[8:7] = 2'b10 (Addition)	SW[3:1] = 3'b000 SW[6:4] = 3'b000 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000000	6'b000000	Figure (10)
	SW[3:1] = 3'b000 SW[6:4] = 3'b001 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000001	6'b000001	
	SW[3:1] = 3'b000 SW[6:4] = 3'b010 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000010	6'b000010	
	SW[3:1] = 3'b000 SW[6:4] = 3'b011 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000011	6'b000011	
	SW[3:1] = 3'b000 SW[6:4] = 3'b100 KEY[0] = 1'b0	6'b000000	6'b000000	

	SW[9] = 1'b1			
	SW[3:1] = 3'b000 SW[6:4] = 3'b101 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000101	6'b000101	
	SW[3:1] = 3'b000 SW[6:4] = 3'b110 KEY[0] = 1'b1 SW[9] = 1'b0	6'b000101	6'b000101	
	SW[3:1] = 3'b000 SW[6:4] = 3'b111 KEY[0] = 1'b1 SW[9] = 1'b0	6'b000101	6'b000101	
SW[8:7] = 2'b11 (Multiplication)	SW[3:1] = 3'b010 SW[6:4] = 3'b000 KEY[0] = 1'b SW[9] = 1'b	6'b000000	6'b000000	Figure (11)
	SW[3:1] = 3'b010 SW[6:4] = 3'b001 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000010	6'b000010	
	SW[3:1] = 3'b010 SW[6:4] = 3'b010 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000100	6'b000100	
	SW[3:1] = 3'b010 SW[6:4] = 3'b011 KEY[0] = 1'b1 SW[9] = 1'b1	6'b000110	6'b000110	
	SW[3:1] = 3'b010 SW[6:4] = 3'b100 KEY[0] = 1'b0 SW[9] = 1'b1	6'b000000	6'b000000	
	SW[3:1] = 3'b010 SW[6:4] = 3'b101 KEY[0] = 1'b1 SW[9] = 1'b1	6'b001010	6'b001010	
	SW[3:1] = 3'b010 SW[6:4] = 3'b110 KEY[0] = 1'b1 SW[9] = 1'b0	6'b001010	6'b001010	
	SW[3:1] = 3'b010	6'b001010	6'b001010	

	SW[6:4] = 3'b111 KEY[0] = 1'b1 SW[9] = 1'b0			
SW[8:7] = 2'b01 (Shift-left)	SW[3:1] = 3'b101 SW[6:4] = 3'b101 KEY[0] = 1'b1 SW[9] = 1'b1	6'b101101 6'b011010 6'b110100 6'b101000 6'b010000	6'b101101 6'b011010 6'b110100 6'b101000 6'b010000	Figure (9)
	SW[3:1] = 3'b011 SW[6:4] = 3'b010 KEY[0] = 1'b1 SW[9] = 1'b1	6'b011010 6'b110100	6'b011010 6'b110100	
	SW[3:1] = 3'b011 SW[6:4] = 3'b010 KEY[0] = 1'b0 SW[9] = 1'b1	6'b011010	6'b011010	
	SW[3:1] = 3'b011 SW[6:4] = 3'b010 KEY[0] = 1'b1 SW[9] = 1'b1	6'b110100	6'b110100	
	SW[3:1] = 3'b011 SW[6:4] = 3'b010 KEY[0] = 1'b1 SW[9] = 1'b0	6'b110100 6'b110100	6'b110100 6'b110100	

Table (6) - Tabulated test bench test vectors, simulated outputs, and expected outputs.





DE10-Lite FPGA Test Results (3.2)

Refer to figures (17-20) below for images of the DE10-Lite FPGA board displaying the outputs of each of the four ALU operations on the 7-segment display. For information regarding the operating conditions of the DE10-Lite FPGA board when these photos were taken, refer to table (7) below.

Figure (#)	SW[9:1] Values	Comments
Figure (12), XNOR	9'b100111111	N/A
Figure (13), Shift-left	9'b001111111	The enable signal (SW[9]) was set low after 3-bits had been shifted to the left.
Figure (14), Addition	9'b110111111	N/A
Figure (15), Multiplication	9'b111111111	N/A

Table (7) - DE10-Lite FPGA board operating conditions for photos.

