SEQUENCE DETECTOR USING VERILOG

Varsha A S

Department of electronics and Telecommunication , Dr .Ambedkar Institute of Technology , Mallathahalli , Bengaluru 560058 , Karnataka , India

ABSTRACT

A sequence detector is a digital circuit designed to identify a specific sequence of binary values in an input stream. This paper presents the design and implementation of a sequence detector using Verilog, a hardware description language (HDL) that facilitates the modeling and simulation of digital systems. The primary function of the sequence detector is to recognize a predefined sequence of bits (e.g., 1011, 1101, etc.) in a continuous input bit stream and generate an output signal indicating whether the sequence has been detected.

The system is implemented using finite state machines (FSMs), where each state corresponds to a partial match of the desired sequence. The Verilog code defines the behavior of the FSM, including the transitions between states based on the input bits, and the corresponding output when the sequence is detected. The system's architecture is designed to be modular, efficient, and capable of handling sequences of varying lengths.

The implementation employs a Moore or Mealy type FSM, depending on the design requirements, with inputs, outputs, and clock signals properly synchronized. The design is verified through simulation in a testbench environment, ensuring that the sequence detector performs as expected under various input conditions. Results demonstrate the ability of the sequence detector to reliably recognize the target sequence, making it suitable for applications such as data transmission, error detection, and pattern recognition in digital systems.

In summary, this project demonstrates how Verilog can be used to model and simulate a sequence detector, providing an efficient and scalable solution for sequence recognition in digital logic circuits.

INTRODUCTION

In digital systems, sequence detection is a fundamental task that involves identifying a specific sequence of bits in a stream of data. Sequence detectors are widely used in applications such as data communication, error detection, and signal processing. A sequence detector monitors an input stream and outputs a signal when a predetermined sequence of bits is detected.

This project focuses on the design and implementation of a sequence detector using Verilog, a hardware description language (HDL) used for modeling electronic systems. The goal of this project is to create a Verilog-based design for detecting a specific binary sequence, such as "101" or "1101", within a stream of input bits. Upon detection of the specified sequence, the circuit will generate an output signal, indicating the successful recognition of the sequence.

Objectives:

Design a Sequence Detector: Implement a sequence detector capable of recognizing a specific sequence of bits in a binary input stream.

Verilog Implementation: Use Verilog to design the sequence detector, focusing on the logic design, state transitions, and control signals.

Simulation and Testing: Simulate the design using a Verilog simulator (such as ModelSim or Vivado) to verify functionality and correctness.

Optimization: Explore different methods to optimize the design for resource efficiency, speed, or power consumption.

Key Concepts:

Finite State Machine (FSM): The sequence detector is typically designed using an FSM, where the states represent the progress of the detection process.

State Transitions: The FSM will transition between different states based on the input bit and the detected sequence.

Synchronous Design: The sequence detection is synchronized with a clock signal, ensuring that the detector works reliably with timed inputs.

The project will involve both theoretical aspects, such as state machine design, and practical skills in digital design and simulation using Verilog. The resulting sequence detector can be used as a component in larger systems such as communication interfaces, protocol decoders, or any application requiring sequence recognition.

TYPES OF SEQUENCE DETECTOR

Here are the primary types of sequence detectors:

1. Moore Sequence Detector

Definition: In a Moore sequence detector, the output is determined by the state of the system rather than the input.

How it works: The output is generated based on the current state, and each state represents a part of the sequence.

Advantages:

Fewer transitions than Mealy machines.

Output is stable, as it depends only on the current state.

Example: Detecting a sequence like 101 would involve creating states for partial progress (e.g., 0, 1, and 101 states), where the output is generated when the final state (representing the complete sequence) is reached.

2. Mealy Sequence Detector

Definition: In a Mealy sequence detector, the output depends on both the current state and **the current input.**

How it works: The system transitions between states based on the input, and the output is immediately determined by the current state and input combination.

