Digital Design Diploma

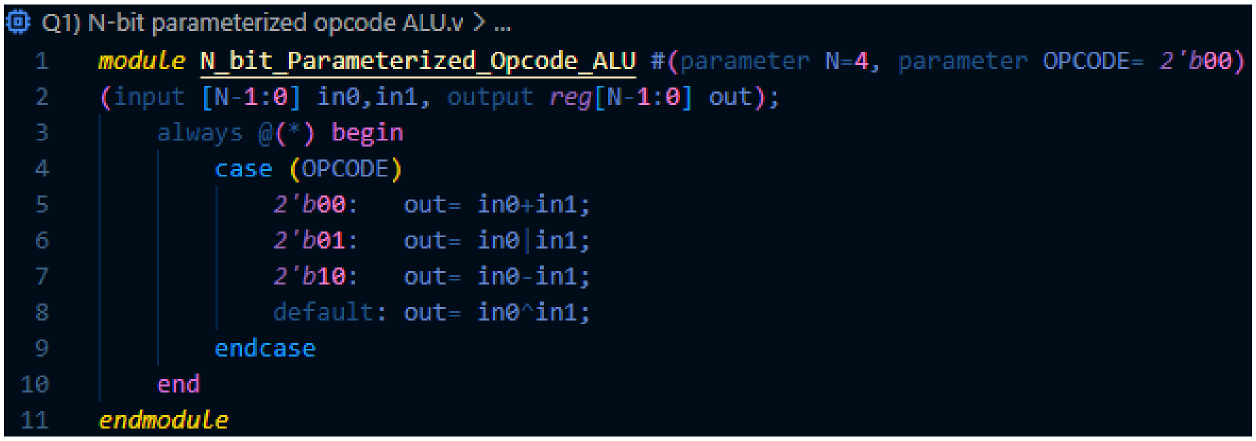
Assignment 2

Combinational & Sequential Logic Design

Name	كريم حسن عاطف علي
Group	G2

Submitted to: Eng. Kareem Waseem

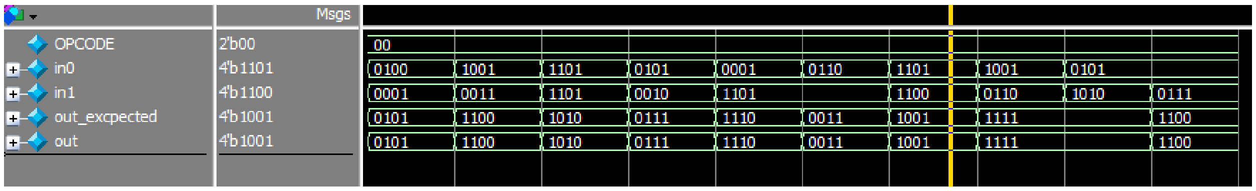
1) N-bit Parameterized Opcode ALU:



Q1) N-bit parameterized opcode ALU.v > ...

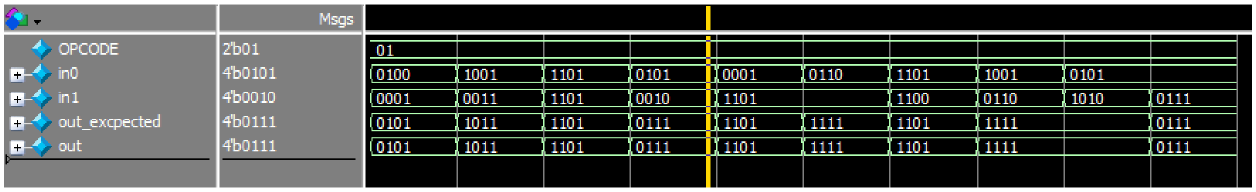
```verilog
module N_bit_Parameterized_Opcode_ALU #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, output reg[N-1:0] out);
    always @(*) begin
        case (OPCODE)
            2'b00:   out= in0+in1;
            2'b01:   out= in0|in1;
            2'b10:   out= in0-in1;
            default: out= in0^in1;
        endcase
    end
endmodule
```

Figure 1: Q1 Code



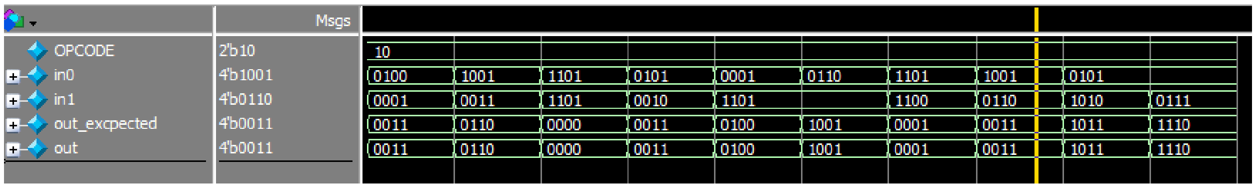
```
              Msgs
OPCODE        2'b00    00
in0           4'b1101  0100 1001 1101 0101 0001 0110 1101 1001 0101
in1           4'b1100  0001 0011 1101 0010 1101      1100 0110 1010 0111
out_excpected 4'b1001  0101 1100 1010 0111 1110 0011 1001 1111      1100
out           4'b1001  0101 1100 1010 0111 1110 0011 1001 1111      1100
```

Figure 2: Q1 Wave (Addition)



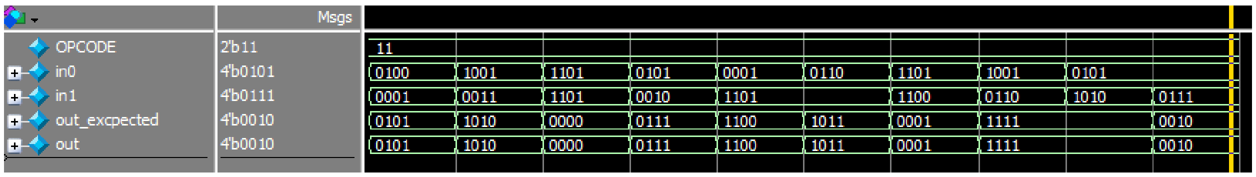
```
              Msgs
OPCODE        2'b01    01
in0           4'b0101  0100 1001 1101 0101 0001 0110 1101 1001 0101
in1           4'b0010  0001 0011 1101 0010 1101      1100 0110 1010 0111
out_excpected 4'b0111  0101 1011 1101 0111 1101 1111 1101 1111      0111
out           4'b0111  0101 1011 1101 0111 1101 1111 1101 1111      0111
```

Figure 3: Q1 Wave (OR)



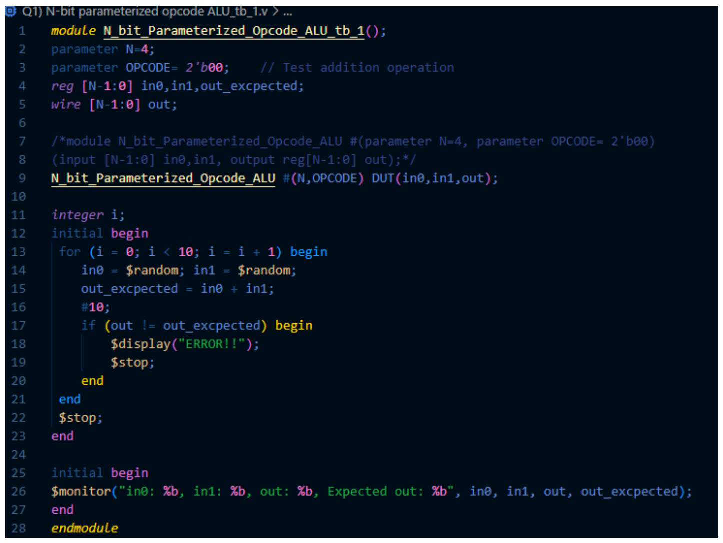
```
              Msgs
OPCODE        2'b10    10
in0           4'b1001  0100 1001 1101 0101 0001 0110 1101 1001 0101
in1           4'b0110  0001 0011 1101 0010 1101      1100 0110 1010 0111
out_excpected 4'b0011  0011 0110 0000 0011 0100 1001 0001 0011 1011 1110
out           4'b0011  0011 0110 0000 0011 0100 1001 0001 0011 1011 1110
```

