ELEC 374 CPU Design – Phase 1

David-Alexandre Edwards (20099471),

Elan Bibas (20099396),

Hannah Berthiaume (20114482)

Verilog Code

Register.v

```
module register(input clock, input clear, input enable, input [31:0] BusMuxOut, output [31:0] BusMuxIn);
    reg [31:0] q;
    // Behavioral section for writing to the register
    always @ ( posedge clock)
        begin
            if(clear) begin
                q <= 32'b0;
            end
            else if(enable) begin
                q <= BusMuxOut;
            end
        end

    assign BusMuxIn = q;

endmodule
```

MDR.v

```
module MDR(
    input clk, clear, MDRin, read,
    input [31:0] BusMuxOut,
    input [31:0] Mdatain,
    output [31:0] BusMuxInMDR
);

// Behavioral section for writing to the register
reg [31:0] Din;
reg [31:0] In;
always @ (posedge clk)
    begin
        if (read) begin
            Din <= Mdatain;
        end
        else begin
            Din<= BusMuxOut;
        end
        if(clear) begin
            In <= 32'b0;
        end
        else if(MDRin) begin
            In <= Din;
        end
    end
    assign BusMuxInMDR = In;
endmodule
```

BoothMulti.v

```
module boothmult(output [63:0] result,input signed [31:0] X,input signed [31:0] Y);



    reg [2:0] cc[(32 / 2)-1:0];
    reg [32:0] pp[(32 / 2)-1:0];
    reg [32*2-1:0] spp[(32 / 2)-1:0];
    reg [32*2-1:0] product;
    wire [32:0] inv_X;
    integer k,i;

    assign inv_X = {~X[31], ~X} + 1;

    always @ (X or Y or inv_X)
    begin
        cc[0] = {Y[1],Y[0],1'b0};

        for(k=1;k<(32 / 2);k=k+1)
            cc[k] = {Y[2*k+1], Y[2*k], Y[2*k-1]};

        for(k=0;k<(32 / 2);k=k+1)
        begin
            case(cc[k])
                3'b001 , 3'b010 : pp[k] = {X[32-1],X};
                3'b011 : pp[k] = {X,1'b0};
                3'b100 : pp[k] = {inv_X[32-1:0],1'b0};
                3'b101 , 3'b110 : pp[k] = inv_X;
                default : pp[k] = 0;
            endcase

            spp[k] = $signed(pp[k]);

            for(i=0;i<k;i=i+1)
                spp[k] = {spp[k],2'b00};
        end

        product = spp[0];

        for(k=1;k<(32 / 2);k=k+1)
            product = product + spp[k];
    end

    assign result = product;

endmodule
```

ALU.v

```
module ALU(
```

```
    input [4:0] aluControl,
    input [31:0] BusMuxInY,
     input [31:0] BusMuxOut,
    output [31:0] Zlowout,
     output [31:0] Zhighout
);

//have two outputs instead
reg [31:0] COut;
reg [31:0] temp;
reg [31:0] temp1;
reg [31:0] temp2;
wire [63:0] ZOut;
integer i;

boothmult Mult(ZOut, BusMuxInY, BusMuxOut);

always @ (aluControl) begin

     temp1 = BusMuxOut;
     temp2 = BusMuxInY;
    if(aluControl == 5'b00011) begin //Add
        COut = BusMuxInY + BusMuxOut;
     end
    else if(aluControl == 5'b00100) begin //Sub
        COut = BusMuxInY - BusMuxOut;
     end
    else if(aluControl == 5'b00101) begin //Shift Right
        for (i = 0 ; i < 31 ; i = i + 1) begin
            COut[i] = BusMuxInY[i+1];
          end
        COut[31] = 0;
     end
    else if(aluControl == 5'b00110) begin //Shift Left
         for (i = 1 ; i < 32 ; i = i + 1) begin
            COut[i] = BusMuxInY[i-1];
          end
        COut[0] = 0;
     end
    else if(aluControl == 5'b00111) begin //Rotate Right
        for (i = 0 ; i < 31 ; i = i + 1) begin
            COut[i] = BusMuxInY[i+1];
          end
        COut[31] = BusMuxInY[0];
     end
    else if(aluControl == 5'b01000) begin //Rotate Left
         for (i = 1 ; i < 32 ; i = i + 1) begin
            COut[i] = BusMuxInY[i-1];
```

```
        end
      COut[0] = BusMuxInY[31];
    end
  else if(aluControl == 5'b01001) begin //AND
  //loop with for loop, then use logial and for each bit
      for (i = 0 ; i < 32 ; i = i + 1) begin
          COut = (BusMuxInY & temp1);
        end
    end
  else if(aluControl == 5'b01010) begin //OR
      for (i = 0 ; i < 32 ; i = i + 1) begin
          COut[i] = BusMuxInY[i] | BusMuxOut[i];
        end
    end
  else if(aluControl == 5'b01110) begin //Multiply
      COut = ZOut[31:0];
      temp = ZOut[63:32];
    end
  else if(aluControl == 5'b01111) begin //Divide
      COut = BusMuxInY / BusMuxOut;
    end
  else if(aluControl == 5'b10000) begin // Negate
      for (i = 0 ; i < 32 ; i = i + 1) begin
            COut[i] = ~temp1[i];
        end
        COut[0] = COut[0] + 1'b1;
    end
  else if(aluControl == 5'b10001) begin // Not
      for (i = 0 ; i < 32 ; i = i + 1) begin
            COut[i] = ~temp1[i];
        end
    end
    if(aluControl != 5'b01111)
      temp = {32{COut[31]}};
  // assign the results to z high and z low

end

assign Zhighout = temp; // ZOut[31:0]
assign Zlowout = COut;  // ZOut[63:32]

endmodule

// Z register holds the results of the operation in ALU
```

