ECEN 474 - Project 1 Report
Vending Machine FSM Simulation

Gael

May 28, 2025

Abstract

A vending machine is designed using VHDL and implemented as a finite state machine (FSM). The process involves defining the FSM structure, implementing state transitions based on coin inputs, and configuring output conditions for purchase and change return. The FSM is then tested and validated through simulation in Quartus using the University Waveform function to ensure correct functionality.

1 Overview

The project involved designing a vending machine controller using VHDL, implementing a finite state machine (FSM) to track coin inputs and determine when a purchase could be made. The system was tested using Quartus' University Waveform tool to validate correct state transitions and outputs.

The vending machine accepts nickels (5¢), dimes (10¢), and quarters (25¢). The FSM tracks the total inserted value and allows a purchase when at least 20¢ is collected. If 25¢ is inserted, change is returned.

$\mathbf{I_1}$	$\mathbf{I_2}$	**Coin Received**
0	0	None
0	1	Quarter
1	0	Nickel
1	1	Dime

Table 1: Input Mapping to Coins Received

1.1 FSM Design Process

The FSM was developed using a state transition approach, where the system moves through predefined states based on coin inputs. The state diagram figure 4 illustrates the FSM transitions, where I1 and I2 determine the type of coin value inserted. The FSM tracks the accumulated value and decides when a purchase is valid. The system was structured such

that upon reaching a minimum of 20¢, the purchase output (A) was activated, allowing the user to make a selection. If the user inserted 25¢, the FSM would not only activate the purchase signal but also enable the change return output (C), ensuring the user received the correct refund. The FSM was structured using a series of defined states, where each state represented the amount of money inserted. State transitions were triggered based on the coin input signals (I1, I2), and the system was designed to return to the initial state (S0) after a purchase was made or if the reset (R) signal was activated.

To validate the FSM, extensive testing was performed using Quartus' University Waveform tool as seen in the addendum section, which provided a way to simulate and analyze the state transitions and output behavior. The simulation results confirmed that the FSM correctly handled various scenarios, including different coin insertion sequences, overpayment cases, and reset functionality. During testing, an issue was initially observed where the A signal remained high even after a valid purchase was made. This issue was resolved by ensuring that after a valid transaction, the FSM returned to the initial state. Additionally, the FSM's ability to return change was tested by inserting 25¢ and confirming that the C output was activated accordingly. By thoroughly analyzing state transitions and using waveform simulations to visualize the FSM's operation, the design was refined to ensure reliable performance. The final implementation successfully demonstrated a working vending machine FSM capable of tracking inserted coins, enabling purchases at the correct threshold, and handling overpayment scenarios accurately.

Current State	$\mathbf{I_1}$	$\mathbf{I_2}$	**Next State**	**Outputs (A, C)**
S0	0	1	S25	(0, 0)
S0	1	0	S5	(0, 0)
S0	1	1	S10	(0, 0)
S5	0	1	S30	(0, 0)
S5	1	0	S10	(0, 0)
S5	1	1	S15	(0, 0)
S10	0	1	S0	(0, 1)
S10	1	0	S15	(0, 0)
S10	1	1	S20	(0, 0)
S15	0	1	S5	(0, 1)
S15	1	0	S20	(0, 0)
S15	1	1	S25	(0, 0)
S20	X	X	S0	(1, 0)
S25	X	X	S5	(1, 1)
S30	X	X	S10	(1, 1)

Table 2: State Transition Table for Vending Machine FSM

2 Results

The implementation of the vending machine FSM was tested using Quartus University Waveform tool to validate its state transitions, outputs, and overall functionality. The primary objective of the simulation was to ensure that the FSM correctly responded to different coin inputs, activated the purchase signal (A) when 20¢ was inserted, and returned change (C) when 25¢ was inserted. The simulation results provided insight into how the system handled different test cases, including proper state progression, reset functionality, and overpayment scenarios. During the initial phase of testing, the FSM appeared to transition correctly through states, but an issue was discovered where the A signal remained high even after completing a valid transaction. This was traced back to the state transition logic, where the system was not correctly resetting to S0 after a purchase was made. After adjusting the state transition conditions, the FSM behaved as expected, properly resetting to S0 after a transaction. Another key test case involved inserting 25¢, which required the system to activate the purchase output while simultaneously enabling the change return signal. The waveform confirmed that the FSM correctly returned change by activating C, followed by a transition back to S5, ensuring that the extra 5¢ was accounted for.

