Emre YILMAZ – 1901042606 – Gebze Technical University

CSE331 – Homework 2

Initial note: I set the ALU as the top level entity. If you want to try it first, you must only open the alutest.vwf file after compilation OR you can create a new simulation file. If you want to test multiplier module, firstly you must set the multiplier.v module as the top level entity. After that you can compile and open the datapath_test.vwf file OR you can create a new simulation file. Multiplier module runs correct, however I cannot set the clock and it does not work in ALU. If you run the multiplier.v module alone, you will see that it runs correctly. I showed the screenshots of multiplier in this PDF. You can check them and test them.

Regards.

PART – 1

I developed the ***Control Unit*** with help of Gizem Sungu. I modify a little bit her Control Unit that is made in PS Session. Its FSM logic is the same as PDF. I only made little changes.

State0 -> Write=0, Add=0, Shift=0, if product[0]==0 then Next State is State 1, else Next State is State 2

State1 -> Write=1, Add=1, Shift=0, Next State = State2

State2 -> Write=0, Add=0, Shift=1, Next State = State3

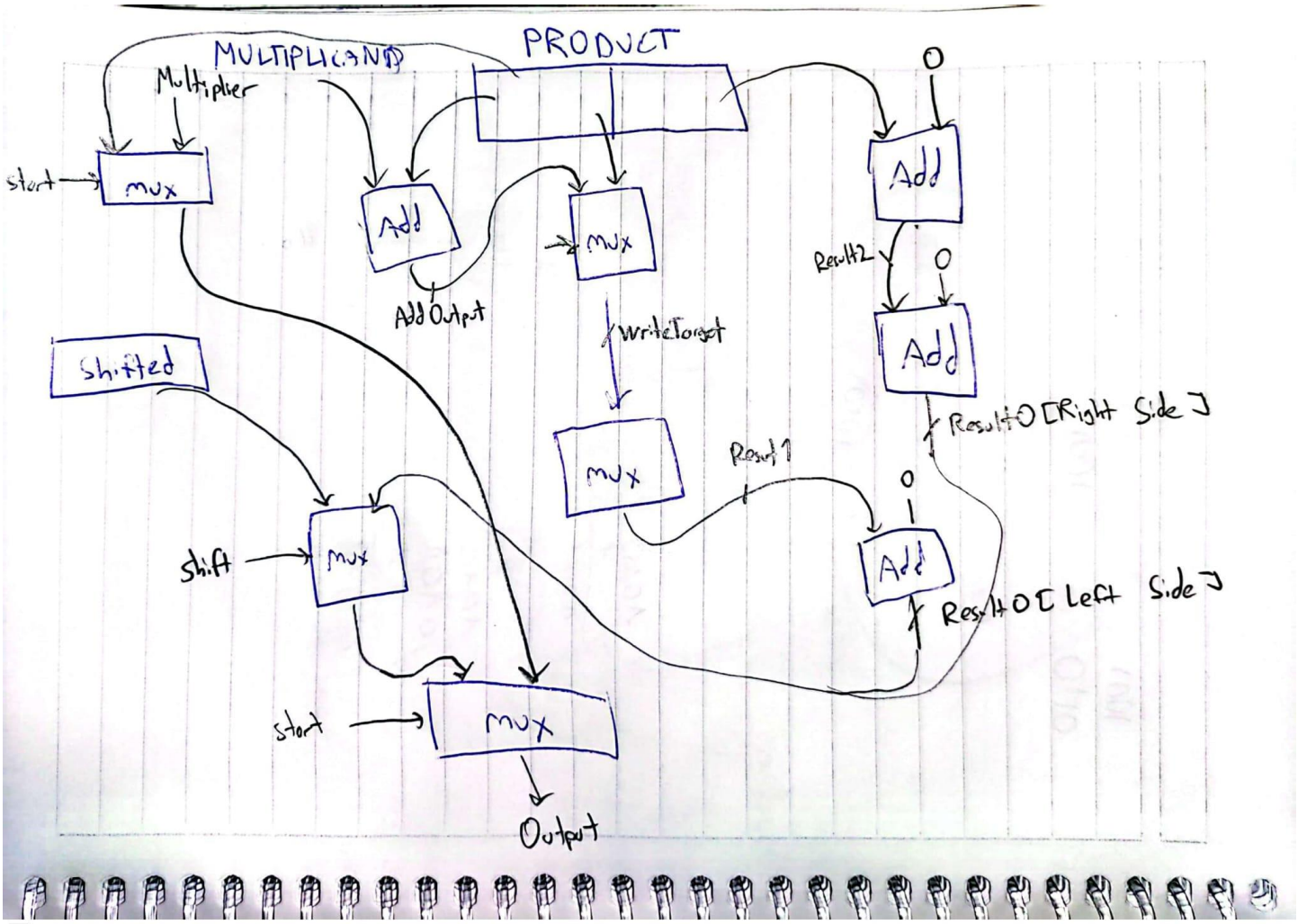
State3 -> Write0, Add=0, Shift=0, if we are done Next State is S4, else Next State is S0

