DAYANANDA SAGAR COLLEGE OF ARTS, SCIENCE AND COMMERCE

Shavige Malleshwara Hills, 1st Stage, Kumaraswamy Layout, Bengaluru, Karnataka

Department of Computer Application-MCA(BU)

Topic: Modeling Digital Components Using Finite State Machines (FSMs)

Submitted To:

Miss Akshatha

Assistant Prof.

DSCASC

Submitted By:

Chandana S (P03CJ24S126027)

Abhishwaran R (P03CJ24S126009)

Keerthana Y (P03CJ24S126050)

Hemalatha B S (P03CJ24S126040)

Problem Statement

The goal of this project is to explore the application of the Theory of Computation (TOC) in digital electronics by designing and simulating basic digital circuits using Finite State Machines (FSMs). Specifically, we will model the behavior of three essential digital components:

- Shift Registers
- Synchronous Counters
- Multiplexers

Each component will be simulated with their logic defined as state transitions, providing a deeper understanding of how FSM principles underpin digital system design.

Explanation of the Problem

FSMs are a mathematical framework to describe how a system behaves based on a sequence of inputs and current states. In digital circuits, they are heavily used to define the behavior of:

- **Sequential Logic Circuits**
- **Control Units**
- **Signal Decoding and Selection Logic**

We will model:

- **Shift Registers**: Devices that shift data bits serially on each clock cycle.
- **Synchronous Counters**: Count sequential binary values on clock pulses.
- **Multiplexers**: Select a specific input based on selector control bits.

This report highlights the integration of TOC (FSM) into digital circuit behavior simulation.

Implementation

We use Python to simulate FSMs for each of the three components. Each simulation includes state tracking, input-based transitions, and output behavior.

FSM Design Summary:

Shift Register FSM

- States: Bit configurations in the register
- Inputs: Data bits, clock pulses
- Transitions: Right/left shifting based on clock

Synchronous Counter FSM

- States: Binary count values (000 to 111 for 3-bit)
- Input: Clock pulses
- Transitions: Increment to next state modulo 2^n

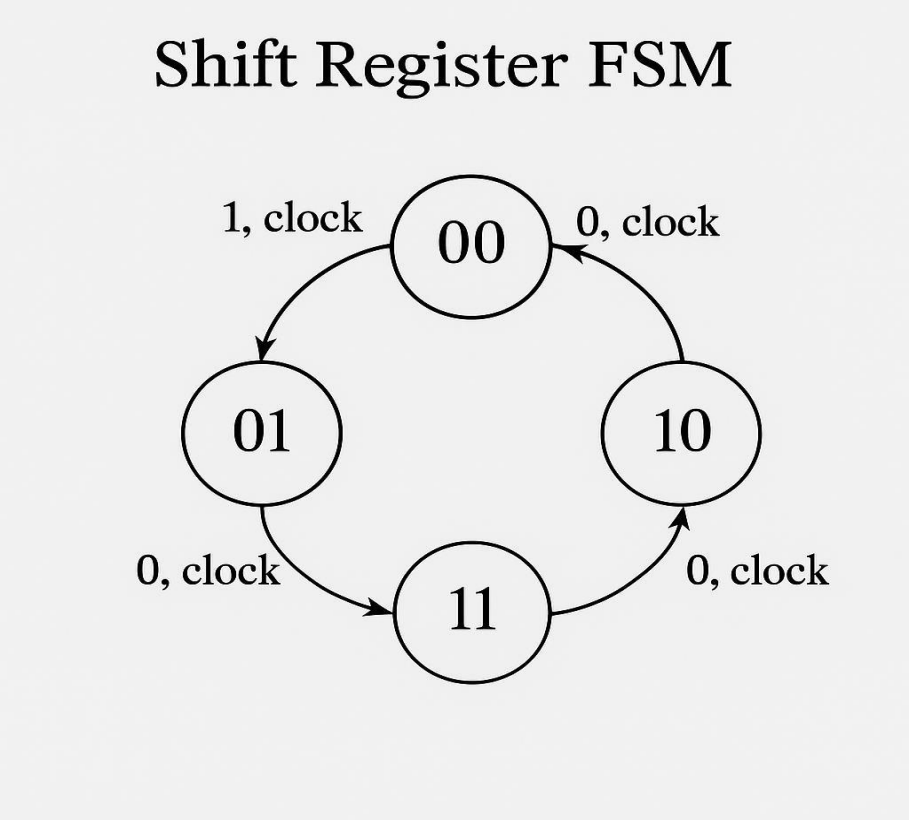
Multiplexer FSM

- States: Selector state (based on control bits)
- Input: Selector lines
- Output: Chosen data input

