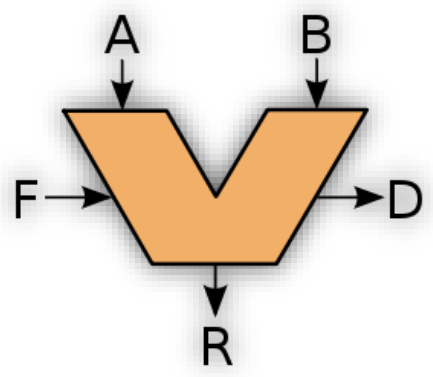


Mini Project Report

ARITHMETIC LOGIC UNIT

Alberto Arriaga Felix (861181553)

EE120A Section 025

TA: Linchao Liao

Lab Partner: Amit Gupta

aarri002@ucr.edu March 20, 2015

Specifications

Our challenge for this mini project was to design and implement an Arithmetic Logic Unit (ALU). We are required to complete this project using all of the knowledge we've gained so far in the laboratory, and additional individual research on the topic.

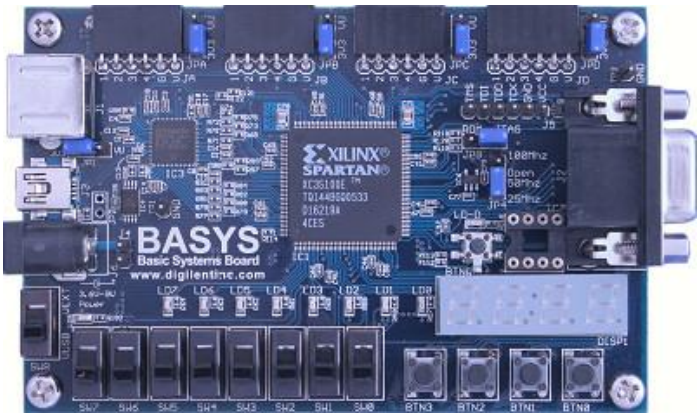
An ALU is a digital circuit that performs a specified set of arithmetic and bitwise logical operations on integer binary numbers called operands. It is a fundamental block in most computer systems. An ALU has a variety of input and output nets, which are the shared electrical connections used to convey digital signals between the ALU and external circuitry. When an ALU is operating, external circuits apply signals to the ALU inputs and, in response, the ALU produces and conveys signals to external circuitry via its outputs.

A basic ALU has three parallel data buses consisting of two input operands (A and B) and a result output (R). The values of the operands A and B in this case are specifically chosen by the user using the ALU via hardware components. There must be registers inside the ALU to store the operands A and B in order for operations to be performed on them. In the circuit there must also be exist an input OpCode, which conveys to the ALU an operation selection code that enumerates the data manipulation that is to be performed on the operands.

The ALU specifications for this project are as follows;

- Two Operands inputs A and B (8bits each)
- 8 arithmetic data manipulations
- 4 bitwise logical operations
- One output R (8bits) to be displayed on 8 LEDS.

This ALU is to be designed to run on simulation using the Xilinx Testbench environment, and hardware on the Basys FPGA Board.



DESIGN AND ARCHITECTURE

To achieve the specifications of this project first we must reduce the problem to its most basic components.

For this ALU we will have the following inputs and outputs;

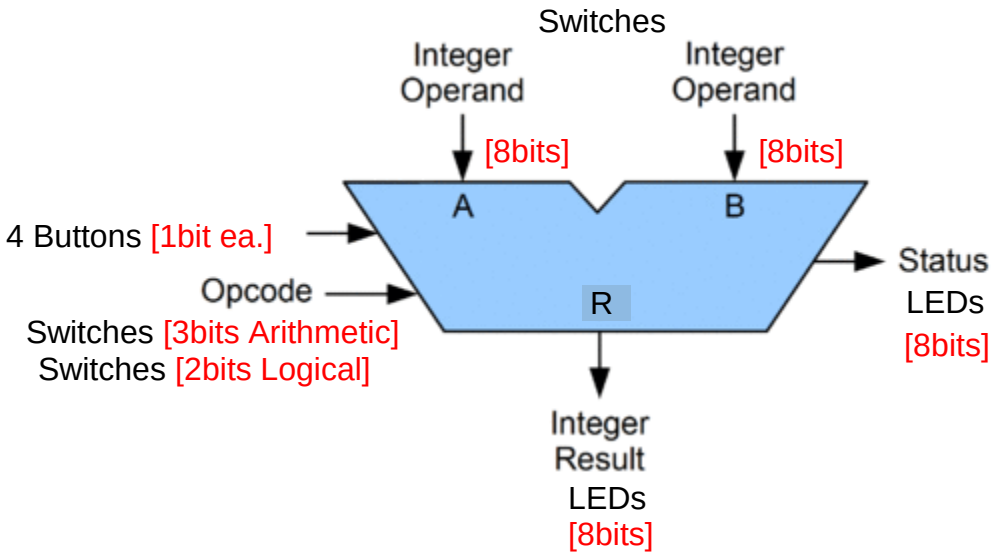
Inputs:

- Two input operands (A, B [8bits]).
- Two input buttons to store data (StoreA, StoreB [1bit])
- Opcode selection [3bits for Arithmetic] [2bits for Logical].
- Two input buttons to display manipulations (DisplayA, DisplayL [1bit])

Outputs:

- A status notification that shows the values of the operands in memory (Status [8bits]).
- Integer result (R [8bits]).

The following diagram shows inputs and outputs.



For this project we are limited on the hardware side to only 8 switches and 4 buttons for input, and 8 LEDs for output. We made the corresponding utilization choices for those elements in the above diagram.

