Lab3: Seven Segment Hierarchy Design

Leonardo Fusser (1946695)

Objectives:

a) Understand hierarchical design process using top module.
b) Implement and test various combinational logic design into hardware using Verilog.
c) Understand the operation of 7-segment display (7sd) of digital system.
d) Code combinational logic behaviorally.

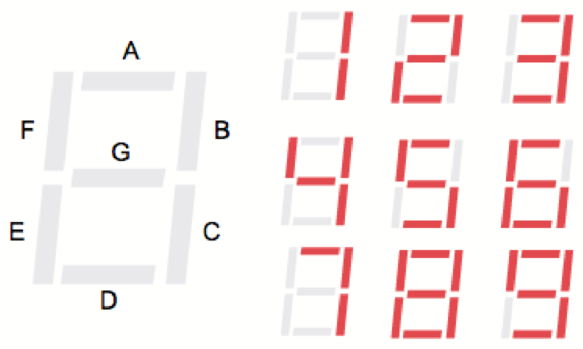
NO full report to submit.

To hand in:

- Listing of the following for each of your designs on LEA:
  1. This sheet with answers and screenshots.
  2. Commented Verilog source files - module and top module for all parts.
  3. UCF file.
  4. Commented resource summary report copied in a word document (must be commented.
- The main file must include a prolog header: Lab number and name, author, description, version number.
- Each and every module should include a prolog header.

Theory:

Seven-segment displays (7sd) are some of the most common electronic display devices in use. They can be used to display any decimal digit by illuminating particular segments and leaving other segments dark. 7sd devices are constructed from seven LEDs that have been arranged in a figure "8" pattern as shown in the figure below.



Seven-Segment Display with 9 illumination patterns

These LEDs function identically to the individual LEDs – they emit light when a small current passes through them. The 7sd device can display a particular digit if certain LED segments are illuminated while others remain dark. As examples, if only segments b and c are illuminated, then the display will show a "1". To cause an illuminating current to flow through any given LED segment, a logic signal must be impressed across the segment LED.

In a typical 7sd circuit, a current-limiting resistor is placed on the cathode lead, and a transistor is used on the anode lead to provide additional current (most signal pins on digital ICs – like the FPGA on the Digilent board – cannot provide enough current to light all the display segments, so a transistor is used to provide more current).

In order to display a pattern, a 7sd device requires seven logic signals, one for each segment.
The Digilent board uses a common anode display, which means that all the anode connections for a given digit are tied together into a common circuit node as shown below. To illuminate a given segment in a given digit, a "1" must be applied to the digit's anode, and "0" must be applied to the segment's cathodes (NOTE: With Digilent Basys2 board, a "1" is applied to a digit's anode by applying a "0" to the circuit node that drives the transistor; thus, the anode signals AN3 - AN0 are "active low").

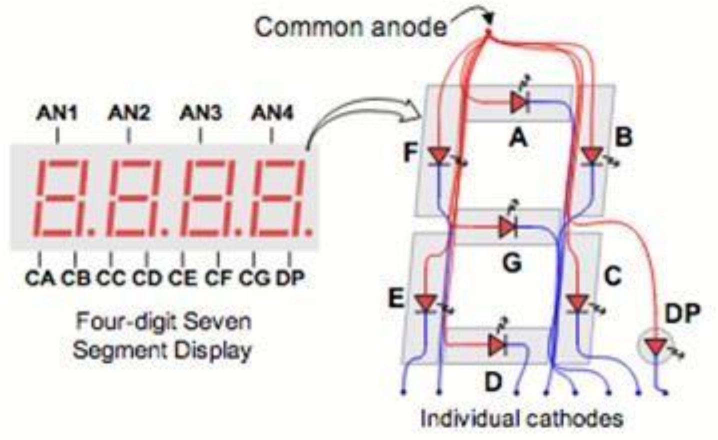


Figure 1: Common anode 7 segment display

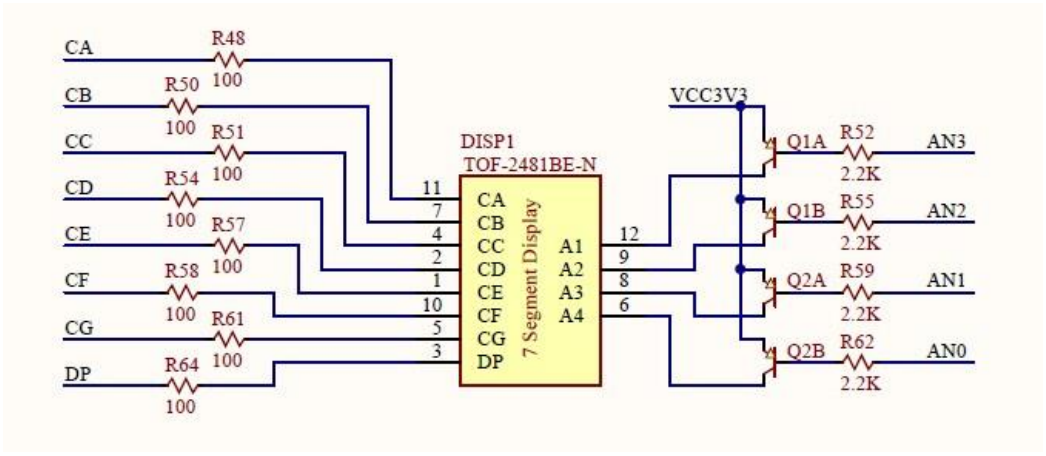


Figure 2: 7 segment display on Basys 2 board

A ***seven-segment decoder (SSD)*** receives four signals that represent the four bits of a binary number and produces seven output signals that can drive the seven segments in the seven-segment display. Typically, the input signals are named B3-B0, and the output signals are given a letter to indicate which segment they must drive (A-F). As discussed above, each of the seven outputs could be thought of as a separate 4-input logic design problem.

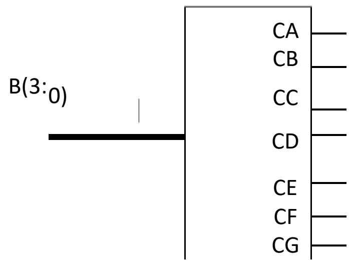


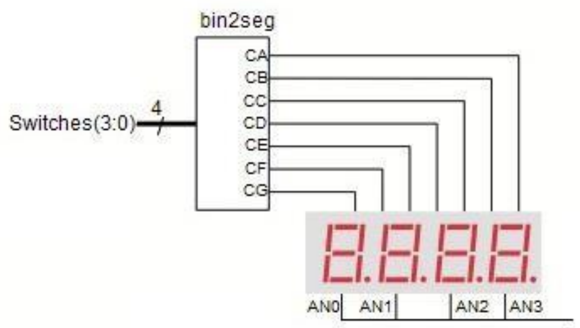
Figure 3: *seven-segment decoder*

Refer to Basys2 Reference manual for further details.

Lab Work:

Part A: 7-segment decoder

In part A, you are required to design a 7-segment decoder that can drive all the 4 hexadecimal digits (with same pattern, duplicate across the 4 digits display) on the 7sd device of a Basys board, based on the binary code entered via 4 switches.

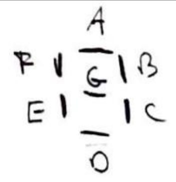


1. Determine truth table of binary to 7-segment decoder circuit for Basys2 board. Your decoder should display the decimal digits 0-9 for bit patterns 0000-1001, and A-F for bit patterns 1010-1111 (you will need to get a little bit creative to show all the hex digits - think about lower-case letters).

   Example of the truth table is as shown below. See next page for complete truth table.

B[3]	B[2]	B[1]	B[0]	CA	CB	CC	CD	CE	CF	CG
0	0	0	0	0	0	0	0	0	0	1
	.									
	.									
	.									
1	1	1	1	0	1	1	1	0	0	0

Hï±Φ˜ï‡δ¬÷G



Part A

1.

