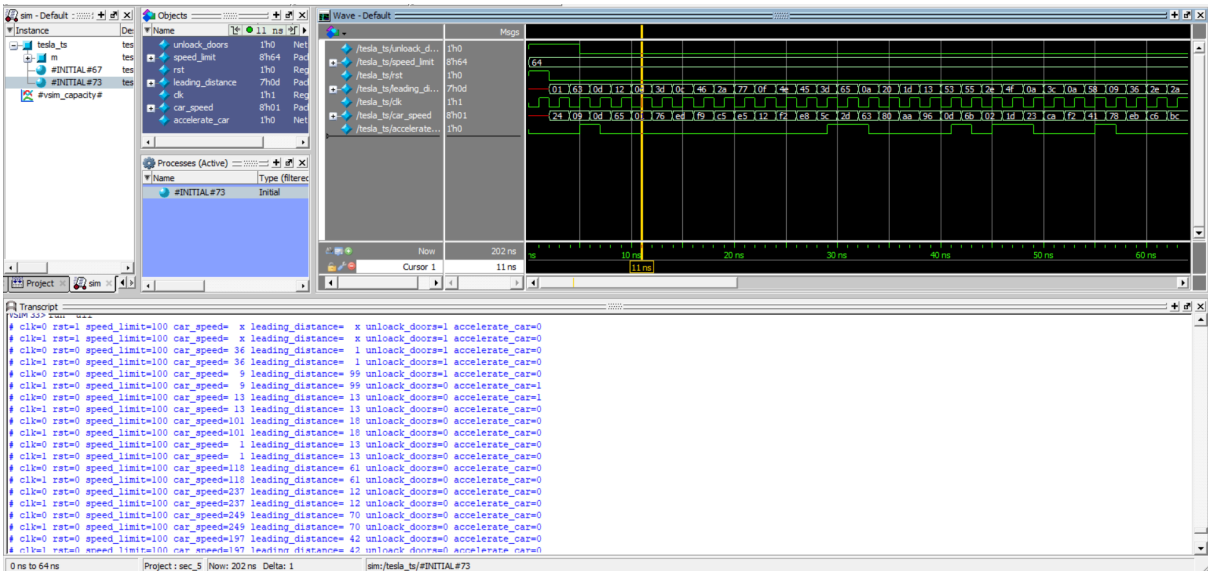
q1

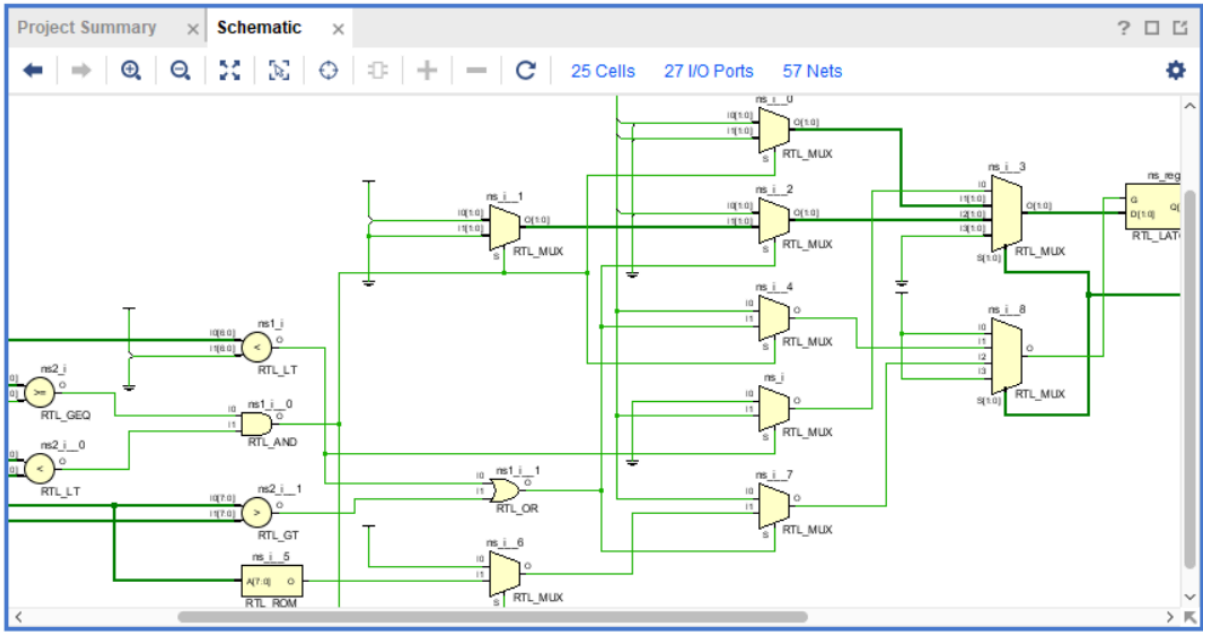
```
module tesla #( parameter MIN_DISTANCE = 40 )
              ( output reg unloack_doors,accelerate_car,
                input [7:0] speed_limit,car_speed,
                input [6:0] leading_distance,
                input clk,rst);
parameter STOP = 2'b00;
parameter ACCELERATE = 2'b01;
parameter DECELERATE = 2'b10;

reg [1:0] cs,ns;

always @(posedge clk , posedge rst) begin
    if (rst)
        cs <= STOP;
    else
        cs <= ns;
end
always @(*) begin
    case (cs)
        STOP       : begin
                     if (leading_distance < MIN_DISTANCE)
                         ns = STOP;
                     else
                         ns = ACCELERATE;
                     end
        ACCELERATE : begin
                     if ((leading_distance >= MIN_DISTANCE) && (car_speed < speed_limit))
                         ns = ACCELERATE;
                     else if ((leading_distance < MIN_DISTANCE) || (car_speed > speed_limit))
                         ns = DECELERATE;
                     end
        DECELERATE : begin
                     if ((leading_distance < MIN_DISTANCE) || (car_speed > speed_limit))
                         ns = DECELERATE;
                     else if ((leading_distance >= MIN_DISTANCE) && (car_speed < speed_limit))
                         ns = ACCELERATE;
                     else if (car_speed == 0)
                         ns = STOP;
                     end
        default    : ns = STOP;
    endcase
end
```

