1- Implement the Boolean function below using transistor-level Verilog coding.
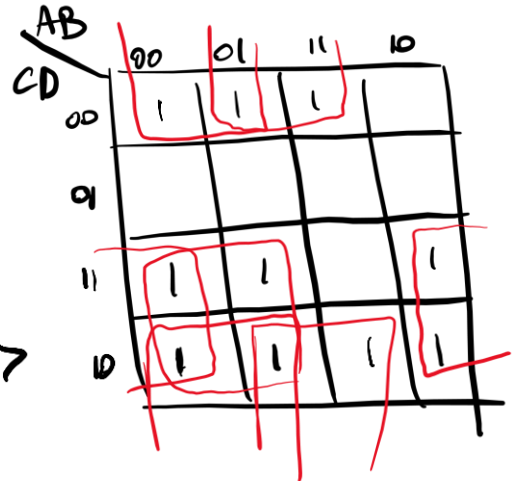$$f(A,B,C,D) = A'C + AB'C + BD' + A'C'D'$$
In your testbench, test the module for <u>all</u> different combinations of inputs.

For the first question, we simplify our function using Karnaugh map, then we use transistors, ~f and dual of ~f to create f.

① $f = A'C + AB'C + BD' + A'C'D'$

K-Map $\Rightarrow$

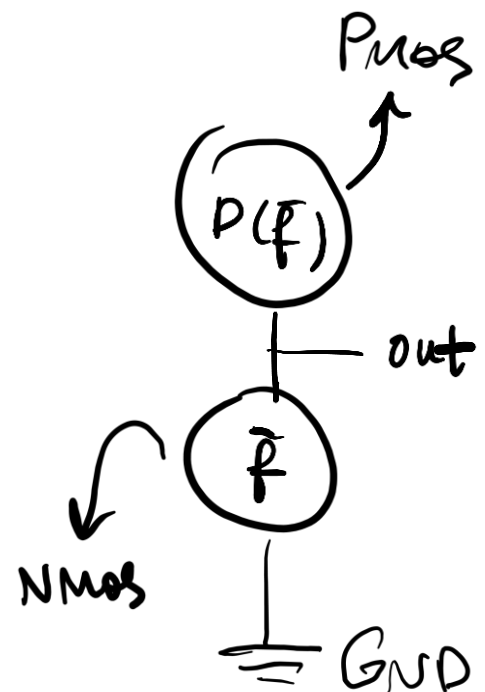$f = A'D' + BD' + A'C + B'C$

$\downarrow f'$

$f' = (A+D) \cdot (B'+D) \cdot (A+C') \cdot (B+C')$

$\downarrow$

$D(f') = (AD) + (B'D) + (AC') + (BC')$

PMOS

$P(f)$

out

$\tilde{f}$

NMOS

GND

V_{dd}

B'

A

B

A W_1 W_2 W_3 W_4

D D C' C'

out

A W_5 D

B' W_6 D

A W_7 C'

B C'

