CSC 347 LAB REPORT

Lab #10 - Arithmetic Logic Unit (ALU)

Student Name: Vitaliy Prymak

Date Performed: 04/26/23

Date Submitted: 05/02/23

Objective:

The objectives of this lab are to be further familiar with the behavioral modeling of circuits, to design a 4-bit ALU capable of performing various arithmetic and logic operations, and to use the seven-segment display from the previous lab to display calculation results.

Equipment and Software used:

- *Xilinx*
- *Edaplayground*
- *Board*

Design procedure:

An arithmetic logic unit (ALU) is a combinational circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. The block diagram of the ALU to design in this lab is shown below and can perform eight different operations as shown in the table.

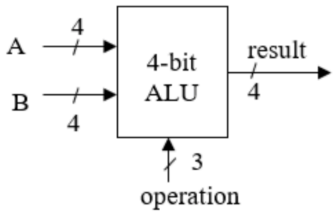

Figure 1 4 bit ALU

operation	result =
0 0 0 (0)	A and B
0 0 1 (1)	A or B
0 1 0 (2)	Not B
0 1 1 (3)	A << B
1 0 0 (4)	A + B
1 0 1 (5)	A – B
1 1 0 (6)	A* B
1 1 1 (7)	A >> B

Table 1 ALU's operations

1. Write a Verilog code to implement the ALU in behavioral level

```
module Alu(A,B,operation,result);

 input [3:0] A, B;
 input [2:0] operation;
 output [3:0]result;

  reg [3:0]result;
  always @ (A, B, operation)
 begin
 case (operation)
 3'b000: result = A&B;
 3'b001: result = A|B;
 3'b010: result = ~B;
 3'b011: result = A<<B;
 3'b100: result = A+B;
 3'b101: result = A-B;
 3'b110: result = A*B ;
 3'b111: result = A^B;
endcase
end
endmodule
```

Figure 2 ALU module in Verilog code

On Figure 1 we can see that ALU has 4 A inputs & 4 B inputs & therefore 4 results. As well as one switch aka operation, which tells what operation will be performed.

2. Write a top level Verilog module to display the ALU result on a 7- segment display from Lab9

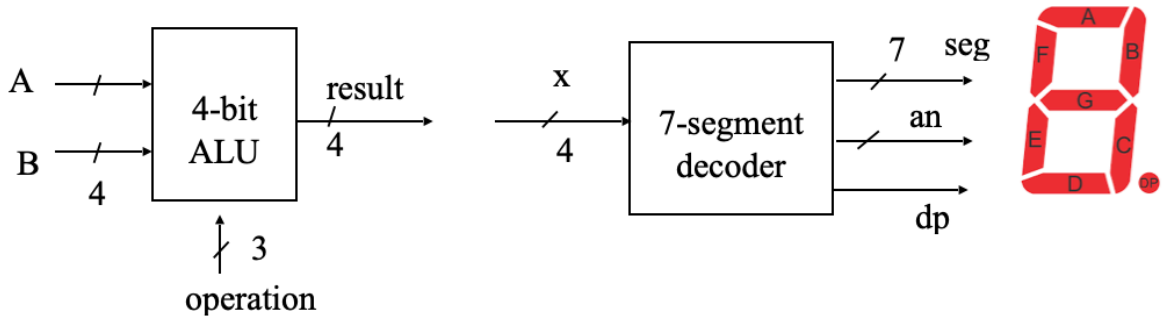

Figure 3 ALU and 7-segment decoder

```
module bin7seg (x, seg,an,dp);
input [3:0] x ; //4-bit input to display
output [6:0] seg; // segments from a to g
output [3:0] an;
output dp; // decimal point

 assign an = 4'b1110;
 assign dp = 1;

reg [6:0] seg; // re-declare seg as the type of reg
always @(x)
case (x)

 0: seg = 7'b0000001;
 1: seg = 7'b1001111;
 2: seg = 7'b0010010;
 3: seg = 7'b0000110;
 4: seg = 7'b1001100;
 5: seg = 7'b0100100;
 6: seg = 7'b0100000;
 7: seg = 7'b0001111;
 8: seg = 7'b0000000;
 9: seg = 7'b0000100;
 'hA: seg = 7'b0001000;
 'hb: seg = 7'b1100000;
 'hC: seg = 7'b0110001;
 'hd: seg = 7'b1000010;
 'hE: seg = 7'b0110000;
 'hF: seg = 7'b0111000;
 default: seg = 7'b0000001;
 endcase
 endmodule
```