Additional tests were conducted to verify the functionality of the reset (R) input. When activated, R was expected to immediately transition the FSM back to S0, regardless of the current state. The simulation results confirmed that pressing reset forced the system back to its initial state, effectively clearing any previously inserted values. These test cases demonstrated the robustness of the FSM, proving that it could correctly track inserted amounts, transition between states based on inputs, and respond appropriately to purchase and change-return conditions. The final implementation met all project specifications, ensuring that the vending machine FSM operated correctly under all tested scenarios.

2.1 VHDL Code explanation

The vending machine FSM is implemented in VHDL, using a finite state machine architecture that defines a sequence of states and transitions based on input signals. The VHDL code consists of three key sections: entity declaration, state transitions, and state behavior logic.

- Clock: Controls the timing of state transitions.
- R: Reset input, which forces the FSM to return to S0
- inputOne and inputTwo: Value of coin in binary
- A: When a purchase is valid
- C: When change is given

Unfortunately, due to incorrect formatting and comp ability issues, VHDL code presented will have their corresponding listing above

Listing 1: Entity declaration

```
ENTITY Vending_Machine IS
    PORT (
        clock     : IN STD_LOGIC; -- Clk signal
        R         : IN STD_LOGIC;  -- Reset
        inputOne, inputTwo : IN STD_LOGIC;  -- Inputs
        C, A    : OUT STD_LOGIC  -- Outputs C being change and A is
            purchase once 20 cents has been met
    );
END Vending_Machine;
```

Listing 2: state transition logic

```
TYPE state_type IS (S0, S5, S10, S15, S20, S25, S30);
SIGNAL state, next_state : state_type;
```

Next, the state transition process ensures that state changes occur on the rising edge of the clock. If the reset signal (R) is high, the FSM immediately transitions to S0, clearing any previous inputs. Otherwise, the FSM updates its state based on the next state logic.

Listing 3: state transition process

```
PROCESS (clock, R)
BEGIN
    IF R = '1' THEN
        state <= S0;  -- Reset to S0 when R is high
    ELSIF rising_edge(clock) THEN
        state <= next_state;  -- Update state on rising clock edge
    END IF;
END PROCESS;
```

Each state defines transitions based on inserted coins. Plese refer to listing 4 for full code.

1. If inputOne = 0 and inputTwo = 1, a quarter (25¢) is inserted, transitioning to S25.
2. If inputOne = 1 and inputTwo = 0, a nickel (5¢) is inserted, transitioning to S5.
3. If inputOne = 1 and inputTwo = 1, a dime (10¢) is inserted, transitioning to S10.

Listing 4: state transition process

```
WHEN S0 =>
    IF inputOne = '0' AND inputTwo = '1' THEN next_state <= S25;
    ELSIF inputOne = '1' AND inputTwo = '0' THEN next_state <= S5;
    ELSIF inputOne = '1' AND inputTwo = '1' THEN next_state <= S10;
    END IF;
```

3 Conclusion

This project provided valuable experience in designing, implementing, and testing a finite state machine (FSM) using VHDL. The vending machine FSM was structured to track

inserted coins, determine when a purchase could be made, and correctly return change when necessary. The development process required careful planning of state transitions and output conditions to ensure proper operation. By using Quartus Waveform tool, the FSM's behavior was simulated and verified across multiple test scenarios. One of the key challenges encountered during the implementation was ensuring that the FSM properly reset to its initial state (S0) after a valid transaction. Initially, the A output remained active beyond the expected state, requiring refinements in the transition logic. Additionally, validating the change return function (C) when 25¢ was inserted necessitated adjustments to the state progression. The simulation results confirmed that these issues were successfully resolved, demonstrating the FSM's ability to handle various coin insertion sequences, overpayment cases, and reset conditions. The successful completion of this project further emphasized the effectiveness of using state diagrams and transition tables in FSM design, ensuring a structured and logical approach to system development.

4 Addendum

The full VHDL implementation is included below:

Listing 5: Full VHDL Code for Vending Machine FSM

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Vending_Machine IS
    PORT (
        clock    : IN STD_LOGIC; -- Clk signal
        R        : IN STD_LOGIC;  -- Reset
        inputOne, inputTwo : IN STD_LOGIC;  -- Inputs
        C, A    : OUT STD_LOGIC  -- Outputs C being change and A is
            purchase once 20 cents has been met
    );
END Vending_Machine;

ARCHITECTURE behavior OF Vending_Machine IS
    -- state type
    TYPE state_type IS (S0, S5, S10, S15, S20, S25, S30);
    SIGNAL state, next_state : state_type;

BEGIN
    -- Synchronous process for state transitions
    PROCESS (clock, R)
    BEGIN
        IF R = '1' THEN
            state <= S0;  -- Reset to S0 when R is high
        ELSIF rising_edge(clock) THEN
            state <= next_state;  -- Update state on rising clock edge
        END IF;
    END PROCESS;
```

