Switching Theory Lab – CS226

Name: M. Maheeth Reddy **Roll No.**: 1801CS31 **Date**: 09-May-2020

Lab No.: 13

Ans 1:

Source Code:

```
// Binary to Gray Code Converter
module p1(gray,binary);
    // Mode Declaration
    input [3:0] binary;     // Binary Form
    output [3:0] gray;      // Gray Form

    assign gray[3] = binary[3];
    assign gray[2] = binary[3] ^ binary[2];
    assign gray[1] = binary[2] ^ binary[1];
    assign gray[0] = binary[1] ^ binary[0];
endmodule
```

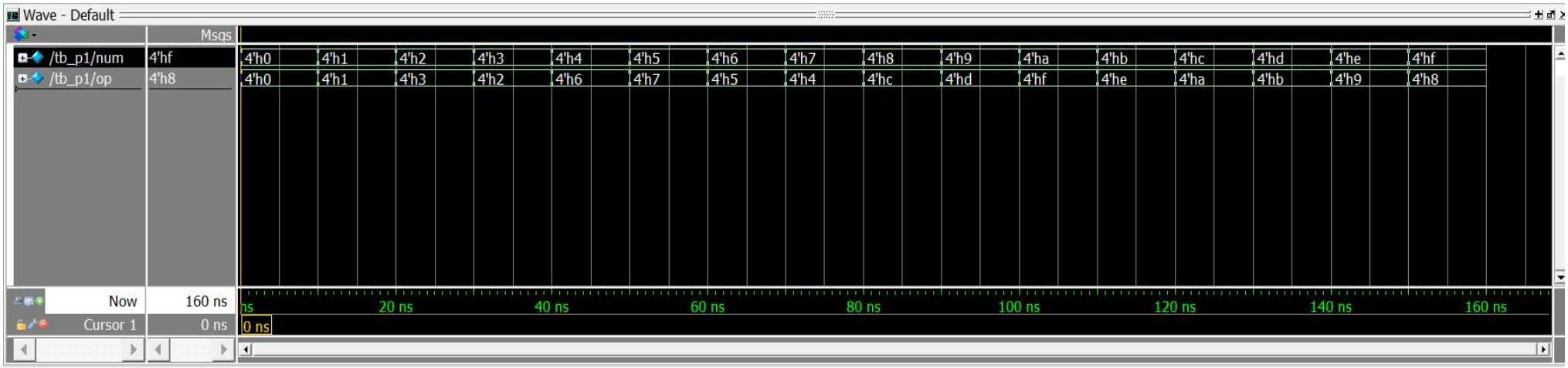
Text Output

```
VSIM 27> run
# binary     gray
#   0000       0000
#   0001       0001
#   0010       0011
#   0011       0010
#   0100       0110
#   0101       0111
#   0110       0101
#   0111       0100
#   1000       1100
#   1001       1101
#   1010       1111
#   1011       1110
#   1100       1010
#   1101       1011
#   1110       1001
#   1111       1000
```

Testbench:

```
module tb_p1();
    // Mode Declaration
    reg [3:0] num;
    wire [3:0] op;

    p1 UUT (op,num);    // Instantiate p1

    // Testcase: All 4-bit binary numbers
    initial begin
        num = 4'b0000;
        repeat(15) begin
            #10 num = num + 1;
        end
    end

    initial begin
        $display("binary    gray");
        $monitor(" %b      %b",num,op);
    end
endmodule
```

Simulated Waveform



Simulation Time: **160ns**

Ans 2:

Source Code:

```
// Module for given Sequential Circuit
module p2 (Y,S,clk);
    output Y;       // Output Y
    input S, clk;

    wire S1,S2;
    reg [2:1] S_prev;  // Stores previous values of S

    assign S2 = S_prev[2];   // Value of S before 2 clock cycles
    assign S1 = S_prev[1];   // Value of S before 1 clock cycle

    assign Y = (S & S1 & S2) | (~S & ~S1 & ~S2);

    always @ (posedge clk) begin
        S_prev = {S, S_prev[2]};
    end
endmodule
```

Testbench:

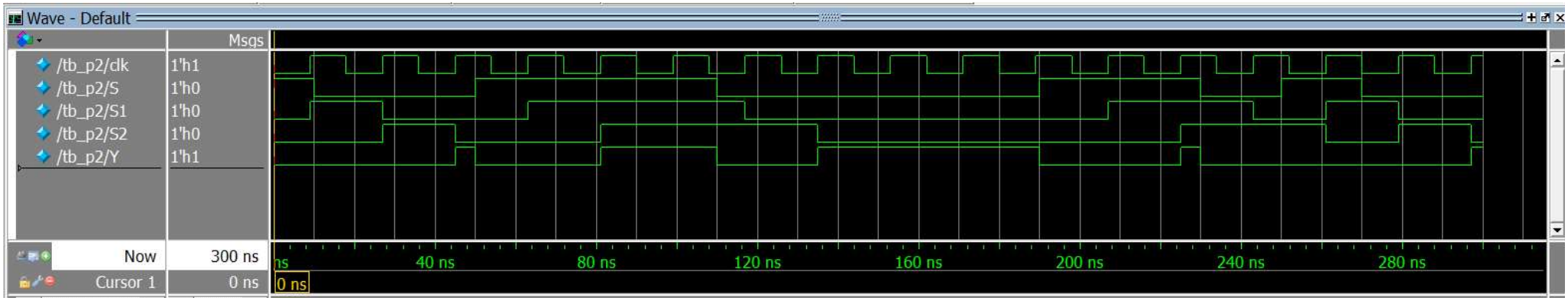
```
// Testbench for p2.v
module tb_p2();
    // Mode Declaration
    wire Y, S1, S2;
    reg S, clk;

    p2 UUT (Y, S, clk);   //Instantiate p2

    always #9 clk = ~clk;

    // Assign S1 and S2
    assign S1 = UUT.S_prev[2];
    assign S2 = UUT.S_prev[1];

    initial begin
        clk = 0;
        UUT.S_prev = 2'b00;

        S = 1; #10;
        S = 0; #40;
        S = 1; #60;
```

