Mod Two-digit Up-Down Counter

Special Assignment and Lab Project Report

Course: 2EC201CC23: Digital Logic Design

Submitted by:

23BEC168
[Heer Sadiwala]

23BEC169
[Meet Sakariya]

Department of Electronics and Communication Engineering,

Institute of Technology,

Nirma University,

Ahmedabad 382 481

[November- 2024]

ABSTRACT:

A 2-digit up-down counter, a crucial component of digital electronics, is demonstrated in this project. It is useful for tasks like inventory tracking, event counting, etc. The primary objective is to develop a counter that allows users to freely choose the count direction by counting both up and down between 00 and 99.

Our design addresses the problem of accurate counting in both directions. We have developed a straightforward counter by utilizing fundamental digital logic component. As user input controls whether the counter rises or falls, it is very user friendly. A small, simple design is ensured by the efficient use of minimal hardware components.

Contents:

INTRODUCTION:

Description:

- Digital counters are key building blocks in electronics. It plays a important role in systems that need to count or keep track of sequences like clocks, displays, event counters, etc.
- This project focuses on creating a 2-digit up-down counter, a device that counts both up and down within a range of 00 to 99, allowing users to control the count direction.
- A 2-digit up-down counter is a simple circuit that increases or decreases a count displayed on a 7-segment screen. It uses basic digital components like 7-segment, and various ICs.
- The circuit is designed with the main aim of keeping the count stable and reliable.
- Users can easily switch between counting up or down, and the count updates in real-time on the display.

Applications:

- Industrial automation: Used to keep track of production counts, inventory, etc.
- Digital clocks and timers: Essential in creating counters for displaying seconds, minutes, or hours.
- Event counting: Useful in sports, or any event where items or occurrences are need to be counted.
- Consumer electronics: Found in microwave ovens, washing machines, and other devices where countdown or count-up functionalities are needed.

Literature Survey/State of the Art Technology Available:

There are several existing technologies and approaches that perform the same function as a 2-digit up-down counter.

1. **Microcontroller-Based Counters**
   Microcontrollers like Arduino, and Raspberry Pi can be programmed to function as up-down counters. These systems use software code to manage counting operations. They can include buttons, and sensors, but may be overpowered for simple counting needs.

- *Example:* Arduino-based counters can count up or down based on digital input (e.g., a button press) and display the count on a 7-segment display.

2. Digital Signal Processor (DSP) Modules

Digital Signal Processors can act as counters, especially in applications that require high-speed counting. DSPs are often used in audio processing, telecommunications, and high-speed industrial applications. Although more complex, they're useful as counters with high precision and high-frequency requirements.

- *Example:* A DSP-based counter might be used in systems requiring precise timing and high-speed operations, such as RF (radio frequency) counting.

3. Smartphone Apps
Several smartphone applications performs the functionality of counters and can be used for event counting or time-tracking purposes. While they are not hardware-based counters, these apps are convenient for quick counting tasks.

- *Example:* Apps like Tally Counter allow users to count up and down by tapping the screen and can be used to track items or events in a digital format.

Proposed Solution/Methodology:

To fulfil the need of an efficient counter circuit, the 2-digit up-down counter is designed with the following primary goals:

- Accuracy
- User Control
- Compact and Efficient Design

Methodology:

1. Circuit Design and Simulation:

- We've designed a schematic counter circuit, adding the user control to manage the count direction.
- We've checked and simulated the circuit in simulation tool "Tinkercad" to validate the logic, identify issues, and recorrect the circuit before physical assembly.

2. Testing:

- We've assembled the circuit on a breadboard and checked particularly the counting accuracy and display functionality.

3. Finalization

- After checking the circuit, we finalized the circuit and ensure that the circuit is durable and user-friendly.

Justification for approach:

1. Simplicity and Cost-Effectiveness:

- Microcontrollers like Arduino or Raspberry Pi can be relatively expensive and overpowered for basic counting tasks. It doesn't require programming or complex setup.

