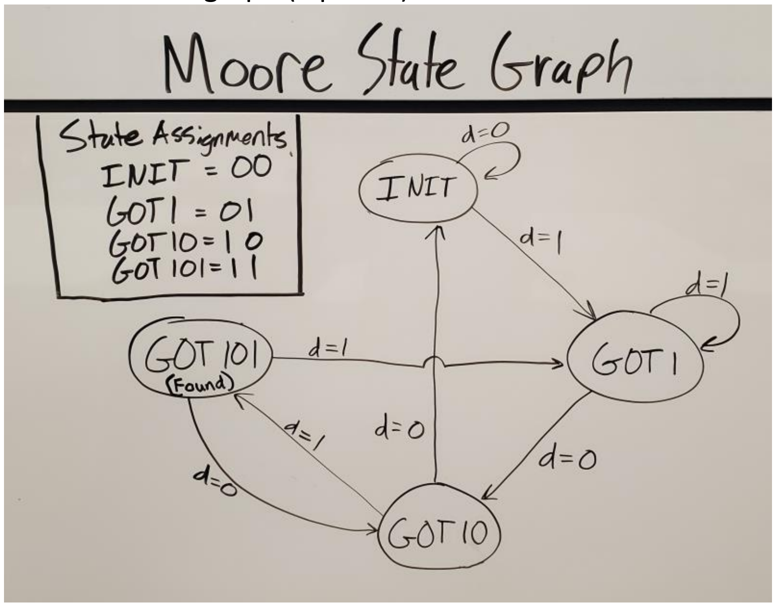
Homework 7 – Verilog State Machines (65 points)

1. Use D-type flip-flops and combinational logic to design a synchronous Moore finite-state machine that monitors input “A” and asserts a binary output “B” if the sequence “101” is observed. For example:

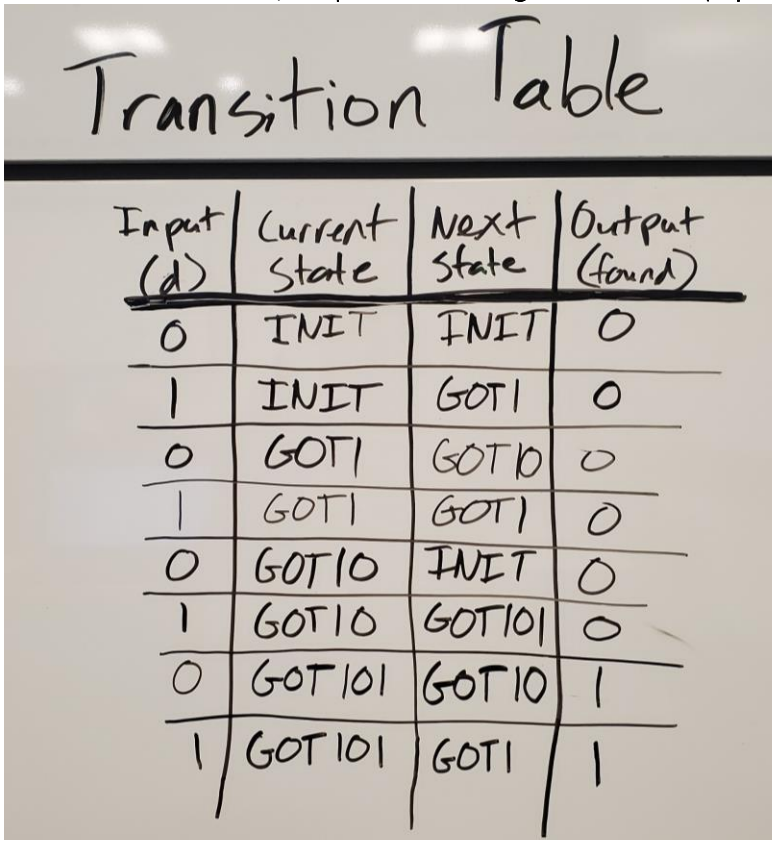
```
If the input data is:    A=010101101
The output will be: B=000101001
```

--------------------------------------→ time

a) Draw the state graph (5 points)



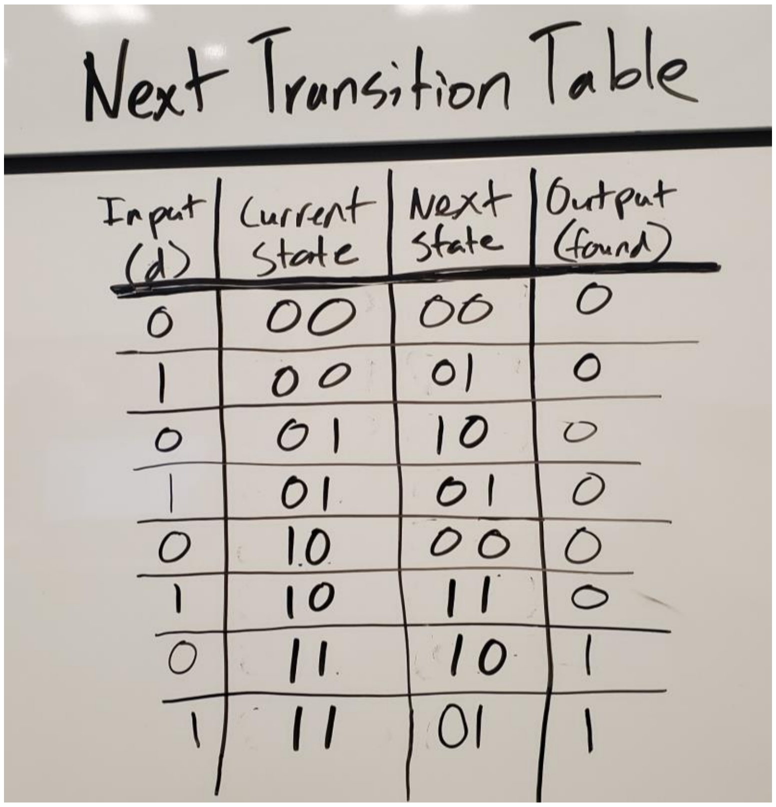
b) Create the next state/output table using state names (5 points)