Text Output

```
VSIM 37> run
# time=0 clk=0  S=1  S1=0  S2=0  Y=0
# time=9 clk=1  S=1  S1=1  S2=0  Y=0
# time=10 clk=1  S=0  S1=1  S2=0  Y=0
# time=18 clk=0  S=0  S1=1  S2=0  Y=0
# time=27 clk=1  S=0  S1=0  S2=1  Y=0
# time=36 clk=0  S=0  S1=0  S2=1  Y=0
# time=45 clk=1  S=0  S1=0  S2=0  Y=1
# time=50 clk=1  S=1  S1=0  S2=0  Y=0
# time=54 clk=0  S=1  S1=0  S2=0  Y=0
# time=63 clk=1  S=1  S1=1  S2=0  Y=0
# time=72 clk=0  S=1  S1=1  S2=0  Y=0
# time=81 clk=1  S=1  S1=1  S2=1  Y=1
# time=90 clk=0  S=1  S1=1  S2=1  Y=1
# time=99 clk=1  S=1  S1=1  S2=1  Y=1
# time=108 clk=0  S=1  S1=1  S2=1  Y=1
# time=110 clk=0  S=0  S1=1  S2=1  Y=0
# time=117 clk=1  S=0  S1=0  S2=1  Y=0
# time=126 clk=0  S=0  S1=0  S2=1  Y=0
# time=135 clk=1  S=0  S1=0  S2=0  Y=1
# time=144 clk=0  S=0  S1=0  S2=0  Y=1
# time=153 clk=1  S=0  S1=0  S2=0  Y=1
# time=162 clk=0  S=0  S1=0  S2=0  Y=1
# time=171 clk=1  S=0  S1=0  S2=0  Y=1
# time=180 clk=0  S=0  S1=0  S2=0  Y=1
# time=189 clk=1  S=0  S1=0  S2=0  Y=1
# time=190 clk=1  S=1  S1=0  S2=0  Y=0
# time=198 clk=0  S=1  S1=0  S2=0  Y=0
# time=207 clk=1  S=1  S1=1  S2=0  Y=0
# time=216 clk=0  S=1  S1=1  S2=0  Y=0
# time=225 clk=1  S=1  S1=1  S2=1  Y=1
# time=230 clk=1  S=0  S1=1  S2=1  Y=0
# time=234 clk=0  S=0  S1=1  S2=1  Y=0
# time=243 clk=1  S=0  S1=0  S2=1  Y=0
# time=250 clk=1  S=1  S1=0  S2=1  Y=0
# time=252 clk=0  S=1  S1=0  S2=1  Y=0
# time=261 clk=1  S=1  S1=1  S2=0  Y=0
# time=270 clk=0  S=0  S1=1  S2=0  Y=0
# time=279 clk=1  S=0  S1=0  S2=1  Y=0
# time=288 clk=0  S=0  S1=0  S2=1  Y=0
# time=297 clk=1  S=0  S1=0  S2=0  Y=1
```

```
        S = 0; #80;
        S = 1; #40;
        S = 0; #20;
        S = 1; #20;
        S = 0; #20;
    end

    // Display all parameters
    initial begin
        $monitor("time=%g clk=%b  S=%b  S1=%b  S2=%b  Y=%b",$time,clk,S,S1,S2,Y);
    end
endmodule
```

Simulated Waveform



Simulation Time: **300ns**
1 Clock Cycle: **9ns**

Ans 3:

Source Code:

```
// Module for given 16-bit ALU
module p3(op, zero_flag, opmode, ip1, ip2);
    // Mode Declaration
    input [15:0] ip1,ip2;   // ALU Inputs
    input [2:0] opmode;     // ALU Operation Mode
    output reg [15:0] op;   // ALU Output
    output zero_flag;       // ALU Zero Flag

    always @(opmode or ip1 or ip2)
        case(opmode)
            3'b000: op = ip1 + ip2;
            3'b001: op = ip1 - ip2;
            3'b010: op = ip1 ^ ip2;
            3'b011: op = ip1 && ip2;
            3'b100: op = ip1 || ip2;
            3'b101: op = ip1 + 16'b0000_0001;
            3'b110: op = ip1 << 1;
            3'b111: op = ip1 >> 1;
            default: op = 3'bx;
        endcase

    assign zero_flag = (op == 16'b0);
endmodule
```

Text Output

```
VSIM 54> run
# opmode = 000
# ip1 = 0000000000100001
# ip2 = 0000000010010001
# op  = 0000000010110010
# zero_flag  = 0
# ----------
# opmode = 001
# ip1 = 0000000010010001
# ip2 = 0000000000100001
# op  = 0000000001110000
# zero_flag  = 0
# ----------
# opmode = 010
# ip1 = 0000000001010101
# ip2 = 0000000011111111
# op  = 0000000010101010
# zero_flag  = 0
# ----------
# opmode = 011
# ip1 = 0000000000110011
# ip2 = 0000000001010101
# op  = 0000000000000001
# zero_flag  = 0
# ----------
```

Test Bench:

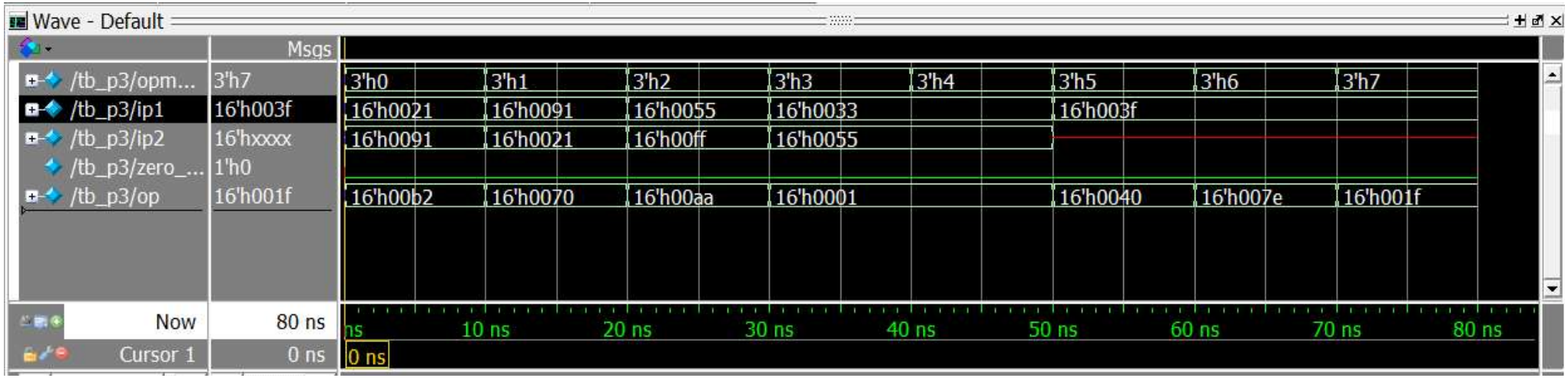
```
// Testbench for p3.v
module tb_p3();
    // Mode Declaration
    reg [2:0] opmode;
    reg [15:0] ip1,ip2;
    wire zero_flag;
    wire [15:0] op;

    p3 UUT (op, zero_flag, opmode, ip1, ip2);   // Instantiate p3

    // For ip1 and ip2, obtain outputs for each mode
    initial begin
        ip1 = 16'b0010_0001; ip2 = 16'b1001_0001; opmode = 3'b000; #10;
        ip1 = 16'b1001_0001; ip2 = 16'b0010_0001; opmode = 3'b001; #10;
        ip1 = 16'b0101_0101; ip2 = 16'b1111_1111; opmode = 3'b010; #10
        ip1 = 16'b0011_0011; ip2 = 16'b0101_0101; opmode = 3'b011; #10;
        ip1 = 16'b0011_0011; ip2 = 16'b0101_0101; opmode = 3'b100; #10;
        ip1 = 16'b0011_1111; ip2 = 16'bx;         opmode = 3'b101; #10;
```

```
        ip1 = 16'b0011_1111; ip2 = 16'bx;          opmode = 3'b110; #10;
        ip1 = 16'b0011_1111; ip2 = 16'bx;          opmode = 3'b111;
        #10;
    end

    // Display all parameters
    initial begin
          $monitor("ip1=%b ip2=%b opmode=%b
                    op=%b zero_flag=%b",
                   ip1,ip2,opmode,op,zero_flag);
    end
endmodule
```

Text Output

```
# opmode = 100
# ip1 = 0000000000110011
# ip2 = 0000000001010101
# op  = 0000000000000001
# zero_flag  = 0
# ----------
# opmode = 101
# ip1 = 0000000000111111
# ip2 = xxxxxxxxxxxxxxxx
# op  = 0000000001000000
# zero_flag  = 0
# ----------
# opmode = 110
# ip1 = 0000000000111111
# ip2 = xxxxxxxxxxxxxxxx
# op  = 0000000001111110
# zero_flag  = 0
# ----------
# opmode = 111
# ip1 = 0000000000111111
# ip2 = xxxxxxxxxxxxxxxx
# op  = 0000000000011111
# zero_flag  = 0
# ----------
```

Simulation Waveform



Simulation Time: **80ns**

Ans 4:

Source Code:

```
// Module for Encoder
module p4(op,z1,z2,z3,z4,z5,z6,z7,z8);
    // Mode Declaration
    // Sensor Signal for each zone
    input z1,z2,z3,z4,z5,z6,z7,z8;

    // Output
    output [2:0] op;

    assign op[0] = (z2|z4|z6|z8);
    assign op[1] = (z3|z4|z7|z8);
    assign op[2] = (z5|z6|z7|z8);
endmodule
```

Text Output

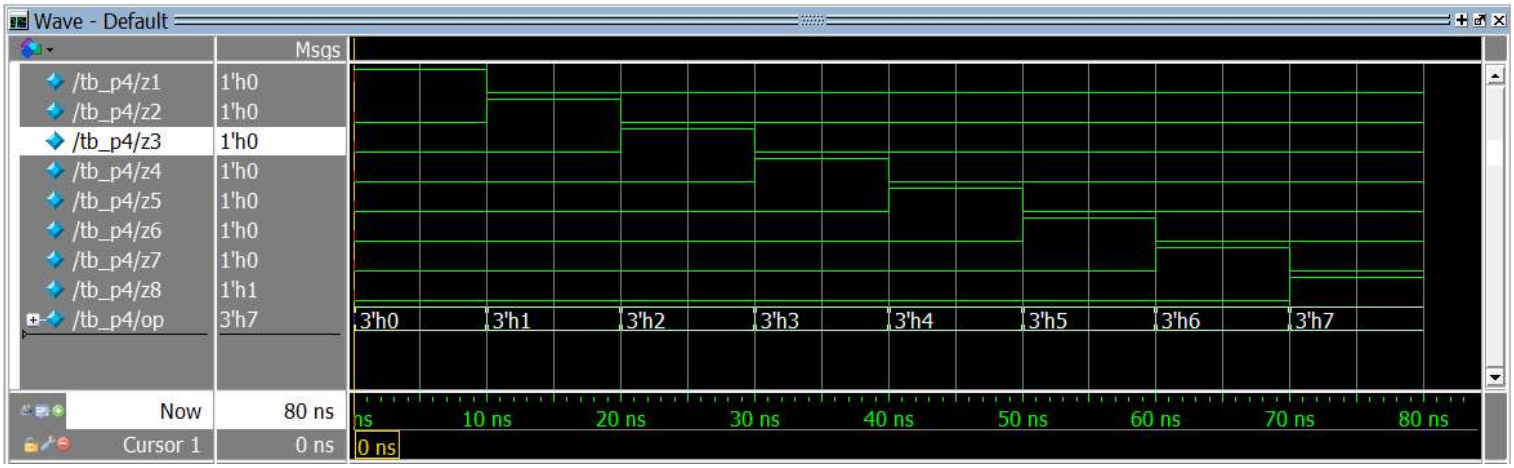
```
VSIM 58> run
# z1=1 z2=0 z3=0 z4=0 z5=0 z6=0 z7=0 z8=0 op=000
# z1=0 z2=1 z3=0 z4=0 z5=0 z6=0 z7=0 z8=0 op=001
# z1=0 z2=0 z3=1 z4=0 z5=0 z6=0 z7=0 z8=0 op=010
# z1=0 z2=0 z3=0 z4=1 z5=0 z6=0 z7=0 z8=0 op=011
# z1=0 z2=0 z3=0 z4=0 z5=1 z6=0 z7=0 z8=0 op=100
# z1=0 z2=0 z3=0 z4=0 z5=0 z6=1 z7=0 z8=0 op=101
# z1=0 z2=0 z3=0 z4=0 z5=0 z6=0 z7=1 z8=0 op=110
# z1=0 z2=0 z3=0 z4=0 z5=0 z6=0 z7=0 z8=1 op=111
```