```
PROCESS (state, inputOne, inputTwo)
BEGIN
    -- Default assignments
    C <= '0';
    A <= '0';
    next_state <= state;

    -- The following case show state based on cents currently and
       inputs change it to the state which corresponds to the value
    CASE state IS
        WHEN S0 =>
            IF inputOne = '0' AND inputTwo = '1' THEN next_state <=
               S25;
            ELSIF inputOne = '1' AND inputTwo = '0' THEN next_state <=
                S5;
            ELSIF inputOne = '1' AND inputTwo = '1' THEN next_state <=
                S10;
            END IF;

        WHEN S5 =>
            IF inputOne = '0' AND inputTwo = '1' THEN next_state <=
               S30;
            ELSIF inputOne = '1' AND inputTwo = '0' THEN next_state <=
                S10;
            ELSIF inputOne = '1' AND inputTwo = '1' THEN next_state <=
                S15;
            END IF;

        WHEN S10 =>
            IF inputOne = '0' AND inputTwo = '1' THEN
                next_state <= S0;
                C <= '1';
            ELSIF inputOne = '1' AND inputTwo = '0' THEN next_state <=
                S15;
            ELSIF inputOne = '1' AND inputTwo = '1' THEN next_state <=
                S20;
            END IF;

        WHEN S15 =>
            IF inputOne = '0' AND inputTwo = '1' THEN
                next_state <= S5;
                C <= '1';
            ELSIF inputOne = '1' AND inputTwo = '0' THEN next_state <=
                S20;
            ELSIF inputOne = '1' AND inputTwo = '1' THEN next_state <=
                S25;
            END IF;

        WHEN S20 =>
            A <= '1';  -- Activate output A since cents is met!
            next_state <= S0;

        WHEN S25 =>
            A <= '1';
```

```
                C <= '1';
                next_state <= S5;