			[∅] B_ϕ	[1] B_1	[2] B_2	[3] B_3	[∅] CA	[1] CB	[2] CC	[3] CD	[4] CE	[5] CF	[6] CG	
0		→	0	0	0	0	0	0	0	0	0	0	1	✓
1		→	0	0	0	1	1	0	0	1	1	1	1	✓
2		→	0	0	1	0	0	0	1	0	0	1	0	✓
3		→	0	0	1	1	0	0	0	0	1	1	0	✓
4		→	0	1	0	0	1	0	0	1	1	0	0	✓
5		→	0	1	0	1	0	1	0	0	1	0	0	✓
6		→	0	1	1	0	0	1	0	0	0	0	0	✓
7		→	0	1	1	1	0	0	0	1	1	1	1	✓
8		→	1	0	0	0	0	0	0	0	0	0	0	/
9		→	1	0	0	1	0	0	0	0	1	0	0	/
A		→	1	0	1	0	0	0	0	1	0	0	0	✓
B		→	1	0	1	1	1	1	0	0	0	0	0	✓
C		→	1	1	0	0	0	1	1	0	0	0	1	✓
D		→	1	1	0	1	1	0	0	0	0	1	0	✓
E		→	1	1	1	0	0	1	1	0	0	0	0	
F		→	1	1	1	1	0	1	1	1	0	0	0	

2. Create a new ISE project named lab3.
3. Create a new Verilog design source file for this project, and name it "bin2seg" with the following input and output parameters:
    - Input "B", which is a 4-bit bus that contains the binary form of the number to be displayed on the 7-segment display. B[3] is the MSB, and B[0] is the LSB.
    - Output "a_to_g", which is a 7-bit bus that controls the 7 segments of digit display on 7sd. a_to_g[0] controls segment *a*, while a_to_g[6] controls segment *g*.
    - Output "dp" that controls the decimal point display on the 7sd.
4. Write the Verilog code for a seven-segment decoder circuit that can drive segments of 7sd device on Basys board, based on the 4-bit binary input.
5. In a new file (e.g. lab3a.v), now create the top-level module for this project, named top3a. This module should read the 4-bit binary code from switches and display it as a hexadecimal digit (and duplicate the display by 4 times) on all the 4 7-segment display of Basys2 board.
    a) This top module should have the following input and output parameters:
        - Input "sw": a 4-bit bus with sw[3] is the MSB, and sw[0] is the LSB. These switches define the binary number that will be display in hexadecimal digits, on the 7-segment display. This matches sw3-0 of Basys board.
        - Output "seg": a 7-bit bus that controls each segment of the SSD.
        - Output "dp": controls decimal point display on the SSD.
        - Output "an": a 4-bit bus that controls the common anodes of each SSD digit. an[3] controls the MSB of the SSD display, while an[0] control the LSB (right most digit of the display).

    b) Instantiate bin2seg module and complete the Verilog code based on the design requirement.

```
module top3a(
     input [3:0] sw,
     output [6:0] seg,
     output [3:0] an,
     output dp
     );
```

6. Generate an UCF file for this top module.
7. Compile and generate programming bit file. Download to your Basys board for testing and verification. Demonstrate your working code to your instructor.

Part B: Seven Segment Display

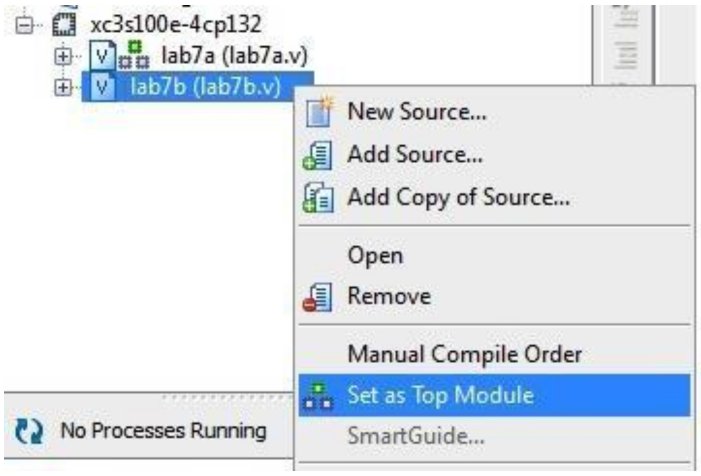
In part B, instead of displaying the hexadecimal digit on all the 7sd, you are required to use a push button(s) to decide the position of the display.

Example, when push button 0 is pressed, the right most 7sd will be displayed. When multiple buttons are pressed, the 7sd will light up more than 1 digit.

8. In a new file (e.g. lab3b.v) create a new Verilog source for this project, named top3b.

```
module top3b(
        input       [3:0] sw,
        input       [3:0] btn,
        output      [6:0] seg,
        output      [3:0] an,
        output            dp
);
```

9. Set this file as the top module. Note that in ISE, you can only have ONE top module at any one time. However, all these modules can share some common lower hierarchy modules.



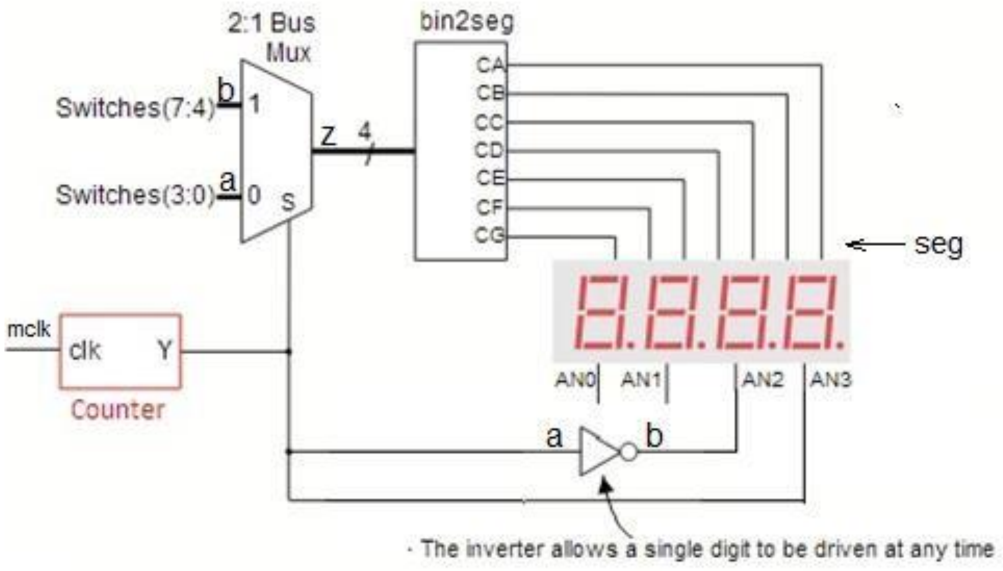
10. Instantiate your bin2seg module created from part A in your part B top module – top3b. Add any necessary code to meet the requirements of the design as stated above.
11. Again, create the necessary UCF file for this module. Compile and generate bit programming file.
12. Download to your Basys board for testing and verification. Demonstrate your working code to your instructor.

Part C: A Real 2 Digits Seven Segment Display

In a fully functional seven segment display, different data should be displayed on all digits at the same time. To do that, instead of using push button, we use a counter to cycle through selection patterns at a reasonable speed. Use the `counter.v` file as part of the design. The code is written here, and it will be explained later.

```
module counter(

    input clk,
    output div_out
    );
reg [18:0]CNT;

always @(posedge clk)
      begin

      CNT <= CNT +1;
            end

            assign div_out=CNT[18]; // tapped for 96 Hz

endmodule
```

The diagram below illustrates the block diagram of top-level module. The right most 7sd shall display the hexadecimal code entered via switch3 – switch 0, while the 7sd next to it display hexadecimal code represented by switch7 – switch 4.



13. In a new file (e.g. lab3c.v) create a new Verilog source for this project, named top3c.
14. Set this file as the top module. Implement the design based on the diagram above. You are also required to implement a 2:1 4-bit bus mux (see previous lab). Your top module should have four chips: U1, U2, U3, and U4 (bin2seg, invert, counter, mux21). Your top module declaration should look something like this:

```
module top3c(
      input [7:0] sw,
      input mclk,
      output [6:0] seg,
      output [3:0] an,
      output dp
);
```

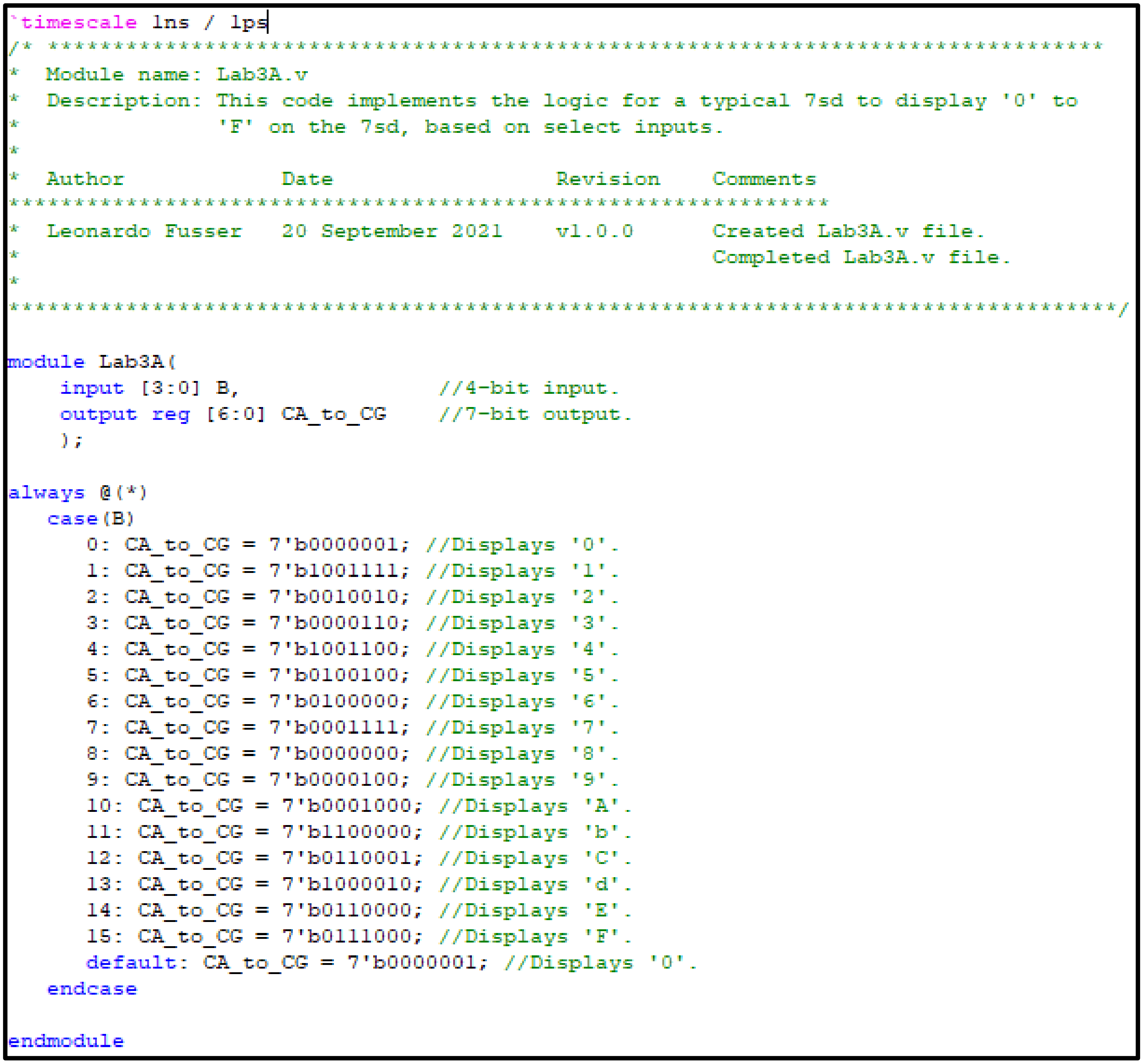
15. Demonstrate your working design to your instructor.

