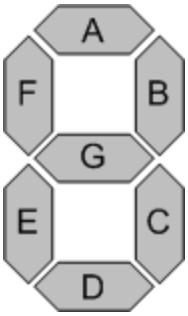
Go Board Project 5 – 7 Segment Displays

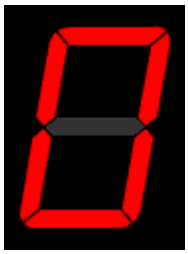
Video: https://youtu.be/iT9MVuIZFJ8
Link: https://nandland.com/project-5-seven-segment-display/

Let's learn about these fun peripherals

The previous project taught us about debouncing a switch, and how to instantiate one module from another. Thus far, we've only been using **Switches** and **LEDs** to learn about FPGAs.
Let's introduce a new Go Board peripheral. The Go Board has a two digit 7-Segment display.
This project will teach you how this works and how to drive it by building on the code from the previous project.

I also created a YouTube video for this project, should you prefer to follow along with that.



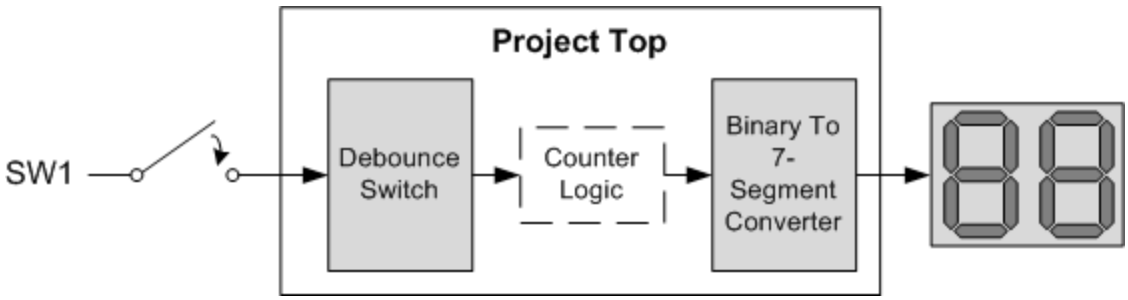


First, let's show how a 7-Segment Display works.
On the Go Board, there are two 7-Segment Displays. Each display has 7-inputs, each input drives one particular segment of the display. The segments are labeled in the figure to the right.
So for example, in order to create the number 1, Segments B and C need to be illuminated.
You've probably only ever seen displays that show the numbers 0-9.
However it should be noted that it's possible to display the hex characters A-F, as shown in the GIF to the left. This might be useful in a project coming up… hint hint.
For this project, you'll need to figure out when to drive each individual segment.

Project Description

Increment a Binary Counter to Drive One Digit of the 7-Segment Display Each Time a Switch is Released.

The previous project toggled the state of the LED when the switch was released. This project will keep track of the number of times (0-9) that the button was pressed in a counter. Once the counter gets up to 9, it will reset back to 0 and continue to count up.
The value of the counter should be displayed on the 7-Segment Display. The image below shows the block diagram concept for how the code is designed.



Project Block Diagram – Seven Segment Display

Your skills are improving. Try this project on your own. If you need any hints, you can refer to the solution below.

VHDL Solution

Most of this code should look familiar to you. Not many new concepts are introduced here.
The project makes use of the **Debounce_Switch module** that we have designed.
Additionally, there's a new module that is Binary_To_7Segment. Take a look at the solution below, as well as the code in the **Binary_To_7Segment file** and make sure you understand everything.