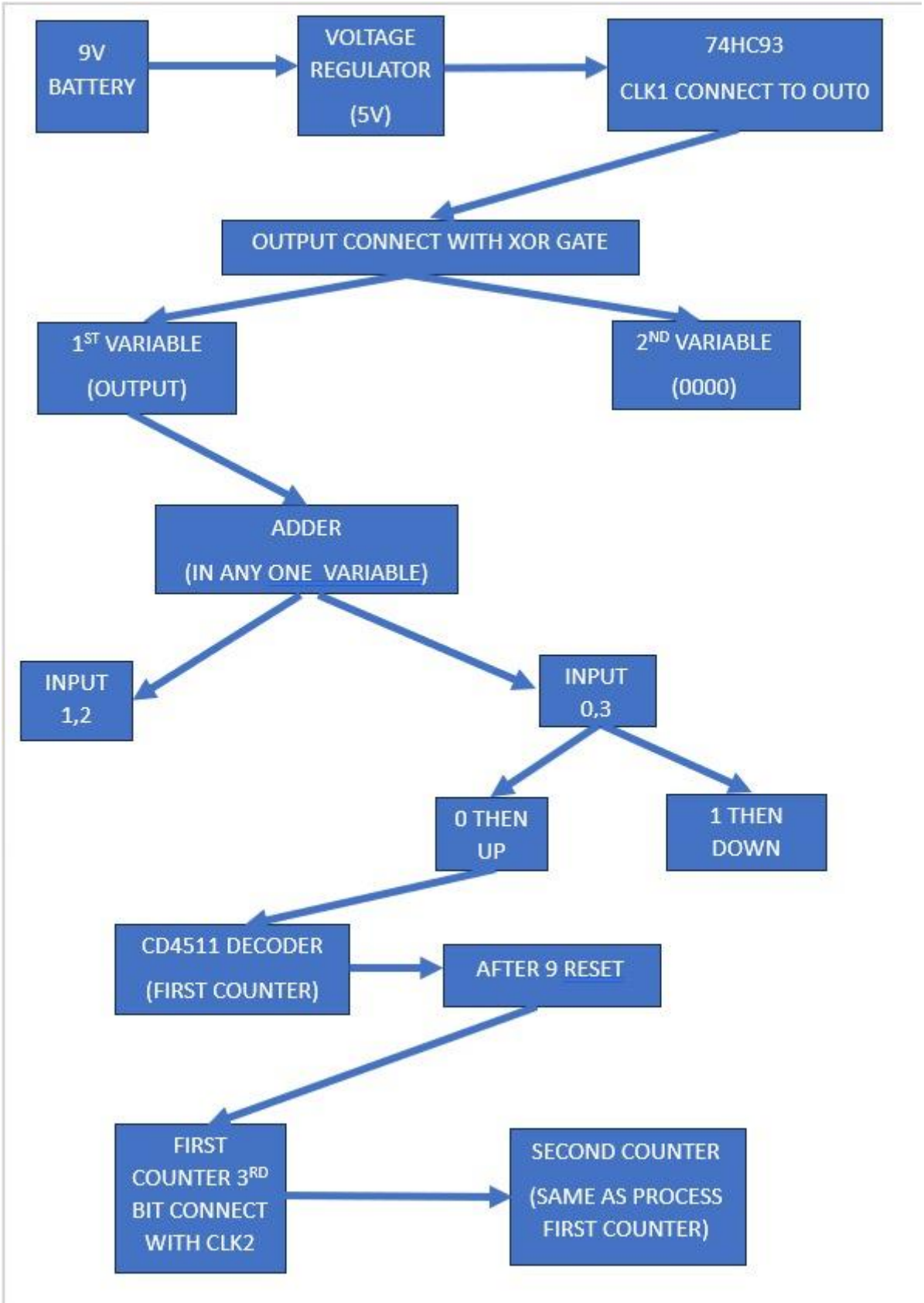
2. Application-Specific Design:

- DSPs are optimized for high-speed and complex data processing. A 2-digit counter is more appropriate for low-frequency and straightforward counting.

3. Physical Presence and Real-Time Interaction:

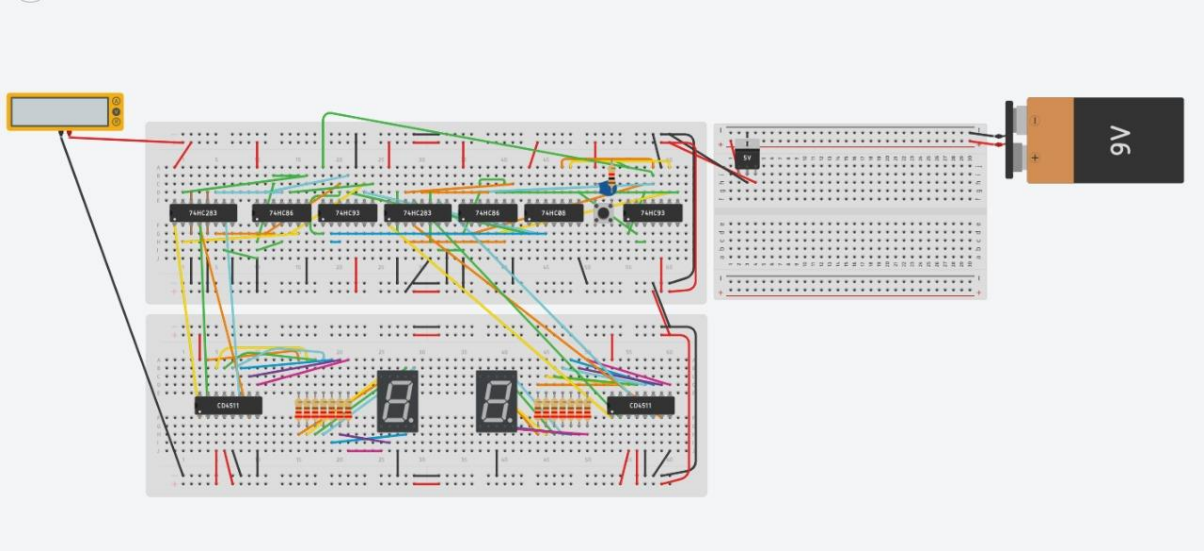
Unlike smartphone applications, a physical 2-digit up-down counter allows users to interact directly with the device, which is important for practical hands-on learning.