GND
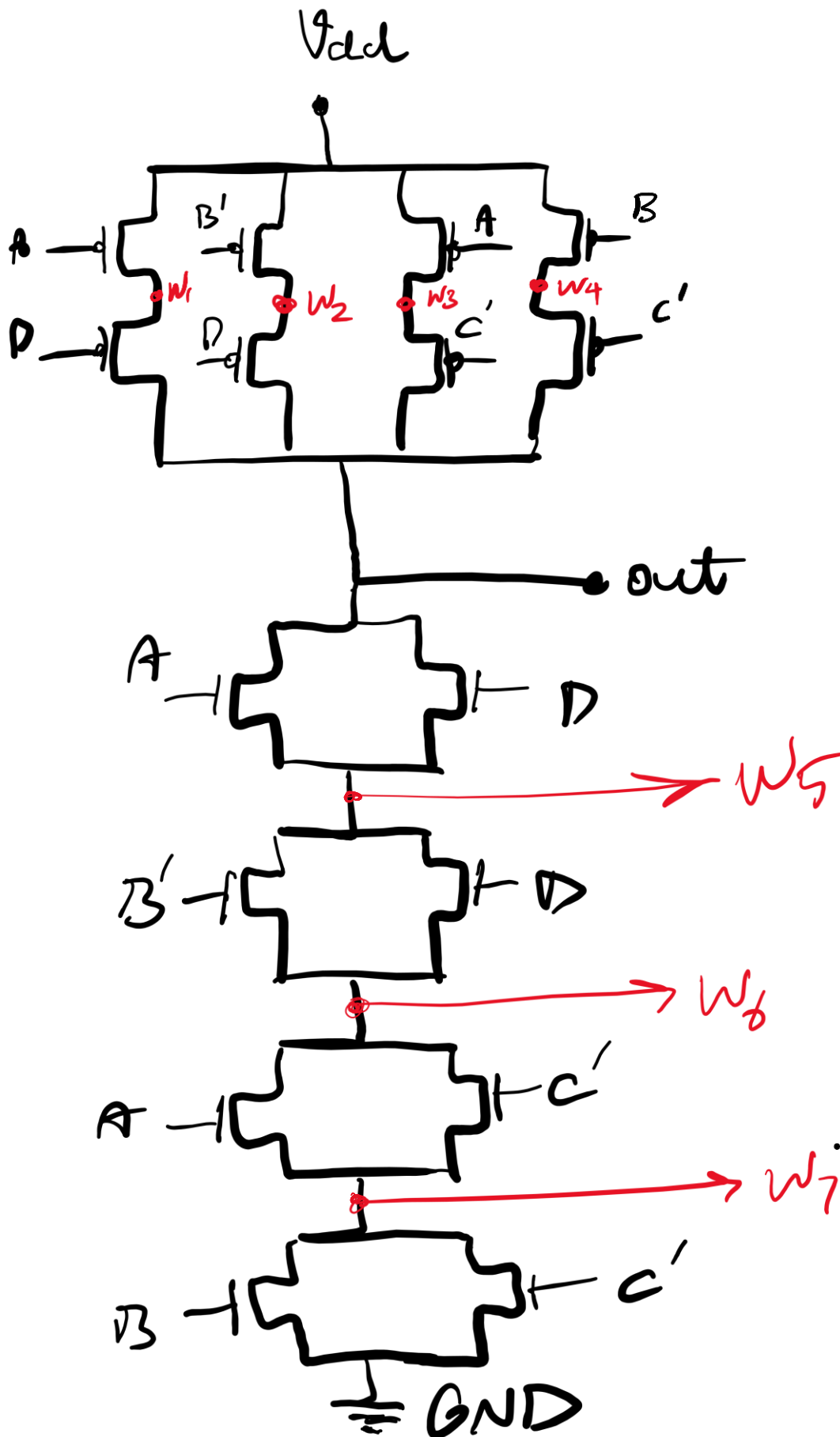
The Verilog code, implementation of module:

```verilog
problem1 > V Foroutan.Nazanin.402243084.problem1.module.v
 1    module fun(input a, a_not, b, b_not, c, c_not, d, d_not, output out);
 2
 3        wire w1, w2, w3, w4, w5, w6, w7;
 4        supply1 vdd;
 5        supply0 gnd;
 6
 7        pmos(w1, vdd, a);
 8        pmos(out, w1, d);
 9        pmos(w2, vdd, b_not);
10        pmos(out, w2, d);
11        pmos(w3, vdd, a);
12        pmos(out, w3, c_not);
13        pmos(w4, vdd, b);
14        pmos(out, w4, c_not);
15
16        nmos(out, w5, a);
17        nmos(out, w5, d);
18        nmos(w5, w6, b_not);
19        nmos(w5, w6, d);
20        nmos(w6, w7, a);
21        nmos(w6, w7, c_not);
22        nmos(w7, gnd, b);
23        nmos(w7, gnd, c_not);
24
25    endmodule
```
,

Part of test bench bench implementation:

```verilog
problem1 > V Foroutan.Nazanin.402243084.problem1.testbench.v
 1    `include "Foroutan.Nazanin.402243084.problem1.module.v"
 2
 3    module problem1_TestBench();
 4
 5        reg a, not_a, b, not_b, c, not_c, d, not_d;
 6        wire out;
 7        fun fun1(a, not_a, b, not_b, c, not_c, d, not_d, out);
 8
 9        initial begin
10
11            $dumpfile("waveform_problem1.vcd");
12            $dumpvars(0, problem1_TestBench);
13
14            #1;
15            a = 0;
16            b = 0;
17            c = 0;
18            d = 0;
19            not_a = ~a;
20            not_b = ~b;
21            not_d = ~d;
22            not_c = ~c;
23            $display("A:%d , B:%d , C:%d , D:%d --> %d",a,b,c,d, out);
24
```