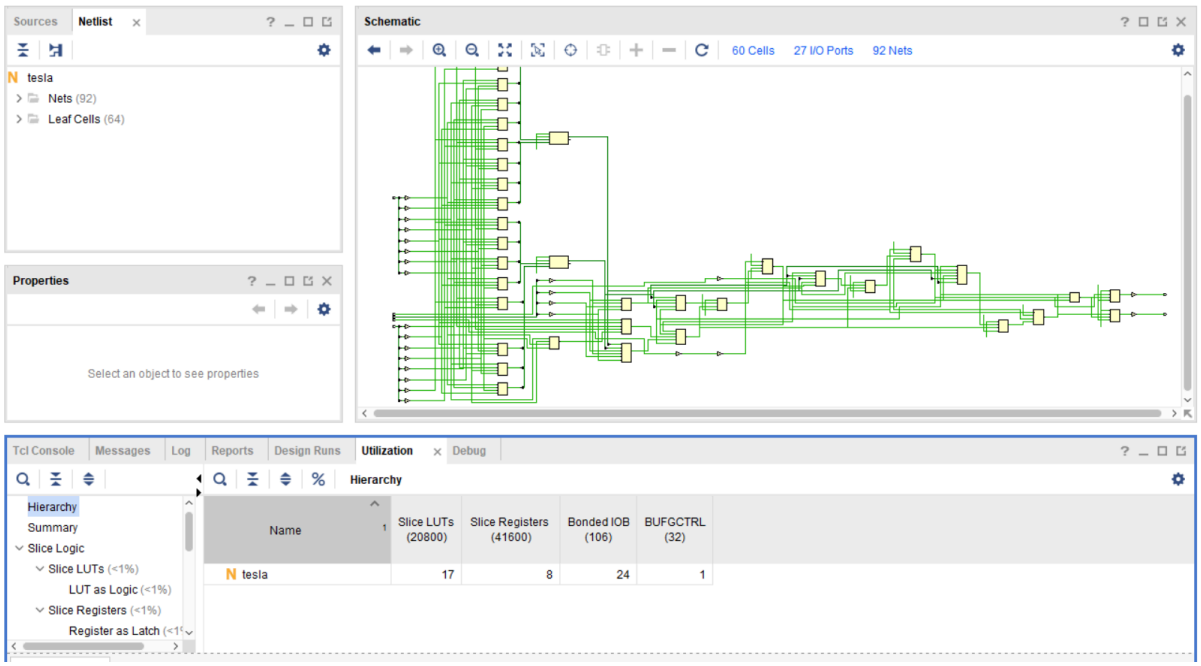
```
always @(cs) begin
    case (cs)
        STOP       : begin
                     unloack_doors = 1;
                     accelerate_car = 0;
                     end
        ACCELERATE : begin
                     unloack_doors = 0;
                     accelerate_car = 1;
                     end
        DECELERATE : begin
                     unloack_doors = 0;
                     accelerate_car = 0;
                     end
    endcase
end
endmodule

module tesla_ts ();
wire unloack_doors,accelerate_car;
reg [7:0] speed_limit,car_speed;
reg [6:0] leading_distance;
reg clk,rst;
tesla m (unloack_doors,accelerate_car,speed_limit,car_speed,leading_distance,clk,rst);
initial begin
    clk = 0;
    forever begin
        #1 clk = ~clk;
    end
end
initial begin
    rst = 1;
    speed_limit = 8'd100;
    @(negedge clk)
    rst = 0;
    repeat (100) begin
    car_speed = $random;
    leading_distance = $random;
    @(negedge clk);
    end
    $stop;
end
initial begin
    $monitor ("clk=%b rst=%b speed_limit=%d car_speed=%d leading_distance=%d unloack_doors=%b accelerate_car=%b",clk,rst,speed_limit,car_speed,leading_distance,unloack_doors,accelerate_car);

end
endmodule
```



dofile.do

```
1  vlib work
2  vlog assignment5q1.v
3  vsim -voptargs=+acc work.tesla_ts
4  add wave*
5  run -all
6
```



