ELEC 374 – Digital Systems Engineering

Lab 1 Report

Divaydeep Singh, 20143859

Lucas Austin, 20061953

ALU

```
module alu(
         input [3:0] select,
         input clk,
         input signed [31:0] A,
         input signed [31:0] B,
         output [63:0] Z,
         output carry
);

wire [63:0] mult_res;
wire[31:0] ror_res, rol_res;
reg [31:0] RLo, RHi;
assign Z = {RHi, RLo};
initial RHi = 0;

multiply mult(mult_res,A,B);
rotate_right ror(ror_res,A,B);
rotate_left rol(rol_res,A,B);

always @ (posedge clk)
begin
      case(select)
            4'b0001:
                  RLo <= A+B;
            4'b0010:
                  RLo <= A-B;
            4'b0011:
                  begin
                     RLo <= mult_res[31:0];
                     RHi <= mult_res[63:32];
                  end
            4'b0101:
                  begin
                     RHi <= A%B;
                     RLo <= (A-RHi)/B;
                  end
```

```
                end
            4'b0110:
                RLo <= A & B;
            4'b0111:
                RLo <= A|B;
            4'b1000:
                RLo <= ~B+1;
            4'b1010:
                RLo <= ~B;
            4'b1100:
                RLo <= A<<B;
            4'b1101:
                RLo <= A>>B;
            4'b1110:
                RLo <= rol_res;
            4'b1111:
                RLo <= ror_res;
            default:
                RLo <= RLo;
        endcase
    end
endmodule
```

BUS

```
module bus(
    input [31:0] busInR0, busInR1, busInR2, busInR3, busInR4, busInR5, busInR6, busInR7, busInR8, busInR9, busInR10, busInR11,
                 busInR12, busInR13, busInR14, busInR15, busInHI, busInLO, busInZ, busInPC, busInMDR, busInInPort, busInC,
    input R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out, HIout,
                 LOout, ZHighout, ZLowout, PCout, MDRout, InPortout, Cout,
    output [31:0] bus,
    input clk
    );

    wire[4:0] intermediate;


    bus_enc encoder(R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out,
                    R15out, HIout, LOout, ZHighout, ZLowout, PCout, MDRout, InPortout, Cout, intermediate,clk);

    bus_mux multiplexer(busInR0, busInR1, busInR2, busInR3, busInR4, busInR5, busInR6, busInR7, busInR8, busInR9, busInR10, busInR11,
                busInR12, busInR13, busInR14, busInR15, busInHI, busInLO, busInZ, busInPC, busInMDR, busInInPort, busInC,
                intermediate, bus,clk);


endmodule
```

BUS Encoder

```
module bus_enc(
   input R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out, HIout,
                  LOout, ZHighout, ZLowout, PCout, MDRout, InPortout, Cout,
   output reg[4:0] encoded,
   input clk
   );

   wire[31:0] outCombined;

   assign outCombined = {R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out, HIout,
                  LOout, ZHighout, ZLowout, PCout, MDRout, InPortout, Cout};


   always @ (outCombined, clk)
      case (outCombined)
         32'b00000000000000000000000000000001   :  encoded<=0;
         32'b00000000000000000000000000000010   :  encoded<=1;
         32'b00000000000000000000000000000100   :  encoded<=2;
         32'b00000000000000000000000000001000   :  encoded<=3;
         32'b00000000000000000000000000010000   :  encoded<=4;
         32'b00000000000000000000000000100000   :  encoded<=5;
         32'b00000000000000000000000001000000   :  encoded<=6;
         32'b00000000000000000000000010000000   :  encoded<=7;
         32'b00000000000000000000000100000000   :  encoded<=8;
         32'b00000000000000000000001000000000   :  encoded<=9;
         32'b00000000000000000000010000000000   :  encoded<=10;
         32'b00000000000000000000100000000000   :  encoded<=11;
         32'b00000000000000000001000000000000   :  encoded<=12;
         32'b00000000000000000010000000000000   :  encoded<=13;
         32'b00000000000000000100000000000000   :  encoded<=14;
         32'b00000000000000001000000000000000   :  encoded<=15;
         32'b00000000000000010000000000000000   :  encoded<=16;
         32'b00000000000000100000000000000000   :  encoded<=17;
         32'b00000000000001000000000000000000   :  encoded<=18;
         32'b00000000000010000000000000000000   :  encoded<=19;
         32'b00000000000100000000000000000000   :  encoded<=20;
         32'b00000000001000000000000000000000   :  encoded<=21;
         32'b00000000010000000000000000000000   :  encoded<=22;
         32'b00000000100000000000000000000000   :  encoded<=23;
         default                                :  encoded<=5'bx;
      endcase
```

