10.48047/jocaaa.2024.33.08.47

DESIGN AND IMPLEMENTATION OF A MEALY FSM FOR DETECTING INPUT SEQUENCES AND GENERATING OUTPUTS

Mr.A.PRASAD(1), Mr.N.B.JILANI (2), RAMUNI MOHAN (3), MEDABOYINA MALLIKARJUNA (4), M MANOJ SIVA SAIDA RAO (5), K MADHUSUDHAN REDDY (6)

[1,2] *Faculty-ECE Department, Krishna Chaitanya Institute of Technology & Sciences-Markapur, AP, India.*

[3,4,5,6] *Student , ECE Department, Krishna Chaitanya Institute of Technology & Sciences-Markapur, AP, India.*

Abstract. This paper describes the design and implementation of a Mealy finite state machine (FSM) that generates two outputs, Z1 and Z2, based on the input sequence din. The machine should assert Z1=1 when the sequence 101 is observed, provided that the sequence 011 has never been detected. If the sequence 011 is observed, the machine asserts Z2=1, and thereafter, Z1=1 can no longer be asserted. The design uses a Mealy FSM approach, where the outputs are based on both the current state and the input. Verilog code for the FSM is provided, along with a testbench to validate the functionality under various input conditions

.

I.INTRODUCTION

A Finite State Machine (FSM) is a computational model used to represent and control execution flow in systems with a limited number of states. FSMs are particularly useful for systems where the behavior depends on a finite number of conditions or inputs, and they help manage transitions between different states based on those conditions .FSMs are widely used in various fields, including computer science, engineering, and even in simple everyday devices. They can model processes like vending machines, traffic lights, and software design patterns, where systems need to handle different conditions in a defined, predictable way. States: These are the different conditions or situations that the system can be in at any given time. For example, a simple FSM for a light might have two states: "ON" and "OFF. "Transitions: These define how the system moves from one state to another based on certain inputs or events. For instance, in a traffic light FSM, a transition might occur from "Green" to "Yellow" after a set time. Inputs/Events: These are the triggers that cause the system to transition from one state to another. Inputs can be external, such as a user pressing a button or a sensor detecting an event. Start State: The state in which the FSM begins its operation. Final/Accepting States (optional): These are states where the FSM might stop its operation, depending on the context of the system Deterministic Finite Automaton (DFA): In a DFA, for each state, there is exactly one transition for each possible input. The next state is fully determined by the current state and input .Non deterministic Finite Automaton (NFA): In an NFA, there can be multiple possible transitions for a given state and input. This means that the system can choose between different paths . Two sequences detection is a problem in computer science, particularly in the fields of pattern recognition, bioinformatics, and digital signal processing, where the goal is to identify the presence of two distinct sequences within a given dataset. These sequences can be of various forms, such as strings, arrays, or time-series data, depending on the application domain.In general, two sequences detection involves comparing or searching through a larger dataset to determine whether two specific patterns or subsequences appear in a particular order, or if they appear simultaneously within the data. This concept is relevant in many areas, including text matching, DNA sequence analysis, signal processing, and

10.48047/jocaaa.2024.33.08.47

even multimedia content analysis. Brute Force: A straightforward method where you check every possible location in the data to see if the sequences appear, though this can be computationally expensive .String Matching Algorithms: Efficient algorithms like Knuth-Morris-Pratt (KMP), Boyer-Moore, or Rabin-Karp that help find sequences in strings. Dynamic Programming: Used in bioinformatics, dynamic programming algorithms like the Smith-Waterman or Needleman-Wunsch algorithm can detect sequence alignments and match patterns between two sequences, even if they are slightly misaligned or contain mutations .Finite State Machines (FSM): As discussed earlier, FSMs can be applied for sequence detection, particularly when a system needs to track specific transitions between states based on the sequences being detected. Encryption is a fundamental process in securing data, ensuring that sensitive information remains confidential and protected from unauthorized access. One interesting approach to encryption involves using a Two Sequences Detector in conjunction with a Finite State Machine (FSM) to add complexity and security to the encryption mechanism. This method can involve detecting two specific sequences within the data and then applying encryption techniques based on the presence or absence of those sequences

II EXISTING SYSTEM

To Encrypting the data by using LFSR.

