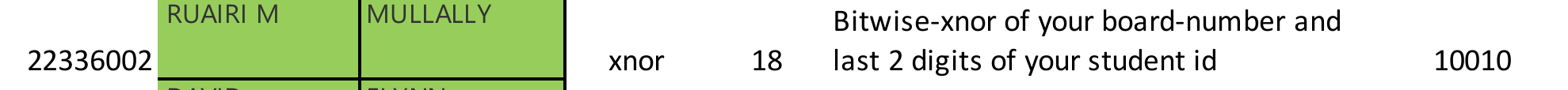
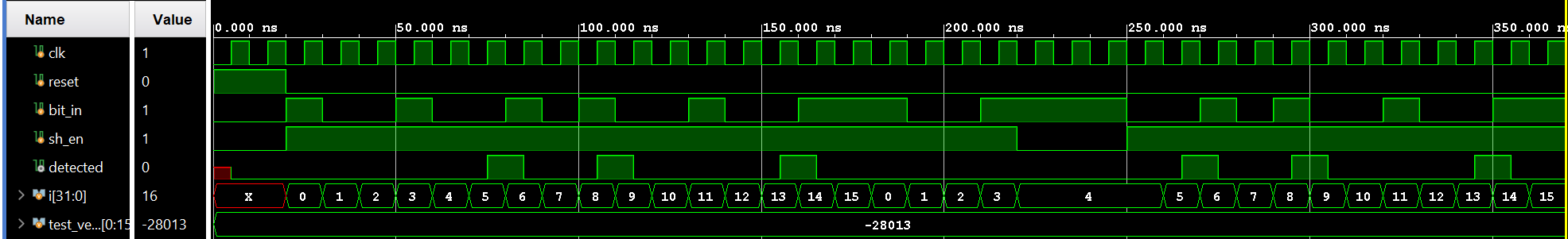
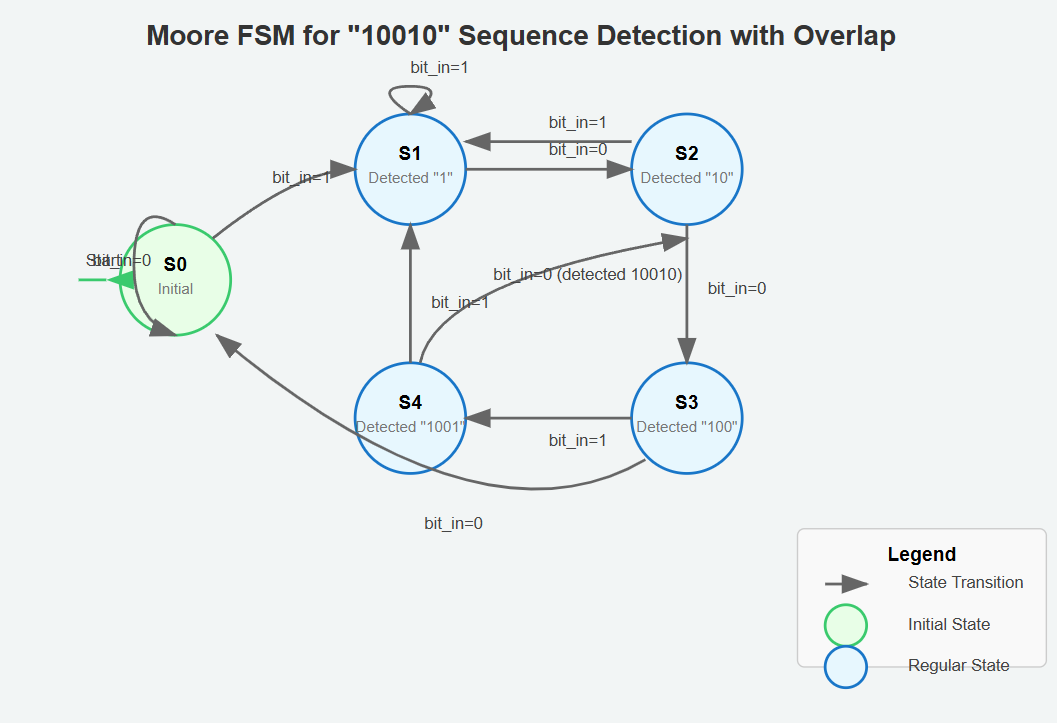
Codeword