BUS Multiplexer

```
module bus_mux(
   input [31:0] busInR0, busInR1, busInR2, busInR3, busInR4, busInR5, busInR6, busInR7, busInR8, busInR9, busInR10, busInR11,
                busInR12, busInR13, busInR14, busInR15, busInHI, busInLO, busInZ, busInPC, busInMDR, busInInPort, busInC,
   input [4:0] encoded,
   output reg [31:0] bus,
   input clk
);

   always @ (encoded, clk)
      case (encoded)
      23 :  bus<=busInR0;
      22 :  bus<=busInR1;
      21 :  bus<=busInR2;
      20 :  bus<=busInR3;
      19 :  bus<=busInR4;
      18 :  bus<=busInR5;
      17 :  bus<=busInR6;
      16 :  bus<=busInR7;
      15 :  bus<=busInR8;
      14 :  bus<=busInR9;
      13 :  bus<=busInR10;
      12 :  bus<=busInR11;
      11 :  bus<=busInR12;
      10 :  bus<=busInR13;
      9  :  bus<=busInR14;
      8  :  bus<=busInR15;
      7  :  bus<=busInHI;
      6  :  bus<=busInLO;
      5  :  bus<=busInZ;
      4  :  bus<=busInZ;
      3  :  bus<=busInPC;
      2  :  bus<=busInMDR;
      1  :  bus<=busInInPort;
      0  :  bus<=busInC;
      default  :  bus<=31'bx;
      endcase

endmodule
```

General Register

```
module gen_reg(
          output reg[31:0] Q,
          input [31:0] D,
          input en, clr, clk
);

          initial Q = 0;

always @(posedge clk)
    begin
          if(clr)
                Q = 0;
          else if(en)
                Q = D;
    end
endmodule
```