Since we have a limited amount of switches, buttons and LEDs on the Basys board. We had to choose an architecture that would allow us to reuse all of the input and output elements on the board.

The switches on the board would be utilized for two purposes:

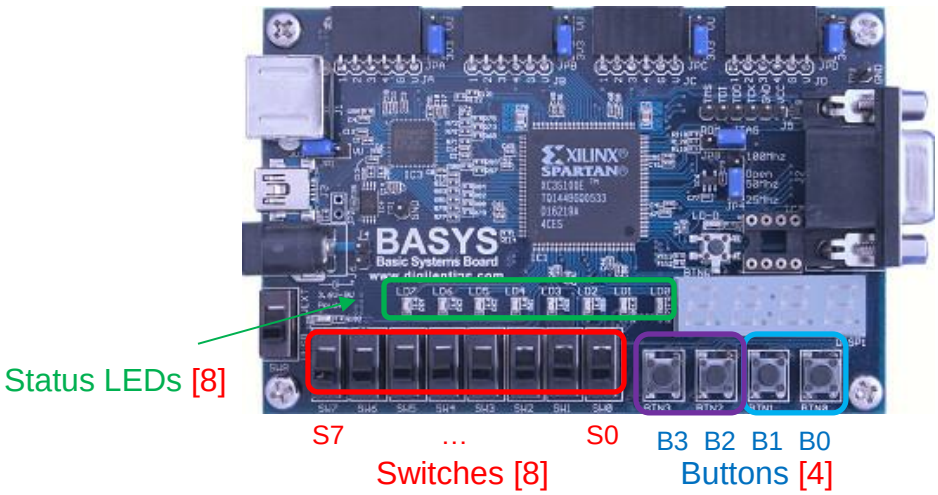
- Data input into Operands A and B.
- OpCode selection for data manipulation.

The buttons would be used for:

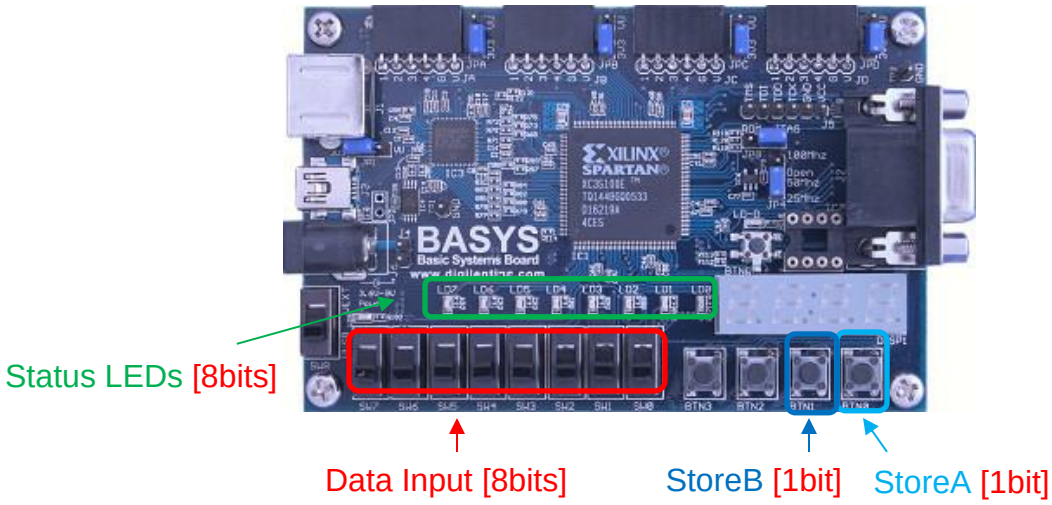
- Data storing actions for operands A and B
- Display actions for arithmetic or logical operations.

The LEDs would be used for:

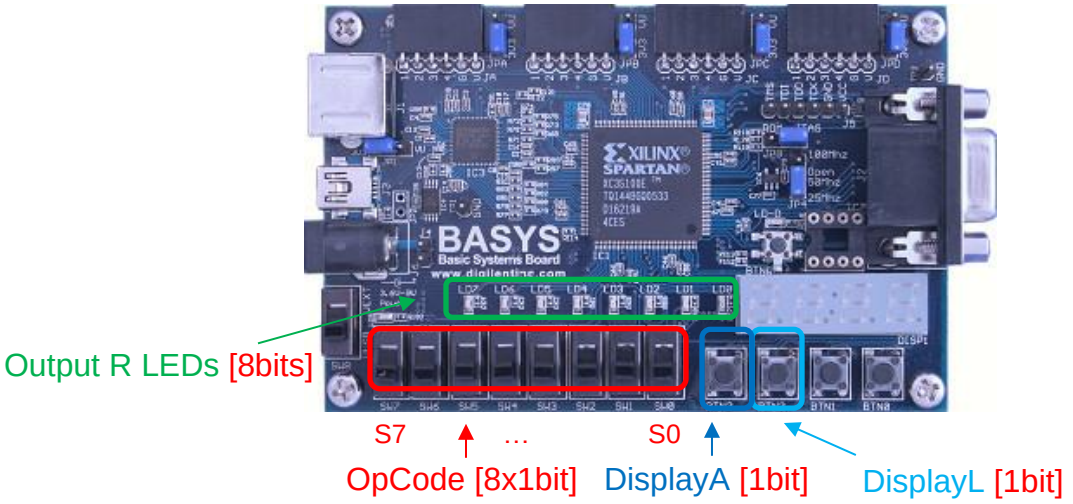
- Displaying status of operands A and B in register file.
- Displaying result R, after data manipulations.