Figure 4: Q1 Wave (Subtraction)



```
              Msgs
OPCODE        2'b11    11
in0           4'b0101  0100 1001 1101 0101 0001 0110 1101 1001 0101
in1           4'b0111  0001 0011 1101 0010 1101      1100 0110 1010 0111
out_excpected 4'b0010  0101 1010 0000 0111 1100 1011 0001 1111      0010
out           4'b0010  0101 1010 0000 0111 1100 1011 0001 1111      0010
```

Figure 5: Q1 Wave (XOR)



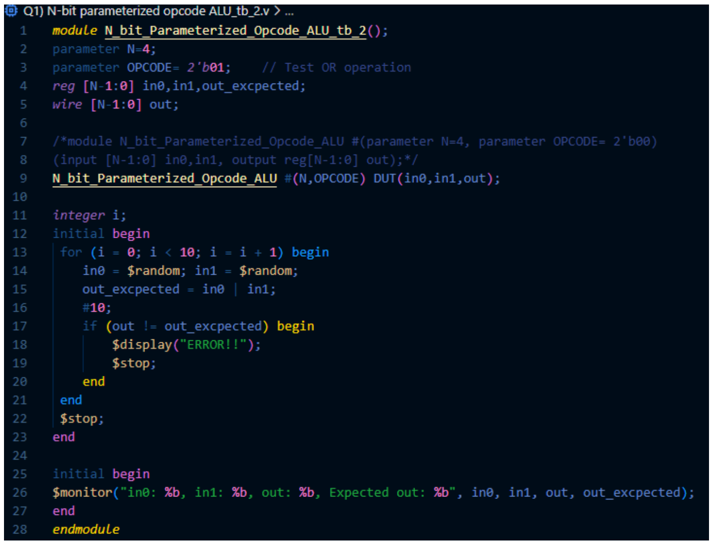
```
Q1) N-bit parameterized opcode ALU_tb_1.v > ...
module N_bit_Parameterized_Opcode_ALU_tb_1();
parameter N=4;
parameter OPCODE= 2'b00;    // Test addition operation
reg [N-1:0] in0,in1,out_excpected;
wire [N-1:0] out;

/*module N_bit_Parameterized_Opcode_ALU #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, output reg[N-1:0] out);*/
N_bit_Parameterized_Opcode_ALU #(N,OPCODE) DUT(in0,in1,out);

integer i;
initial begin
 for (i = 0; i < 10; i = i + 1) begin
    in0 = $random; in1 = $random;
    out_excpected = in0 + in1;
    #10;
    if (out != out_excpected) begin
        $display("ERROR!!");
        $stop;
    end
 end
 $stop;
end

initial begin
$monitor("in0: %b, in1: %b, out: %b, Expected out: %b", in0, in1, out, out_excpected);
end
endmodule
```

Figure 6: Q1 testbench (Addition)



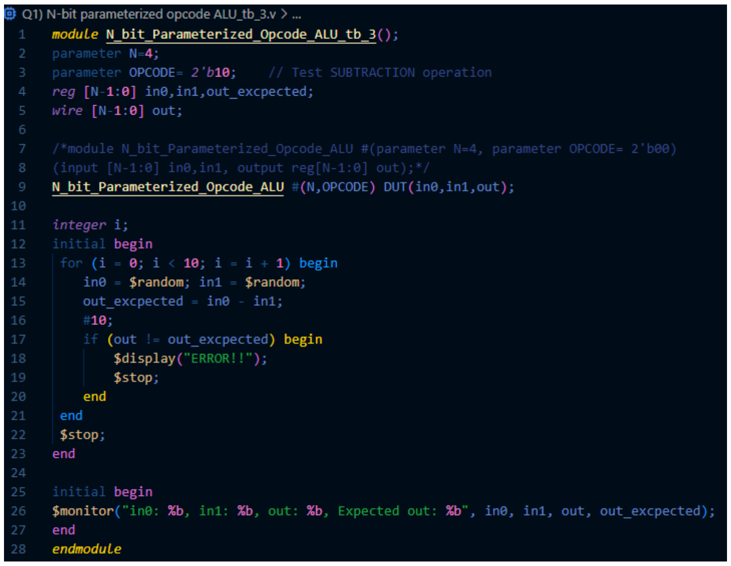
```
Q1) N-bit parameterized opcode ALU_tb_2.v > ...
module N_bit_Parameterized_Opcode_ALU_tb_2();
parameter N=4;
parameter OPCODE= 2'b01;    // Test OR operation
reg [N-1:0] in0,in1,out_excpected;
wire [N-1:0] out;

/*module N_bit_Parameterized_Opcode_ALU #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, output reg[N-1:0] out);*/
N_bit_Parameterized_Opcode_ALU #(N,OPCODE) DUT(in0,in1,out);

integer i;
initial begin
 for (i = 0; i < 10; i = i + 1) begin
    in0 = $random; in1 = $random;
    out_excpected = in0 | in1;
    #10;
    if (out != out_excpected) begin
        $display("ERROR!!");
        $stop;
    end
 end
 $stop;
end

initial begin
$monitor("in0: %b, in1: %b, out: %b, Expected out: %b", in0, in1, out, out_excpected);
end
endmodule
```

Figure 7: Q1 testbench (OR)



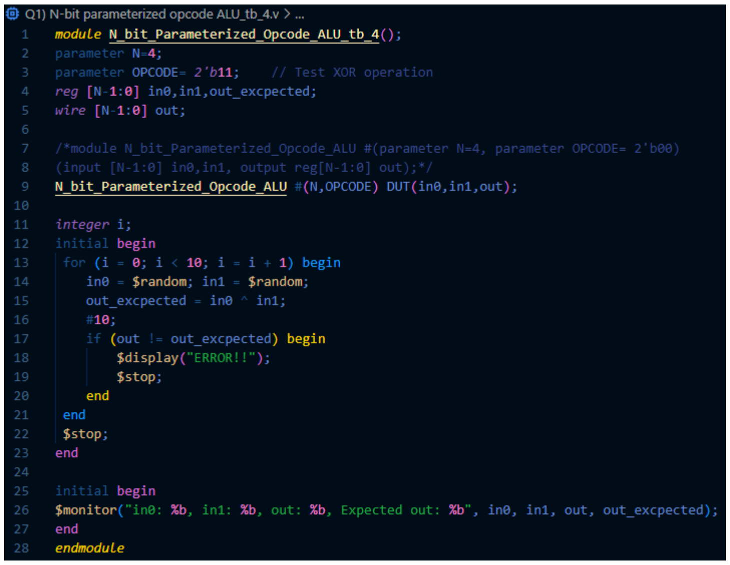
```
Q1) N-bit parameterized opcode ALU_tb_3.v > ...
module N_bit_Parameterized_Opcode_ALU_tb_3();
parameter N=4;
parameter OPCODE= 2'b10;    // Test SUBTRACTION operation
reg [N-1:0] in0,in1,out_excpected;
wire [N-1:0] out;

/*module N_bit_Parameterized_Opcode_ALU #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, output reg[N-1:0] out);*/
N_bit_Parameterized_Opcode_ALU #(N,OPCODE) DUT(in0,in1,out);

integer i;
initial begin
 for (i = 0; i < 10; i = i + 1) begin
    in0 = $random; in1 = $random;
    out_excpected = in0 - in1;
    #10;
    if (out != out_excpected) begin
        $display("ERROR!!");
        $stop;
    end
 end
 $stop;
end

initial begin
$monitor("in0: %b, in1: %b, out: %b, Expected out: %b", in0, in1, out, out_excpected);
end
endmodule
```

Figure 8: Q1 testbench (Subtraction)



```
Q1) N-bit parameterized opcode ALU_tb_4.v > ...
module N_bit_Parameterized_Opcode_ALU_tb_4();
parameter N=4;
parameter OPCODE= 2'b11;    // Test XOR operation
reg [N-1:0] in0,in1,out_excpected;
wire [N-1:0] out;

/*module N_bit_Parameterized_Opcode_ALU #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, output reg[N-1:0] out);*/
N_bit_Parameterized_Opcode_ALU #(N,OPCODE) DUT(in0,in1,out);

integer i;
initial begin
 for (i = 0; i < 10; i = i + 1) begin
    in0 = $random; in1 = $random;
    out_excpected = in0 ^ in1;
    #10;
    if (out != out_excpected) begin
        $display("ERROR!!");
        $stop;
    end
 end
 $stop;
end

initial begin
$monitor("in0: %b, in1: %b, out: %b, Expected out: %b", in0, in1, out, out_excpected);
end
endmodule
```