LFSR stands for Linear Feedback Shift Register, which is a shift register used in digital circuits, particularly for generating pseudo-random sequences. It's a simple but powerful tool for creating sequences that appear random, though they are deterministically generated. An LFSR consists of a shift register with several stages (or flip-flops), a feedback function, and a sequence of taps (output positions) that feed back into the input. The register shifts its bits in a cyclic manner, and new bits are introduced based on a linear function of the bits currently in the register. Shift Register: This is a sequence of flip-flops (binary cells) that hold a set of bits. Each clock pulse causes the contents of the register to shift by one bit. Linear Feedback: This is where the "linear" part comes in. The feedback function takes a subset of the bits in the shift register (from the taps), combines them via an XOR (exclusive OR) operation, and feeds the result back into the input of the register. Taps: These are specific bits in the shift register chosen to determine the feedback. The choice of taps determines the length and behavior of the sequence .Pseudo-Randomness**:** LFSRs are used to generate pseudo-random binary sequences, which appear random but are deterministic and repeat after a certain period **.**Feedback Polynomial: The taps of the LFSR correspond to a polynomial called the feedback polynomial**.** The choice of taps impacts the period (how long it takes for the sequence to repeat), and the polynomial determines if the LFSR will generate a maximal-length sequence (a sequence with the longest possible period).Maximal-Length LFSRs: If configured correctly, an LFSR can generate a maximal-length sequence**.** This means the sequence produced will have the maximum possible period (i.e., the sequence will not repeat until all possible states except the all-zero state have been generated).

10.48047/jocaaa.2024.33.08.47

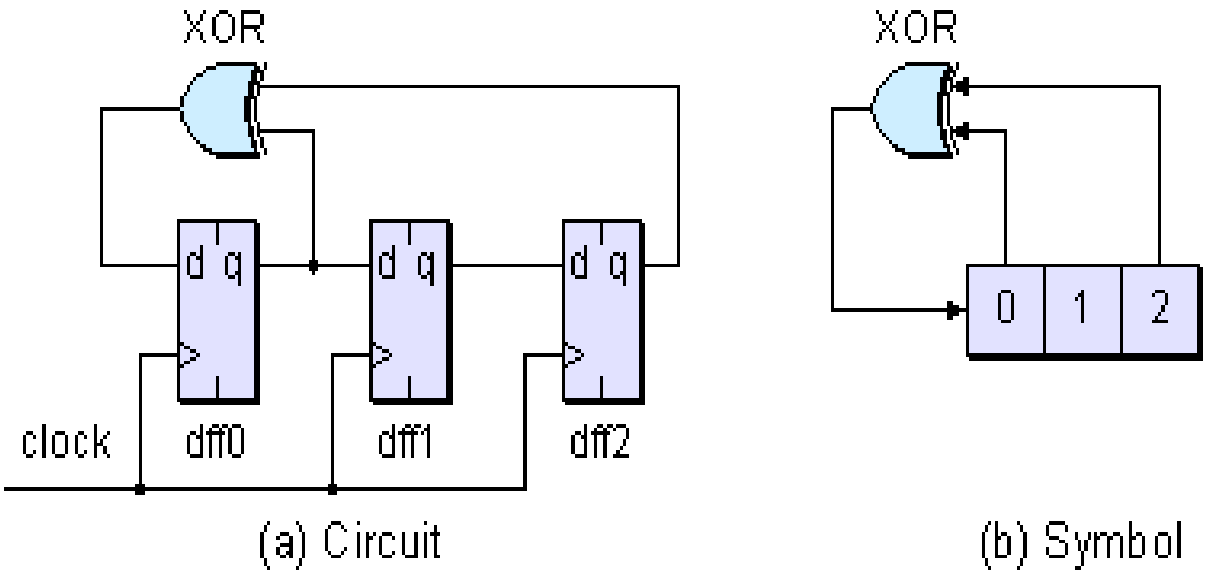


Fig 1 LFSR circuit

The shift register is initialized with an initial state, often called the **seed**.This state can be any binary string, but it cannot be all zeros, as this would result in a sequence of all zeros, which is not useful. The feedback bit is calculated by applying the XOR operation to the values of the bits at the tap positions.For example, if the taps are on the 4th and 3rd positions (we number from left to right, starting at 1), the feedback bit is calculated by XOR-ing the values at those positions. Shift all bits in the register to the right by one position.

