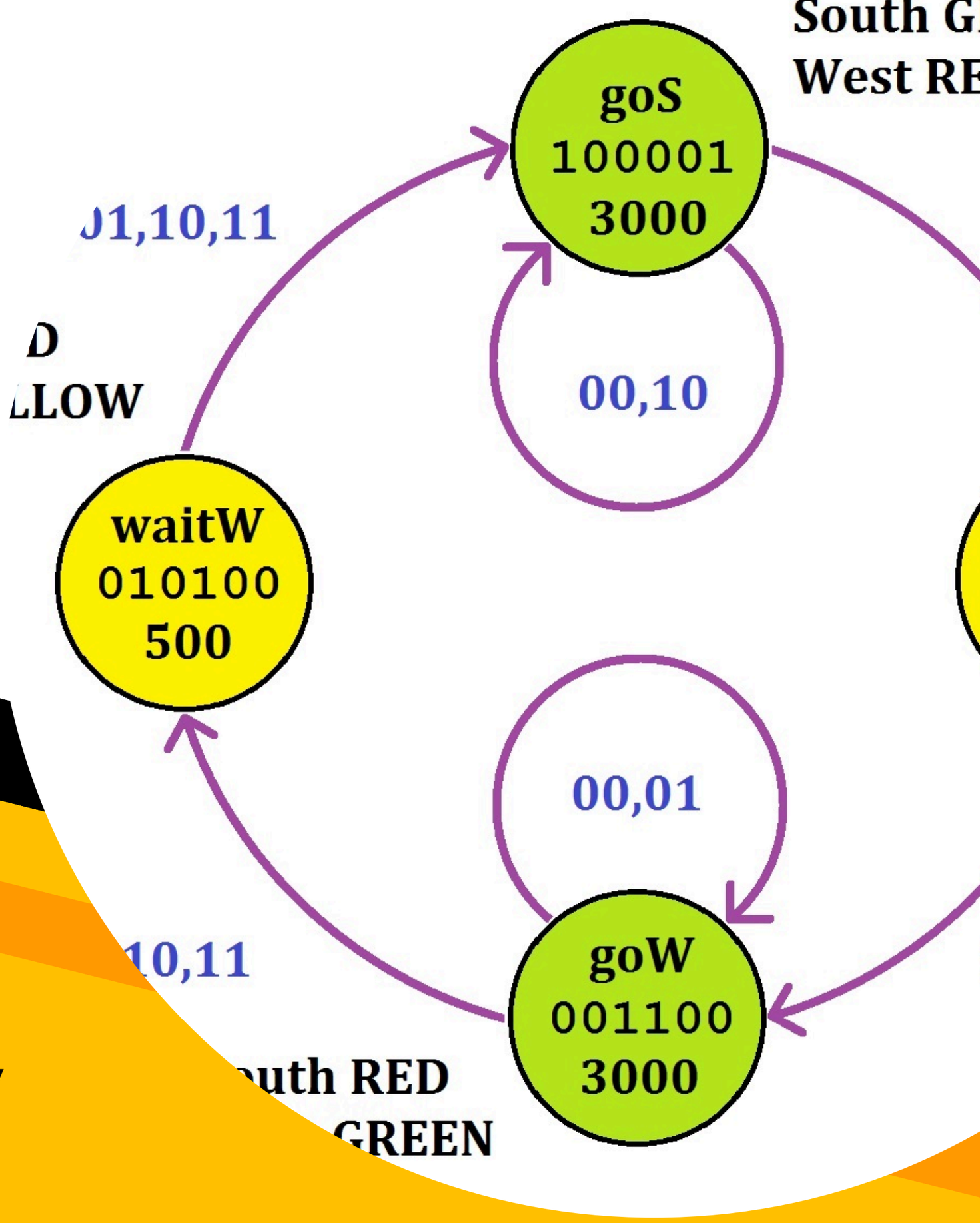
MUST TRY

ASSIGNMENT

FINITE STATE MACHINE

Applications



PROV LOGIC

1. Elevator Controller FSM (3-4 Floors) - RTL Design

Problem Statement:

Design an FSM-based Elevator Controller that handles requests between 4 floors (F0 to F3). The elevator should:

- Move up or down based on requests from each floor.
- Open the door when it reaches the requested floor.
- Keep the door open for a few clock cycles (say, 2-3) and then close.
- Reject requests to the current floor if the door is already open.
- Reset to ground floor (F0) on reset.

Inputs:

- clk, reset
- floor_request[3:0] → One-hot signal (1 if requested)
- current_floor[1:0] → Binary-encoded current floor

Outputs:

- move_up, move_down, door_open

States:

- IDLE
- MOVING_UP
- MOVING_DOWN
- DOOR_OPEN
- DOOR_CLOSE

2. Vending Machine FSM (₹1, ₹2 Coins, ₹5 Product)

Problem Statement:

Design an FSM for a Vending Machine that accepts coins of ₹1 and ₹2 denominations. The machine should:

- Accumulate the total inserted amount.
- Dispense the product when ₹5 or more is reached.
- Return excess amount if inserted > ₹5.
- Reset after dispensing.

