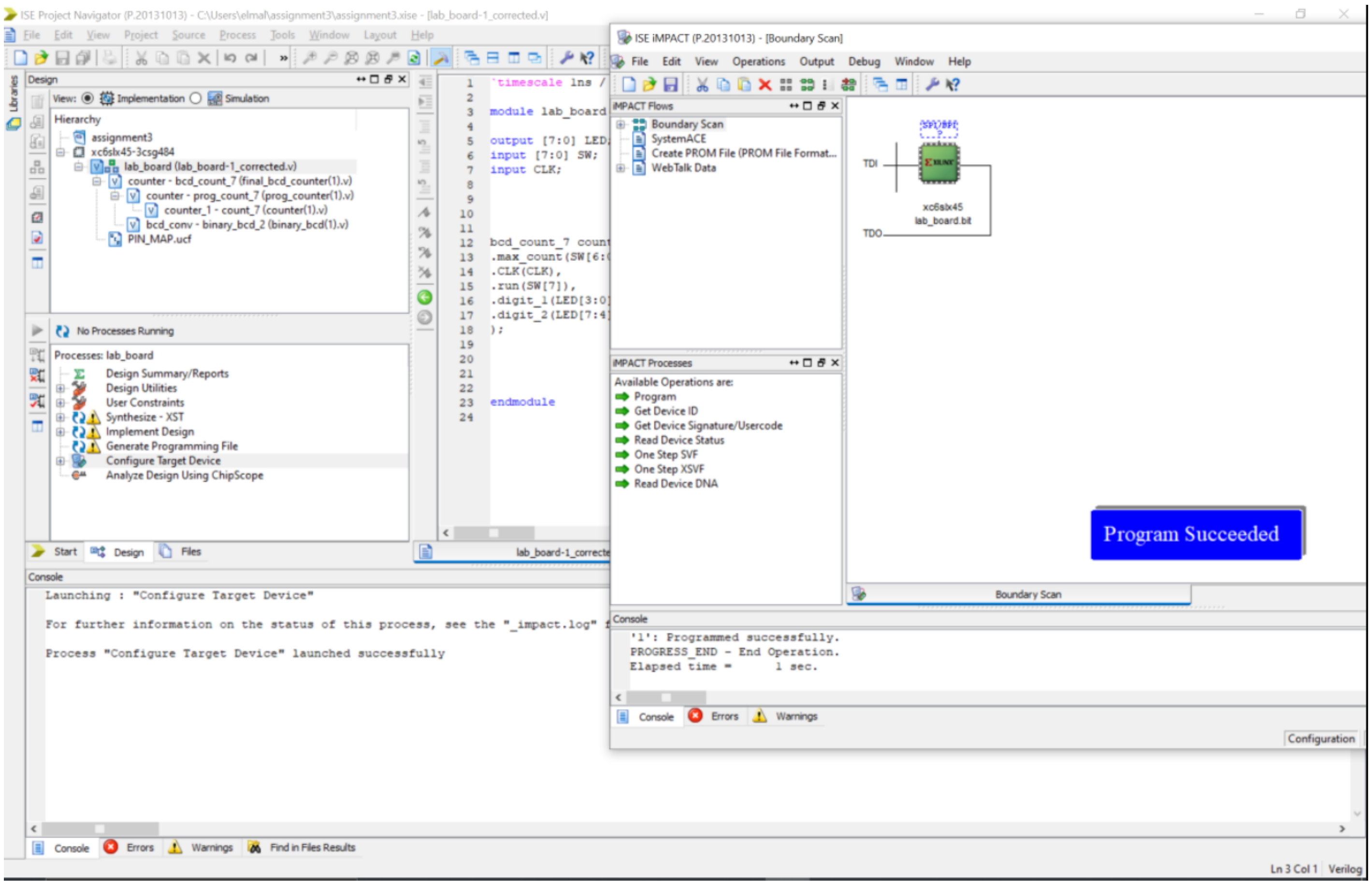
# CDA 4203L
# Computer System Design Lab

## Lab 3 Report
## Programmable BCD Counter

| | |
|---|---|
| Today's Date: | 02/13/2022 |
| Team Members: | Ahmed El Maliki |
| | Stella Kariuki |
| | Ivan Gonzalez |
| Work Distribution: | Ahmed: worked on the code provided and made sure it was working properly.<br>Stella: contribute with the code and to test its functionality<br>Ivan: tested the functionality of the codes and recorded the outputs as well as the waveform |
| No. of Hours Spent: | 8 hours |
| Exercise Difficulty:<br>(Easy, Average, Hard) | Hard |
| Any Other Feedback: | We needed more explanation of how the board works to get familiar with it. We successfully completed the tutorial for this lab and our board program compiles but the board does not do anything. We used the UCF file provided for the tutorial |