Outputs of the control unit:

```
//Outputs
always @(posedge CLK)
      begin
         case(current_state)

            S0:
               begin
               write = 1'b0;
               shift = 1'b0;
               add = 1'b0;
               end
            S1:
               begin
               write = 1'b1;
               shift = 1'b0;
               add = 1'b1;
               end


            S2:
               begin
               write = 1'b0;
               shift = 1'b1;
               add = 1'b0;
               end


            S3:
               begin
               write = 1'b0;
               shift = 1'b0;
               add = 1'b0;
               end


         endcase

      end
```

Datapath:

Firstly, I must explain the structural part:

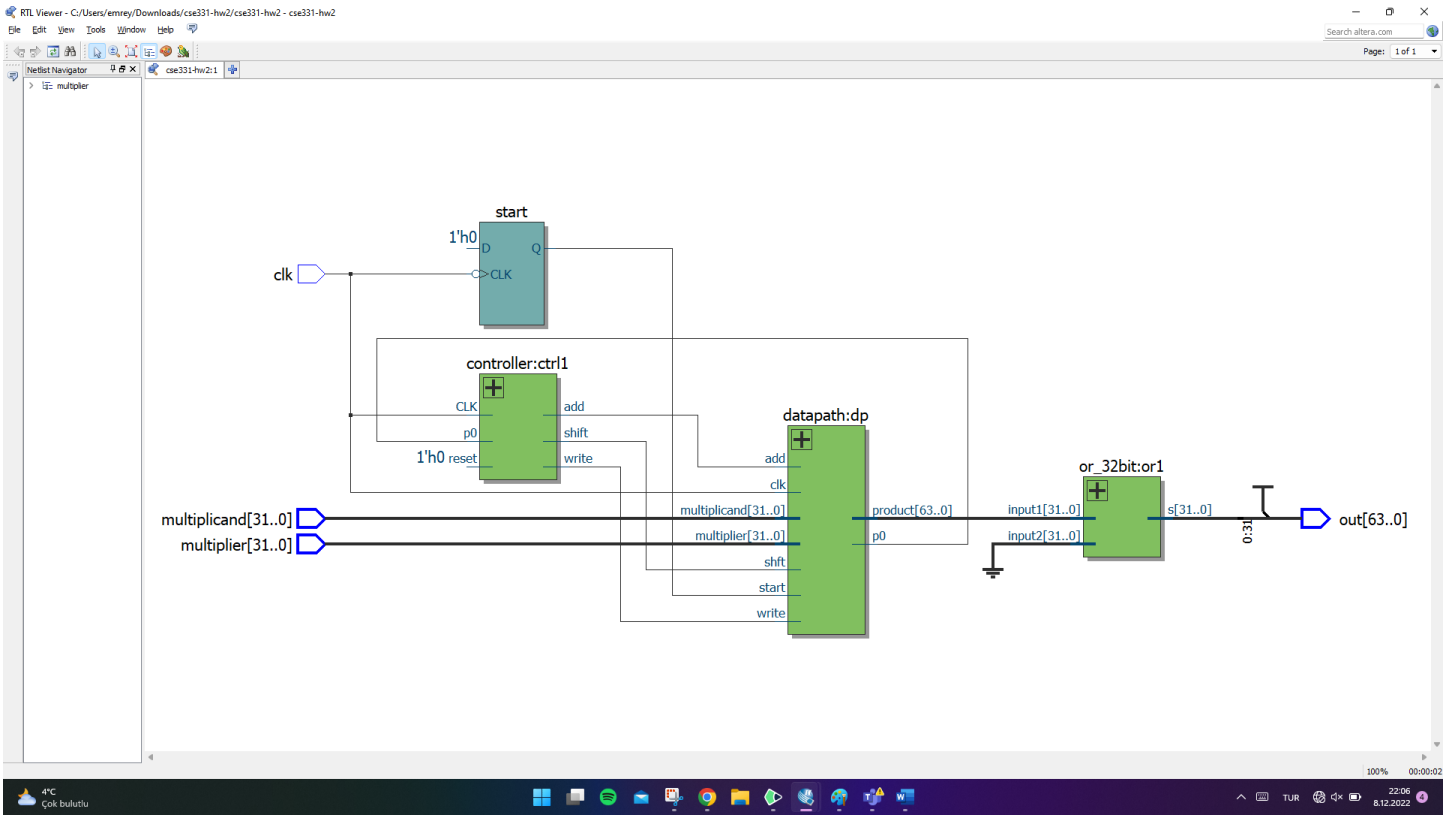


I tried to implement the datapath that I wrote above. I use several wires. These wires correspond the add result, shift result, multiplier result etc. If you look at the code, I do all the operations and connect them multiplexers. The first input of final multiplexer above is multiplicand+product[63:32] or the product[63:32] itself. It is indicated by the signals that connects to the multiplexer. There is a critical point above. The selection input of mux at the left-corner of the page is only 1 time. In this case it copies the multiplier to the product's right half. I use 5 multiplexer and 4 adders in the datapath. Result0 corresponds the calculation result (product itself or the result of addition). It is connected to the final multiplexer.