16. Also, modify the counter's frequency to 2.98Hz by modifying the `counter.v` file:

```
module counter(

    input clk,
    output div_out
    );
reg [18:0]CNT;

always @(posedge clk)
      begin

      CNT <= CNT +1;
            end

            //assign div_out=CNT[18]; // tapped for 96 Hz
            assign div_out=cnt[23];   // tapped for 2.98Hz

endmodule
```

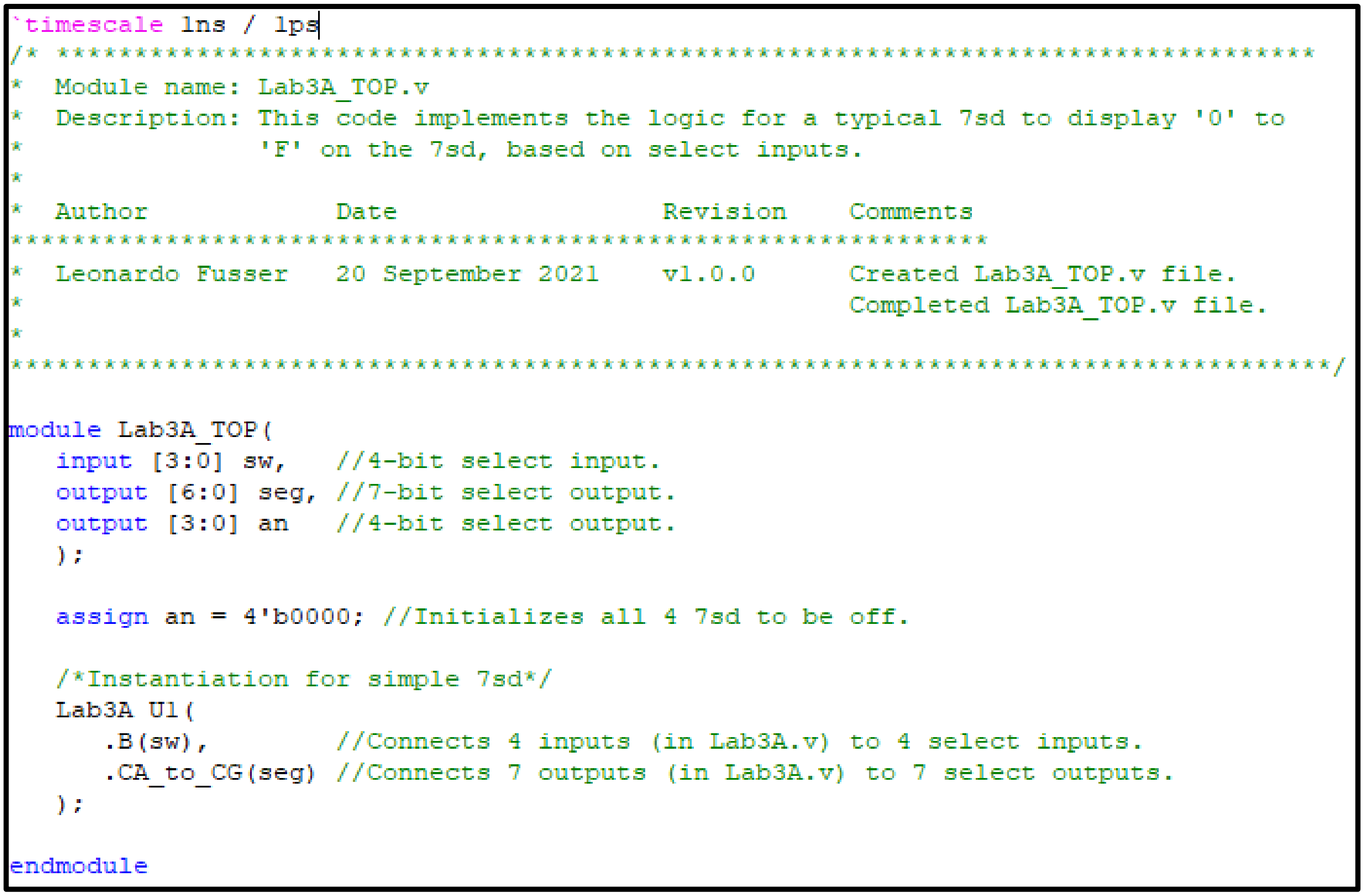
Code:

[Lab3A.v -> Part A]



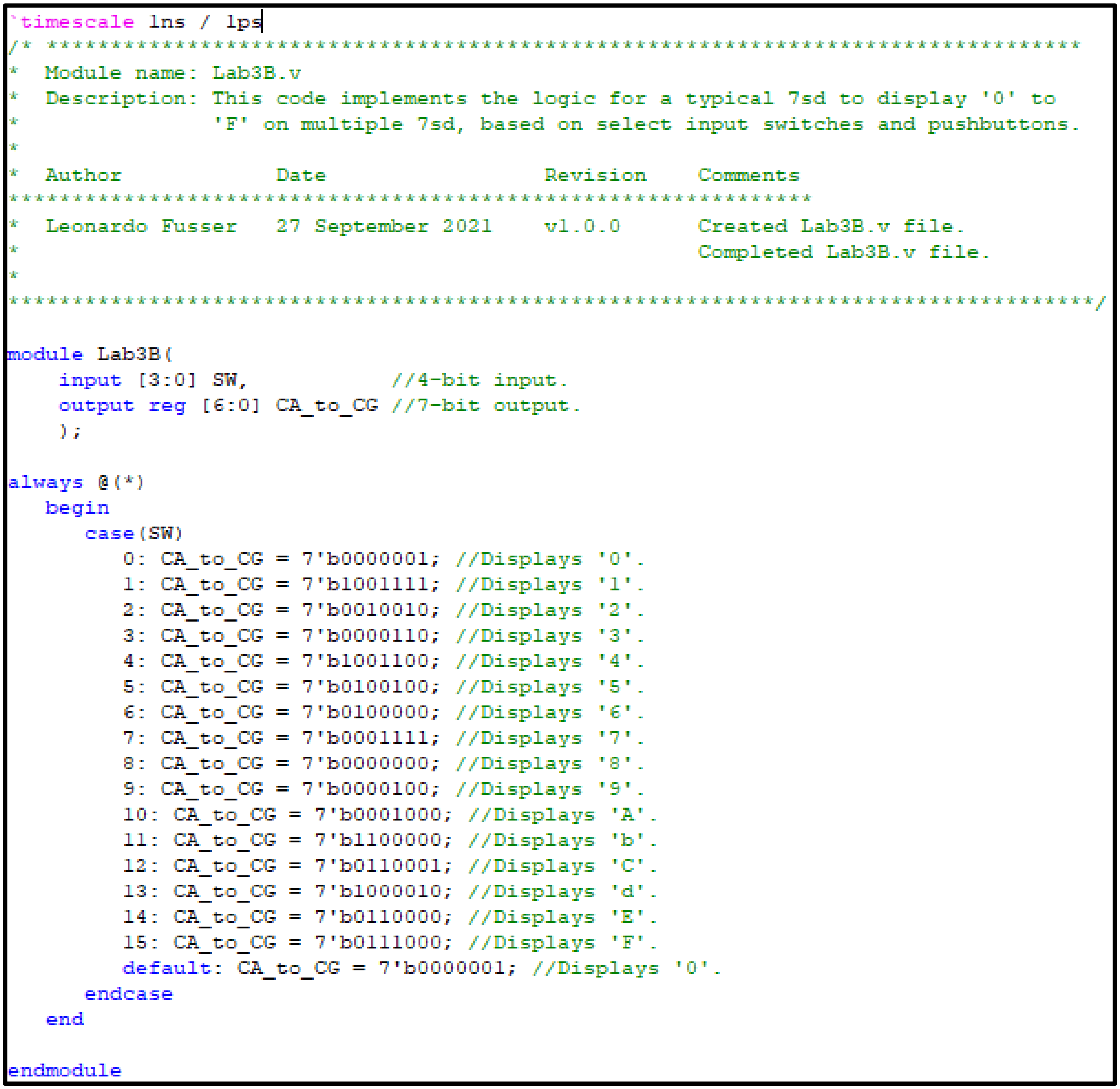
```
`timescale 1ns / 1ps
/* ***************************************************************************
*  Module name: Lab3A.v
*  Description: This code implements the logic for a typical 7sd to display '0' to
*               'F' on the 7sd, based on select inputs.
*
*  Author            Date                  Revision     Comments
*************************************************************
*  Leonardo Fusser   20 September 2021     v1.0.0       Created Lab3A.v file.
*                                                       Completed Lab3A.v file.
*
*****************************************************************************/

module Lab3A(
    input [3:0] B,              //4-bit input.
    output reg [6:0] CA_to_CG   //7-bit output.
    );

always @(*)
   case(B)
      0: CA_to_CG = 7'b0000001; //Displays '0'.
      1: CA_to_CG = 7'b1001111; //Displays '1'.
      2: CA_to_CG = 7'b0010010; //Displays '2'.
      3: CA_to_CG = 7'b0000110; //Displays '3'.
      4: CA_to_CG = 7'b1001100; //Displays '4'.
      5: CA_to_CG = 7'b0100100; //Displays '5'.
      6: CA_to_CG = 7'b0100000; //Displays '6'.
      7: CA_to_CG = 7'b0001111; //Displays '7'.
      8: CA_to_CG = 7'b0000000; //Displays '8'.
      9: CA_to_CG = 7'b0000100; //Displays '9'.
      10: CA_to_CG = 7'b0001000; //Displays 'A'.
      11: CA_to_CG = 7'b1100000; //Displays 'b'.
      12: CA_to_CG = 7'b0110001; //Displays 'C'.
      13: CA_to_CG = 7'b1000010; //Displays 'd'.
      14: CA_to_CG = 7'b0110000; //Displays 'E'.
      15: CA_to_CG = 7'b0111000; //Displays 'F'.
      default: CA_to_CG = 7'b0000001; //Displays '0'.
   endcase

endmodule
```

