CSC 347/ENS 211

Title and Experiment #	Lab9: ALU and Seven-Segment Display
Name	Gianna Galard
Date Performed	16-Nov-21
Date Submitted	17-Nov-21

The student pledges this work to be their own *Gianna Galard*

Link: https://www.edaplayground.com/x/BsXs

Objective:

The objective of this week's lab was to become familiar with the behavioral modeling of circuits by designing a 4-bit ALU capable of performing various arithmetic and logic operations and a seven-segment display decoder to display the results.

Language and Compiler used:

- Verilog
- Edaplayground.com

Design Procedure:

Figure 1. 4-bit ALU system

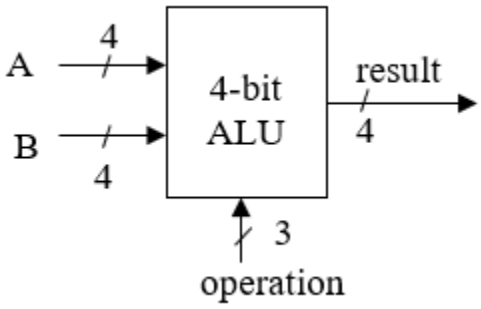


Figure 2. Module ALU

```
module Alu(A,B,operation,result);
  input [3:0] A,B;
  input [2:0] operation;
  output [3:0] result;
  reg [3:0] result; // redeclare the signals that appear on the left-hand side of the
                    // assignment statements inside the always block
  always @(A,B,operation)
    begin
      case(operation)
        3'b000: result = A & B;
        3'b001: result = A | B;
        3'b010: result = ~B; //  bit-wise operator and returns the invert of the argument
                             // use ! for if true or false of single bit
        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
```

For the ALU Design shown in figure 1, 3 inputs and 1 output were passed. The inputs were two 4 bit numbers, A and B, and one 3-bit number called “operation.” Operation signaled to the decoder, which is an “always” Verilog statement, what to do with A and B and then assign the result of what happens with A and B to the “result” variable. The

3-bit nature of the “operation” variable means up to (2^3) 8 different operations to signal what to do in the case structure. In figure 2, notice that we had to redeclare the result; the result is on the left side of an assignment statement. When using an always block, you must redeclare everything on the left side of an assignment state as a variable.

Figure 3. Module bin7seg

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

  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;
      10: seg = 7'b0001000;
      11: seg = 7'b1100000;
      12: seg = 7'b0110001;
      13: seg = 7'b1000010;
      14: seg = 7'b0110000;
      15: seg = 7'b0111000;
      default: seg = 7'b1111110;
    endcase
endmodule
```

The bin7seg module is a display driver to display a 4-bit binary number. We used given diagrams that work with active low outputs, which means that 1=OFF and 0=ON. For example, If a = 0, then the top bar would not show.

Figure 4. Module toplevelmodule

```
59 module toplevelmodule(A,B,operation, seg, dp);
60 // result and x get tied together internally , make it a wire
61   input [3:0] A, B;
62   input [2:0] operation;
63   wire [3:0] result;
64   output [6:0] seg;
65   output dp;
66
67   // turn off decimal point
68   // active low -> 1 not 0
69   assign dp = 1;
```

The toplevelmodule is shown in figure 4, and it is what calls both the Alu and bin7seg. Therefore it must take in all the parameters as inputs that those two circuits require. The result parameter is passed through the ALU instantiation before being passed in as the x variable that bin7seg takes.

Figure 5. testbench.sv

SV/Verilog Testbench

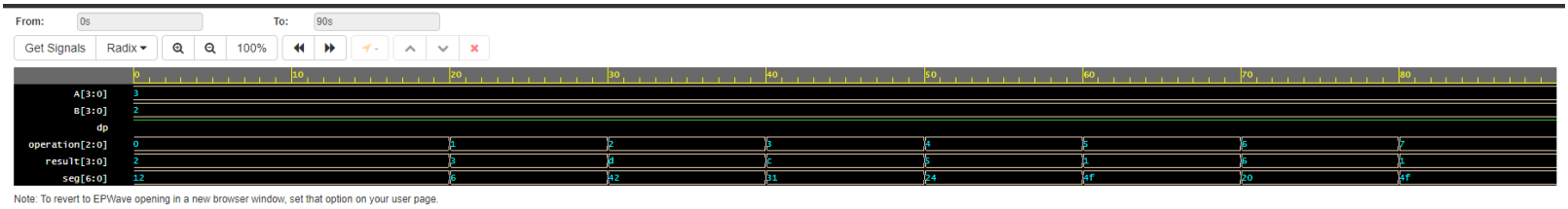
```
// Author : Gianna Rose
// Lab 9
// https://www.edaplayground.com/x/BsXs

module test;
  // inputs
  reg [3:0] A,B;
  reg [2:0] operation;
  // outputs
  wire [3:0] result;
  wire [6:0] seg;
  wire dp;

  // instantiate the ALU
  Alu uut0(A, B, operation, result);

  // instantiate toplevelmodule
  toplevelmodule uut1(A,B,operation, seg, dp);

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

      // display the inputs and outputs
      $monitor("operation = %b A = %d B = %d  Result = %d seg = %b", operation, A,B, result, seg);

      // initialize inputs
      operation = 0; A = 3; B = 2;
      for(int i = 0; i < 8; i = i + 1) begin
        #10 operation = i; end
      #10 $finish;
    end