DLC Project Report Nazanin Foroutan

Picture of the outputs in the terminal:

```
[Running] Foroutan.Nazanin.402243084.problem1.testbench.v
VCD info: dumpfile waveform_problem1.vcd opened for output.
A:0 , B:0 , C:0 , D:0 --> 1
A:0 , B:0 , C:0 , D:1 --> 0
A:0 , B:0 , C:1 , D:0 --> 1
A:0 , B:0 , C:1 , D:1 --> 1
A:0 , B:1 , C:0 , D:0 --> 1
A:0 , B:1 , C:0 , D:1 --> 0
A:0 , B:1 , C:1 , D:0 --> 1
A:0 , B:1 , C:1 , D:1 --> 1
A:1 , B:0 , C:0 , D:0 --> 0
A:1 , B:0 , C:0 , D:1 --> 0
A:1 , B:0 , C:1 , D:0 --> 1
A:1 , B:0 , C:1 , D:1 --> 1
A:1 , B:1 , C:0 , D:0 --> 1
A:1 , B:1 , C:0 , D:1 --> 0
A:1 , B:1 , C:1 , D:0 --> 1
A:1 , B:1 , C:1 , D:1 --> 0
[Done] exit with code=0 in 0.024 seconds
```

The picture of the waves:
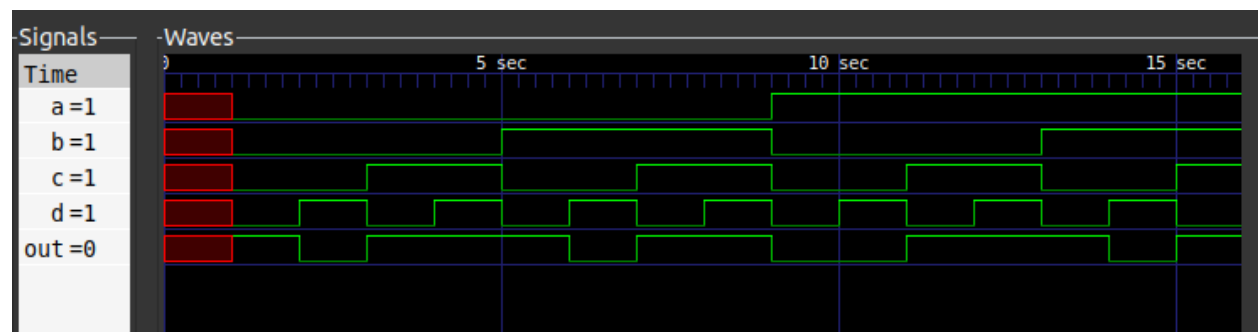
2- Implement an 4-bit signed adder using gate-level Verilog coding. The adder should have an overflow detector.
In your testbench, test it for different inputs (once with two positive numbers without overflow, once with two positive numbers with overflow, once with two negative numbers without overflow, and once with two negative numbers with overflow).

$$S_0 = a.\bar{b} + \bar{a}.b$$

$$C_0 = a.b_0$$
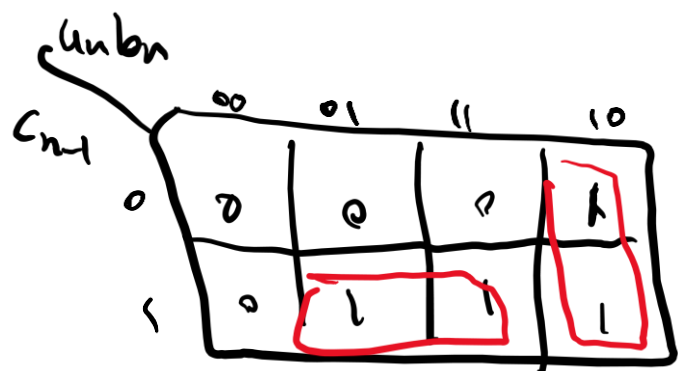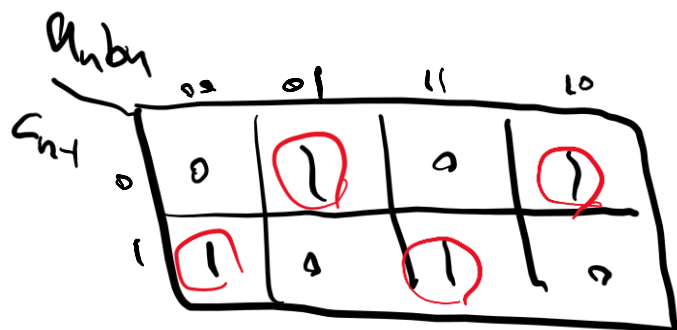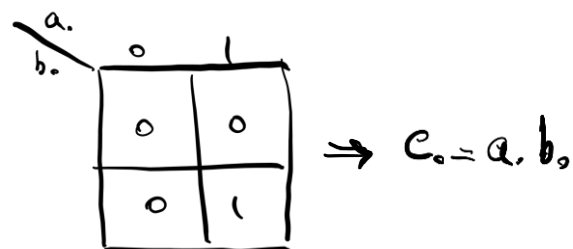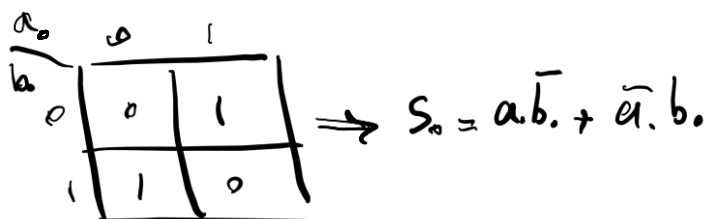
$a_n b_n C_{n-1}$			S_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S_n = \overline{a_n}\left(b_n \oplus C_{n-1}\right) + a_n\left(\overline{b_n} \oplus C_{n-1}\right)$$