[Lab3A_TOP.v -> Part A]



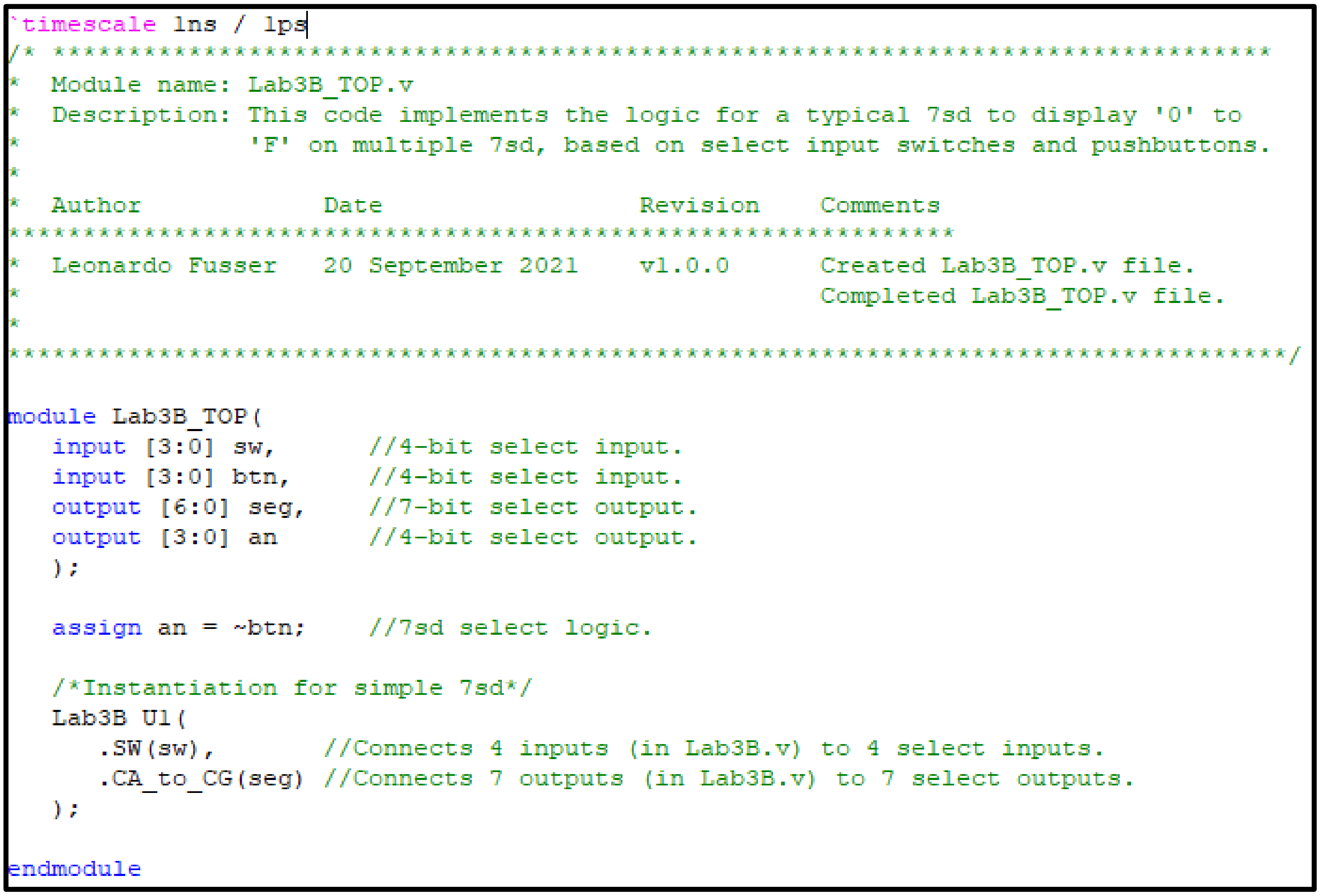
```
`timescale 1ns / 1ps
/* ***************************************************************************************
*   Module name: Lab3A_TOP.v
*   Description: This code implements the logic for a typical 7sd to display '0' to
*                'F' on the 7sd, based on select inputs.
*
*   Author              Date                    Revision     Comments
***************************************************************
*   Leonardo Fusser     20 September 2021       v1.0.0       Created Lab3A_TOP.v file.
*                                                            Completed Lab3A_TOP.v file.
*
*************************************************************************************/

module Lab3A_TOP(
   input [3:0] sw,    //4-bit select input.
   output [6:0] seg, //7-bit select output.
   output [3:0] an   //4-bit select output.
   );

   assign an = 4'b0000; //Initializes all 4 7sd to be off.

   /*Instantiation for simple 7sd*/
   Lab3A U1(
      .B(sw),        //Connects 4 inputs (in Lab3A.v) to 4 select inputs.
      .CA_to_CG(seg) //Connects 7 outputs (in Lab3A.v) to 7 select outputs.
   );

endmodule
```

[Lab3B.v -> Part B]



