Project 87: Sequence Detector

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma , Ayush Jain , Gati Goyal, Nikunj Agrawal

Documentation Specialist: Dhruv Patel & Nandini Maheshwari

Contents

1 Project Overview

A Sequence Detector identifies a specific binary sequence (e.g., 1011) in a stream of input bits and outputs a signal (1) when the sequence is detected. It can be designed as an overlapping or non-overlapping detector using a Moore or Mealy state machine. The design process involves creating a state transition diagram, coding the logic in Verilog or VHDL, and simulating it using tools like ModelSim or Xilinx Vivado to verify functionality with test cases. The project can be enhanced by counting occurrences of the sequence or allowing dynamic sequence reconfiguration. This design is commonly used in communication and control systems.

2 Sequence Detector

2.1 Key Components of Sequence Detector

Input Signals:

- Data Input (X): Serial bitstream to be monitored for the sequence.
- Clock (CLK): Synchronizes state transitions.
- Reset (RST): Initializes the system to a default state.

Output Signal:

- Detection Output (Y): High (1) when the target sequence is detected.

State Machine:

- States: Represent progress toward recognizing the sequence.
- Transitions: Define how the system moves between states based on input bits.

Logic Design:

- Combinational Logic: Determines state transitions and outputs.
- Sequential Logic: Maintains the current state using flip-flops.

Memory:

- Used to store the current state of the sequence detector.

Controller:

- Governs the flow of data and manages state transitions.

Testbench:

- Simulates the detector with various input patterns to verify correctness.

2.2 Working of Sequence Detector

A Sequence Detector works by identifying a predefined binary sequence in a continuous stream of input bits. The process starts with the system being initialized to an idle state, usually through a reset signal. This ensures the detector begins operation from a known state. Input bits (X) are fed into the detector serially, synchronized with a clock signal (CLK). Each bit is processed in real-time, and the detector evaluates whether the input matches any part of the target sequence.

The detector operates as a Finite State Machine (FSM), with states representing progress toward detecting the full sequence. For example, to detect the sequence 1011, the detector transitions through states corresponding to partial matches: detecting 1, then 10, then 101, and finally 1011. Transitions between states are determined by the current input bit and the detector's current state. Depending on the design, the detector can be either a Moore FSM, where the output depends solely on the current state, or a Mealy FSM, where the output depends on both the current state and input.

When the detector recognizes the complete sequence, it generates an output signal (Y) that goes high (1). If the sequence is not yet complete, the output remains low (0). In an overlapping detector, the system can immediately continue checking for another occurrence of the sequence, even if the sequence starts within the detected bits. In contrast, a non-overlapping detector resets to the idle state before searching for the sequence again.

This detection process is repeated continuously for each new input bit, enabling the system to monitor real-time data streams. Sequence detectors are commonly used in applications like communication systems, pattern recognition, and control logic, where recognizing specific binary patterns is crucial.

2.3 RTL Code

Listing 1: Sequence Detector

```


module sequence_detector (
    input  logic clk, reset,
    input  logic bit_in,        // Serial input bit stream
    output logic sequence_detected // Asserted when sequence 1011 is
       detected
);

    // Define states for Moore machine
    typedef enum logic [2:0] {
        IDLE, S1, S10, S101, S1011
    } state_t;

    state_t current_state, next_state;

    // State transition logic
    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            current_state <= IDLE;
        else
            current_state <= next_state;
    end

    // Next state logic
    always_comb begin
        next_state = current_state;
        sequence_detected = 0;
        case (current_state)
            IDLE:   next_state = (bit_in) ? S1 : IDLE;
            S1:     next_state = (bit_in) ? S1 : S10;
            S10:    next_state = (bit_in) ? S101 : IDLE;
            S101:   next_state = (bit_in) ? S1 : S1011;
            S1011: begin
                next_state = (bit_in) ? S1 : S10;
                sequence_detected = 1;
            end
        endcase
    end
endmodule
```