I tried to develop the datapath as structural. But I do not know why, signals come from control unit was always 0. So, I had to use behavioural Verilog. I add behavioural lines to the datapath and I connect the output to the behavioural part. I check the clock and go to always block. Inside it, if the start signal comes, I got the multiplier to the Product initially. After that start signal will be always 0, (this signal will be done 0 in the Multiplier module.) Then, I check the signals. If write and add signals come, I update the Product with the add result. If shift signal comes, I shift the product 1 bit. That is all. The trick here is start signal. Initially we must copy the multiplier to the left side of the product. It was important. At the end of the calculation result was the half of the actual result. So, I had to shift it 1 bit.

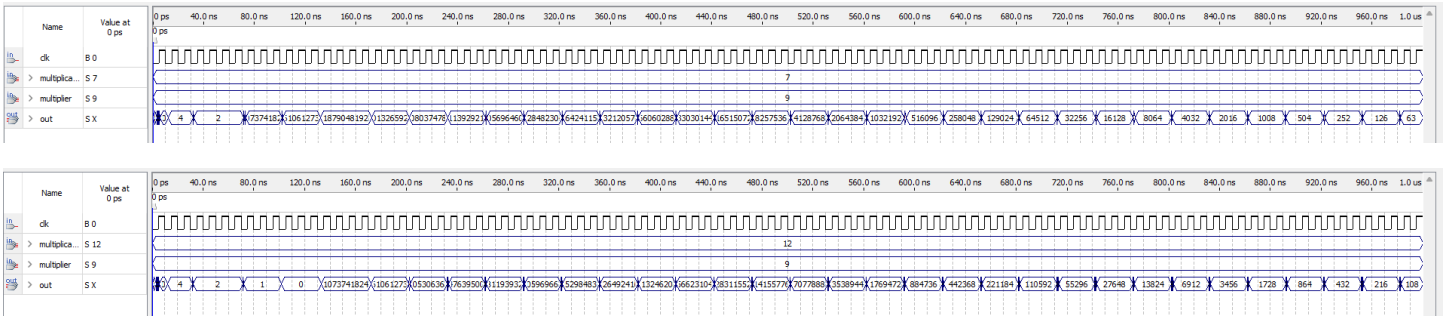
Multiplier: It was simple module. I set the start signal as initially 1 then 0. In this way we could start the multiplication calculation by copying the multiplier to the product's left side. This module calls the control unit, after that it connects the outputs of the control unit to the datapath.