Transition Table

Input (d)	Current State	Next State	Output (found)
0	INIT	INIT	0
1	INIT	GOT1	0
0	GOT1	GOT10	0
1	GOT1	GOT1	0
0	GOT10	INIT	0
1	GOT10	GOT101	0
0	GOT101	GOT10	1
1	GOT101	GOT1	1

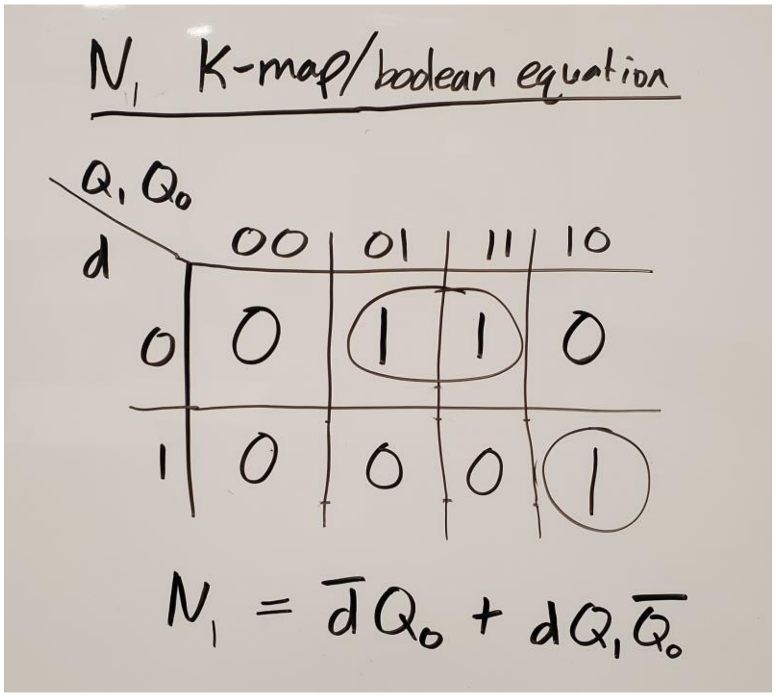
c) Make state assignments and substitute numbers for names in the next state table (5 points)



Next Transition Table

Input (d)	Current State	Next State	Output (found)
0	00	00	0
1	00	01	0
0	01	10	0
1	01	01	0
0	10	00	0
1	10	11	0
0	11	10	1
1	11	01	1

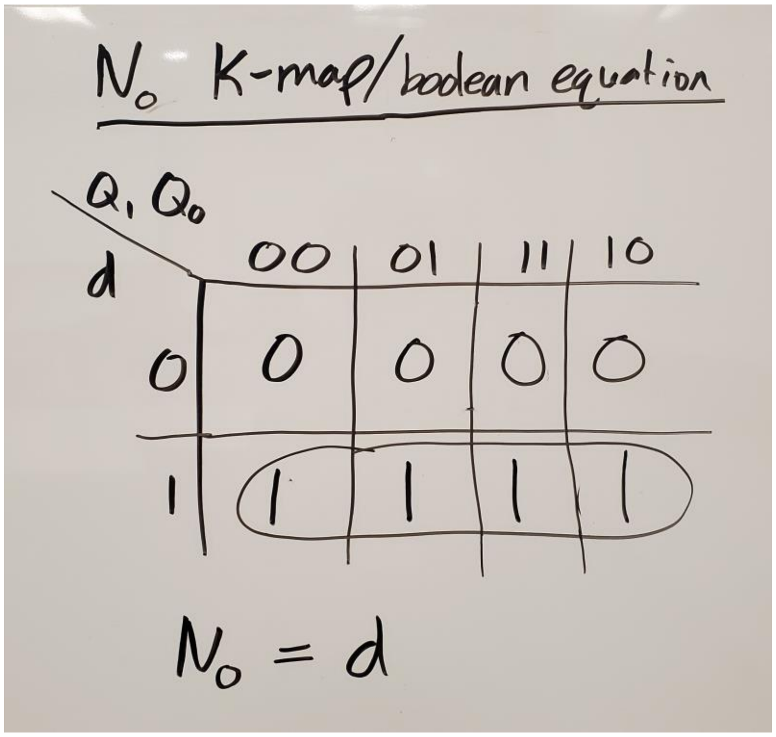
d) Determine the minimal circuit realization of the next state logic and output (5 points)



N_1 K-map/boolean equation

$d \backslash Q_1Q_0$	00	01	11	10
0	0	1	1	0
1	0	0	0	1

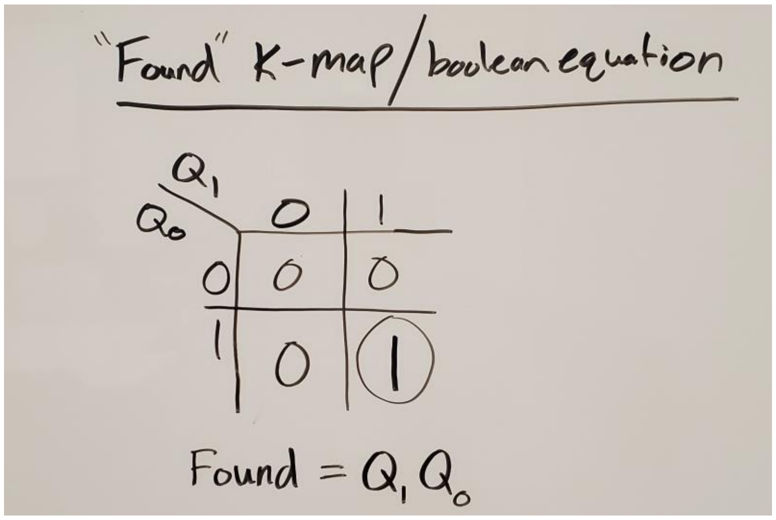
$$N_1 = \bar{d}Q_0 + dQ_1\overline{Q_0}$$



