Project 90: Moore Machine for Even Parity Generator

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma , Ayush Jain , Gati Goyal, Nikunj Agrawal

Documentation Specialist: Dhruv Patel & Nandini Maheshwari

Contents

1 Project Overview

The Moore Machine for an Even Parity Generator is a finite-state machine designed to ensure the parity of a serial binary input stream remains even. It consists of two states: S0 (Even) and S1 (Odd). The machine transitions between these states based on the input bit, generating an output (parity bit) depending solely on the current state. Using HDL (Verilog/VHDL), the design includes a state diagram, state transition table, and RTL coding. Simulated with tools like ModelSim or Vivado, the machine can be implemented on FPGA for real-world applications such as error detection in communication systems.

2 Moore Machine for Even Parity Generator

2.1 Key Components of Moore Machine for Even Parity Generator

States:

- ***S0 (Even Parity State):*** Represents an even number of 1s encountered so far.
- ***S1 (Odd Parity State):*** Represents an odd number of 1s encountered so far.

Inputs:
Serial binary input stream (0s and 1s). **Outputs:**

- Parity bit generated based solely on the current state.
- ***S0: Output*** $=$ 0 (even parity).
- ***S1: Output*** $=$ 1 (odd parity).

State Transition Logic:

- Determines the next state based on the current state and input bit.

State Diagram and Table:

- Graphical and tabular representations of transitions and outputs.

RTL Implementation:

- Registers to store the current state and combinational logic for transitions.

Clock:

- Synchronizes state transitions.

Reset:

- Initializes the machine to S0 (even parity).

2.2 Working of Moore Machine for Even Parity Generator

Initialization:

- The machine starts in the S0 (Even Parity State), assuming no 1s have been encountered.

State Transition Logic:

- Based on the input bit, the machine transitions between states:
- ***S0*** $\rightarrow$ ***S0:*** Input = 0 (parity remains even).

- ***S0 → S1:*** Input = 1 (parity changes to odd).
- ***S1 → S1:*** Input = 0 (parity remains odd).
- ***S1 → S0:*** Input = 1 (parity changes to even).

Output Generation:

- The output is determined solely by the current state:
- ***S0:*** Output = 0 (even parity).
- ***S1:*** Output = 1 (odd parity).

Operation:

- For each input bit, the current state transitions according to the state table.
- The machine ensures the parity bit reflects the cumulative parity of the input sequence at any time.

2.3 RTL Code

Listing 1: Moore Machine for Even Parity Generator

```
1
2  module moore_even_parity (
3      input  logic clk, reset,
4      input  logic bit_in,          // Serial input bit stream
5      output logic parity_bit       // Even parity output
6  );
7
8      // Define states
9      typedef enum logic {EVEN, ODD} state_t;
10     state_t current_state, next_state;
11
12     // State transition logic
13     always_ff @(posedge clk or posedge reset) begin
14         if (reset)
15             current_state <= EVEN; // Start with even parity
16         else
17             current_state <= next_state;
18     end
19
20     // Next state logic
21     always_comb begin
22         case (current_state)
23             EVEN: next_state = (bit_in) ? ODD : EVEN; // Flip state if
                   bit_in is 1
24             ODD:  next_state = (bit_in) ? EVEN : ODD; // Flip state if
                   bit_in is 1
25         endcase
26     end
27
28     // Output logic (Moore output depends only on the current state)
29     always_ff @(posedge clk or posedge reset) begin
30         if (reset)
31             parity_bit <= 1'b0; // Start with even parity
32         else
33             parity_bit <= (current_state == ODD); // Parity bit is 1
                   if in ODD state
34     end
35 endmodule
```