Bus.v

```
module bus(input [31:0] BusMuxInR0, input [31:0] BusMuxInR1, input [31:0] BusMuxInR2, input [31:0]
BusMuxInR3,input [31:0] BusMuxInR4,input [31:0] BusMuxInR5,input [31:0] BusMuxInR6,input [31:0] BusMuxInR7,input
[31:0] BusMuxInR8,input [31:0] BusMuxInR9,input [31:0] BusMuxInR10,
input [31:0] BusMuxInR11,input [31:0] BusMuxInR12,input [31:0] BusMuxInR13,input [31:0] BusMuxInR14,input [31:0]
BusMuxInR15,input [31:0] BusMuxInHi,input [31:0] BusMuxInLo,input [31:0] BusMuxInZHi,input [31:0]
BusMuxInZLo,input [31:0] BusMuxInPC,
input [31:0] BusMuxInMDR,input [31:0] BusMuxInRInP,input [31:0] BusMuxInRCSign, input R0out, input R1out,input
R2out,input R3out,input R4out,input R5out,input R6out,input R7out,input R8out,input R9out,input R10out,input
R11out,input R12out,
input R13out,input R14out,input R15out,input HIout,input LOout,input Zhighout,input Zlowout,input PCout, input
MDRout,input InPortout,input Cout, output [31:0] BusMuxOut);

reg [31:0] out;

always @ (BusMuxInR0 or BusMuxInR1 or BusMuxInR2 or BusMuxInR3 or BusMuxInR4 or BusMuxInR5 or BusMuxInR6
or BusMuxInR7 or BusMuxInR8 or BusMuxInR9 or BusMuxInR10 or BusMuxInR11 or BusMuxInR12 or BusMuxInR13 or
BusMuxInR14 or BusMuxInR15 or BusMuxInHi or BusMuxInLo or BusMuxInZHi or BusMuxInZLo or BusMuxInPC or
BusMuxInMDR or BusMuxInRInP or BusMuxInRCSign or R2out or R4out or MDRout) begin
   if(R0out) begin
      out = BusMuxInR0;
   end
   else if(R1out) begin
      out = BusMuxInR1;
   end
   else if(R2out) begin
      out = BusMuxInR2;
   end
   else if(R3out) begin
      out = BusMuxInR3;
   end
   else if(R4out) begin
      out = BusMuxInR4;
   end
   else if(R5out) begin
      out = BusMuxInR5;
   end
   else if(R6out) begin
      out = BusMuxInR6;
   end
   else if(R7out) begin
      out = BusMuxInR7;
   end
   else if(R8out) begin
      out = BusMuxInR8;
   end
   else if(R9out) begin
```

```
        out = BusMuxInR9;
    end
    else if(R10out) begin
        out = BusMuxInR10;
    end
    else if(R11out) begin
        out = BusMuxInR11;
    end
    else if(R12out) begin
        out = BusMuxInR12;
    end
    else if(R13out)begin
        out = BusMuxInR13;
    end
    else if(R14out)begin
        out = BusMuxInR14;
    end
    else if(R15out)begin
        out = BusMuxInR15;
    end
    else if(HIout)begin
        out = BusMuxInHi;
    end
    else if(LOout)begin
        out = BusMuxInLo;
    end
    else if(Zhighout)begin
        out = BusMuxInZHi;
    end
    else if(Zlowout)begin
        out = BusMuxInZLo;
    end
    else if(PCout)begin
        out = BusMuxInPC;
    end
    else if(MDRout)begin
        out = BusMuxInMDR;
    end
    else if(InPortout)begin
        out = BusMuxInRInP;
    end
    else if(Cout) begin
        out =  BusMuxInRCSign;
    end
    else begin
        out = 32'bx;
    end
end
```

```
assign BusMuxOut = out;
endmodule
```