**Board compiles and gets programmed with our computer but when the code gets run there is no output from the board.**


ISE Project Navigator (P.20131013) - C:\Users\elmal\assignment3\assignment3.xise - [lab_board-1_corrected.v]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
assignment3
xc6slx45-3csg484
lab_board (lab_board-1_corrected.v)
counter - bcd_count_7 (final_bcd_counter(1).v)
counter - prog_count_7 (prog_counter(1).v)
counter_1 - count_7 (counter(1).v)
bcd_conv - binary_bcd_2 (binary_bcd(1).v)
PIN_MAP.ucf
No Processes Running
Processes: lab_board
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
Start Design Files
lab_board-1_correcte
ISE iMPACT (P.20131013) - [Boundary Scan]
File Edit View Operations Output Debug Window Help
iMPACT Flows
Boundary Scan
SystemACE
Create PROM File (PROM File Format...
WebTalk Data
iMPACT Processes
Available Operations are:
Program
Get Device ID
Get Device Signature/Usercode
Read Device Status
One Step SVF
One Step XSVF
Read Device DNA
TDI
TDO
xc6slx45
lab_board.bit
Program Succeeded
Boundary Scan
Console Errors Warnings
Configuration
Console Errors Warnings Find in Files Results
Ln 3 Col 1 Verilog

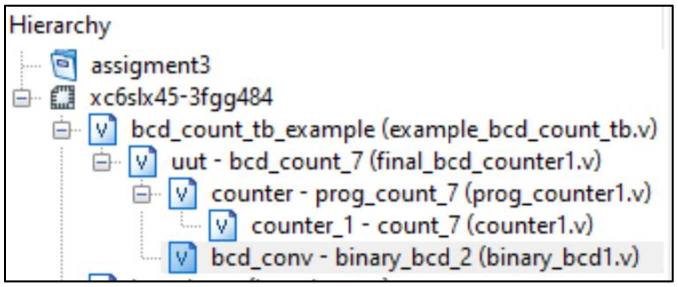
```
`timescale 1ns /

module lab_board

output [7:0] LED,
input [7:0] SW;
input CLK;




bcd_count_7 count
.max_count(SW[6:0
.CLK(CLK),
.run(SW[7]),
.digit_1(LED[3:0]
.digit_2(LED[7:4]
);



endmodule
```

```
Launching : "Configure Target Device"

For further information on the status of this process, see the "_impact.log"

Process "Configure Target Device" launched successfully
```

```
'1': Programmed successfully.
PROGRESS_END - End Operation.
Elapsed time =      1 sec.
```

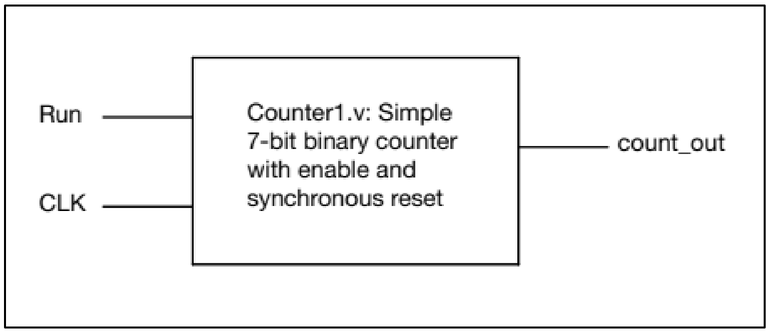
**Problem 1:** Draw overall block diagram of the programmable counter. Briefly explain how your design works. For each block, include the Verilog code. *Use as many pages as needed.*