2.4 Testbench

Listing 2: Moore Machine for Even Parity Generator

```
1
2  module tb_moore_even_parity();
3      logic clk, reset;
4      logic bit_in;
5      logic parity_bit;
6
7      moore_even_parity uut (
8          .clk(clk),
9          .reset(reset),
10         .bit_in(bit_in),
11         .parity_bit(parity_bit)
12     );
13
14     // Clock generation
15     initial begin
16         clk = 0;
17         forever #5 clk = ~clk; // 10ns clock period
18     end
19
20     // Test scenario
21     initial begin
22         reset = 1; bit_in = 0; // Start with reset asserted
23         #10 reset = 0;         // Deassert reset
24
25         // Apply serial input bits
26         #10 bit_in = 1;  // Transition to ODD
27         #10 bit_in = 0;  // Stay in ODD
28         #10 bit_in = 1;  // Transition to EVEN
29         #10 bit_in = 1;  // Transition to ODD
30         #10 bit_in = 0;  // Stay in ODD
31         #10 bit_in = 1;  // Transition to EVEN
32
33         #50 $stop; // Stop simulation
34     end
35
36     // Monitor outputs
37     initial begin
38         $monitor("Time: %0t | Bit In: %b | Parity Bit: %b | Current
               State: %s",
39                 $time, bit_in, parity_bit, uut.current_state.name());
40     end
41 endmodule
```