Datapath.v

```
module DataPath(
    input [31:0] MDatain,
    input PCout, Zlowout, MDRout, MARin, Zin, PCin, MDRin, IRin, Yin, incPC, Read,
     input [4:0] aluControl,
     input clock, clear,
    input R2In, R2out, R4In, R4out, R5In
);

//define signals that will connect regiters to bus
wire [31:0] BusMuxOut;
wire [31:0] BusMuxInR0;
wire [31:0] BusMuxInR1;
wire [31:0] BusMuxInR2;
wire [31:0] BusMuxInR3;
wire [31:0] BusMuxInR4;
wire [31:0] BusMuxInR5;
wire [31:0] BusMuxInR6;
wire [31:0] BusMuxInR7;
wire [31:0] BusMuxInR8;
wire [31:0] BusMuxInR9;
wire [31:0] BusMuxInR10;
wire [31:0] BusMuxInR11;
wire [31:0] BusMuxInR12;
wire [31:0] BusMuxInR13;
wire [31:0] BusMuxInR14;
wire [31:0] BusMuxInR15;
wire [31:0] BusMuxInZHi;
wire [31:0] BusMuxInZLo;
wire [31:0] BusMuxInPC;
wire [31:0] BusMuxInMDR;
wire [31:0] BusMuxInRinP;
wire [31:0] BusMuxInRCSign;
wire [31:0] BusMuxInRY;
wire [31:0] ALULoOut;
wire [31:0] ALUHiOut;

//instantiate all registers
 register R0(clock, clear, R0In, BusMuxOut, BusMuxInR0);
 register R1(clock, clear, R1In, BusMuxOut, BusMuxInR1);
 register R2(clock, clear, R2In, BusMuxOut, BusMuxInR2);
 register R3(clock, clear, R3In, BusMuxOut, BusMuxInR3);
 register R4(clock, clear, R4In, BusMuxOut, BusMuxInR4);
 register R5(clock, clear, R5In, BusMuxOut, BusMuxInR5);
```

```
 register R6(clock, clear, R6In, BusMuxOut, BusMuxInR6);
 register R7(clock, clear, R7In, BusMuxOut, BusMuxInR7);
 register R8(clock, clear, R8In, BusMuxOut, BusMuxInR8);
 register R9(clock, clear, R9In, BusMuxOut, BusMuxInR9);
 register R10(clock, clear, R10In, BusMuxOut, BusMuxInR10);
 register R11(clock, clear, R11In, BusMuxOut, BusMuxInR11);
 register R12(clock, clear, R12In, BusMuxOut, BusMuxInR12);
 register R13(clock, clear, R13In, BusMuxOut, BusMuxInR13);
 register R14(clock, clear, R14In, BusMuxOut, BusMuxInR14);
 register R15(clock, clear, R15In, BusMuxOut, BusMuxInR15);
 register RHi(clock, clear, RHiIn, BusMuxOut, BusMuxInRHi);
 register RLO(clock, clear, RLOIn, BusMuxOut, BusMuxInRLo);
 register RZHi(clock, clear, Zin, ALUHiOut, BusMuxInZHi);
 register RZLO(clock, clear, Zin, ALULoOut, BusMuxInZLo);
 register PC(clock, clear, PCIn, BusMuxOut, BusMuxInPC);
 MDR mdr(clock, clear, MDRin, Read, BusMuxOut, MDatain, BusMuxInMDR);
 register RInP(clock, clear, RInPIn, BusMuxOut, BusMuxInRInP);
 register RCSign(clock, clear, RCSignIn, BusMuxOut, BusMuxInRCSign);

 register RY(clock, clear, Yin, BusMuxOut, BusMuxInRY);

//instantiate bus
bus cpuBUS(
    BusMuxInR0,BusMuxInR1,BusMuxInR2,
BusMuxInR3,BusMuxInR4,BusMuxInR5,BusMuxInR6,BusMuxInR7,BusMuxInR8,BusMuxInR9,BusMuxInR10,BusMuxInR11
,BusMuxInR12,BusMuxInR13,BusMuxInR14,BusMuxInR15,
    BusMuxInHi,BusMuxInLo,BusMuxInZHi,BusMuxInZLo,BusMuxInPC,BusMuxInMDR,BusMuxInRInP,BusMuxInRCSign,
    R0out,R1out,R2out,R3out,R4out,R5out,R6out,R7out,R8out,R9out,R10out,R11out,R12out,R13out,R14out,R15out,
    HIout,LOout,Zhighout,Zlowout,PCout,MDRout,INportout,Cout,BusMuxOut
);


ALU alu(aluControl, BusMuxInRY, BusMuxOut, ALULoOut, ALUHiOut);


endmodule
```

Testbenches

Each test bench follows a fairly similar process, The only components that change are the T0-T(N) steps. In each case we make sure to change the Mdatain value to represent the correct operator and registers as well as the Operator variable to make sure we are sending the correct operator code to the ALU.

The following is an example of one of our test benches for the AND operation.

```
`timescale 1ns/10ps

module Tb_add;
```