Test bench:

```
// Testbench for p4.v
module tb_p4();
    // Mode Declaration
    reg z1,z2,z3,z4,z5,z6,z7,z8;
    wire [2:0] op;

    // Instantiate p4
    p4 UUT (op,z1,z2,z3,z4,z5,z6,z7,z8);

    // Testcase: Only one zone is intruded one time
    initial begin
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b10000000;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b01000000;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b00100000;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b00010000;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b00001000;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b00000100;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b00000010;
        #10;
        {z1,z2,z3,z4,z5,z6,z7,z8} = 8'b00000001;
        #10;
```

```
    end

    // Display all parameters
    initial begin
        $monitor("z1=%b z2=%b z3=%b z4=%b z5=%b z6=%b z7=%b z8=%b op=%b",z1,z2,z3,z4
,z5,z6,z7,z8,op);
    end
endmodule
```

Simulation Waveform



Simulation Time: **80ns**

Ans 5(a): Vat Buzzer Behaviour

Source Code:

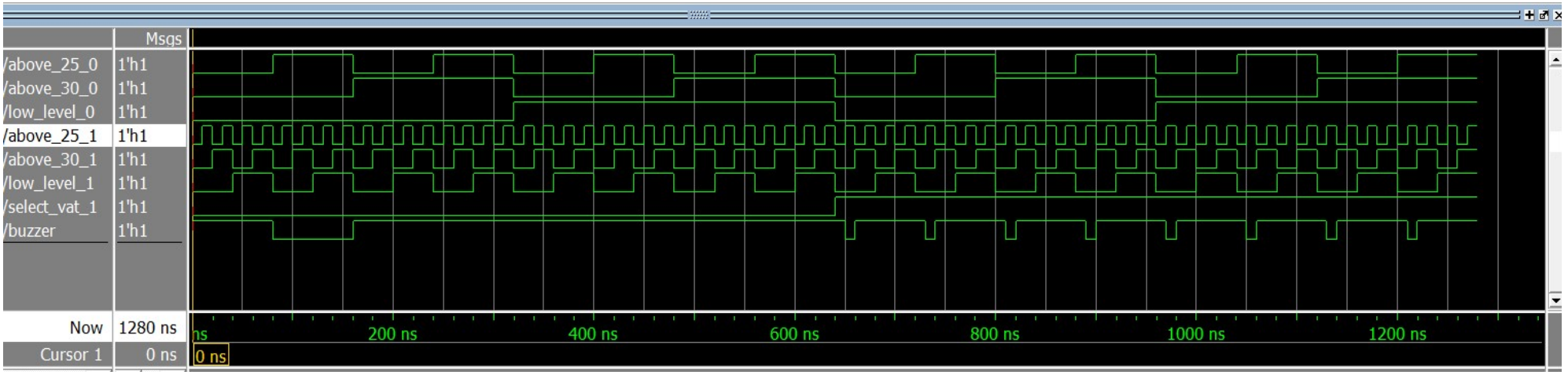
```
// Module (a)
module vat_buzzer_behavior ( output buzzer,
    input above_25_0, above_30_0, low_level_0,
    input above_25_1, above_30_1, low_level_1,
    input select_vat_1 );

    assign buzzer = select_vat_1 ? (low_level_1 | (above_30_1 | ~above_25_1))
                  : (low_level_0 | (above_30_0 | ~above_25_0));
endmodule
```

Testbench:

```
// Testbench for module (a) in p5.v
module tb_p5_vat_buzzer_behavior();
    // Mode Declaration
    reg above_25_0, above_30_0, low_level_0,above_25_1, above_30_1, low_level_1,sele
ct_vat_1;
    wire buzzer;

    // Instantiate Module
    vat_buzzer_behavior UUT (buzzer,above_25_0,above_30_0,low_level_0,above_25_1,abo
ve_30_1,low_level_1,select_vat_1);

    initial begin
        select_vat_1 = 0;
        low_level_0 = 0;
        above_30_0 = 0;
        above_25_0 = 0;
        low_level_1 = 0;
        above_30_1 = 0;
        above_25_1 = 0;

        // Testcase: All possible combinations of input parameters
        repeat(127) begin
            #10;
            select_vat_1 = select_vat_1 ^ (low_level_0 & above_30_0 & above_25_0 & l
ow_level_1 & above_30_1 & above_25_1);
            low_level_0 = low_level_0 ^ (above_30_0 & above_25_0 & low_level_1 & abo
ve_30_1 & above_25_1);
            above_30_0 = above_30_0 ^ (above_25_0 & low_level_1 & above_30_1 & above
_25_1);
```

```
            above_25_0 = above_25_0 ^ (low_level_1 & above_30_1 & above_25_1);
            low_level_1 = low_level_1 ^ (above_30_1 & above_25_1);
            above_30_1 = above_30_1 ^ above_25_1;
            above_25_1 = above_25_1 ^ 1;
        end
    end

    initial begin
        $display("select_vat_1  low_level_0  above_30_0  above_25_0  low_level_1  above_30_1  above_25_1  buzzer
");
        $monitor("    %b            %b           %b          %b          %b           %b          %b        %b"
,select_vat_1,low_level_0,above_30_0,above_25_0,low_level_1,above_30_1,above_25_1,buzzer);
    end
endmodule
```