Block Diagram & Circuit Diagram:



Block Diagram of a Mod 2-digit Up-Down Counter

Figure 1: Block Diagram of the system

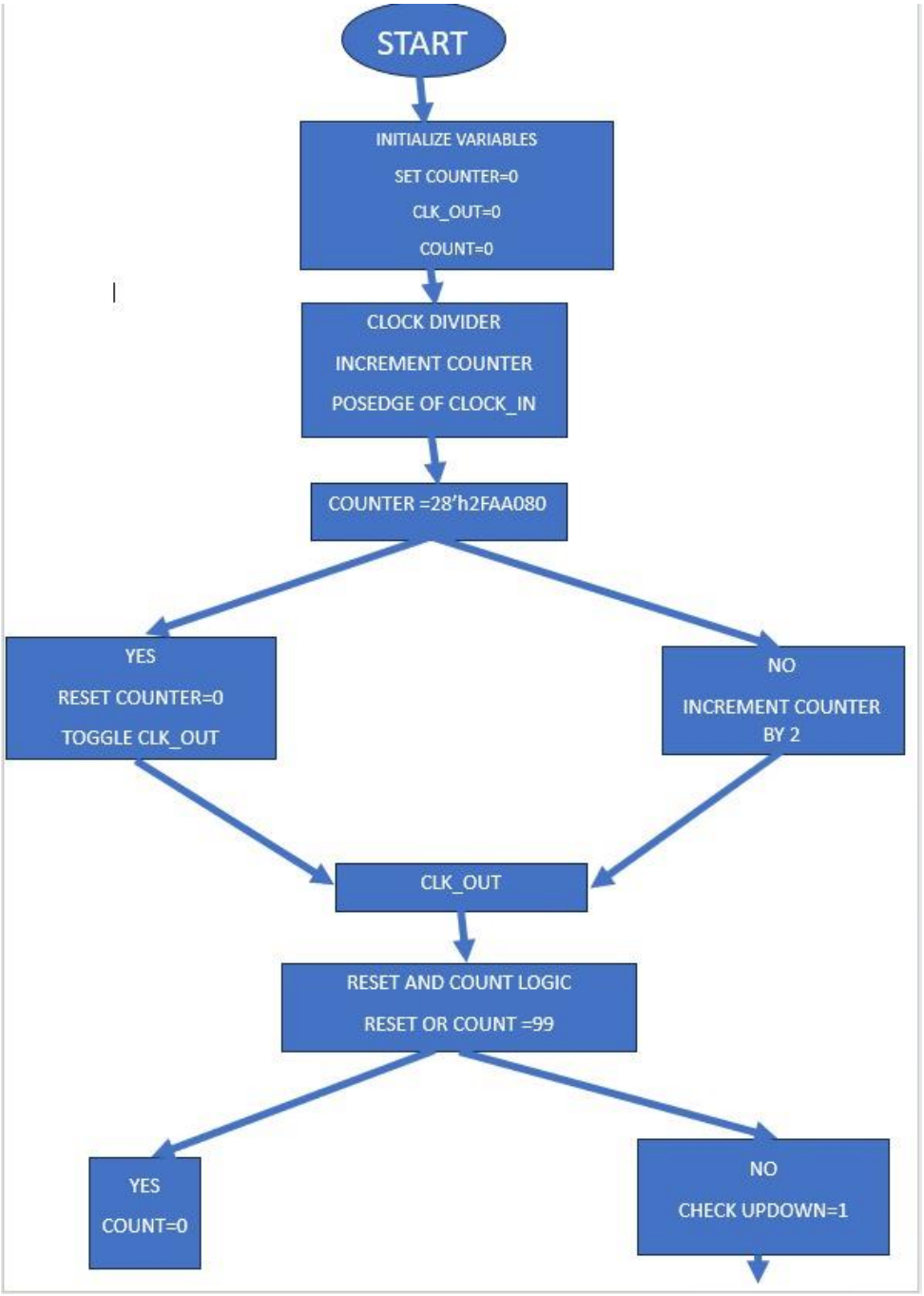


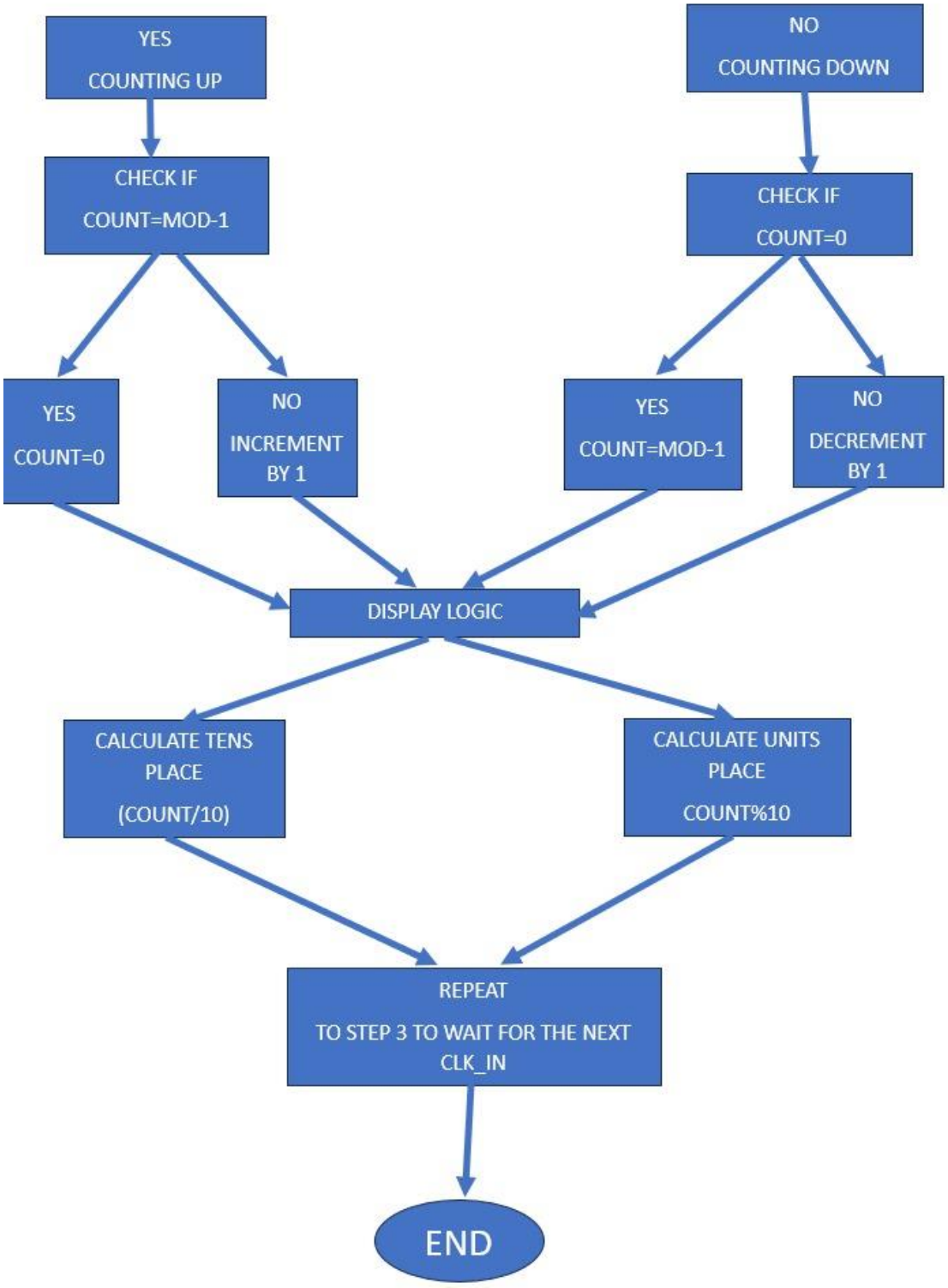
Circuit Diagram of a Mod 2-digit Up-Down Counter

Figure 2: Circuit Diagram of the system

Design:

- Flowchart: The following is the flowchart of the code for the choosen system. It breakdowns how actually the code proceeds to meet the desired output of the system.


START
INITIALIZE VARIABLES
SET COUNTER=0
CLK_OUT=0
COUNT=0
CLOCK DIVIDER
INCREMENT COUNTER
POSEDGE OF CLOCK_IN
COUNTER =28'h2FAA080
YES
RESET COUNTER=0
TOGGLE CLK_OUT
NO
INCREMENT COUNTER
BY 2
CLK_OUT
RESET AND COUNT LOGIC
RESET OR COUNT =99
YES
COUNT=0
NO
CHECK UPDOWN=1



Flowchart of a Mod 2-digit Up-Down Counter's Code

Figure 3: Flowchart of the code

Details of All Hardware Components Used and Its Justification:

1) CD4511 (BCD to 7-Segment Latch/Decoder/Driver): **Reduces the complexity** of wiring and has built-in **latching capability**, meaning it holds the output until new data is provided, ensuring stable display readings.

2) 74HC283 (4-bit Binary Full Adder): used to **add or subtract values** based on the control inputs, which is useful for counting up or down in the sequence.

3) 74HC86 (Quad 2-Input XOR Gate): useful for **direction control (up/down counting)** by toggling the count direction based on a control input.

4) 74HC93 (4-bit Binary Counter): Can be easily used to form multi-digit counters, making it suitable of counting from **0 to 15** in binary.

5) 74HC08 (Quad 2-Input AND Gate): Helps in generating **control signals** for the binary counter.