This is the design of the multiplication module. The module on the right fixes the result that is comes from datapath wrong. You can create this simulation using RTL viewer.



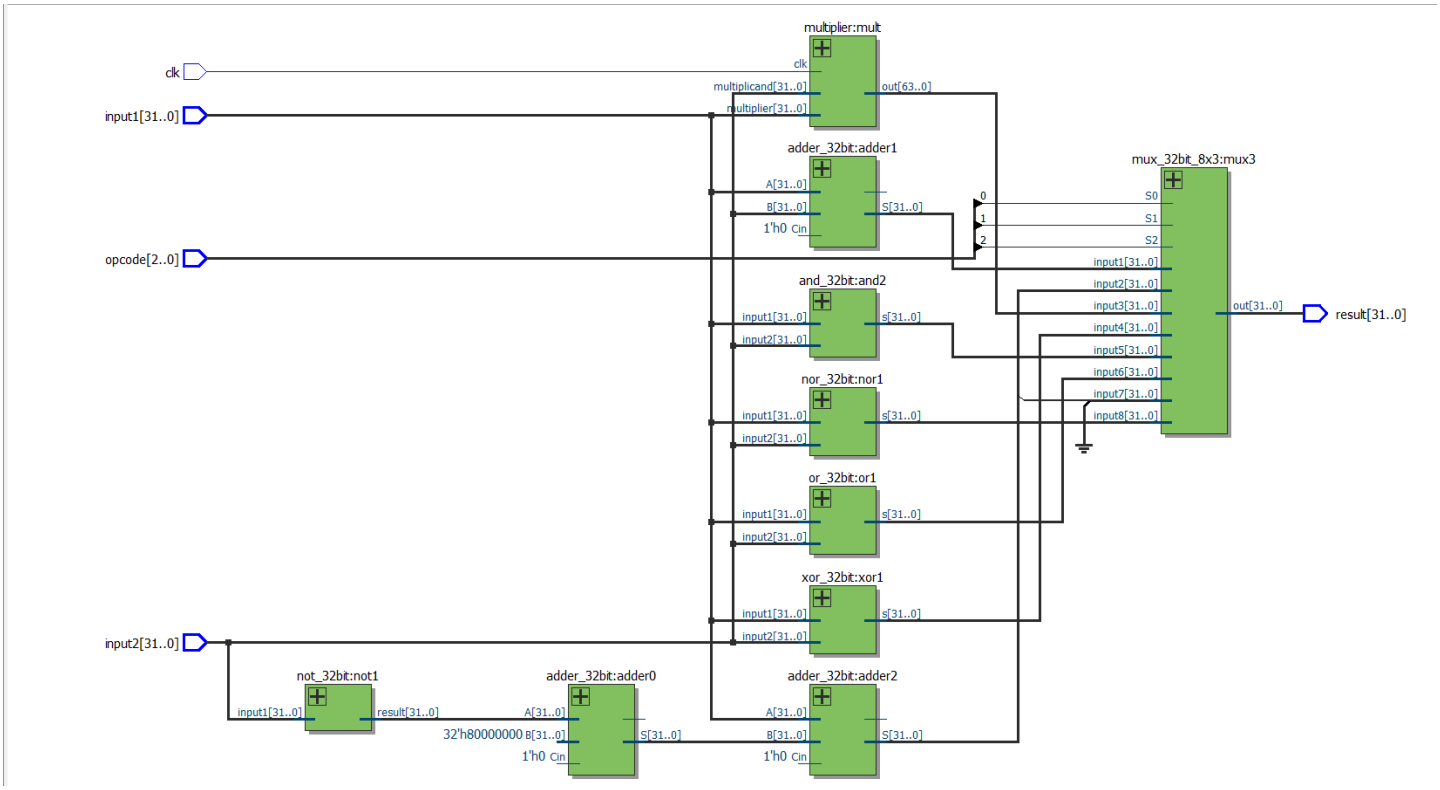
The adder and multiplexer that are used in the first part of homework will be explained in the PART – 2.

TEST OF PART – 1



The test file named datapath_test is included by the zip. You can test it yourself. But, If you want to test the PART-1 you have to make multiplier.v ***top level entity.*** I am sorry about that; I had no time to fix it.

PART – 2



RTL design of the ALU module. You can create this simulation using RTL viewer.