Figure 9: Q1 testbench (XOR)

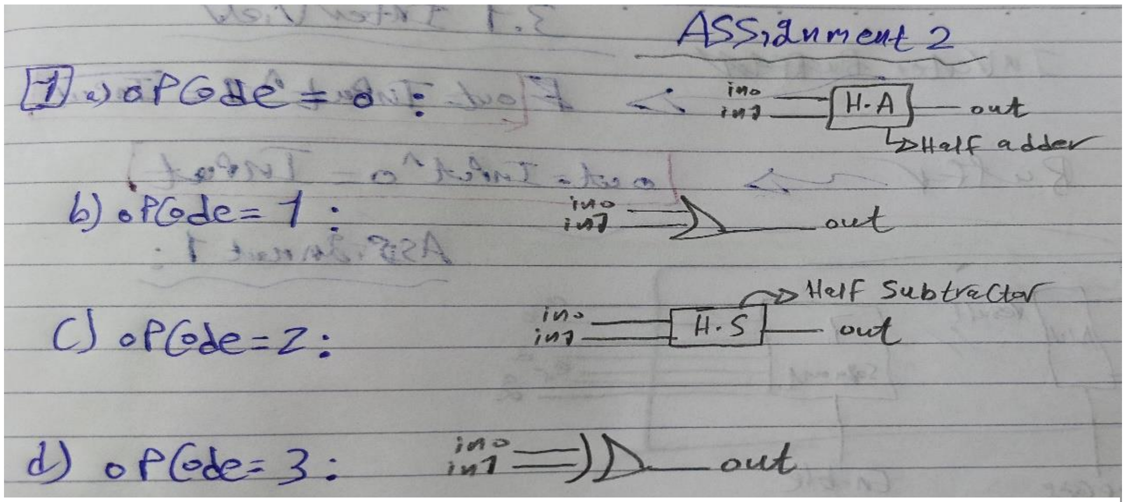
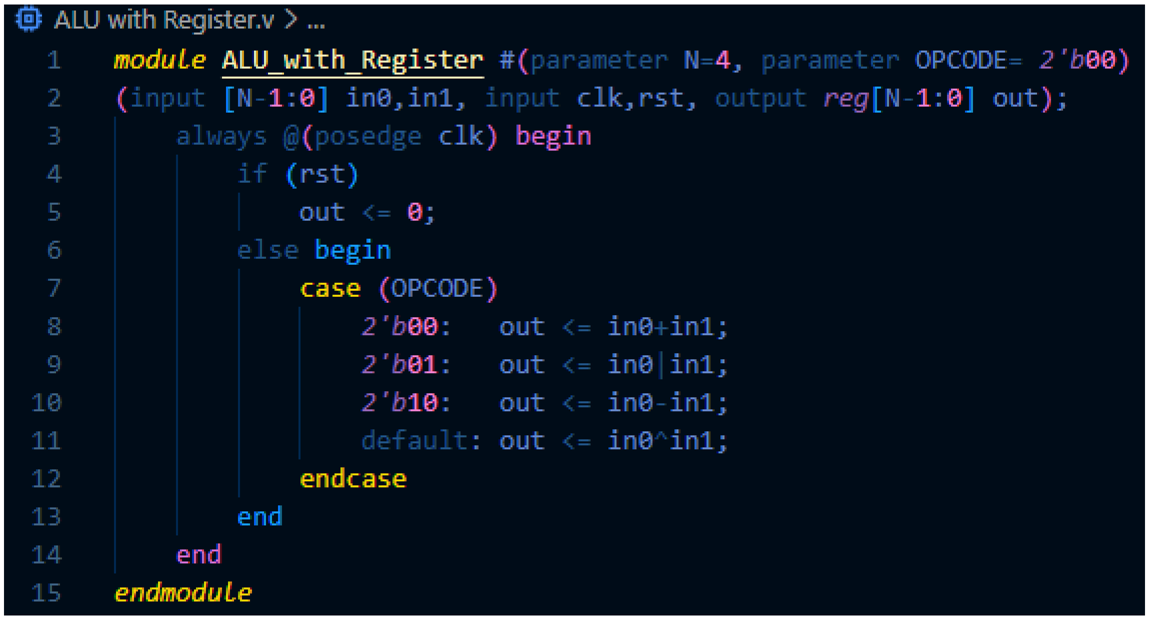


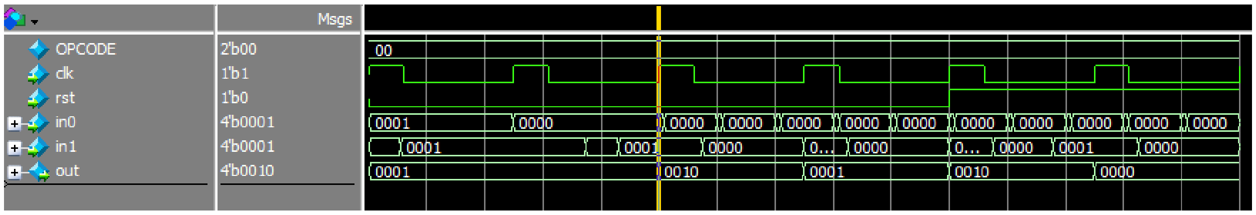
Figure 10: Q1 Synthesis

2) ALU with Register:


ALU with Register.v > ...

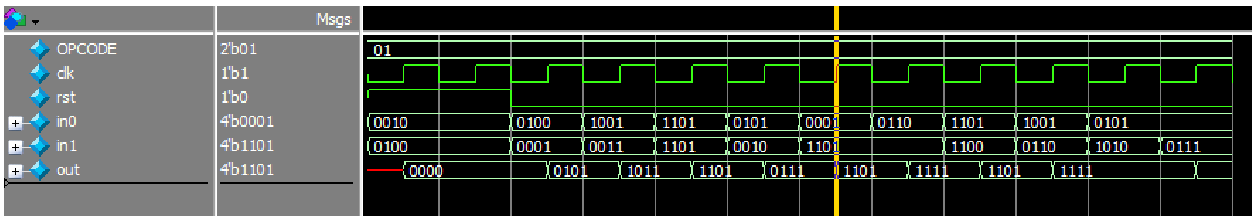
```verilog
module ALU_with_Register #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, input clk,rst, output reg[N-1:0] out);
    always @(posedge clk) begin
        if (rst)
            out <= 0;
        else begin
            case (OPCODE)
                2'b00:   out <= in0+in1;
                2'b01:   out <= in0|in1;
                2'b10:   out <= in0-in1;
                default: out <= in0^in1;
            endcase
        end
    end
endmodule
```

Figure 11: Q2 Code



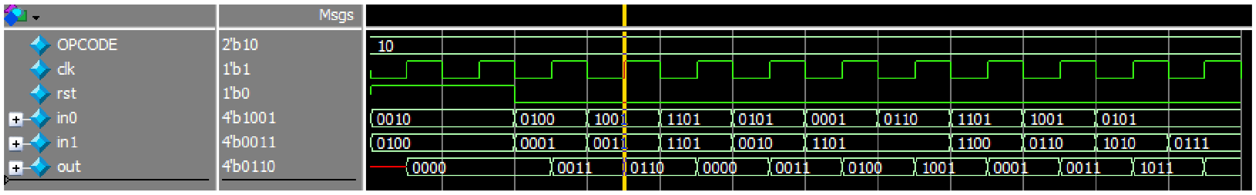
	Msgs
OPCODE	2'b00
clk	1'b1
rst	1'b0
in0	4'b0001
in1	4'b0001
out	4'b0010

Figure 12: Q2 Wave (Addition)