3 Results

3.1 Simulation

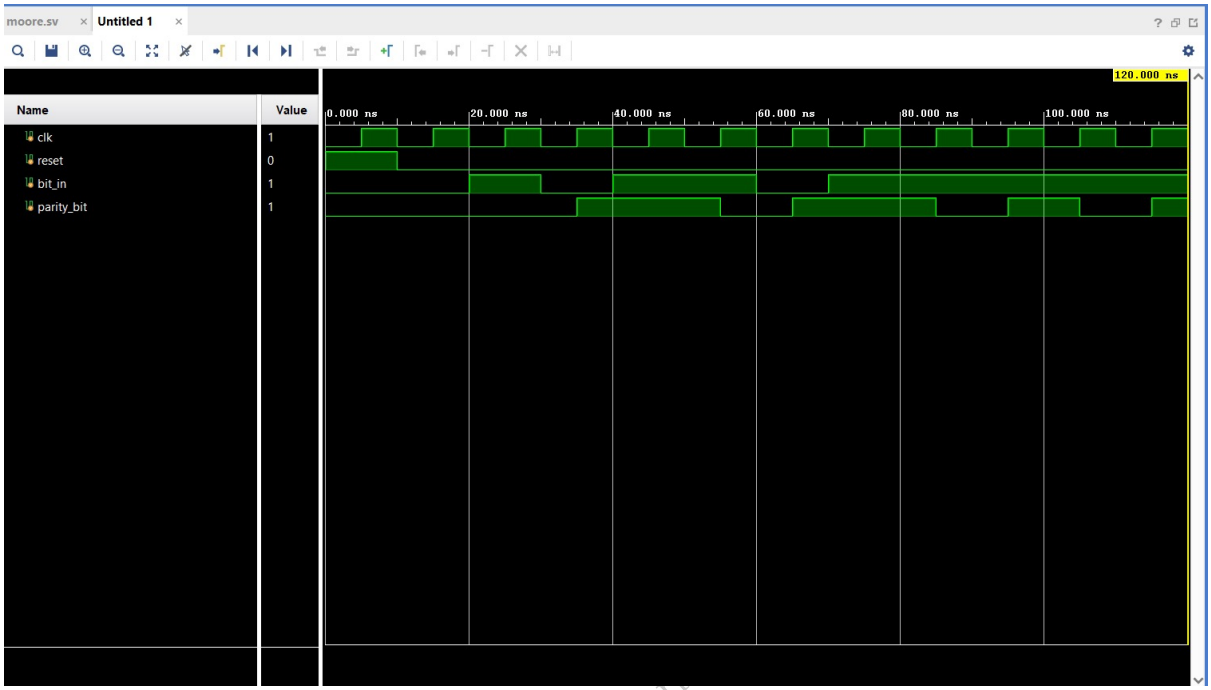

Figure 1: Simulation of Moore Machine for Even Parity Generator

3.2 Schematic

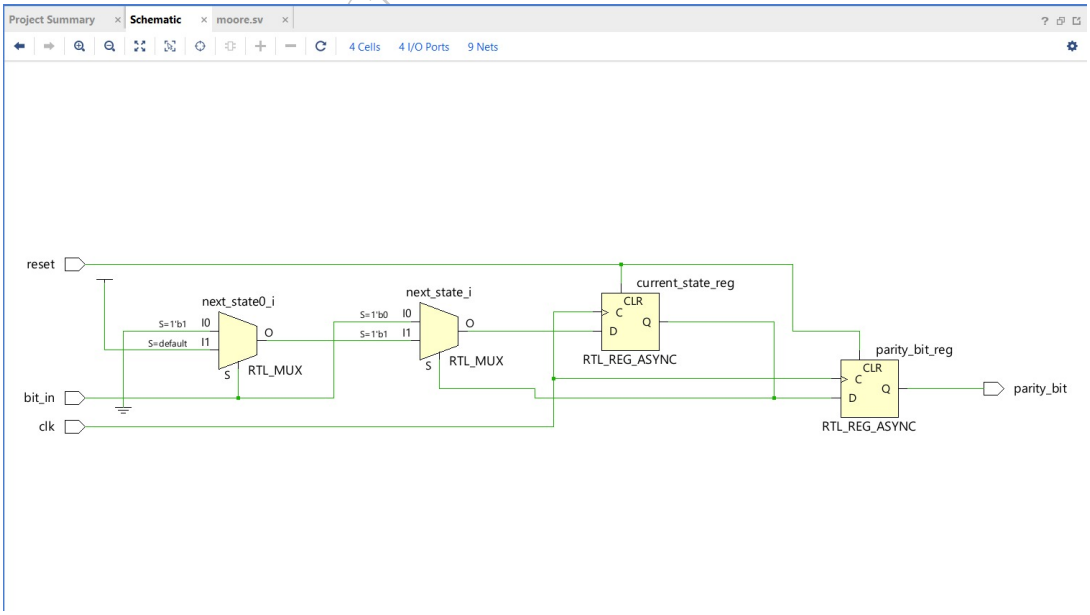

Figure 2: Schematic of Moore Machine for Even Parity Generator

3.3 Synthesis Design

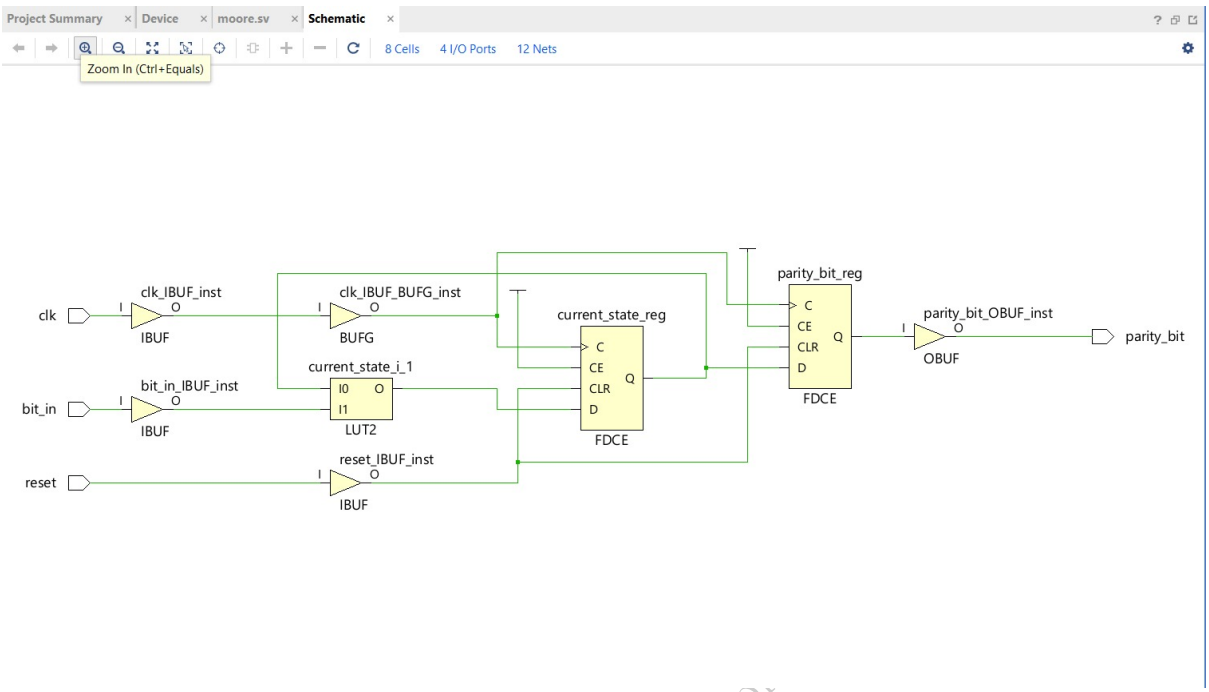

Figure 3: Synthesis Design of Moore Machine for Even Parity Generator

4 Advantages of Moore Machine for Even Parity Generator

- Outputs depend only on the state, ensuring consistency.
- Simple and easy to design and implement.
- Ideal for real-time error detection in communication systems.
- Robust with clearly defined states, reducing glitches.
- Scalable for additional parity rules or FSM integration.
- Hardware-efficient, suitable for low-power applications.

5 Disadvantages of Moore Machine for Even Parity Generator

- **Latency:** Output changes only after state transitions, causing a delay.
- **Inflexibility:** Outputs depend solely on the state, limiting dynamic behavior.
- **Complexity for Large Inputs:** Requires more states for complex parity systems.
- **Hardware Overhead:** May need additional resources for state storage and transitions.
- **Limited Adaptability:** Not ideal for scenarios requiring direct input-output dependency.

6 Applications of Moore Machine for Even Parity Generator

Error Detection in Communication Systems:

- Generates parity bits to detect errors during data transmission.

Data Integrity Verification:

- Ensures the correctness of data in storage and transmission.

Digital Systems Design:

- Used in designing synchronous circuits requiring parity-based logic.

Network Protocols:

- Implements parity checks in low-level communication protocols.

Educational Tools:

- Demonstrates finite state machine principles in digital electronics courses.

Embedded Systems:

- Provides lightweight parity solutions for resource-constrained environments.

7 Conclusion

The Moore Machine for an Even Parity Generator is a robust and efficient solution for real-time parity generation, ensuring error detection in digital communication systems. Its state-dependent output provides consistency, simplicity, and ease of implementation, making it ideal for embedded systems and hardware designs. While it has limitations, such as latency and scalability challenges, its advantages outweigh these drawbacks for many applications. By leveraging tools like Verilog/VHDL and simulation platforms, the Moore Machine demonstrates the power of finite-state machines in solving practical problems in digital electronics and communication.

8 FAQs

What is a Moore Machine?

- A Moore Machine is a type of finite-state machine where outputs depend only on the current state, not the input.

What is an even parity generator?

- It generates a parity bit to ensure the total number of 1s in a binary sequence (including the parity bit) is even.

How does a Moore Machine generate even parity?

- The machine tracks the number of 1s in the input using two states: S0 (even) and S1 (odd), with outputs determined by the current state.

Where is it used?

- In error detection for communication systems, digital design, and embedded systems.

What are its advantages?

- Consistent output, simple design, low resource requirements, and suitability for real-time systems.

Are there any drawbacks?

- Output latency, limited adaptability, and potential complexity for larger systems.

How is it implemented?

- Using hardware description languages (e.g., Verilog, VHDL) or state diagrams and synthesizing for hardware like FPGAs.

Why choose Moore over Mealy?

- Moore machines offer simpler, state-driven outputs, reducing risks of glitches compared to input-dependent Mealy machines.