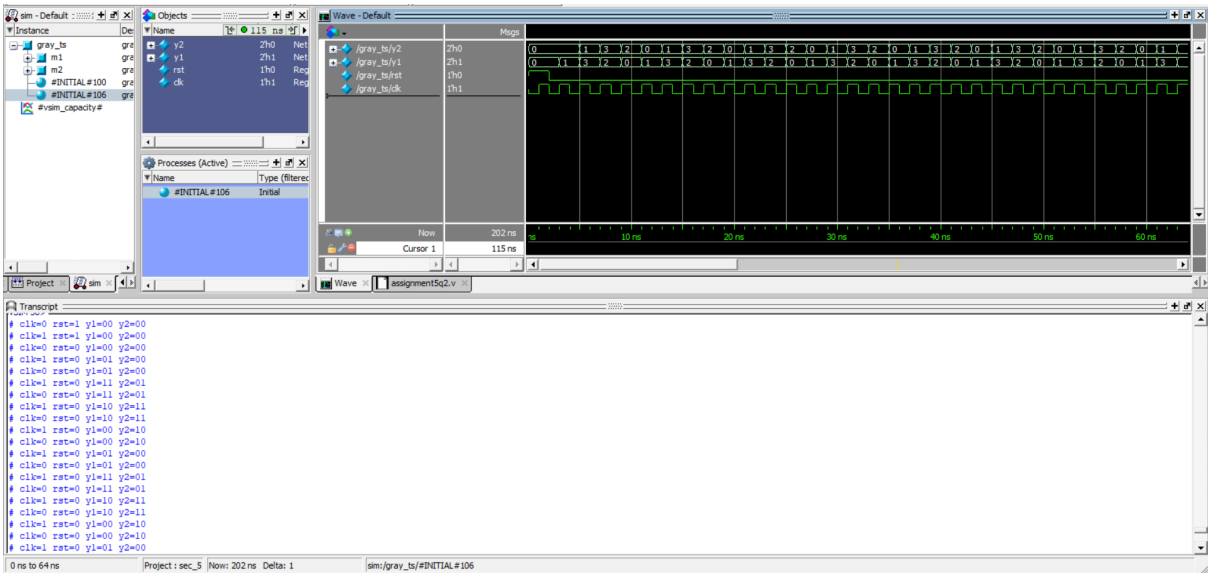

Sources
Netlist
tesla
Nets (92)
Leaf Cells (64)
Properties
Select an object to see properties
Schematic
60 Cells
27 I/O Ports
92 Nets
Tcl Console
Messages
Log
Reports
Design Runs
Utilization
Debug
Hierarchy
Summary
Slice Logic
Slice LUTs (<1%)
LUT as Logic (<1%)
Slice Registers (<1%)
Register as Latch
Hierarchy
Name
Slice LUTs (20800)
Slice Registers (41600)
Bonded IOB (106)
BUFGCTRL (32)
tesla
17
8
24
1

q2

```
module gray_fsm ( output reg [1:0] y,
                  input clk,rst);
parameter A = 2'b00;
parameter B = 2'b01;
parameter C = 2'b10;
parameter D = 2'b11;

reg [1:0] cs,ns;
always @(posedge clk , posedge rst) begin
    if (rst)  cs <= A;
    else cs <= ns;
end
always @(*) begin
    case (cs)
        A      : ns = B;
        B      : ns = C;
        C      : ns = D;
        D      : ns = A;
        default: ns = A;
    endcase
end
always @(cs) begin
    case (cs)
        A      : y = 2'b00;
        B      : y = 2'b01;
        C      : y = 2'b11;
        D      : y = 2'b10;
    endcase
end
endmodule
module gray_fsm_ts ();
wire [1:0] y;
reg  clk,rst;
gray_fsm m (y,clk,rst);
initial begin
    clk = 0;
    forever #1 clk = ~clk;
end
initial begin
    rst = 1;
    @(negedge clk)
    rst = 0;
    repeat (100) @(negedge clk);
    $stop;
end
initial begin
    $monitor ("clk=%b rst=%b y=%b",clk,rst,y);
end
```

```
module gray_counter ( output reg [1:0] gray_out,
                      input clk, rst );
    reg [1:0] counter;

    always @(posedge clk , posedge rst) begin
        if (rst) begin
            counter <= 2'b00;
            gray_out <= 2'b00;
        end
        else begin
            counter <= counter + 2'b01;
            gray_out <= {counter[1],^ counter};
        end
    end
endmodule

module gray_counter_ts ();
wire [1:0] y;
reg  clk,rst;
gray_counter m (y,clk,rst);
initial begin
    clk = 0;
    forever begin
        #1 clk = ~clk;
    end
end
initial begin
    rst = 1;
    @(negedge clk)
    rst = 0;
    repeat (100) begin
    @(negedge clk);
    end
    $stop;
end
initial begin
    $monitor ("clk=%b rst=%b y=%b",clk,rst,y);

end
endmodule
```

```
module gray_ts ();
wire [1:0] y1,y2;
reg  clk,rst;
gray_fsm m1 (y1,clk,rst);
gray_counter m2 (y2,clk,rst);
initial begin
    clk = 0;
    forever begin
        #1 clk = ~clk;
    end
end
initial begin
    rst = 1;
    @(negedge clk)
    rst = 0;
    repeat (100) begin
    @(negedge clk);
    end
    $stop;
end
initial begin
    $monitor ("clk=%b rst=%b y1=%b y2=%b",clk,rst,y1,y2);

end
endmodule
```