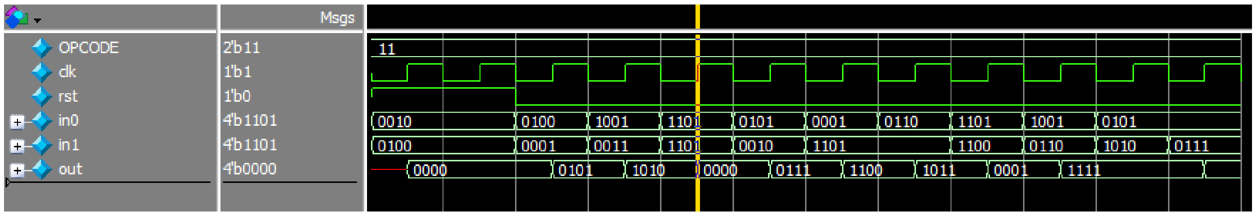
	Msgs
OPCODE	2'b01
clk	1'b1
rst	1'b0
in0	4'b0001
in1	4'b1101
out	4'b1101

Figure 13: Q2 Wave (OR)



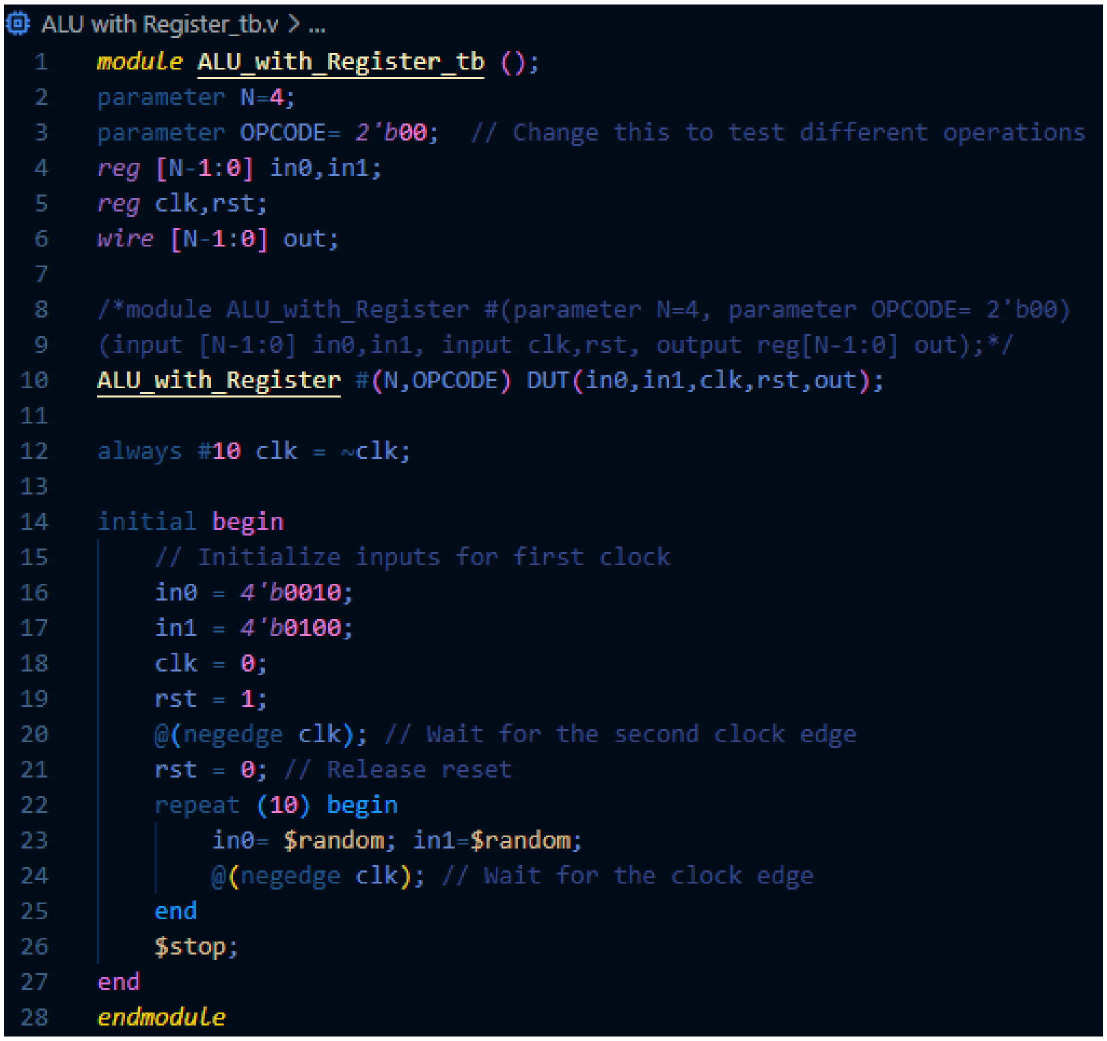
	Msgs
OPCODE	2'b10
clk	1'b1
rst	1'b0
in0	4'b1001
in1	4'b0011
out	4'b0110

Figure 14: Q2 Wave (Subtraction)



	Msgs
OPCODE	2'b11
clk	1'b1
rst	1'b0
in0	4'b1101
in1	4'b1101
out	4'b0000

Figure 15: Q2 Wave (XOR)



```verilog
module ALU_with_Register_tb ();
parameter N=4;
parameter OPCODE= 2'b00;  // Change this to test different operations
reg [N-1:0] in0,in1;
reg clk,rst;
wire [N-1:0] out;

/*module ALU_with_Register #(parameter N=4, parameter OPCODE= 2'b00)
(input [N-1:0] in0,in1, input clk,rst, output reg[N-1:0] out);*/
ALU_with_Register #(N,OPCODE) DUT(in0,in1,clk,rst,out);

always #10 clk = ~clk;

initial begin
    // Initialize inputs for first clock
    in0 = 4'b0010;
    in1 = 4'b0100;
    clk = 0;
    rst = 1;
    @(negedge clk); // Wait for the second clock edge
    rst = 0; // Release reset
    repeat (10) begin
        in0= $random; in1=$random;
        @(negedge clk); // Wait for the clock edge
    end
    $stop;
end
endmodule
```

Figure 16: Q2 Testbench

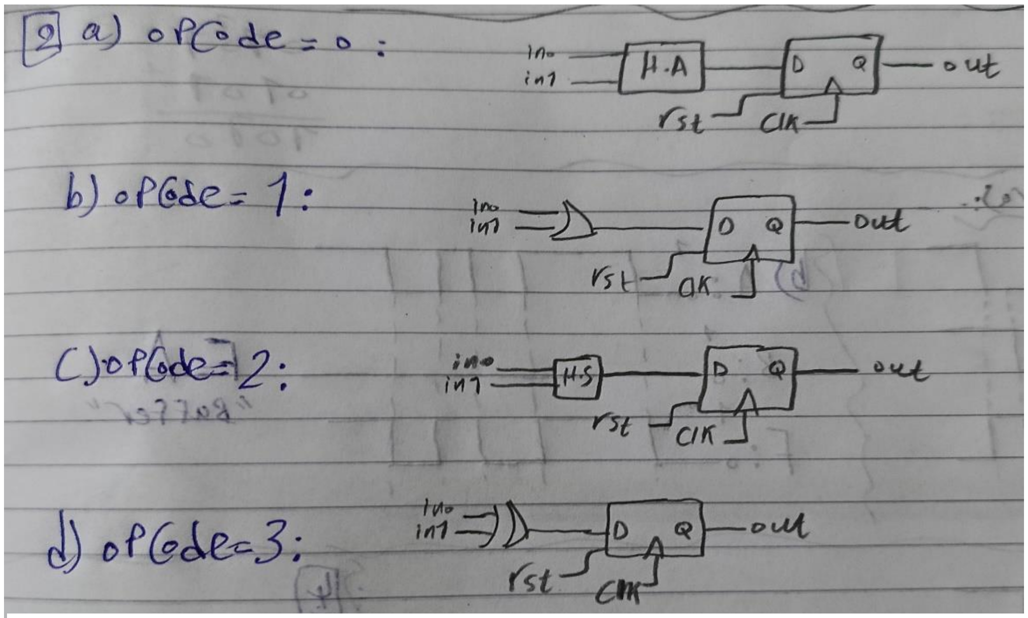
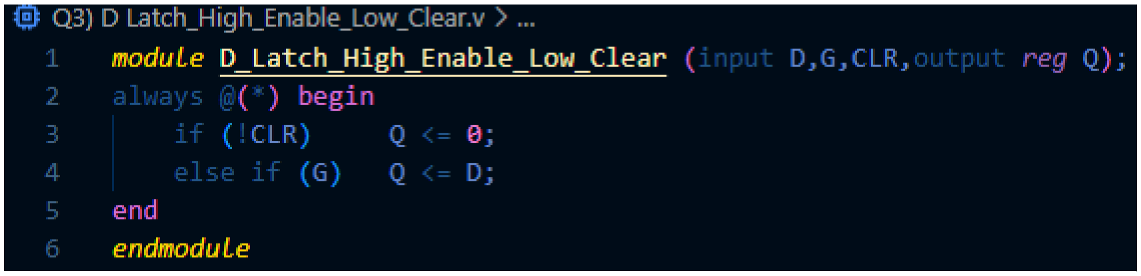