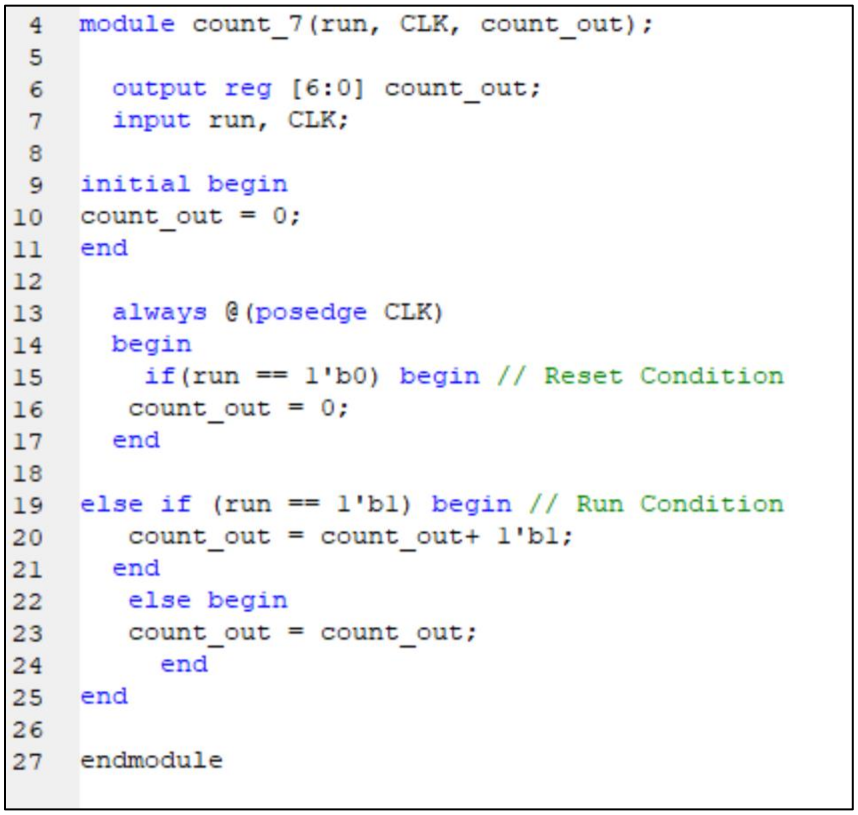
Based on the Hierarchy of our project, we will discuss our block diagrams from bottom to top.



```
Hierarchy
  assigment3
  xc6slx45-3fgg484
    bcd_count_tb_example (example_bcd_count_tb.v)
      uut - bcd_count_7 (final_bcd_counter1.v)
        counter - prog_count_7 (prog_counter1.v)
          counter_1 - count_7 (counter1.v)
        bcd_conv - binary_bcd_2 (binary_bcd1.v)
```

## Counter1.v

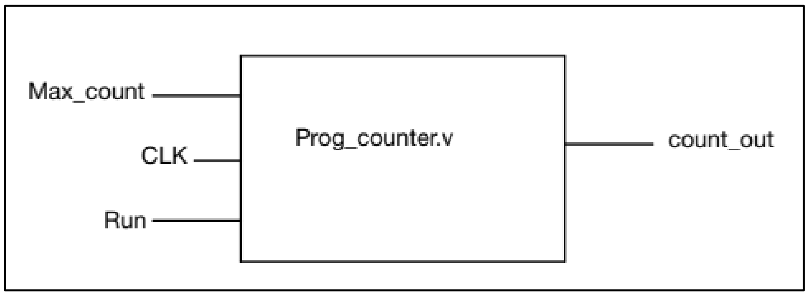

Run
CLK
Counter1.v: Simple 7-bit binary counter with enable and synchronous reset
count_out