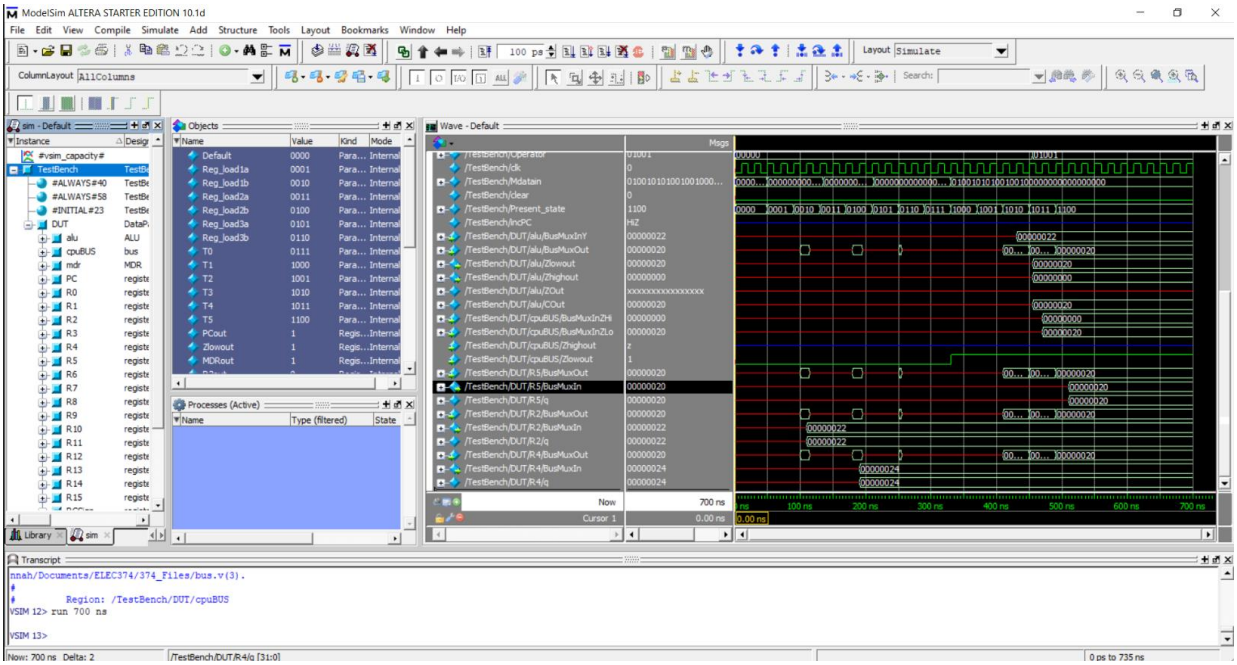
```
    reg PCout, Zlowout, MDRout, R2out, R4out;
    reg MARin, Zin, PCin, MDRin, IRin, Yin;
    reg IncPC,Read, R5in, R2in, R4in;
     reg [4:0] Operator;
    reg clk;
    reg [31:0] Mdatain;
     reg clear;

    parameter   Default = 4'b0000, Reg_load1a= 4'b0001, Reg_load1b= 4'b0010, Reg_load2a= 4'b0011,
                Reg_load2b = 4'b0100, Reg_load3a = 4'b0101, Reg_load3b = 4'b0110, T0= 4'b0111,
                T1= 4'b1000,T2= 4'b1001, T3= 4'b1010, T4= 4'b1011, T5= 4'b1100;
    reg[3:0] Present_state= Default;

DataPath DUT(Mdatain, PCout, Zlowout, MDRout, MARin, Zin, PCin, MDRin, IRin, Yin, incPC, Read, Operator, clk, clear,
R2in, R2out, R4in, R4out, R5in);


initial begin
    clk = 0;
    forever #10 clk = ~clk;
end



always @(posedge clk)     //finite state machine; if clk rising-edge
    begin
        case (Present_state)
            Default     :   #40 Present_state = Reg_load1a;
            Reg_load1a  :   #40 Present_state = Reg_load1b;
            Reg_load1b  :   #40 Present_state = Reg_load2a;
            Reg_load2a  :   #40 Present_state = Reg_load2b;
            Reg_load2b  :   #40 Present_state = Reg_load3a;
            Reg_load3a  :   #40 Present_state = Reg_load3b;
            Reg_load3b  :   #40 Present_state = T0;
            T0          :   #40 Present_state = T1;
            T1          :   #40 Present_state = T2;
            T2          :   #40 Present_state = T3;
            T3          :   #40 Present_state = T4;
            T4          :   #40 Present_state = T5;
        endcase
    end

always @(Present_state)     // do the required job ineach state
    begin
        case (Present_state)            //assert the required signals in each clk cycle
            Default: begin
                PCout <= 0;   Zlowout <= 0;   MDRout<= 0;   //initialize the signals
                R2out <= 0;   R4out <= 0;   MARin <= 0;   Zin <= 0;
```

```
    PCin <=0;   MDRin <= 0;   IRin  <= 0;   Yin <= 0;
    IncPC <= 0;   Read <= 0;   Operator <= 5'b00000;
    R5in <= 0; R2in <= 0; R4in <= 0; Mdatain <= 32'h00000000;
        clear <= 0;
end
Reg_load1a: begin
    Mdatain<= 32'h00000022;
    #10 Read <= 1; MDRin <= 1;
end
Reg_load1b: begin
    #10 MDRout<= 1; R2in <= 1;
    #15 MDRout<= 0; R2in <= 0;     // initialize R2 with the value $22
end
Reg_load2a: begin
        Read <= 0; MDRin <= 0;
    #5 Mdatain <= 32'h00000024;
    #10 Read <= 1; MDRin <= 1;

end
Reg_load2b: begin
    #10 MDRout<= 1; R4in <= 1;
    #15 MDRout<= 0; R4in <= 0;      // initialize R4 with the value $24
end
Reg_load3a: begin
        Read <= 0; MDRin <= 0;
    #2 Mdatain <= 32'h00000026;
    #4 Read <= 1; MDRin <= 1;
end
Reg_load3b: begin
    MDRout <= 1; R5in <= 1;
    #5 MDRout <= 0; R5in <= 0;      // initialize R5 with the value $26
end
T0: begin
    PCout <= 1; MARin <= 1; IncPC <= 1; Zin <= 1;
end
T1: begin
    Zlowout <= 1; PCin <= 1; Read <= 1; MDRin <= 1;
    Mdatain <= 32'h4A920000;       //opcode for “and R5, R2, R4”
end
T2: begin
    MDRout<= 1; IRin <= 1;
end
T3: begin
    R2out <= 1; Yin <= 1;

end
T4: begin
        R2out <= 0;
```