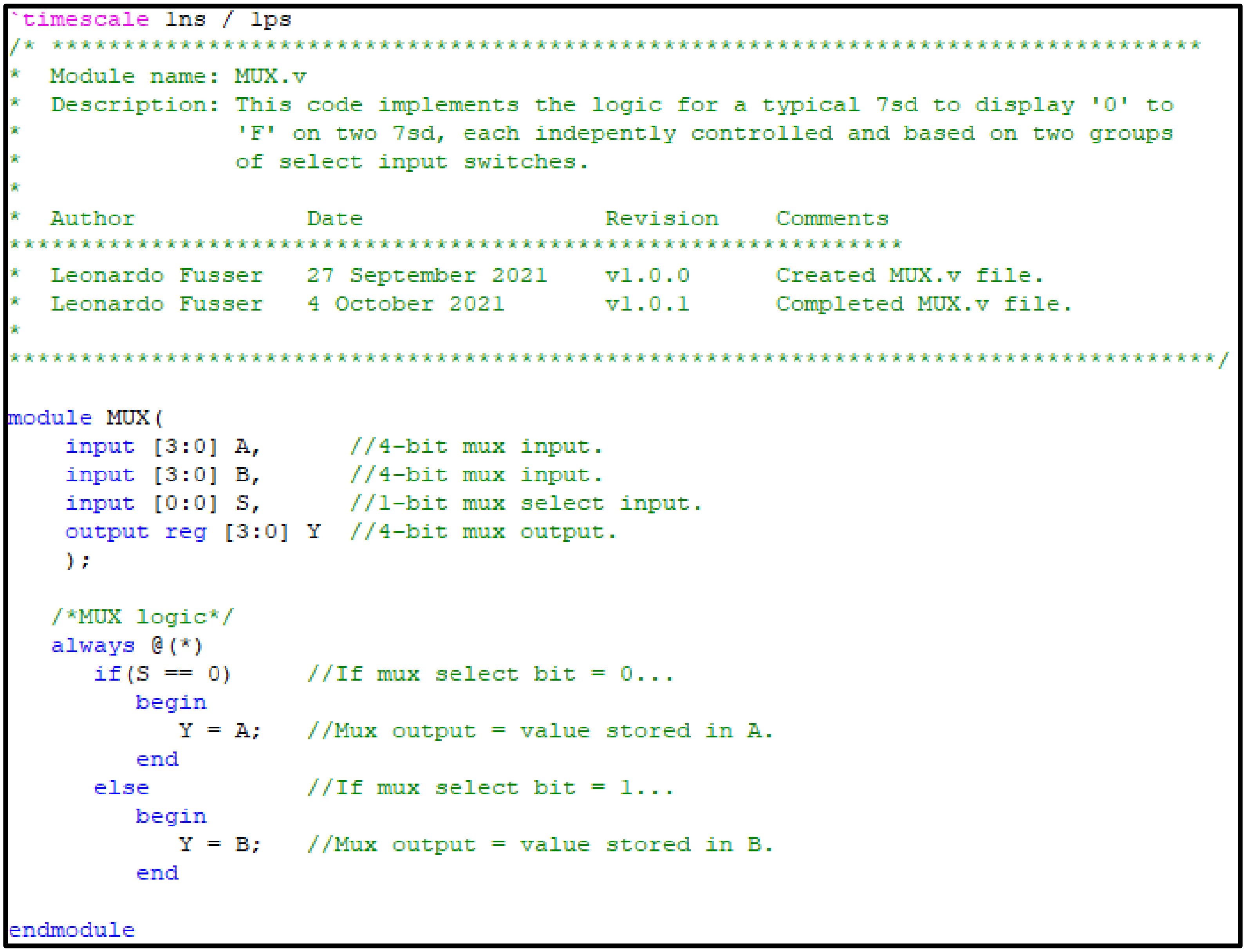
```verilog
`timescale 1ns / 1ps
/* ****************************************************************************
*  Module name: Lab3B.v
*  Description: This code implements the logic for a typical 7sd to display '0' to
*               'F' on multiple 7sd, based on select input switches and pushbuttons.
*
*  Author            Date                  Revision     Comments
*****************************************************************
*  Leonardo Fusser   27 September 2021     v1.0.0       Created Lab3B.v file.
*                                                       Completed Lab3B.v file.
*
*****************************************************************************/

module Lab3B(
    input [3:0] SW,           //4-bit input.
    output reg [6:0] CA_to_CG //7-bit output.
    );

always @(*)
   begin
      case(SW)
         0: CA_to_CG = 7'b0000001; //Displays '0'.
         1: CA_to_CG = 7'b1001111; //Displays '1'.
         2: CA_to_CG = 7'b0010010; //Displays '2'.
         3: CA_to_CG = 7'b0000110; //Displays '3'.
         4: CA_to_CG = 7'b1001100; //Displays '4'.
         5: CA_to_CG = 7'b0100100; //Displays '5'.
         6: CA_to_CG = 7'b0100000; //Displays '6'.
         7: CA_to_CG = 7'b0001111; //Displays '7'.
         8: CA_to_CG = 7'b0000000; //Displays '8'.
         9: CA_to_CG = 7'b0000100; //Displays '9'.
         10: CA_to_CG = 7'b0001000; //Displays 'A'.
         11: CA_to_CG = 7'b1100000; //Displays 'b'.
         12: CA_to_CG = 7'b0110001; //Displays 'C'.
         13: CA_to_CG = 7'b1000010; //Displays 'd'.
         14: CA_to_CG = 7'b0110000; //Displays 'E'.
         15: CA_to_CG = 7'b0111000; //Displays 'F'.
         default: CA_to_CG = 7'b0000001; //Displays '0'.
      endcase
   end

endmodule
```

[Lab3B_TOP.v -> Part B]



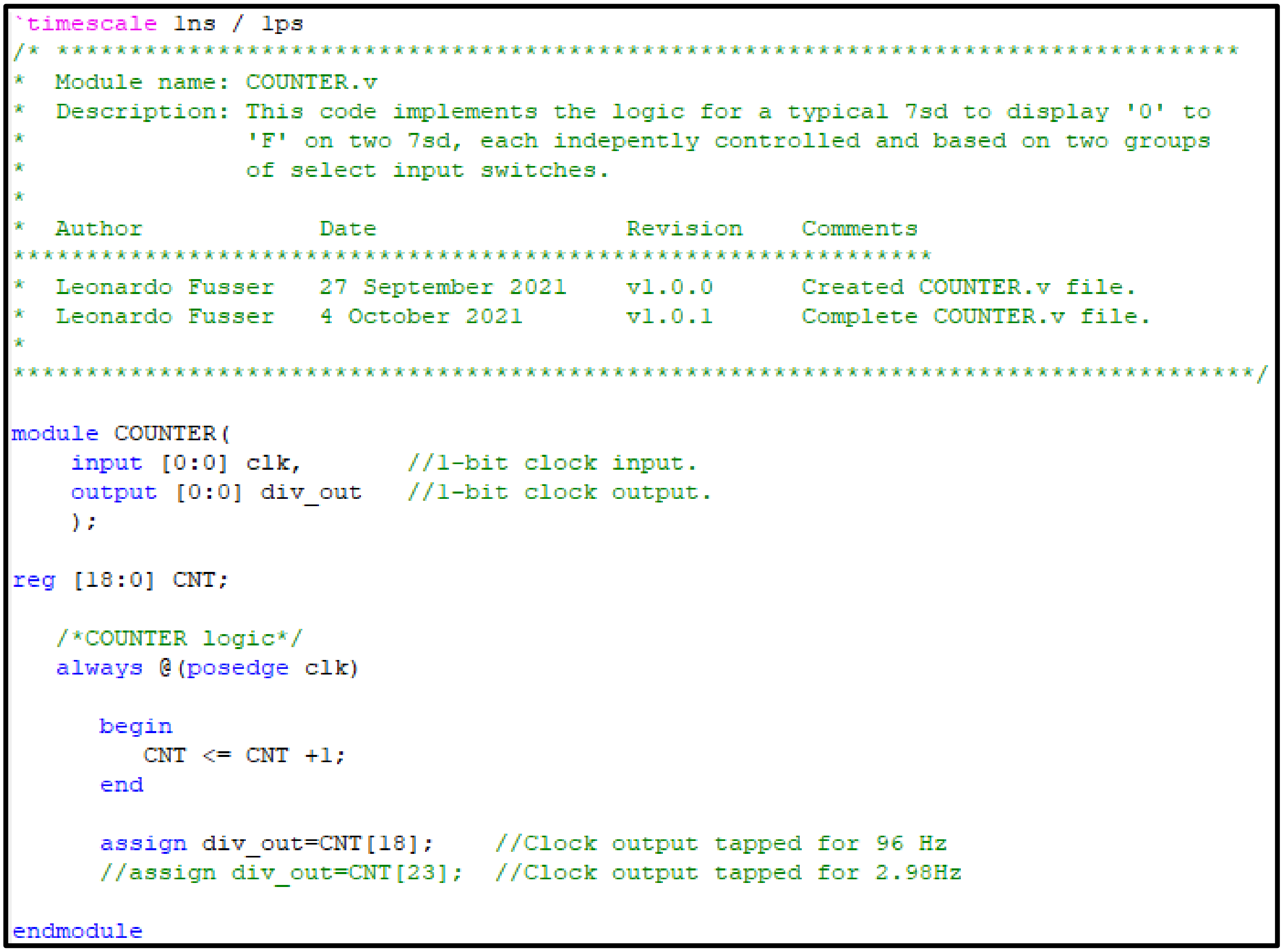
```
`timescale 1ns / 1ps
/* ****************************************************************************
*  Module name: Lab3B_TOP.v
*  Description: This code implements the logic for a typical 7sd to display '0' to
*               'F' on multiple 7sd, based on select input switches and pushbuttons.
*
*  Author             Date                    Revision     Comments
***************************************************************
*  Leonardo Fusser    20 September 2021     v1.0.0       Created Lab3B_TOP.v file.
*                                                        Completed Lab3B_TOP.v file.
*
*****************************************************************************/

module Lab3B_TOP(
   input [3:0] sw,       //4-bit select input.
   input [3:0] btn,      //4-bit select input.
   output [6:0] seg,     //7-bit select output.
   output [3:0] an       //4-bit select output.
   );

   assign an = ~btn;     //7sd select logic.

   /*Instantiation for simple 7sd*/
   Lab3B U1(
      .SW(sw),        //Connects 4 inputs (in Lab3B.v) to 4 select inputs.
      .CA_to_CG(seg) //Connects 7 outputs (in Lab3B.v) to 7 select outputs.
   );

endmodule
```

[MUX.v -> Part C]



