



भारतीय प्रौद्योगिकी संस्थान गांधीनगर
पालज, गांधीनगर, गुजरात 382 055

INDIAN INSTITUTE OF TECHNOLOGY GANDHINAGAR
PALAJ, GANDHINAGAR, GUJARAT 382 055

Name : Bhavik Patel
Roll no. : 22110047

IITGN

Digital Systems

LAB -8

Question:

In the last lab you have designed 32-bit Register File containing 8 registers. Extend this to 16 registers. Now, make this register file dual-port read. This means that we should be able to give in 2 sets of addresses and read two register contents simultaneously. The data read from the registers should be passed to the ALU which will perform arithmetic and logical operations on these registers' contents. The results (output of the ALU) will be stored back into the Register file.

For instance, we will implement ADD R1, R2, R2 instruction:

This instruction takes in the contents of R1 and R2, adds them and transfers the result (of addition) to R2. There is a separate 1-bit register called Cy which stores the Carry output.

For this lab:

You are to implement the following instructions:

Arithmetic instruction:

0001: **ADD** Rx, Ry, Rz

Logical instruction:

0010: **AND** Rx, Ry, Rz

(where x, y, and z would be any values between 0..15)

Here, 0001 and 0010 are the operation codes which are used to differentiate between different operations. In this case, they differentiate between ADD and AND.


















Verilog Code

```
`timescale 1ns / 1ps
module bit32Register(rst,D,Cy,clk,Mode,Rx,Ry,Rz);
input rst,clk;
input [511:0]D;
input [3:0]Rx,Ry,Rz,Mode;
output reg Cy;
reg [32:0]ans=0;
reg [31:0]out[15:0];
integer i;
always @(*)
begin
    if(rst) begin
        out[0]<=D[31:0];
        out[1]<=D[63:32];
        out[2]<=D[95:64];
        out[3]<=D[127:96];
        out[4]<=D[159:128];
        out[5]<=D[191:160];
        out[6]<=D[223:192];
        out[7]<=D[255:224];
        out[8]<=D[287:256];
        out[9]<=D[319:288];
        out[10]<=D[351:320];
        out[11]<=D[383:352];
        out[12]<=D[415:384];
        out[13]<=D[447:416];
        out[14]<=D[479:448];
        out[15]<=D[511:480];
        Cy=0;
    end
end
```

```
always @(posedge clk ) begin
    if(~rst) begin
        if(Mode==4'b0001)begin
            ans=(out[Rx]+out[Ry]);
            out[Rz]=ans[31:0];
            Cy=ans[32];
        end
        else if(Mode==4'b0010) begin
            // Mode==4'b0010
            ans=(out[Rx]&out[Ry]);
            out[Rz]=ans[31:0];
            Cy=0;
        end
        else begin
            out[Rz]=out[Rz];
            Cy=0;
        end
    end
end

endmodule
```

Initialize the Registers with Values

✓  out[15:0]...	10000000000000011100000000000000,1000000000000000
>  [15][31:...	10000000000000011100000000000000
>  [14][31:...	10000000000000011100000000000000
>  [13][31:...	00000000000000011100000000000000
>  [12][31:...	00000000000000011100000000000000
>  [11][31:...	00000000000000011100000000000000
>  [10][31:...	00000000000000011100000000000000
>  [9][31:0]	00000000000000011100000000000000
>  [8][31:0]	00000000000000011100000000000000
>  [7][31:0]	00000000000000011100000000000000
>  [6][31:0]	00000000000000011100000000000000
>  [5][31:0]	00000000000000011100000000000000
>  [4][31:0]	00000000000000011100000000000000
>  [3][31:0]	00000000000000011100000000000000
>  [2][31:0]	00000000000000011100000000000000
>  [1][31:0]	00000000000000011100000000000000
>  [0][31:0]	00000000000000011100000000000000