DataPath

```
module main1(
      input R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in,
      input R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out,
      input HIin, LOin, PCin, IRin, Yin, InPortout, Zin,
      input HIout, LOout, PCout, MDRout, MDRin, MARin, MDRread, Cout, clk, IncPC, ZLowout, ZHighout,
      input [3:0] ALUselect,
      input [31:0] MDatain,

      output R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15,
      output [63:0] ZReg
      );

      wire[31:0] bus;
      wire clr;
      wire IROut;
      wire [31:0] YData, XData;
      wire [31:0] ZLowData, ZHighData;

      wire [31:0] busInR0, busInR1, busInR2, busInR3, busInR4, busInR5, busInR6, busInR7, busInR8, busInR9, busInR10, busInR11, busInR12, busInR13, busInR14, busInR15, busInPC,
                  busInMAR, busInMDR, busInHI, busInLO, busInZ, busInInPort, busInC;

      gen_reg r0(busInR0, bus, R0in, clr, clk);
      gen_reg r1(busInR1, bus, R1in, clr, clk);
      gen_reg r2(busInR2, bus, R2in, clr, clk);
      gen_reg r3(busInR3, bus, R3in, clr, clk);
      gen_reg r4(busInR4, bus, R4in, clr, clk);
      gen_reg r5(busInR5, bus, R5in, clr, clk);
      gen_reg r6(busInR6, bus, R6in, clr, clk);
      gen_reg r7(busInR7, bus, R7in, clr, clk);
      gen_reg r8(busInR8, bus, R8in, clr, clk);
      gen_reg r9(busInR9, bus, R9in, clr, clk);
      gen_reg r10(busInR10, bus, R10in, clr, clk);
      gen_reg r11(busInR11, bus, R11in, clr, clk);
      gen_reg r12(busInR12, bus, R12in, clr, clk);
      gen_reg r13(busInR13, bus, R13in, clr, clk);
      gen_reg r14(busInR14, bus, R14in, clr, clk);
      gen_reg r15(busInR15, bus, R15in, clr, clk);
```

```
    gen_reg ir(IRout, bus, IRin, clr, clk);
    pc_reg pc(busInPC, bus, PCin, IncPC, clr, clk);

    gen_reg mar(busInMAR, bus, MARin, clr, clk);
    mdr_reg mdr(busInMDR, MDatain, bus, MDRin, MDRread, clr, clk);

    gen_reg hi(busInHI, bus, HIin, clr, clk);
    gen_reg lo(busInLO, bus, LOin, clr, clk);
    gen_reg y(YData, bus, Yin, clr, clk);
    z_reg_64 z(busInZ, ZReg, Zin, ZLowout, ZHighout, clr, clk);

    //ALU
    //alu alu(ALUselect, YData, ZLowData, ZHighData, carry);
    alu alu(ALUselect, clk, YData, bus, ZReg, carry);
    //assign ZData = ZReg[63:0];
    //assign busInZHI = ZReg[63:32];
    //assign busInZLO = ZReg[31:0];
    // Bus
    bus bus_inst(busInR0, busInR1, busInR2, busInR3, busInR4, busInR5, busInR6, busInR7, busInR8, busInR9, busInR10, busInR11,
                 busInR12, busInR13, busInR14, busInR15, busInHI, busInLO, busInZ, busInPC, busInMDR, busInInPort, busInPC,
                 R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out, HIout,
                 LOout, ZHighout, ZLowout, PCout, MDRout, InPortout, Cout, bus,clk);
endmodule
```

Datapath Testbench

```
module datapath_tb;

   reg R0in, R1in, R2in, R3in, R4in, R5in, R6in, R7in, R8in, R9in, R10in, R11in, R12in, R13in, R14in, R15in;

   reg R0out, R1out, R2out, R3out, R4out, R5out, R6out, R7out, R8out, R9out, R10out, R11out, R12out, R13out, R14out, R15out;

   reg HIin, LOin, PCin, IRin, Yin, InPortout, Zin;

   reg HIout, LOout, PCout, MDRout, MDRin, MARin, MDRread, Cout, clk, IncPC, ZLowout, ZHighout;


   reg[3:0] ALUselect;
   reg[31:0] MDatain;
   reg Read;

   wire R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, HI, LO, IR, BusMuxOut;
   wire[63:0] ZReg;

   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, T6 = 4'b1101;

   reg[3:0] Present_state = Default;

   main1 DUT (.R0in(R0in), .R1in(R1in), .R2in(R2in), .R3in(R3in), .R4in(R4in), .R5in(R5in), .R6in(R6in), .R7in(R7in), .R8in(R8in), .R9in(R9in),
              .R10in(R10in), .R11in(R11in), .R12in(R12in), .R13in(R13in), .R14in(R14in), .R15in(R15in), .R0out(R0out), .R1out(R1out), .R2out(R2out),
              .R3out(R3out), .R4out(R4out), .R5out(R5out), .R6out(R6out), .R7out(R7out), .R8out(R8out), .R9out(R9out), .R10out(R10out), .R11out(R11out),
              .R12out(R12out),.R13out(R13out),.R14out(R14out),.R15out(R15out), .HIin(HIin), .LOin(LOin), .PCin(PCin), .IRin(IRin), .Yin(Yin), .InPortout(InPortout),
              .Zin(Zin), .HIout(HIout), .LOout(LOout), .PCout(PCout), .MDRout(MDRout), .MDRin(MDRin), .MARin(MARin), .MDRread(Read), .Cout(Cout), .clk(clk), .IncPC(IncPC),
              .ZLowout(ZLowout), .ZHighout(ZHighout), .ALUselect(ALUselect), .MDatain(MDatain), .R0(R0), .R1(R1), .R2(R2), .R3(R3), .R4(R4), .R5(R5), .R6(R6), .R7(R7), .R8(R8),
              .R9(R9), .R10(R10), .R11(R11), .R12(R12), .R13(R13), .R14(R14), .R15(R15), .ZReg(ZReg));

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