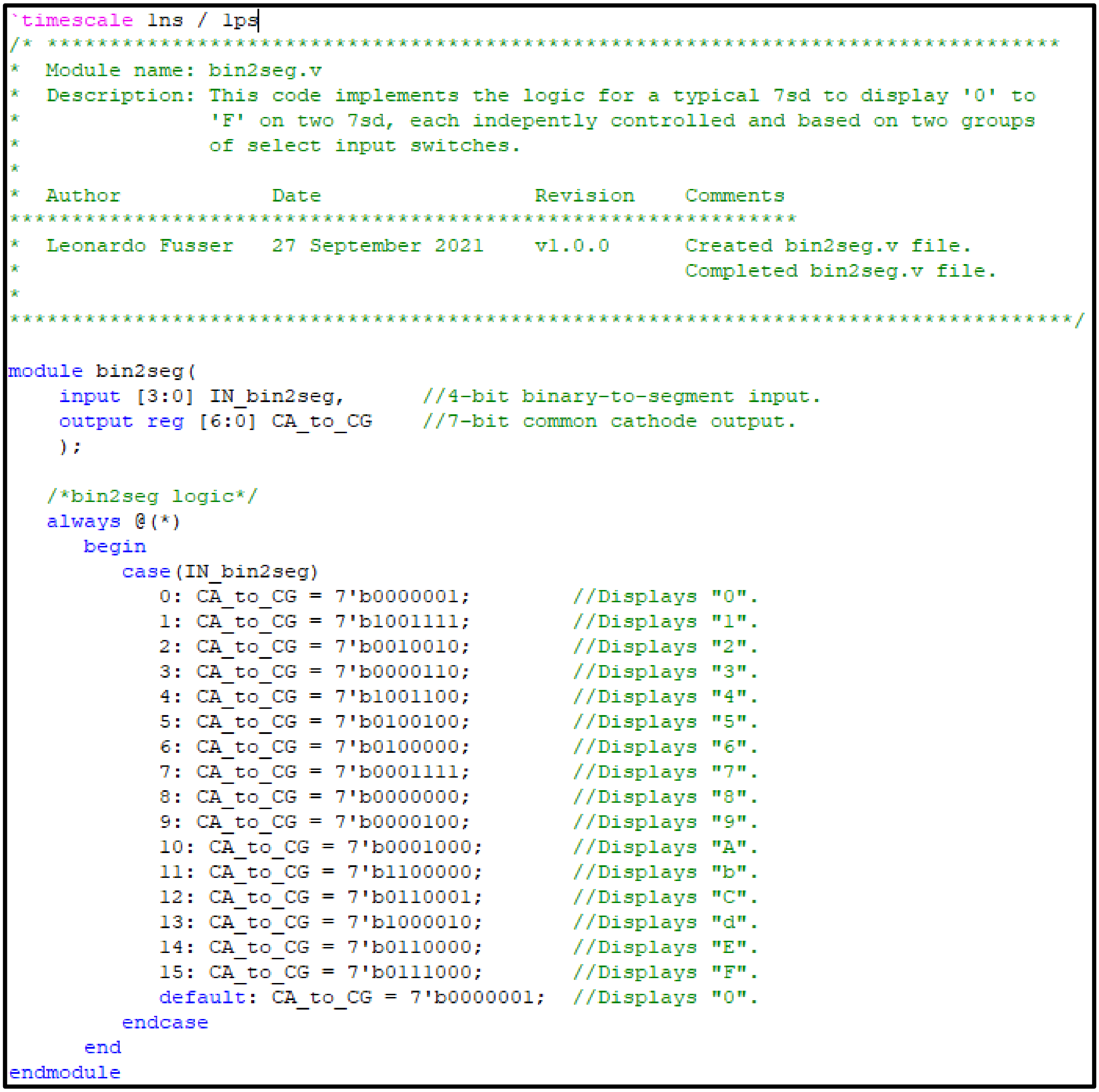
```
`timescale 1ns / 1ps
/* ****************************************************************************
*   Module name: MUX.v
*   Description: This code implements the logic for a typical 7sd to display '0' to
*                'F' on two 7sd, each indepently controlled and based on two groups
*                of select input switches.
*
*   Author              Date                    Revision    Comments
***************************************************************
*   Leonardo Fusser     27 September 2021       v1.0.0      Created MUX.v file.
*   Leonardo Fusser     4 October 2021          v1.0.1      Completed MUX.v file.
*
*****************************************************************************/

module MUX(
    input [3:0] A,      //4-bit mux input.
    input [3:0] B,      //4-bit mux input.
    input [0:0] S,      //1-bit mux select input.
    output reg [3:0] Y  //4-bit mux output.
    );

   /*MUX logic*/
   always @(*)
      if(S == 0)     //If mux select bit = 0...
         begin
            Y = A;   //Mux output = value stored in A.
         end
      else           //If mux select bit = 1...
         begin
            Y = B;   //Mux output = value stored in B.
         end

endmodule
```

[COUNTER.v -> Part C]



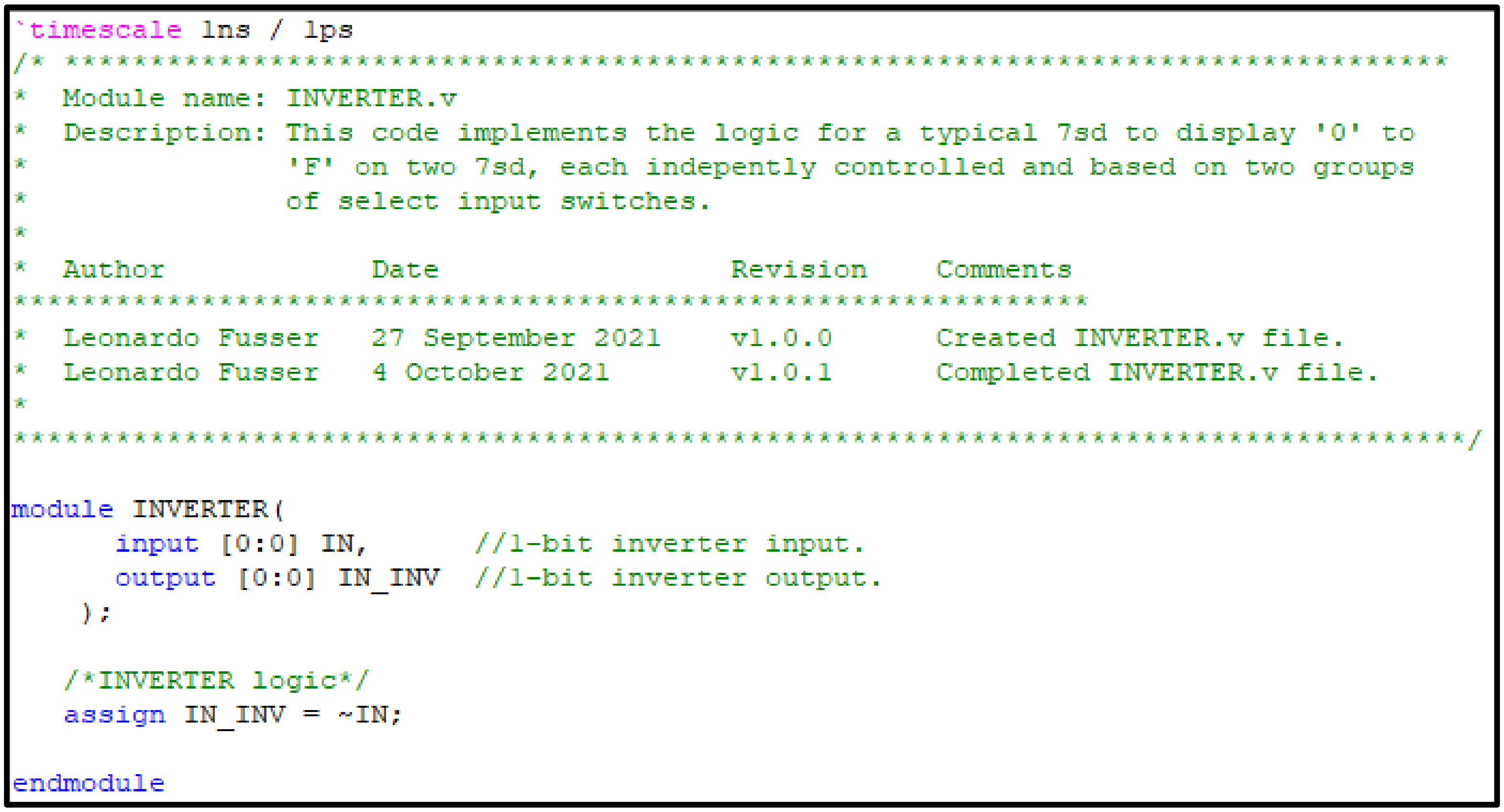
```
`timescale 1ns / 1ps
/* ***************************************************************************************
*  Module name: COUNTER.v
*  Description: This code implements the logic for a typical 7sd to display '0' to
*               'F' on two 7sd, each indepently controlled and based on two groups
*               of select input switches.
*
*  Author             Date                    Revision     Comments
*****************************************************************
*  Leonardo Fusser    27 September 2021       v1.0.0       Created COUNTER.v file.
*  Leonardo Fusser    4 October 2021          v1.0.1       Complete COUNTER.v file.
*
*****************************************************************************************/

module COUNTER(
    input [0:0] clk,        //1-bit clock input.
    output [0:0] div_out    //1-bit clock output.
    );

reg [18:0] CNT;

   /*COUNTER logic*/
   always @(posedge clk)

      begin
         CNT <= CNT +1;
      end

      assign div_out=CNT[18];     //Clock output tapped for 96 Hz
      //assign div_out=CNT[23];   //Clock output tapped for 2.98Hz

endmodule
```