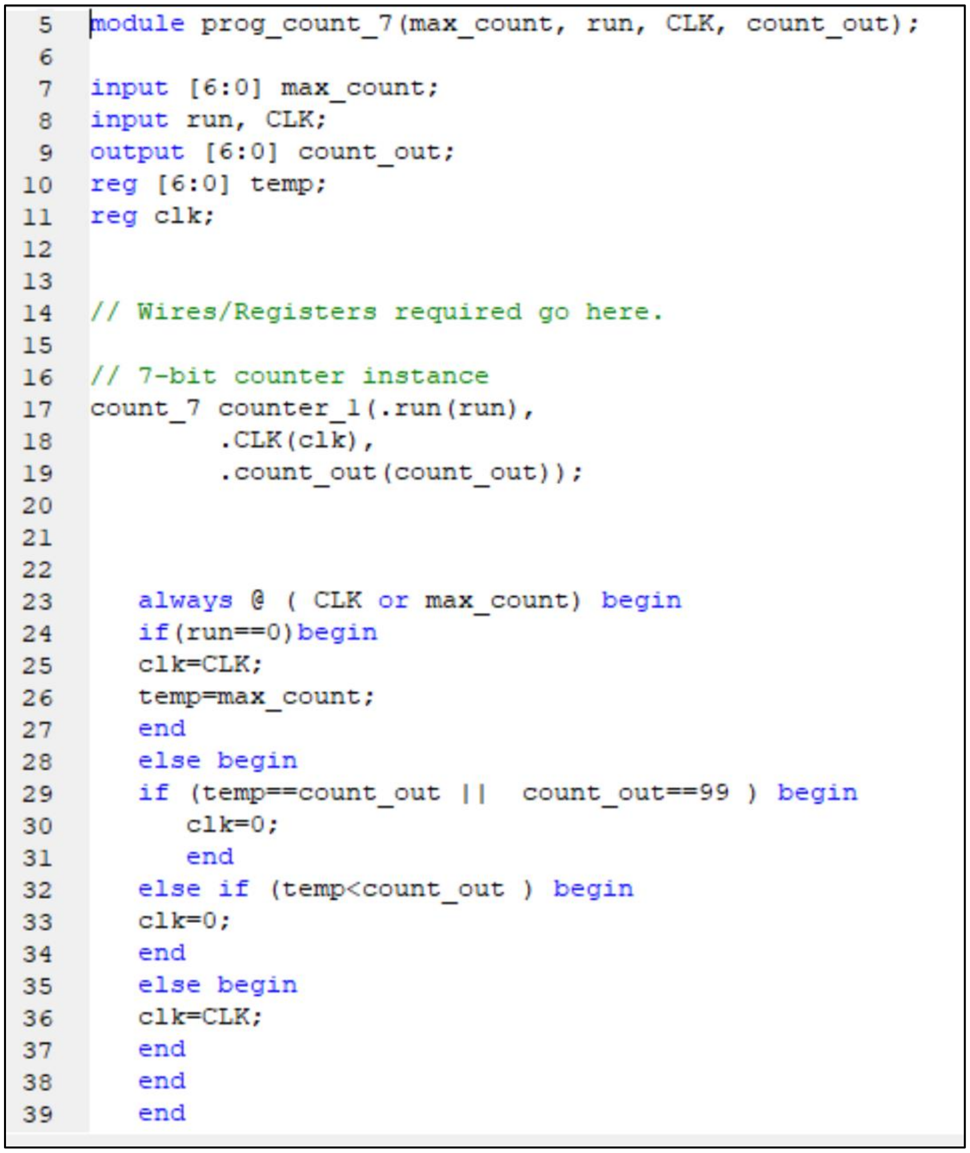


```
module count_7(run, CLK, count_out);

  output reg [6:0] count_out;
  input run, CLK;

initial begin
count_out = 0;
end

  always @(posedge CLK)
  begin
    if(run == 1'b0) begin // Reset Condition
   count_out = 0;
  end

else if (run == 1'b1) begin // Run Condition
   count_out = count_out+ 1'b1;
  end
   else begin
   count_out = count_out;
     end
end

endmodule
```

What this file does is that takes a input from the run and the clock and if the input is 1’b0 it will reset the clock, otherwise if the run is 1’b1 it will initiate the 7-bit binary counter.