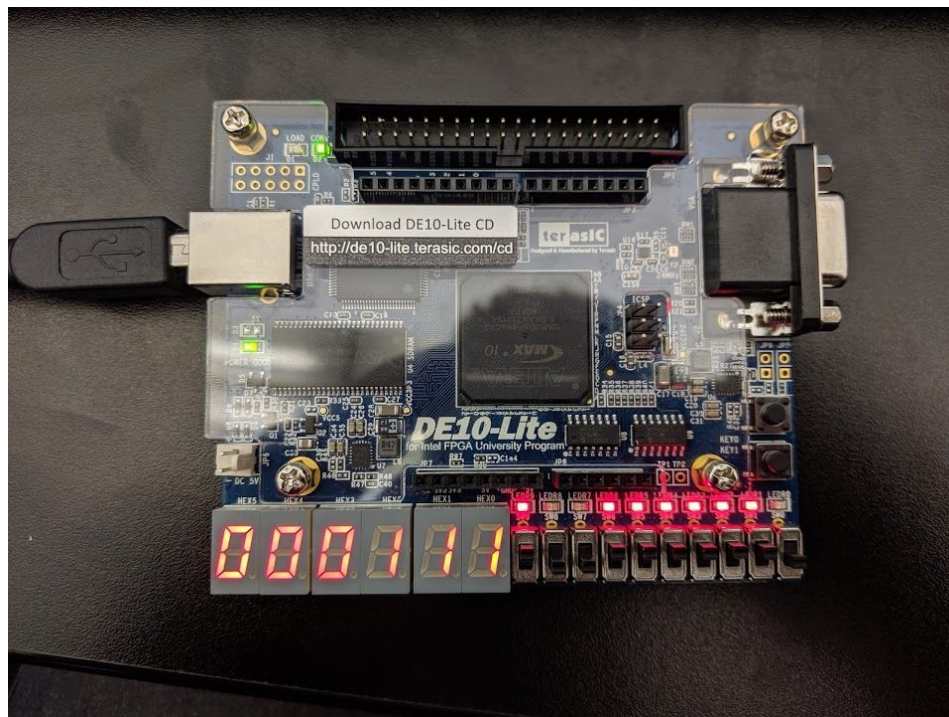


Figure (17) - DE10-Lite FPGA board displaying ALU bitwise XNOR operation results.

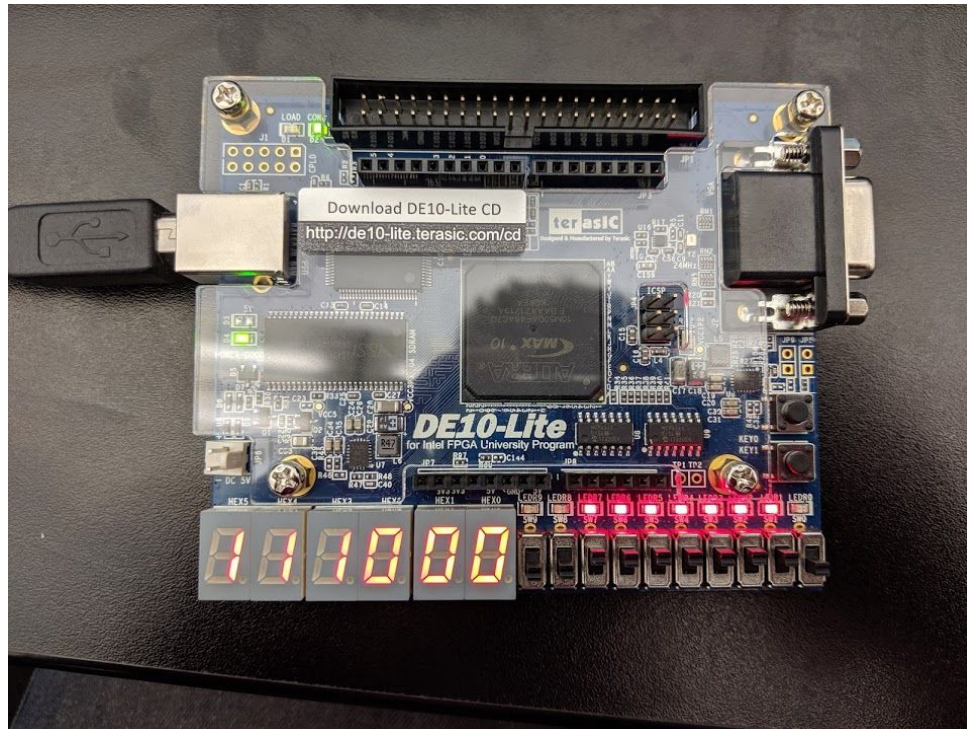


Figure (18) - DE10-Lite FPGA board displaying ALU shift-left operation results.

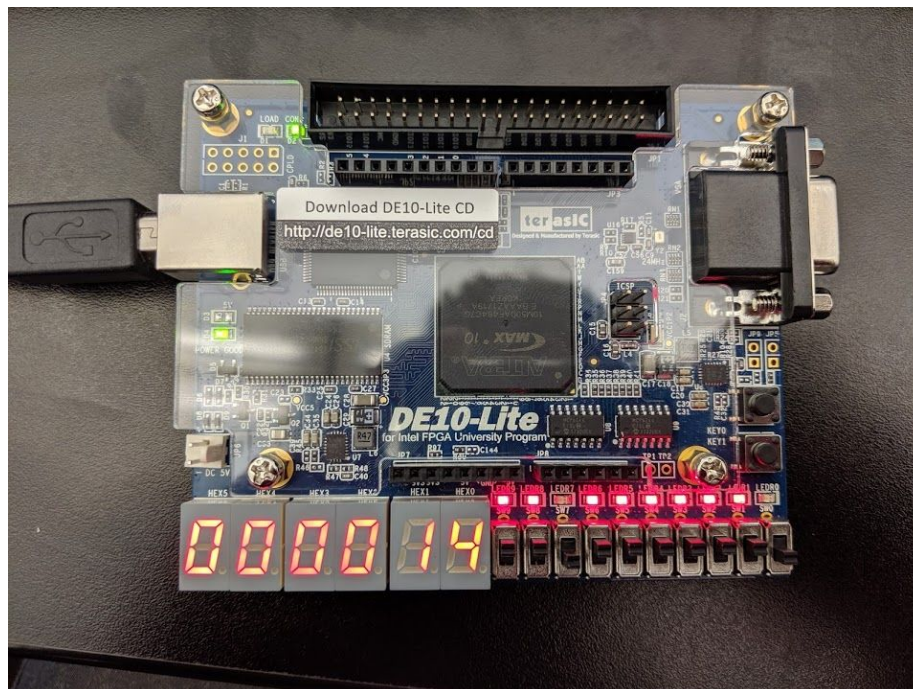


Figure (19) - DE10-Lite FPGA board displaying ALU addition operation results.

