**Verilog Example: Up-down Counter**

EE 316: Digital Logic Design

## Overview

This Verilog example project implements an up-down counter with reset using a high-level state machine (HLSM).

## Inputs

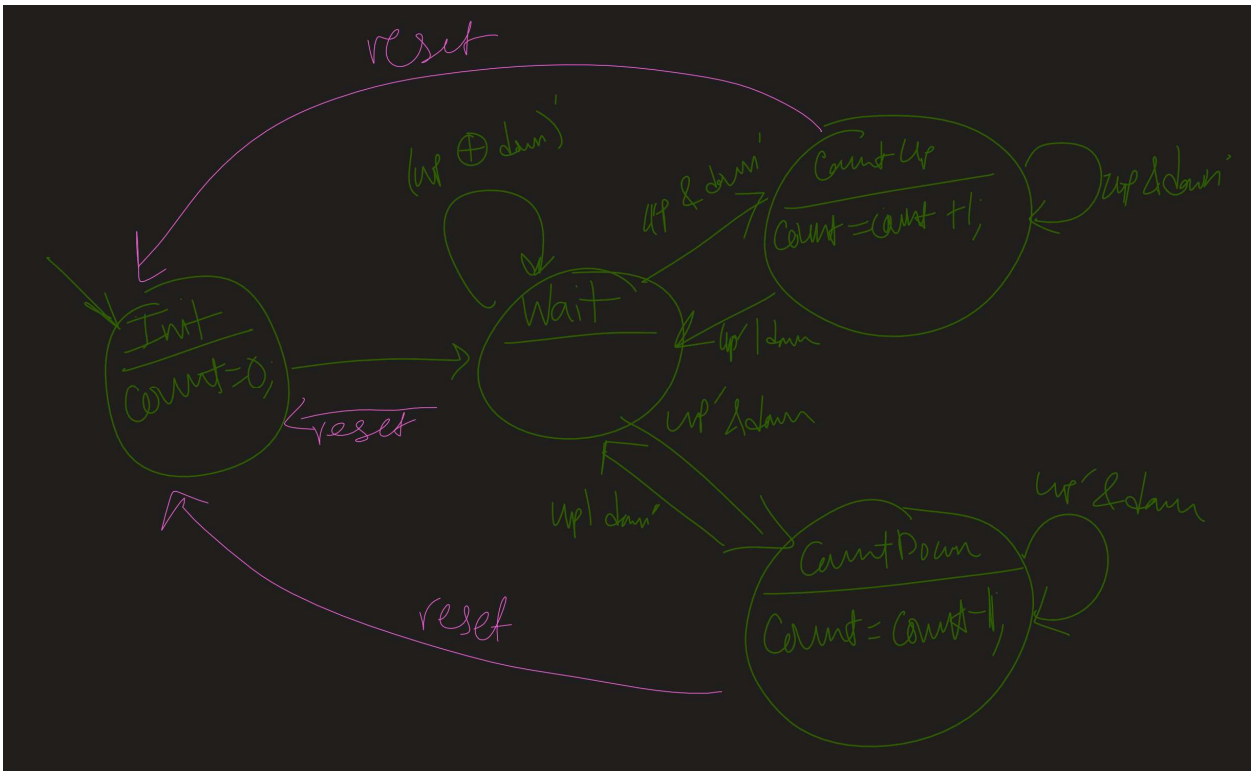
1. Clk – Clock signal, assumed to be 100 MHz
2. Up (btnU) – When asserted, the counter increments by 1.
3. Down (btnD) – When asserted, the counter decrements by 1.
4. Reset (btnC) – When asserted, the counter resets to 0.

When both up and down are asserted, the counter remains the same value.

## Outputs

1. Seven-segment display – Displays the current value of the counter in hexadecimal .using the anodes and segments.

## HLSM Diagram




The HLSM runs at 12.5 MHz with a divide-by-8 clock, and the seven-segment display is driven by a 50 MHz divide-by-2 clock.

**Notes on Style/Design**

1. There are two always blocks, one for the combinational logic and one for the sequential logic. This approach generally follows FSM/HLSM conventions that divide a system/processor into its combinational and sequential parts.
2. The reset is implemented in the sequential always block, not the combinational always block. This makes it a synchronous reset. An asynchronous reset would be placed in the combinational always block, likely using an if statement and driving the current state to a constant value.
3. The sensitivity list in the combinational always block now contains two sets of variables: the input buttons (except reset) and the current-state variables (`cs` and `count`).
4. It is important that you specify the output and next state in your procedural (always) blocks. This prevents inferred latches, which may cause unpredictable behavior.
    a. An HLSM diagram can be misleading because it may not show that there is an output for a state (as in the wait state in the HLSM above), but that simply implies that the output remains the same as an input (see next note).
5. The count register is implemented as a load register with a current `count` reg and a `nextcount` reg. This creates a load register structure so that count is updated synchronously. If it were updated asynchronously, a combinational loop would occur, as the counter would increment infinitely many times in a very small amount of time.
    a. An HLSM diagram can be misleading because it may not reflect the need for a load register structure (as in the HLSM above). However, if you see HLSM outputs assigning new values to themselves (e.g. `count = count + 1;`), that generally indicates the need for a load register structure.