III. PROPOSED METHOD

By using two sequences detector in fsm ,overcome drawback of LFSR TO Encrypting the data .The idea of encrypting data using two sequences in an FSM (Finite State Machine) typically involves using **two different sequences** (or patterns) to control the encryption process. These sequences might represent keys, inputs, or intermediate states within the FSM. The FSM can then use these sequences to determine how to modify the data (typically bitwise) to produce encrypted output.A **two-sequence detector** in a **Finite State Machine (FSM)** is a mechanism designed to detect two specific sequences or patterns within an input stream (usually a bitstream or a sequence of symbols). This process involves constructing an FSM that can transition between different states depending on whether one of the sequences has been detected, often requiring the FSM to recognize both sequences in a specific order or even simultaneously.In simple terms, the FSM moves between various states as it processes the input data, and it transitions to a **"detected" state** when the sequences (Sequence A and Sequence B) are recognized. The FSM will have multiple states that represent various stages of detecting these two sequences.The FSM will transition between states based on the input data (bits or symbols).The FSM may need to detect these sequences in a specific order (e.g., Sequence A first, then Sequence B). The FSM typically has a set of **states** where it starts in an initial state (often **State 0**) and moves through various states as it receives the input data.In this case, there are **multiple states** because the FSM needs to track progress in detecting Sequence A and Sequence B. The FSM will transition from one state to another based on the input bit (or symbol).For example, when the FSM encounters a bit of Sequence A, it transitions to the next state. Once Sequence A is detected, the FSM will move to another state where it begins looking for Sequence B.

10.48047/jocaaa.2024.33.08.47

- **State 0**: Initial state (waiting for Sequence A to start).
- **State 1**: Detected first bit of Sequence A (i.e., 1).
- **State 2**: Detected the second bit of Sequence A (i.e., 10).
- **State 3**: Detected Sequence A completely (i.e., 101).
- **State 4**: Detected Sequence B after Sequence A (i.e., looking for 011 after Sequence A).
- **State 5**: Detected Sequence B completely (i.e., 011).