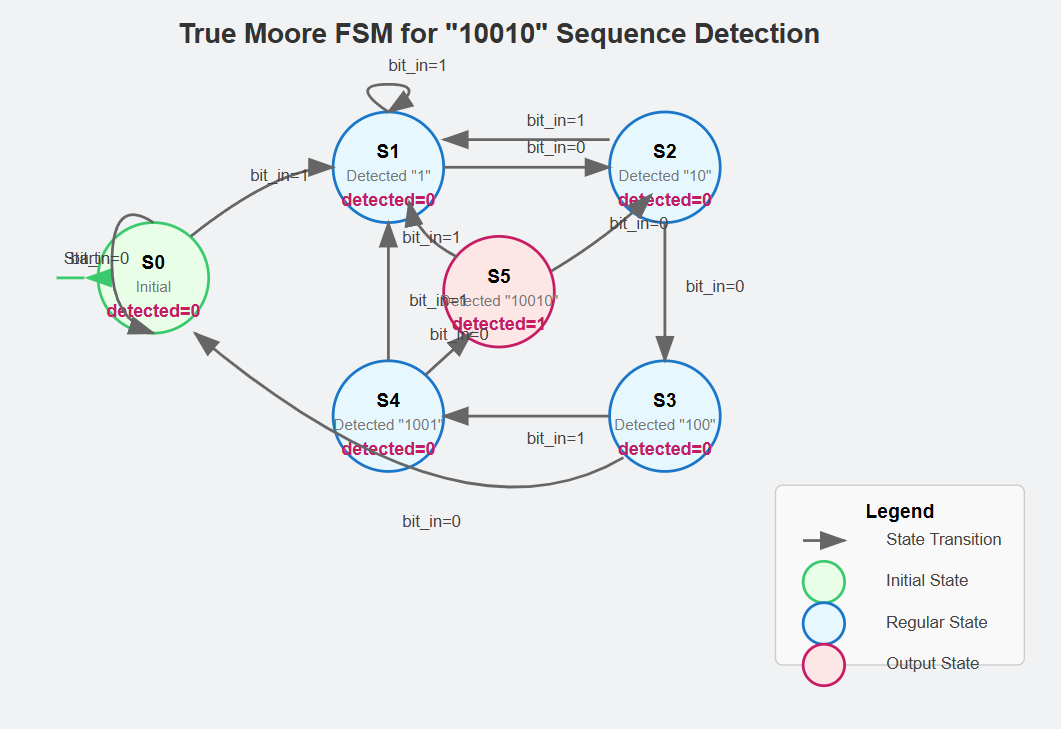


FSM tests:





Mealy



**State Encoding and Transition Logic Optimizations**

Looking at the design of this Moore FSM for the 10010 sequence detector, there are several notable optimizations in both the state encoding and transition logic:

**State Encoding Optimizations**

1. **Binary Encoding**: The design uses 3-bit binary encoding (requiring 3 flip-flops) to represent 6 states:

verilog

Copy

parameter s0 = 3'b000; // Initial state

parameter s1 = 3'b001; // Detected "1"

parameter s2 = 3'b010; // Detected "10"

parameter s3 = 3'b011; // Detected "100"

parameter s4 = 3'b100; // Detected "1001"

parameter s5 = 3'b101; // Detected "10010"

This is more efficient than one-hot encoding, which would require 6 flip-flops.

1. **Semantic State Values**: The state encoding follows a pattern where the binary value increases as we progress through the sequence detection, making the state values somewhat meaningful. This can make debugging easier.
2. **Unused State Handling**: The design includes a default case in the state transition logic that returns to the initial state s0 if an undefined state is encountered, providing robustness against state corruption.