```
always @(posedge clk)
          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          :  #60   Present_state = T1;
                         T1          :  #60   Present_state = T2;
                         T2          :  #60   Present_state = T3;
                         T3          :  #60   Present_state = T4;
                         T4          :  #60   Present_state = T5;
                         T5          :  #60   Present_state = T6;  //Only if T6 is needed
               endcase
          end
```

```
always @(Present_state)
        begin
              case(Present_state)
                    Default:begin
                                R0out <=0; R1out <=0; R2out <=0; R3out <=0; R4out <=0; R5out <=0; R6out <=0; R7out <=0;
                                R8out <=0; R9out <=0; R10out <=0; R11out <=0; R12out <=0; R13out <=0; R14out <=0; R15out <=0;
                                HIout <=0; LOout <=0; InPortout <=0; Cout <=0;
                                LOin <=0; HIin <=0;
                                PCout <=0; ZLowout <=0; ZHighout <=0; MDRout <=0;
                                R2out <=0; R4out <=0; MARin <=0; Zin <=0;
                                PCin <=0; MDRin <=0; IRin <=0; Yin <=0;
                                IncPC <=0; Read <=0; ALUselect <=0;
                                R5in <=0; R2in <=0; R4in<=0; MDatain<=32'h00000000;
        end
        Reg_load1a:begin
                         Read = 0; MDRin = 0;
                         MDatain <= 32'h00000022;      //For everything except ror, rol
                         //MDatain <= 32'h80000022;    //For ror, rol
                         #10 Read <= 1; MDRin <= 1;
                         #15 Read <= 0; MDRin <= 0;
        end
        Reg_load1b: begin
                        #10 MDRout <= 1; R2in <= 1;
                        #15 MDRout <= 0; R2in <= 0;     // initialize R2 with the value $22
        end
        Reg_load2a: begin
                       MDatain <= 32'h00000024;       //Everything except div, shr, shl, ror, rol
                       //MDatain <= 32'h00000003;        //For div, shr, shl, ror, rol
                       #10 Read <= 1; MDRin <= 1;
                       #15 Read <= 0; MDRin <= 0;
        end
        Reg_load2b: begin
                        #10 MDRout <= 1; R4in <= 1;
                        #15 MDRout <= 0; R4in <= 0;  // initialize R4 with the value $24 (3 for division)
        end
       Reg_load3a: begin
                        MDatain <= 32'h00000026;
                       #10 Read <= 1; MDRin <= 1;
                       #15 Read <= 0; MDRin <= 0;
        end
```

```
        Reg_load3b: begin
                          #10 MDRout <= 1; R5in <= 1;
              #15 MDRout <= 0; R5in <= 0;  // initialize R5 with the value $26
         end

        T0: begin                                 // see if you need to de-assert these signals
              PCout <= 1; MARin <= 1; IncPC <= 1; Zin <= 1;
         end
        T1: begin
              PCout <= 0; MARin <= 0; IncPC <= 0; Zin <= 0;
              ZLowout <= 1; PCin <= 1; Read <= 1; MDRin <= 1;
              //MDatain <= 32'h4A920000;       // opcode for "and R5, R2, R4"
              //MDatain <= 32'h52920000;       // opcode for "or R5, R2, R4"
              //MDatain <= 32'h1A920000;       // opcode for "add R5, R2, R4"
              //MDatain <= 32'h22920000;       // opcode for "sub R5, R2, R4"
              //MDatain <= 32'h71200000;       // opcode for "mul R2, R4"
              //MDatain <= 32'h79200000;       // opcode for "div R2, R4"
              //MDatain <= 32'h2A920000;       // opcode for "shr R5, R2, R4"
              //MDatain <= 32'h32920000;       // opcode for "shl R5, R2, R4"
              //MDatain <= 32'h3A920000;       // opcode for "ror R5, R2, R4"
              //MDatain <= 32'h42920000;       // opcode for "rol R5, R2, R4"
              //MDatain <= 32'h81200000;       // opcode for "neg R2, R4"
              MDatain <= 32'h89200000;       // opcode for "not R2, R4"
         end
        T2: begin
              ZLowout <= 0; PCin <= 0; Read <= 0; MDRin <= 0;
              MDRout <= 1; IRin <= 1;
         end