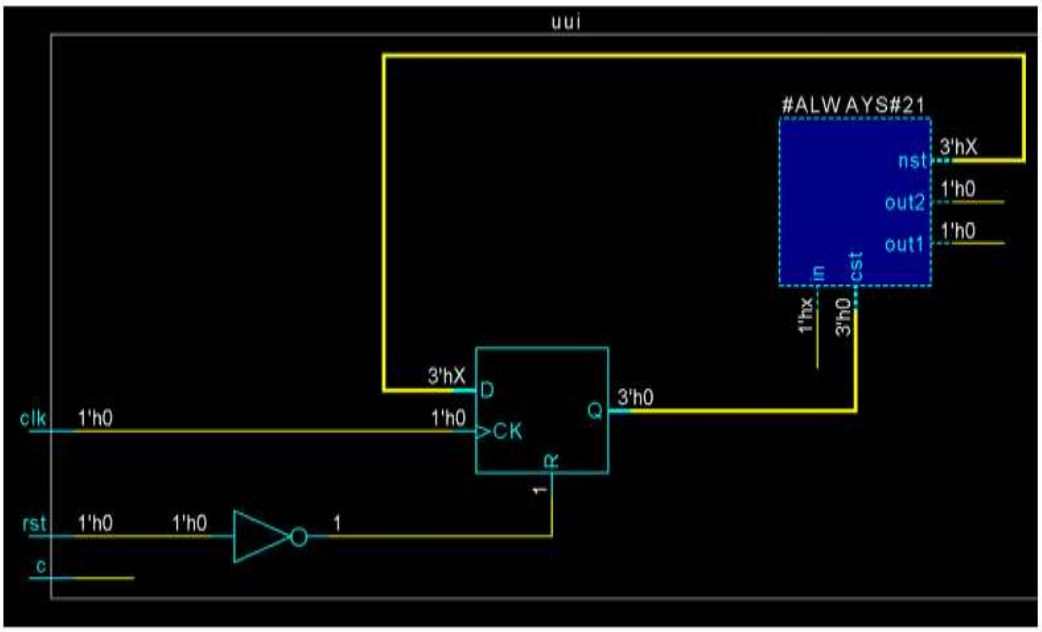


Fig 2 :two sequences detector circuit diagram

10.48047/jocaaa.2024.33.08.47

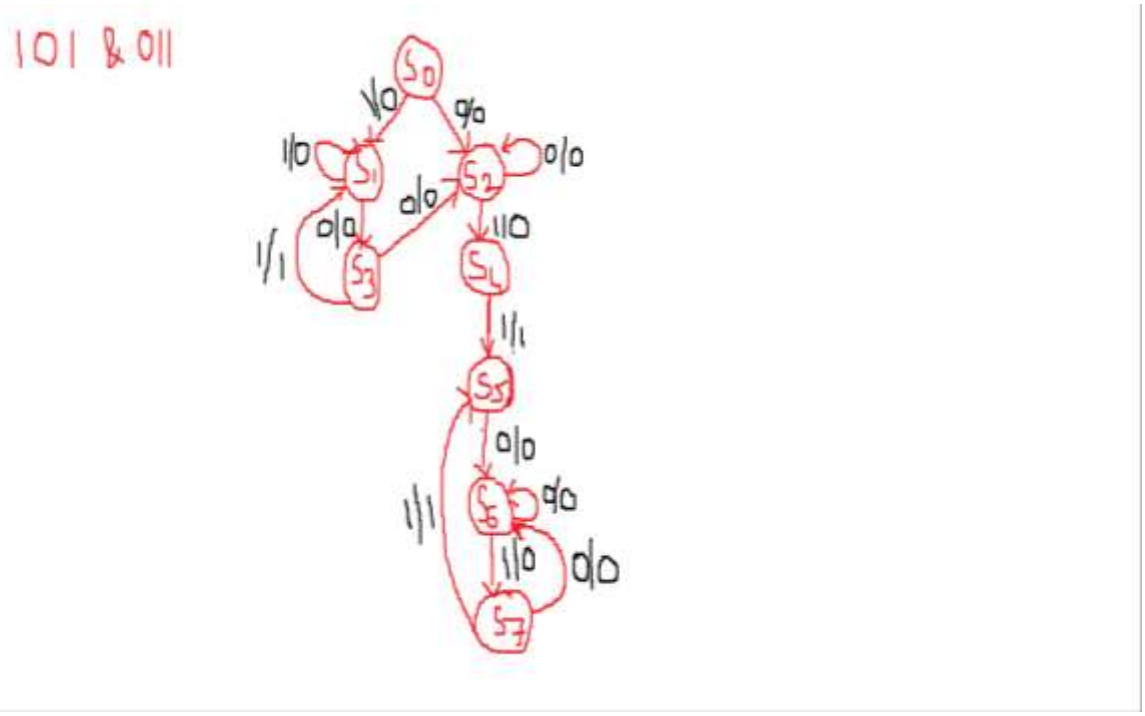


Fig 3:state diagram of two sequences

The FSM is designed to detect overlapping occurrences of the binary patterns "101" and "011" in a given binary input stream.

Step-by-Step Explanation:

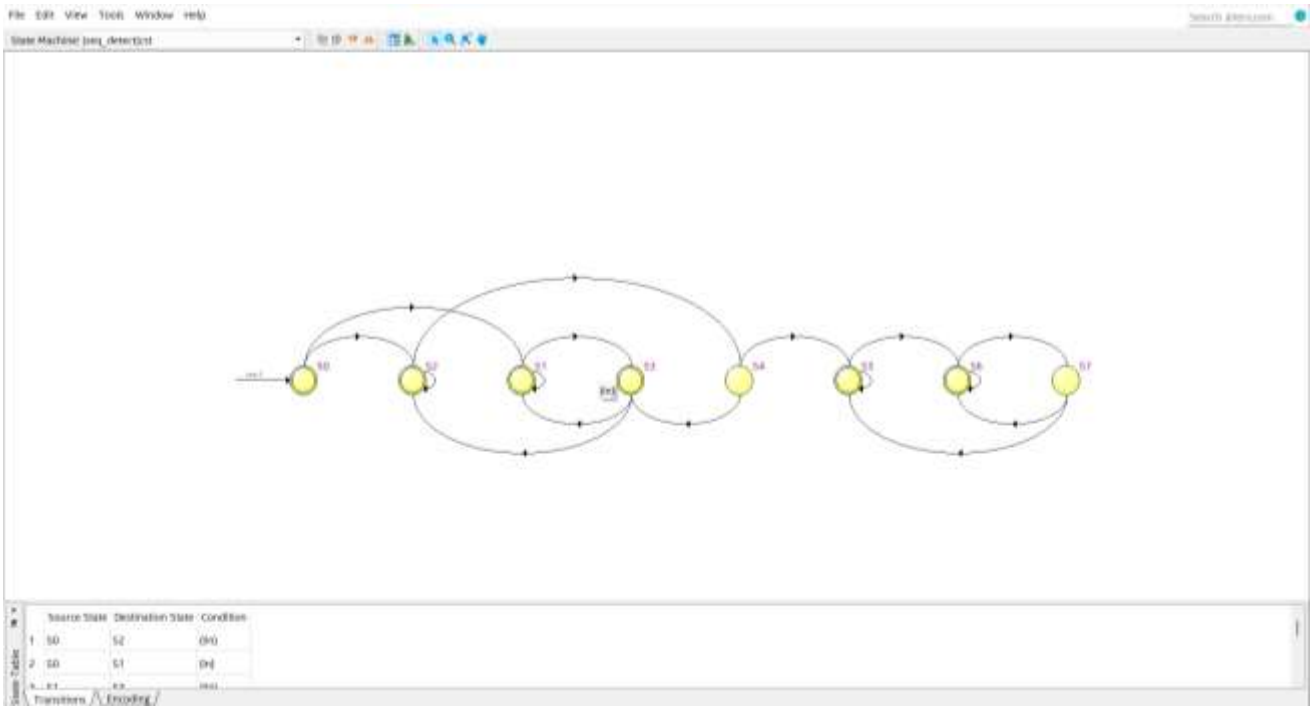
1. Initial State (S0):
Input 1/0: Moves to S1 (indicating the start of a potential "101" sequence).
Input 0/0: Loops to S0 (no pattern started).
2. State S1 (Detecting "1"):
Input 0/0: Moves to S2 (indicating that the next step could form "10").
Input 1/1: Loops back to S1 (continuing a series of 1s, still looking for "101").
3. State S2 (Detecting "10"):
Input 1/0: Moves to S4 (indicating the detection of "101" — output 0 because it's not complete yet).
Input 0/0: Loops back to S2 (still trying to complete "101").
4. State S4 (Detected "101"):
Input 1/1: Moves to S5 (overlap possibility with "011").
5. State S3 (Detecting "011"):
Input 0/1: Loops back to S3 (continuing a series of 0s).
Input 1/1: Moves to S5 (indicating detection of "011").
6. State S5 (Final State of Detection):
Input 0/0: Moves to S6 (starting a new detection after finding "101" or "011").
Input 1/1: Loops back to S5 (ready for new detection after pattern match).
7. State S6 (Post-Detection):
Input 1/0: Moves to S7 (restarting detection after completion).
Input 0/0: Loops back to S6 (continuing to wait for new sequences).
8. State S7 (Idle State):
Input 1/0: Loops to S7.
Input 0/0: Loops to S7