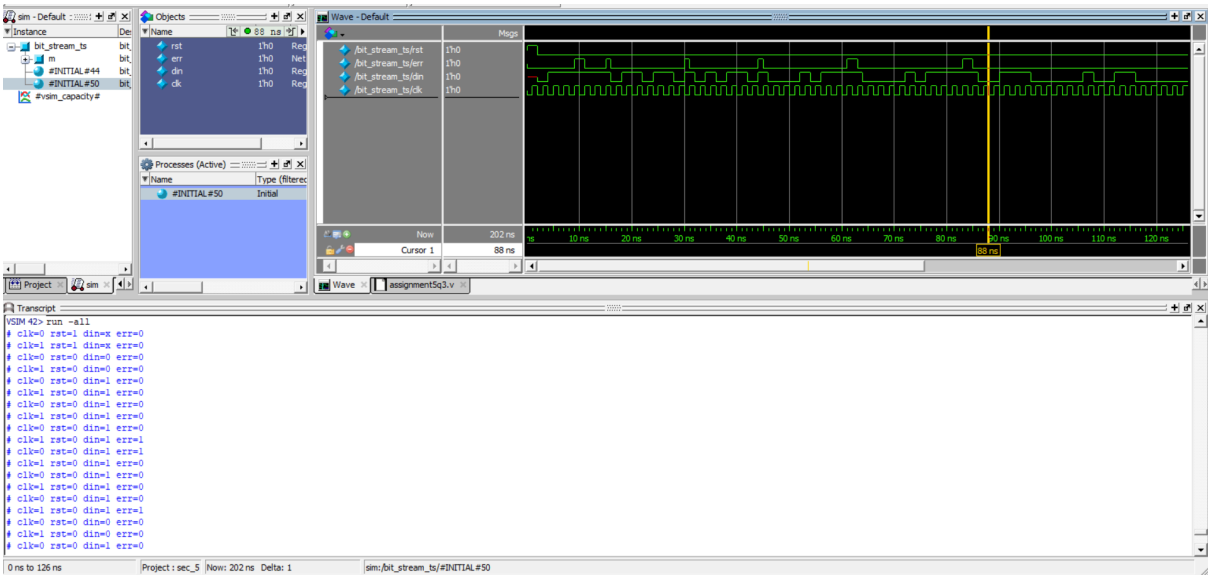


```
vlib work
vlog assignment5q2.v
vsim -voptargs=+acc work.gray_ts
add wave*
run -all

```

q3

```
module  bit_stream ( output [1:0] err,
          input din,clk,rst);
parameter start    = 3'b000;
parameter d0_is_1  = 3'b001;
parameter d1_is_1  = 3'b010;
parameter d0_not_1 = 3'b011;
parameter d1_not_1 = 3'b100;
reg [1:0] cs,ns;
always @(posedge clk , posedge rst) begin
    if (rst)
        cs <= start;
    else
        cs <= ns;
end
always @(cs) begin
    case (cs)
        start  : begin
                 if (din)
                    ns = d0_is_1;
                 else
                    ns = d0_not_1;
                 end
        d0_is_1  : begin
                   if (din)
                      ns = d1_is_1;
                   else
                      ns = d1_not_1;
                   end
        d1_is_1   : ns = start;
        d0_not_1  : ns = d1_not_1;
        d1_not_1  : ns = start;
        default: ns = start;
    endcase
end

assign err = (cs == d1_is_1 && din == 1) ? 2'b11 : 2'b01;

endmodule
```

```
module  bit_stream_ts ();
wire [1:0] err;
reg  din,clk,rst;
bit_stream m (err,din,clk,rst);
initial begin
    clk = 0;
    forever begin
        #1 clk = ~clk;
    end
end
initial begin
    rst = 1;
    @(negedge clk)
    rst = 0;
    repeat (100) begin
    din = $random;
    @(negedge clk);
    end
    $stop;
end
initial begin
    $monitor ("clk=%b rst=%b din=%b err=%b",clk,rst,din,err);

end
endmodule
```