$$C_n = a_n \overline{b_n} + b_n C_{n-1}$$

$a_3 b_3$

S_3

	00	01	11	10
0	0	0	1	0
1	1	0	0	0

overflow $\Rightarrow a_3 b_3 \overline{s_3} + \overline{a_3} \overline{b_3} S_3$

For the module implementation I built one full adder then I used cascade to build the four bit adder.

one bit adder:

```verilog
oblem2 > V  Foroutan.Nazanin.402243084.problem2.module.v
1   module full_adder(input A, B, cin, output sum, cout);
2       wire AandB, AandCin, BandCin, AxorB;
3
4       xor(AxorB, A, B);
5       xor(sum, AxorB, cin);
6
7       and(AandB, A, B);
8       and(AandCin, A, cin);
9       and(BandCin, B, cin);
10      or(cout, AandB, AandCin, BandCin);
11
12  endmodule
13
```

DLC Project Report Nazanin Foroutan

Four bit adder:

```verilog
14
15   module four_bit_adder(input [3:0] A, B, output [3:0] sum, output overflow, cout);
16       wire c1, c2, c3; // Declare cout wire
17       wire A3andB3andS3not, A3notandB3notandS3;
18
19       full_adder fa1(A[0], B[0], 1'b0, sum[0], c1);
20       full_adder fa2(A[1], B[1], c1, sum[1], c2);
21       full_adder fa3(A[2], B[2], c2, sum[2], c3);
22       full_adder fa4(A[3], B[3], c3, sum[3], cout);
23
24       // overflow detection
25       and(A3andB3andS3not, A[3], B[3], ~sum[3]);
26       and(A3notandB3notandS3, ~A[3], ~B[3], sum[3]);
27       or(overflow, A3andB3andS3not, A3notandB3notandS3);
28
29   endmodule
```

Part of test bench:

```verilog
1    `include "Foroutan.Nazanin.402243084.problem2.module.v"
2
3    module problem2_testbench;
4        reg [3:0] A;
5        reg [3:0] B;
6        wire [3:0] Sum;
7        wire overflow;
8        wire cout;
9
10       // Instantiate the four_bit_adder module using positional port mapping
11       four_bit_adder adder(A, B, Sum, overflow, cout);
12
13       initial begin
14           $dumpfile("waveform_problem2.vcd");
15           $dumpvars(0, problem2_testbench);
16
17           // Test case 1: Two positive numbers without overflow
18           A = 4'b0010; // 2
19           B = 4'b0011; // 3
20           #10;
21           $display("A = %b, B = %b, Sum = %b, Overflow = %b, Cout = %b", A, B, Sum, overflow, cout);
22
```
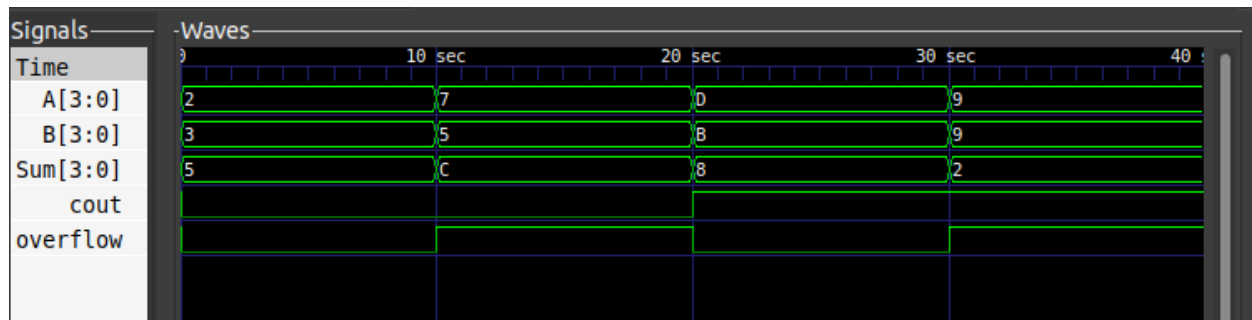
The output shown in terminal:

```
[Running] Foroutan.Nazanin.402243084.problem2.testbench.v
VCD info: dumpfile waveform_problem2.vcd opened for output.
A = 0010, B = 0011, Sum = 0101, Overflow = 0, Cout = 0
A = 0111, B = 0101, Sum = 1100, Overflow = 1, Cout = 0
A = 1101, B = 1011, Sum = 1000, Overflow = 0, Cout = 1
A = 1001, B = 1001, Sum = 0010, Overflow = 1, Cout = 1
[Done] exit with code=0 in 0.024 seconds
```