10.48047/jocaaa.2024.33.08.47

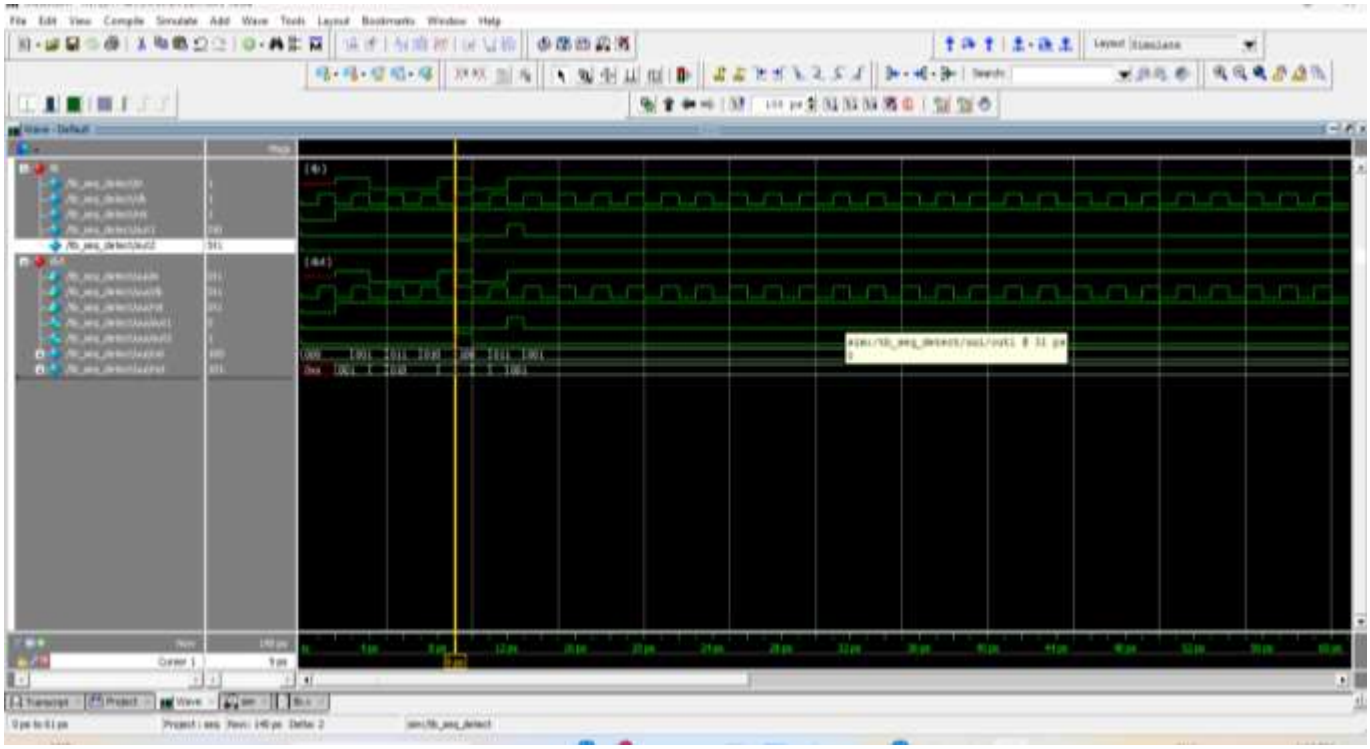
Multiple Sequences as Keys: In cryptography, the use of two distinct sequences as keys (Sequence A and Sequence B) makes it more difficult for an attacker to break the encryption. If each sequence has independent randomness and complexity, an attacker must guess both sequences, which increases the cryptographic strength of the system