Test bench

```
`timescale 1ns / 1ps

module test_bench();
  reg rst,clk;
  reg [3:0]Rx,Ry,Rz,Mode;
  reg [511:0]D;
  integer i;
  wire Cy;
  bit32Register uut(rst,D,Cy,clk,Mode,Rx,Ry,Rz);

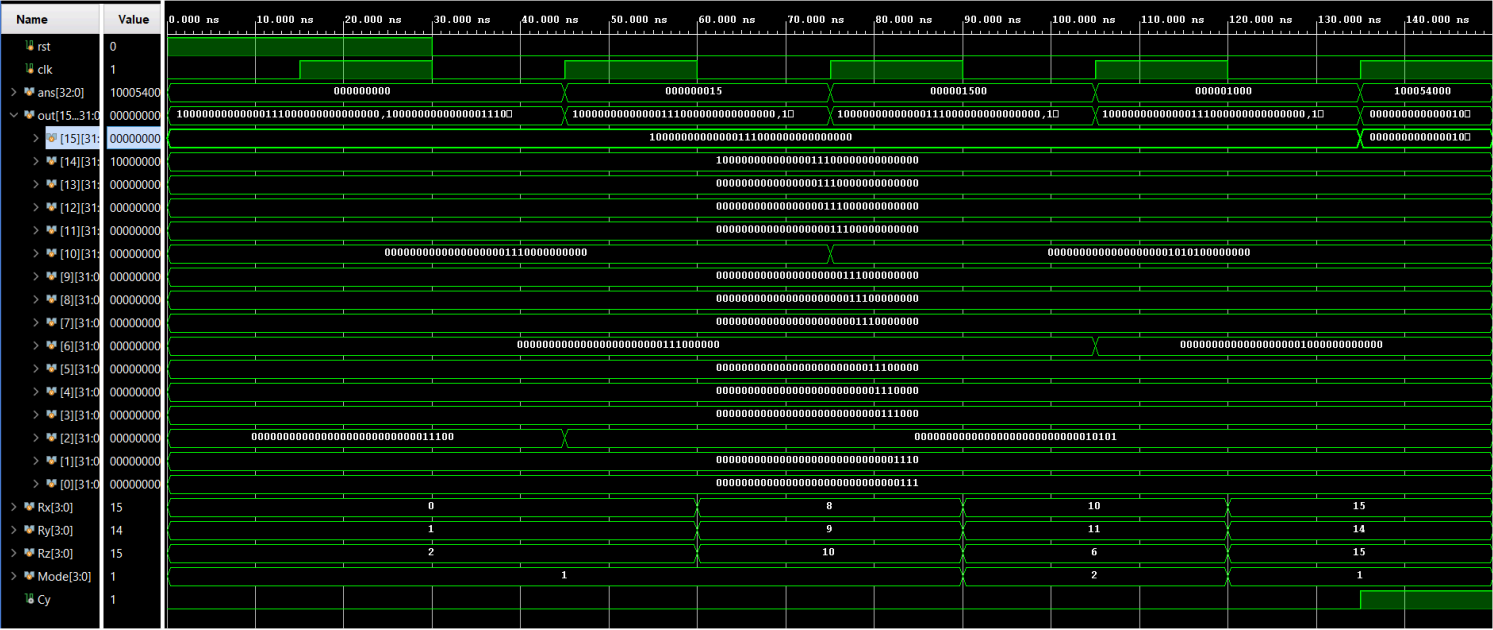
  initial begin
    clk=0;
    forever #15 clk=~clk;
  end

  initial
  begin
    rst=1;
    //Initialise the registers
    D=0;
    for(i=0; i<512;i=i+33) begin
      D[i]=1;
      D[i+1]=1;
      D[i+2]=1;
    end
    D[511]=1;
    D[479]=1;
    Rx=3'b0000;
    Ry=3'b0001;
    Rz=3'b0010;
    Mode = 3'b0001;
  end
endmodule
```

```
always @(posedge clk ) begin
  if(~rst) begin
    if(Mode==4'b0001)begin
      ans=(out[Rx]+out[Ry]);
      out[Rz]=ans[31:0];
      Cy=ans[32];
    end
    else if(Mode==4'b0010) begin
      // Mode==4'b0010
      ans=(out[Rx]&out[Ry]);
      out[Rz]=ans[31:0];
      Cy=0;
    end
    else begin
      out[Rz]=out[Rz];
      Cy=0;
    end
  end
end

endmodule
```

Simulation



[Link of Simulation Image](#)