2.4 Testbench

Listing 2: Sequence Detector

```

module tb_sequence_detector();
    logic clk, reset;
    logic bit_in;
    logic sequence_detected;

    sequence_detector uut (
        .clk(clk),
        .reset(reset),
        .bit_in(bit_in),
        .sequence_detected(sequence_detected)
    );

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

    // Test scenario
    initial begin
        reset = 1; bit_in = 0;
        #10 reset = 0;

        // Test sequence: 1011011
        #10 bit_in = 1; // Start sequence: 1
        #10 bit_in = 0; // 10
        #10 bit_in = 1; // 101
        #10 bit_in = 1; // 1011 - Detected
        #10 bit_in = 0; // Restart
        #10 bit_in = 1; // 101
        #10 bit_in = 1; // 1011 - Detected again
        #10 bit_in = 0; // Restart
        #10 bit_in = 1; // Restart

        // Stop simulation
        #50 $stop;
    end

    // Monitor outputs
    initial begin
        $monitor("Time: %0t | Bit In: %b | Sequence Detected: %b |
           Current State: %b",
                 $time, bit_in, sequence_detected, uut.current_state);
    end
endmodule
```

3 Results

3.1 Simulation

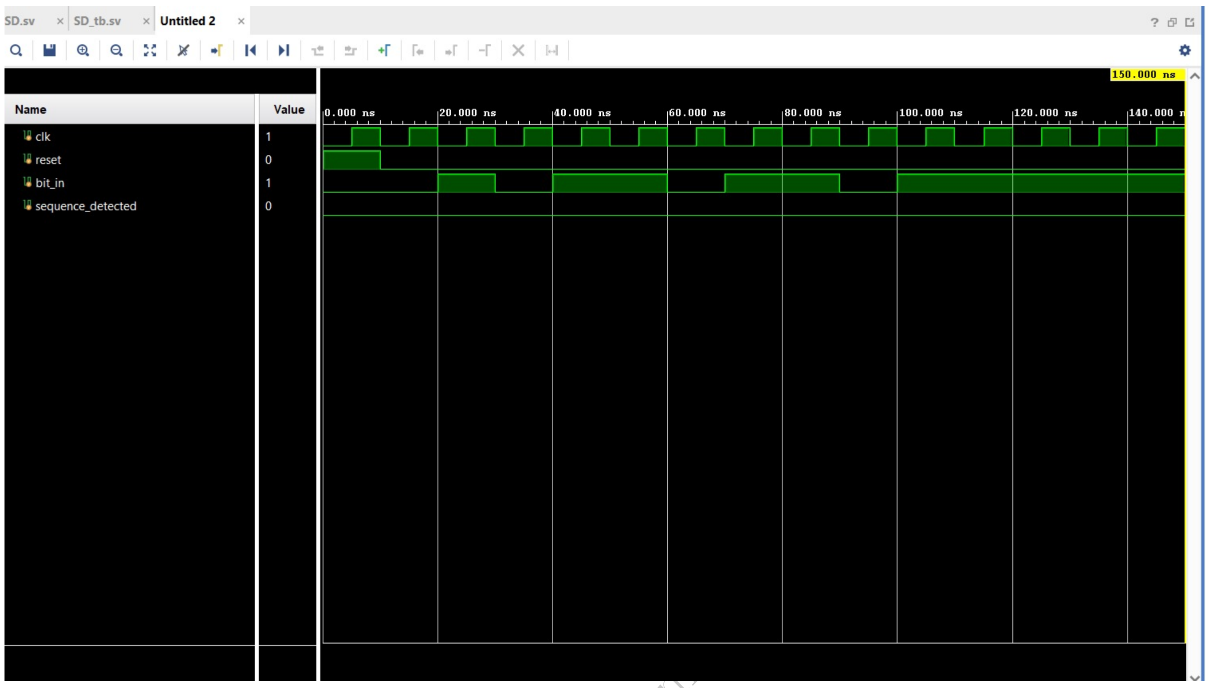

Figure 1: Simulation of Sequence Detector

3.2 Schematic

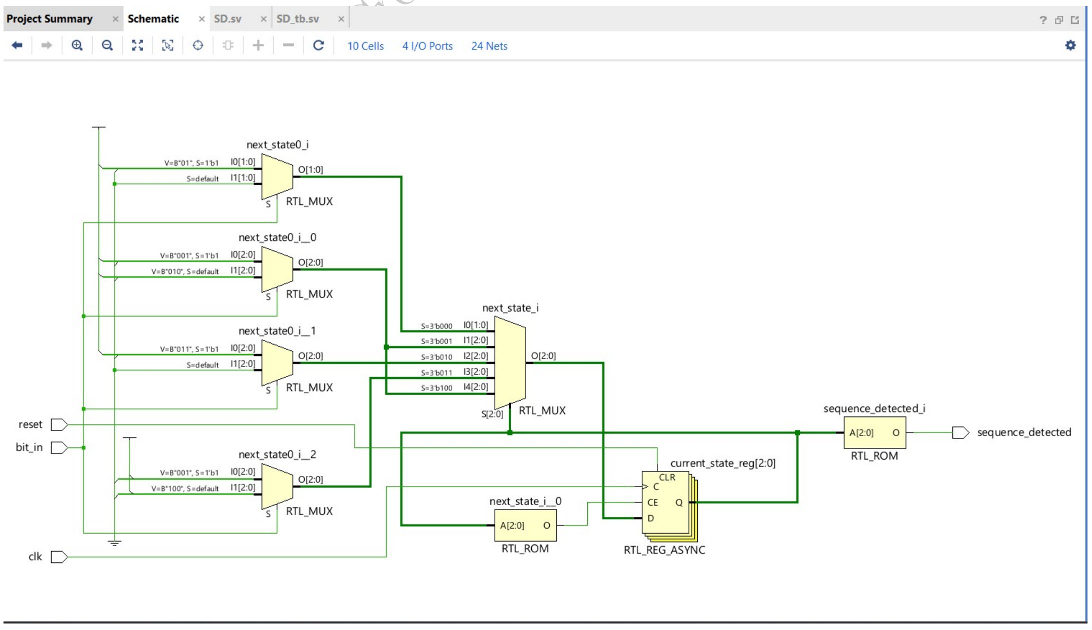

Figure 2: Schematic of Sequence Detector

3.3 Synthesis Design

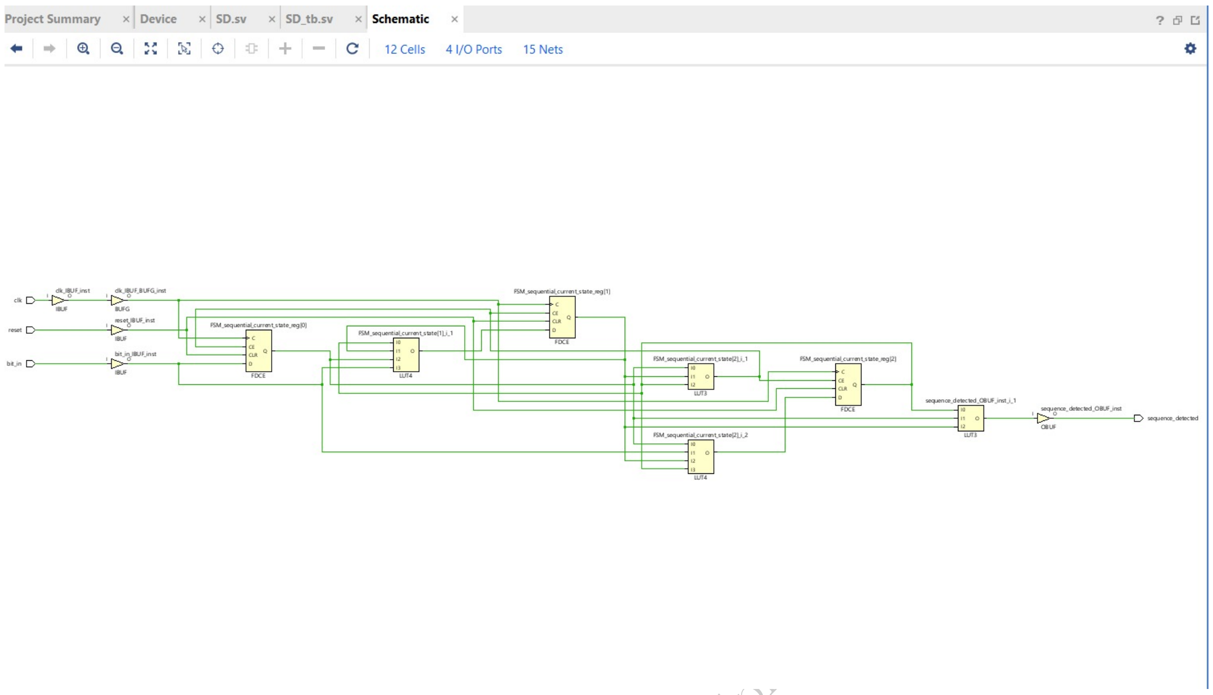
Project Summary
Device
SD.sv
SD_tb.sv
Schematic
12 Cells
4 I/O Ports
15 Nets
clk
reset
bit_in
dlk_IBUF_inst
IBUF
dlk_IBUF_BUFG_inst
BUFG
reset_IBUF_inst
IBUF
bit_in_IBUF_inst
IBUF
FSM_sequential_current_state_reg[0]
FDCE
FSM_sequential_current_state[1]_i_1
LUT4
FSM_sequential_current_state_reg[1]
FDCE
FSM_sequential_current_state[2]_i_1
LUT3
FSM_sequential_current_state_reg[2]
FDCE
FSM_sequential_current_state[2]_i_2
LUT4
sequence_detected_OBUF_inst_i_1
LUT3
sequence_detected_OBUF_inst
OBUF
sequence_detected