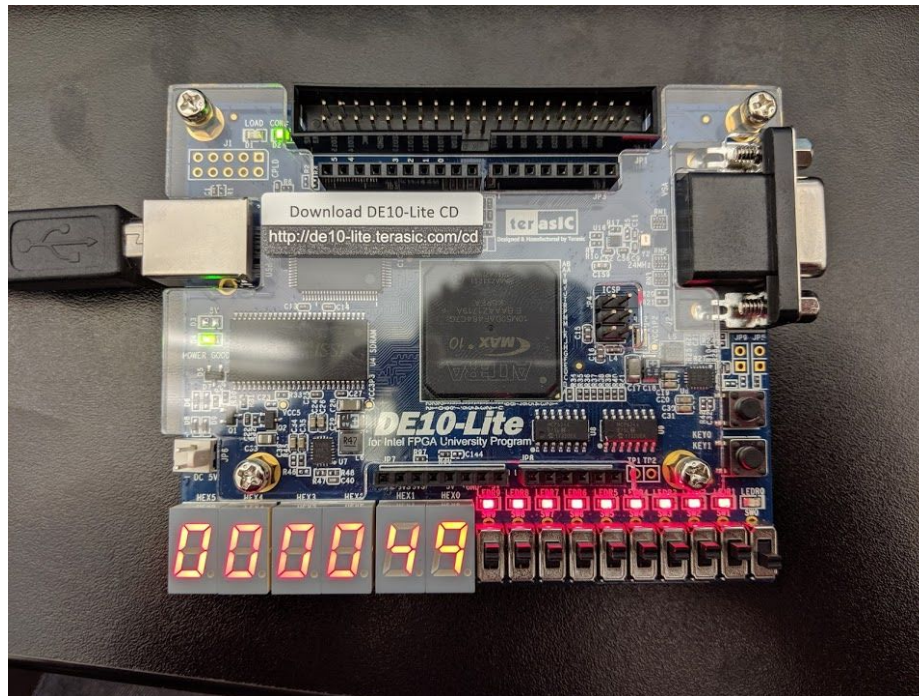


Figure (20) - DE10-Lite FPGA board displaying ALU multiplication operation results.

Conclusion (4)

In conclusion, the project performed and documented above shows the successful implementation of a simple Arithmetic Logic Unit (ALU). It can be scaled to form the basics of many complex digital systems. Many ALUs can be combined to form the main ALU in a computer. Implemented on an FPGA, the unit has dedicated logic gates for this operation. The ALU designed for this project incorporates five different modules, four sub-modules and one top-module. The four sub-modules are each responsible for the arithmetic and logic functions of the ALU, displaying the ALU output on the 7-segment displays, synchronizing the inputs to the 50MHz clock signal, and determining the states of the on-board LEDs. The top-module connects the different sub-modules together and connects the sub-module inputs/outputs to the physical components on the DE10-Lite FPGA board.

Although the ALU operated as desired, there are further improvements that could be made to the overall design. To conserve space, the integer (i) used in the module responsible for performing arithmetic and logic operations, could be changed to a register with less storage. Integers are 32 bit values, which is excessive for its use in this module. Furthermore, the display

module could be implemented in a better manner. Currently, it uses approximately 63 case statements to display all the possible decimal number combinations for the addition and multiplication operations. This design could be improved by incorporating a counter that increments by 1 for every clock cycle and compares its value to that of the output.

All software used to implement this design enabled ease of design in debugging and testing without the need to upload the program to the board to test whether it performs its required function. Verilog coding was used as it is a widely used Hardware Description Language. Logic functions were easily described in each module and tested with a testbench. All these features make it a suitable language to be used for this design.

Appendix A (5) - Verilog Code

- ALUlogic.v (5.1)
- Display.v (5.2)
- Sync.v (5.3)
- LED.v (5.4)
- ALU.v (5.5)
- Top Module Shift-left Operation Test Bench (5.6)
- Top Module Test Bench (5.7)