The wave shown in gtk wave:

3- Implement an ALU with two 6 bit signed inputs A and B and with 4 different operation modes mentioned below.

$$0\text{- } (A <<< 2) + (B >>> 1)$$
$$1\text{- } A + 3B$$
$$2\text{- } -B$$
$$3\text{- } |2A - B|$$

For the ALU implementation we were supposed to build modules for each operation and then use them to make the alu.

```
21  module shift(input [5:0] A, B, output [5:0] res, output cout);
22      wire [5:0] shifted_res;
23      assign shifted_res = (A <<< 2) + (B >>> 1);
24      assign res = shifted_res;
25      assign cout = (shifted_res[5] != res[5]); // Simple carry o
26  endmodule
27
28
29  module add(input [5:0] A, B, output [5:0] res, output cout);
30      assign {cout, res} = A + 3*B;
31  endmodule
32
33
34  module neg(input [5:0] B, output [5:0] res);
35      assign res = -B;
36  endmodule
37
38
39  module abs(input [5:0] A, B, output [5:0] res);
40      assign res = (2*A - B > 0 ? 2*A - B : -(2*A - B));
41  endmodule
42
```

The ALU using modules:

```verilog
module ALU(
    input [5:0] A, B,
    input [1:0] op_code,
    output [5:0] out,
    output cout
);
    wire [5:0] res0, res1, res2, res3;
    wire cout_add, cout_shift;

    shift zero(A, B, res0, cout_shift);
    add one(A, B, res1, cout_add);
    neg two(B, res2);
    abs three(A, B, res3);

    assign out = op_code[1] ? (op_code[0] ? res3 : res2) : (op_code[0] ? res1 : res0);
    assign cout = (op_code == 2'b01) ? cout_add : (op_code == 2'b00) ? cout_shift : 1'b0; // cout for add and shift operations

endmodule
```
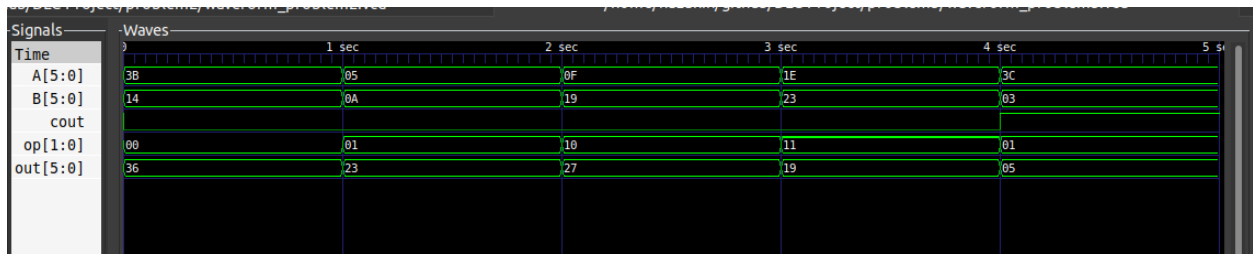
Part of test bench:

```verilog
`include "Foroutan.Nazanin.402243084.problem3.module.v"

module problem3_testbench();
    reg[5:0] A, B;
    reg[1:0] op;
    wire[5:0] out;
    wire cout;

    ALU ALU_6bit (A, B, op, out, cout);

    initial begin
        $dumpfile("waveform_problem3.vcd");
        $dumpvars(0, problem3_testbench);

        A = -5;
        B = 20;
        op = 2'b00;
        #1;
        $display("A: %b\nB: %b\nOperation: %b\nOutput: %b\nCarry Out: %b\n", A, B, op, out, cout);

        A = 5;
        B = 10;
        op = 2'b01;
        #1;
        $display("A: %b\nB: %b\nOperation: %b\nOutput: %b\nCarry Out: %b\n", A, B, op, out, cout);
```

