Lab 4: Finite State Machines

Lab 4A: Traffic Light Controller

Module: traffic_controller

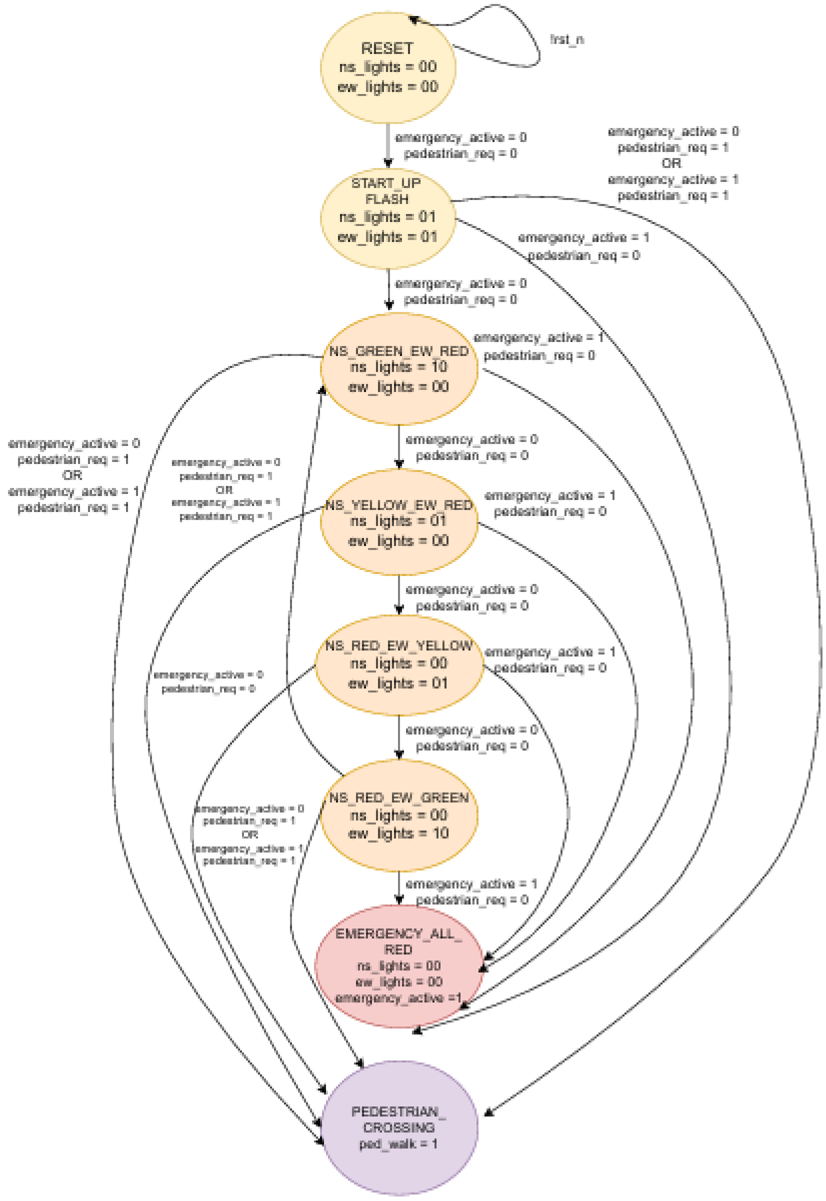
Purpose: The purpose of the traffic_controller module is to design and implement a traffic light control system for a four-way intersection using a **finite state machine (FSM)**. The controller manages the timing and sequencing of green, yellow, and red lights for both **north-south** and **east-west** directions, while also handling **pedestrian crossing requests** and **emergency overrides**. By using an FSM, the design ensures safe, predictable, and efficient traffic flow under normal conditions and provides immediate response to emergency situations.