ALUlogic.v (5.1)

```
//=====
// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 11/14/2019
// Description: ALU logic and arithmetic functions.
//=====

module ALUlogic(rst_n,clk,ena,Dout,A,B,op); //initializing inputs/outputs and
registers
input rst_n,clk,ena;
input [2:0] A,B; //A and B are the ALU inputs
input [1:0] op; //op is the ALU control signal
output [5:0] Dout; //Dout is the ALU output
reg [5:0] Dout;
reg Cout; //Cout is the carry bit for the addition operation
reg [2:0] Sum; //Sum is the 3-bit summation variable for the addition operation
integer i = 0; //will be used to track the number of bits shifted to the left
reg [23:0] toggle; //12500000 DEC is a 23-bits long when converted to binary
reg [5:0] temp; //temporary register used in the shift left operation

always@(posedge clk) begin

    if(!rst_n) begin //when reset these values are returned to 0
        Dout <= 6'b0;
        Cout <= 1'b0;
        Sum <= 3'b0;
        i <= 0;
    end

    else if(ena == 1) begin //ensures that these operations will only occur when
the enable switch is high
        case(op) //op refers to the control signal, the ALU performs 4
different functions depending on its value
            2'b00:begin //when op = 00 the ALU performs a bitwise XNOR
operation
                Dout <= {3'b0, A ^ B}; //^^ refers to the bitwise
XNOR operation
                i <= 0; //sets i to 0 so that if op is changed to 01, the
shift left operation will restart
```



```

end
2'b01:begin //when op = 01 the ALU performs a shift left
operation on the concatenated inputs
    toggle <= toggle + 1; //when op = 01, toggle is
incremented by 1 every clock cycle
    if(toggle == 12500000) begin //once toggle reaches
12500000, 0.5 seconds has passed
        temp <= {A,B}; //temp is set to the initial value
of the A and B inputs
        Dout <= {A,B} << i; //the output is shifted to the
left by 1 for every loop
        i <= i + 1; //increments i for every loop
        toggle <= 0; //toggle is returned to 0 and will
begin to count back up to 12500000
    end
    else if(temp != {A,B}) begin //detects any changes in the
inputs A and B
        Dout <= {A,B}; //sets the output to the
re-evaluated inputs concatenated together
        i <= 0; //sets i to 0 so that the new output can
be re-shifted
    end
end
2'b10:begin //when op = 10 the ALU performs an addition
operation
    {Cout,Sum} <= A + B; //the addition of two 3-bit inputs
results in a 3-bit summation and a 1-bit carry
    Dout <= {2'b0,Cout,Sum};
    i <= 0; //sets i to 0 so that if op is changed to 01, the
shift left operation will restart
end
2'b11:begin //when op = 11 the ALU performs a multiplication
operation, this output is 6-bits
    case(B) //this multiplication operation was
implemented using the addition and shift left approach
        //the position of the 1's in B determines the
position of A in the concatenated output
        //and the number of 1's in B determines the number
of addition operations that will occur
        //i.e. B = 011, there are two 1's, therefore there
will be 2 concatenated values that will
        //be added together, the A will be shifted to the
left by 2-bits in the first value, and
        //shifted by 1-bit in the other
        3'b000:begin
            Dout <= 6'b0; //if B = 3'b0, the output
will be 6'b0 regardless of A's value
        end
        3'b001:begin
            Dout <= {3'b0,A}; //
        end
        3'b010:begin
            Dout <= {2'b0,A,1'b0};

```



```

        end
        3'b011:begin
            Dout <= {2'b0,A,1'b0} + {3'b0,A};
        end
        3'b100:begin
            Dout <= {1'b0,A,2'b0};
        end
        3'b101:begin
            Dout <= {1'b0,A,2'b0} + {3'b0,A};
        end
        3'b110:begin
            Dout <= {1'b0,A,2'b0} + {2'b0,A,1'b0};
        end
        3'b111:begin
            Dout <= {1'b0,A,2'b0} + {2'b0,A,1'b0} +
{3'b0,A};
        end
    endcase
    i <= 0; //sets i to 0 so that if op is changed to 01, the
shift left operation will restart
end
endcase
end
end
endmodule

```

Display.v (5.2)

```

//=====
// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 11/14/2019
// Description: ALU 7-segment display program.
//=====

module display(clk,rst_n,Dout,D0,D1,D2,D3,D4,D5,op,ena); //initializing
inputs/outputs and registers
input clk, rst_n,ena;
input [1:0] op;
input [5:0] Dout;
output reg [7:0] D0,D1,D2,D3,D4,D5; //store the 8-bit values that will be output to
each of the 7-segment displays
//for ease of typing each 7-segment number combination was initialized as a letter
reg [7:0] x = 8'b11000000; //0
reg [7:0] y = 8'b11111001; //1
reg [7:0] a = 8'b10100100; //2
reg [7:0] b = 8'b10110000; //3
reg [7:0] c = 8'b10011001; //4
reg [7:0] d = 8'b10010010; //5
reg [7:0] e = 8'b10000010; //6
reg [7:0] f = 8'b11111000; //7
reg [7:0] g = 8'b10000000; //8
reg [7:0] h = 8'b10011000; //9

```

```

always@(posedge clk) begin
    if(!rst_n) begin //when reset each of the 7-segment displays will show 0
        D0 <= x;
        D1 <= x;
        D2 <= x;
        D3 <= x;
        D4 <= x;
        D5 <= x;
    end
    else if(ena == 1'b1) //stops the output from being displayed in the wrong
base when ena = 0 and op is changed
        case(op)//depending on the value of the control signal, the output
will be displayed in base 2 or 10
            2'b00:begin //the XOR function will be displayed in binary on
HEX[2:0]
                D0 <= x; //each of the 7-segment displays are set to 0
initially to clear them
                D1 <= x;
                D2 <= x;
                D3 <= x;
                D4 <= x;
                D5 <= x;
                //the following if and else if statements determine the
values of each of the first
                //3 output bits (either a 1 or 0) and sets the 7-segment
display values accordingly
                if(Dout[0] == 1'b0)
                    D0 <= x;
                else if(Dout[0] == 1'b1)
                    D0 <= y;
                if(Dout[1] == 1'b0)
                    D1 <= x;
                else if(Dout[1] == 1'b1)
                    D1 <= y;
                if(Dout[2] == 1'b0)
                    D2 <= x;
                else if(Dout[2] == 1'b1)
                    D2 <= y;
            end
            2'b01:begin //shift left function will be displayed in binary
on HEX[5:0]
                D0 <= x; //each of the 7-segment displays are set to 0
initially to clear them
                D1 <= x;
                D2 <= x;
                D3 <= x;
                D4 <= x;
                D5 <= x;
                //the following if and else if statements determine the
values of each of the first
                //6 output bits (either a 1 or 0) and sets the 7-segment
display values accordingly

