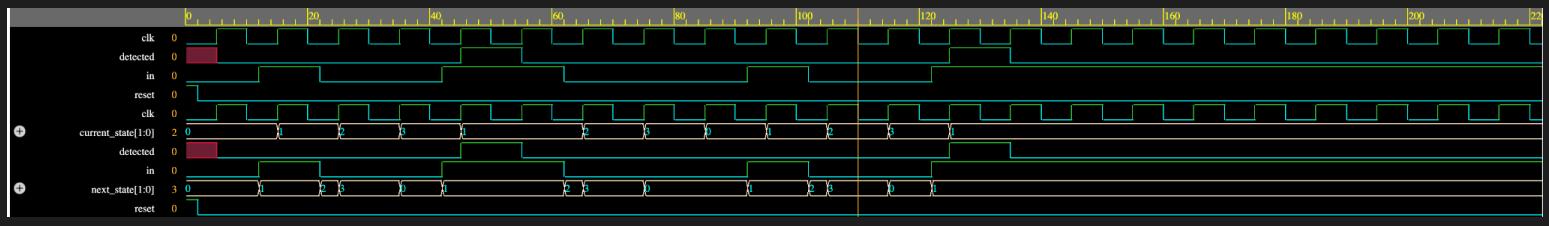
Designing a custom sequence detector



testbench.sv

```
module sequence_detector_1001_tb;
    reg clk;
    reg reset;
    reg in;
    wire detected;

    sequence_detector_1001 uut (
        .clk(clk),
        .reset(reset),
        .in(in),
        .detected(detected)
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #5 clk = ~clk; // 10ns clock period
    end

    initial begin
        $dumpfile("dump.vcd");
        $dumpvars(0, sequence_detector_1001_tb);

        reset = 1;
        in = 0;
        #2 reset = 0; // Release reset after 2ns

        #10 in = 1;
        #10 in = 0;
        #10 in = 0;
        #10 in = 1; // should detect 1001'

        #10 in = 1;
        #10 in = 0;
        #10 in = 0;
        #10 in = 0; // no detection

        #10 in = 1;
        #10 in = 0;
        #10 in = 0;
        #10 in = 1; // should detect 1001'

        #100 $finish;
    end
endmodule
```

design.sv

```
module sequence_detector_1001 (
    input wire clk,
    input wire reset,
    input wire in,
    output reg detected
);

    typedef enum reg [1:0] {
        S0 = 2'b00, // waiting for the first 1
        S1 = 2'b01, // 1 detected, looking for 0
        S2 = 2'b10, // 10 detected, looking for another 0
        S3 = 2'b11  // 100 detected, looking for 1
    } state_t;

    state_t current_state, next_state;

    always @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= S0;
        else
            current_state <= next_state;
    end

    always @(*) begin
        case (current_state)
            S0: if (in == 1)
                    next_state = S1;
                else
                    next_state = S0;
            S1: if (in == 0)
                    next_state = S2;
                else
                    next_state = S1;
            S2: if (in == 0)
                    next_state = S3;
                else
                    next_state = S0;
            S3: if (in == 1)
                    next_state = S1;
                else
                    next_state = S0;
            default: next_state = S0;
        endcase
    end

    always @(posedge clk) begin
        if (reset)
            detected <= 1'b0;
        else if (current_state == S3 && in == 1)
            detected <= 1'b1;
        else
```

```
            detected <= 1'b0;
        end
    endmodule
```