Simulation Waveform



Simulation Time: **1280ns**

Ans 5(b): Alarm Priority

Source Code:

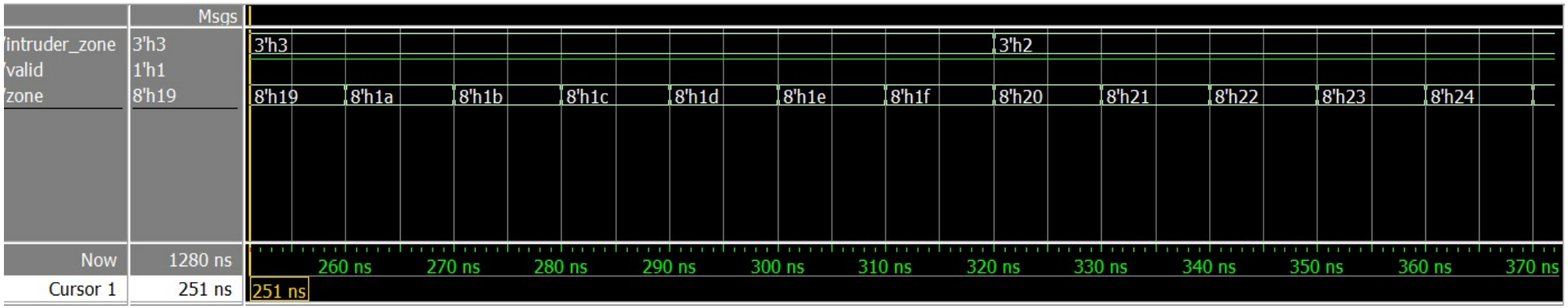
```
// Module (b)
module alarm_priority ( output [2:0] intruder_zone,
                        output valid,
                        input [1:8] zone );
    wire [1:8] winner;

    assign winner[1] = zone[1];
    assign winner[2] = zone[2] & ~zone[1];
    assign winner[3] = zone[3] & ~(zone[2] | zone[1]);
    assign winner[4] = zone[4] & ~(zone[3] | zone[2] | zone[1]);
    assign winner[5] = zone[5] & ~(zone[4] | zone[3] | zone[2] |
    zone[1]);
    assign winner[6] = zone[6] & ~(zone[5] | zone[4] | zone[3] |
    zone[2] | zone[1]);
    assign winner[7] = zone[7] & ~(zone[6] | zone[5] | zone[4] |
    zone[3] | zone[2] | zone[1]);
    assign winner[8] = zone[8] & ~(zone[7] | zone[6] | zone[5] |
    zone[4] | zone[3] | zone[2] |
    zone[1]);
    assign intruder_zone[2] = winner[5] | winner[6] |
    winner[7] | winner[8];
    assign intruder_zone[1] = winner[3] | winner[4] |
    winner[7] | winner[8];
    assign intruder_zone[0] = winner[2] | winner[4] |
    winner[6] | winner[8];

    assign valid = zone[1] | zone[2] | zone[3] | zone[4] |
    zone[5] | zone[6] | zone[7] | zone[8];
endmodule
```

Testbench:

```
// Testbench for module (b) in p5.v
module tb_p5_alarm_priority();
    // Mode Declaration
    wire [2:0] intruder_zone;
    wire valid;
    reg [1:8] zone;

    // Instantiate Module
    alarm_priority DUT (intruder_zo
```

```
ne,valid,zone);

    // Testcase: All possible combinations of input parameters
    initial begin
        zone = 8'b0;
        repeat(127) begin
            #10 zone = zone+1;
        end
    end

    initial begin
        $display("zone intruder_zone valid");
        $monitor(" %d       %d         %d",zone,intruder_zone,valid);
    end
endmodule
```

Simulated Waveform



Waveform shown from **251ns to 370ns**

Simulation Time: **1280ns**

Ans 6:

Model 1 (Decade Counter)

Source Code:

```
// Given Model
module decade_counter ( output reg [3:0] q,
                        input clk );

    always @(posedge clk)
        q <= q == 9 ? 0 : q + 1;
endmodule
```