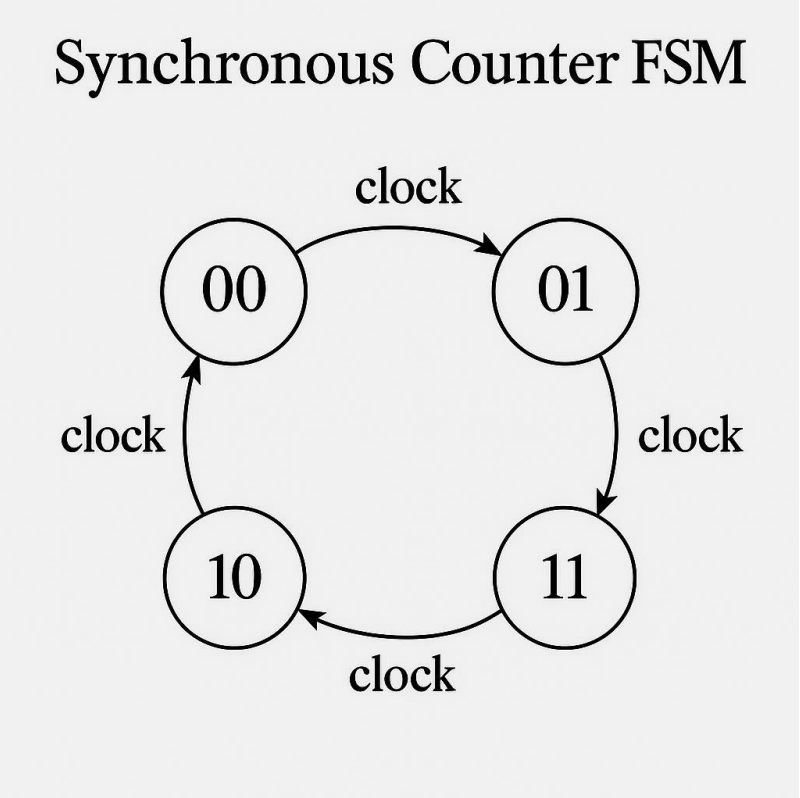
FSM Diagrams

For each component:

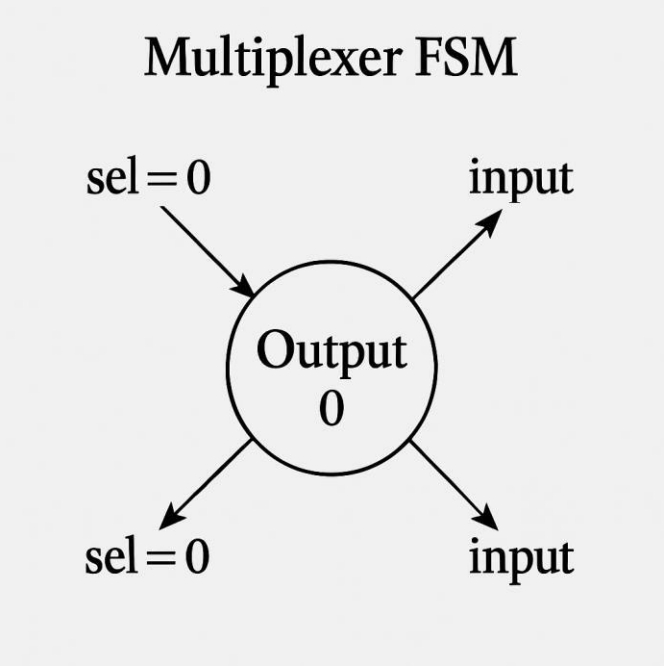
- State diagram
- Transition description
- Input/output mappings



Shift Register FSM



Synchronous counter FSM



Multiplexer FSM

Python Code

- **Shift Register (2-bit)**

```
class ShiftRegister:
def __init__(self):
self.register = [0, 0]  # 2-bit register

def shift_right(self, bit):
self.register = [bit] + self.register[:-1]
print("Register:", self.register)

# Example usage
sr = ShiftRegister()
sr.shift_right(1)
sr.shift_right(0)
sr.shift_right(1)
```

OUTPUT

```
Register: [1, 0]
Register: [0, 1]
Register: [1, 0]
```

- **Synchronous Counter (2-bit)**

```
class SynchronousCounter:
    def __init__(self):
        self.count = 0

    def clock_pulse(self):
        self.count =
    (self.count+ 1) % 4  # 2-bit
    max value = 3 (00 to 11)
        print("Count:",
    format(self.count, '02b'))
```

```
# Example usage
counter =
    SynchronousCounter()
for _ in range(10):
    counter.clock_pulse()
```

OUTPUT

```
Count: 01
Count: 10
Count: 11
Count: 00
Count: 01
Count: 10
Count: 11
Count: 00
Count: 01
Count: 10
```

4-to-1 Multiplexer

```
class Multiplexer:
    def __init__(self, inputs):
        self.inputs = inputs
    def select(self, sel):
        output = self.inputs[sel]
        print(f"Selected Input [{sel}]: {output}")
mux = Multiplexer([10, 20, 30, 40])
mux.select(2)

mux.select(0)
```

OUTPUT

Selected Input [2]: 30

Selected Input [0]: 10

Shift Register (2-bit)

```
Register: [1, 0]
Register: [0, 1]
Register: [1, 0]
```

Synchronous Counter (2-bit)

```
Count: 01
Count: 10
Count: 11
Count: 00
Count: 01
Count: 10
Count: 11
Count: 00
Count: 01
Count: 10
```

4-to-1 Multiplexer

```
Selected Input [2]: 30
Selected Input [0]: 10
```

Conclusion

This project showcases how FSMs—one of the foundational concepts in TOC—are essential in modeling and simulating digital circuit behavior. By converting logic into state machines, we successfully simulated shift registers, counters, and multiplexers.

This approach connects abstract computation theory with concrete hardware implementations, showing how FSMs drive automation, control, and logic in digital systems.