Advantages:

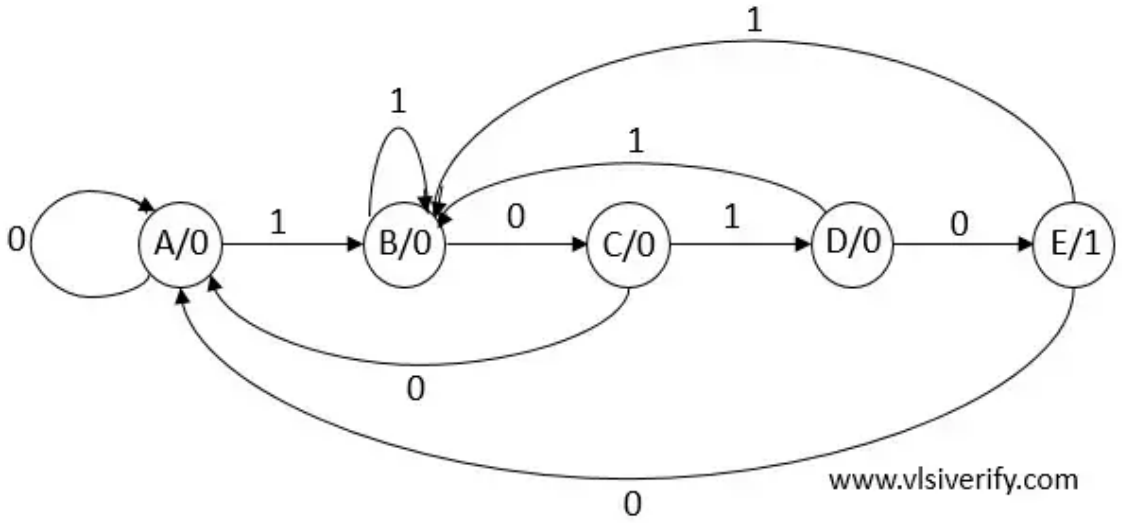
Can be more efficient than Moore machines because the output can change without waiting for the next clock cycle.

Fewer states might be required compared to Moore machines.

Example: A Mealy machine detecting a sequence like 101 might transition through states depending on both the input and the current state, producing output immediately when the full sequence is detected.

MOORE SEQUENCE DETECTOR

1010 non-Overlapping Moore Sequence Detector



1010 Non-Overlapping Moore Sequence Detector

Verilog Code

```
module seq_detector_1010(input bit clk, rst_n, x, output reg z);
  parameter A = 4'h1;
  parameter B = 4'h2;
  parameter C = 4'h3;
  parameter D = 4'h4;
  parameter E = 4'h5; // extra state when compared with Mealy Machine
```

```
 bit [3:0] state, next_state;
 always @(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    state <= A;
  end
  else state <= next_state;
end

 always @(state or x) begin
  case(state)
    A: begin
         if(x == 0) next_state = A;
         else       next_state = B;
       end
    B: begin
         if(x == 0) next_state = C;
         else       next_state = B;
       end
    C: begin
         if(x == 0) next_state = A;
         else       next_state = D;
       end
    D: begin
         if(x == 0) next_state = E;
         else       next_state = B;
       end
    E: begin
         if(x == 0) next_state = A;
         else       next_state = B; //This state only differs when compared with Moore Overlaping
Machine
```

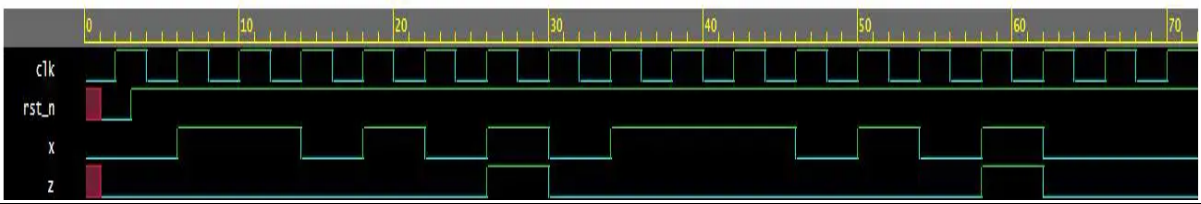
```
        end
      default: next_state = A;
    endcase
  end
  //As output z is only depends on present state
  always@(state) begin
    case(state)
      A : z = 0;
      B : z = 0;
      C : z = 0;
      D : z = 0;
      E : z = 1;
      default : z = 0;
    endcase
  end
endmodule
```