VHDL Code – Project_7_Segment_Top.vhd:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity Project_7_Segment_Top is
6   port (
7     -- Main Clock (25 MHz)
8     i_Clk         : in std_logic;
9
10    -- Input Switch
11    i_Switch_1    : in std_logic;
12
13    -- Segment2 is lower digit, Segment1 is upper digit
14    o_Segment2_A  : out std_logic;
15    o_Segment2_B  : out std_logic;
16    o_Segment2_C  : out std_logic;
17    o_Segment2_D  : out std_logic;
18    o_Segment2_E  : out std_logic;
19    o_Segment2_F  : out std_logic;
20    o_Segment2_G  : out std_logic
21    );
22 end entity Project_7_Segment_Top;
23
24 architecture RTL of Project_7_Segment_Top is
25
```

```
26  signal w_Switch_1 : std_logic;
27  signal r_Switch_1 : std_logic := '0';
28  signal r_Count    : integer range 0 to 9 := 0;
29
30  signal w_Segment2_A : std_logic;
31  signal w_Segment2_B : std_logic;
32  signal w_Segment2_C : std_logic;
33  signal w_Segment2_D : std_logic;
34  signal w_Segment2_E : std_logic;
35  signal w_Segment2_F : std_logic;
36  signal w_Segment2_G : std_logic;
37
38begin
39
40  -- Instantiate Debounce Filter
41  Debounce_Inst : entity work.Debounce_Switch
42    port map (
43      i_Clk    => i_Clk,
44      i_Switch => i_Switch_1,
45      o_Switch => w_Switch_1);
46
47  -- Purpose: When Switch is pressed, increment counter.
48  -- When counter gets to 9, start it back at 0 again.
49  p_Switch_Count : process (i_Clk)
50  begin
51    if rising_edge(i_Clk) then
52      r_Switch_1 <= w_Switch_1;
53
54      -- Increment Count when switch is pushed down
55      if (w_Switch_1 = '1' and r_Switch_1 = '0') then
56        if (r_Count = 9) then
57          r_Count <= 0;
58        else
59          r_Count <= r_Count + 1;
60        end if;
61      end if;
62    end if;
63  end process p_Switch_Count;
64
65  -- Instantiate Binary to 7-Segment Converter
66  SevenSeg1_Inst : entity work.Binary_To_7Segment
67    port map (
68      i_Clk        => i_Clk,
69      i_Binary_Num => std_logic_vector(to_unsigned(r_Count, 4)),
70      o_Segment_A  => w_Segment2_A,
71      o_Segment_B  => w_Segment2_B,
72      o_Segment_C  => w_Segment2_C,
73      o_Segment_D  => w_Segment2_D,
74      o_Segment_E  => w_Segment2_E,
75      o_Segment_F  => w_Segment2_F,
76      o_Segment_G  => w_Segment2_G
77      );
```