        T3: begin
              MDRout <= 0; IRin <= 0;
              R2out <= 1; Yin <= 1;
         end

          end
         T4: begin
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b0110; Zin <= 1;      //T4 and
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b0111; Zin <= 1;      //T4 or
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b0001; Zin <= 1;      //T4 add
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b0010; Zin <= 1;      //T4 sub
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b0011; Zin <= 1;      //T4 mul
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b0101; Zin <= 1;      //T4 div
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b1101; Zin <= 1;      //T4 shr
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b1100; Zin <= 1;      //T4 shl
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b1111; Zin <= 1;      //T4 ror
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b1110; Zin <= 1;      //T4 rol
                //Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b1000; Zin <= 1;      //T4 neg
                Yin <=0; R2out <= 0; R4out <= 1; ALUselect <= 4'b1010; Zin <= 1;     //T4 not
          end
         T5: begin
                ALUselect <= 4'b0000;
                R4out <= 0; Zin <=0; ZLowout <= 1; R5in <= 1;      //T5 and, or, add, sub ,shr, shl, ror, rol
                //R4out <= 0; Zin <=0; ZLowout <= 1; LOin <= 1;    //T5 mul, div
          end
         T6: begin
                //ZLowout <=0; LOin <=0; ZHighout <=1; HIin <=1;      //T6 mul, div

          end
        endcase
    end
endmodule
```

MDR register

```
module mdr_reg(out, MdataIn, bus, en, rd, clr, clk);
output reg [31:0] out;
input [31:0] MdataIn, bus;
input en, rd, clr, clk;
wire [31:0] in;
always @(posedge clk)
         if(clr)
            out=0;
         else if(en)
            if(rd)
               out = MdataIn;
         else
               out = bus;


endmodule
```

Multiplication Algorithm

```
module multiply(
   output[63:0] p,
   input signed [31:0] x,y
);

   reg[2:0] cc[(32/2)-1:0];
   reg[32:0] pp[(32/2)-1:0];
   reg[63:0] spp[(32/2)-1:0];
   reg[63:0] prod;
   wire[32:0] inv_x;
   integer kk,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(kk=1;kk<(32/2);kk=kk+1)
            cc[kk] = {y[2*kk+1],y[2*kk],y[2*kk-1]};
         for(kk=0;kk<(32/2);kk=kk+1)
         begin
            case(cc[kk])
               3'b001,3'b010:pp[kk]={x[32-1],x};
               3'b011:pp[kk]={x,1'b0};
               3'b100:pp[kk]={inv_x[32-1:0],1'b0};
               3'b101,3'b110:pp[kk] = inv_x;
               default:pp[kk]=0;
            endcase
            spp[kk]=$signed(pp[kk]);
            for(i=0;i<kk;i=i+1)
               spp[kk]={spp[kk],2'b00};
            end
            prod=spp[0];
            for(kk=1;kk<(32/2);kk=kk+1)
               prod=prod+spp[kk];
            end
            assign p = prod;
endmodule
```

PC Reg