```

```

        if(Dout[0] == 1'b0)
            D0 <= x;
        else if(Dout[0] == 1'b1)
            D0 <= y;
        if(Dout[1] == 1'b0)
            D1 <= x;
        else if(Dout[1] == 1'b1)
            D1 <= y;
        if(Dout[2] == 1'b0)
            D2 <= x;
        else if(Dout[2] == 1'b1)
            D2 <= y;
        if(Dout[3] == 1'b0)
            D3 <= x;
        else if(Dout[3] == 1'b1)
            D3 <= y;
        if(Dout[4] == 1'b0)
            D4 <= x;
        else if(Dout[4] == 1'b1)
            D4 <= y;
        if(Dout[5] == 1'b0)
            D5 <= x;
        else if(Dout[5] == 1'b1)
            D5 <= y;
    end
2'b10:begin //addition function will be displayed in decimal on
HEX[1:0]
    D0 <= x; //each of the 7-segment displays are set to 0
initially to clear them
    D1 <= x;
    D2 <= x;
    D3 <= x;
    D4 <= x;
    D5 <= x;
    //the following case statement compares the ALU addition
operation output to each of
    //the predetermined cases (0-14 DEC) and returns the
appropriate 7-segment display
    //values
    case(Dout)
        //00
        6'b000000:begin
            D0 <= x;
            D1 <= x;
        end
        //01
        6'b000001: begin
            D0 <= y;
            D1 <= x;
        end
        //02
        6'b000010: begin
            D0 <= a;

```

```
        D1 <= x;
end
//03
6'b000011: begin
    D0 <= b;
    D1 <= x;
end
//04
6'b000100: begin
    D0 <= c;
    D1 <= x;
end
//05
6'b000101: begin
    D0 <= d;
    D1 <= x;
end
//06
6'b000110: begin
    D0 <= e;
    D1 <= x;
end
//07
6'b000111: begin
    D0 <= f;
    D1 <= x;
end
//08
6'b001000: begin
    D0 <= g;
    D1 <= x;
end
//09
6'b001001: begin
    D0 <= h;
    D1 <= x;
end
//10
6'b001010: begin
    D0 <= x;
    D1 <= y;
end
//11
6'b001011: begin
    D0 <= y;
    D1 <= y;
end
//12
6'b001100: begin
    D0 <= a;
    D1 <= y;
end
//13
```

```

        6'b001101: begin
            D0 <= b;
            D1 <= y;
        end
        //14
        6'b001110: begin
            D0 <= c;
            D1 <= y;
        end
    endcase
end
2'b11:begin //multiplication function will be displayed in
decimal on HEX[1:0]
    D0 <= x; //each of the 7-segment displays are set to 0
initially to clear them
    D1 <= x;
    D2 <= x;
    D3 <= x;
    D4 <= x;
    D5 <= x;
    //the following case statement compares the ALU
multiplication operation output
    //to each of the predetermined cases (0-49 DEC) and
returns the appropriate
    //7-segment display values
    case(Dout)
        //00
        6'b000000:begin
            D0 <= x;
            D1 <= x;
        end
        //01
        6'b000001: begin
            D0 <= y;
            D1 <= x;
        end
        //02
        6'b000010: begin
            D0 <= a;
            D1 <= x;
        end
        //03
        6'b000011: begin
            D0 <= b;
            D1 <= x;
        end
        //04
        6'b000100: begin
            D0 <= c;
            D1 <= x;
        end
        //05
        6'b000101: begin