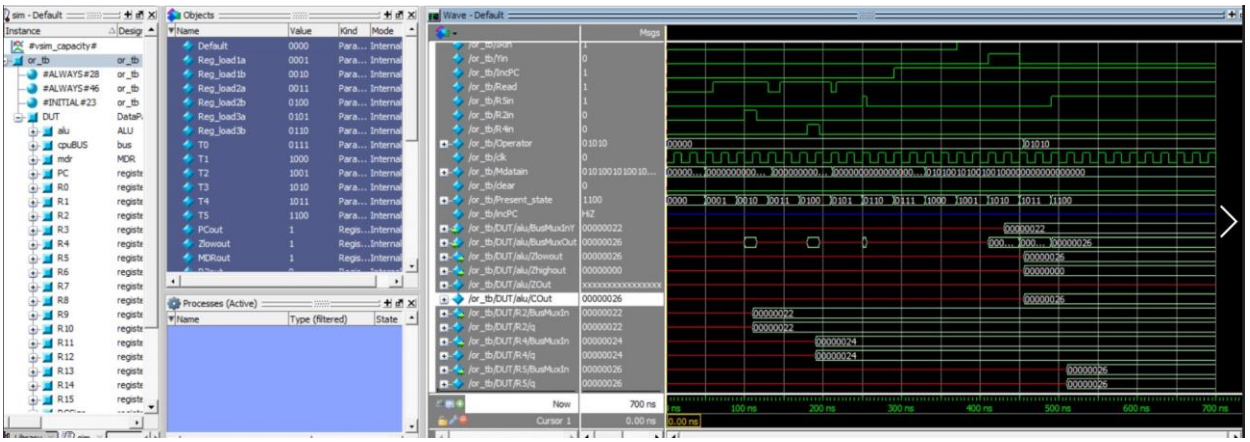
```
                Yin <= 0;
            R4out<= 1;
                #5 Operator <= 5'b01001;
                #10 Zin <= 1;
        end
        T5: begin
                R4out <= 0;
                Zin <= 0;
            Zlowout<= 1; R5in <= 1;

        end
    endcase
  end
endmodule
```