```
module pc_reg(
         output reg [31:0] Q,
         input [31:0] D,
         input wr, inc, clr, clk
);
         initial Q=0;
always @(posedge clk)
   begin
         if(clr)
            Q = 0;
         else if(wr)
            Q = D;
         else if(inc)
            Q = Q+1;
   end
endmodule
```

Rotate Register

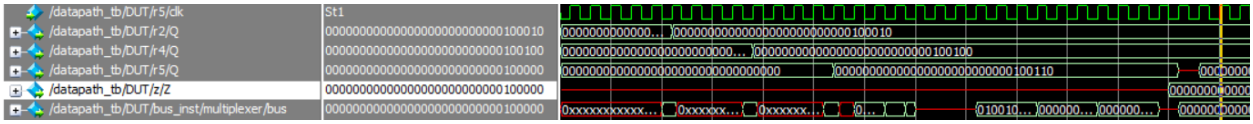
```
module rotate_right(
output [31:0] R,
input [31:0] A, B
);

   wire [4:0] N;
   assign N = A % 32;
   assign R = (N == 31) ? {B[30:0], B[31:31]} :

                                    (N == 30) ? {B[29:0], B[31:30]} :
                                    (N == 29) ? {B[28:0], B[31:29]} :
                                    (N == 28) ? {B[27:0], B[31:28]} :
                                    (N == 27) ? {B[26:0], B[31:27]} :
                                    (N == 26) ? {B[25:0], B[31:26]} :
                                    (N == 25) ? {B[24:0], B[31:25]} :
                                    (N == 24) ? {B[23:0], B[31:24]} :
                                    (N == 23) ? {B[22:0], B[31:23]} :
                                    (N == 22) ? {B[21:0], B[31:22]} :
                                    (N == 21) ? {B[20:0], B[31:21]} :
                                    (N == 20) ? {B[19:0], B[31:20]} :
                                    (N == 19) ? {B[18:0], B[31:19]} :
                                    (N == 18) ? {B[17:0], B[31:18]} :
                                    (N == 17) ? {B[16:0], B[31:17]} :
                                    (N == 16) ? {B[15:0], B[31:16]} :
                                    (N == 15) ? {B[14:0], B[31:15]} :
                                    (N == 14) ? {B[13:0], B[31:14]} :
                                    (N == 13) ? {B[12:0], B[31:13]} :
                                    (N == 12) ? {B[11:0], B[31:12]} :
                                    (N == 11) ? {B[10:0], B[31:11]} :
                                    (N == 10) ? {B[9:0], B[31:10]} :
                                    (N == 9) ? {B[8:0], B[31:9]} :
                                    (N == 8) ? {B[7:0], B[31:8]} :
                                    (N == 7) ? {B[6:0], B[31:7]} :
                                    (N == 6) ? {B[5:0], B[31:6]} :
                                    (N == 5) ? {B[4:0], B[31:5]} :
                                    (N == 4) ? {B[3:0], B[31:4]} :
                                    (N == 3) ? {B[2:0], B[31:3]} :
                                    (N == 2) ? {B[1:0], B[31:2]} :
                                    (N == 1) ? {B[0:0], B[31:1]} :
                                     B[31:0];
endmodule
```

```
module rotate_left(
        output [31:0] R,
        input [31:0] A, B
);