Inputs:

- clk, reset
- coin[1:0] → 2'b01 for ₹1, 2'b10 for ₹2

Outputs:

- product_dispense
- return_change[2:0]

States:

- ₹0, ₹1, ₹2, ₹3, ₹4, ₹5_OR_MORE

3. Parity Checker FSM (Even/Odd)

Problem Statement:

Design a serial Parity Checker FSM which checks if the number of 1's in a serial data stream is even or odd. It should:

- Output 1 for even parity, 0 for odd.
- Support both even and odd parity modes (configurable).
- Check parity bit at the end of the stream.

Inputs:

- clk, reset
- data_in (1-bit serial input)
- valid (1 when data_in is valid)
- mode (0 → even, 1 → odd)

Outputs:

- parity_ok → 1 if parity matches expected

States:

- EVEN_STATE
- ODD_STATE

4. Password Checker FSM (4-bit Password: 4 inputs)

Problem Statement:

Design an FSM-based Password Checker to verify a 4-bit password entered bit-by-bit (MSB first). It should:

- Compare input sequence with preset password (e.g., 1011).
- Output access_granted when correct sequence is entered.
- Reset FSM if any bit is wrong or after successful match.

Inputs:

- clk, reset
- input_bit
- valid (1 when bit is entered)

Outputs:

- access_granted

States:

- S0 → No bits matched
- S1 → 1st bit matched
- S2 → 2 bits matched
- S3 → 3 bits matched
- S4 → All matched → access granted
- ERROR → Wrong bit entered

5. Digital Lock System FSM (Keypad-Based 4-Digit Code)

Problem Statement:

Design an FSM-based Digital Lock System where a 4-digit password is entered via a keypad (0-9). The system should:

- Unlock when correct 4-digit code is entered in sequence (e.g., 4-2-5-1).
- Lock again after one cycle or on reset.
- Allow only 3 wrong attempts; then enter lockout.
- Reset after admin resets or power cycle.

Inputs:

- clk, reset
- keypad_input[3:0] (BCD input)
- enter (high when a digit is entered)

Outputs:

- unlocked
- attempts_left[1:0]
- locked_out

States:

- IDLE
- DIGIT_1, DIGIT_2, DIGIT_3, DIGIT_4
- ACCESS_GRANTED
- WRONG_CODE
- LOCKED_OUT

6. Serial Data Pattern Detector (1101 Detection with Overlapping)

Problem Statement:

Design an FSM that detects the binary sequence "1101" from a serial input stream. The detection should support overlapping sequences (e.g., input: 11101101 should detect twice).

Inputs:

- clk, reset
- data_in (1-bit serial input)

Outputs:

- detected → High when the pattern 1101 is found

States:

- S0: No match
- S1: 1 matched (1)
- S2: 2 matched (11)
- S3: 3 matched (110)
- S4: Pattern detected

7. Automatic Washing Machine Controller FSM

Problem Statement:

Design an FSM to simulate the working of a semi-automatic washing machine with the following operations:

- Fill → Wash → Rinse → Spin → Stop
- Each stage takes a fixed number of cycles
- Machine should respond to start, pause, and reset signals.

Inputs:

- clk, reset, start, pause

Outputs:

- stage[2:0] → Indicates current stage
- done → High when complete

States:

- IDLE
- FILL
- WASH
- RINSE
- SPIN
- DONE

8. SPI Master FSM Controller

Problem Statement:

Design an FSM to act as an SPI Master that communicates with a slave. It should:

- Manage signals like SCLK, MOSI, SS
- Send an 8-bit data frame
- Support start_tx to begin transmission
- Assert tx_done after transmission completes

Inputs:

- clk, reset
- start_tx
- data_in[7:0]

Outputs:

- SCLK, MOSI, SS
- tx_done

States:

- IDLE
- LOAD
- SHIFT
- DONE

9. Token Ring Arbiter FSM

Problem Statement:

Design an FSM for a 4-node token ring bus system where each node can access the bus only when it has the token.

The FSM:

- Passes the token in a round-robin fashion
- Waits for a node to complete its task before moving the token
- Can reset to Node 0 on system reset

Inputs:

- clk, reset
- task_done[3:0] → Signals from nodes when done

Outputs:

- token_owner[1:0]

States:

- NODE_0, NODE_1, NODE_2, NODE_3

10. Smart Door Access with RFID FSM

Problem Statement:

Design an FSM that controls a Smart Door Lock using an RFID reader. It should:

- Wait for RFID scan
- Match scanned ID with stored ID
- Open door if match, deny otherwise
- Lock door after 3 seconds

Inputs:

- clk, reset
- rfid_data[7:0]
- valid_scan

Outputs:

- door_unlock
- access_granted
- access_denied

States:

- IDLE
- CHECK_ID
- ACCESS_GRANTED
- ACCESS_DENIED
- DOOR_LOCK