```
78
79  o_Segment2_A <= not w_Segment2_A;
80  o_Segment2_B <= not w_Segment2_B;
81  o_Segment2_C <= not w_Segment2_C;
82  o_Segment2_D <= not w_Segment2_D;
83  o_Segment2_E <= not w_Segment2_E;
84  o_Segment2_F <= not w_Segment2_F;
85  o_Segment2_G <= not w_Segment2_G;
86
87end architecture RTL;
```

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity Project_7_Segment_Top is
6   port (
7     -- Main Clock (25 MHz)
8     i_Clk          : in std_logic;
9
10    -- Input Switch
11    i_Switch_1     : in std_logic;
12
13    -- Segment2 is lower digit, Segment1 is upper digit
14    o_Segment2_A   : out std_logic;
15    o_Segment2_B   : out std_logic;
16    o_Segment2_C   : out std_logic;
17    o_Segment2_D   : out std_logic;
18    o_Segment2_E   : out std_logic;
19    o_Segment2_F   : out std_logic;
20    o_Segment2_G   : out std_logic
21    );
22end entity Project_7_Segment_Top;
23
```

```
24architecture RTL of Project_7_Segment_Top is
25
26  signal w_Switch_1 : std_logic;
27  signal r_Switch_1 : std_logic := '0';
28  signal r_Count    : integer range 0 to 9 := 0;
29
30  signal w_Segment2_A : std_logic;
31  signal w_Segment2_B : std_logic;
32  signal w_Segment2_C : std_logic;
33  signal w_Segment2_D : std_logic;
34  signal w_Segment2_E : std_logic;
35  signal w_Segment2_F : std_logic;
36  signal w_Segment2_G : std_logic;
37
38begin
39
40  -- Instantiate Debounce Filter
41  Debounce_Inst : entity work.Debounce_Switch
42    port map (
43      i_Clk    => i_Clk,
44      i_Switch => i_Switch_1,
45      o_Switch => w_Switch_1);
46
47  -- Purpose: When Switch is pressed, increment counter.
48  -- When counter gets to 9, start it back at 0 again.
49  p_Switch_Count : process (i_Clk)
50  begin
51    if rising_edge(i_Clk) then
52      r_Switch_1 <= w_Switch_1;
53
54      -- Increment Count when switch is pushed down
```

```
55      if (w_Switch_1 = '1' and r_Switch_1 = '0') then
56        if (r_Count = 9) then
57          r_Count <= 0;
58        else
59          r_Count <= r_Count + 1;
60        end if;
61      end if;
62    end if;
63  end process p_Switch_Count;
64
65  -- Instantiate Binary to 7-Segment Converter
66  SevenSeg1_Inst : entity work.Binary_To_7Segment
67    port map (
68      i_Clk        => i_Clk,
69      i_Binary_Num => std_logic_vector(to_unsigned(r_Count, 4)),
70      o_Segment_A  => w_Segment2_A,
71      o_Segment_B  => w_Segment2_B,
72      o_Segment_C  => w_Segment2_C,
73      o_Segment_D  => w_Segment2_D,
74      o_Segment_E  => w_Segment2_E,
75      o_Segment_F  => w_Segment2_F,
76      o_Segment_G  => w_Segment2_G
77      );
78
79  o_Segment2_A <= not w_Segment2_A;
80  o_Segment2_B <= not w_Segment2_B;
81  o_Segment2_C <= not w_Segment2_C;
82  o_Segment2_D <= not w_Segment2_D;
83  o_Segment2_E <= not w_Segment2_E;
84  o_Segment2_F <= not w_Segment2_F;
85  o_Segment2_G <= not w_Segment2_G;
```

```
86

87end architecture RTL;
```

Verilog Solution

Most of this code should look familiar to you. Not many new concepts are introduced here.
The project makes use of the **Debounce_Switch module** that we have designed.
Additionally, there's a new module that is Binary_To_7Segment. Take a look at the solution below, as well as the code in the **Binary_To_7Segment file** and make sure you understand everything.

Verilog Code – Project_7_Segment_Top.v:

```
1module Project_7_Segment_Top
2  (input  i_Clk,      // Main Clock (25 MHz)
3   input  i_Switch_1,
4   output o_Segment2_A,
5   output o_Segment2_B,
6   output o_Segment2_C,
7   output o_Segment2_D,
8   output o_Segment2_E,
9   output o_Segment2_F,
10  output o_Segment2_G
11  );
12
13 wire w_Switch_1;
14 reg  r_Switch_1 = 1'b0;
15 reg [3:0] r_Count = 4'b0000;
16
17 wire w_Segment2_A;
18 wire w_Segment2_B;
19 wire w_Segment2_C;
20 wire w_Segment2_D;
21 wire w_Segment2_E;
22 wire w_Segment2_F;
23 wire w_Segment2_G;
24
25 // Instantiate Debounce Filter
26 Debounce_Switch Debounce_Switch_Inst
27   (.i_Clk(i_Clk),
28    .i_Switch(i_Switch_1),
29    .o_Switch(w_Switch_1));
30
31 // Purpose: When Switch is pressed, increment counter.
32 // When counter gets to 9, start it back at 0 again.
33 always @(posedge i_Clk)
```

```
34  begin
35    r_Switch_1 <= w_Switch_1;
36
37      // Increment Count when switch is pushed down
38      if (w_Switch_1 == 1'b1 && r_Switch_1 == 1'b0)
39      begin
40        if (r_Count == 9)
41          r_Count <= 0;
42        else
43          r_Count <= r_Count + 1;
44      end
45  end
46
47  // Instantiate Binary to 7-Segment Converter
48  Binary_To_7Segment Inst
49    (.i_Clk(i_Clk),
50     .i_Binary_Num(r_Count),
51     .o_Segment_A(w_Segment2_A),
52     .o_Segment_B(w_Segment2_B),
53     .o_Segment_C(w_Segment2_C),
54     .o_Segment_D(w_Segment2_D),
55     .o_Segment_E(w_Segment2_E),
56     .o_Segment_F(w_Segment2_F),
57     .o_Segment_G(w_Segment2_G)
58     );
59
60  assign o_Segment2_A = ~w_Segment2_A;
61  assign o_Segment2_B = ~w_Segment2_B;
62  assign o_Segment2_C = ~w_Segment2_C;
63  assign o_Segment2_D = ~w_Segment2_D;
64  assign o_Segment2_E = ~w_Segment2_E;
65  assign o_Segment2_F = ~w_Segment2_F;
66  assign o_Segment2_G = ~w_Segment2_G;
67
68endmodule
```