Functional Simulation Runs

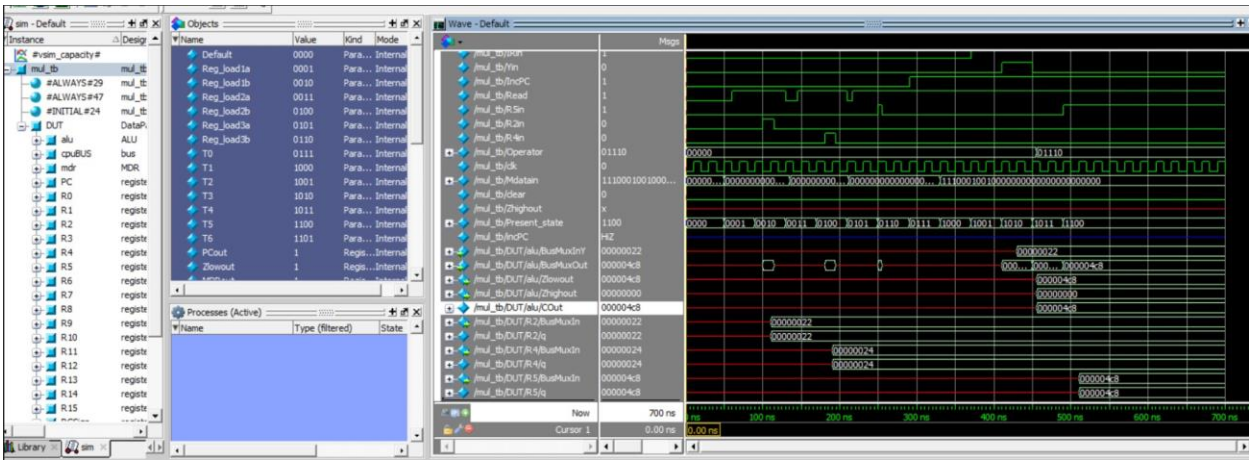
Logical And Operation



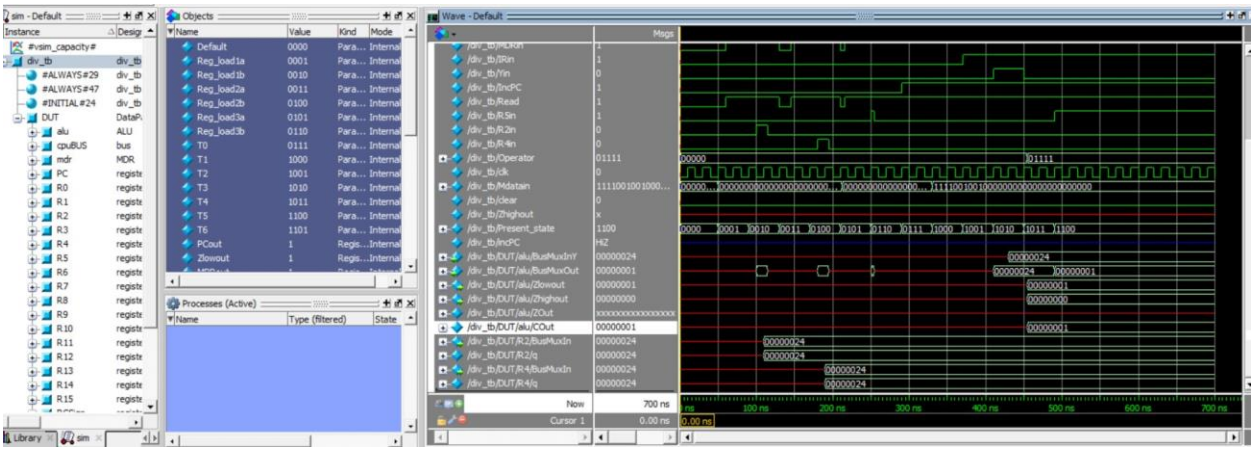
Logical OR Operation



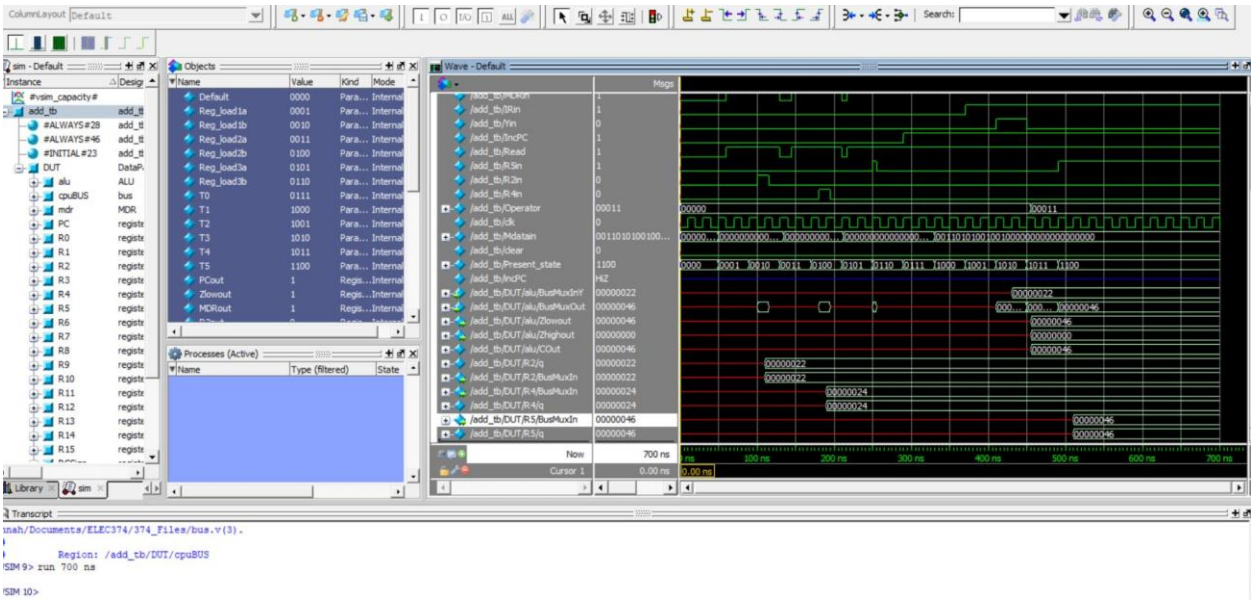
Multiplication Operation