[bin2seg.v -> Part C]



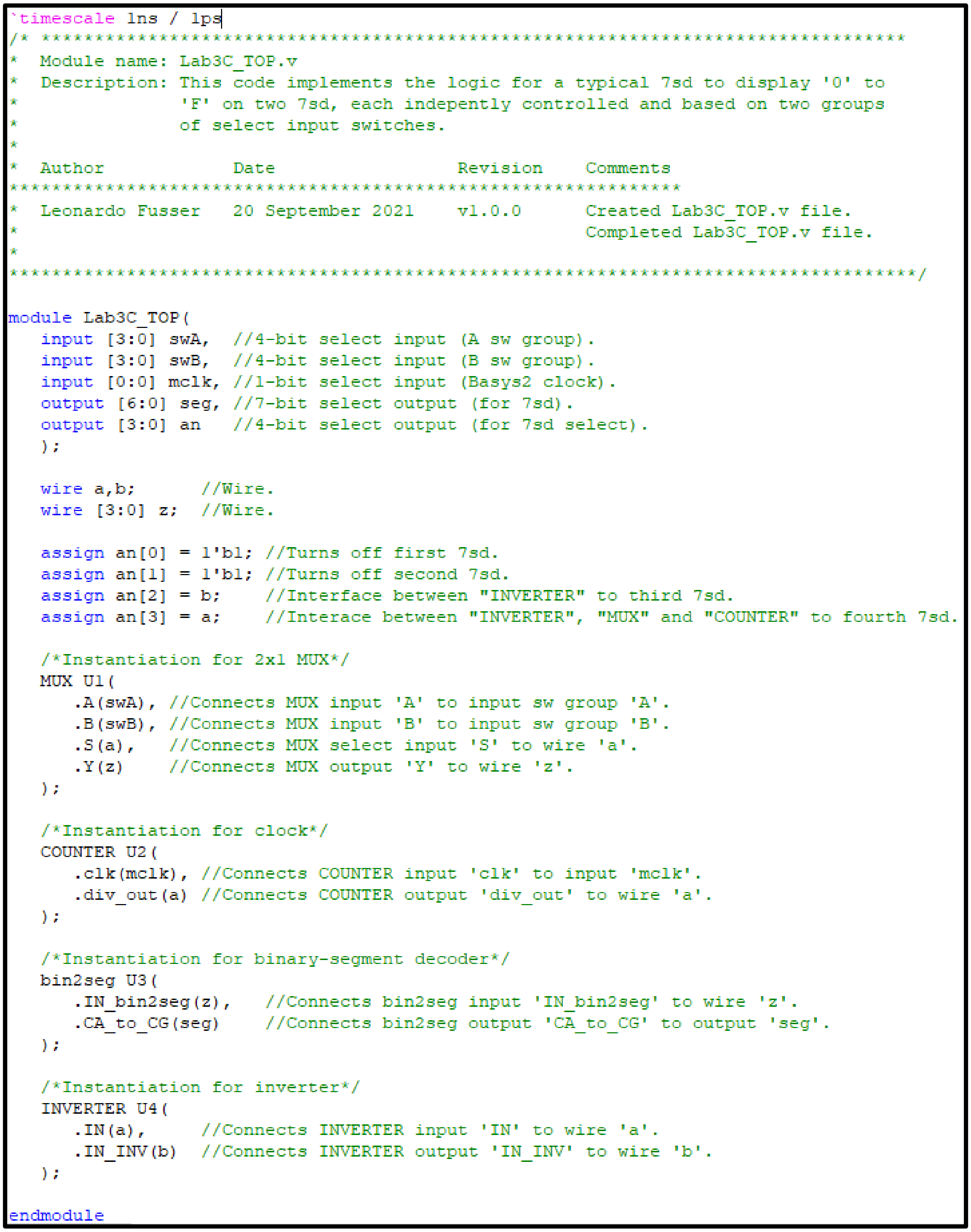
```
`timescale 1ns / 1ps
/* ***********************************************************************************
*  Module name: bin2seg.v
*  Description: This code implements the logic for a typical 7sd to display '0' to
*               'F' on two 7sd, each indepently controlled and based on two groups
*               of select input switches.
*
*  Author             Date                 Revision    Comments
***************************************************************
*  Leonardo Fusser    27 September 2021    v1.0.0      Created bin2seg.v file.
*                                                      Completed bin2seg.v file.
*
*************************************************************************************/

module bin2seg(
    input [3:0] IN_bin2seg,       //4-bit binary-to-segment input.
    output reg [6:0] CA_to_CG     //7-bit common cathode output.
    );

   /*bin2seg logic*/
   always @(*)
      begin
         case(IN_bin2seg)
            0: CA_to_CG = 7'b0000001;          //Displays "0".
            1: CA_to_CG = 7'b1001111;          //Displays "1".
            2: CA_to_CG = 7'b0010010;          //Displays "2".
            3: CA_to_CG = 7'b0000110;          //Displays "3".
            4: CA_to_CG = 7'b1001100;          //Displays "4".
            5: CA_to_CG = 7'b0100100;          //Displays "5".
            6: CA_to_CG = 7'b0100000;          //Displays "6".
            7: CA_to_CG = 7'b0001111;          //Displays "7".
            8: CA_to_CG = 7'b0000000;          //Displays "8".
            9: CA_to_CG = 7'b0000100;          //Displays "9".
            10: CA_to_CG = 7'b0001000;         //Displays "A".
            11: CA_to_CG = 7'b1100000;         //Displays "b".
            12: CA_to_CG = 7'b0110001;         //Displays "C".
            13: CA_to_CG = 7'b1000010;         //Displays "d".
            14: CA_to_CG = 7'b0110000;         //Displays "E".
            15: CA_to_CG = 7'b0111000;         //Displays "F".
            default: CA_to_CG = 7'b0000001;    //Displays "0".
         endcase
      end
endmodule
```

[INVERTER.v -> Part C]



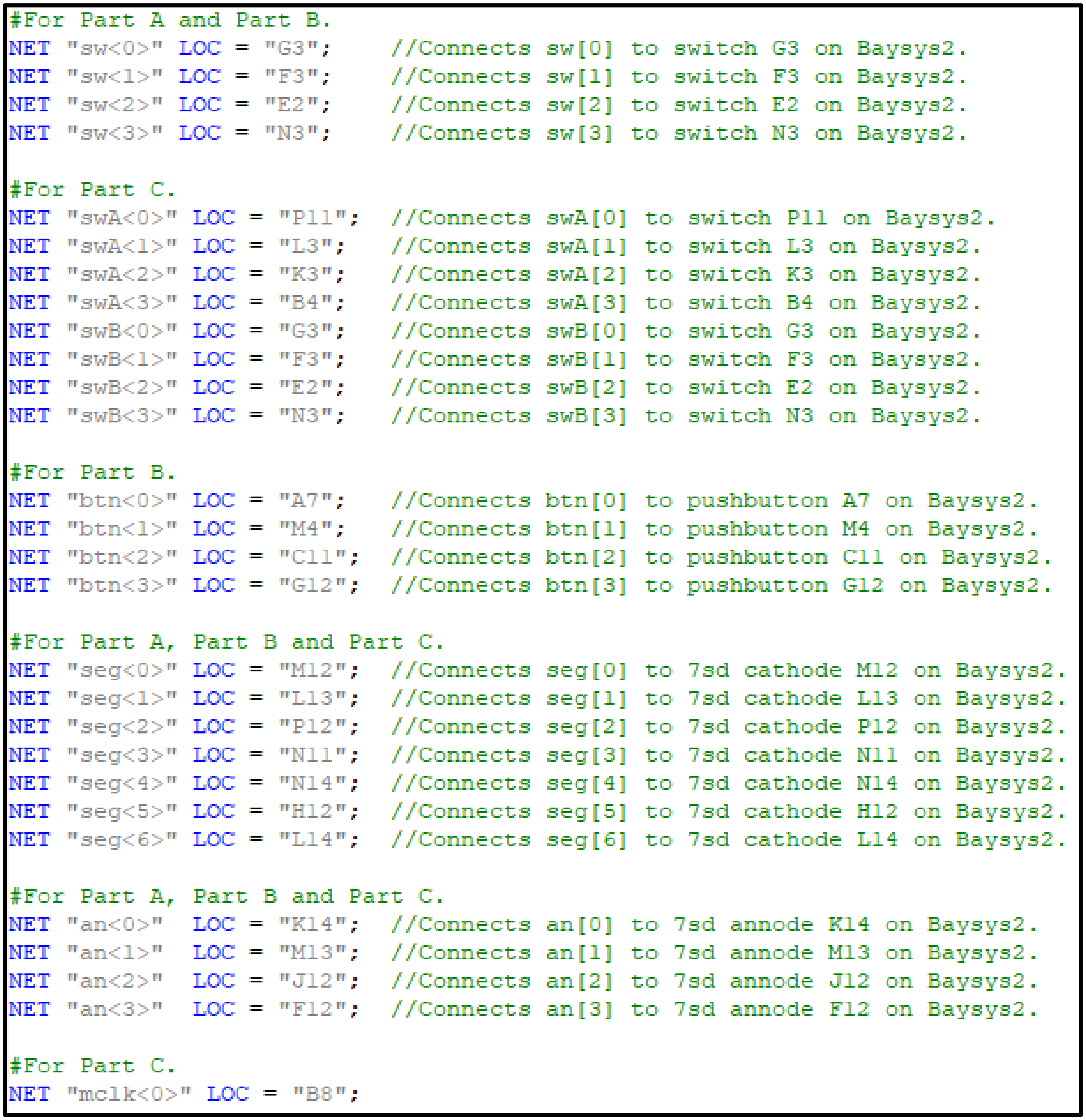
```
`timescale 1ns / 1ps
/* ************************************************************************************
*   Module name: INVERTER.v
*   Description: This code implements the logic for a typical 7sd to display '0' to
*                'F' on two 7sd, each indepently controlled and based on two groups
*                of select input switches.
*
*   Author              Date                  Revision     Comments
**************************************************************
*   Leonardo Fusser    27 September 2021     v1.0.0       Created INVERTER.v file.
*   Leonardo Fusser    4 October 2021        v1.0.1       Completed INVERTER.v file.
*
************************************************************************************/

module INVERTER(
      input [0:0] IN,      //1-bit inverter input.
      output [0:0] IN_INV  //1-bit inverter output.
    );

   /*INVERTER logic*/
   assign IN_INV = ~IN;

endmodule
```