**Transition Logic Optimizations**

1. **Efficient Overlap Handling**: The FSM cleverly handles overlapping patterns by transitioning from:
   * State s5 (after detecting "10010") to state s2 when receiving a "0"
   * This immediately puts us at the "10" position of a potential new sequence
   * When sending "1001010010", this optimization allows detection of both patterns
2. **Input-Based Branching**: The transition logic follows a clean pattern where each state has at most two possible next states based on the input bit, making the logic easy to implement with simple multiplexers.
3. **Shift Enable Optimization**: The shift enable (sh\_en) input is handled efficiently:

verilog

Copy

// Default: maintain current state

next\_state = current\_state;

// Only calculate next state when sh\_en is high

if (sh\_en) begin

case (current\_state)

// State transition logic

endcase

end

This approach:

* + Avoids unnecessary state computation when sh\_en is low
  + Defaults to maintaining the current state, eliminating the need for conditional logic in each state case

1. **Reset Simplicity**: The design uses a synchronous reset implementation that directly sets the state to s0 and clears the output, which is typically more efficient to implement than complex reset logic.

**Potential Additional Optimizations**

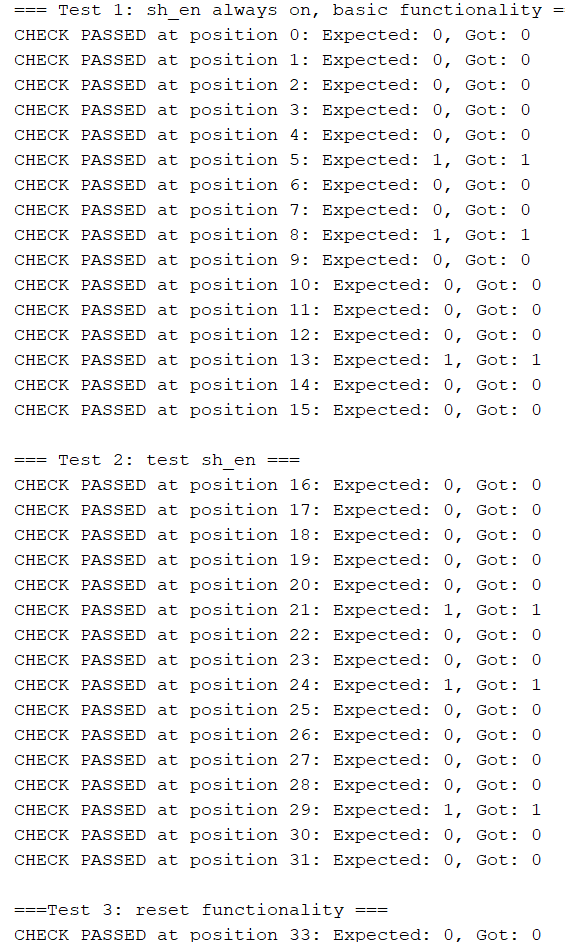
While the current design is quite optimized, a few additional optimizations could be considered:

1. **Gray Code Encoding**: Using Gray code for state encoding could reduce switching activity, as only one bit changes between adjacent states, potentially saving power.
2. **One-Hot Encoding**: For FPGAs specifically, one-hot encoding (where each state uses a dedicated flip-flop) often leads to faster and more efficient implementations despite using more flip-flops.
3. **State Minimization**: A formal state minimization technique (like the Quine-McCluskey algorithm) could potentially reduce the total number of states needed, though in this specific pattern detection case, it's unlikely to yield benefits.

Grey codes

**Reduced Switching Activity**: Only one-bit changes during state transitions, which reduces power consumption in the flip-flops and associated logic.

Only 2 bits switched for 5 -> 1



FSM:

* Functional diagram of FSM
* State machine diagram
* Explanation of how the FSM works
* Minimisation strategy
* Testing: show vectors, and describe strategy

Top module:

* Functional diagram of whole module
* Counter to count detection of sequence
* Top level test bench
* Test plan for TLM: individual module testing, design of appropriate test vectors and validation plan, how the integrated module is tested
* Waveforms showing correct output
* Target to board
* Use peripherals and demonstrate how to use
* State how many times codeword is found in the full cycle of LFSR
* Discuss utilization and show registered flip-flops
* Timing report
* SS of project hierarchy
* Submit code and bitfile