For both the VHDL and the Verilog solutions above, we instantiated two modules:
Debounce_Switch and
Binary_To_7Segment.
Are you starting to see how more complicated pieces of code get created?

When you design some block of code that has a useful purpose, it's worth considering if it should be in its own module or entity. It makes reuse much easier if you can instantiate one module, rather than copy and pasting the same code in multiple designs.

One thing to keep in mind is there is a sweet-spot for the minimum and maximum amount of code that should be in a single file.
For example, you wouldn't want to wrap up a single AND Gate in a file **AND_Gate.vhd**, there's not enough code in there to make it worth while.
You'll get a feel for deciding when to wrap up code in its own file as you do more designs.
There's nothing functionally wrong with putting *everything* inside one file,

but you'll find it gets unwieldy quickly.

Synthesis Report (selection)

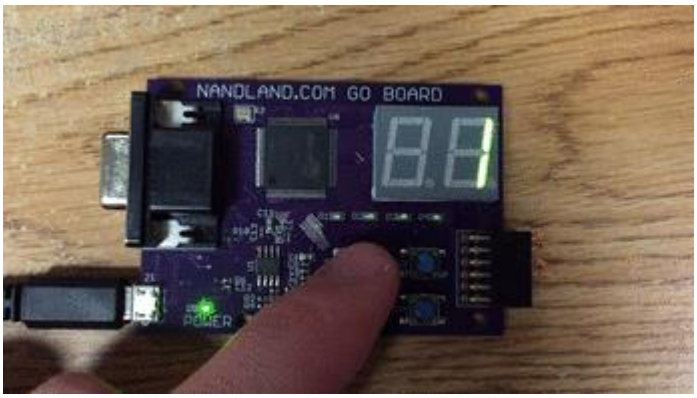
```
Resource Usage Report for Project_7_Segment_Top

I/O ports: 9

I/O Register bits:                  0
Register bits not including I/Os:   31 (2%)
Total load per clock:
   Project_7_Segment_Top|i_Clk: 1

Mapping Summary:
Total  LUTs: 52 (4%)
```

The synthesis report above shows that we are starting to figure out how to use some of our resources, though we still only used two percent of Registers and four percent of LUTs, so we have lots of room in the FPGA for more stuff.



Pressing Switch 1 Increments 7-Segment Display

Great! We have successfully driven the 7-Segment display! Not too bad right?
There's lots of fun projects that are available to you with a couple 7-Segment displays.

I'll leave it to you to get the second one running. Here's a thought, how will you convert a binary number to a base-10 number that can be displayed on two 7-Segment displays?

I'll give a hint for one possible solution:
The **Double Dabbler**.
Now let's move on to our next project.
This one is going to introduce **simulation** and **test benches**.

[Simulating an LED Blink Module]