Figure 4 7-segment module

On Figure 4 Code represents 7-segment module which implementation is to display a 4-bit binary number. Here dp stands for decimal point but we do not need it so we disable it (dp =1), an is 4 bit enabler for those 4 bits that something is on 0000 or off 1111. And 7 is 0, 1, 2, 3, 4, 5, 6, 7. It use up side down value for 0 will true and 1 is false.

```
module toplevelmodule(A, B, operation, result, seg, an, dp);
input [3:0] A;
input [3:0] B;
input [2:0] operation;
output [3:0] result;
output [0:6] seg; // segments from a to g
output [3:0] an;   //4-bit enable signal
output dp;
```

Figure 5 ALU with display

On Figure 5 is top level module which consist of ALU & 7 segment display, therefore it take all inputs and outputs from ALU and 7 segment and put it inside of paransethis. The output result from ALU pass as input x to 7 segment display.

3. Write a test bench program to test the adder/ subtractor. Fill in the blanks and add all the test cases after the comment "Initialize Inputs". For simplicity, Set A = 6 and B = 2 and Operation from 0 - 7.

It does bitwise logic like 2 && 5 is true && true is true (1)

operation	A = 6	B = 2	Result =	Seg
0 0 0 (0)	0110	0010	0010	2
0 0 1 (1)	0110	0010	0110	6
0 1 0 (2)	0110	0010	1101	d
0 1 1 (3)	0110	0010	1000	8
1 0 0 (4)	0110	0010	1000	8
1 0 1 (5)	0110	0010	0100	4
1 1 0 (6)	0110	0010	1100	C
1 1 1 (7)	0110	0010	0001	1

ALU has 4 bit 4 A inputs & 4 B inputs therefore 4 bit 4 results and operation is 3 bit which will tell ALU which operation to perform.

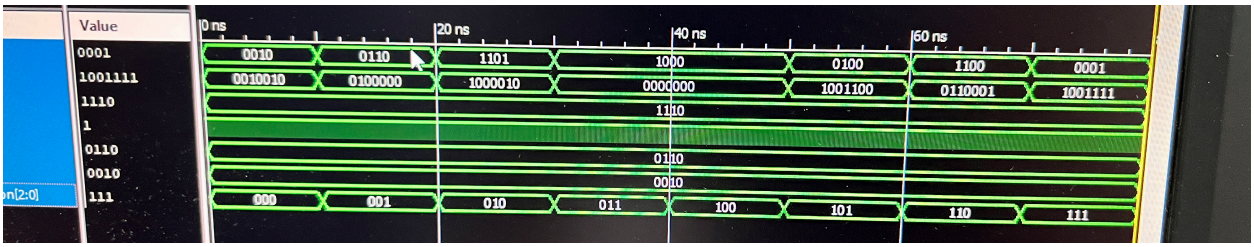

Figure 6 Waive form of ALU & 7 segment

It can be seen that the waveforms match the table 6. We then downloaded our design to the Basys2 circuit board, and tested all the eight possible results by watching the 7-segment display as press each operation on the data switches.

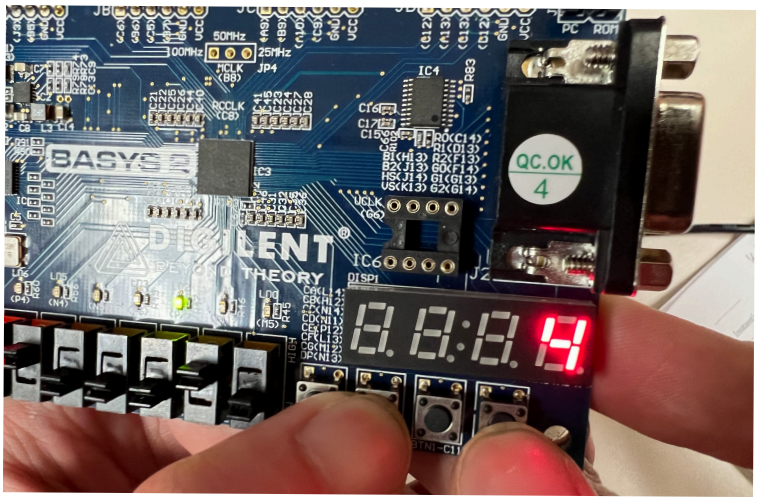

Figure 7 Basys2 circuit board

Conclusion:

I've been able to verify Seg obtained in table 6 match with 7-segment display on Basys2 board while entered each set of test values on the data switches and pressing buttons for corresponding operations.