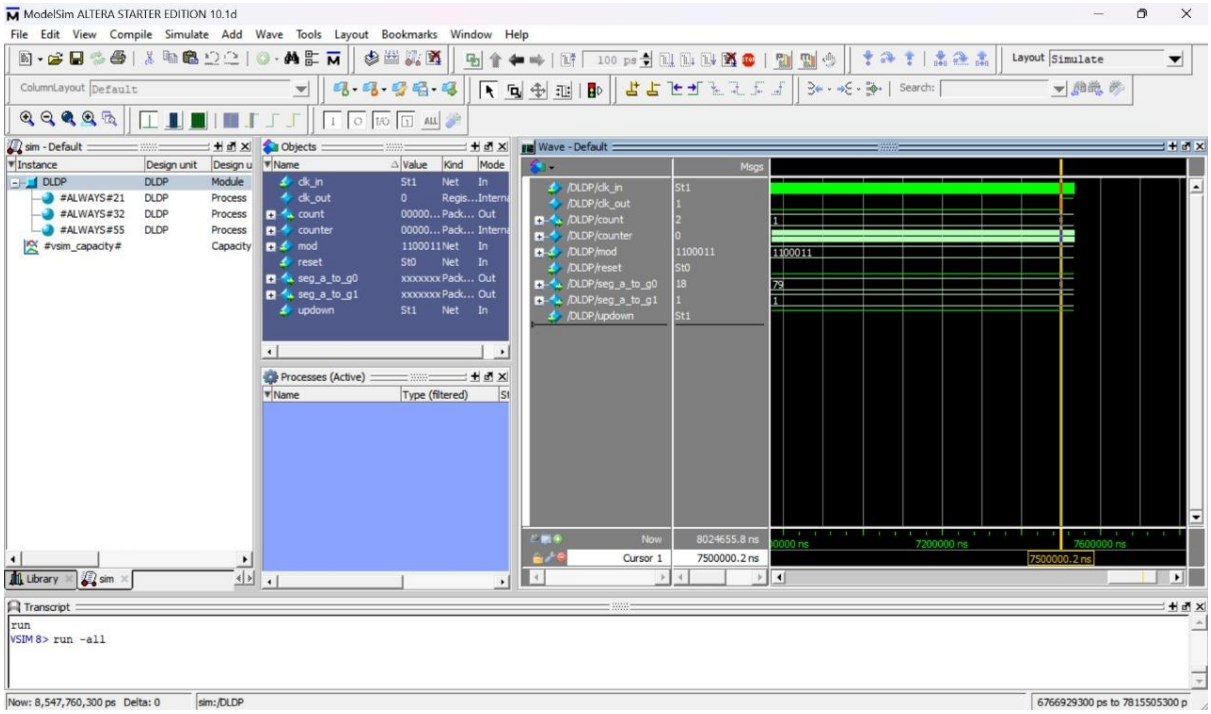
6) 9V Battery: The components used typically operate within a voltage range of **2V to 6V** or **5V to 15V**, so a **9V battery** is a convenient choice.

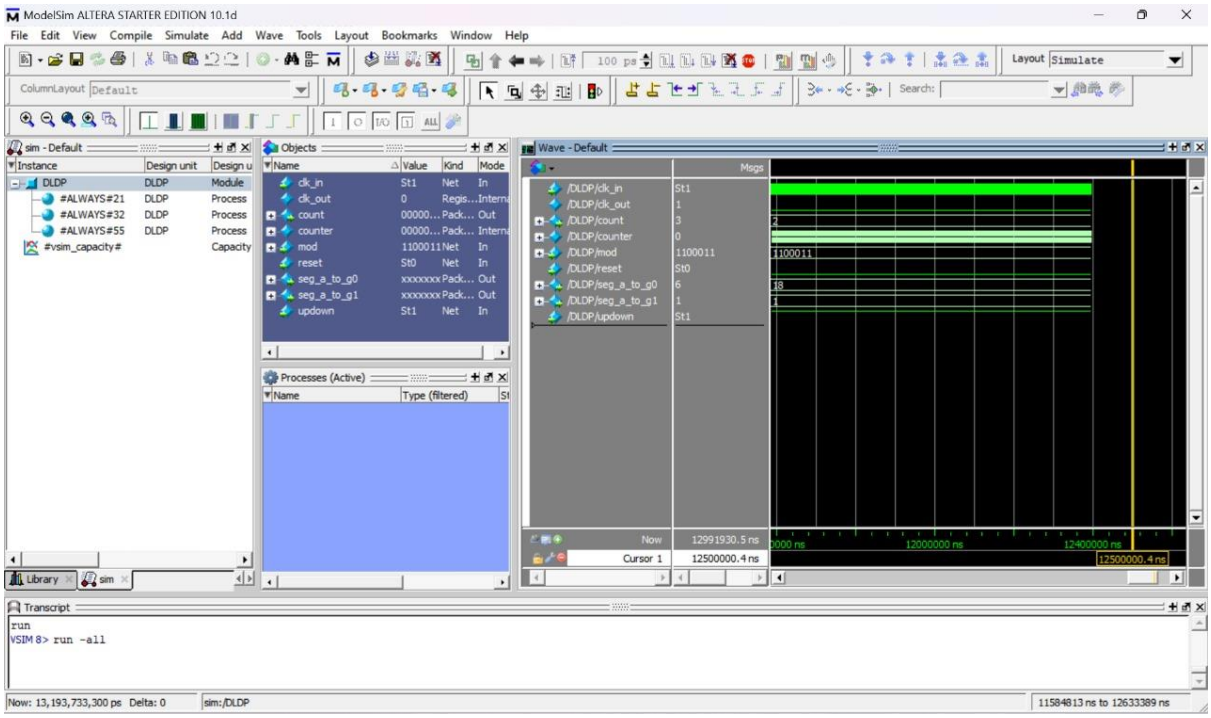
7) 7805(Voltage Regulator): It Converts the 9V battery Voltage to 5V Voltage. Voltage regulators (VRs) keep the voltages from a power supply within a range that is compatible with the other electrical components. While voltage regulators are most commonly used for DC/DC power conversion, some can perform AC/AC or AC/DC power conversion as well.