N_0 K-map/boolean equation

$d \backslash Q_1Q_0$	00	01	11	10
0	0	0	0	0
1	1	1	1	1

$N_0 = d$

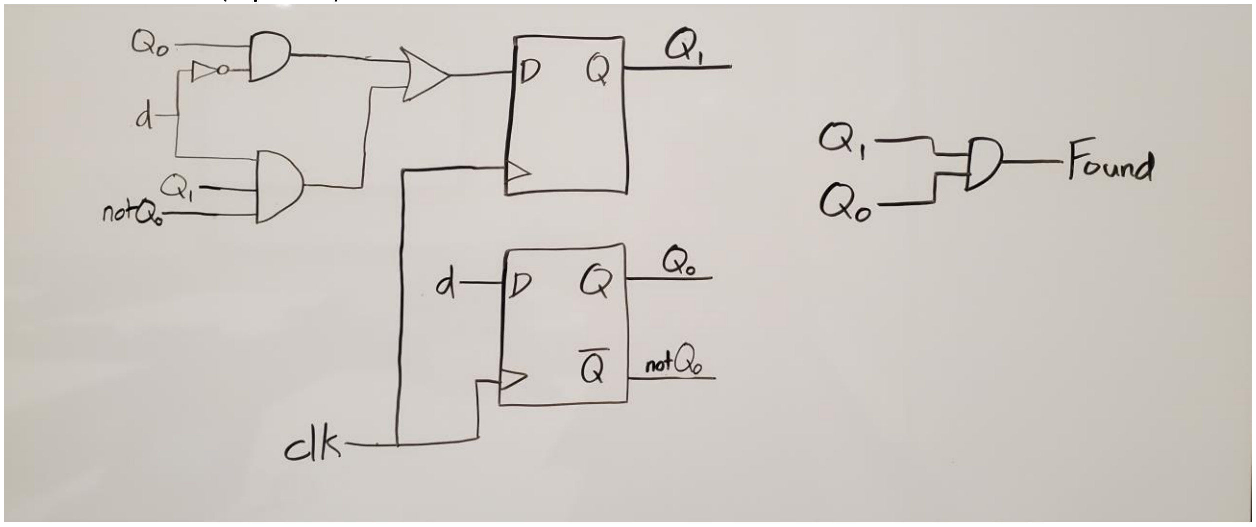


"Found" K-map/boolean equation

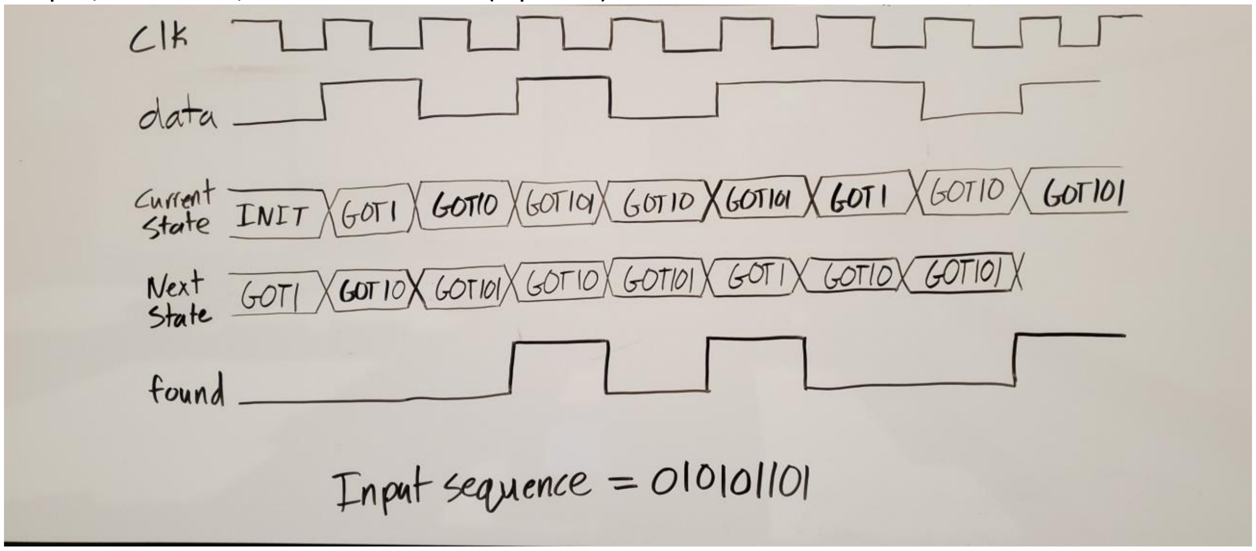
$Q_0 \backslash Q_1$	0	1
0	0	0
1	0	1

$\text{Found} = Q_1 Q_0$

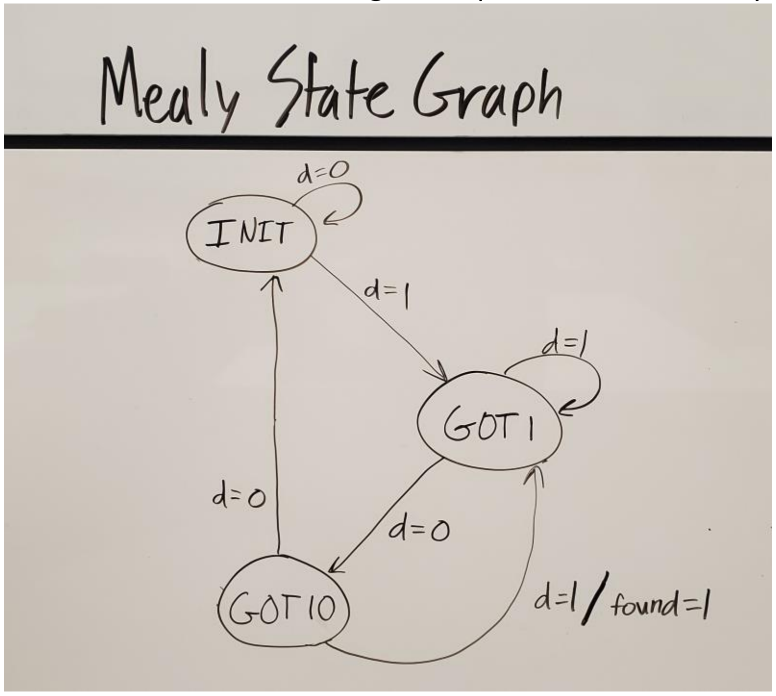
e) Draw the circuit (5 points)



f) Draw a timing diagram using the input sequence, A = 010101101. Show the input, output, next state, and current state (5 points).