```

```
        D0 <= d;
        D1 <= x;
    end
    //06
    6'b000110: begin
        D0 <= e;
        D1 <= x;
    end
    //07
    6'b000111: begin
        D0 <= f;
        D1 <= x;
    end
    //08
    6'b001000: begin
        D0 <= g;
        D1 <= x;
    end
    //09
    6'b001001: begin
        D0 <= h;
        D1 <= x;
    end
    //10
    6'b001010: begin
        D0 <= x;
        D1 <= y;
    end
    //11
    6'b001011: begin
        D0 <= y;
        D1 <= y;
    end
    //12
    6'b001100: begin
        D0 <= a;
        D1 <= y;
    end
    //13
    6'b001101: begin
        D0 <= b;
        D1 <= y;
    end
    //14
    6'b001110: begin
        D0 <= c;
        D1 <= y;
    end
    //15
    6'b001111: begin
        D0 <= d;
        D1 <= y;
    end
end
```

```
//16
6'b010000: begin
    D0 <= e;
    D1 <= y;
end
//17
6'b010001: begin
    D0 <= f;
    D1 <= y;
end
//18
6'b010010: begin
    D0 <= g;
    D1 <= y;
end
//19
6'b010011: begin
    D0 <= h;
    D1 <= y;
end
//20
6'b010100: begin
    D0 <= x;
    D1 <= a;
end
//21
6'b010101: begin
    D0 <= y;
    D1 <= a;
end
//22
6'b010110: begin
    D0 <= a;
    D1 <= a;
end
//23
6'b010111: begin
    D0 <= b;
    D1 <= a;
end
//24
6'b011000: begin
    D0 <= c;
    D1 <= a;
end
//25
6'b011001: begin
    D0 <= d;
    D1 <= a;
end
//26
6'b011010: begin
    D0 <= e;
```

```
        D1 <= a;
end
//27
6'b011011: begin
    D0 <= f;
    D1 <= a;
end
//28
6'b011100: begin
    D0 <= g;
    D1 <= a;
end
//29
6'b011101: begin
    D0 <= h;
    D1 <= a;
end
//30
6'b011110: begin
    D0 <= x;
    D1 <= b;
end
//31
6'b011111: begin
    D0 <= y;
    D1 <= b;
end
//32
6'b100000: begin
    D0 <= a;
    D1 <= b;
end
//33
6'b100001: begin
    D0 <= b;
    D1 <= b;
end
//34
6'b100010: begin
    D0 <= c;
    D1 <= b;
end
//35
6'b100011: begin
    D0 <= d;
    D1 <= b;
end
//36
6'b100100: begin
    D0 <= e;
    D1 <= b;
end
//37
```



```
6'b100101: begin
    D0 <= f;
    D1 <= b;
end
//38
6'b100110: begin
    D0 <= g;
    D1 <= b;
end
//39
6'b100111: begin
    D0 <= h;
    D1 <= b;
end
//40
6'b101000: begin
    D0 <= x;
    D1 <= c;
end
//41
6'b101001: begin
    D0 <= y;
    D1 <= c;
end
//42
6'b101010: begin
    D0 <= a;
    D1 <= c;
end
//43
6'b101011: begin
    D0 <= b;
    D1 <= c;
end
//44
6'b101100: begin
    D0 <= c;
    D1 <= c;
end
//45
6'b101101: begin
    D0 <= d;
    D1 <= c;
end
//46
6'b101110: begin
    D0 <= e;
    D1 <= c;
end
//47
6'b101111: begin
    D0 <= f;
    D1 <= c;
```

```

        end
        //48
        6'b110000: begin
            D0 <= g;
            D1 <= c;
        end
        //49
        6'b110001: begin
            D0 <= h;
            D1 <= c;
        end
    end
endcase
end
default: begin //by default each of the 7-segment displays will
show 0
        D0 <= x;
        D1 <= x;
        D2 <= x;
        D3 <= x;
        D4 <= x;
        D5 <= x;
    end
endcase
end
endmodule

```

Sync.v (5.3)

```

//=====
// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 11/14/2019
// Description: Two D flip-flop (DFF) synchronizer.
//=====

module sync(rst_n,clk,in,out); //initializing inputs/outputs and registers
input rst_n,clk,in; //in is the input to the first DFF
output out; //out is the output of the second DFF
reg d1, out; //d1 is the output of the first DFF, and input to the second DFF

always @(posedge clk) begin

    if (!rst_n) begin //when reset both DFF outputs are set to 0
        d1 <= 1'b0;
        out <= 1'b0;
    end
    else begin
        d1 <= in; //the output of the first DFF is set to its input
        out <= d1; //the output of the second DFF is set to the output of the
    end
end

```

```

first DFF
    end
end
endmodule

```

LED.v (5.4)

```

//=====
// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 11/14/2019
// Description: LED operation based off their corresponding switch state.
//=====

module LED(clk,rst_n,led,op,A,B,ena); //initializing inputs/outputs and registers
input clk,rst_n,ena;
input [2:0] A,B;
input [1:0] op;
output reg [9:0] led; //10-bit output signal that will determine if the LEDs are
on/off

always@(posedge clk) begin
    if(!rst_n)
        led <= 10'b000000001; //when reset, all the relevant LEDs ([9:1])
will turn on
    else
        led <= {ena,op,B,A,1'b1}; //otherwise the LEDs will be on/off
depending on the state of their switches
end
endmodule

```

ALU.v (5.5)

```

//=====
// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 11/14/2019
// Description: ALU top module.
//=====

module ALU(MAX10_CLK1_50,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,KEY,LEDR,SW);
//input/output declarations
input MAX10_CLK1_50; //MAX10_CLK1_50
output [7:0] HEX0;
output [7:0] HEX1;
output [7:0] HEX2;
output [7:0] HEX3;
output [7:0] HEX4;
output [7:0] HEX5;