Interface Signals

- **clk** – Input: 1 Hz system clock used for timing and FSM transitions.
- **rst_n** – Input: Active-low asynchronous reset; initializes the controller to all-red safe state.
- **emergency** – Input: Emergency override signal; when high, forces all lights to red (flashing).
- **pedestrian_req** – Input: Pedestrian crossing request; triggers pedestrian walk cycle.
- **ns_lights [1:0]** – Output: North-South traffic lights. Encoded as 2’b01 = Red, 2’b10 = Yellow, 2’b11 = Green.
- **ew_lights [1:0]** – Output: East-West traffic lights. Encoded as 2’b01 = Red, 2’b10 = Yellow, 2’b11 = Green.
- **ped_walk** – Output: Pedestrian walk signal; high when it is safe for pedestrians to cross.
- **emergency_active** – Output: Indicates when the FSM is in emergency mode.

4.1 State Machine

Diagram



4.1.1 All States Clearly Labeled

The FSM consists of the following states:

- IDLE – Reset/initial state, system is inactive.
- STARTUP_FLASH – Startup sequence, lights flash red for initialization.
- NS_GREEN_EW_RED – North-South green, East-West red.
- NS_YELLOW_EW_RED – North-South yellow, East-West red.

- NS_RED_EW_GREEN – North-South red, East-West green.
- NS_RED_EW_YELLOW – North-South red, East-West yellow.
- EMERGENCY_ALL_RED – Emergency override state, all signals red (flashing).
- PEDESTRIAN_CROSSING – Pedestrian walk phase, all vehicle lights red.

4.1.2 All Transitions with Conditions

- **IDLE → STARTUP_FLASH** : occurs automatically on reset release.
- **STARTUP_FLASH → NS_GREEN_EW_RED** : if no emergency and no pedestrian request.
- **NS_GREEN_EW_RED → NS_YELLOW_EW_RED** : after 30 cycles, if no higher-priority request.
- **NS_YELLOW_EW_RED → NS_RED_EW_GREEN** : after 5 cycles, if no higher-priority request.
- **NS_RED_EW_GREEN → NS_RED_EW_YELLOW** : after 30 cycles, if no higher-priority request.
- **NS_RED_EW_YELLOW → NS_GREEN_EW_RED** : after 5 cycles, if no higher-priority request.
- **Any traffic state → EMERGENCY_ALL_RED** : if emergency = 1.
- **Any traffic state → PEDESTRIAN_CROSSING** : if pedestrian_req = 1.
- **EMERGENCY_ALL_RED → prev_state** : after 15 cycles or when emergency = 0.
- **PEDESTRIAN_CROSSING → prev_state** : after 15 cycles or when pedestrian_req = 0.

4.1.3 Reset State Identified

- On reset (rst_n = 0), the FSM initializes to **IDLE** state with all outputs inactive.

4.1.4 Timing Relationships Specified

- State durations are determined by the **counter**.
- **Green**: 30 cycles (30 seconds).
- **Yellow**: 5 cycles (5 seconds).
- **Pedestrian crossing**: 15 cycles (15 seconds).
- **Emergency red**: 15 cycles (15 seconds, flashing).

- Counter resets whenever a state transition occurs, ensuring precise timing.

State Transition Table

NOTE:

“PREVIOUS_STATE” means the controller remembers where it was interrupted and resumes the normal sequence from there, instead of always restarting from NS side.