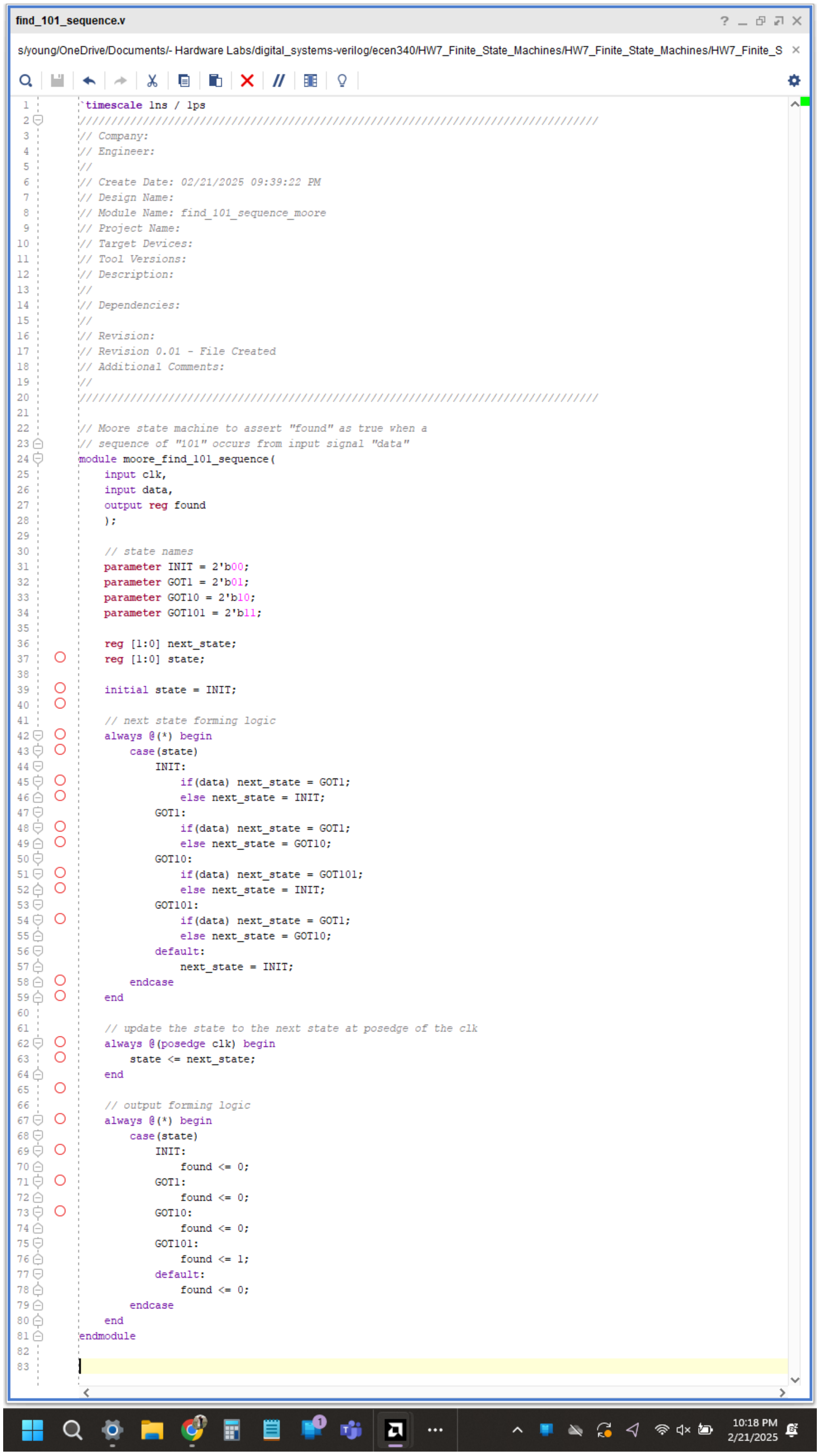


2. Draw the modified State diagram of problem 1 for a Mealy-type State machine (5 points).



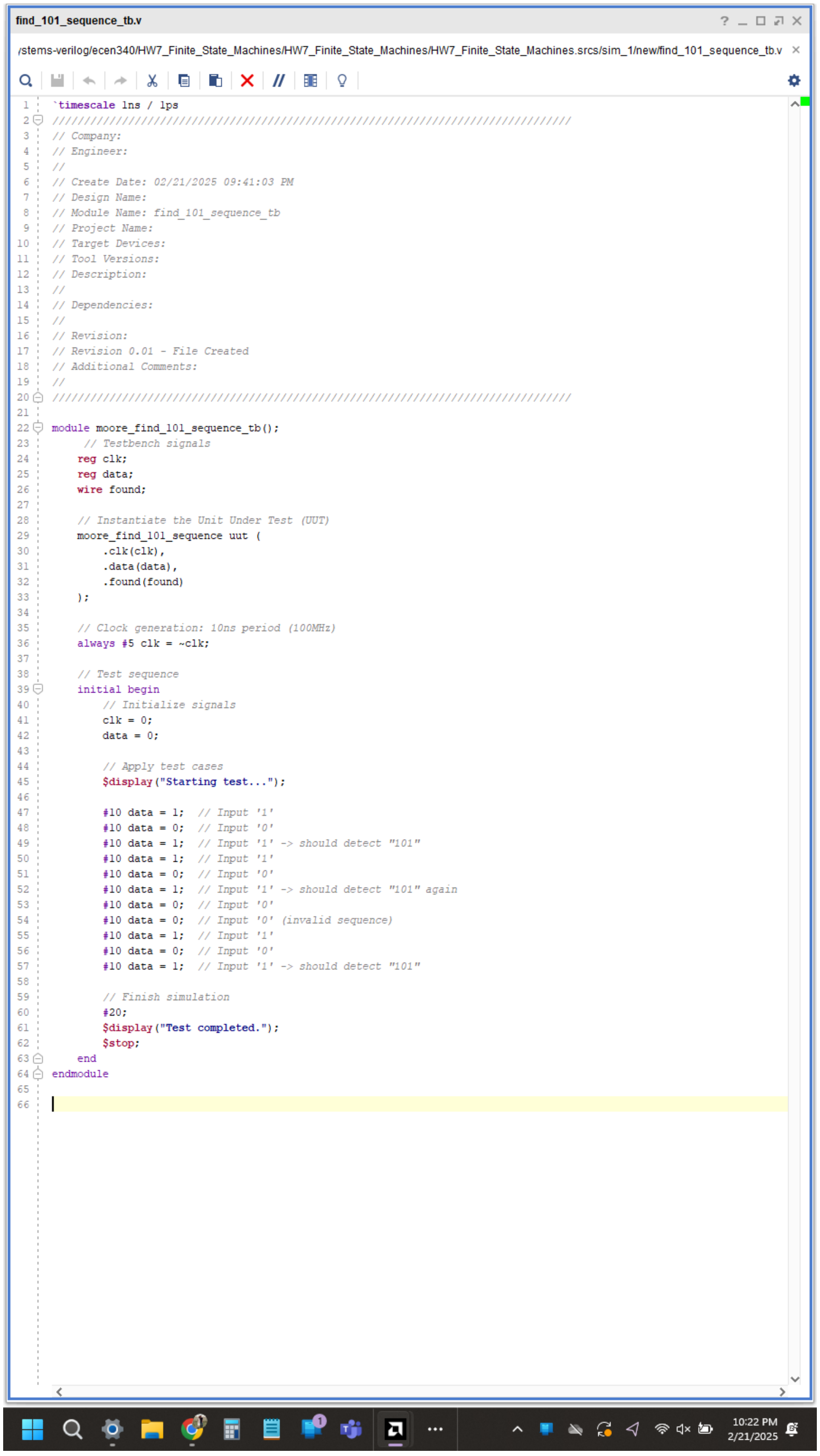
3. Write the Verilog code for the above state machine using Moore-type outputs. When writing the code, please do the following:

- Use parameters for the state names (5 points).
  For example:
  parameter got1 = 2’b00;
- Use a case statement to determine the next state based on the current state (5 points).
- Place the state flip-flops in their own “always” block (5 points).
- Use a separate case statement for the “found” signal (5 points).


find_101_sequence.v
s/young/OneDrive/Documents/- Hardware Labs/digital_systems-verilog/ecen340/HW7_Finite_State_Machines/HW7_Finite_State_Machines/HW7_Finite_S
10:18 PM
2/21/2025

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/21/2025 09:39:22 PM
// Design Name:
// Module Name: find_101_sequence_moore
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// Moore state machine to assert "found" as true when a
// sequence of "101" occurs from input signal "data"
module moore_find_101_sequence(
    input clk,
    input data,
    output reg found
    );

    // state names
    parameter INIT = 2'b00;
    parameter GOT1 = 2'b01;
    parameter GOT10 = 2'b10;
    parameter GOT101 = 2'b11;

    reg [1:0] next_state;
    reg [1:0] state;

    initial state = INIT;

    // next state forming logic
    always @(*) begin
        case(state)
            INIT:
                if(data) next_state = GOT1;
                else next_state = INIT;
            GOT1:
                if(data) next_state = GOT1;
                else next_state = GOT10;
            GOT10:
                if(data) next_state = GOT101;
                else next_state = INIT;
            GOT101:
                if(data) next_state = GOT1;
                else next_state = GOT10;
            default:
                next_state = INIT;
        endcase
    end

    // update the state to the next state at posedge of the clk
    always @(posedge clk) begin
        state <= next_state;
    end