DLC Project Report Nazanin Foroutan

The waves:

4- Design and implement a counter that only count odd numbers from 1 to 9.
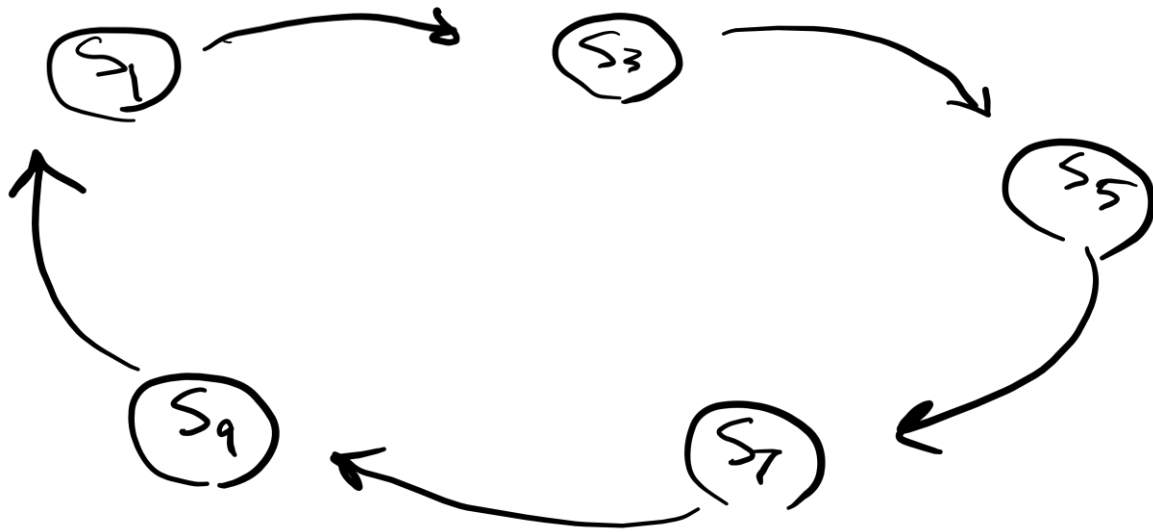
* Reset =0 → Returns to S_1

The module code:

```
1   module odd_counter (
2       input wire clk,
3       input wire reset, // Add reset input
4       output reg [3:0] count
5   );
6
7   always @(posedge clk or posedge reset) begin
8       if (reset) begin
9           count <= 4'b0001; // Reset to 1
10      end else begin
11          case (count)
12              4'b0001: count <= 4'b0011;
13              4'b0011: count <= 4'b0101;
14              4'b0101: count <= 4'b0111;
15              4'b0111: count <= 4'b1001;
16              4'b1001: count <= 4'b0001;
17              default: count <= 4'b0001; // Default to 1
18          endcase
19      end
20  end
21
22  endmodule
```

DLC Project Report Nazanin Foroutan

The test bench code:

```verilog
3    module odd_counter_tb;
4
5    reg clk;
6    reg reset; // Add reset signal
7    wire [3:0] count;
8
9    // Instantiate the odd_counter module
10   odd_counter uut (
11       .clk(clk),
12       .reset(reset), // Connect reset signal
13       .count(count)
14   );
15
16   // Clock generation
17   initial begin
18       clk = 0;
19       forever #5 clk = ~clk; // Toggle clock every 5 time units
20   end
21
22   initial begin
23       $monitor("Time = %0t, Count = %0d", $time, count);
24
25       // Initial reset
26       reset = 1; // Apply reset
27       #10;
28       reset = 0; // Release reset
29
30       // Wait for count to be 5
31       wait (count == 4'b0101);
32       reset = 1;
33       #10;
34       reset = 0;
35
36       #100; // Run for additional 100 time units
37
38       $finish;
39   end
```