Testbench Code

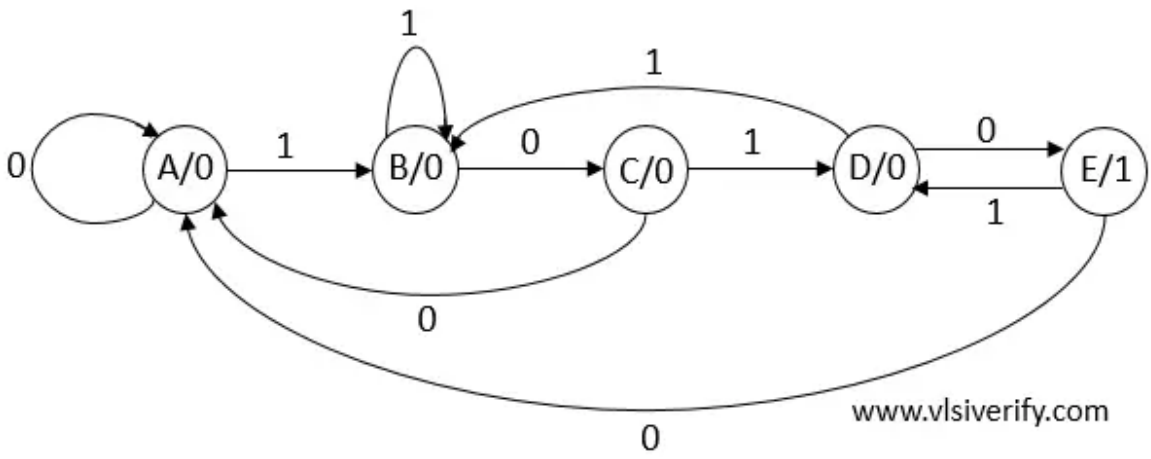
```
module TB;
  reg clk, rst_n, x;
  wire z;

  seq_detector_1010 sd(clk, rst_n, x, z);
  initial clk = 0;
  always #2 clk = ~clk;

  initial begin
    x = 0;
    #1 rst_n = 0;
    #2 rst_n = 1;
```

```
    #3 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #10;
    $finish;
  end

  initial begin
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end
endmodule
```

Output



1010 Overlapping Moore Sequence Detector Verilog Code



1010 Overlapping Moore Sequence Detector

Verilog code

```
module seq_detector_1010(input bit clk, rst_n, x, output reg z);
  parameter A = 4'h1;
  parameter B = 4'h2;
  parameter C = 4'h3;
  parameter D = 4'h4;
  parameter E = 4'h5; // extra state when compared with Mealy Machine

  bit [3:0] state, next_state;
  always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
      state <= A;
    end
    else state <= next_state;
  end

  always @(state or x) begin
    case(state)
      A: begin
```

```
        if(x == 0) next_state = A;
        else       next_state = B;
      end
    B: begin
        if(x == 0) next_state = C;
        else       next_state = B;
      end
    C: begin
        if(x == 0) next_state = A;
        else       next_state = D;
      end
    D: begin
        if(x == 0) next_state = E;
        else       next_state = B;
      end
    E: begin
        if(x == 0) next_state = A;
        else       next_state = D;
      end
    default: next_state = A;
  endcase
end
//As output z is only depends on present state
always@(state) begin
  case(state)
    A : z = 0;
    B : z = 0;
    C : z = 0;
    D : z = 0;
    E : z = 1;
```

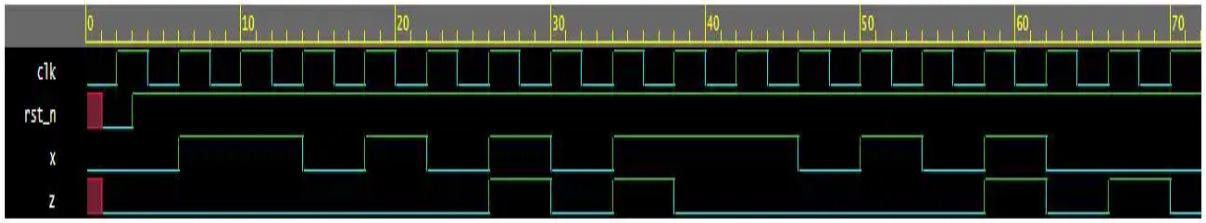
```
      default : z = 0;
    endcase
  end
endmodule
```

Testbench Code

```
module TB;
  reg clk, rst_n, x;
  wire z;

  seq_detector_1010 sd(clk, rst_n, x, z);
  initial clk = 0;
  always #2 clk = ~clk;

  initial begin
    x = 0;
    #1 rst_n = 0;
    #2 rst_n = 1;

    #3 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 1;
    #4 x = 1;
    #4 x = 0;
```

