Homework #3

1. Write the Verilog code for a sequential circuit that counts the following sequence in 4 bits:

0,5,A,4,2,D,7,3,C and then starts over

Inputs should be 4 bit starting count and the CLK, Outputs should be the 4 bits of the count

[3:0] Cstart for the initial count and [3:0] Count for the count sequence

You will have to first design the counter and then implement your design

**module custom\_counter (**

**input wire clk, // Clock signal**

**input wire reset, // Reset signal**

**input wire [3:0] Cstart, // 4-bit input to specify the starting count**

**output reg [3:0] Count // 4-bit output representing the current count**

**);**

**// Define the states for the sequence**

**localparam S0 = 4'b0000; // 0**

**localparam S1 = 4'b0101; // 5**

**localparam S2 = 4'b1010; // A**

**localparam S3 = 4'b0100; // 4**

**localparam S4 = 4'b0010; // 2**

**localparam S5 = 4'b1101; // D**

**localparam S6 = 4'b0111; // 7**

**localparam S7 = 4'b0011; // 3**

**localparam S8 = 4'b1100; // C**

**reg [3:0] state; // Current state of the counter**

**// Sequential logic to update state on clock edge**

**always @(posedge clk or posedge reset) begin**

**if (reset) begin**

**// Initialize state to the starting count on reset**

**state <= Cstart;**

**end else begin**

**// State transition logic**

**case (state)**

**S0: state <= S1;**

**S1: state <= S2;**

**S2: state <= S3;**

**S3: state <= S4;**

**S4: state <= S5;**

**S5: state <= S6;**

**S6: state <= S7;**

**S7: state <= S8;**

**S8: state <= S0;**

**default: state <= S0; // Safety case for invalid state**

**endcase**

**end**

**end**

**// Output logic**

**always @(\*) begin**

**Count = state; // The output is the current state**

**end**

**endmodule**

1. Then write a test bench to test it and report any errors in the count

**module tb\_custom\_counter();**

**reg clk;**

**reg reset;**

**reg [3:0] start\_count;**

**wire [3:0] current\_count;**

**// Instantiate the custom\_counter module**

**custom\_counter uut (**

**.clk(clk),**

**.reset(reset),**

**.Cstart(start\_count),**

**.Count(current\_count)**

**);**

**// Clock generation**

**always #10 clk = ~clk;**

**reg [3:0] expected\_sequence [8:0];**

**integer i;**

**initial begin**

**expected\_sequence[0] = 4'b0000; // 0**

**expected\_sequence[1] = 4'b0101; // 5**

**expected\_sequence[2] = 4'b1010; // A**

**expected\_sequence[3] = 4'b0100; // 4**

**expected\_sequence[4] = 4'b0010; // 2**

**expected\_sequence[5] = 4'b1101; // D**

**expected\_sequence[6] = 4'b0111; // 7**

**expected\_sequence[7] = 4'b0011; // 3**

**expected\_sequence[8] = 4'b1100; // C**

**// Initialize signals**

**clk = 0;**

**reset = 1; // Assert reset**

**start\_count = 4'b0000;**

**#20 reset = 0; // De-assert reset**

**// Test for two full cycles of the sequence**

**for (i = 0; i < 18; i = i + 1) begin**

**@(posedge clk); // Wait for positive clock edge**

**#1; // Small delay to check output after clock edge**

**if (current\_count !== expected\_sequence[i % 9]) begin**

**$display("Error at step %d: Expected %h, but got %h", i, expected\_sequence[i % 9], current\_count);**

**end else begin**

**$display("Step %d: Count = %h (Correct)", i, current\_count);**

**end**

**end**

**$stop; // End simulation**

**end**

**endmodule**