4-bit binary counter.

Daniel Josué Rodríguez Agraz
Communications and Electronics Enginnering, Unniversidad de Guadalajara

***Abstract*—This document presents the design and verification process of a 4-bit binary counter digital circuit. The design and verification were conducted using the SystemVerilog hardware description language, with simulation and compilation performed in Questasim software. The implementation for the design was done using a state machine approach. Additionally, a test bench was developed to verify each instance and the final design, demonstrating the successful implementation of the circuit.**

I. Objectives

- Use Verilog to design and simulate a 4-bit binary counter.
- Implement a finite state machine for circuit design.
- Design a test bench for the circuit verification.

II. Introduction

A counter is a digital circuit responsible of counting electronic events, such as pulses, by progressing through a sequence of binary states. An n-bit binary counter is made of n flip-flops and is able to count to up to $2^n - 1$. We can divide digital counters into two main categories; synchronous and asynchronous [1].

A. Asynchronous Counter

An asynchronous counter is a digital counter where the internal flip-flops are connected in a cascade configuration. In this arrangement, the clock input for each flip-flop is derived from the output of the previous flip-flop, rather than all flip-flops sharing the same clock signal. This results in a ripple effect, where the change in state propagates through the flip-flops with some delay, which can limit the speed of the counter [1].

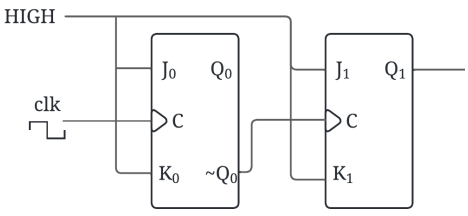


Fig. 1: 2 bit asynchronous counter.

B. Synchronous Counter

The synchronous digital counter has all of its flip-flops receiving the same clock signal simultaneously. Since there is no ripple or propagation delay between the flip-flops, this type of counter is significantly faster than an asynchronous counter [1].

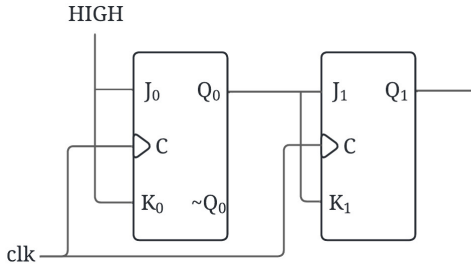


Fig. 2: 2 bit synchronous counter.

III. Methodology

This project was developed using the SystemVerilog hardware description language, with Questasim software used for compilation and simulation. The project began with the definition of the counter's inputs and outputs, which are represented in Figure 3. The counter features a preset value, which sets the starting point of the count and the value to which the counter returns after completing a full cycle. In addition to the preset, a reset input was implemented to restart both the count and the preset value.

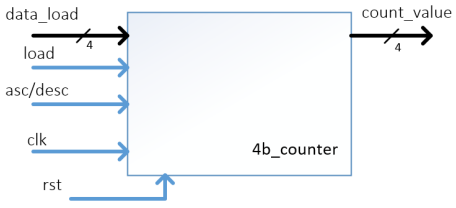
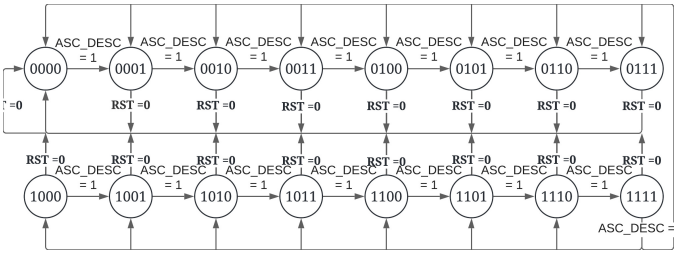
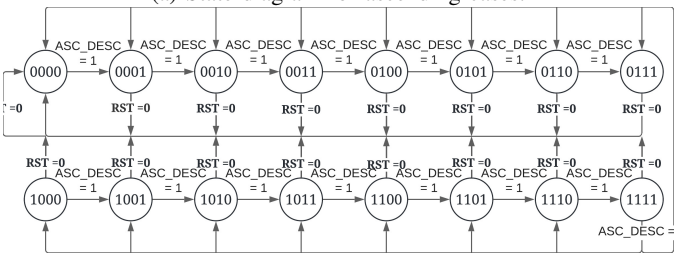


Fig. 3: Representation of a 4-bit binary counter.

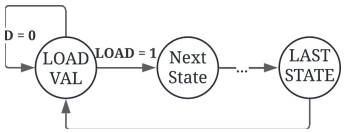
Figure 4 illustrates the flow diagrams for the state machine implemented in the counter. Two separate state machines were used for this implementation, as shown in the state diagram: one state machine manages the ascending count, while the other handles the descending count. The transitions between these states are governed by control variables, which determine whether the counter switches between counting up or down.



(a) State diagram for ascending cases.



(b) State diagram for descending cases.



(c) State diagram for LOAD signal.

Fig. 4: State diagrams for 4-bit digital counter.

Below the implementation in verilog of the 4-bit digital counter is shown.