HOMEWORK:

https://edaplayground.com/x/NUbW

```
1  // Vitaliy Prymak
2
3  // Instantiate add_subtractor
4    module halfadder (S,C,x,y);
5     input x,y;
6     output S,C;
7  //Instantiate primitive gates
8     xor (S,x,y);
9     and (C,x,y);
10 endmodule
11
12 module fulladder (S,C,x,y,cin);
13    input x,y,cin;
14    output S,C;
15    wire S1,D1,D2; //Outputs of first XOR and two AND gates
16 //Instantiate the halfadder
17     halfadder HA1 (S1,D1,x,y);
18     halfadder HA2 (S,D2,S1,cin);
19     or g1(C,D2,D1);
20 endmodule
21
22 module four_bit_adder (S, C4, A, B, C0);
23    input [3:0] A,B;
24    input C0;
25    output [3:0] S;
26    output C4;
27    wire C1,C2, C3;  //Intermediate carries
28 //Instantiate the fulladder
29    fulladder  FA0 (S[0],C1,A[0],B[0],C0);
30    fulladder  FA1 (S[1],C2,A[1],B[1],C1);
31    fulladder  FA2 (S[2],C3,A[2],B[2],C2);
32    fulladder  FA3 (S[3],C4,A[3],B[3],C3);
33
34 endmodule
```

```
36
37 module adder_subtractor(S, C, A, B, M);
38    input [3:0] A,B;
39    input M;
40    output [3:0] S;
41    output C;
42
43    wire [3:0] D;  //Output of XOR gates
44
45     xor U0(D[0], B[0], M);
46     xor U1(D[1], B[1], M);
47     xor U2(D[2], B[2], M);
48     xor U3(D[3], B[3], M);
49
50     four_bit_adder U4(S, C, A, D, M);
51
52 endmodule
53
54 // Instantiate mux4x1
55   module mux4x1(i0, i1, i2, i3, select, y);
56
57     input [3:0] i0,i1,i2,i3;
58     input [1:0] select;
59     output [3:0] y;
60     reg [3:0] y; //
61     always @ (i0 or i1 or i2 or i3 or select)
62       |
63       case (select)
64         2'b00: y = i0;
65         2'b01: y = i1;
66         2'b10: y = i2;
67         2'b11: y = i3;
68       endcase
69   endmodule
```

```
70
71 module Alu(A, B, operation, result);
72
73 //inputs and outputs
74   input [1:0] operation;
75   input [3:0] A,B;
76   output [3:0] result;
77
78   // Instantiate AND gates and OR gates
79   wire [3:0] andOutput, orOutput, S;
80   and andGate0(andOutput[0], A[0], B[0]);
81   and andGate1(andOutput[1], A[1], B[1]);
82   and andGate2(andOutput[2], A[2], B[2]);
83   and andGate3(andOutput[3], A[3], B[3]);
84
85   or orGate0(orOutput[0], A[0], B[0]);
86   or orGate1(orOutput[1], A[1], B[1]);
87   or orGate2(orOutput[2], A[2], B[2]);
88   or orGate3(orOutput[3], A[3], B[3]);
89
90   // Assign operation[0] to M
91   wire M;
92   assign M=operation[0]; // can i do just M=operation[0];
93
94   // Instantiate add_subtractor
95   adder_subtractor AS1(S, C, A, B, M);
96
97   // Instantiate mux4x1
98   mux4x1 Mx1(S, S, andOutput, orOutput, operation, result);
99
100  endmodule
```

SV/Verilog Testbench

```
// Vitaliy Prymak
module test;

  reg [3:0] A,B;
  reg [1:0] operation;


  wire  [3:0] result;

  // Instantiate the unit under test for top level module
  Alu uut(A, B, operation, result);
    // we do not wright module before Alu and write uut after Alu

  initial
    begin
      $dumpfile("dump.vcd");
      $dumpvars(1,test);

      // display inputs and outpus
      $display("Operation 0 = A + B");
      $display("Opeartion 1 = A - B");
      $display("Opeartion 2 = A & B");
      $display("Operation 3 = A | B");
      $monitor("operation = %d, A = %b, B = %b, result = %b", operation, A, B, result);

      // initialize inputs
      A = 6; B= 2; operation = 0;
      #2 A = 6; B = 2; operation = 1;
      #2 A = 6; B = 2; operation = 2;
      #2 A = 6; B = 2; operation = 3;

      #2 $finish;
    end
endmodule
```

```
Operation 0 = A + B
Opeartion 1 = A - B
Opeartion 2 = A & B
Operation 3 = A | B
operation = 0, A = 0110, B = 0010, result = 1000
operation = 1, A = 0110, B = 0010, result = 0100
operation = 2, A = 0110, B = 0010, result = 0010
operation = 3, A = 0110, B = 0010, result = 0110
```