## Program_counter.v

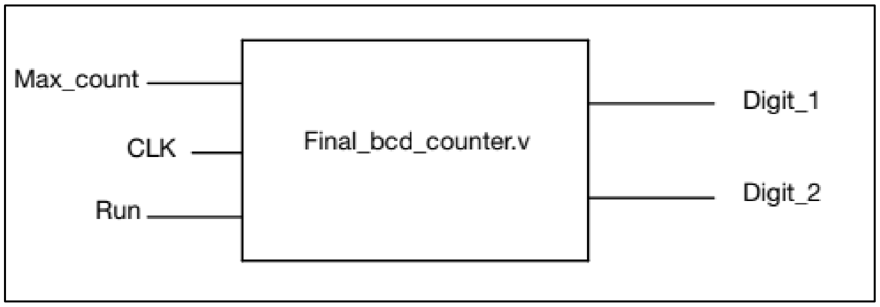


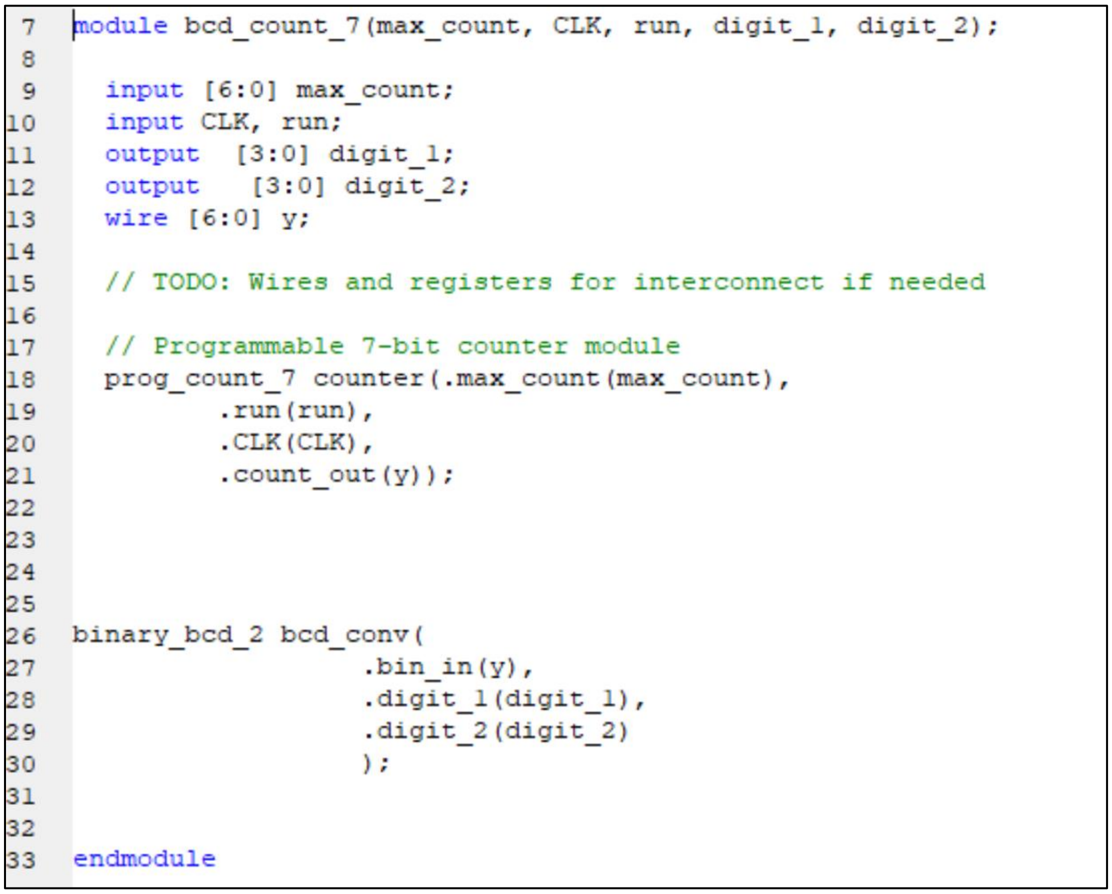


```
module prog_count_7(max_count, run, CLK, count_out);

input [6:0] max_count;
input run, CLK;
output [6:0] count_out;
reg [6:0] temp;
reg clk;


// Wires/Registers required go here.

// 7-bit counter instance
count_7 counter_1(.run(run),
        .CLK(clk),
        .count_out(count_out));



   always @ ( CLK or max_count) begin
   if(run==0)begin
   clk=CLK;
   temp=max_count;
   end
   else begin
   if (temp==count_out ||  count_out==99 ) begin
      clk=0;
      end
   else if (temp<count_out ) begin
   clk=0;
   end
   else begin
   clk=CLK;
   end
   end
   end
```

This program will use the 7-bit binary counter and add programmability to it. The file provide the logic to stop a counter when it reaches a designed value. The maximum value is stated to be 99. In this program we have included registers to keep track of the values.