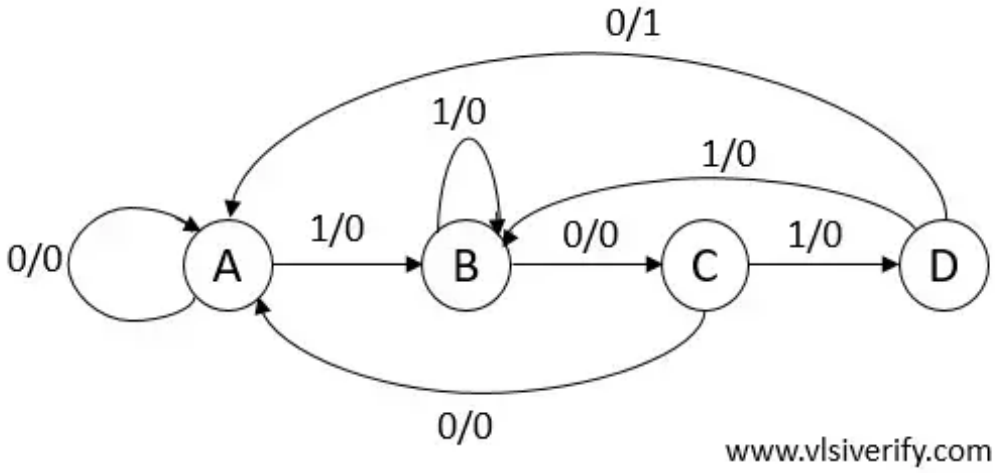
```
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #10;
    $finish;
  end

  initial begin
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end
endmodule
```

Output



MEALY SEQUENCE DETECTOR

1010 non-Overlapping Mealy Sequence Detector



1010 Non-Overlapping Mealy Sequence Detector

Verilog Code

```
module seq_detector_1010(input bit clk, rst_n, x, output z);
  parameter A = 4'h1;
  parameter B = 4'h2;
  parameter C = 4'h3;
  parameter D = 4'h4;

  bit [3:0] state, next_state;
  always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
      state <= A;
    end
    else state <= next_state;
  end

  always @(state or x) begin
    case(state)
      A: begin
          if(x == 0) next_state = A;
```

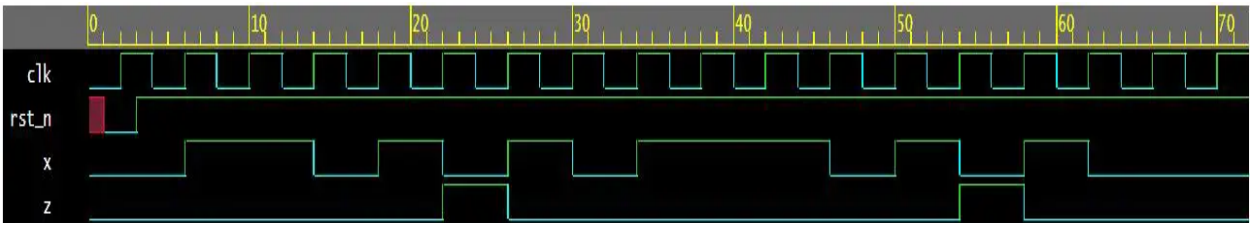
```
        else       next_state = B;
      end
    B: begin
        if(x == 0) next_state = C;
        else       next_state = B;
      end
    C: begin
        if(x == 0) next_state = A;
        else       next_state = D;
      end
    D: begin
      if(x == 0) next_state = A; //This state only differs when compared with Mealy Overlaping Machine
        else       next_state = B;
      end
    default: next_state = A;
  endcase
 end
 assign z = (state == D) && (x == 0)? 1:0;
endmodule
```