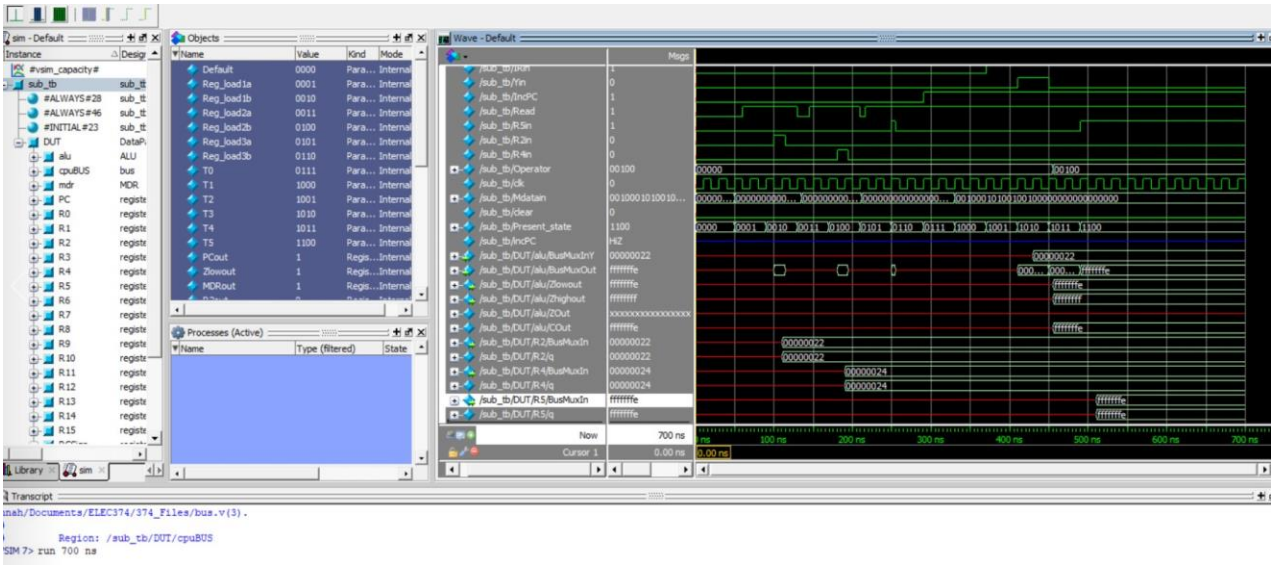
Division Operation



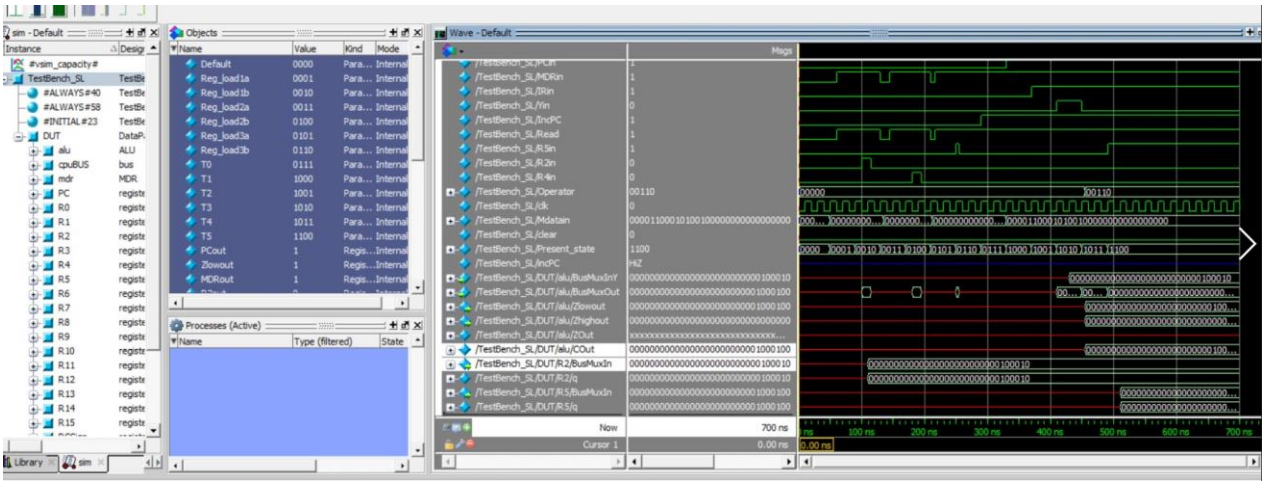
Add Operation



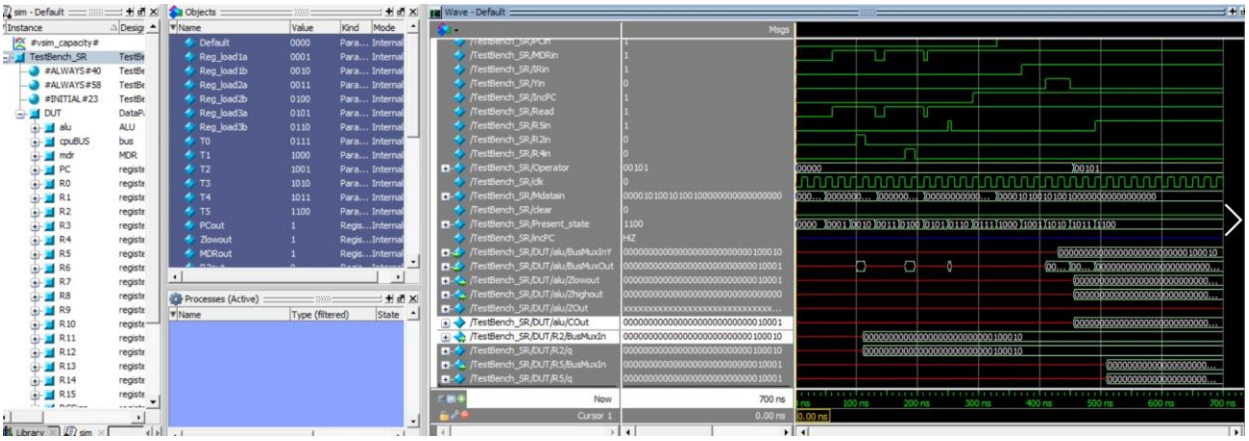
Subtraction Operation



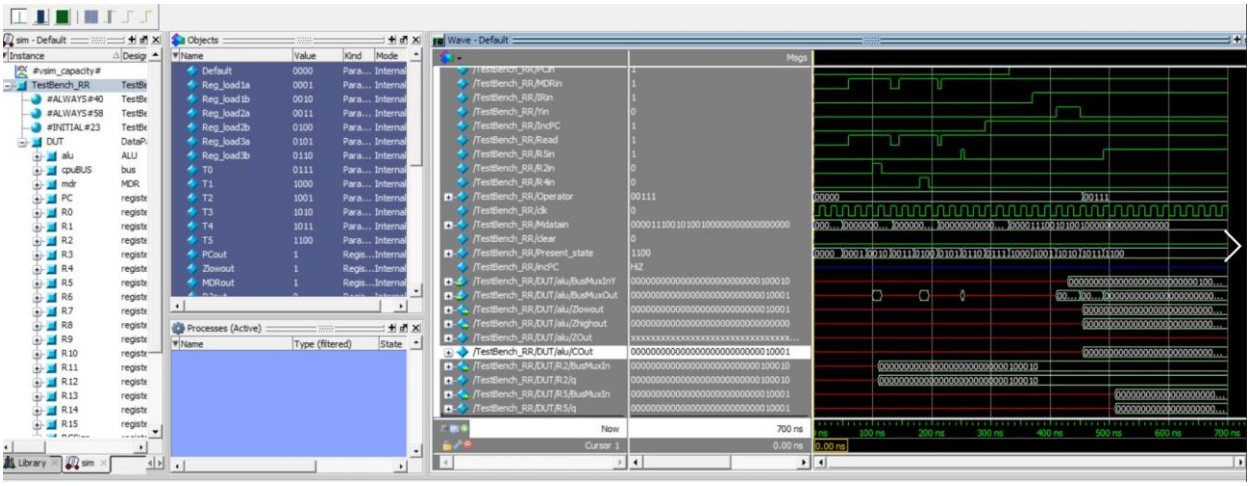