Figure 17: Q2 Synthesis

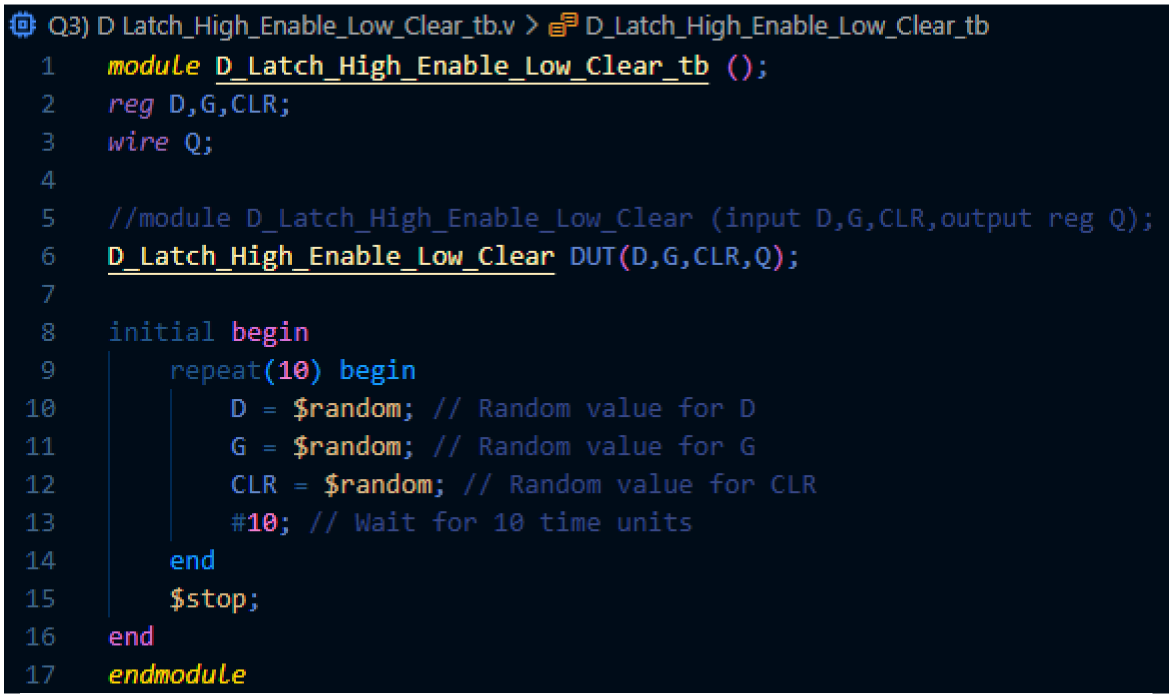
3) Data Latch with active low Clear:



Q3) D Latch_High_Enable_Low_Clear.v > ...

```verilog
module D_Latch_High_Enable_Low_Clear (input D,G,CLR,output reg Q);
always @(*) begin
    if (!CLR)     Q <= 0;
    else if (G)   Q <= D;
end
endmodule
```

Figure 18: Q3 Code



Q3) D Latch_High_Enable_Low_Clear_tb.v > D_Latch_High_Enable_Low_Clear_tb

```verilog
module D_Latch_High_Enable_Low_Clear_tb ();
reg D,G,CLR;
wire Q;

//module D_Latch_High_Enable_Low_Clear (input D,G,CLR,output reg Q);
D_Latch_High_Enable_Low_Clear DUT(D,G,CLR,Q);

initial begin
    repeat(10) begin
        D = $random; // Random value for D
        G = $random; // Random value for G
        CLR = $random; // Random value for CLR
        #10; // Wait for 10 time units
    end
    $stop;
end
endmodule
```

Figure 19: Q3 Testbench

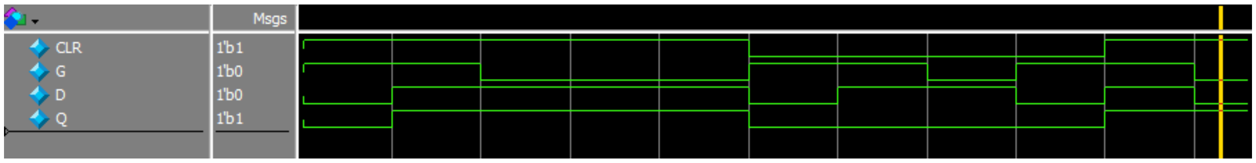
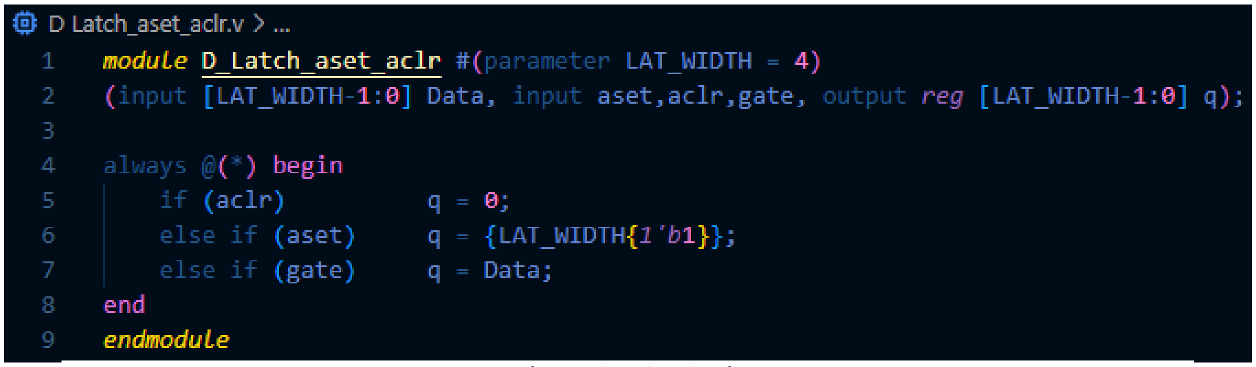


Figure 20: Q3 Wave

4) D Latch with Asynchronous set and clr:

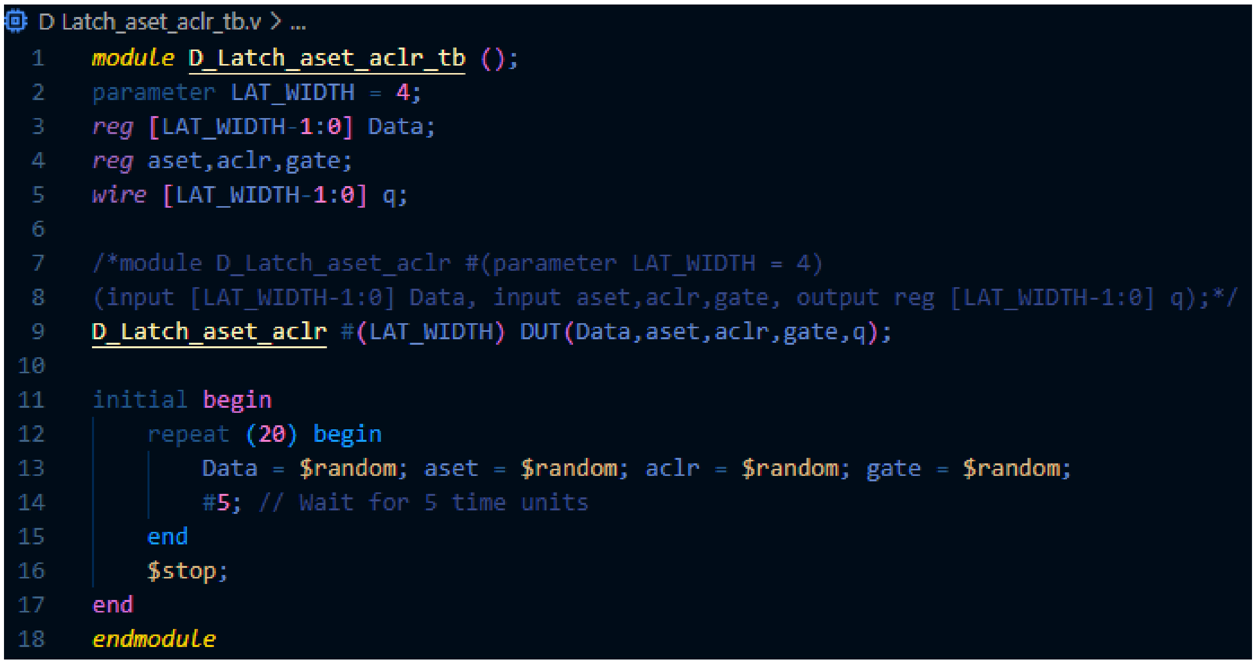


D Latch_aset_aclr.v > ...

```verilog
module D_Latch_aset_aclr #(parameter LAT_WIDTH = 4)
(input [LAT_WIDTH-1:0] Data, input aset,aclr,gate, output reg [LAT_WIDTH-1:0] q);

always @(*) begin
    if (aclr)           q = 0;
    else if (aset)      q = {LAT_WIDTH{1'b1}};
    else if (gate)      q = Data;
end
endmodule
```

Figure 21: Q4 Code



D Latch_aset_aclr_tb.v > ...

```verilog
module D_Latch_aset_aclr_tb ();
parameter LAT_WIDTH = 4;
reg [LAT_WIDTH-1:0] Data;
reg aset,aclr,gate;
wire [LAT_WIDTH-1:0] q;

/*module D_Latch_aset_aclr #(parameter LAT_WIDTH = 4)
(input [LAT_WIDTH-1:0] Data, input aset,aclr,gate, output reg [LAT_WIDTH-1:0] q);*/
D_Latch_aset_aclr #(LAT_WIDTH) DUT(Data,aset,aclr,gate,q);

initial begin
    repeat (20) begin
        Data = $random; aset = $random; aclr = $random; gate = $random;
        #5; // Wait for 5 time units
    end
    $stop;
end
endmodule
```

Figure 22: Q4 Testbench

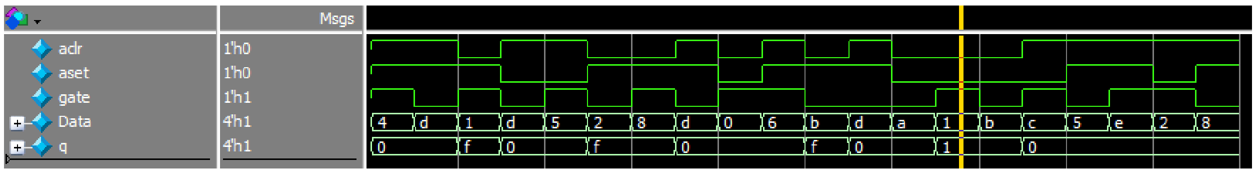
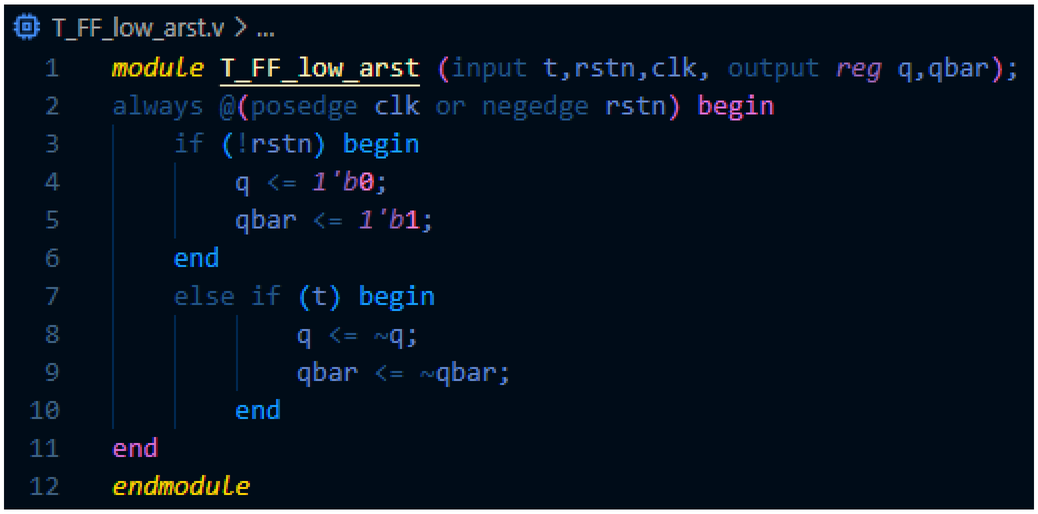


Figure 23: Q4 Wave

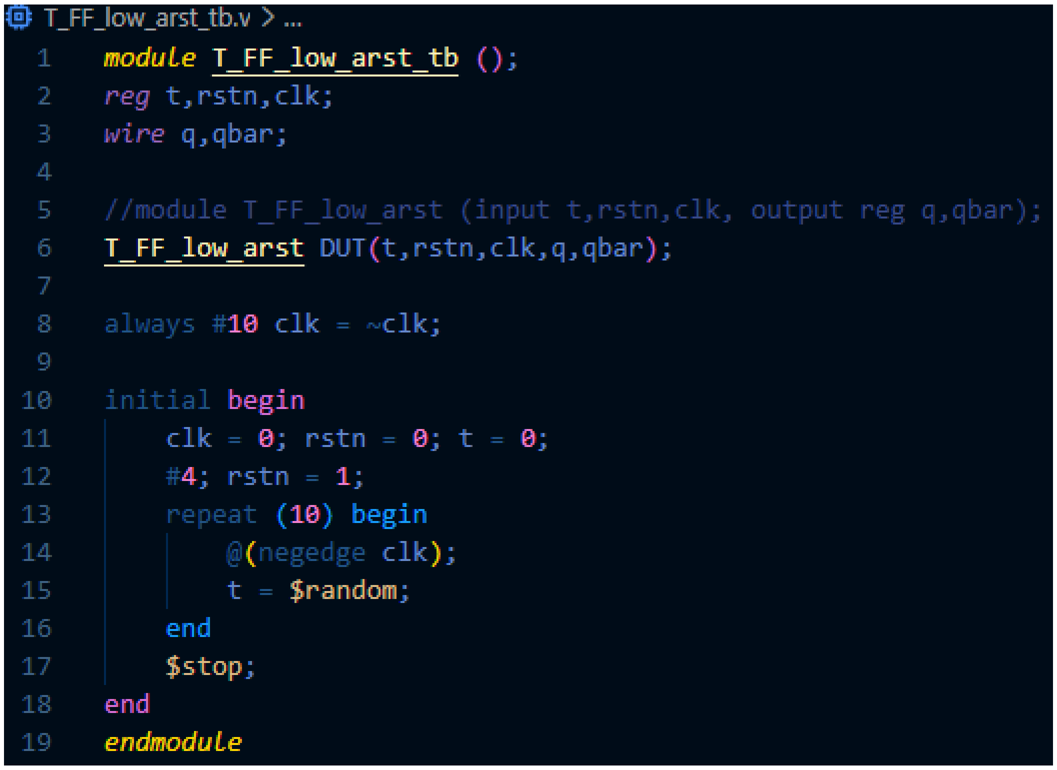
5)

a) T FF with Asynchronous Active Low rst:



```
T_FF_low_arst.v > ...
module T_FF_low_arst (input t,rstn,clk, output reg q,qbar);
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        q <= 1'b0;
        qbar <= 1'b1;
    end
    else if (t) begin
            q <= ~q;
            qbar <= ~qbar;
        end
end
endmodule
```