```

```
input [1:0] KEY;
output [9:0] LEDR;
input [9:0] SW;

//reg/wire declarations
wire enable; //connects the enable signals to each of the separate modules
wire [1:0] op_in; //
wire [2:0] A;
wire [2:0] B;
wire [5:0] Dout;

//sync module
sync u_sync1(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[1]),
.out(A[0]));

sync u_sync2(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[2]),
.out(A[1]));

sync u_sync3(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[3]),
.out(A[2]));

sync u_sync4(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[4]),
.out(B[0]));

sync u_sync5(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[5]),
.out(B[1]));

sync u_sync6(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[6]),
.out(B[2]));

sync u_sync7(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[7]),
.out(op_in[0]));

sync u_sync8(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.in(SW[8]),
.out(op_in[1]));

sync u_sync9(.rst_n(KEY[0]),
```

```

.clk(MAX10_CLK1_50),
.in(SW[9]),
.out(enable));

//ALU logic module
ALUlogic u_ALUlogic(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.Dout(Dout),
.A(A),
.B(B),
.op(op_in),
.ena(enable));

//LED module
LED u_LED(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.ena(enable),
.A(A),
.B(B),
.op(op_in),
.led(LED_R));

//display module
display u_display(.rst_n(KEY[0]),
.clk(MAX10_CLK1_50),
.Dout(Dout),
.D0(HEX0),
.D1(HEX1),
.D2(HEX2),
.D3(HEX3),
.D4(HEX4),
.D5(HEX5),
.op(op_in),
.ena(enable));

endmodule

```

Top Module Shift-left Operation Test Bench (5.6)

```

//=====
// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 15/11/2019
// Description: ALU top module test bench for shift-left operation.
//=====

`timescale 1ns/1ns
module ALUtestbench; //initializing all registers and wires
reg MAX10_CLK1_50;
reg [1:0] KEY;
reg [9:0] SW;

```

```

wire [9:0] LEDR;
wire [7:0] HEX0;
wire [7:0] HEX1;
wire [7:0] HEX2;
wire [7:0] HEX3;
wire [7:0] HEX4;
wire [7:0] HEX5;

ALU uut(.KEY(KEY),.SW(SW),.LEDR(LEDR),.MAX10_CLK1_50(MAX10_CLK1_50),
.HEX0(HEX0),.HEX1(HEX1),.HEX2(HEX2),.HEX3(HEX3),.HEX4(HEX4),.HEX5(HEX5));

initial begin
    KEY = 2'b01; //KEY[0] is set to 1 (rst_n)
    MAX10_CLK1_50 = 1'b0; //clock state is initialized at 0
    SW = 10'b1011011010; //SW[8:7] = 2'b01, shift-left operation
end

always begin
    #10
    MAX10_CLK1_50 = ~MAX10_CLK1_50; //clock state changes every 10 ns, which is
a frequency of 50MHz
end

initial begin //after 100 ns the 3-bit inputs A and B are changed to ensure that
the shift-left operation restarts
    #100
    SW[6:4] = 3'b010;
    SW[3:1] = 3'b011;
end

initial begin //the reset signal is toggled after 120 ns to ensure the shift-left
operation restarts
    #120
    KEY[0] = 1'b0;
    #20
    KEY[0] = 1'b1;
    #20
    SW[9] = 1'b0; //the enable signal is toggled after 160 ns to ensure proper
operation
    #100
    SW[9] = 1'b1;
end
endmodule

```

Top Module Test Bench (5.7)

```
//=====
```

```

// ENGI 4054: Digital VLSI Design
// Author: Martti Muzyka and Obatosin Obat-Olowu
// Date: 15/11/2019
// Description: ALU top module test bench for XNOR, addition and multiplication
operations.
//=====

`timescale 1ns/1ns
module ALUtestbench; //initializing all registers and wires
reg MAX10_CLK1_50;
reg [1:0] KEY;
reg [9:0] SW;
wire [9:0] LEDR;
wire [7:0] HEX0;
wire [7:0] HEX1;
wire [7:0] HEX2;
wire [7:0] HEX3;
wire [7:0] HEX4;
wire [7:0] HEX5;

ALU uut(.KEY(KEY),.SW(SW),.LEDR(LEDR),.MAX10_CLK1_50(MAX10_CLK1_50),
.HEX0(HEX0),.HEX1(HEX1),.HEX2(HEX2),.HEX3(HEX3),.HEX4(HEX4),.HEX5(HEX5));

initial begin
    KEY = 2'b01; //KEY[0] is set to 1 (rst_n)
    MAX10_CLK1_50 = 1'b0; //clock state is initialized at 0
    SW = 10'b1000000000; //SW[8:7] = 2'b00, XNOR operation
    /*SW = 10'b1100000000; //SW[8:7] = 2'b10, addition operation*/
    /*SW = 10'b1110001010; //SW[8:7] = 2'b11, multiplication operation*/
end

always begin
    #10
    MAX10_CLK1_50 = ~MAX10_CLK1_50; //clock state changes every 10 ns, which is
a frequency of 50MHz
end

always begin //the 3-bit input B is incremented by 1 every 40 ns
    #40
    SW[6:4] = 3'b001;
    #40
    SW[6:4] = 3'b010;
    #40
    SW[6:4] = 3'b011;
    #40
    SW[6:4] = 3'b100;
    #40

```

```

        SW[6:4] = 3'b101;
        #40
        SW[6:4] = 3'b110;
        #40
        SW[6:4] = 3'b111;
    end

    initial begin //the reset signal is toggled after 200 ns to ensure proper operation
        #200
        KEY[0] = 1'b0;
        #50
        KEY[0] = 1'b1;
        #50
        SW[9] = 1'b0; //the enable signal is toggled after 300 ns to ensure proper
operation
        #100
        SW[9] = 1'b1;
    end
endmodule

```

Appendix B (6) - Summary Reports

- ALU.fit.summary (6.1)
- ALU.map.summary (6.2)
- ALU.sta.summary (6.3)