## Final_bcd_counter1.v

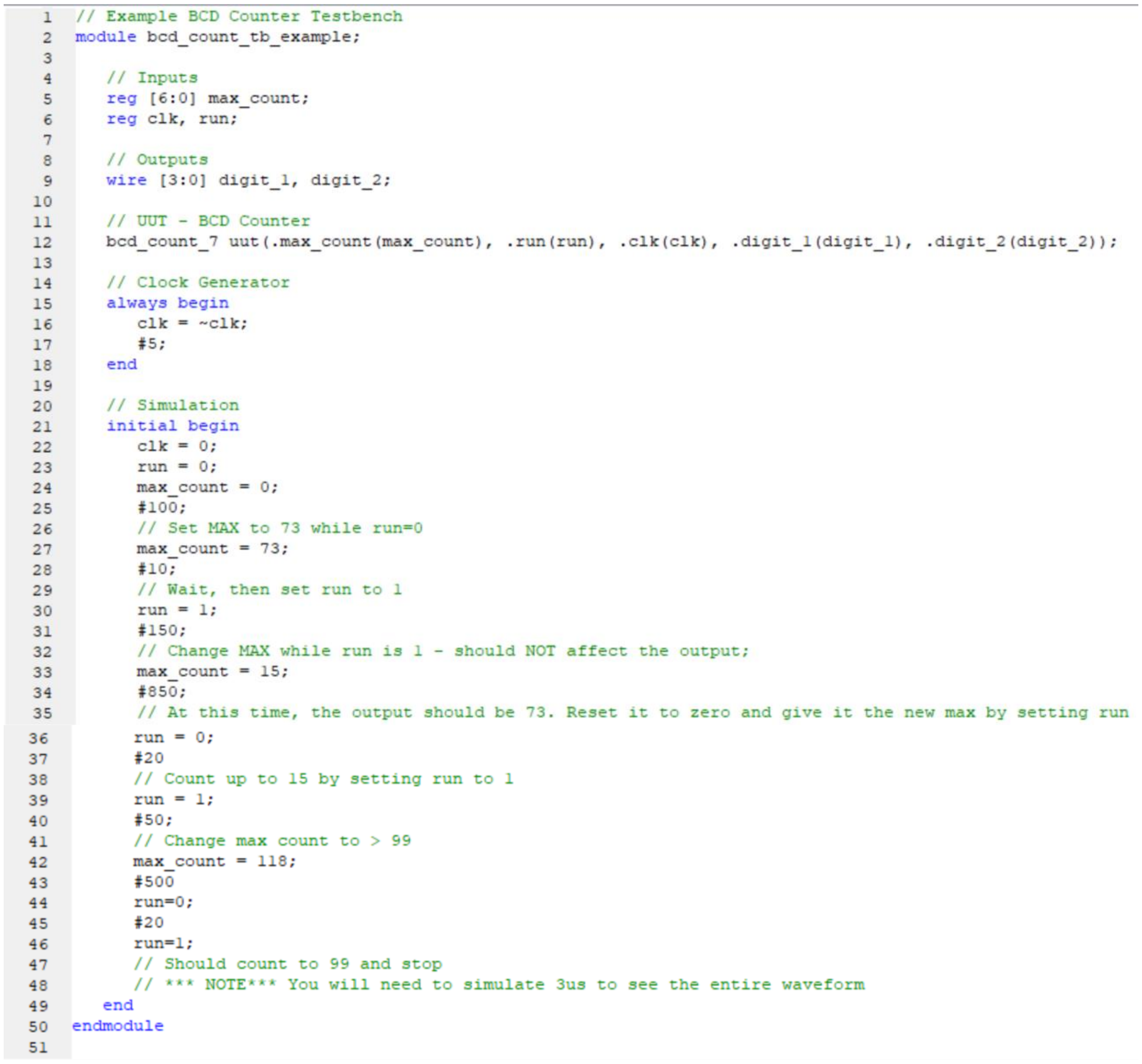




```
module bcd_count_7(max_count, CLK, run, digit_1, digit_2);

  input [6:0] max_count;
  input CLK, run;
  output  [3:0] digit_1;
  output   [3:0] digit_2;
  wire [6:0] y;

  // TODO: Wires and registers for interconnect if needed

  // Programmable 7-bit counter module
  prog_count_7 counter(.max_count(max_count),
         .run(run),
         .CLK(CLK),
         .count_out(y));




binary_bcd_2 bcd_conv(
                  .bin_in(y),
                  .digit_1(digit_1),
                  .digit_2(digit_2)
                  );


endmodule
```