Waves and outputs:
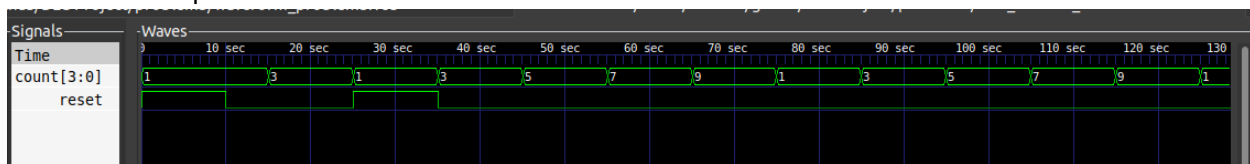

```
Time = 0, Count = 1
Time = 15, Count = 3
Time = 25, Count = 1
Time = 35, Count = 3
Time = 45, Count = 5
Time = 55, Count = 7
Time = 65, Count = 9
Time = 75, Count = 1
Time = 85, Count = 3
Time = 95, Count = 5
Time = 105, Count = 7
Time = 115, Count = 9
Time = 125, Count = 1
Time = 135, Count = 3
```

5- Design and implement a sequence detector to detect 0110110 sequence.

In the Verilog code the main part for state machine:

```verilog
// Next state logic
always @(*) begin
    if(reset == 1) begin
        next_state = S0;
    end else begin
        case (current_state)
            S0: next_state = (in_bit == 0) ? S1 : S0;
            S1: next_state = (in_bit == 1) ? S2 : S1;
            S2: next_state = (in_bit == 1) ? S3 : S1;
            S3: next_state = (in_bit == 0) ? S4 : S0;
            S4: next_state = (in_bit == 1) ? S5 : S1;
            S5: next_state = (in_bit == 1) ? S6 : S1;
            S6: next_state = (in_bit == 0) ? S7 : S0;
            S7: next_state = (in_bit == 0) ? S1 : S5;
            default: next_state = S0;
        endcase
    end
end
```

The encoding for the state machines:

```
// State encoding
parameter S0 = 3'b000;
parameter S1 = 3'b001;
parameter S2 = 3'b010;
parameter S3 = 3'b011;
parameter S4 = 3'b100;
parameter S5 = 3'b101;
parameter S6 = 3'b110;
parameter S7 = 3'b111;
```

The outputs shown in terminal:

```
[Running] Foroutan.Nazanin.402243084.problem5.testbench.v
Time: 5000 | reset: 1 | in_bit: 0 | Current State: 000 | Next State: 000 | seq_detected: 0
Time: 15000 | reset: 0 | in_bit: 0 | Current State: 000 | Next State: 001 | seq_detected: 0
Time: 25000 | reset: 0 | in_bit: 1 | Current State: 001 | Next State: 010 | seq_detected: 0
Time: 35000 | reset: 0 | in_bit: 1 | Current State: 010 | Next State: 011 | seq_detected: 0
Time: 45000 | reset: 0 | in_bit: 0 | Current State: 011 | Next State: 100 | seq_detected: 0
Time: 55000 | reset: 0 | in_bit: 1 | Current State: 100 | Next State: 101 | seq_detected: 0
Time: 65000 | reset: 0 | in_bit: 1 | Current State: 101 | Next State: 110 | seq_detected: 0
Time: 75000 | reset: 0 | in_bit: 0 | Current State: 110 | Next State: 111 | seq_detected: 0
Time: 85000 | reset: 0 | in_bit: 1 | Current State: 111 | Next State: 101 | seq_detected: 1
Time: 95000 | reset: 0 | in_bit: 0 | Current State: 101 | Next State: 001 | seq_detected: 0
[Done] exit with code=0 in 0.025 seconds
```