Listing 1: 2 bit full adder code hardware description in verilog.

```
module Counter_4B(
  input reg [3:0] D_in,
  input wire LOAD, ASC_DESC,
  input wire clk, rst,
  output reg [3:0] COUNT
);

reg [3:0] D;
reg [3:0] next_state, present_state;

assign COUNT = present_state;

localparam [3:0]   s0 = 4'h0,
                   s1 = 4'h1,
                      .
                      .
                      .
                   s15 = 4'hF;

always @(posedge clk or posedge rst)
begin
  if (rst)
  begin
    present_state <= 'h0;
  end
  else if (~LOAD)
  begin
    D <= D_in;
    present_state <=D;
  end
  else
  begin
    present_state <= next_state;
  end
end

always @(*)
begin
// Ascending
if (ASC_DESC)
begin
  case (present_state)
        s0:  next_state <= s1;
        s1:  next_state <= s2;
                .
                .
                .
        s15: next_state <= D;
        default: next_state <= D; // Default state
    endcase
    end
    else // Descending
    begin
    case (present_state)
        s0: next_state <= s15;
        s1: next_state <= D;
              .
              .
              .
        s15: next_state <= s14;
        default: next_state <= D;
  endcase
end
end
endmodule
```

IV. Results

For the verification process, we loaded several preset values, as shown in Table **??**, and applied a range of test cases to validate the counter's functionality. These test cases involved altering the states of the control signals, such as the reset, reload, ascending, and descending states. By manipulating these control values, we ensured that the counter behaved as expected under different conditions, including resetting to the preset value, counting up and down correctly, and reloading when necessary.

In the Figures 5 and 6 , the value '0' is loaded as the preset, and the ascend signal is low, resulting in a descending output.

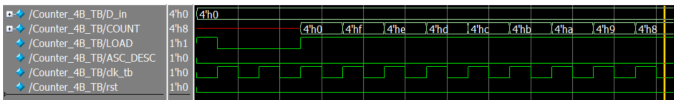


Fig. 5: Simulation results for a ’0’ preset

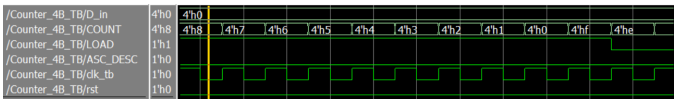


Fig. 6: Simulation results for a ’0’ preset

To verify the correct operation of the reload function, a reload was performed. The result shows that the counter correctly returned to the preset value of ’0’, as shown in Figure 7, with the ascend signal still low, ensuring a descending output.

Fig. 7: Simulation results for a 0 preset and a reload of 0.

Next, in Figure 8, the preset value was set to ’7’, and the descend operation was verified.

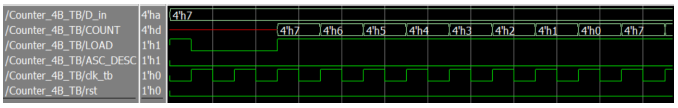


Fig. 8: Simulation results for a 7 preset.

After loading a new value, a reload was conducted to ensure the counter correctly returns to the preset. Figure 9 shows the counter reloading a value of ’a’ after the previous ’7’ preset, and a change to an ascending operation.

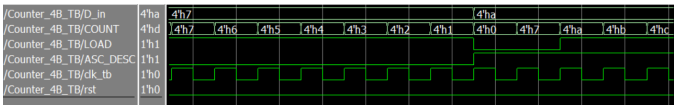


Fig. 9: Simulation results for a 7 preset and a reload of a value a.

In Figures 10 and 11, the counter was tested with an ascending count, with a preset value of ’a’ and the reset signal active, verifying the counter’s ability to start correctly after a reset.

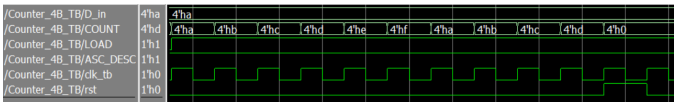


Fig. 10: Simulation results for an ascending count with an ’a’ value preset, and a reset signal

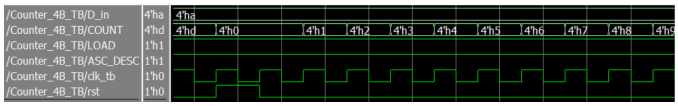


Fig. 11: Simulation results for an ascending count with an ’a’ value preset, and a reset signal.

Lastly, after the reset, a reload was performed with the preset value of ’a’. The simulation results are presented in Figure 12, confirming the correct operation of the reload and reset functions.

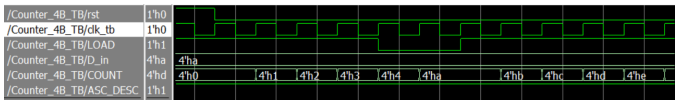


Fig. 12: Simulation results for an ascending count with an ’a’ value preset, and a reset signal.

V. Conclusions

n this project, we successfully designed and implemented a 4-bit binary counter using the SystemVerilog hardware description language. The design utilized two state machines to manage both ascending and descending counts, and included a preset and reset functionality to provide flexibility and control over the counting process. The entire design was thoroughly verified through simulations in Questasim, ensuring correct operation under various conditions. This approach, combined with the implementation of a testbench, demonstrated the effectiveness of the design.

References

[1] T. L. Floyd, *Fundamentos de Sistemas Digitales*, 9th ed.