// OUTPUT
// operation = 000 A =  3 B =  2  Result =  2 seg = 0010010
// operation = 001 A =  3 B =  2  Result =  3 seg = 0000110
// operation = 010 A =  3 B =  2  Result = 13 seg = 1000010
// operation = 011 A =  3 B =  2  Result = 12 seg = 0110001
// operation = 100 A =  3 B =  2  Result =  5 seg = 0100100
// operation = 101 A =  3 B =  2  Result =  1 seg = 1001111
// operation = 110 A =  3 B =  2  Result =  6 seg = 0100000
// operation = 111 A =  3 B =  2  Result =  1 seg = 1001111

endmodule
```

A and B need to be 4-bit inputs which will be 3 and 2, respectively. Operation is the 3-bit input that will tell the ALU what operation to perform with A and B and assign to the 4-bit result variable. The 7-bit seg variable gets calculated from the toplevelmodule that will call

bin7seg, which will run its result through a case statement and assign seg depending on the result from the ALU.

Conclusion :

```
// OUTPUT
// operation = 000 A =  3 B =  2  Result =  2 seg = 0010010
// operation = 001 A =  3 B =  2  Result =  3 seg = 0000110
// operation = 010 A =  3 B =  2  Result = 13 seg = 1000010
// operation = 011 A =  3 B =  2  Result = 12 seg = 0110001
// operation = 100 A =  3 B =  2  Result =  5 seg = 0100100
// operation = 101 A =  3 B =  2  Result =  1 seg = 1001111
// operation = 110 A =  3 B =  2  Result =  6 seg = 0100000
// operation = 111 A =  3 B =  2  Result =  1 seg = 1001111
```



Homework:

Link: https://www.edaplayground.com/x/Bh4i

design.sv

```
// Author: Gianna Rose
// Lab 9 HW
// https://www.edaplayground.com/x/Bh4i

module fulladder (S, C, x, y, cin);
   input x, y, cin;
   output S, C;

   //internal signals
   wire S1, D1, D2;

   //Instantiate the half adders
    halfadder    HA1 (S1, D1, x, y);
    halfadder    HA2 (S, D2, S1, cin);
    or U3(C, D2, D1);

endmodule

module halfadder (S, C, x, y);
   input x, y;
   output S, C;

//Instantiate primitive gates
   xor U1(S, x, y);
   and U2(C, x, y);
```

```
endmodule

module four_bit_adder (S, C4, A, B, Cin);
   input [3:0] A,B;
   input Cin;
   output [3:0] S;
   output C4;


//Declare intermediate carries
 wire C1, C2, C3;

//Instantiate the fulladder
  fulladder FA0(S[0], C1, A[0], B[0], Cin);
  fulladder FA1(S[1], C2, A[1], B[1], C1);
  fulladder FA2(S[2], C3, A[2], B[2], C2);
  fulladder FA3(S[3], C4, A[3], B[3], C3);

endmodule

module adder_subtractor(S, C, A, B, M);
   input [3:0] A,B;
   input M;
   output [3:0] S;
   output C;

   //Declare outputs of XOR gates
  wire [3:0]N;

  // Instantiate the XOR gates
  xor XOR0(N[0],B[0], M);
  xor XOR1(N[1],B[1], M);
  xor XOR2(N[2],B[2], M);
  xor XOR3(N[3],B[3], M);
```

```
  // Declare carry
  wire C4;

  // Instantiate the 4-bit full adder
  four_bit_adder FBA(S, C4, A, N, M);

endmodule

module mux4x1(i0, i1, i2, i3, select, y);
   input [3:0] i0,i1,i2,i3;
   input [1:0] select;
   output [3:0] y;
   reg [3:0] y;
   always @ (i0 or i1 or i2 or i3 or select)
          case (select)
            2'b00: y = i0;
            2'b01: y = i1;
            2'b10: y = i2;
            2'b11: y = i3;
          endcase
endmodule

module Alu(A, B, operation, result);
  // inputs and outputs
  input [1:0] operation;
  input [3:0] A, B;
  output [3:0] result;

  // wires store S and the outputs of AND/OR gates
  wire [3:0] S, AND, OR;
  wire C4, M;

  // instantiate AND gate and OR gate
```

```
  assign AND = A & {B[3],B[2],B[1],B[0]};
  assign OR  = A | {B[3],B[2],B[1],B[0]};

  // connect M operation[0]
  assign M = operation[0];

  // instantiate add_subtractor
  // module adder_subtractor(S, C, A, B, M);
  adder_subtractor AS(S, C4, A, B, M);

  // instantiate mux4x1
  // module mux4x1(i0, i1, i2, i3, select, y);
  mux4x1 MUX(S, S, AND, OR, operation, result);

  endmodule
```

testbench.sv

```
// Author: Gianna Rose
// Lab 9 HW
// https://www.edaplayground.com/x/Bh4i

module test;
  // inputs
  reg [3:0] A, B;
  reg [1:0] operation;
  // outputs
  wire [3:0] result;

  // nstantiate the Alu
  Alu uut(A, B, operation, result);

  initial
```

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

      // display the inputs and outputs
      $monitor( "operation = %b  A =%d  B =%d Result =%d",
operation, A, B, result);

      // initialize inputs
      operation = 0; A = 3; B = 2;
      for(int i = 0; i < 8; i = i + 1) begin
        #10 operation = i; end
      #10 $finish;
    end

  // OUTPUT
// operation = 00  A = 3  B = 2 Result = 5
// operation = 01  A = 3  B = 2 Result = 1
// operation = 10  A = 3  B = 2 Result = 2
// operation = 11  A = 3  B = 2 Result = 3
// operation = 00  A = 3  B = 2 Result = 5
// operation = 01  A = 3  B = 2 Result = 1
// operation = 10  A = 3  B = 2 Result = 2
// operation = 11  A = 3  B = 2 Result = 3

  endmodule
```