   wire [4:0] N;
   assign N = A % 32;
   assign R = (N == 31) ? {B[0:0], B[31:1]} :
                          (N == 30) ? {B[1:0], B[31:2]} :
                          (N == 29) ? {B[2:0], B[31:3]} :
                                  (N == 28) ? {B[3:0], B[31:4]} :
                                  (N == 27) ? {B[4:0], B[31:5]} :
                                  (N == 26) ? {B[5:0], B[31:6]} :
                                  (N == 25) ? {B[6:0], B[31:7]} :
                                  (N == 24) ? {B[7:0], B[31:8]} :
                                  (N == 23) ? {B[8:0], B[31:9]} :
                                  (N == 22) ? {B[9:0], B[31:10]} :
                                  (N == 21) ? {B[10:0], B[31:11]} :
                                  (N == 20) ? {B[11:0], B[31:12]} :
                                  (N == 19) ? {B[12:0], B[31:13]} :
                                  (N == 18) ? {B[13:0], B[31:14]} :
                                  (N == 17) ? {B[14:0], B[31:15]} :
                                  (N == 16) ? {B[15:0], B[31:16]} :
                                  (N == 15) ? {B[16:0], B[31:17]} :
                                  (N == 14) ? {B[17:0], B[31:18]} :
                                  (N == 13) ? {B[18:0], B[31:19]} :
                                  (N == 12) ? {B[19:0], B[31:20]} :
                                  (N == 11) ? {B[20:0], B[31:21]} :
                                  (N == 10) ? {B[21:0], B[31:22]} :
                                  (N == 9) ? {B[22:0], B[31:23]} :
                                  (N == 8) ? {B[23:0], B[31:24]} :
                                  (N == 7) ? {B[24:0], B[31:25]} :
                                  (N == 6) ? {B[25:0], B[31:26]} :
                                  (N == 5) ? {B[26:0], B[31:27]} :
                                  (N == 4) ? {B[27:0], B[31:28]} :
                                  (N == 3) ? {B[28:0], B[31:29]} :
                                  (N == 2) ? {B[29:0], B[31:30]} :
                                  (N == 1) ? {B[30:0], B[31:31]} :
                                   B[31:0];
endmodule
```

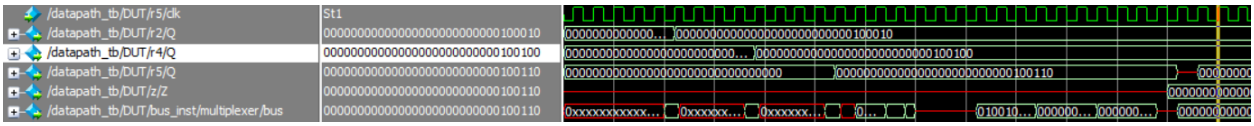
Z_Register

```
module z_reg_64 (
          output reg [31:0] Z,
          input [63:0] D,
          input ZIn, ZLowOut, ZHighOut, clr, clk
);

reg[63:0] ZData;