Lab 4B: Vending Machine Controller

Module: vending_machine

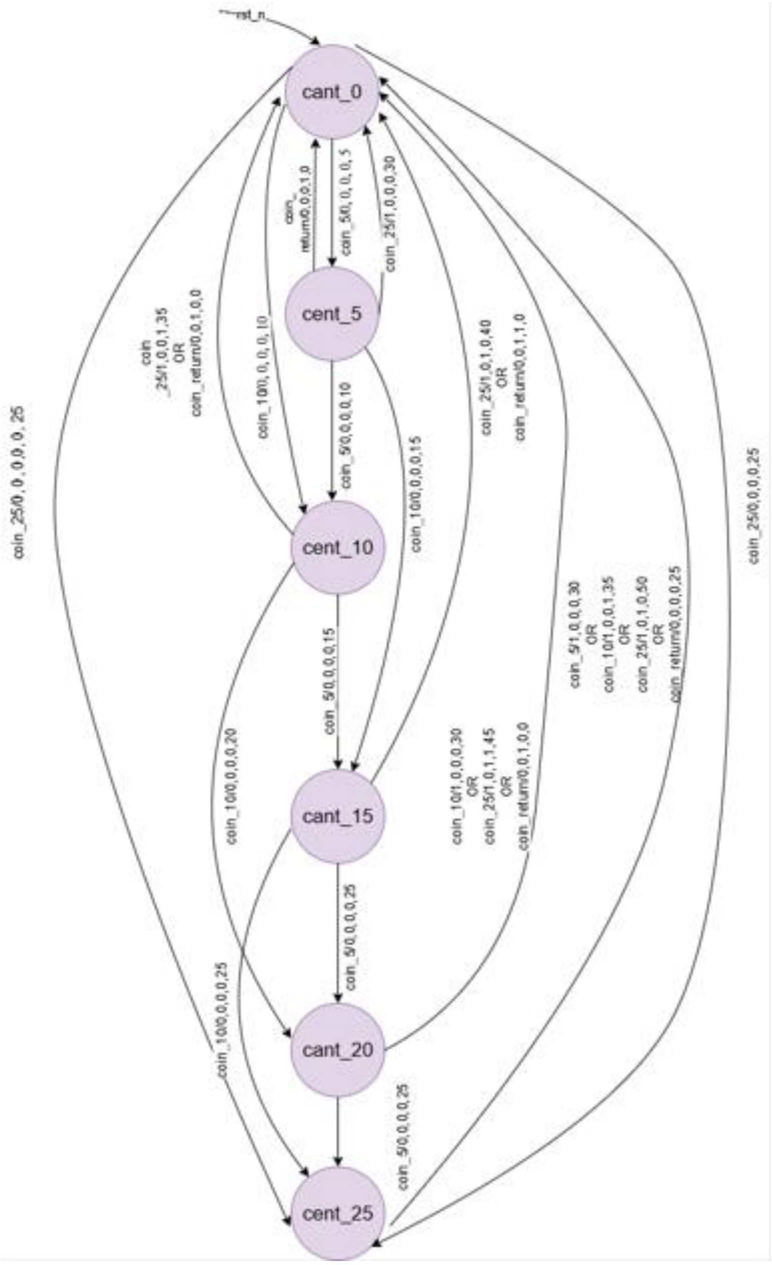
Purpose: The vending machine module is designed to accept coins of 5, 10, and 25 cents, track the total amount inserted, and dispense an item when the required cost is reached. It also supports a coin return function to give back the balance in appropriate denominations. The FSM ensures correct handling of coin inputs, manages the amount display, and generates control signals for dispensing items and returning coins, providing a reliable and user-friendly vending system.

Interface Signals

- **clk (input):** System clock used to drive the FSM and timing of operations.
- **rst_n (input):** Active-low reset; initializes the machine to idle state and clears balance.
- **coin_5 (input):** Signal that a 5-cent coin is inserted.
- **coin_10 (input):** Signal that a 10-cent coin is inserted.
- **coin_25 (input):** Signal that a 25-cent coin is inserted.
- **coin_return (input):** User presses coin return button to get back inserted balance.
- **dispense_item (output):** Asserted when the total inserted amount reaches the required value for an item; triggers item release.
- **return_5 (output):** Activates when a 5-cent coin must be returned to the user.
- **return_10 (output):** Activates when a 10-cent coin must be returned.
- **return_25 (output):** Activates when a 25-cent coin must be returned.
- **amount_display [5:0] (output):** Binary output showing the current inserted amount (in cents) for display purposes.

4.2 State Machine

Diagram



4.2.1 All States Clearly Labeled

Each state corresponds to the total balance (e.g., IDLE, 5, 10, 15,25).