I want to begin with small modules of the main module. I used

nor_32bit,

or_32bit,

not_32bit

and_32bit modules.

All of them are simple modules. They contain inbuilt functions “or, nor, not, and” 32 times. In every step, they send the function next bit of inputs and next bit of output. You do not have to see all of them, but I am going to show you one of them:

Nor_32bit module:

```
module nor_32bit(input [31:0] input1, input [31:0] input2, output [31:0] s);

nor nor1(s[0], input1[0], input2[0]);
nor nor2(s[1], input1[1], input2[1]);
nor nor3(s[2], input1[2], input2[2]);
nor nor4(s[3], input1[3], input2[3]);
nor nor5(s[4], input1[4], input2[4]);
nor nor6(s[5], input1[5], input2[5]);
nor nor7(s[6], input1[6], input2[6]);
nor nor8(s[7], input1[7], input2[7]);
nor nor9(s[8], input1[8], input2[8]);
nor nor10(s[9], input1[9], input2[9]);
nor nor11(s[10], input1[10], input2[10]);
nor nor12(s[11], input1[11], input2[11]);
nor nor13(s[12], input1[12], input2[12]);
nor nor14(s[13], input1[13], input2[13]);
nor nor15(s[14], input1[14], input2[14]);
nor nor16(s[15], input1[15], input2[15]);
nor nor17(s[16], input1[16], input2[16]);
nor nor18(s[17], input1[17], input2[17]);
nor nor19(s[18], input1[18], input2[18]);
nor nor20(s[19], input1[19], input2[19]);
nor nor21(s[20], input1[20], input2[20]);
nor nor22(s[21], input1[21], input2[21]);
nor nor23(s[22], input1[22], input2[22]);
nor nor24(s[23], input1[23], input2[23]);
nor nor25(s[24], input1[24], input2[24]);
nor nor26(s[25], input1[25], input2[25]);
nor nor27(s[26], input1[26], input2[26]);
nor nor28(s[27], input1[27], input2[27]);
nor nor29(s[28], input1[28], input2[28]);
nor nor30(s[29], input1[29], input2[29]);
nor nor31(s[30], input1[30], input2[30]);
nor nor32(s[31], input1[31], input2[31]);

endmodule
```

Next modules I must explain are ***adder*** modules. Adder_32bit makes addition with 32-bit inputs. 32-bit adder uses 16-bit adder, 16-bit adder uses 4-bit adder, and 4-bit adder uses full_adder modules. I am going to show all of them. Full adder is simple as you know as a logic circuit. 32-bit adder divide into two the inputs and outputs, then send to 16-bit adder. All of them makes same thing until full adder.

```
module adder_32bit(input [31:0] A, input [31:0] B, input Cin, output [31:0] S, output Cout);

wire[3:0] firstCarry;

adder_16bit adder1(A[31:16], B[31:16], Cin, S[31:16], firstCarry[0]);
adder_16bit adder2(A[15:00], B[15:0], firstCarry[0], S[15:0], Cout);


endmodule
```

```
module adder_16bit(input [15:0] A, input [15:0] B, input Cin, output [15:0] S, output Cout);

wire[3:0] firstCarry;

adder_4bit adder1(A[3:0], B[3:0], Cin, S[3:0], firstCarry[0]);
adder_4bit adder2(A[7:4], B[7:4], firstCarry[0], S[7:4], firstCarry[1]);
adder_4bit adder3(A[11:8], B[11:8], firstCarry[1], S[11:8], firstCarry[2]);
adder_4bit adder4(A[15:12], B[15:12], firstCarry[2], S[15:12], Cout);

endmodule
```

```
 module adder_4bit(input [3:0] A, input [3:0] B, input Cin, output [3:0] S, output Cout);

 wire[3:0] carryWire;

 full_adder full1(A[0], B[0], Cin, S[0], carryWire[0]);
 full_adder full2(A[1], B[1], carryWire[0], S[1], carryWire[1]);
 full_adder full3(A[2], B[2], carryWire[1], S[2], carryWire[2]);
 full_adder full4(A[3], B[3], carryWire[2], S[3], Cout);

 endmodule
```

```
module full_adder(input A, B, Cin, output s, Cout);

wire w1;
wire w2;
wire w3;

xor xor1(w1, A, B);
xor xor2(s, Cin, w1);
and and1(w2, Cin, w1);
and and2(w3, A, B);
or or1(Cout, w2, w3);

endmodule
```