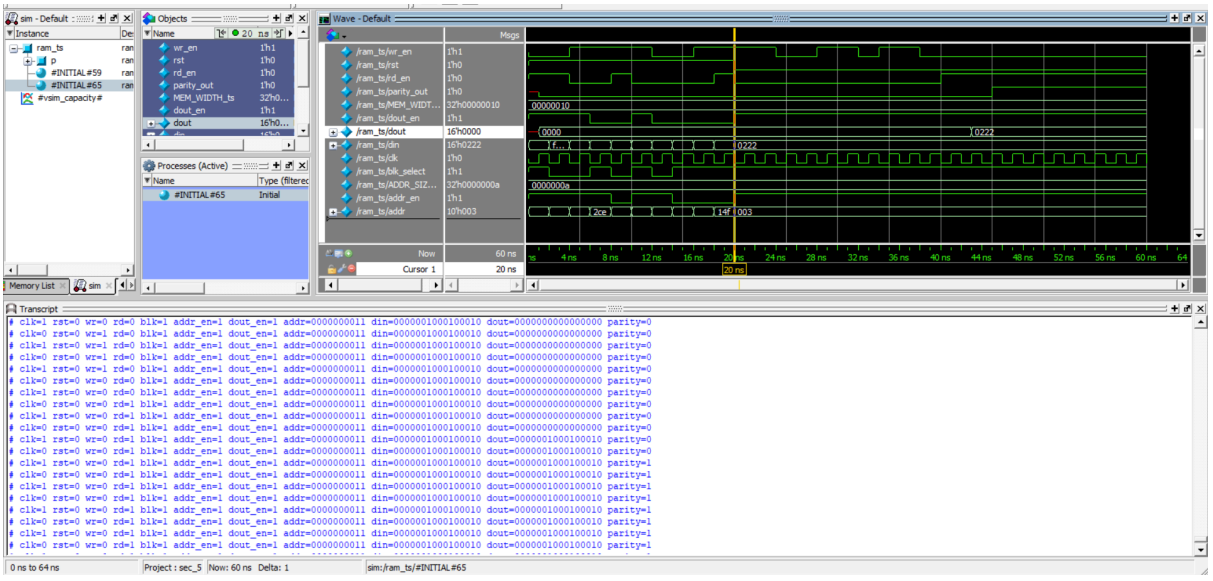
dofile3.do

```
vlib work
vlog assignment5q3.v
vsim -voptargs=+acc work.bit_stream_ts
add wave*
run -all