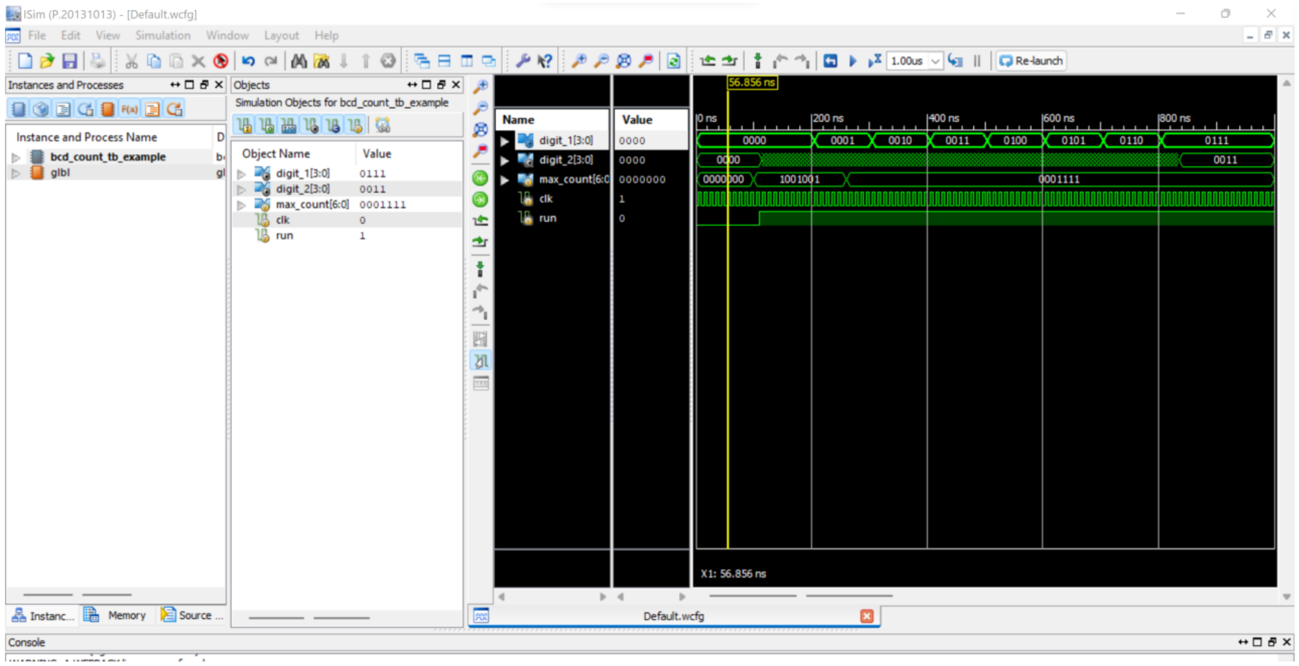
// This is the top module for the programmable BCD counter. It implements a programmable 7-bit counter and a binary-to-bcd converter that can output two digits.

**Problem 2:** Testbench code: Include your Verilog testbench. *Use as many pages as needed.*



```
// Example BCD Counter Testbench
module bcd_count_tb_example;

   // Inputs
   reg [6:0] max_count;
   reg clk, run;

   // Outputs
   wire [3:0] digit_1, digit_2;

   // UUT - BCD Counter
   bcd_count_7 uut(.max_count(max_count), .run(run), .clk(clk), .digit_1(digit_1), .digit_2(digit_2));

   // Clock Generator
   always begin
      clk = ~clk;
      #5;
   end

   // Simulation
   initial begin
      clk = 0;
      run = 0;
      max_count = 0;
      #100;
      // Set MAX to 73 while run=0
      max_count = 73;
      #10;
      // Wait, then set run to 1
      run = 1;
      #150;
      // Change MAX while run is 1 - should NOT affect the output;
      max_count = 15;
      #850;
      // At this time, the output should be 73. Reset it to zero and give it the new max by setting run
      run = 0;
      #20
      // Count up to 15 by setting run to 1
      run = 1;
      #50;
      // Change max count to > 99
      max_count = 118;
      #500
      run=0;
      #20
      run=1;
      // Should count to 99 and stop
      // *** NOTE*** You will need to simulate 3us to see the entire waveform
   end
endmodule
```

**Problem 3:** Simulation Waveforms. Include waveforms that demonstrate the functionality. *Use as many pages as needed.*



| Object Name | Value |
|---|---|
| digit_1[3:0] | 0111 |
| digit_2[3:0] | 0011 |
| max_count[6:0] | 0001111 |
| clk | 0 |
| run | 1 |

| Name | Value |
|---|---|
| digit_1[3:0] | 0000 |
| digit_2[3:0] | 0000 |
| max_count[6:0] | 0000000 |
| clk | 1 |
| run | 0 |