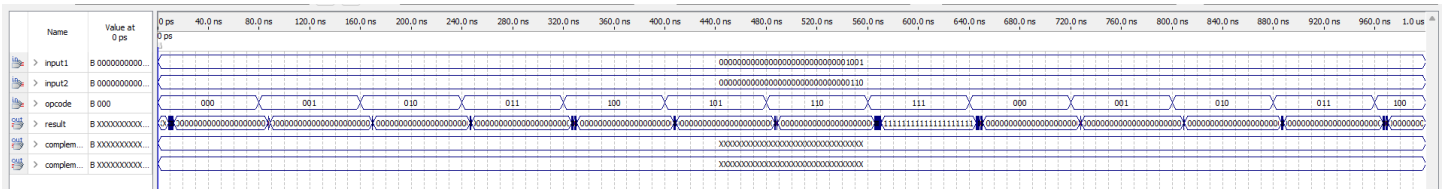
Next modules are ***multiplexer*** modules. In ALU we need 32-bit 8x3 multiplexer. I write a 32-bit multiplexer 8x3 that uses 1-bit 8x3 multiplexer 32 times. It is very simple module. Divides the input 32 parts and send to 1-bit 8x3 multiplexer. Here the images of multiplexers.

```
module mux_32bit_8x3(input [31:0] input1, input [31:0] input2, input [31:0] input3, input [31:0] input4, input [31:0] input5, input [31:0] in
input [31:0] input8, input S0, input S1, input S2, output [31:0] out);

mux_1bit_8x3 mux1(input1[0], input2[0], input3[0], input4[0], input5[0], input6[0], input7[0], input8[0], S0, S1, S2, out[0]);
mux_1bit_8x3 mux2(input1[1], input2[1], input3[1], input4[1], input5[1], input6[1], input7[1], input8[1], S0, S1, S2, out[1]);
mux_1bit_8x3 mux3(input1[2], input2[2], input3[2], input4[2], input5[2], input6[2], input7[2], input8[2], S0, S1, S2, out[2]);
mux_1bit_8x3 mux4(input1[3], input2[3], input3[3], input4[3], input5[3], input6[3], input7[3], input8[3], S0, S1, S2, out[3]);
mux_1bit_8x3 mux5(input1[4], input2[4], input3[4], input4[4], input5[4], input6[4], input7[4], input8[4], S0, S1, S2, out[4]);
mux_1bit_8x3 mux6(input1[5], input2[5], input3[5], input4[5], input5[5], input6[5], input7[5], input8[5], S0, S1, S2, out[5]);
mux_1bit_8x3 mux7(input1[6], input2[6], input3[6], input4[6], input5[6], input6[6], input7[6], input8[6], S0, S1, S2, out[6]);
mux_1bit_8x3 mux8(input1[7], input2[7], input3[7], input4[7], input5[7], input6[7], input7[7], input8[7], S0, S1, S2, out[7]);
mux_1bit_8x3 mux9(input1[8], input2[8], input3[8], input4[8], input5[8], input6[8], input7[8], input8[8], S0, S1, S2, out[8]);
mux_1bit_8x3 mux10(input1[9], input2[9], input3[9], input4[9], input5[9], input6[9], input7[9], input8[9], S0, S1, S2, out[9]);
mux_1bit_8x3 mux11(input1[10], input2[10], input3[10], input4[10], input5[10], input6[10], input7[10], input8[10], S0, S1, S2, out[10]);
mux_1bit_8x3 mux12(input1[11], input2[11], input3[11], input4[11], input5[11], input6[11], input7[11], input8[11], S0, S1, S2, out[11]);
mux_1bit_8x3 mux13(input1[12], input2[12], input3[12], input4[12], input5[12], input6[12], input7[12], input8[12], S0, S1, S2, out[12]);
mux_1bit_8x3 mux14(input1[13], input2[13], input3[13], input4[13], input5[13], input6[13], input7[13], input8[13], S0, S1, S2, out[13]);
mux_1bit_8x3 mux15(input1[14], input2[14], input3[14], input4[14], input5[14], input6[14], input7[14], input8[14], S0, S1, S2, out[14]);
mux_1bit_8x3 mux16(input1[15], input2[15], input3[15], input4[15], input5[15], input6[15], input7[15], input8[15], S0, S1, S2, out[15]);
mux_1bit_8x3 mux17(input1[16], input2[16], input3[16], input4[16], input5[16], input6[16], input7[16], input8[16], S0, S1, S2, out[16]);
mux_1bit_8x3 mux18(input1[17], input2[17], input3[17], input4[17], input5[17], input6[17], input7[17], input8[17], S0, S1, S2, out[17]);
mux_1bit_8x3 mux19(input1[18], input2[18], input3[18], input4[18], input5[18], input6[18], input7[18], input8[18], S0, S1, S2, out[18]);
mux_1bit_8x3 mux20(input1[19], input2[19], input3[19], input4[19], input5[19], input6[19], input7[19], input8[19], S0, S1, S2, out[19]);
mux_1bit_8x3 mux21(input1[20], input2[20], input3[20], input4[20], input5[20], input6[20], input7[20], input8[20], S0, S1, S2, out[20]);
mux_1bit_8x3 mux22(input1[21], input2[21], input3[21], input4[21], input5[21], input6[21], input7[21], input8[21], S0, S1, S2, out[21]);
mux_1bit_8x3 mux23(input1[22], input2[22], input3[22], input4[22], input5[22], input6[22], input7[22], input8[22], S0, S1, S2, out[22]);
mux_1bit_8x3 mux24(input1[23], input2[23], input3[23], input4[23], input5[23], input6[23], input7[23], input8[23], S0, S1, S2, out[23]);
mux_1bit_8x3 mux25(input1[24], input2[24], input3[24], input4[24], input5[24], input6[24], input7[24], input8[24], S0, S1, S2, out[24]);
mux_1bit_8x3 mux26(input1[25], input2[25], input3[25], input4[25], input5[25], input6[25], input7[25], input8[25], S0, S1, S2, out[25]);
mux_1bit_8x3 mux27(input1[26], input2[26], input3[26], input4[26], input5[26], input6[26], input7[26], input8[26], S0, S1, S2, out[26]);
mux_1bit_8x3 mux28(input1[27], input2[27], input3[27], input4[27], input5[27], input6[27], input7[27], input8[27], S0, S1, S2, out[27]);
mux_1bit_8x3 mux29(input1[28], input2[28], input3[28], input4[28], input5[28], input6[28], input7[28], input8[28], S0, S1, S2, out[28]);
mux_1bit_8x3 mux30(input1[29], input2[29], input3[29], input4[29], input5[29], input6[29], input7[29], input8[29], S0, S1, S2, out[29]);
mux_1bit_8x3 mux31(input1[30], input2[30], input3[30], input4[30], input5[30], input6[30], input7[30], input8[30], S0, S1, S2, out[30]);
mux_1bit_8x3 mux32(input1[31], input2[31], input3[31], input4[31], input5[31], input6[31], input7[31], input8[31], S0, S1, S2, out[31]);

endmodule
```