Functions on Basys board:



Data input is done in [8bit] binary form via the switches.
StoreA is pressed to store the data currently on Data input switches to register A.
StoreB is pressed to store the data currently on Data input switches to register B.
Status LEDs show the data currently on register A or B when StoreA or Store B are pressed.

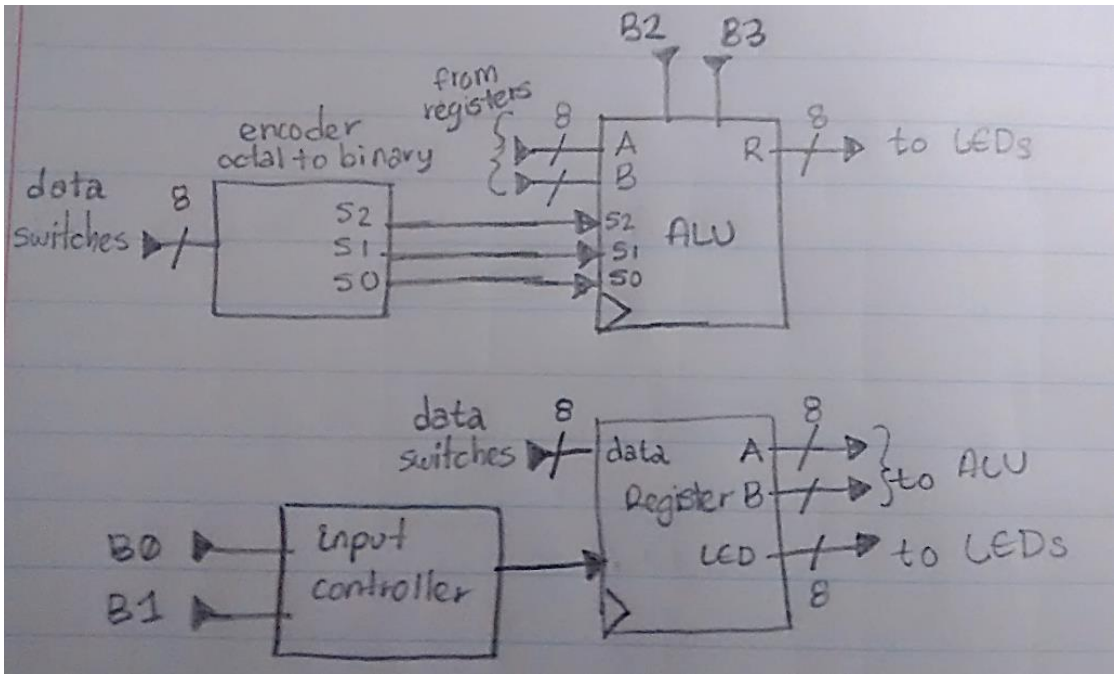


Opcode selections is done in [8x1bit] form via the switches.
DisplayA is pressed to perform the selected arithmetic operation on OpCode switches.
DisplayL is pressed to perform the selected logical operation on OpCode switches.
Output R LEDs show the result of the arithmetic or logical operations done by the ALU.

The following table shows the OpCodes for Arithmetic or Logical operations and their respective switch.

Opcodes			Arithmetic	
S2	S1	S0	Operation	Switch #
0	0	0	A + B	0
0	0	1	A - B	1
0	1	0	A + 1	2
0	1	1	A - 1	3
1	0	0	-A	4
1	0	1	A*1	5
1	1	0	A*2	6
1	1	1	A/2	7
Opcodes			Logical	
S2	s1	s0	Operation	Switch #
x	0	0	A AND B	0
x	0	1	A OR B	1
x	1	0	A XOR B	2
x	1	1	NOT A	3

The basic design of our architecture:



We used the following components:

- Encoder
- Register File
- Input/Register Controller
- ALU

The Encoder takes the 8 switches and converts them from [8bit] input to [8x1bit] inputs. This makes it easier on the user when selecting an operation they want to perform with the ALU.

The Register File is used to store the operands A and B that are user specified using the buttons StoreA (B0) and StoreB (B1).

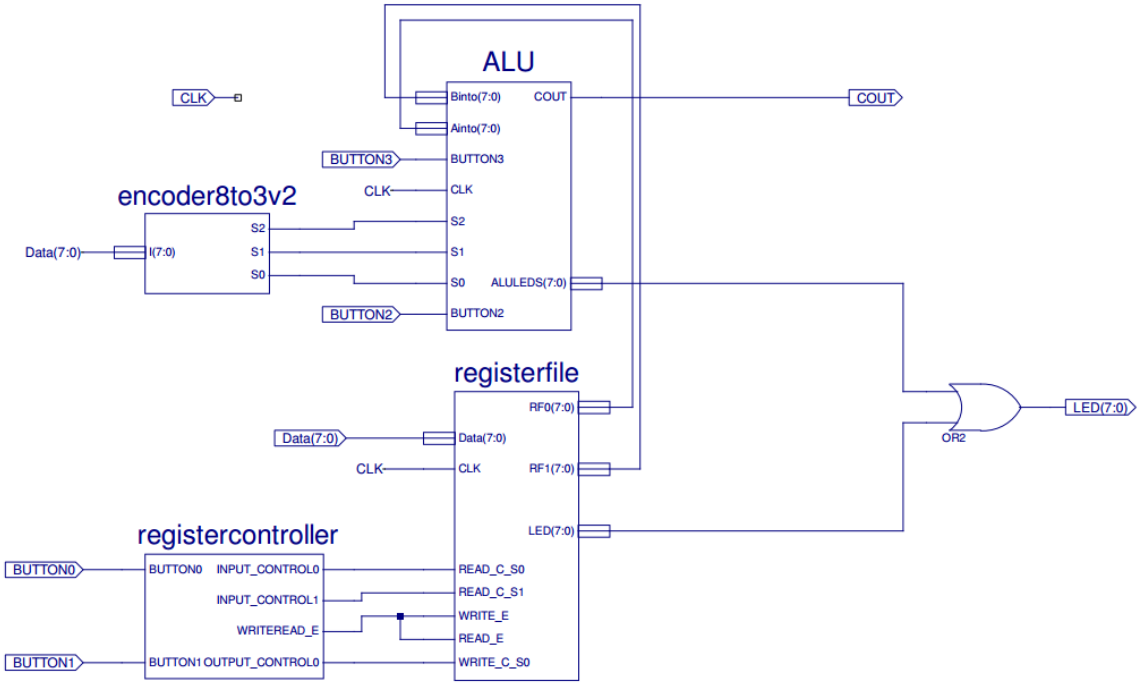
The Input Controller controls the input to the register file, which includes the selections of a specific register via the buttons StoreA (B0) and StoreB (B1).