[Lab3C_TOP.v -> Part C]



```
`timescale 1ns / 1ps
/* ***********************************************************************************
*  Module name: Lab3C_TOP.v
*  Description: This code implements the logic for a typical 7sd to display '0' to
*               'F' on two 7sd, each indepently controlled and based on two groups
*               of select input switches.
*
*  Author            Date                   Revision    Comments
*****************************************************************
*  Leonardo Fusser   20 September 2021     v1.0.0      Created Lab3C_TOP.v file.
*                                                      Completed Lab3C_TOP.v file.
*
***********************************************************************************/

module Lab3C_TOP(
   input [3:0] swA,  //4-bit select input (A sw group).
   input [3:0] swB,  //4-bit select input (B sw group).
   input [0:0] mclk, //1-bit select input (Basys2 clock).
   output [6:0] seg, //7-bit select output (for 7sd).
   output [3:0] an   //4-bit select output (for 7sd select).
   );

   wire a,b;      //Wire.
   wire [3:0] z;  //Wire.

   assign an[0] = 1'b1; //Turns off first 7sd.
   assign an[1] = 1'b1; //Turns off second 7sd.
   assign an[2] = b;    //Interface between "INVERTER" to third 7sd.
   assign an[3] = a;    //Interace between "INVERTER", "MUX" and "COUNTER" to fourth 7sd.

   /*Instantiation for 2x1 MUX*/
   MUX U1(
      .A(swA), //Connects MUX input 'A' to input sw group 'A'.
      .B(swB), //Connects MUX input 'B' to input sw group 'B'.
      .S(a),   //Connects MUX select input 'S' to wire 'a'.
      .Y(z)    //Connects MUX output 'Y' to wire 'z'.
   );

   /*Instantiation for clock*/
   COUNTER U2(
      .clk(mclk), //Connects COUNTER input 'clk' to input 'mclk'.
      .div_out(a) //Connects COUNTER output 'div_out' to wire 'a'.
   );

   /*Instantiation for binary-segment decoder*/
   bin2seg U3(
      .IN_bin2seg(z),   //Connects bin2seg input 'IN_bin2seg' to wire 'z'.
      .CA_to_CG(seg)    //Connects bin2seg output 'CA_to_CG' to output 'seg'.
   );

   /*Instantiation for inverter*/
   INVERTER U4(
      .IN(a),     //Connects INVERTER input 'IN' to wire 'a'.
      .IN_INV(b)  //Connects INVERTER output 'IN_INV' to wire 'b'.
   );

endmodule
```

[Lab3_CONSTRAINTS.ucf -> Part A, Part B and Part C]



```
#For Part A and Part B.
NET "sw<0>" LOC = "G3";    //Connects sw[0] to switch G3 on Baysys2.
NET "sw<1>" LOC = "F3";    //Connects sw[1] to switch F3 on Baysys2.
NET "sw<2>" LOC = "E2";    //Connects sw[2] to switch E2 on Baysys2.
NET "sw<3>" LOC = "N3";    //Connects sw[3] to switch N3 on Baysys2.

#For Part C.
NET "swA<0>" LOC = "P11";  //Connects swA[0] to switch P11 on Baysys2.
NET "swA<1>" LOC = "L3";   //Connects swA[1] to switch L3 on Baysys2.
NET "swA<2>" LOC = "K3";   //Connects swA[2] to switch K3 on Baysys2.
NET "swA<3>" LOC = "B4";   //Connects swA[3] to switch B4 on Baysys2.
NET "swB<0>" LOC = "G3";   //Connects swB[0] to switch G3 on Baysys2.
NET "swB<1>" LOC = "F3";   //Connects swB[1] to switch F3 on Baysys2.
NET "swB<2>" LOC = "E2";   //Connects swB[2] to switch E2 on Baysys2.
NET "swB<3>" LOC = "N3";   //Connects swB[3] to switch N3 on Baysys2.

#For Part B.
NET "btn<0>" LOC = "A7";   //Connects btn[0] to pushbutton A7 on Baysys2.
NET "btn<1>" LOC = "M4";   //Connects btn[1] to pushbutton M4 on Baysys2.
NET "btn<2>" LOC = "C11";  //Connects btn[2] to pushbutton C11 on Baysys2.
NET "btn<3>" LOC = "G12";  //Connects btn[3] to pushbutton G12 on Baysys2.

#For Part A, Part B and Part C.
NET "seg<0>" LOC = "M12";  //Connects seg[0] to 7sd cathode M12 on Baysys2.
NET "seg<1>" LOC = "L13";  //Connects seg[1] to 7sd cathode L13 on Baysys2.
NET "seg<2>" LOC = "P12";  //Connects seg[2] to 7sd cathode P12 on Baysys2.
NET "seg<3>" LOC = "N11";  //Connects seg[3] to 7sd cathode N11 on Baysys2.
NET "seg<4>" LOC = "N14";  //Connects seg[4] to 7sd cathode N14 on Baysys2.
NET "seg<5>" LOC = "H12";  //Connects seg[5] to 7sd cathode H12 on Baysys2.
NET "seg<6>" LOC = "L14";  //Connects seg[6] to 7sd cathode L14 on Baysys2.

#For Part A, Part B and Part C.
NET "an<0>"  LOC = "K14";  //Connects an[0] to 7sd annode K14 on Baysys2.
NET "an<1>"  LOC = "M13";  //Connects an[1] to 7sd annode M13 on Baysys2.
NET "an<2>"  LOC = "J12";  //Connects an[2] to 7sd annode J12 on Baysys2.
NET "an<3>"  LOC = "F12";  //Connects an[3] to 7sd annode F12 on Baysys2.

#For Part C.
NET "mclk<0>" LOC = "B8";
```

Questions:

1. What is the alternative way to implement the 7-segment decoder? Show sample codes of the alternative way. Which method would you prefer, and why?

   - An alternative way to implement the 7-segment decoder would be to use multiple if/else if statements to cover all the possible combinations for the 7-segment decoder. It would look something like this:

   ```
   if (INPUT == 0){OUTPUT = 7'b00000001;}      //Displays "0".
   else if (INPUT == 1){OUTPUT = 7'b1001111;} //Displays "1".
   ...
   else {OUTPUT = 7'b00000001;}                //Displays "0".
   ```

     The preferred way would be what is shown in the code on previous pages, using a case statement to cover all the possible combinations for the 7-segment decoder (see Lab3A.v on previous pages). This is an easier way to do implement the same logic as the functionality is clearer when using this method.

2. What are the advantages of a hierarchical design?

   - The main advantage of a hierarchical design is flexibility. If a hierarchical design is followed, you can add or remove code easily within a project without much trouble. This is especially useful when unexpected changes that require addition to code are required. Another advantage of a hierarchical design is ease of debugging. When trouble occurs, and extensive debugging is required, entire portions of code can be disabled or enabled easily, which saves time while debugging issues in code. This is especially useful in projects with large amounts of code, for obvious reasons.

3. What does AN do and why it is necessary?

   - ANx (AN0 – AN3) refers to the four anode pins on the 7-segment displays on the Baysys2 board. On the Baysys2 board, the 7-segment displays are in a common cathode arrangement. To enable or disable the 7-segment displays, the corresponding ANx for the display has to be set to '0' to turn on the specific display or set to '1' to turn off the specific display. Without the ANx pins, we would not be able to control the individual 7-segment displays on the Baysys2 board.

4. What problems did you experience, how did you overcome them? What did you learn in this experiment?

   - Many of the problems encountered were due to misunderstanding the file structure of the Xilinx ISE software (did not know how to organize different Verilog files properly). This was solved by reading documentation provided in the software in regard to file structure/organization. Other issues encountered were minor issues, such as not coding the top module properly (example: wires not created/defined properly, module instantiation not correct, etc...). The main takeaway from this lab was the functioning of a "real" 7-segment display (how to control multiple displays individually). The functionality basics of the 7-segment display was also something learned in this lab. Other takeaways such as Xilinx ISE proper file structure was also something learned in this lab.