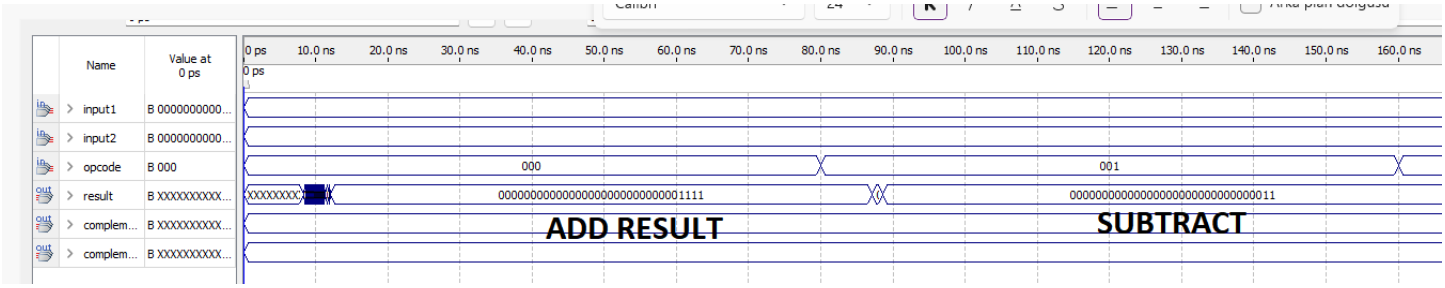
Now, we can dive into ALU. ALU has 3-bit opcode input, 32-bit input1 and 32-bit input2. Firstly, I declared 8 wires to connect the multiplexer. All of them must carry the result of an operation. W1 is simple since it is adder. Only I call the 32-bit adder module to calculation then connect the W1. W2 is subtraction. To subtract, I do a not operation on input2 to get its complement. After that I add it 1 using 32-bit adder. Then I add it with input1 and the subtraction calculation is ready. I connect it to W2. W3 is multiplication operation. It uses the module in part1. W4, W5, W6 is very simple. They use xor_32bit, and_32bit and or_32bit modules to make the calculations. W8 is nor operation, so it uses nor_32bit module. W7 is set-less-than output. To calculate it, I use the subtraction operation. If subtraction result is negative, it means that the input1 is less than input2. So W7 must be 1 in this case. To get this result, I apply a and operation on most significant bit of subtraction result with 1'b1. The result represents the set less than operation. Part 2 was simple. Now, I will show a result screenshot of part 2.