            WHEN S30 =>
                A <= '1';
                C <= '1';
                next_state <= S10;
        END CASE;
    END PROCESS;
END behavior;
```

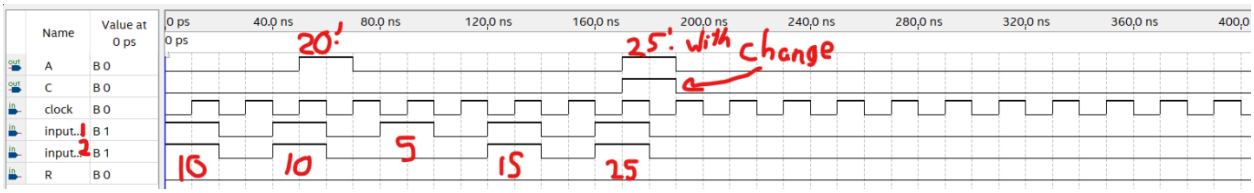

Figure 1: Basic testing such as reaching 20 cents and 25 cents to see if C signal works

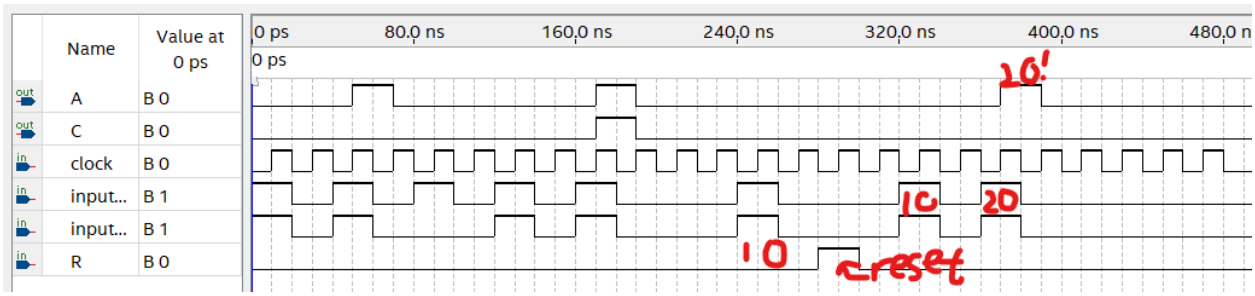

Figure 2: Testing reset

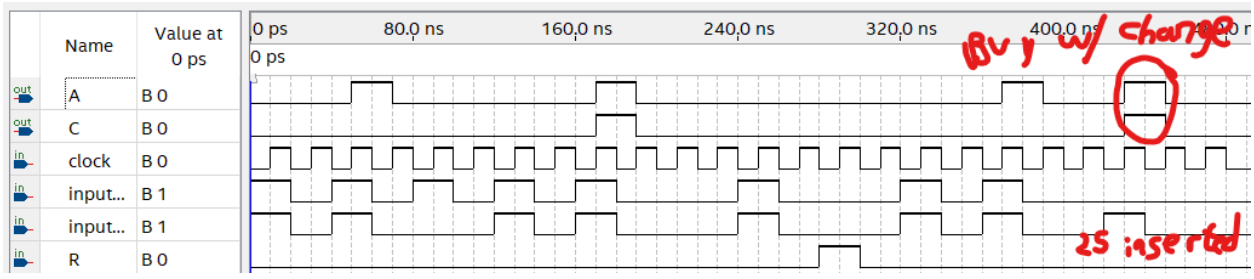

Figure 3: Quarter inserted

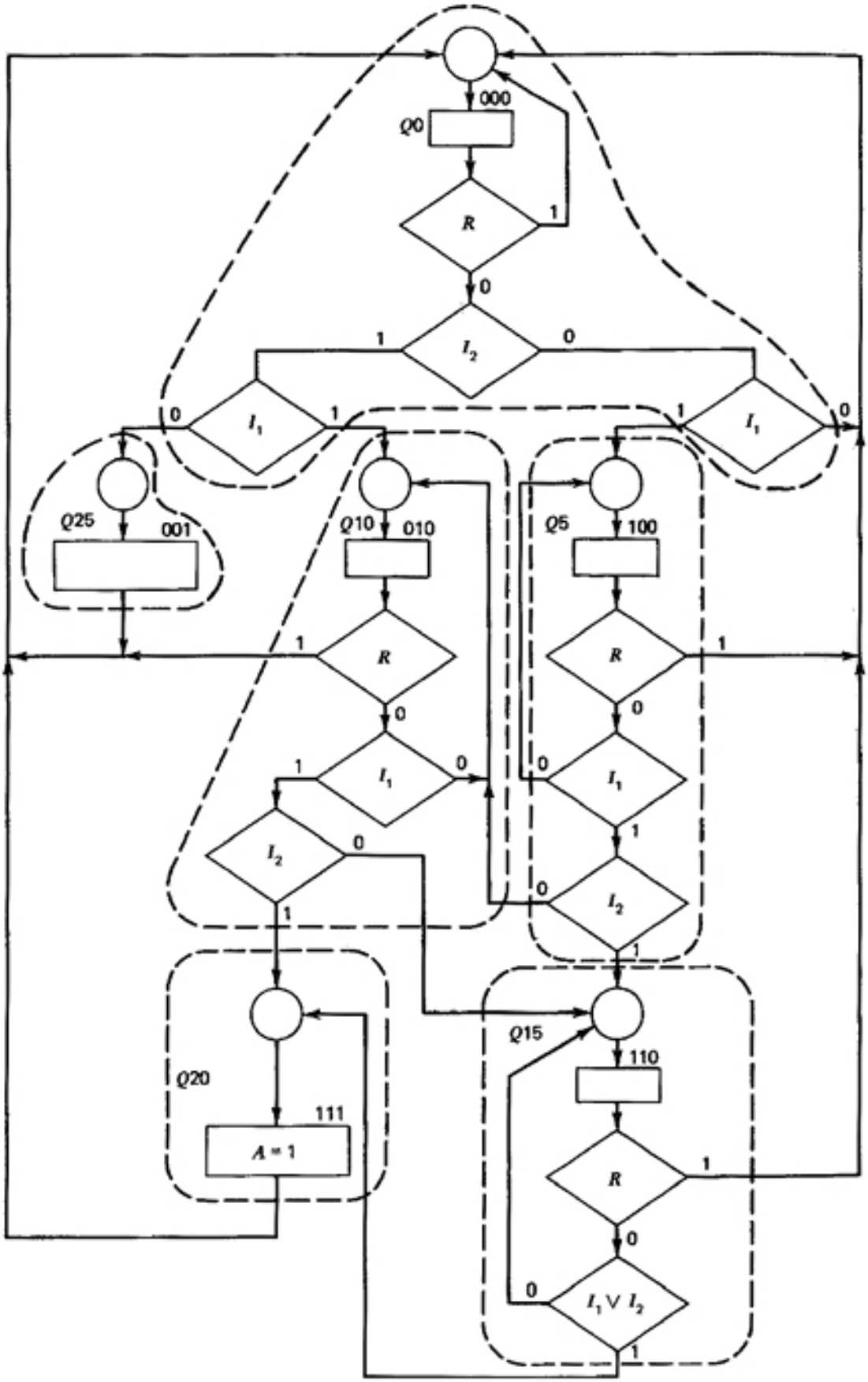

Figure 4: State diagram of vending machine