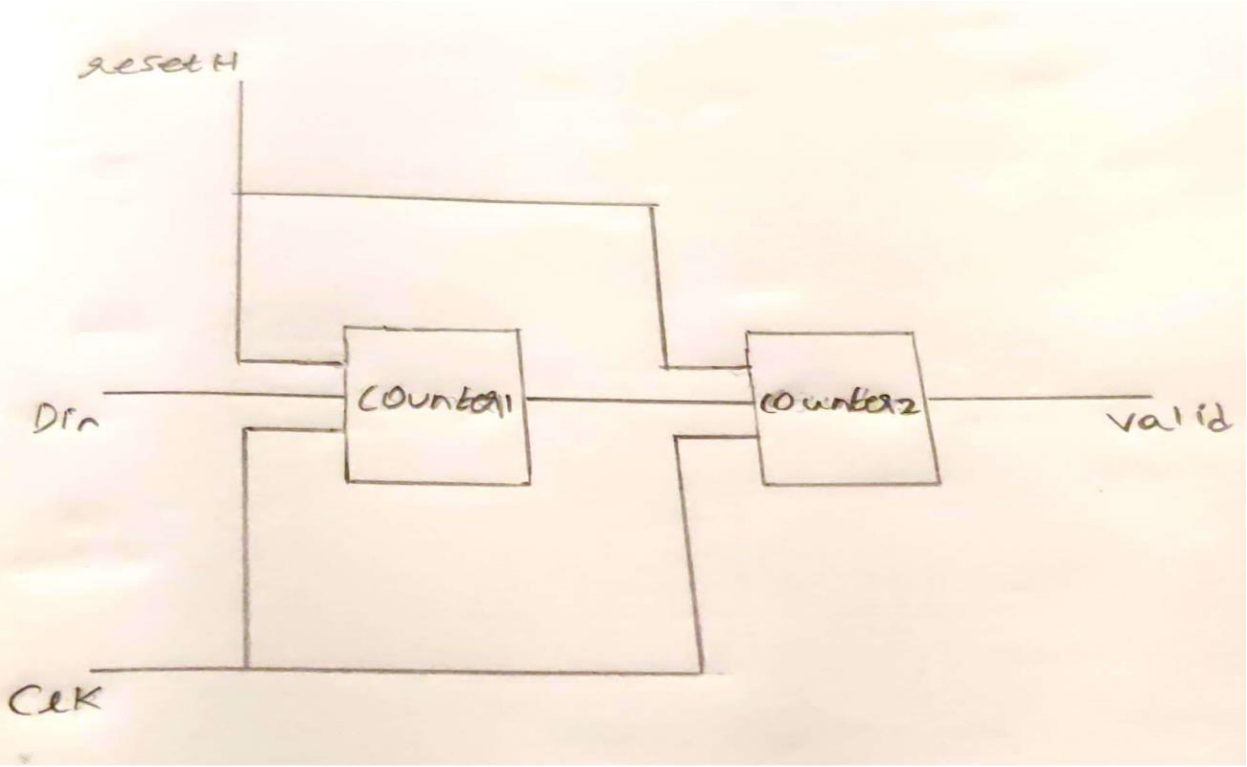
Design Report for SerialTOFED:-

Block diagram for SerialTOFED:-



The Design uses two counters, one which will increment for every posedge of a clock cycle and counts the number of bits and the other counter counts the number of 1bits received.

The testbench is a self-checking one as it compares the output of the design with the result from the TB. Here the **data(din)** is generated as a datastream based on the **FBIBBLE_SIZE** and number of testcases(32+1 dummy). The data is then grouped into 5 bits based on **FBIBBLE_SIZE** parameter and then the ones are counted with the **countones** function of system Verilog. The TB checks if the output of the design matches with the desired output and if not, it points the error for that case. We also use a package **SerialTOFEDDefs** which contains the global definitions for serial TOFED problem.

My design works correctly because, when the **resetH** is asserted high, the system is reset and the **data(din)** starts flowing for active low reset. For every posedge of a clock, the **data(din)** flows 1 bit. So here, counter 1 just increments **temp** variable each time the **clk** encounters a positive edge. The counter 2 checks if the **data(din)** is equal to 1'b1. If it is equal, then it increments the **one** variable.

Since a **fbibble** is a sequence of **5 bits**, the design checks the value of **one** variable for every multiples of 5 and then reset the **one** variable to 0. Before resetting the **one** variable, it also checks if the value of **one** variable is equal to 3. Since a fbibble contains **3 bits as logic high and 2 bits s logic low**. So, if the value of **one** variable is equal to 3, it sets the **valid** port to **TRUE** else it is set to **FALSE**. From this we can check if the data we are receiving is a fbibble or not.

Here we are also using **enum** function for **valid** port to generate TRUE or FALSE boolean outputs.