always @(clk, D)
   begin
       if(clr)
          begin
             Z<=64'h0000000000000000;
          end
       else if(ZIn)
          begin
             ZData<=D;
          end
       if(ZLowOut)
          begin
             Z<=ZData[31:0];
          end
       else if(ZHighOut)
          begin
             Z<=ZData[63:32];
          end
   end
endmodule
```

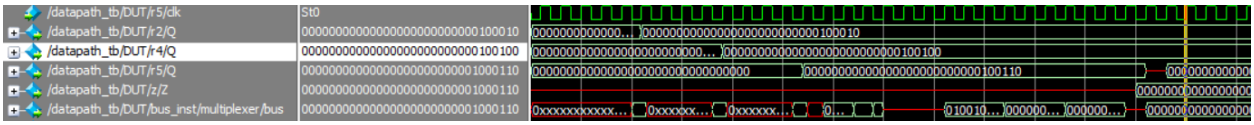
Waveform Part 3a



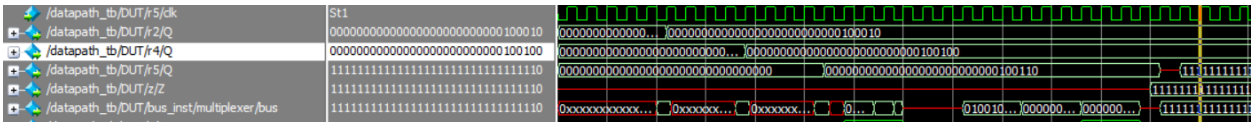
Waveform Part 3b



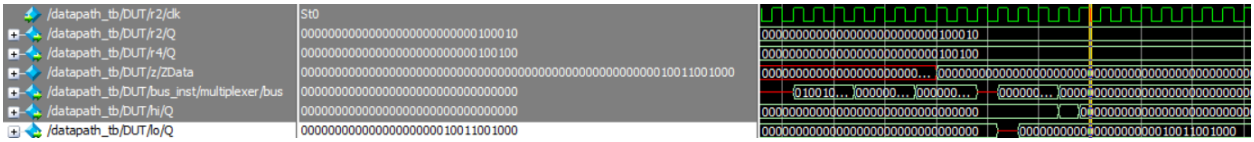
Waveform Part 3c



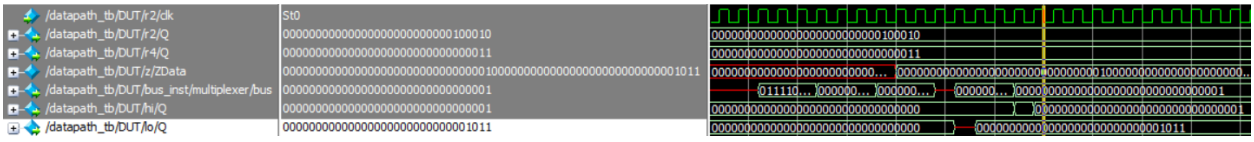
Waveform Part 3d



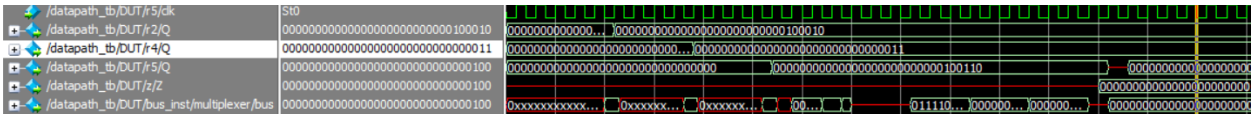
Waveform Part 3e



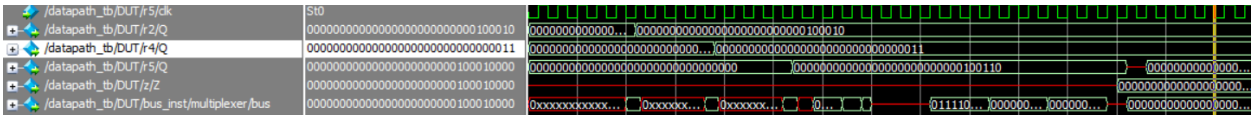
Waveform Part 3f



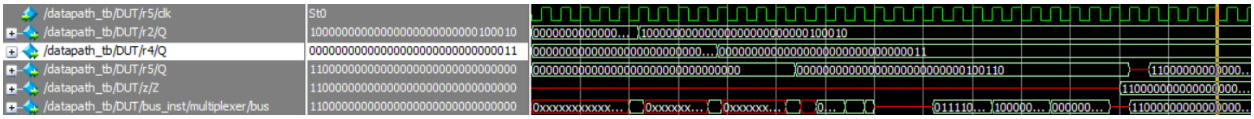
Waveform Part 3g



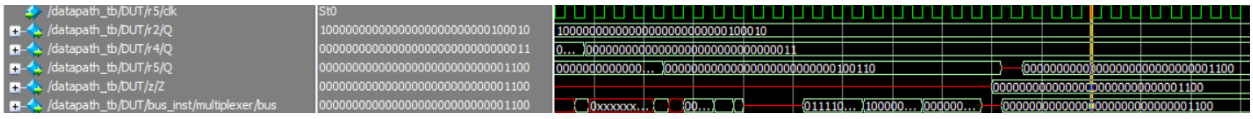
Waveform Part 3h



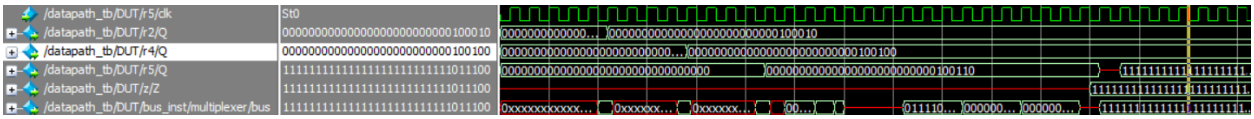
Waveform Part 3i



Waveform Part 3j



Waveform Part 3k



Waveform Part 3l