Note: Since, initial value of the counter is not specified, assume it to be zero

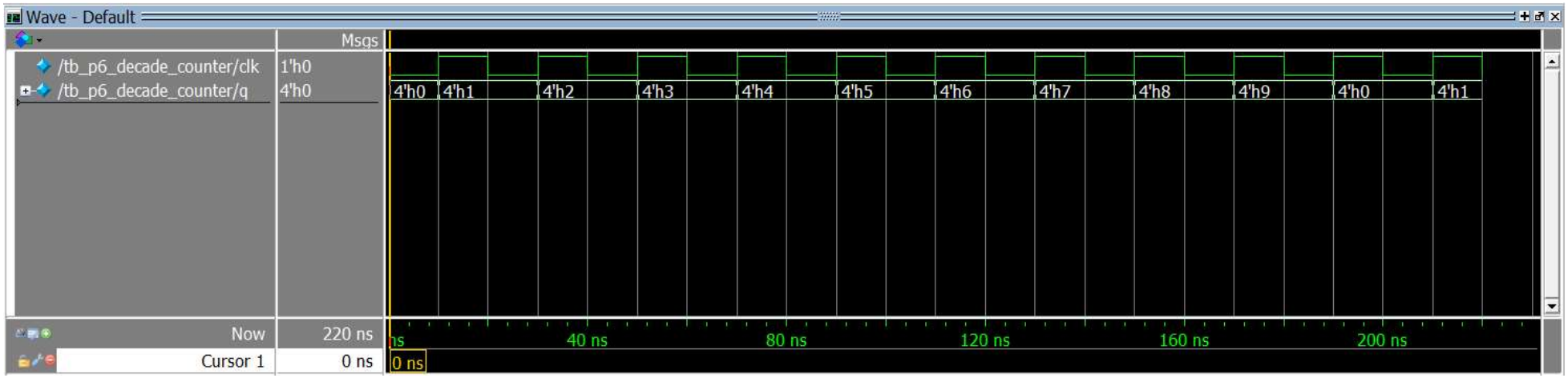
Testbench:

```
// Testbench for given model
module tb_p6_decade_counter();
    // Mode declaration
    reg clk;
    wire [3:0] q;

    // Instantiate given model
    decade_counter UUT (q,clk);

    initial begin
        clk = 0;
        // Assume initial value is zero
        UUT.q = 4'b0000;
    end

    always #10 clk = ~clk;

    // Display all parameters
    initial begin
        $display("time\t clk q");
        $monitor("%g\t  %b  %h",$time,clk,q);
    end
endmodule
```

Text Output

```
VSIM 76> run
# time    clk q
# 0         0  0
# 10        1  1
# 20        0  1
# 30        1  2
# 40        0  2
# 50        1  3
# 60        0  3
# 70        1  4
# 80        0  4
# 90        1  5
# 100       0  5
# 110       1  6
# 120       0  6
# 130       1  7
# 140       0  7
# 150       1  8
# 160       0  8
# 170       1  9
# 180       0  9
# 190       1  0
# 200       0  0
# 210       1  1
```

Simulation Waveform



Simulation Time: **210ns**
1 Clock Cycle: **10ns**

Model 2 (Decoded Counter)

Source Code:

```
// Given Model
module decoded_counter ( output ctrl,
                         input clk );
    reg [3:0] count_value;
    always @(posedge clk)
        count_value <= count_value + 1;

    assign ctrl = count_value == 4'b0111 ||
                  count_value == 4'b1011;
endmodule
```

Note: Since, initial value of the counter is not specified, assume it to be zero

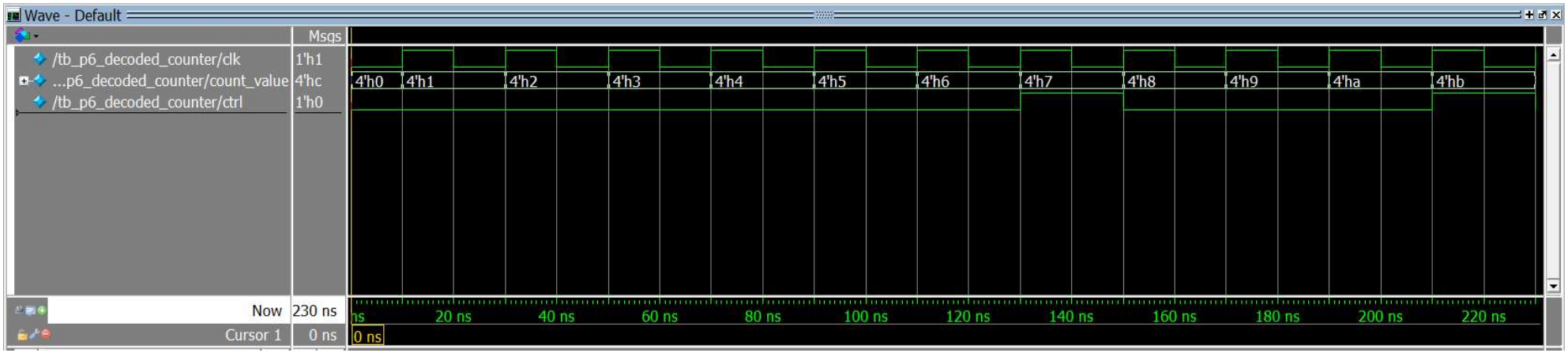
Testbench:

```
// Testbench for given model
module tb_p6_decoded_counter();
    // Mode Declaration
    reg clk;
    wire ctrl;
    wire [3:0] count_value;

    // Instantiate given model
    decoded_counter DUT (ctrl,clk);

    assign count_value = DUT.count_value;
    initial begin
        clk = 0;
        // Assume initial value is zero
        DUT.count_value = 4'b0000;
    end

    always #10 clk = ~clk;

    // Display all parameters
    initial begin
        $display("time\t clk count_value ctrl");
        $monitor("%g\t   %b      %d       %b",$time,clk,DUT.count_value,ctrl);
    end
endmodule
```

Text Output

```
VSIM 85> run
# time    clk count_value ctrl
# 0         0         0       0
# 10        1         1       0
# 20        0         1       0
# 30        1         2       0
# 40        0         2       0
# 50        1         3       0
# 60        0         3       0
# 70        1         4       0
# 80        0         4       0
# 90        1         5       0
# 100       0         5       0
# 110       1         6       0
# 120       0         6       0
# 130       1         7       1
# 140       0         7       1
# 150       1         8       0
# 160       0         8       0
# 170       1         9       0
# 180       0         9       0
# 190       1        10       0
# 200       0        10       0
# 210       1        11       1
```

Simulated Waveform:

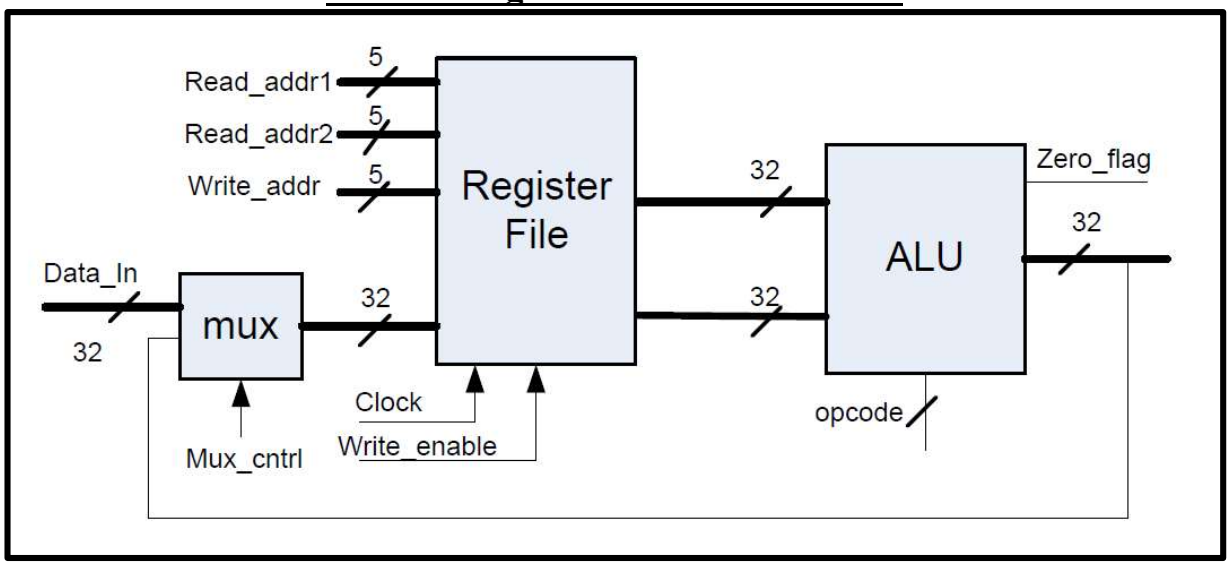


Simulation Time: **220ns**

1 Clock Cycle: **10ns**

Ans 7:

Circuit Diagram of the Architecture



Source Code:

```
// Module for 32x32 Register File
module Register_File (Data_Out_1, Data_Out_2, Data_in, Read_Addr_1, Read_Addr_2, Write_Ad
dr, Write_Enable, Clock);
    // Mode Declaration
    input [31:0] Data_in;                                // Data to be written
    input [4:0] Read_Addr_1, Read_Addr_2, Write_Addr; // Addresses to Read and Write
    input Write_Enable;                                  // Enable for Write Mode
    input Clock;                                         // Clock
    output [31:0] Data_Out_1, Data_Out_2;                // Data to be read from Read_Addr_1
 & Read_Addr_2

    reg [31:0] Reg_File [31:0];                          // Memory declaration for 32 32-
bit Words

    assign Data_Out_1 = Reg_File[Read_Addr_1];
    assign Data_Out_2 = Reg_File[Read_Addr_2];

    always @ (posedge Clock) begin
        if (Write_Enable) begin
            Reg_File[Write_Addr] = Data_in;
        end
    end
endmodule

// Module for ALU
module alu (op,zero_flag,ip0,ip1,opcode);
    // Mode Declaration
```

```
    output zero_flag;       // Zero Flag
    output [31:0] op;       // ALU Output
    input [1:0] opcode;     // ALU Mode of Operation
    input [31:0] ip0,ip1;   // ALU Inputs

    //    opcode :     00             01              10             11
    // Operation :  Addition      Subtraction     Bitwise AND    Bitwise OR
    assign op = opcode[0] ? (opcode[1] ? (ip0&ip1) : (ip0|ip1)) : (opcode[1] ? (ip0-
ip1) : (ip0+ip1));
    assign zero_flag = (op == 32'b0);
endmodule

// Module for 2:1 Multiplexer
module mux2_1 (op,ip0,ip1,Mux_cntrl);
    output [31:0] op;       // MUX Output
    input Mux_cntrl;        // MUX Control Line
    input [31:0] ip0,ip1;   // MUX Inputs

    // Mux_cntrl :    0            1
    //    Output : Input 0      Input 1
    assign op = Mux_cntrl ? ip1 : ip0;
endmodule


// Module for given Architecture
module p7 (zero_flag, result, Data_in, Read_Addr_1, Read_Addr_2, Write_Addr, Write_Enable
, Mux_cntrl, opcode, Clock);
    // Mode Declaration
    output zero_flag;
    output [31:0] result;
    input [31: 0] Data_in;
    input [4: 0] Read_Addr_1, Read_Addr_2, Write_Addr;
    input Mux_cntrl, Write_Enable, Clock;
    input [1:0] opcode;

    wire [31:0] mux_OP, Data_Out_1, Data_Out_2, result;

//  Import Modules for all blocks
/*    Module      | Name of  |         Parameters
 *                | Instance |
------------------------------------------------------------*/
    mux2_1            MUX      (mux_OP,Data_in,result,Mux_cntrl);
    Register_File   regFile    (Data_Out_1, Data_Out_2, mux_OP, Read_Addr_1, Read_Addr_2,
 Write_Addr, Write_Enable, Clock);
    alu               ALU      (result,zero_flag,Data_Out_1,Data_Out_2,opcode);
endmodule
```