8) Two 7-Segment Displays: Used to **display the numerical output** of the counter in a human-readable form.

Simulation Results:

- Modelsim Simulation:

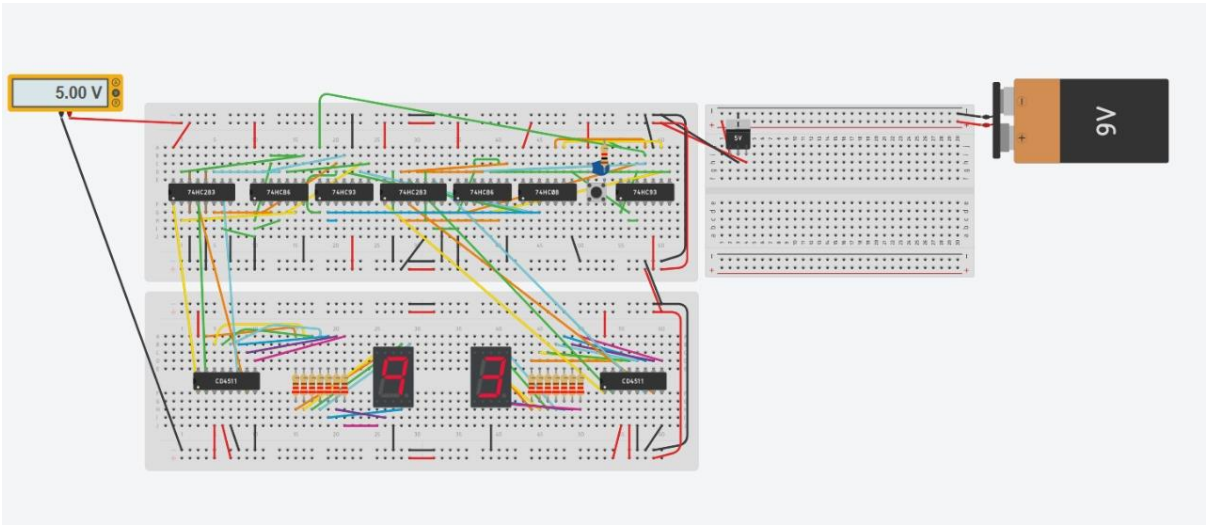




Modelsim simulation of the system

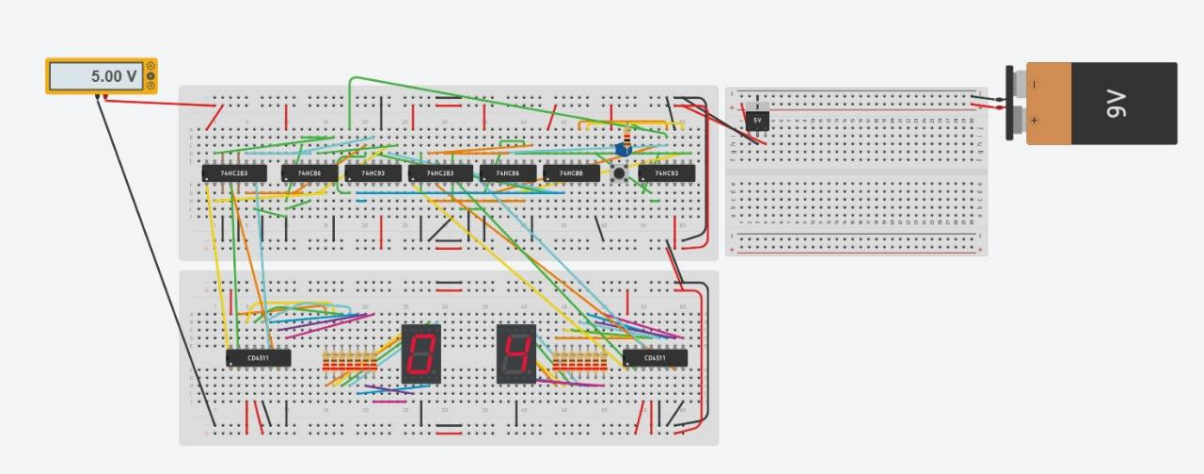
Figure 4: Modelsim simulation

- Circuit Simulation:



Circuit Diagram of a Mod 2-digit Up-Down Counter (down counting)