Figure 24: Q5a Code



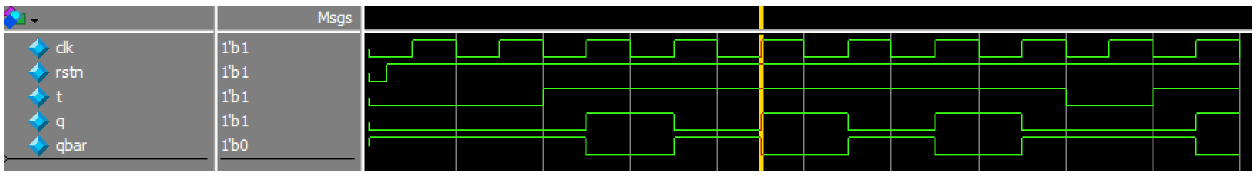
```
T_FF_low_arst_tb.v > ...
module T_FF_low_arst_tb ();
reg t,rstn,clk;
wire q,qbar;

//module T_FF_low_arst (input t,rstn,clk, output reg q,qbar);
T_FF_low_arst DUT(t,rstn,clk,q,qbar);

always #10 clk = ~clk;

initial begin
    clk = 0; rstn = 0; t = 0;
    #4; rstn = 1;
    repeat (10) begin
        @(negedge clk);
        t = $random;
    end
    $stop;
end
endmodule
```

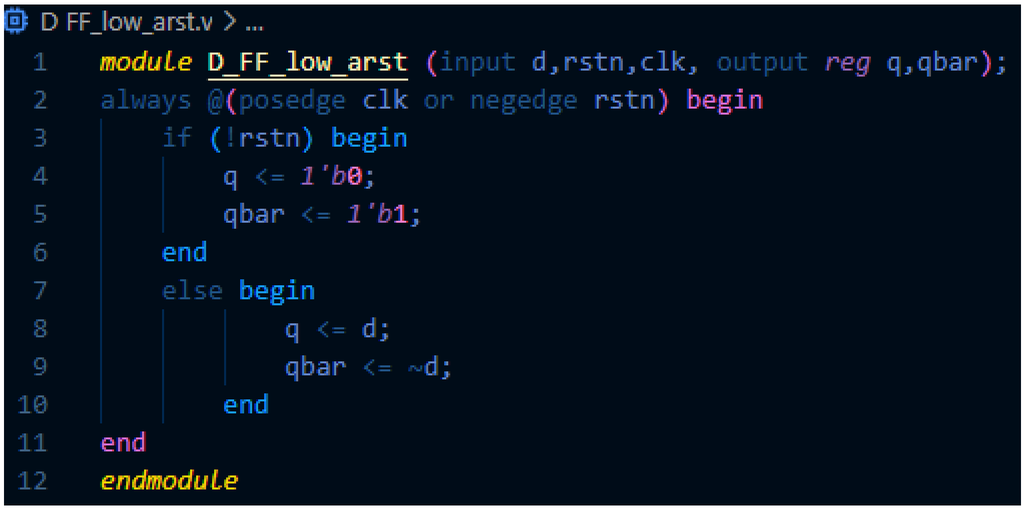
Figure 25: Q5a Testbench



```
Msgs
clk   1'b1
rstn  1'b1
t     1'b1
q     1'b1
qbar  1'b0
```

Figure 26: Q5a Wave

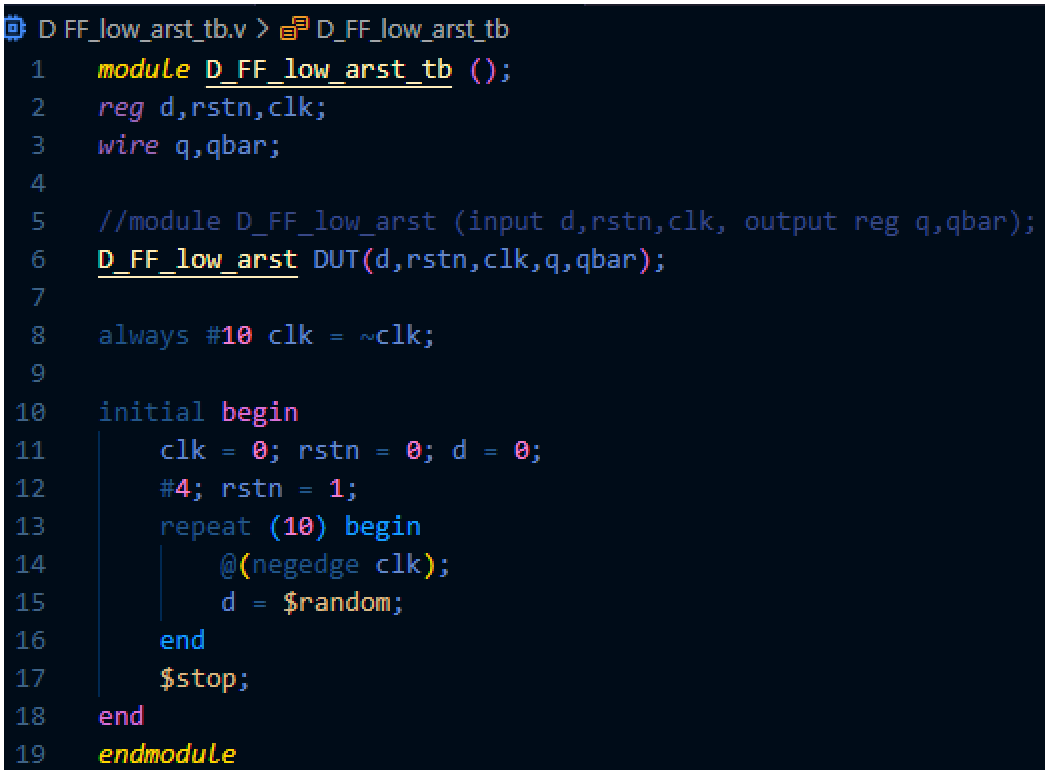
b) D FF_low_arst:



D FF_low_arst.v > ...

```verilog
module D_FF_low_arst (input d,rstn,clk, output reg q,qbar);
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        q <= 1'b0;
        qbar <= 1'b1;
    end
    else begin
            q <= d;
            qbar <= ~d;
        end
end
endmodule
```

Figure 27: Q5b Code



D FF_low_arst_tb.v > D_FF_low_arst_tb

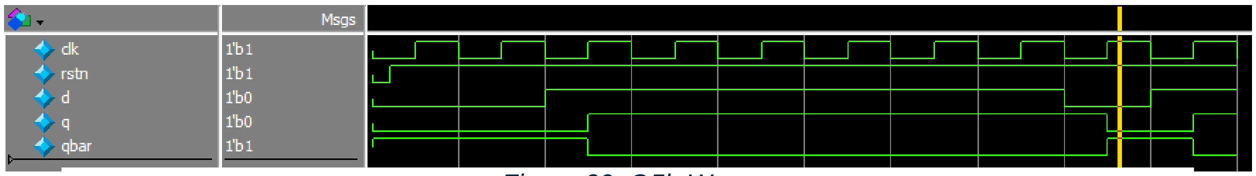
```verilog
module D_FF_low_arst_tb ();
reg d,rstn,clk;
wire q,qbar;

//module D_FF_low_arst (input d,rstn,clk, output reg q,qbar);
D_FF_low_arst DUT(d,rstn,clk,q,qbar);

always #10 clk = ~clk;

initial begin
    clk = 0; rstn = 0; d = 0;
    #4; rstn = 1;
    repeat (10) begin
        @(negedge clk);
        d = $random;
    end
    $stop;
end
endmodule
```