ALU.fit.summary (6.1)

Fitter Status : Successful - Thu Nov 14 17:29:24 2019

Quartus Prime Version : 16.1.0 Build 196 10/24/2016 SJ Lite Edition

Revision Name : ALU

Top-level Entity Name : ALU

Family : MAX 10

Device : 10M50DAF484C7G

Timing Models : Final

Total logic elements : 287 / 49,760 (< 1 %)

Total combinational functions : 280 / 49,760 (< 1 %)

Dedicated logic registers : 116 / 49,760 (< 1 %)

Total registers : 116

Total pins : 71 / 360 (20 %)

Total virtual pins : 0

Total memory bits : 0 / 1,677,312 (0 %)

Embedded Multiplier 9-bit elements : 0 / 288 (0 %)

Total PLLs : 0 / 4 (0 %)

UFM blocks : 0 / 1 (0 %)

ADC blocks : 0 / 2 (0 %)

ALU.map.summary (6.2)

Analysis & Synthesis Status : Successful - Thu Nov 14 17:29:15 2019

Quartus Prime Version : 16.1.0 Build 196 10/24/2016 SJ Lite Edition

Revision Name : ALU

Top-level Entity Name : ALU

Family : MAX 10

Total logic elements : 285

 Total combinational functions : 279

 Dedicated logic registers : 116

Total registers : 116

Total pins : 71

Total virtual pins : 0

Total memory bits : 0

Embedded Multiplier 9-bit elements : 0

Total PLLs : 0

UFM blocks : 0

ADC blocks : 0

ALU.sta.summary (6.3)

TimeQuest Timing Analyzer Summary

Type : Slow 1200mV 85C Model Setup 'MAX10_CLK1_50'

Slack : 12.202

TNS : 0.000

Type : Slow 1200mV 85C Model Hold 'MAX10_CLK1_50'

Slack : 0.348

TNS : 0.000

Type : Slow 1200mV 85C Model Minimum Pulse Width 'MAX10_CLK1_50'

Slack : 9.630

TNS : 0.000

Type : Slow 1200mV 0C Model Setup 'MAX10_CLK1_50'

Slack : 12.914

TNS : 0.000

Type : Slow 1200mV 0C Model Hold 'MAX10_CLK1_50'

Slack : 0.312

TNS : 0.000

Type : Slow 1200mV 0C Model Minimum Pulse Width 'MAX10_CLK1_50'

Slack : 9.645

TNS : 0.000

Type : Fast 1200mV 0C Model Setup 'MAX10_CLK1_50'

Slack : 16.531

TNS : 0.000

Type : Fast 1200mV 0C Model Hold 'MAX10_CLK1_50'

Slack : 0.152

TNS : 0.000

Type : Fast 1200mV 0C Model Minimum Pulse Width 'MAX10_CLK1_50'

Slack : 9.326

TNS : 0.000

Appendix C (7) - References

- [1] B. Kaushik, V. Anand, K. Yasmeen and A. Kaur, "Power Efficient Arithmetic and Logical Unit Design on FPGA," *2018 6th Edition of International Conference on Wireless Networks & Embedded Systems (WECON)*, Rajpura (near Chandigarh), India, 2018, pp. 112-115.doi: 10.1109/WECON.2018.8782051
- [2] K. Rajagopalan and P. Sutton, "A flexible multiplication unit for an FPGA logic block," *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, Sydney, NSW, 2001, pp. 546-549 vol. 4.doi: 10.1109/ISCAS.2001.922295
- [3] R. W. Hix and R. L. Haggard, "Comparative design methodologies for FPGA based computer arithmetic," *Proceedings The Twenty-Ninth Southeastern Symposium on System Theory*, Cookeville, TN, USA, 1997, pp. 374-378.doi: 10.1109/SSST.1997.581670

- [4] S. Carrillo, H. Carrillo and F. Viveros, "Design and Implementation of an Arithmetic Processor Unit Based on the Logarithmic Number System," in *IEEE Latin America Transactions*, vol. 8, no. 6, pp. 605-617, Dec. 2010. doi: 10.1109/TLA.2010.5688085

Appendix D (8) - Index

Verilog 2, 19, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42
Arithmetic Logic Unit (ALU) 8, 11
XNOR 8, 9, 24, 27
Addition 9, 10, 25, 28
Shift-left 9, 10, 25, 27
Multiplication 9, 10, 26, 28
7-segment display 9, 10, 14, 16
RTL simulations 19, 20, 22, 23, 24, 25, 26
Concatenated 9, 10
Control signal 9, 10, 11, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26
Synchronized reset 9, 15, 16, 17, 18, 21, 22, 23, 24, 25, 26
Enable signal 9, 15, 16, 17, 18, 22, 23, 24, 25, 26
Pin assignment 11
I/O standards 11
Top-module 12, 13, 20, 21, 22, 23, 24, 25, 26
Sub-module 12, 13, 14, 15, 16, 17, 18
Block diagram 12, 13, 14, 15
Timing diagram 15, 16, 17, 18
D flip-flop 17
Design hierarchy 19
Modelsim 19, 22, 23, 24, 25, 26
Testbench 19, 20, 21, 22, 23, 24, 25, 26, 43, 44
Test vectors 21, 22, 23, 24, 25, 26
Summary reports 45,46,47