    // output forming logic
    always @(*) begin
        case(state)
            INIT:
                found <= 0;
            GOT1:
                found <= 0;
            GOT10:
                found <= 0;
            GOT101:
                found <= 1;
            default:
                found <= 0;
        endcase
    end
endmodule

```

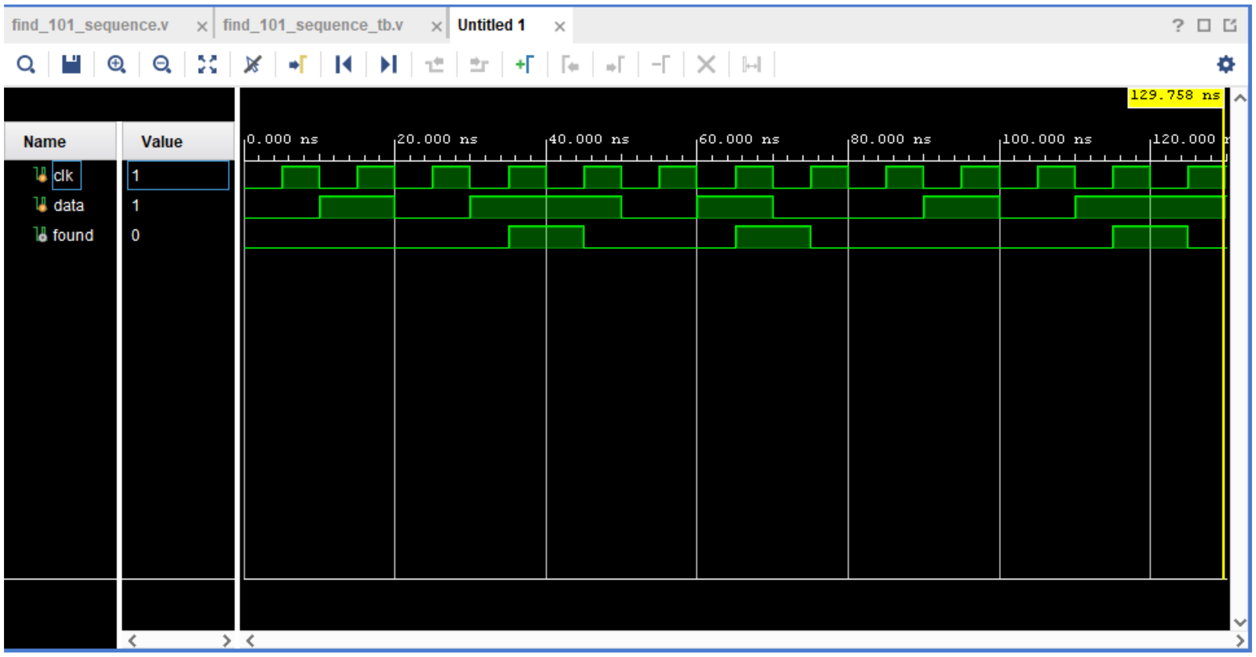
Figure 1: Moore state machine Verilog code



/stems-verilog/ecen340/HW7_Finite_State_Machines/HW7_Finite_State_Machines/HW7_Finite_State_Machines.srcs/sim_1/new/find_101_sequence_tb.v

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/21/2025 09:41:03 PM
// Design Name:
// Module Name: find_101_sequence_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module moore_find_101_sequence_tb();
     // Testbench signals
    reg clk;
    reg data;
    wire found;

    // Instantiate the Unit Under Test (UUT)
    moore_find_101_sequence uut (
        .clk(clk),
        .data(data),
        .found(found)
    );

    // Clock generation: 10ns period (100MHz)
    always #5 clk = ~clk;

    // Test sequence
    initial begin
        // Initialize signals
        clk = 0;
        data = 0;

        // Apply test cases
        $display("Starting test...");

        #10 data = 1;  // Input '1'
        #10 data = 0;  // Input '0'
        #10 data = 1;  // Input '1' -> should detect "101"
        #10 data = 1;  // Input '1'
        #10 data = 0;  // Input '0'
        #10 data = 1;  // Input '1' -> should detect "101" again
        #10 data = 0;  // Input '0'
        #10 data = 0;  // Input '0' (invalid sequence)
        #10 data = 1;  // Input '1'
        #10 data = 0;  // Input '0'
        #10 data = 1;  // Input '1' -> should detect "101"

        // Finish simulation
        #20;
        $display("Test completed.");
        $stop;
    end
endmodule

```

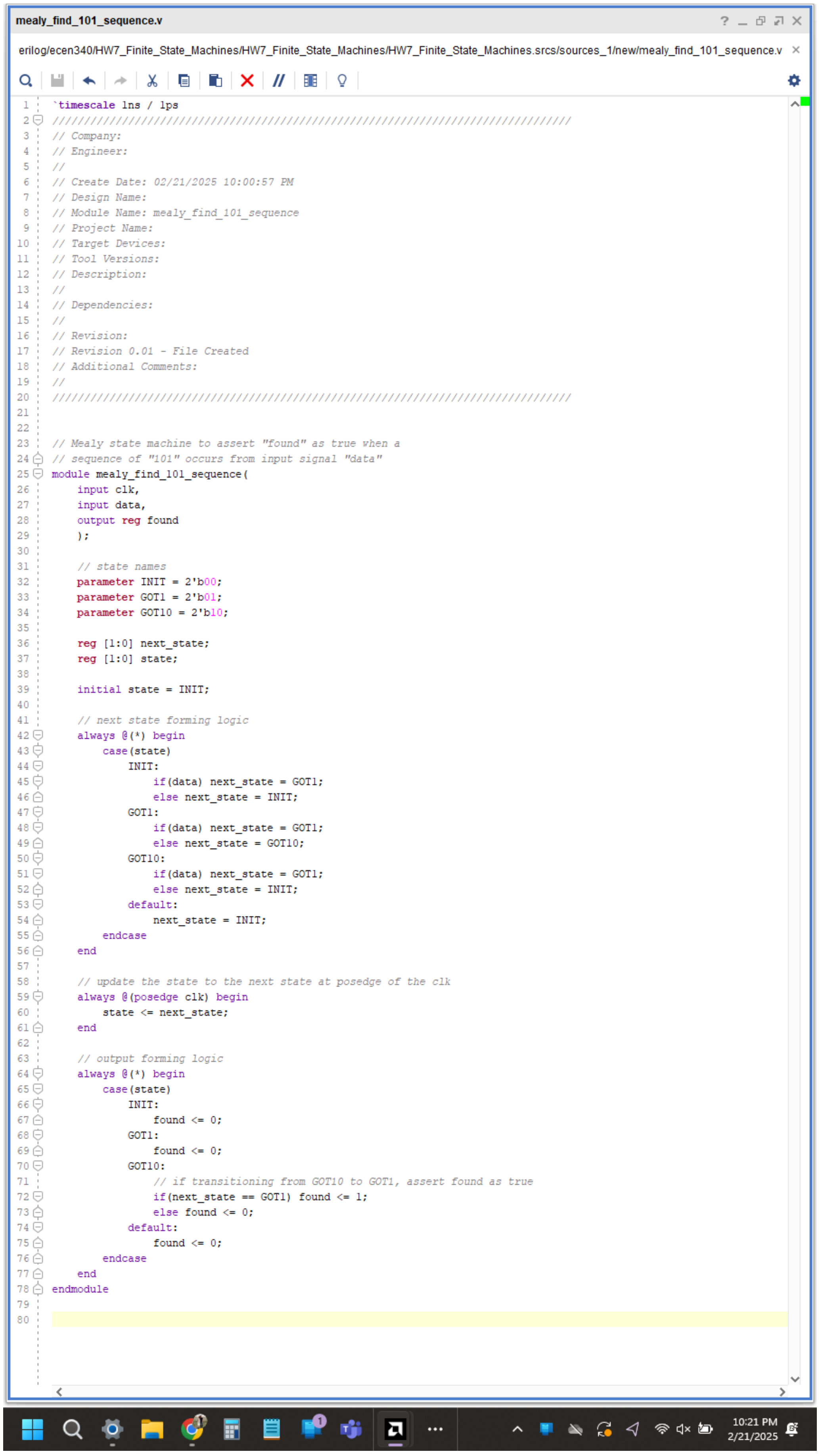
Figure 2: Moore state machine Verilog test bench code