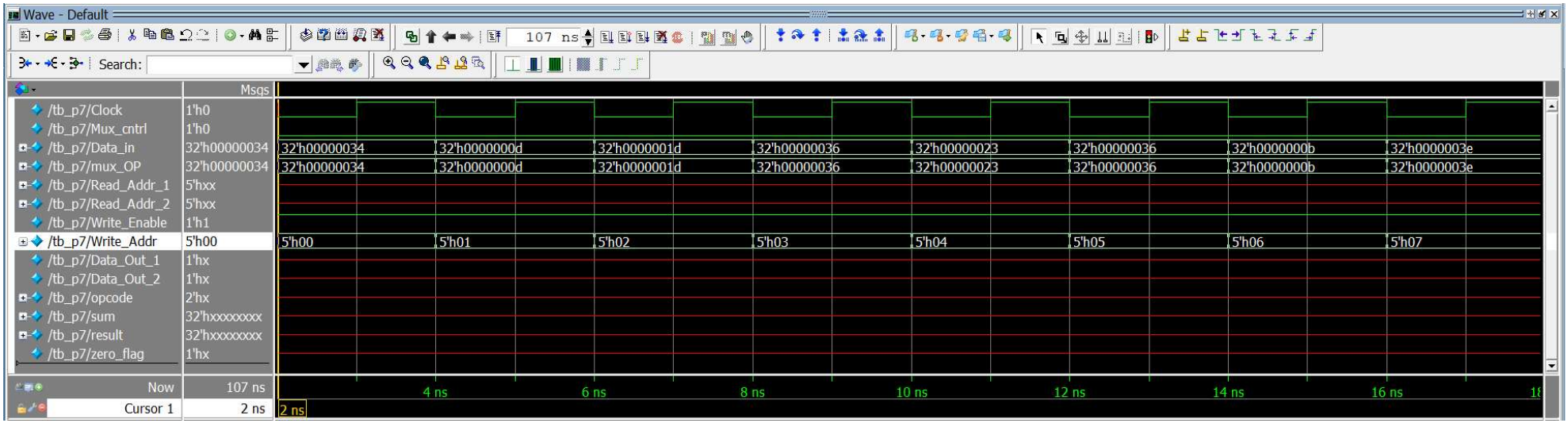
Test Bench:

```
// Testbench for p7 module in p7.v
module tb_p7 ();
    // Mode Declarations
    // All terms explained in p7.v
    wire zero_flag;
    wire [31:0] result;
    reg [31:0] Data_in;
    reg [4:0] Read_Addr_1, Read_Addr_2, Write_Addr;
    reg Write_Enable, Mux_cntrl, Clock = 0;
    reg [1:0] opcode;

    always #1 Clock = ~Clock;

    // Instantiate p7
    p7 DUT_p7 (zero_flag, result, Data_in, Read_Addr_1, Read_Addr_2, Write_Addr, Write_En
able, Mux_cntrl, opcode, Clock);

    wire [31:0] sum, mux_OP;
    assign sum = DUT_p7.regFile.Reg_File[31];   // Value Stored at register 31 (Sum of in
put Data)
    assign mux_OP = DUT_p7.mux_OP;              // Output of 2:1 Multiplexer
    assign Data_Out_1 = DUT_p7.Data_Out_1;      // Data Output 1 from Register File
    assign Data_Out_2 = DUT_p7.Data_Out_2;      // Data Output 2 from Register File

    integer i;
    initial begin
        // Write Data to Register File
        Mux_cntrl = 0;
        Write_Enable = 1;
        for(i = 0; i < 25; i = i+1) begin
            #2;
            Write_Addr = i;
            Data_in = $urandom%(64);
        end
        // Initialize register 31 to zero to avoid garbage values
        #2;
        Write_Addr = 31;
        Data_in = 31'b0;
        // Calculate Sum of data in registers 0 to 24
        #2;
        Data_in = 31'bx;
        Mux_cntrl = 1;
        Read_Addr_1 = 31;
       Read_Addr_2 = 31;
```

```
        opcode = 0;

        for (i = 0; i < 25; i = i+1) begin
            #2 Read_Addr_1 = i;
        end
    end

    // Display all Parameters
    initial begin
        $monitor("time=%g\tMux_cntrl=%b\tData_in=%d\tmux_OP=%d\tRead_Addr_1=%d\tRead_Addr_2=%d\tWrite_Enable=%b\tWrit
e_Addr=%d\tData_Out_1=%d\tData_Out_2=%d\topcode=%b\tSum=%d",
$time,Mux_cntrl,Data_in,mux_OP,Read_Addr_1,Read_Addr_2,Write_Enable,Write_Addr,Data_Out_1,Data_Out_2,opcode,sum);
    end
endmodule
```

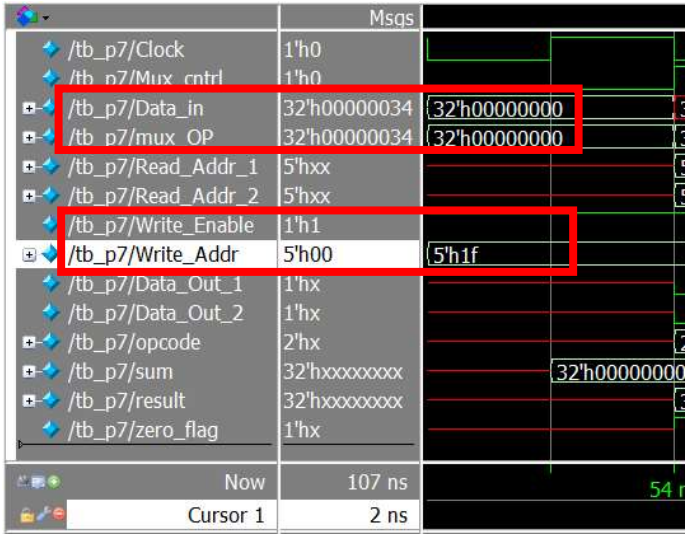
Simulated Waveforms: (1 Clock Cycle = **1ns**)

Writing Data to Register File



Simulation Time: **2ns to 18ns**

Initialize Register 31 to zero to avoid garbage values:



Simulation Time: **52ns to 54ns**

Calculate Sum of Data in Registers 0 to 24:

Simulation Time: **98ns to 108ns**