TEST OF PART 2

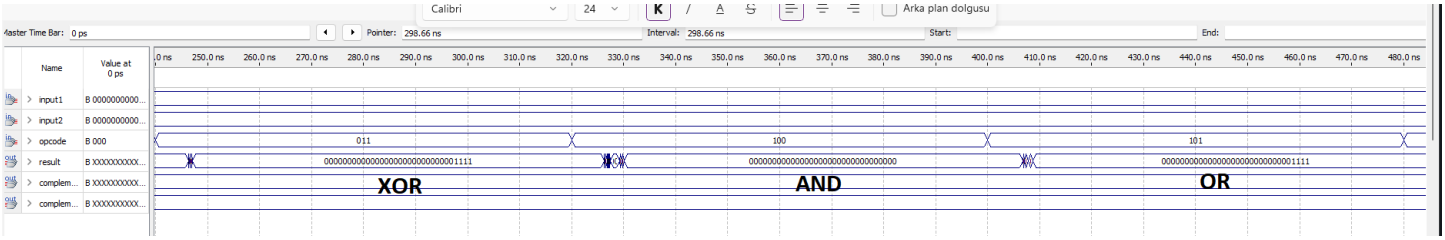
This picture above is test case. I give input as 1001=9 and 0110=6 and the result screenshots are below:



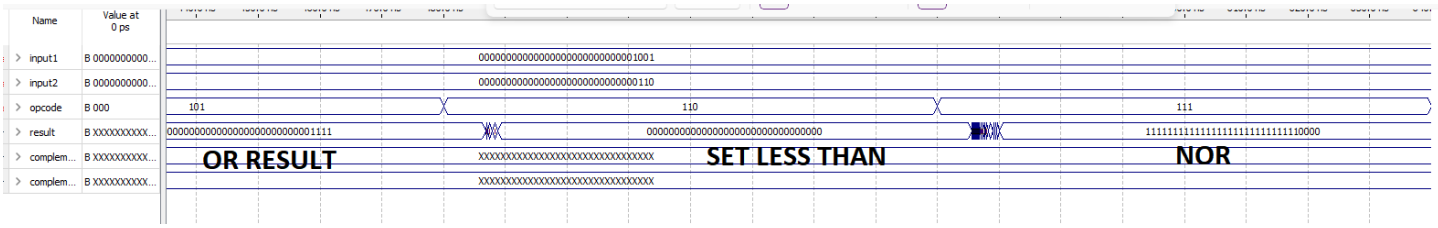
000 is Add result. 001 is Subtraction result.



011 is XOR result. 100 is AND result. 101 is OR result.



110 SLT result, 111 is NOR result.



The test file named alu_test is included by the zip. You can test it yourself. I set ALU.v as the top level entity by default but if there would be a problem, you have to make alu.v ***top level entity.*** I am sorry about that, I had no time to fix it.