Figure 5: Circuit Diagram of the system



Circuit Diagram of a Mod 2-digit Up-Down Counter (up counting)

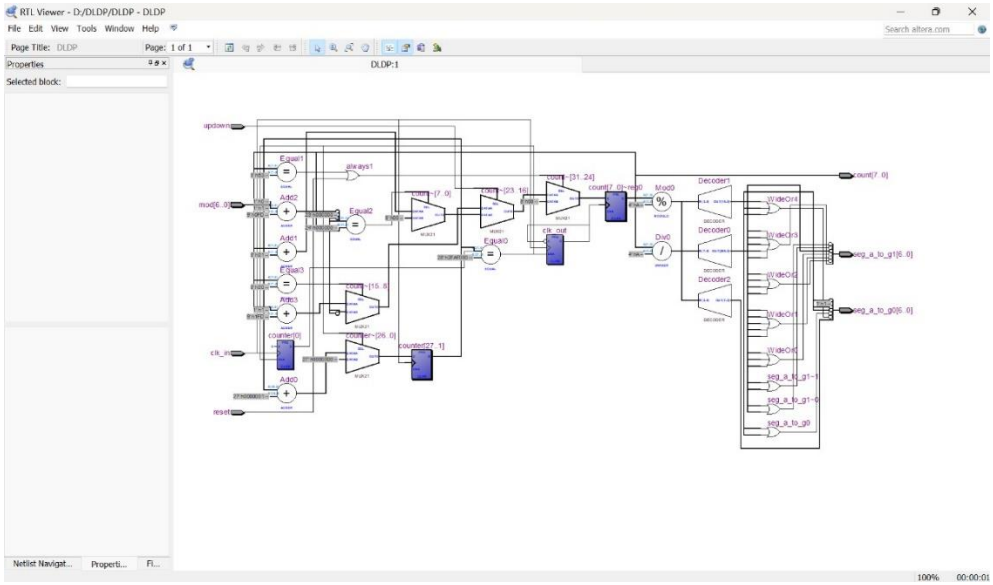
Figure 6: Circuit Diagram of the system

- Simulation plays a crucial role in verifying its functionality before implementing it in hardware such as:

- Validation of the Counter's Behavior
- Verification of Control Inputs
- Timing Analysis
- Functional Verification of Output
- Debugging and Error Correction

Results and Measurements:

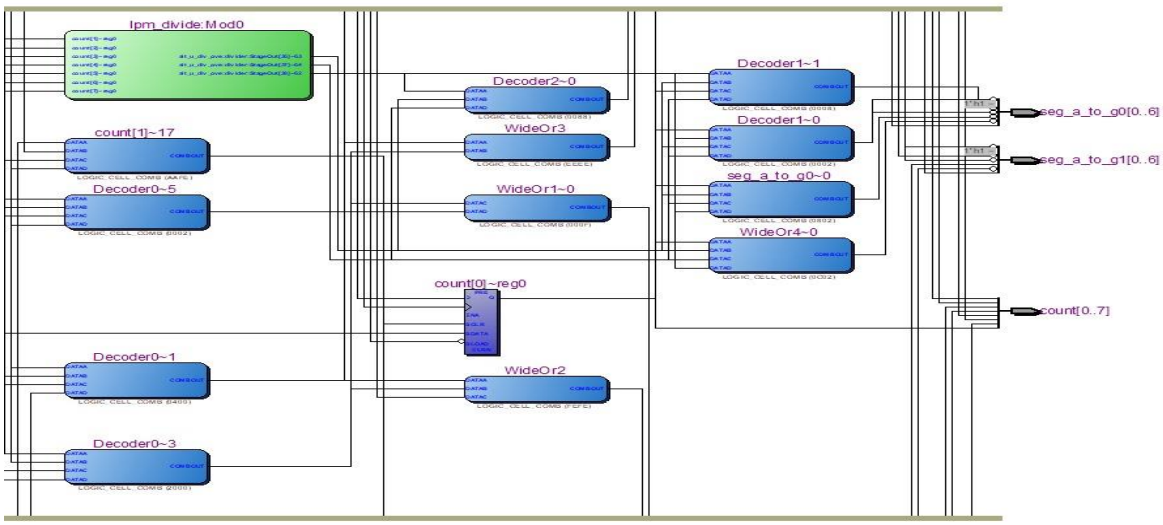
RTL View:



RTL View of a Mod 2-digit Up-Down Counter

Figure 7: **RTL View** ***of the system***

Technology Map View:



Technology Map View of a Mod 2-digit Up-Down Counter

Figure 8: Technology Map View of the system

 The RTL view is a high-level abstraction of the design, showing how data flows between components and the operations performed on the data.

 The Technology Map View shows how the high-level RTL design is translated into actual hardware components on the FPGA.

FPGA Implementation:

The following actions are performed in order to implement the code on FPGA:

To implemented the design on the FPGA, We've :

- Assigned I/O pins.
- Generated a .sof file to run the FPGA.
- Downloaded the file to the FPGA board.
- Connected the board, apply power, and observe the counter's output.
- Tested the up/down count functionality.