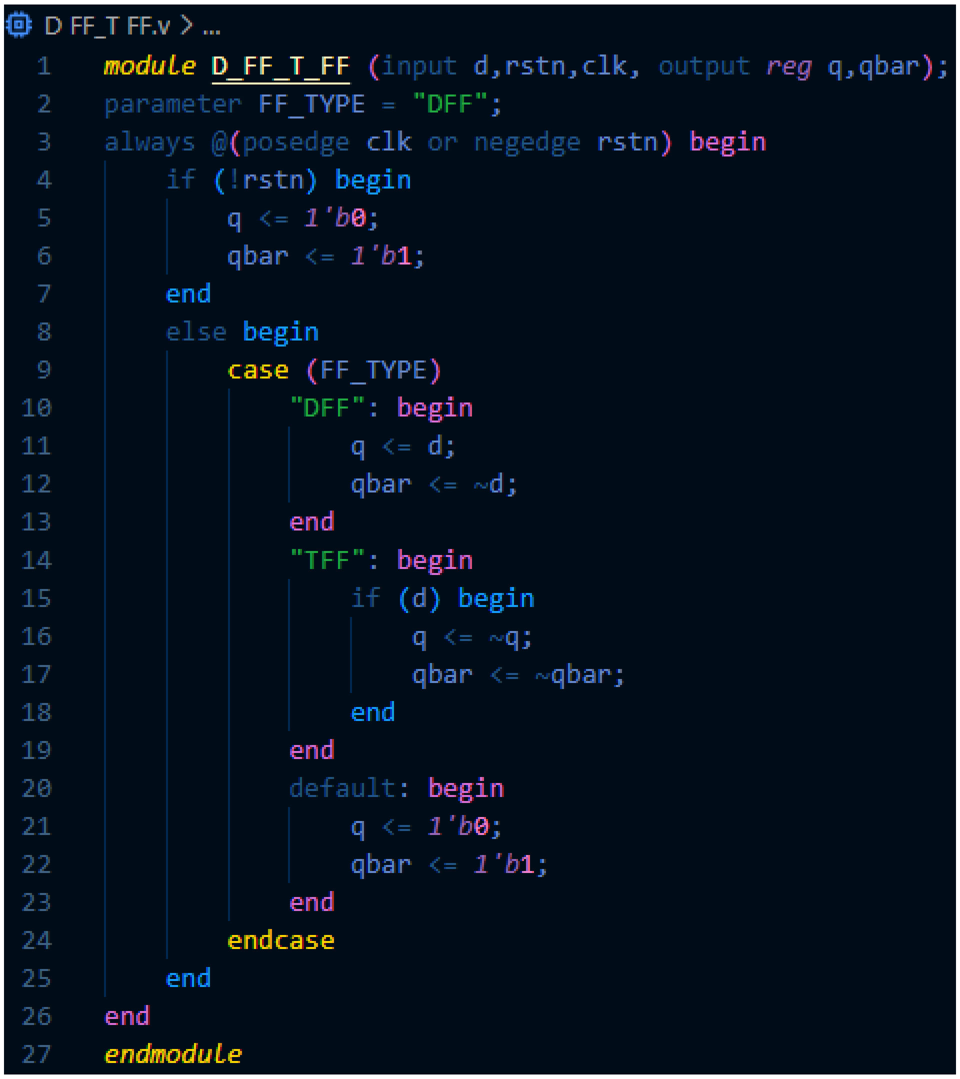
Figure 28: Q5b Testbench



	Msgs
clk	1'b1
rstn	1'b1
d	1'b0
q	1'b0
qbar	1'b1

Figure 29: Q5b Wave

c) Parameterized Asynchronous FF with Active low rst (D FF & T FF):



D FF_T FF.v > ...

```verilog
module D_FF_T_FF (input d,rstn,clk, output reg q,qbar);
parameter FF_TYPE = "DFF";
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        q <= 1'b0;
        qbar <= 1'b1;
    end
    else begin
        case (FF_TYPE)
            "DFF": begin
                q <= d;
                qbar <= ~d;
            end
            "TFF": begin
                if (d) begin
                    q <= ~q;
                    qbar <= ~qbar;
                end
            end
            default: begin
                q <= 1'b0;
                qbar <= 1'b1;
            end
        endcase
    end
end
endmodule
```

Figure 30: Q5c Code

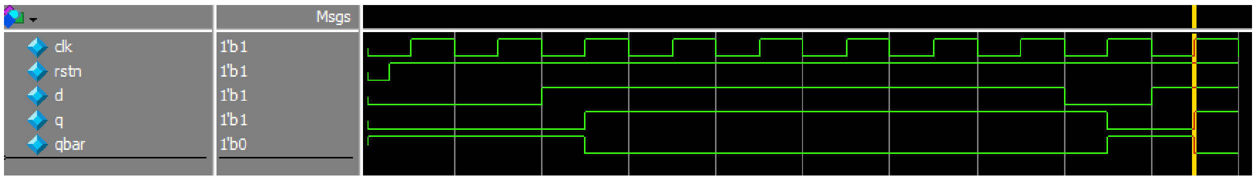


Figure 31: Q5c Wave (DFF)

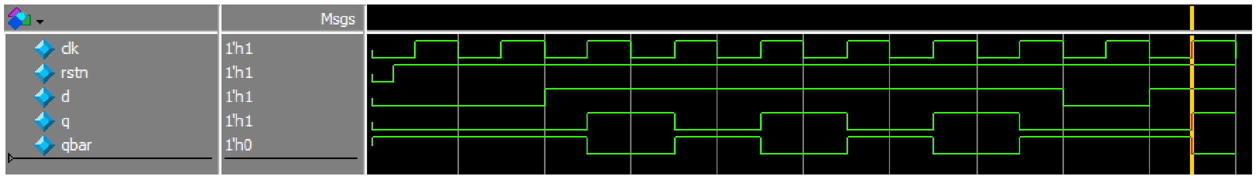
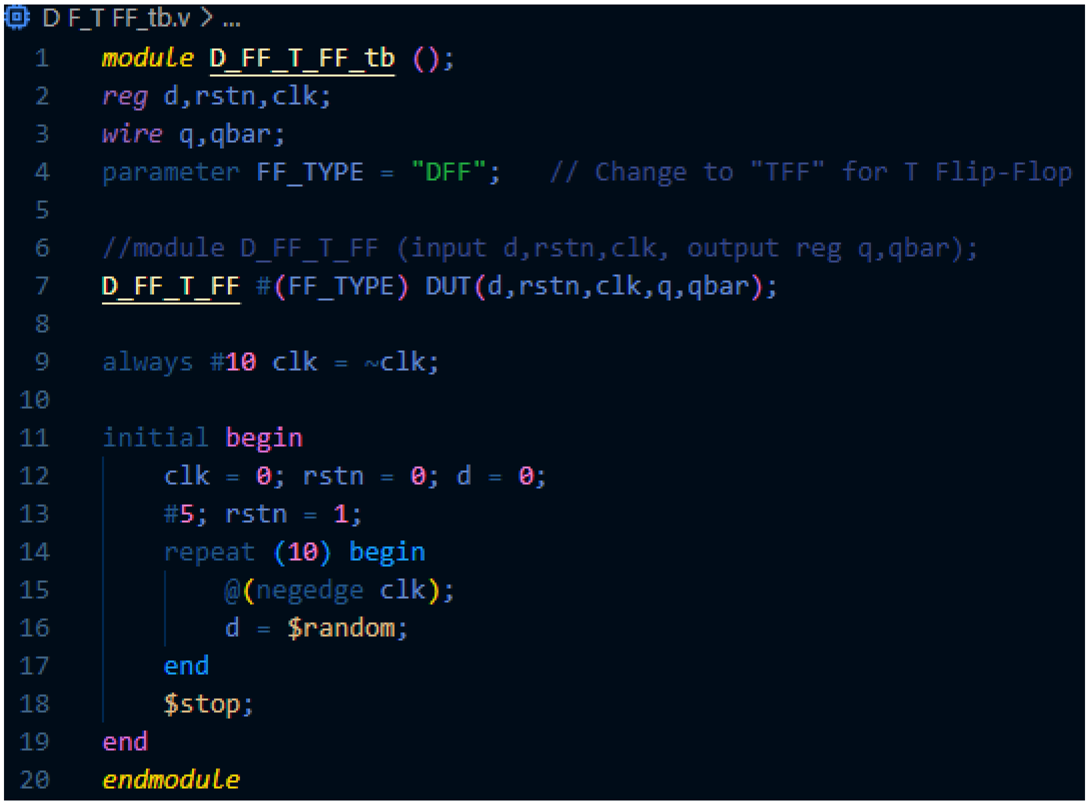


Figure 32: Q5c Wave (TFF)



D F_T FF_tb.v > ...

```verilog
module D_FF_T_FF_tb ();
reg d,rstn,clk;
wire q,qbar;
parameter FF_TYPE = "DFF";    // Change to "TFF" for T Flip-Flop

//module D_FF_T_FF (input d,rstn,clk, output reg q,qbar);
D_FF_T_FF #(FF_TYPE) DUT(d,rstn,clk,q,qbar);

always #10 clk = ~clk;

initial begin
    clk = 0; rstn = 0; d = 0;
    #5; rstn = 1;
    repeat (10) begin
        @(negedge clk);
        d = $random;
    end
    $stop;
end
endmodule
```

Figure 33: Q5c Testbench