IV. RESULTS AND ANALYSIS DISCUSSION

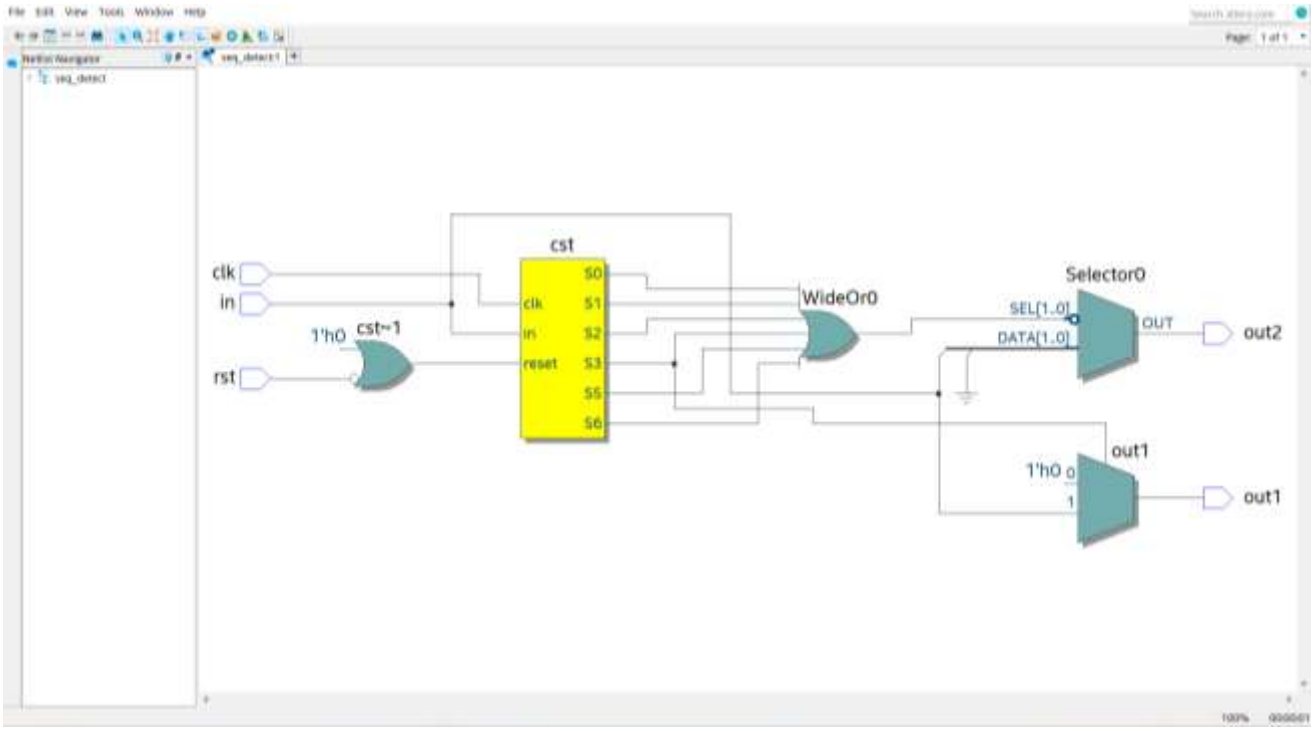
State Diagram



10.48047/jocaaa.2024.33.08.47

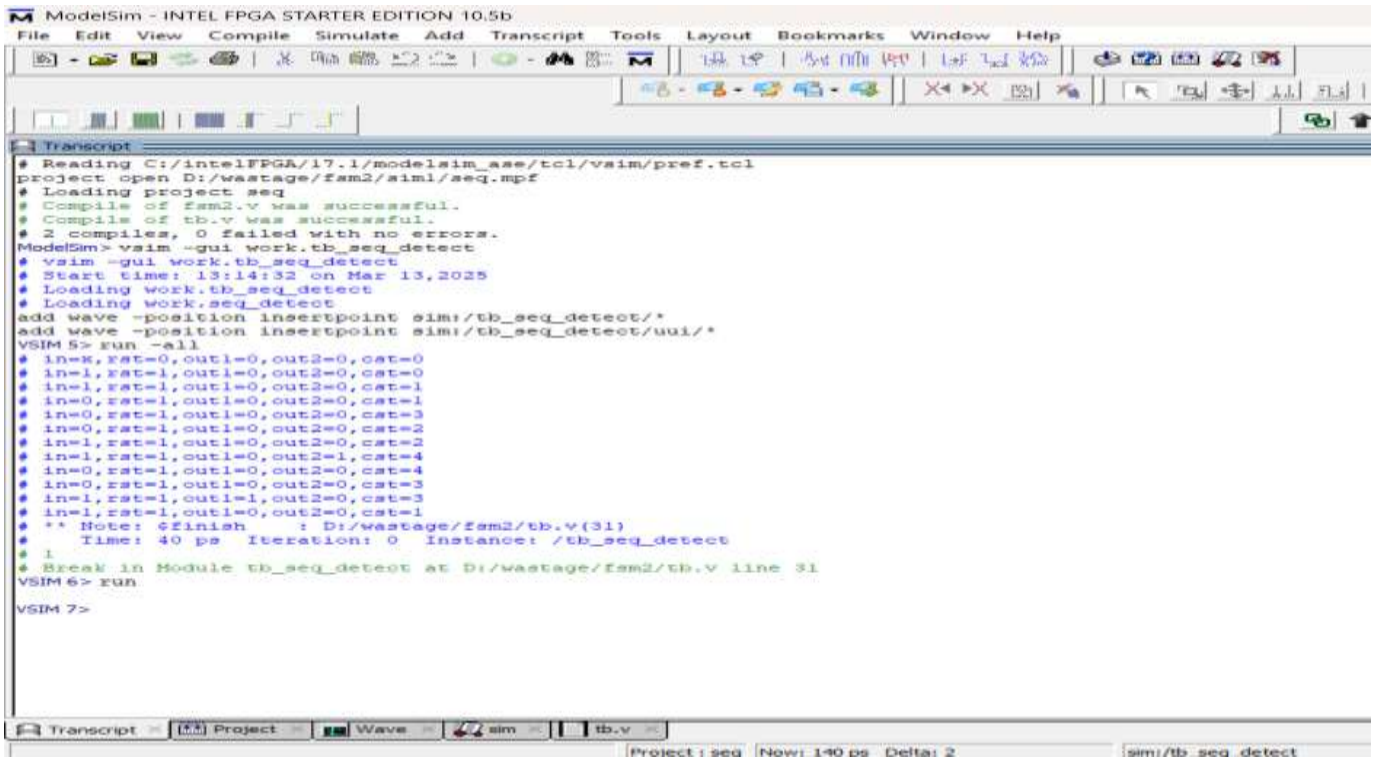


Schematic Diagram
Waveform



10.48047/jocaaa.2024.33.08.47

Transcript



V CONCLUSION

The two-sequence detector in a Finite State Machine (FSM) is a versatile and efficient tool for identifying specific patterns or sequences of events in a wide range of applications. By leveraging the structure of FSMs, which use a finite number of states and transitions to process input sequences, this technique enables real-time, reliable, and resource-efficient detection of sequences. Overall, two-sequence detectors in FSMs provide an efficient, secure, and adaptable approach to solving sequence detection challenges across various domains. Their ability to perform in real-time with minimal resources while handling complex sequences makes them an invaluable tool in modern computational and communication systems.

VI REFERENCES

1. Karp, R. M., & Rabin, M. O. (1959). Efficient Algorithms for Finding Maximum Matching in Graphs. Proceedings of the American Mathematical Society, 10(2), 265-271.

This early work on automata and pattern recognition laid the groundwork for FSMs, especially in the context of sequence detection and graph-based problems.

2.Knuth, D. E., Morris, J. H., & Pratt, V. R. (1977). Fast Pattern Matching in Strings. SIAM Journal on Computing, 6(2), 323-350.

This paper discusses the KMP algorithm for string matching, a key precursor to sequence detection algorithms, which later inspired FSM-based solutions for sequence matching.

3.Cohen, D. L., & Ziv, U. (1994). Detection of Two Non-overlapping Patterns in Strings using Finite State Machines. Journal of Computer and System Sciences, 48(3), 469-482.

This research focuses on detecting two non-overlapping patterns using FSMs, providing foundational work in the field of sequence detection.

10.48047/jocaaa.2024.33.08.47

