Name _________Andrew Quick_____________ abc123________qxt050__________

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
```

Name _________Andrew Quick_____________ abc123________qxt050__________

```
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
```

Name _________Andrew Quick_____________   abc123________qxt050__________

```
        endmodule
```

2. 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;
```

Name _________Andrew Quick_____________ abc123________qxt050__________

```
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
```

Name __________Andrew Quick______________   abc123_________qxt050___________

```
    end


    $stop;  // End simulation

  end


endmodule
```