Testbench Code

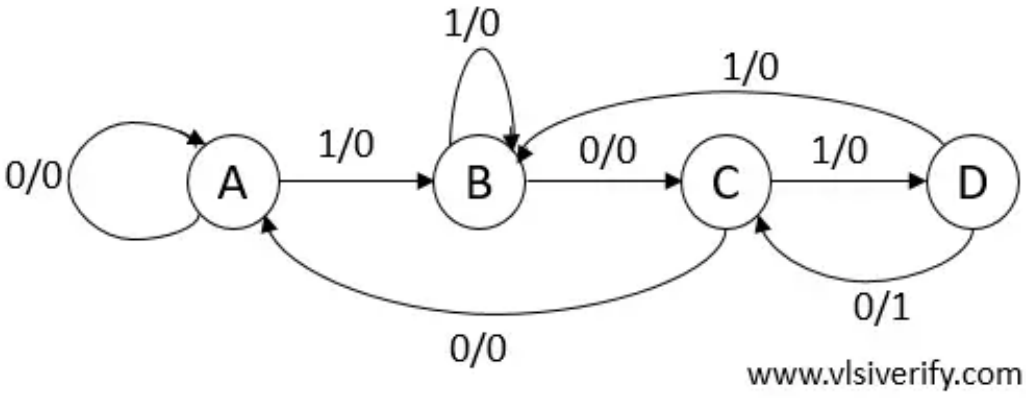
```
module TB;
 reg clk, rst_n, x;
 wire z;

 seq_detector_1010 sd(clk, rst_n, x, z);
 initial clk = 0;
 always #2 clk = ~clk;

 initial begin
  x = 0;
```

```
    #1 rst_n = 0;
    #2 rst_n = 1;

    #3 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 1;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #4 x = 1;
    #4 x = 0;
    #10;
    $finish;
  end

  initial begin
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end
endmodule
```

Output



1010 Overlapping Mealy Sequence Detector



1010 Overlapping Mealy Sequence Detector

Verilog Code

```
module seq_detector_1010(input bit clk, rst_n, x, output z);
  parameter A = 4'h1;
  parameter B = 4'h2;
  parameter C = 4'h3;
  parameter D = 4'h4;

  bit [3:0] state, next_state;
  always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
      state <= A;
    end
    else state <= next_state;
  end

  always @(state or x) begin
```

```
    case(state)
      A: begin
           if(x == 0) next_state = A;
           else       next_state = B;
         end
      B: begin
           if(x == 0) next_state = C;
           else       next_state = B;
         end
      C: begin
           if(x == 0) next_state = A;
           else       next_state = D;
         end
      D: begin
           if(x == 0) next_state = C;
           else       next_state = B;
         end
      default: next_state = A;
    endcase
  end
  assign z = (state == D) && (x == 0)? 1:0;
endmodule
```

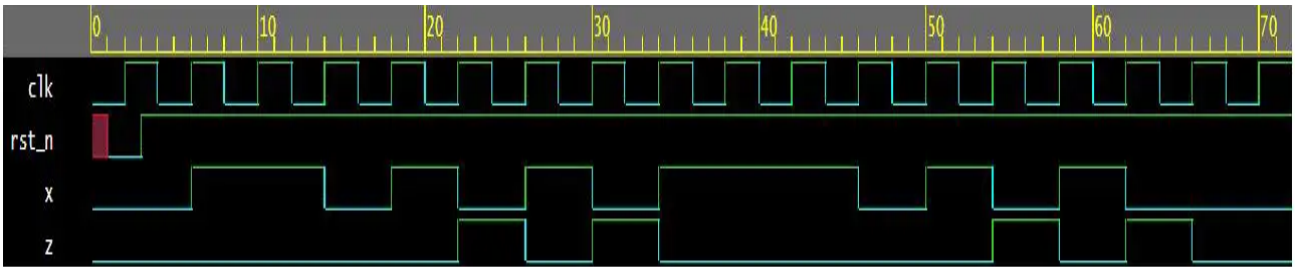
Testbench Code

```
module TB;
  reg clk, rst_n, x;
  wire z;

  seq_detector_1010 sd(clk, rst_n, x, z);
  initial clk = 0;
  always #2 clk = ~clk;
```

```
initial begin
  x = 0;
  #1 rst_n = 0;
  #2 rst_n = 1;

  #3 x = 1;
  #4 x = 1;
  #4 x = 0;
  #4 x = 1;
  #4 x = 0;
  #4 x = 1;
  #4 x = 0;
  #4 x = 1;
  #4 x = 1;
  #4 x = 1;
  #4 x = 0;
  #4 x = 1;
  #4 x = 0;
  #4 x = 1;
  #4 x = 0;
  #10;
  $finish;
end

initial begin
  // Dump waves
  $dumpfile("dump.vcd");
  $dumpvars(0);
end
```

endmodule

Output



Advantages of Sequence Detectors

Pattern Recognition: Sequence detectors are excellent at identifying predefined patterns or sequences in a data stream. This makes them useful in situations where data must be checked for specific conditions (e.g., validating communication protocols or data formats).

Reliability: Sequence detectors can provide reliable detection of specific sequences, especially in noisy or uncertain environments where it's difficult to manually track patterns.

Efficient State Management: Sequence detectors can be implemented using finite state machines (FSMs), which are highly efficient in managing and transitioning between different states based on input sequences. FSMs are deterministic, meaning they always produce predictable results, which is crucial in safety-critical systems.