```

q4

```
module ram #( parameter MEM_WIDTH = 16,
              parameter MEM_DEPTH = 1024,
              parameter ADDR_SIZE = 10,
              parameter ADDR_PIPELINE = "FALSE",
              parameter DOUT_PIPELINE = "TRUE",
              parameter PARITY_ENABLE = 1)
             ( output reg [MEM_WIDTH-1:0] dout,
               output reg parity_out,
               input [MEM_WIDTH-1:0] din,
               input [ADDR_SIZE-1:0] addr,
               input wr_en,rd_en,blk_select,addr_en,dout_en,clk,rst);
  reg [MEM_WIDTH-1:0] mem [MEM_DEPTH-1:0] ;
  reg [ADDR_SIZE-1:0] addr_reg;
  reg [MEM_WIDTH-1:0] dout_reg;
  always @(posedge clk) begin
    if (rst)
       dout_reg <= 0;
    else if (addr_en && (ADDR_PIPELINE == "TRUE"))
       addr_reg <= addr;
  end
  always @(posedge clk) begin
    if (rst)
       dout_reg <= 0;
    else if (wr_en && blk_select)
       mem[addr] <= din;
    else if (rd_en && blk_select)
       dout_reg <= mem[(ADDR_PIPELINE=="TRUE") ? addr_reg : addr];
  end
  always @(posedge clk) begin
    if (rst)
       dout <= 0;
    else if (dout_en && (DOUT_PIPELINE=="TRUE"))
       dout <= dout_reg;
    else if (!DOUT_PIPELINE)
       dout <= dout_reg;
  end
  always @(posedge clk) begin
      if (rst)
         parity_out <= 0;
      else
          if (PARITY_ENABLE)
              parity_out <= ^dout;
            else
              parity_out <= 0;
  end
endmodule
```

```
module ram_ts();
parameter MEM_WIDTH_ts = 16;
parameter ADDR_SIZE_ts = 10;
wire [MEM_WIDTH_ts-1:0] dout;
wire parity_out;
reg [MEM_WIDTH_ts-1:0] din;
reg [ADDR_SIZE_ts-1:0] addr;
reg wr_en,rd_en,blk_select,addr_en,dout_en,clk,rst;

ram #(.MEM_WIDTH(MEM_WIDTH_ts),.ADDR_SIZE(ADDR_SIZE_ts)) p (dout,parity_out,din,addr,wr_en,rd_en,blk_select,addr_en,dout_en,clk,rst);

initial begin
    clk = 0;
    forever begin
        #1 clk = ~clk;
    end
end
initial begin
    rst = 1;
    repeat (10) begin
        wr_en = $random;
        rd_en = $random;
        addr = $random;
        din = $random;
        addr_en = $random;
        dout_en = $random;
        blk_select = $random;
        @(negedge clk);
    end
    rst = 0;
    rd_en = 0;
    addr_en = 1;
    dout_en = 0;
    dout_en = 1;
    blk_select = 1;
    addr = 10'b0000000011;
    din = 16'h222;
    repeat(10) begin
        wr_en = $random;
        @(negedge clk);
    end
    rd_en = 1;
    wr_en = 0;
    dout_en = 1;
    din = 16'h222;
    repeat(10) @(negedge clk);
    $stop;
end
```

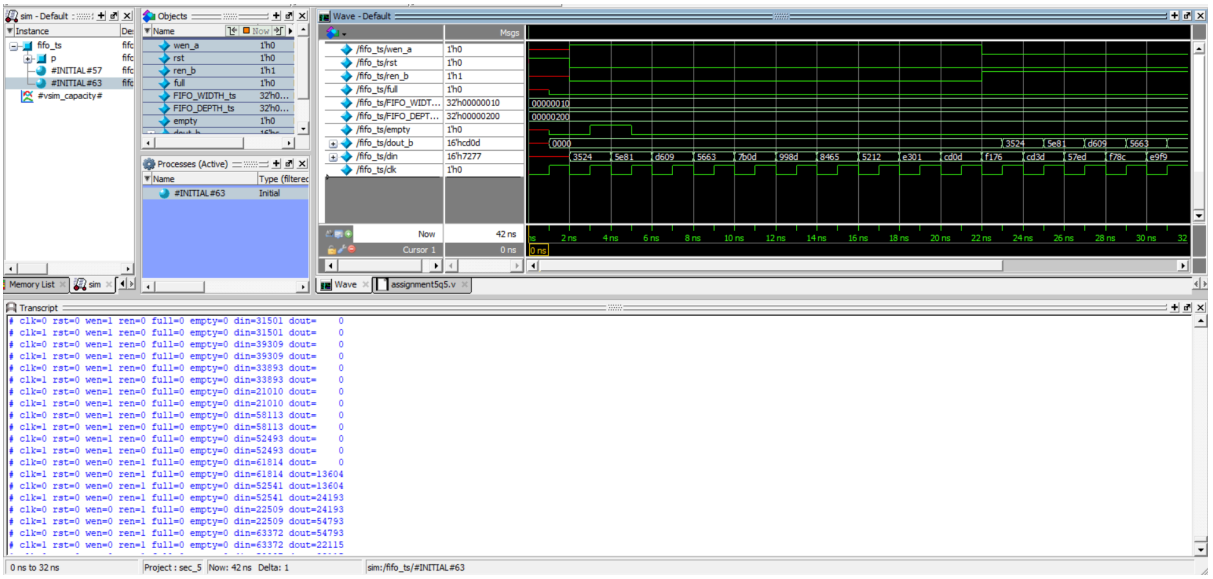


```
vlib work
vlog assignment5q4.v
vsim -voptargs=+acc work.ram_ts
add wave*
run -all