The diagram:
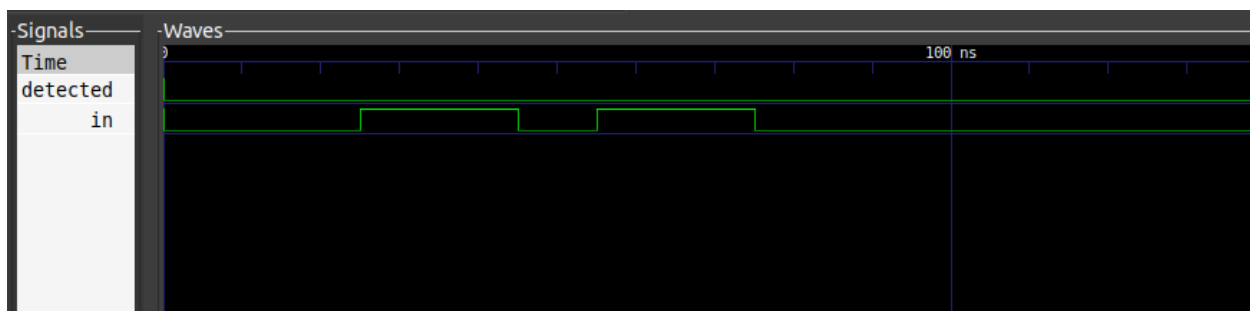
The solution for the state machine:
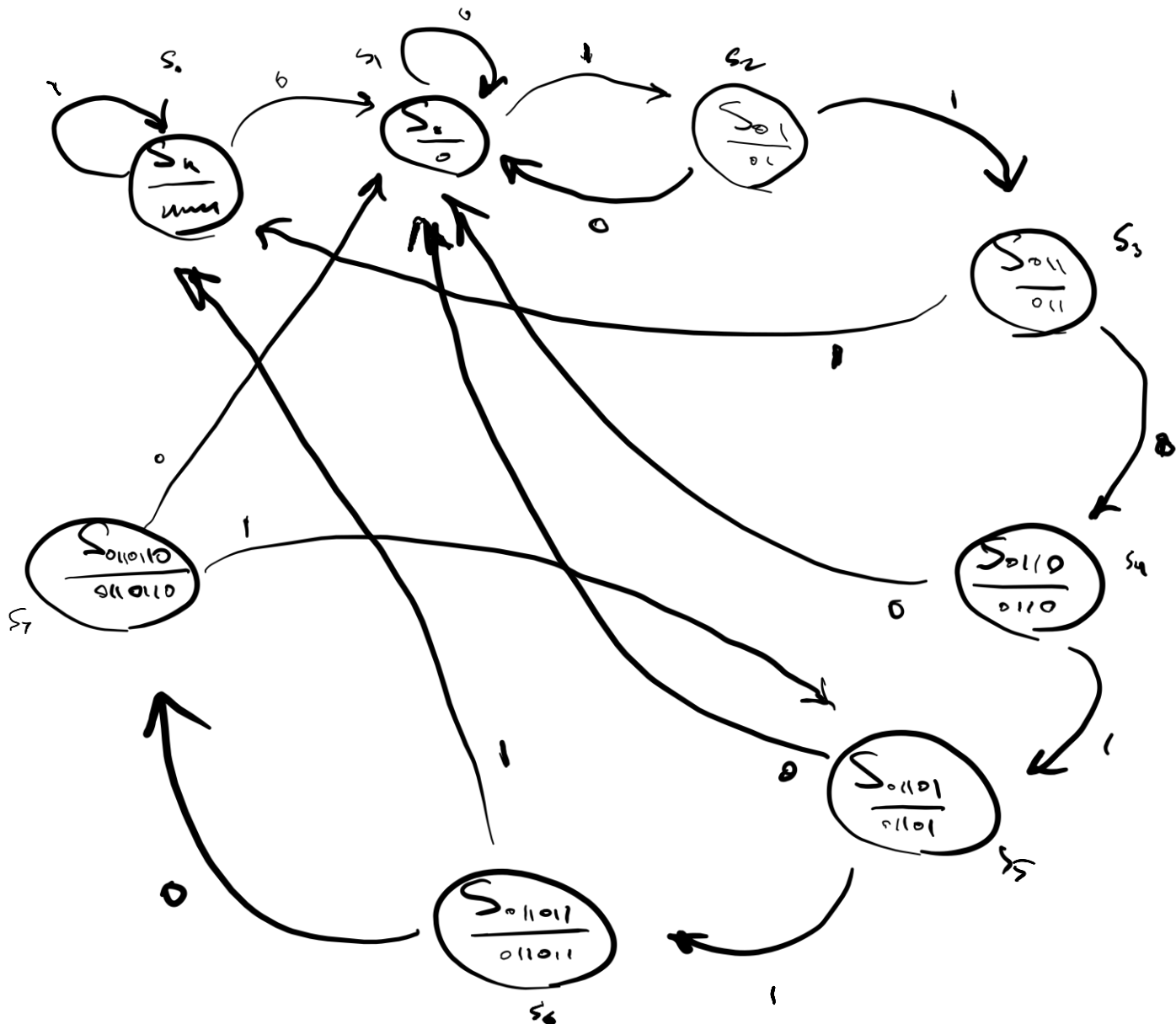
$$S_n \longrightarrow S_0 \qquad S_{0110} \rightarrow S_4$$

$$S_0 \longrightarrow S_1 \qquad S_{01101} \rightarrow S_5$$

$$S_{01} \longrightarrow S_2 \qquad S_{011011} \rightarrow S_6$$

$$S_{011} \rightarrow S_3 \qquad S_{0110110} \rightarrow S_7$$