Low Latency: Once designed and deployed, sequence detectors can often operate with very low latency, meaning they can detect sequences and respond in real-time, which is beneficial for high-speed applications like data processing, communication systems, or control systems.

Versatility: Sequence detectors can be used in a wide variety of applications, ranging from error detection to event monitoring, making them a versatile component in many digital systems.

Customizable: Sequence detectors can be designed for any specific sequence of interest. This flexibility makes them useful in applications requiring highly customized pattern recognition.

Error Detection: In communication systems, sequence detectors are often used to identify and correct errors, such as those caused by bit errors in transmission. This is achieved by detecting invalid or incorrect sequences of bits.

Applications of Sequence Detectors

Digital Communication Systems: In data communication, sequence detectors are used to identify specific sequences of bits (e.g., start-of-frame or end-of-frame markers) for synchronization and framing purposes. They help ensure data integrity and are used in protocols for error detection and correction.

Pattern Matching in Embedded Systems: In embedded systems, such as microcontrollers or FPGA-based systems, sequence detectors are used to recognize patterns in sensor data, control inputs, or even communication signals between devices.

Control Systems: Sequence detectors are useful in control systems where specific sequences of events or states trigger actions. For example, they can be used to monitor a sequence of machine states to ensure that a process runs in the correct order.

Protocol Checking in Communication: In communication protocols like UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), or I2C, sequence detectors can be employed to verify that the transmitted signals adhere to the correct format or protocol, ensuring data integrity.

Data Compression: Sequence detectors can be part of algorithms used for data compression, where they detect repeating patterns or sequences in data that can be compressed into smaller representations (e.g., in lossless data compression algorithms).

Finite State Machines (FSM) in Digital Systems: Sequence detectors are often implemented as FSMs, where each state in the FSM corresponds to a stage in the recognition of a sequence. This is commonly used in systems like vending machines, digital counters, or simple calculators where certain input sequences need to trigger specific actions.

Security Systems: In security applications, sequence detectors can be used for detecting patterns that represent unauthorized access, like detecting sequences of failed login attempts or unusual data access patterns.

Fault Detection in Systems: Sequence detectors can monitor for specific sequences of events in fault detection systems. For example, if a machine undergoes a series of errors or failures, a sequence detector can recognize the sequence and trigger an alert or shut down the system to prevent further damage.

Test and Debugging: In testing and debugging digital systems, sequence detectors can be used to ensure that specific sequences of operations are followed, particularly when simulating or emulating hardware behavior during testing.

Video and Image Processing: In digital video and image processing, sequence detectors can help detect and process frames, patterns, or sequences of image features to track objects, motion, or specific visual patterns.

CONCLUSION

Sequence detectors are fundamental components in many digital systems for recognizing specific patterns of data. Their versatility, efficiency, and ease of integration make them ideal for applications ranging from communication and control systems to security and pattern recognition tasks. Whether in simple digital circuits or complex embedded systems, sequence detectors play a vital role in ensuring data integrity, control flow, and operational correctness**.**

In this project, we successfully designed and implemented a sequence detector that can recognize a specific sequence of bits within a given input stream. The primary objective was to develop a digital system capable of detecting patterns in real-time data using a state machine

approach. Through the use of finite state machines (FSMs), we were able to model the behavior of the detector, ensuring its functionality across different input scenarios.

Key highlights and conclusions from the project include:

Accurate Pattern Recognition: The sequence detector was able to correctly identify a predefined sequence, such as "1010" or any other sequence specified during the design phase. The FSM transitions were carefully mapped to ensure that the detector responded correctly to every valid and invalid input combination.

Hardware and Software Integration: The project demonstrated how digital logic can be implemented both in hardware (using flip-flops, logic gates) and in software (using simulation tools like VHDL or Verilog). The implementation in FPGA or ASIC allowed for real-time sequence detection in a compact, efficient manner.

Performance and Efficiency: The sequence detector operated efficiently, with minimal delay between input recognition and output generation. The use of a minimal number of states ensured low complexity, contributing to faster processing speeds and reduced resource usage.

In conclusion, the sequence detector project successfully demonstrated the principles of state machine design and digital logic in a practical setting. The system achieved the goal of recognizing specific sequences of bits with accuracy and efficiency, providing a foundation for further exploration into more complex sequence detection systems and applications in embedded systems and digital signal processing.