```

q5

```
module fifo # ( parameter FIFO_WIDTH = 16 ,
                parameter FIFO_DEPTH = 512)
              ( output reg full,empty,
                output reg [FIFO_WIDTH-1:0] dout_b,
                input [FIFO_WIDTH-1:0] din,
                input wen_a,ren_b,clk_a,clk_b,rst);
 reg [FIFO_WIDTH-1:0] mem [FIFO_DEPTH-1:0] ;
 reg [8:0] rd_ptr,wr_ptr,count;
 always @(posedge clk_a) begin
    if (rst) begin
        wr_ptr <= 0;
        full <= 0;
        count <= 0;
    end
    else begin
        if ((wen_a) && (!full)) begin
            mem[wr_ptr] <= din;
            wr_ptr <= wr_ptr + 1;
            count <= count + 1;
        end
    end
    if (count == (FIFO_DEPTH-1))
        full <= 1;
    else
        full <= 0;
 end
  always @(posedge clk_b) begin
    if (rst) begin
        dout_b <= 0;
        rd_ptr <= 0;
        empty <= 0;
    end
    else begin
        if ((ren_b) && (!empty)) begin
            dout_b <= mem[rd_ptr];
            rd_ptr <= rd_ptr + 1;
            count <= count - 1;
        end
    end
    if (count == 0)
          empty <= 1;
    else
          empty <= 0;
 end
endmodule
```

```
module  fifo_ts ();
parameter FIFO_WIDTH_ts = 16 ;
parameter FIFO_DEPTH_ts = 512;
wire full,empty;
wire [FIFO_WIDTH_ts-1:0] dout_b;
reg [FIFO_WIDTH_ts-1:0] din;
reg wen_a,ren_b,clk,rst;

fifo #(.FIFO_WIDTH(FIFO_WIDTH_ts),.FIFO_DEPTH(FIFO_DEPTH_ts)) p (full,empty,dout_b,din,wen_a,ren_b,clk,clk,rst);

initial begin
    clk = 0;
    forever begin
        #1 clk = ~clk;
    end
end
initial begin
    rst = 1;
    @(negedge clk);
    rst = 0;
    wen_a = 1;
    ren_b = 0;
    repeat(10) begin
        din = $random;
        @(negedge clk);
    end
    wen_a = 0;
    ren_b = 1;
    repeat(10) begin
        din = $random;
        @(negedge clk);
    end
    $stop;
end
initial begin
    $monitor ("clk=%b rst=%b wen=%b ren=%b full=%b empty=%b din=%d dout=%d",clk,rst,wen_a,ren_b,full,empty,din,dout_b);

end

endmodule
```




```
vlib work
vlog assignment5q5.v
vsim -voptargs=+acc work.fifo_ts
add wave*
run -all

```

q6

```
    reg [3:0] addfulladder_out;
    reg [3:0] adderhalfadder_out;
    reg [5:0] multiplier_out;

    c_addsub_0 adder1 (
  .A(a),        // input wire [2 : 0] A
  .B(b),        // input wire [2 : 0] B
  .C_IN(cin),  // input wire C_IN
  .S(addfulladder_out)        // output wire [3 : 0] S
);
    c_addsub_1 adder2 (
  .A(a),  // input wire [2 : 0] A
  .B(b),  // input wire [2 : 0] B
  .S(adderhalfadder_out)  // output wire [3 : 0] S
);
    mult_gen_0 multiplier (
  .A(a),  // input wire [2 : 0] A
  .B(b),  // input wire [2 : 0] B
  .P(multiplier_out)  // output wire [5 : 0] P
);
```