The ALU is the basic component necessary for the arithmetic and logical operations, it is controlled by the buttons DisplayA (B3) and DisplayL (B2).

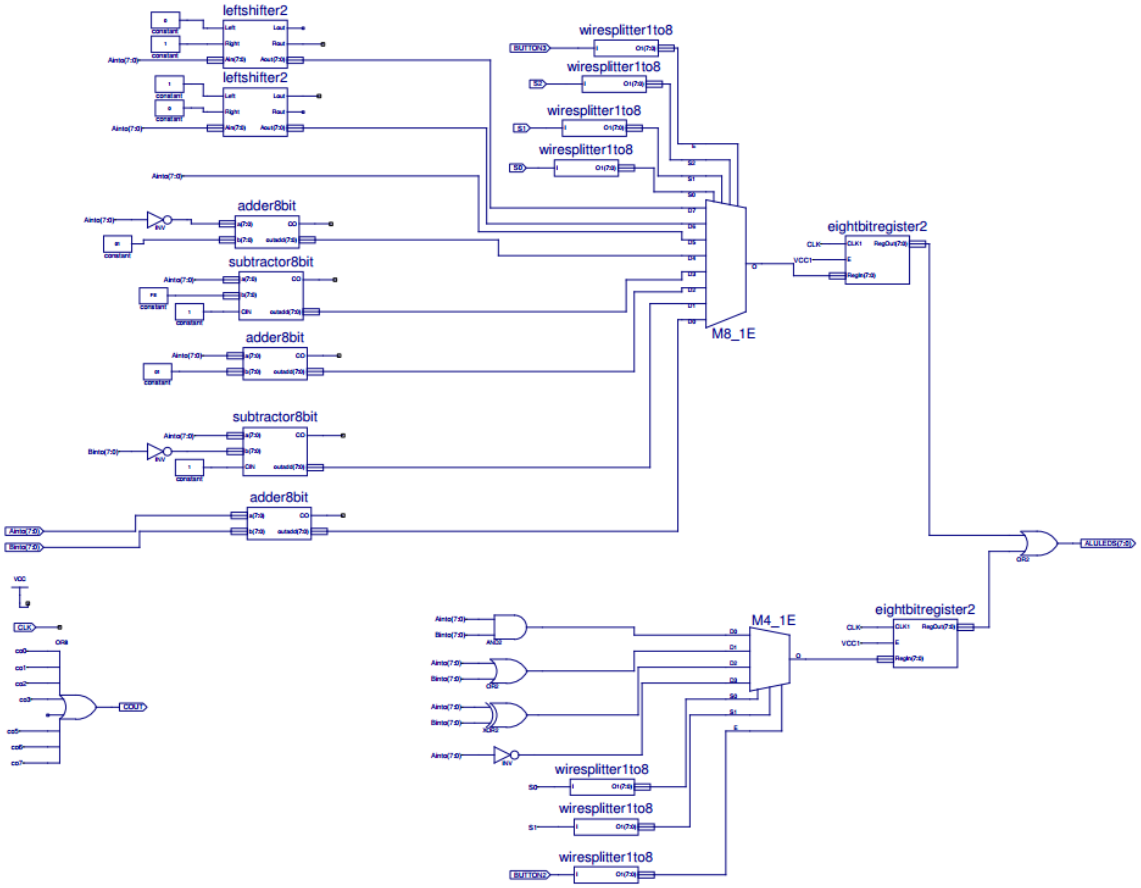
RECORDS

The schematics of our design in Xilinx.