Circuit Implementation:

For this implementation, We've:

- Drawn a diagram of the 2-digit up-down counter using components like ICs, resistors, battery, and 7-segment displays.
- Assemble the circuit on a breadboard.
- Test the counter functionality.

Conclusion, Learning Outcome, and Future Scope:

- Conclusion:

The 2-digit up-down counter project was successful because it was able to design, implement, and test the given circuit as working digital counter circuit. It was possible to count upwards and downwards directions as well as to display the result on a 2-digit 7-segment display. It behaves exactly with regard to mode selection (up/down). This design received a

stable and glitch-free counting mechanism. This reflects good understanding of digital logic principles.

- Learning Outcome:

By the end of this project, we gained a better understanding of the following concepts:

1) Gain practical experience in constructing circuits.
2) Understand binary counting sequences.
3) Applying concepts such as counters, and logic gates.
4) Enhance the ability to analyze and debug digital systems.
5) Designing control logic for counting and its direction.

- Future Scope:

There are several ways the present design of the 2-digit up-down counter can be improved and expanded

1) More Digits Expansion : The counter could be extended for more digits and support higher ranges.
2) User Programmable Range of Counting: Future design may include features for user to define both the upper and lower limits of counting.
3) Variable Frequency Clock Source : Addition of a variable clock source will allow the user to vary the counting speed
4) Real-World Applications: The counter can easily be modified to application software in digital timers, event counters or score boards.

These modifications will lead to valuable learning experience which will further enhance the understanding of digital design principles.

Appendix:

The following is the Verilog code for the system:

```
module DLDP(
    input  wire clk_in,           // Input clock signal (50 MHz)
    input reset,                  // Reset signal to initialize the counter
    input updown,                 // Control signal to determine counting direction (up or down)
    input [6:0] mod,              // Modulus value for counting (max count value)
    output reg [7:0] count,       // 8-bit output for the current count value
    output reg [6:0] seg_a_to_g1, // 7-segment display output for the tens place
    output reg [6:0] seg_a_to_g0  // 7-segment display output for the units place
);

 reg clk_out;                    // Register to hold the divided clock output
 reg [27:0] counter;             // 28-bit counter for clock division

 initial
begin
    counter = 0;                 // Initialize counter to 0
    clk_out = 1'b0;              // Initialize the divided clock output to low
    count = 8'b0;                // Initialize count to 0
end

 always @(posedge clk_in)       // Trigger on the rising edge of the input clock
begin
    if (counter == 28'h2FAF080) // Check if the counter has reached the value for 1 Hz output
    begin
        counter <= 0;           // Reset the counter
        clk_out <= ~clk_out;    // Toggle the clk_out signal
    end
    else
        counter <= counter + 28'd2; // Increment the counter by 2
end
```

```
always @(posedge clk_out)       // Trigger on the rising edge of the divided clock
begin
    if(reset || count == 7'd99) // Check if reset is active or count has reached 99
        count <= 7'b0000000;    // Reset count to 0
    else
    begin
        if(updown == 1)         // If counting up
        begin
            if(count == mod - 1) // Check if count has reached the modulus value
                count <= 7'b0000000; // Reset count to 0
            else
                count <= count + 1; // Increment count by 1
        end
        else                    // If counting down
        begin
            if(count == 7'b0000000) // Check if count is at 0
                count <= mod - 1; // Set count to modulus - 1
            else
                count <= count - 1; // Decrement count by 1
        end
    end
end
```

```
always @(*) begin               // Combinational block to update the 7-segment display outputs
    // Tens place
    case (count / 10)           // Determine the tens digit of the count
        0: seg_a_to_g1 = 7'b0000001; // 0
        1: seg_a_to_g1 = 7'b1001111; // 1
        2: seg_a_to_g1 = 7'b0010010; // 2
        3: seg_a_to_g1 = 7'b0000110; // 3
        4: seg_a_to_g1 = 7'b1001100; // 4
        5: seg_a_to_g1 = 7'b0100100; // 5
        6: seg_a_to_g1 = 7'b0100000; // 6
        7: seg_a_to_g1 = 7'b0001111; // 7
        8: seg_a_to_g1 = 7'b0000000; // 8
        9: seg_a_to_g1 = 7'b0000100; // 9
        default: seg_a_to_g1 = 7'b0000001; // Off
    endcase

    // Units place
    case (count % 10)           // Determine the units digit of the count
        0: seg_a_to_g0 = 7'b0000001; // 0
        1: seg_a_to_g0 = 7'b1001111; // 1
        2: seg_a_to_g0 = 7'b0010010; // 2
        3: seg_a_to_g0 = 7'b0000110; // 3
        4: seg_a_to_g0 = 7'b1001100; // 4
        5: seg_a_to_g0 = 7'b0100100; // 5
        6: seg_a_to_g0 = 7'b0100000; // 6
        7: seg_a_to_g1 = 7'b0001111; // 7
        8: seg_a_to_g1 = 7'b0000000; // 8
        9: seg_a_to_g1 = 7'b0000100; // 9
        default: seg_a_to_g1 = 7'b0000001; // Off
    endcase
    end
    endmodule
```