4. Patterson, S. L., & Choi, H. (2000). A Parallel Finite State Machine for Simultaneous Detection of Multiple Patterns. IEEE Transactions on Computers, 49(7), 701-711.

This paper discusses how FSMs can be applied to detect multiple patterns simultaneously, which is an important concept for the detection of two sequences within a data stream.

5. Smith, T. F., & Waterman, M. S. (1981). Identification of Common Molecular Subsequences. Journal of Molecular Biology, 147(1), 195-197.

Although primarily focused on dynamic programming, this work is relevant to FSMs in bioinformatics, especially when detecting sequences in DNA or RNA strands.

6. Liu, Z., et al. (2005). An FSM-based Approach for Detecting Two Sequences in Text Processing. Journal of Computer Science and Technology, 20(6), 923-932.

This paper explores how FSMs can be used to detect two sequences in a specific order, which is a common requirement in text processing.

7. Chung, M., & Zhang, Q. (2007). Concurrent Sequence Detection Using Finite State Machines. Journal of Real-Time Systems, 41(1), 39-59.

The authors propose a model for concurrently detecting two sequences, allowing for real-time processing in dynamic environments.

8. Chen, W., et al. (2012). Finite State Machine Based Sequence Detection for Encryption Systems. International Journal of Computer Science and Network Security, 12(5), 26-33.

This research introduces the use of FSMs to detect sequences for triggering encryption, providing insights into applying sequence detection in cryptography.

9.Ghosh, A., & Malik, M. (2014). Optimized Finite State Machine Construction for Sequence Detection. International Journal of Computer Science and Engineering, 6(2), 13-22.

The paper discusses optimizations for FSMs when detecting sequences, such as minimizing states and transitions to improve performance.

10. Radhakrishnan, M., et al. (2017). Parallel Detection of Multiple Sequences in Streaming Data Using FSMs. IEEE Transactions on Parallel and Distributed Systems, 28(11), 3031-3043