Name	Value
clk	1
data	1
found	0

Figure 3: Moore state machine simulation

4. Modify the Moore-type output code for the Mealy-type outputs (10 points).

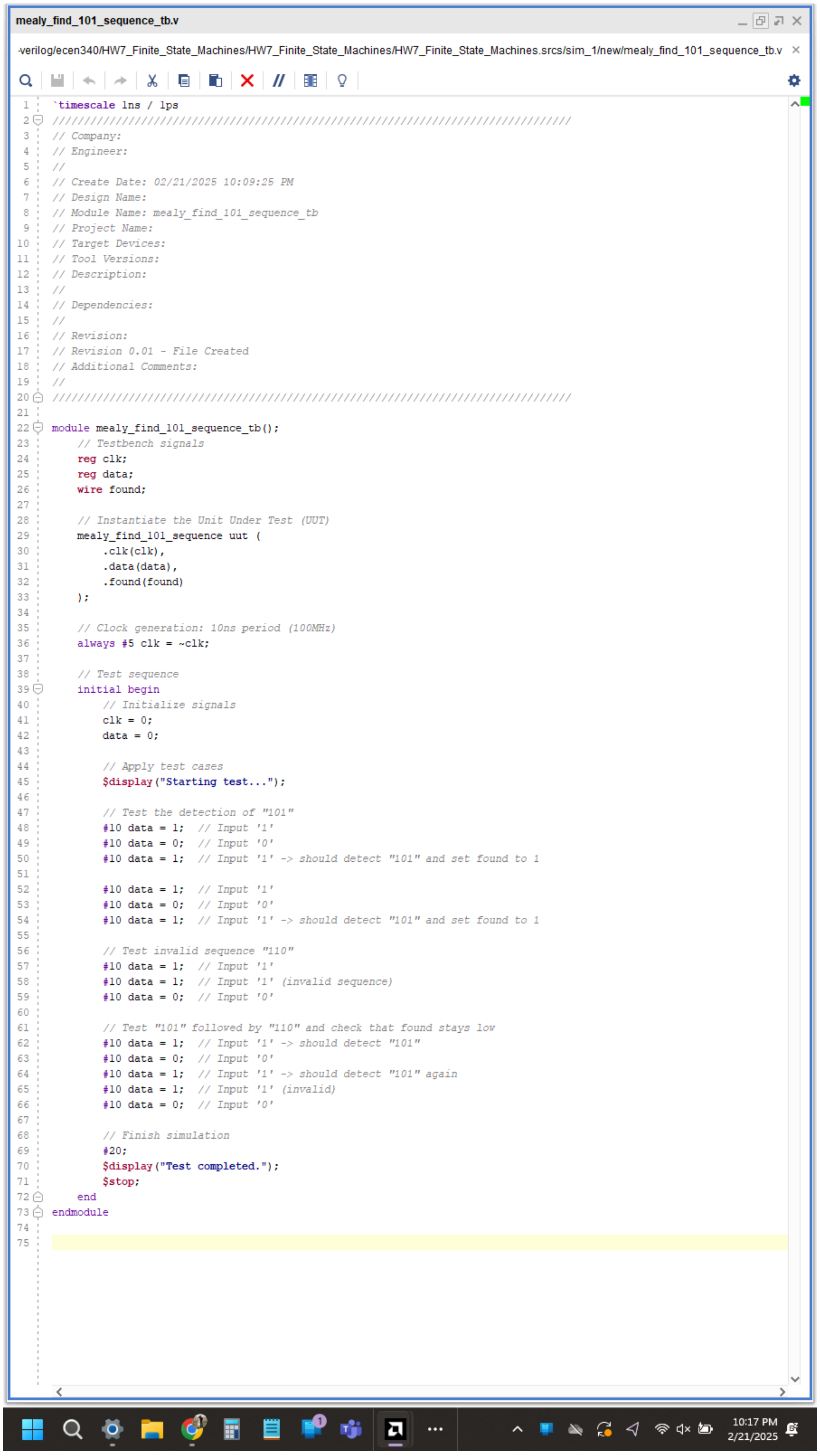

mealy_find_101_sequence.v

erilog/ecen340/HW7_Finite_State_Machines/HW7_Finite_State_Machines/HW7_Finite_State_Machines.srcs/sources_1/new/mealy_find_101_sequence.v