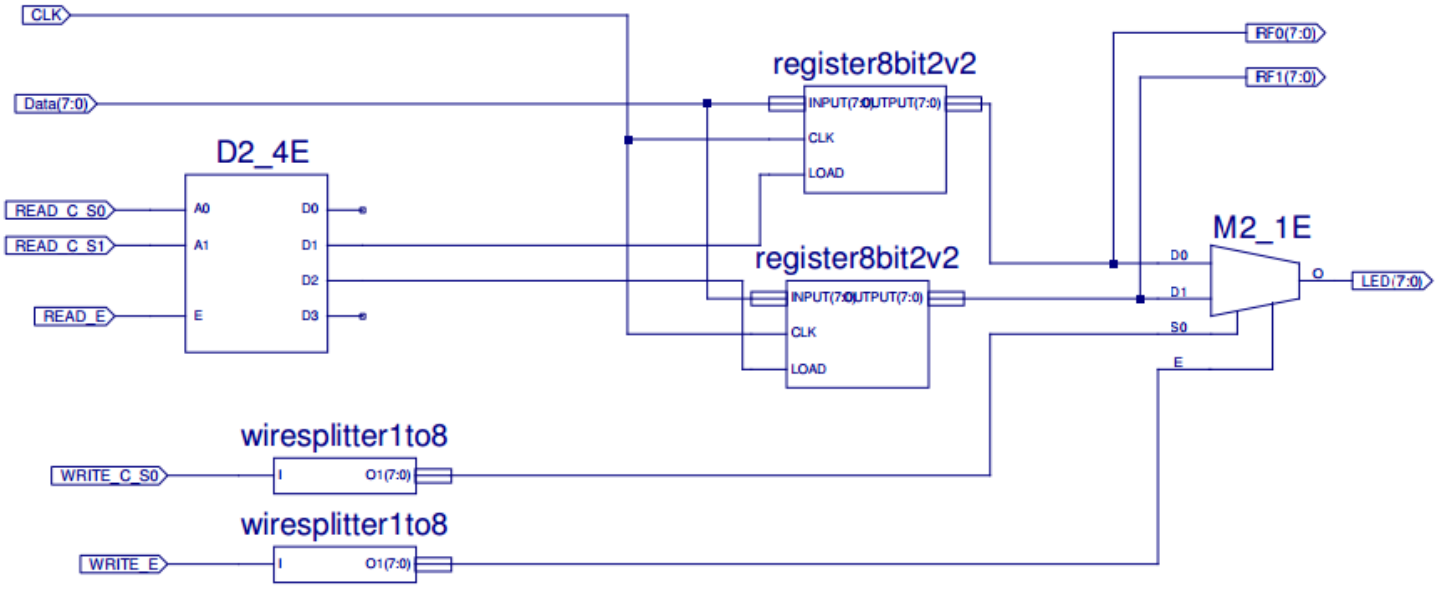
Main Schematic:



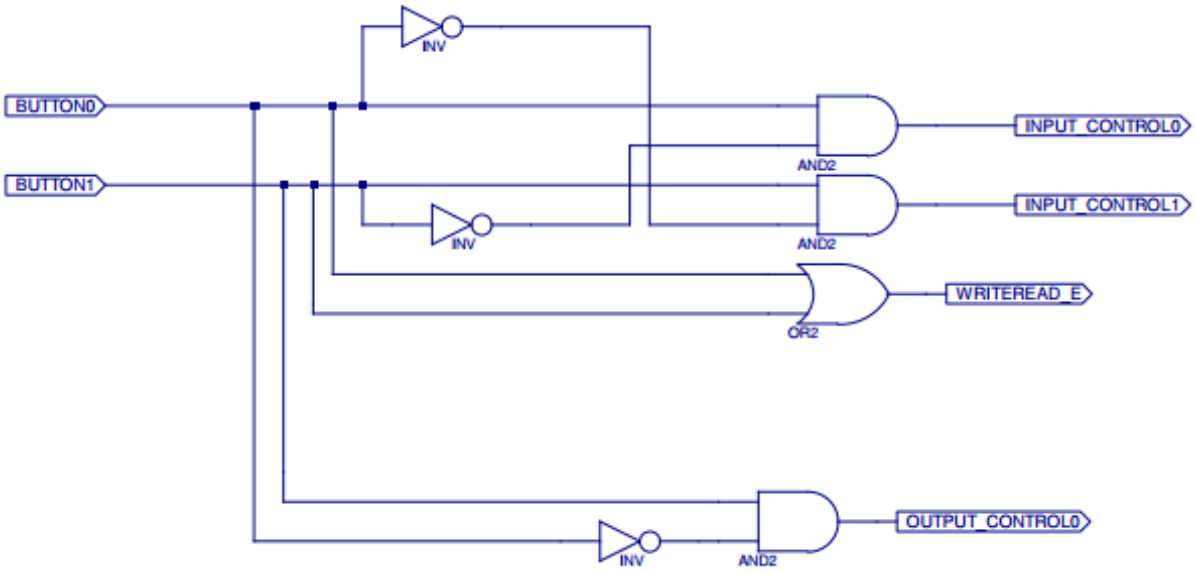
ALU:



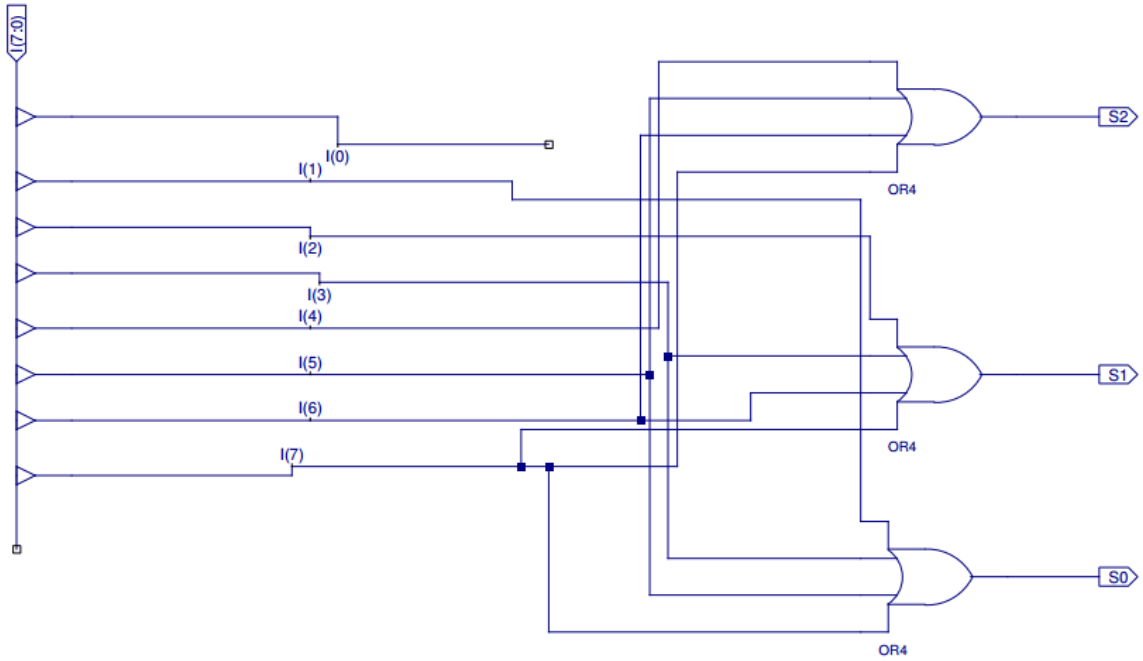
Register File:



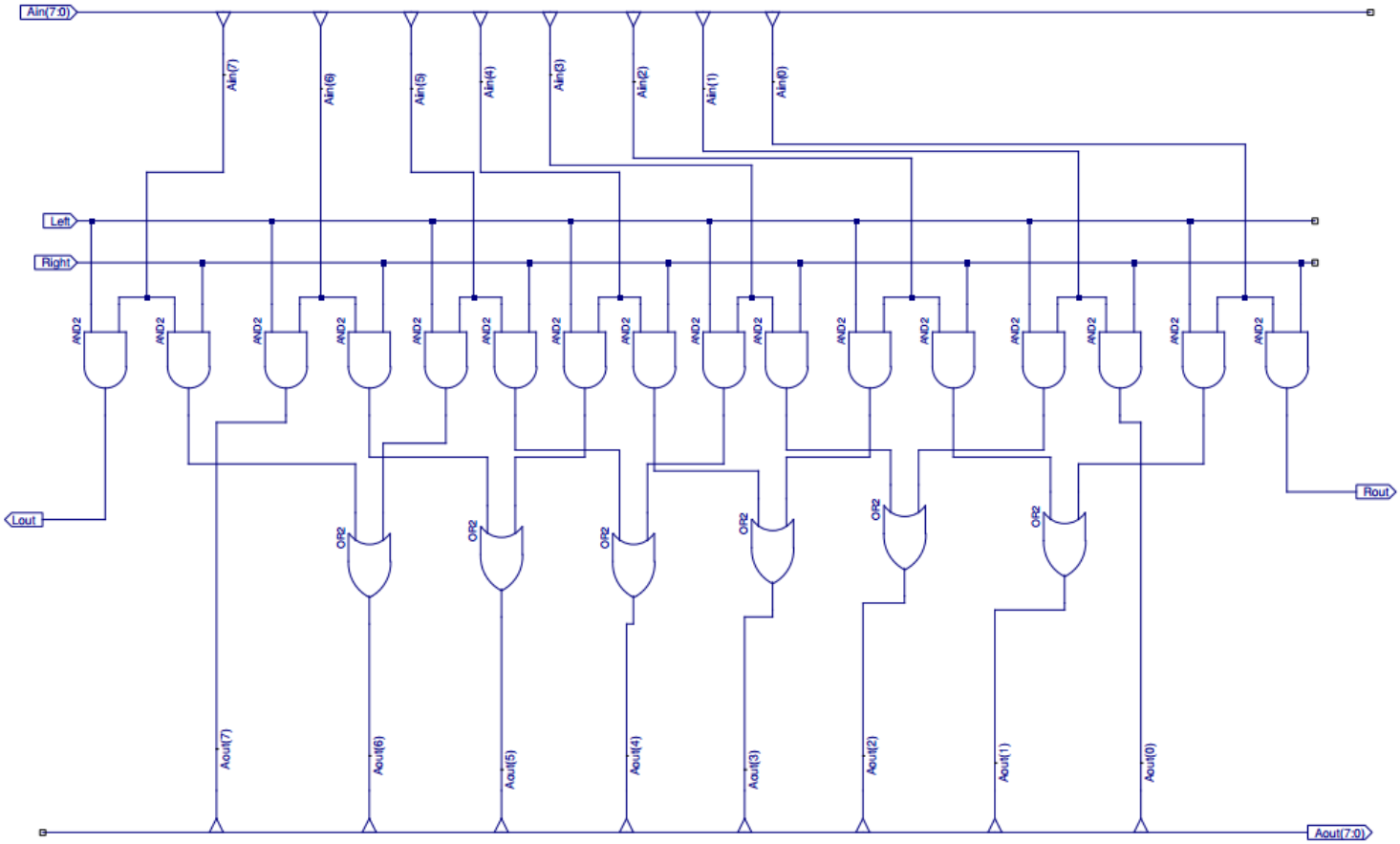
Register Controller:



Encoder Octal to 3bit Binary



Shifter

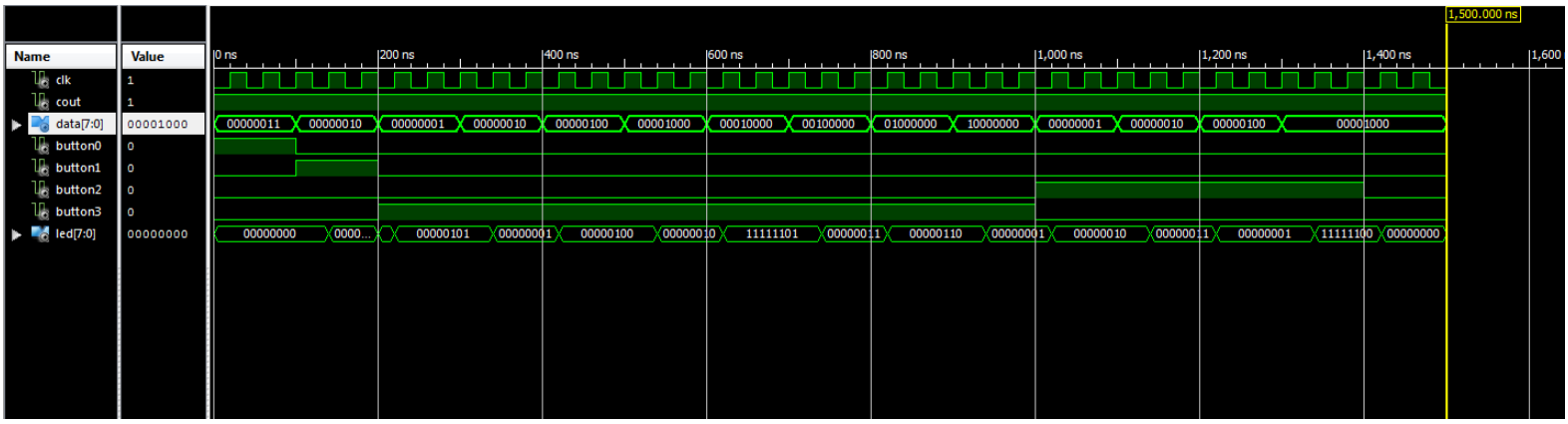


Simulation Files

TestBench Code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY MiniProject_MiniProject_sch_tb IS
END MiniProject_MiniProject_sch_tb;
ARCHITECTURE behavioral OF MiniProject_MiniProject_sch_tb IS

   COMPONENT MiniProject
   PORT( Data   :  IN STD_LOGIC_VECTOR (7 DOWNTO 0);
         COUT :  OUT   STD_LOGIC;
         LED  :  OUT   STD_LOGIC_VECTOR (7 DOWNTO 0);
         BUTTON3 :  IN STD_LOGIC;
         CLK  :  IN STD_LOGIC;
         BUTTON2 :  IN STD_LOGIC;
         BUTTON0 :  IN STD_LOGIC;
         BUTTON1 :  IN STD_LOGIC);
   END COMPONENT;

   SIGNAL Data :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL COUT :  STD_LOGIC;
   SIGNAL LED  :  STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL BUTTON3 :  STD_LOGIC;
   SIGNAL CLK  :  STD_LOGIC;
   SIGNAL BUTTON2 :  STD_LOGIC;
   SIGNAL BUTTON0 :  STD_LOGIC;
   SIGNAL BUTTON1 :  STD_LOGIC;

BEGIN

   UUT: MiniProject PORT MAP(
      Data => Data,
      COUT => COUT,
      LED => LED,
      BUTTON3 => BUTTON3,
      CLK => CLK,
      BUTTON2 => BUTTON2,
      BUTTON0 => BUTTON0,
      BUTTON1 => BUTTON1
   );

   CLKtest: Process
   Begin
      CLK <= '0';
   Wait for 20ns;
      CLK <= '1';
   Wait for 20ns;
   End Process;

   Data1: Process
   Begin
      Data <= "00000011";
   Wait for 100ns;
      Data <= "00000010";
   Wait for 100ns;
      Data <= "00000001";
   Wait for 100ns;
      Data <= "00000010";
   Wait for 100ns;
      Data <= "00000100";
   Wait for 100ns;
      Data <= "00001000";
   Wait for 100ns;
      Data <= "00010000";
   Wait for 100ns;
      Data <= "00100000";
   Wait for 100ns;
      Data <= "01000000";
   Wait for 100ns;
      Data <= "10000000";
   Wait for 100ns;
      Data <= "00000001";
   Wait for 100ns;
      Data <= "00000010";
   Wait for 100ns;
      Data <= "00000100";
   Wait for 100ns;
      Data <= "00001000";
   Wait;
   End Process;

   BUTTON0Test: Process
   Begin
      BUTTON0 <= '1';
   Wait for 100ns;
      BUTTON0 <= '0';
   Wait;
   End Process;

   BUTTON1Test: Process
   Begin
      BUTTON1 <= '0';
   Wait for 100ns;
      BUTTON1 <= '1';
   Wait for 100ns;
      BUTTON1 <= '0';
   Wait;
   End Process;



   BUTTON2Test: Process
   Begin
      BUTTON2 <= '0';
   Wait for 1us;
      BUTTON2 <= '1';
   Wait for 400ns;
      BUTTON2 <= '0';
   Wait;
   End Process;

   BUTTON3Test: Process
   Begin
      BUTTON3 <= '0';
   Wait for 200ns;
      BUTTON3 <= '1';
   Wait for 800ns;
      BUTTON3 <= '0';
   Wait;
   End Process;


-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Simulation Waveform:

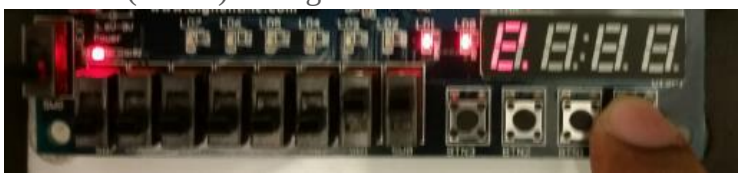


Hardware Implementation constrains file:

```
1   NET "CLK" LOC = "p54";
2
3   NET "BUTTON0" LOC = "p69";
4   NET "BUTTON1" LOC = "p48";
5   NET "BUTTON2" LOC = "p47";
6   NET "BUTTON3" LOC = "p41";
7
8   NET "Data(7)" LOC = "p6";
9   NET "Data(6)" LOC = "p10";
10  NET "Data(5)" LOC = "p12";
11  NET "Data(4)" LOC = "p18";
12  NET "Data(3)" LOC = "p24";
13  NET "Data(2)" LOC = "p29";
14  NET "Data(1)" LOC = "p36";
15  NET "Data(0)" LOC = "p38";
16
17  NET "LED(7)" LOC = "p2";
18  NET "LED(6)" LOC = "p3";
19  NET "LED(5)" LOC = "p4";
20  NET "LED(4)" LOC = "p5";
21  NET "LED(3)" LOC = "p7";
22  NET "LED(2)" LOC = "p8";
23  NET "LED(1)" LOC = "p14";
24  NET "LED(0)" LOC = "p15";
```

Hardware Demo:

StoreA 3 (11 bin) in Register A:

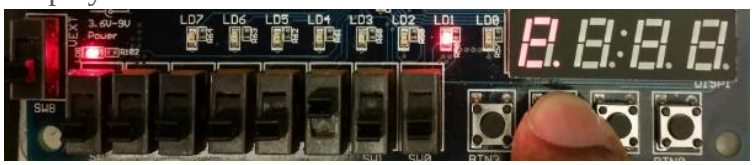


StoreB 1 (01 bin) in Register B:

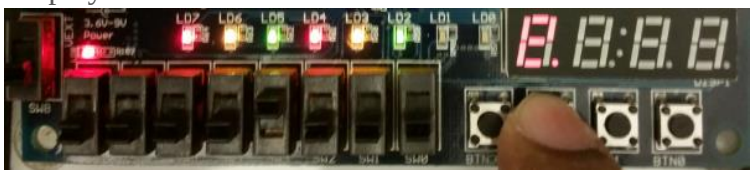
DisplayA show A + B = 4 (100 bin):

DisplayA show A – B = 2 (10 bin):

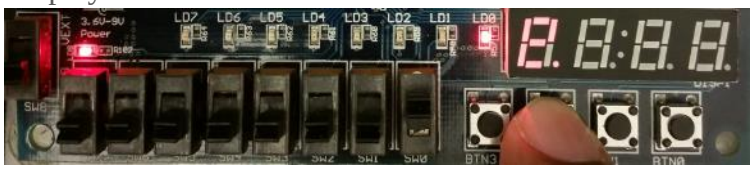
DisplayL show A XOR B = 10 bin



DisplayL show NOT A = 1111100 bin



DisplayL show A AND B = 01 bin



PROBLEMS

The main problem we encountered was in how to utilize the limited amount of input switches and output leds to meet the specifications. We had started our original design thinking we only had 4 bit inputs for our operands, but that was later changed to 8bit input operands. We had to redesign and include the register file in order to implement 8bit input operands. Our previous register file also only had one output for the LEDs to show the contents of the registers. So we had to add data bus outputs for each register inside the register file and send those directly to the ALU for operations.

We also had problems implementing the extender, so we decided not to use an extender and give every operation inside the ALU its own separate components. That also made it easier to build the ALU, since we divided each operation into an encapsulated component.

CONCLUSION

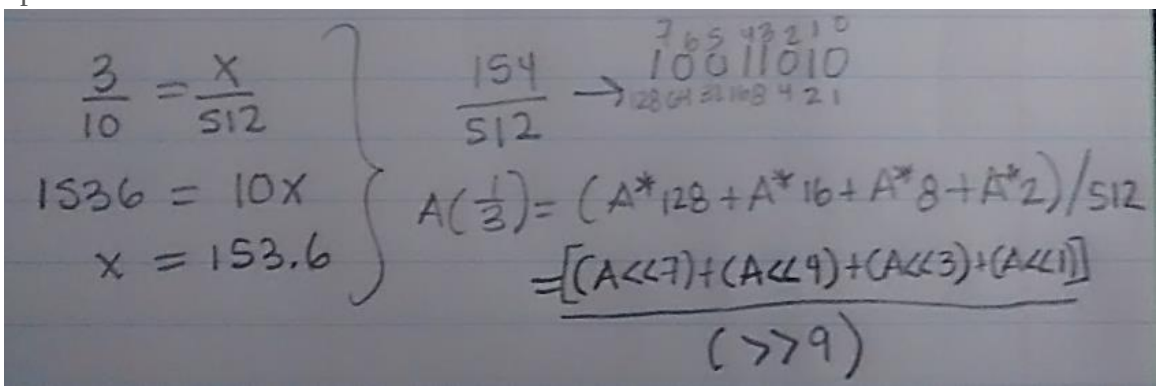
Overall the design meets the specifications of the mini project. Everything works as intended, including negative numbers that are displayed in two's complement if ever an arithmetic operation results in a negative number. One of the ways to improve the design would be by adding the extender to the ALU. That would greatly decrease the number of components inside the ALU. Another way to improve the system would be adding error checking to the Opcode and Data inputs on the switches, so no strange states can be selected.

MINI PROJECT TA QUESTION

If one of the operation is A/3, please describe the design procedure.

The design procedure would be as follows:

We would need to use strength reduction to convert the operation of A/3 to basic shifter operations.



This changes the multiplication of A(1/3) to a series of left shifts and a right shift. We already have a component dedicated to shifting in our design. We only would need to modify it slightly to account for higher magnitudes of left and right shifts.

The overall design of the component that would be included in our ALU would be: Input A, and Output F.