4.2.2 All Transitions with Conditions

- **coin_5 →** Add 5¢ and move to next balance state.

- **coin_10 →** Add 10¢ and move to next balance state.
- **coin_25 →** Add 25¢ and move to next balance state.

4.2.3 Reset State Identified

- On reset (rst_n = 0), machine goes to **IDLE** state with balance = 0.

4.2.4 FSM Type

Designed **Mealy FSM**, because outputs (dispense_item, return_5/10/25, amount_display) depend on both the **current state** and the **input (coin_x, coin_return)** at the same clock cycle.

State Transition Table

.

Current State	Input condition	Next State	Outputs (dispense, return_25, return_10, return_5, amount_display)
IDLE (0¢)	`coin_5 = 1`	`cent_5`	0, 0, 0, 0, 5
	`coin_10 = 1`	`cent_10`	0, 0, 0, 0, 10
	`coin_25 = 1`	`cent_25`	0, 0, 0, 0, 25
	`coin_return = 1`	`IDLE`	0, 0, 0, 0, 0
	no input	`IDLE`	0, 0, 0, 0, 0
cent_5 (5¢)	`coin_5 = 1`	`cent_10`	0, 0, 0, 0, 10
	`coin_10 = 1`	`cent_15`	0, 0, 0, 0, 15
	`coin_25 = 1`	`IDLE`	**1**, 0, 0, 0, 30 (dispense, no change)
	`coin_return = 1`	`IDLE`	0, 0, 0, **1**, 0 (return 5¢)
	no input	`cent_5`	0, 0, 0, 0, 5
cent_10 (10¢)	`coin_5 = 1`	`cent_15`	0, 0, 0, 0, 15
	`coin_10 = 1`	`cent_20`	0, 0, 0, 0, 20
	`coin_25 = 1`	`IDLE`	**1**, 0, 0, **1**, 35 (dispense, return 5¢)
	`coin_return = 1`	`IDLE`	0, 0, **1**, 0, 0 (return 10¢)
	no input	`cent_10`	0, 0, 0, 0, 10
cent_15 (15¢)	`coin_5 = 1`	`cent_20`	0, 0, 0, 0, 20
	`coin_10 = 1`	`cent_25`	0, 0, 0, 0, 25
	`coin_25 = 1`	`IDLE`	**1**, 0, **1**, 0, 40 (dispense, return 10¢)
	`coin_return = 1`	`IDLE`	0, 0, **1**, **1**, 0 (return 10¢ + 5¢)
	no input	`cent_15`	0, 0, 0, 0, 15

cent_20 (20¢)	`coin_5 = 1`	`cent_25`	0, 0, 0, 0, 25
	`coin_10 = 1`	`IDLE`	**1**, 0, 0, 0, 30 (dispense, no change)
	`coin_25 = 1`	`IDLE`	**1**, 0, **1**, **1**, 45 (dispense, return 10¢ + 5¢)
	`coin_return = 1`	`IDLE`	0, 0, **1**, 0, 0 (return 10¢)
	no input	`cent_20`	0, 0, 0, 0, 20
cent_25 (25¢)	`coin_5 = 1`	`IDLE`	**1**, 0, 0, 0, 30 (dispense, no change)
	`coin_10 = 1`	`IDLE`	**1**, 0, 0, **1**, 35 (dispense, return 5¢)
	`coin_25 = 1`	`IDLE`	**1**, 0, **1**, 0, 50 (dispense, return 10¢)
	`coin_return = 1`	`IDLE`	0, **1**, 0, 0, 0 (return 25¢)
	no input	`cent_25`	0, 0, 0, 0, 25

4.3 Design Review Checklist

Pre-Implementation Review

- Specification completely understood
- State diagrams complete with all transitions

Code Quality Checklist

- Consistent naming conventions (ns_lights, ew_lights, ped_walk, etc.)
- No combinational loops
- No unintended latches (synchronous FSM design)
- [Reset strategy consistent (asynchronous active-low rst_n)
- Comments explain design intent (purpose, signals, state diagram documented)