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/21/2025 10:00:57 PM
// Design Name:
// Module Name: mealy_find_101_sequence
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


// Mealy state machine to assert "found" as true when a
// sequence of "101" occurs from input signal "data"
module mealy_find_101_sequence(
    input clk,
    input data,
    output reg found
    );

    // state names
    parameter INIT = 2'b00;
    parameter GOT1 = 2'b01;
    parameter GOT10 = 2'b10;

    reg [1:0] next_state;
    reg [1:0] state;

    initial state = INIT;

    // next state forming logic
    always @(*) begin
        case(state)
            INIT:
                if(data) next_state = GOT1;
                else next_state = INIT;
            GOT1:
                if(data) next_state = GOT1;
                else next_state = GOT10;
            GOT10:
                if(data) next_state = GOT1;
                else next_state = INIT;
            default:
                next_state = INIT;
        endcase
    end

    // update the state to the next state at posedge of the clk
    always @(posedge clk) begin
        state <= next_state;
    end

    // output forming logic
    always @(*) begin
        case(state)
            INIT:
                found <= 0;
            GOT1:
                found <= 0;
            GOT10:
                // if transitioning from GOT10 to GOT1, assert found as true
                if(next_state == GOT1) found <= 1;
                else found <= 0;
            default:
                found <= 0;
        endcase
    end
endmodule

```

10:21 PM
2/21/2025

Figure 4: Mealy state machine Verilog code



mealy_find_101_sequence_tb.v

-verilog/ecen340/HW7_Finite_State_Machines/HW7_Finite_State_Machines/HW7_Finite_State_Machines.srcs/sim_1/new/mealy_find_101_sequence_tb.v

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/21/2025 10:09:25 PM
// Design Name:
// Module Name: mealy_find_101_sequence_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module mealy_find_101_sequence_tb();
    // Testbench signals
    reg clk;
    reg data;
    wire found;

    // Instantiate the Unit Under Test (UUT)
    mealy_find_101_sequence uut (
        .clk(clk),
        .data(data),
        .found(found)
    );

    // Clock generation: 10ns period (100MHz)
    always #5 clk = ~clk;

    // Test sequence
    initial begin
        // Initialize signals
        clk = 0;
        data = 0;

        // Apply test cases
        $display("Starting test...");

        // Test the detection of "101"
        #10 data = 1;  // Input '1'
        #10 data = 0;  // Input '0'
        #10 data = 1;  // Input '1' -> should detect "101" and set found to 1

        #10 data = 1;  // Input '1'
        #10 data = 0;  // Input '0'
        #10 data = 1;  // Input '1' -> should detect "101" and set found to 1

        // Test invalid sequence "110"
        #10 data = 1;  // Input '1'
        #10 data = 1;  // Input '1' (invalid sequence)
        #10 data = 0;  // Input '0'

        // Test "101" followed by "110" and check that found stays low
        #10 data = 1;  // Input '1' -> should detect "101"
        #10 data = 0;  // Input '0'
        #10 data = 1;  // Input '1' -> should detect "101" again
        #10 data = 1;  // Input '1' (invalid)
        #10 data = 0;  // Input '0'

        // Finish simulation
        #20;
        $display("Test completed.");
        $stop;
    end
endmodule

```

10:17 PM
2/21/2025

Figure 5: Mealy state machine Verilog test bench code

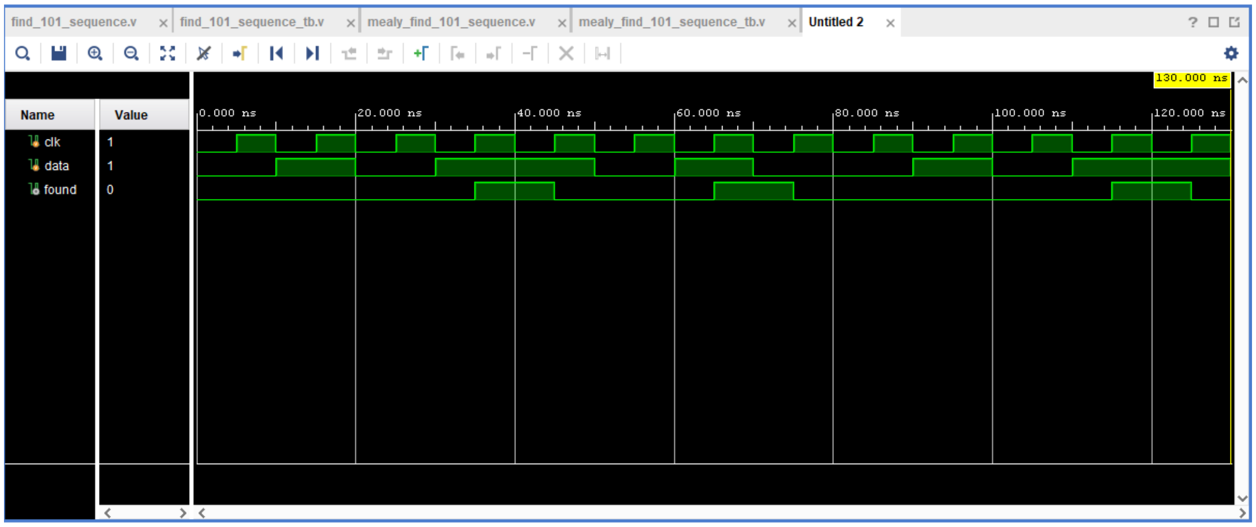


Figure 6: Mealy state machine simulation