Figure 3: Synthesis Design of Sequence Detector

4 Advantages of Sequence Detector

- Efficiently identifies specific binary patterns in real-time.
- Requires minimal hardware, making it cost-effective.
- Customizable to detect any desired sequence.
- Supports overlapping detection for improved efficiency.
- Versatile applications in communication, control systems, and pattern recognition.
- Easily simulated and tested using modern tools.
- Scalable for longer or more complex sequences.

5 Disadvantages of Sequence Detector

- Fixed design limits flexibility for multiple sequences.
- Complexity increases with longer sequences.
- Prone to errors in noisy environments.
- Requires more hardware for complex detection.
- Sensitive to timing issues like clock skew.
- Limited scalability for detecting multiple sequences.
- Debugging complex state machines can be challenging.

6 Applications of Sequence Detector

Pattern Recognition:

- Employed in digital systems for recognizing predefined patterns, such as in speech recognition or biometric systems.

Control Systems:

- Used in control logic for detecting sequences of events or inputs, enabling state transitions in automated systems.

Error Detection:

- In digital circuits, sequence detectors can identify incorrect data sequences or flag errors in transmitted data.

Automated Testing:

- Employed in testing systems to detect specific sequences in test vectors, ensuring proper function of digital devices.

State Machines:

- Sequence detectors serve as key components in state machines that drive decision-making in complex digital systems.

7 Conclusion

Sequence detectors are fundamental components in digital systems, playing a crucial role in pattern recognition, error detection, synchronization, and control. By efficiently identifying specific sequences within a data stream, they support applications across communication systems, signal processing, control logic, and more. Despite their advantages, such as minimal hardware requirements and customizability, they do have limitations, including increased complexity for longer sequences and sensitivity to noise. With careful design and optimization, these challenges can be managed, making sequence detectors versatile and reliable tools in both simple and complex digital systems.

8 FAQs

1.What is a sequence detector?

- A sequence detector is a digital circuit designed to detect a specific sequence of bits in an input stream and output a signal (usually high or 1) when the sequence is detected.

2.What is the difference between Moore and Mealy sequence detectors?

- Moore FSM: The output depends only on the current state.
- Mealy FSM: The output depends on both the current state and the current input.

3.How do sequence detectors work?

- Sequence detectors use a finite state machine (FSM) to track progress toward detecting the sequence. The machine transitions through states based on input bits, and when the target sequence is matched, it generates an output signal.

4.What are the advantages of using a sequence detector?

- Sequence detectors are cost-effective, customizable, and efficient for recognizing patterns in data streams. They have low hardware requirements and can be used in various applications such as communication systems, error detection, and control systems.

5.What are the disadvantages of a sequence detector?

- Sequence detectors can become complex for long sequences, are sensitive to noisy inputs, and may require more hardware resources for detecting multiple sequences or handling overlapping patterns.

6.What applications use sequence detectors?

- Sequence detectors are used in communication systems, pattern recognition, data encryption, control systems, error detection, and digital signal processing, among others.

7.Can sequence detectors detect overlapping sequences?

- Yes, sequence detectors can be designed to detect overlapping sequences, meaning they can continue searching for another sequence without resetting after detecting one.

8.What is the impact of sequence length on the detector?

- As the length of the sequence increases, the complexity of the sequence detector also increases. More states and transitions are required, which can complicate the design and increase hardware requirements.

9.How can sequence detectors be optimized?

- Optimizations can include minimizing the number of states, reducing hardware resources, using efficient state transition logic, and employing tools for simulation and testing to identify and fix design issues early.

10.What tools are used for designing and testing sequence detectors?

- Tools like Xilinx Vivado, Quartus, ModelSim, and Verilog/VHDL simulators are commonly used for designing, simulating, and testing sequence detectors.