LSL Operation



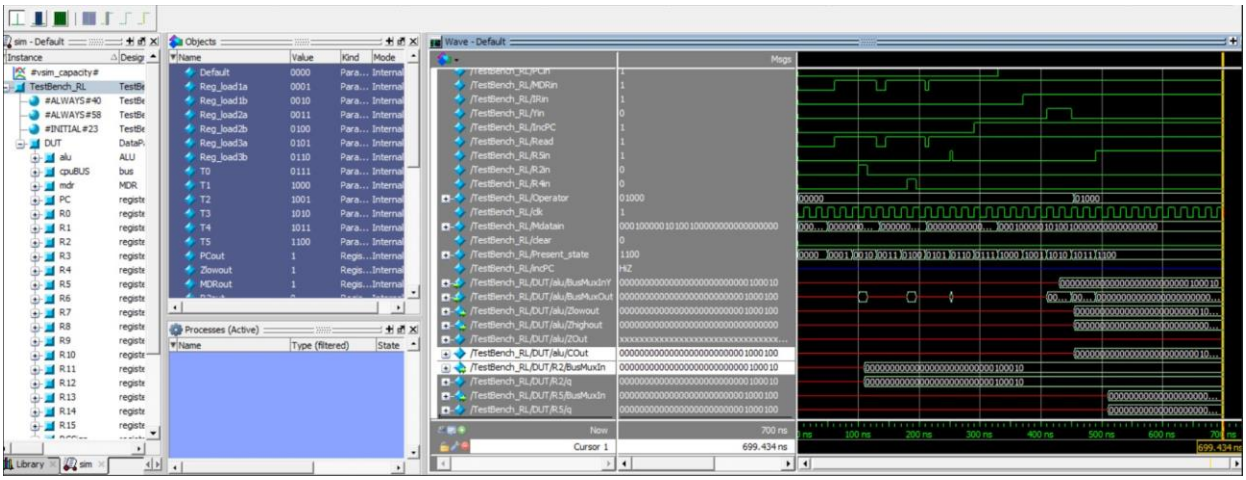
LSR Operation



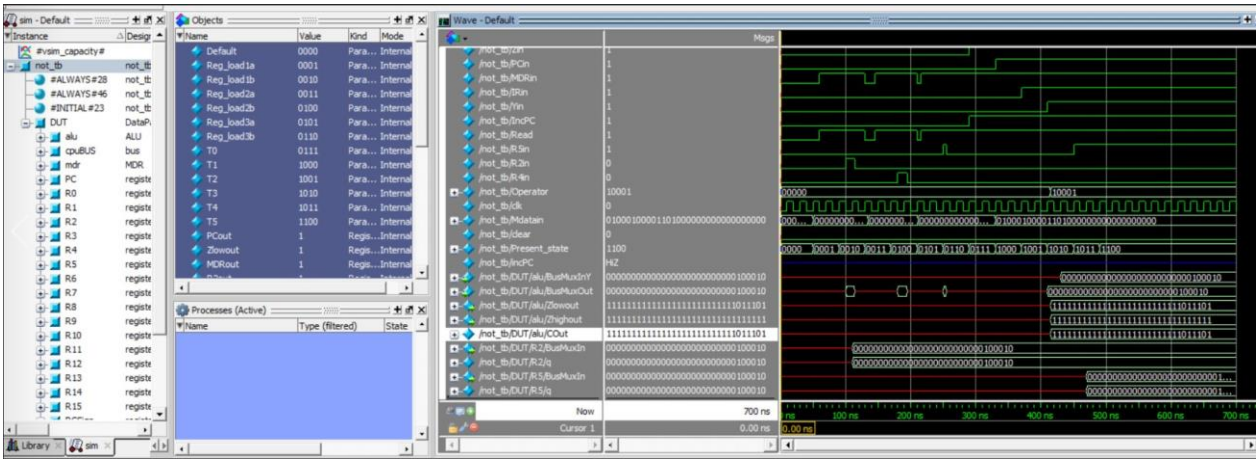
Rotate Right Operation



Rotate Left Operation



Not Operation



Negate Operation