# **Vending Machine FSM**

## **🎯 Objective**

**Design and simulate a sequential digital circuit (FSM) for a real-world application: a vending machine that accepts money and dispenses a product based on the amount received. The design must also handle change return and implement using Finite State Machines (FSMs).**

## **🧠 What Is a Vending Machine?**

**A vending machine accepts money and provides goods automatically. In this lab, it's modeled using an FSM that:**

* **Tracks inserted money**
* **Decides when to dispense a product**
* **Calculates and returns any change**
* **Handles multiple denominations (0, 5, 10, 20 Taka)**

## **⚙️ Vending Machine as an FSM**

### **Components:**

* **Inputs: Amount inserted (e.g., 0, 5, 10, 20)**
* **Outputs:**
  + **Whether to dispense a product (Give Product? Yes/No)**
  + **How much change to return (0/5/10/15 Taka)**
* **States: Represent the current balance**
* **Transitions: Based on the new amount inserted**

## **🧪 Examplesb**

**There are three FSM cases, each with:**

* **A state diagram**
* **A state table**
* **A Verilog code snippet (not fully shown in the slides)**

### **🔹 Case 1: 10 Taka Product**

#### **📌 States:**

* **S0: 0 Taka inserted**
* **S1: 5 Taka inserted**

#### **📈 State Diagram:**

| **Input (Taka)** | **From** | **To** | **Output (Product, Change)** |
| --- | --- | --- | --- |
| **0** | **S0** | **S0** | **No product, 0 Taka** |
| **5** | **S0** | **S1** | **No product, 0 Taka** |
| **10** | **S0** | **S0** | **Give product, 0 Taka** |
| **20** | **S0** | **S0** | **Give product, 10 Taka** |
| **5** | **S1** | **S0** | **Give product, 0 Taka** |
| **10** | **S1** | **S0** | **Give product, 5 Taka** |
| **20** | **S1** | **S0** | **Give product, 15 Taka** |

#### **💡 Summary:**

* **Product is given if total >= 10**
* **Returns change accordingly**
* **After product is given, reset to S0**

### **🔹 Case 2: 15 Taka Product**

#### **📌 States:**

* **S0: 0 Taka**
* **S1: 5 Taka**
* **S2: 10 Taka**

#### **📈 State Diagram:**

| **Input** | **From** | **To** | **Output (Product, Change)** |
| --- | --- | --- | --- |
| **0** | **Any** | **Same** | **No product, refund part** |
| **5** | **S0** | **S1** | **No product, 0 Taka** |
| **5** | **S1** | **S2** | **No product, 0 Taka** |
| **5** | **S2** | **S0** | **Give product, 0 Taka** |
| **10** | **S0** | **S2** | **No product, 0 Taka** |
| **10** | **S2** | **S0** | **Give product, 5 Taka** |
| **20** | **S0** | **S0** | **Give product, 5 Taka** |
| **20** | **S2** | **S0** | **Give product, 15 Taka** |

#### **💡 Summary:**

* **Product requires 15 Taka**
* **Handles multiple transitions across 3 states**
* **Provides appropriate change and resets**

### **🔹 Case 3: 20 Taka Product**

**The diagram is shown but details are minimal in the slides. It likely follows similar logic as above, but with 4 or more states to accumulate up to 20 Taka before releasing a product and change.**

## **🧰 State Tables and Assigned Tables**

**Each case includes:**

* **State Table: Defines transitions and outputs based on inputs and current states**
* **State Assigned Table: Uses binary encoding for implementation in Verilog**

**Example snippet from the 10 Taka product case:**

**State S0:**

**Input: 5 → Next: S1, Output: 0 product, 0 change**

**Input: 10 → Next: S0, Output: 1 product, 0 change**

**Input: 20 → Next: S0, Output: 1 product, 10 change**

**State S1:**

**Input: 5 → Next: S0, Output: 1 product, 0 change**

**Input: 10 → Next: S0, Output: 1 product, 5 change**

## **🧑‍💻 FSM Implementation in Verilog**

**Though full code is not visible, the design likely uses:**

### **State Encoding (Binary):**

**parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10;**

### **Always Block:**

**always @(posedge clk or posedge reset) begin**

**if (reset)**

**state <= S0;**

**else**

**state <= next\_state;**

**end**

### **Next-State Logic:**

**case (state)**

**S0: if (money == 5) next\_state = S1;**

**...**

**S1: ...**

**endcase**

### **Output Logic:**

**assign product = (state == S1 && money == 5); // Example**

**assign change = ...**

## **🧠 Learning Outcomes**

**By completing this lab, students learn to:**

* **Convert a real-world application into an FSM**
* **Design and simulate state machines with multiple states**
* **Use state diagrams and tables to guide Verilog implementation**
* **Handle outputs based on current state and input (i.e., Mealy-style FSM)**

## **✅ Summary Table**

| **Feature** | **Details** |
| --- | --- |
| **Application** | **Vending machine FSM** |
| **FSM Type** | **Mostly Mealy FSM (output depends on state + input)** |
| **Input Values** | **0, 5, 10, 20 Taka** |
| **Product Price Cases** | **10, 15, and 20 Taka** |
| **Outputs** | **Product dispensing signal, Change amount** |
| **Design Artifacts** | **State diagrams, tables, assigned binary logic** |
| **Implementation** | **Verilog (FSM coding using always blocks)** |