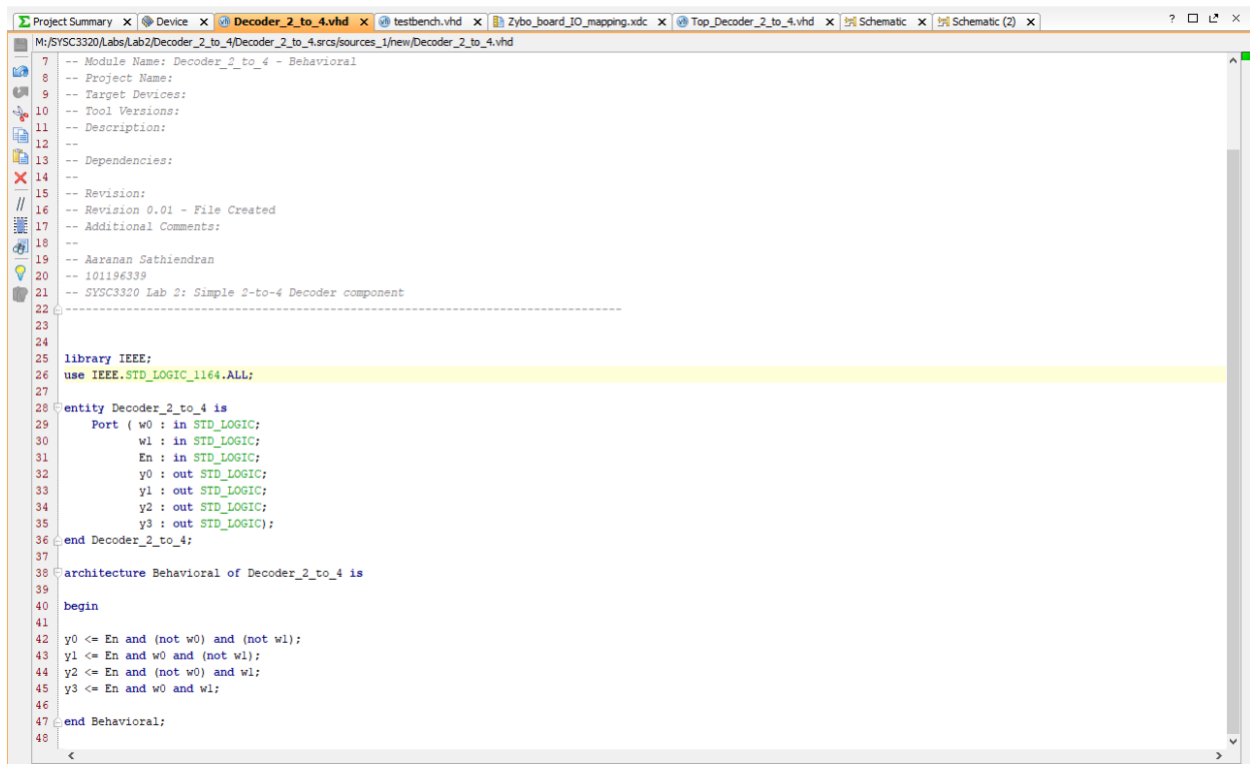


**Department of  
Systems and Computer Engineering**

**SYSC-3320: Laboratory 1  
Decoder and Counter Design and  
Implementation using FPGA and VHDL  
on the Zynq-7000 SoC**

Completed by: Aaranan Sathiendran (101196339)

## Simple 2-to-4 Decoder Component:



The image shows a screenshot of a VHDL code editor window. The title bar includes tabs for 'Project Summary', 'Device', 'Decoder\_2\_to\_4.vhd', 'testbench.vhd', 'Zybo\_board\_IO\_mapping.xdc', 'Top\_Decoder\_2\_to\_4.vhd', 'Schematic', and 'Schematic (2)'. The main editor area displays the VHDL code for a 2-to-4 decoder component. The code is as follows:

```
7  -- Module Name: Decoder_2_to_4 - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -- Asaranan Sathindran
20 -- 101196339
21 -- SYSC3320 Lab 2: Simple 2-to-4 Decoder component
22 -----
23
24
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27
28 entity Decoder_2_to_4 is
29     Port ( w0 : in STD_LOGIC;
30           w1 : in STD_LOGIC;
31           En : in STD_LOGIC;
32           y0 : out STD_LOGIC;
33           y1 : out STD_LOGIC;
34           y2 : out STD_LOGIC;
35           y3 : out STD_LOGIC);
36 end Decoder_2_to_4;
37
38 architecture Behavioral of Decoder_2_to_4 is
39
40     begin
41
42     y0 <= En and (not w0) and (not w1);
43     y1 <= En and w0 and (not w1);
44     y2 <= En and (not w0) and w1;
45     y3 <= En and w0 and w1;
46
47 end Behavioral;
48
```

Figure 1: VHDL code for 2-to-4 decoder

```

19 -- Aaranan Sathindran
20 -- 101196339
21 -- SYSC3320 Lab 2: Testbench for Simple 2-to-4 Decoder component
22 -----
23
24
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27
28 entity testbench is
29 -- Port ( ) :
30 end testbench;
31
32 architecture Behavioral of testbench is
33 SIGNAL Inw0: STD_LOGIC:= '0';
34 SIGNAL Inw1: STD_LOGIC:= '0';
35 SIGNAL InEn: STD_LOGIC:= '0';
36 SIGNAL Outy0: STD_LOGIC:= '0';
37 SIGNAL Outy1: STD_LOGIC:= '0';
38 SIGNAL Outy2: STD_LOGIC:= '0';
39 SIGNAL Outy3: STD_LOGIC:= '0';
40
41 component Decoder_2_to_4 is
42     Port( w0 : in STD_LOGIC;
43           w1 : in STD_LOGIC;
44           En : in STD_LOGIC;
45           y0 : out STD_LOGIC;
46           y1 : out STD_LOGIC;
47           y2 : out STD_LOGIC;
48           y3 : out STD_LOGIC);
49 end component;
50
51 begin
52     Dec_1 : Decoder_2_to_4
53     port map(
54         w0 => Inw0,
55         w1 => Inw1,
56         En => InEn,
57         y0 => Outy0,
58         y1 => Outy1,
59         y2 => Outy2,
60         y3 => Outy3);

```

```

63     InEn <= '1';
64
65     Inw0 <= '0';
66     Inw1 <= '0';
67     wait for 2 ns;
68
69     Inw0 <= '0';
70     Inw1 <= '1';
71     wait for 2 ns;
72
73     Inw0 <= '1';
74     Inw1 <= '0';
75     wait for 2 ns;
76
77     Inw0 <= '1';
78     Inw1 <= '1';
79     wait for 2 ns;
80
81     Inw0 <= '0';
82     Inw1 <= '0';
83     wait for 2 ns;
84
85     InEn <= '0';
86
87     Inw0 <= '0';
88     Inw1 <= '0';
89     wait for 2 ns;
90
91     Inw0 <= '0';
92     Inw1 <= '1';
93     wait for 2 ns;
94
95     Inw0 <= '1';
96     Inw1 <= '0';
97     wait for 2 ns;
98
99     Inw0 <= '1';
100    Inw1 <= '1';
101    wait for 2 ns;
102
103    Inw0 <= '0';
104    Inw1 <= '0';

```

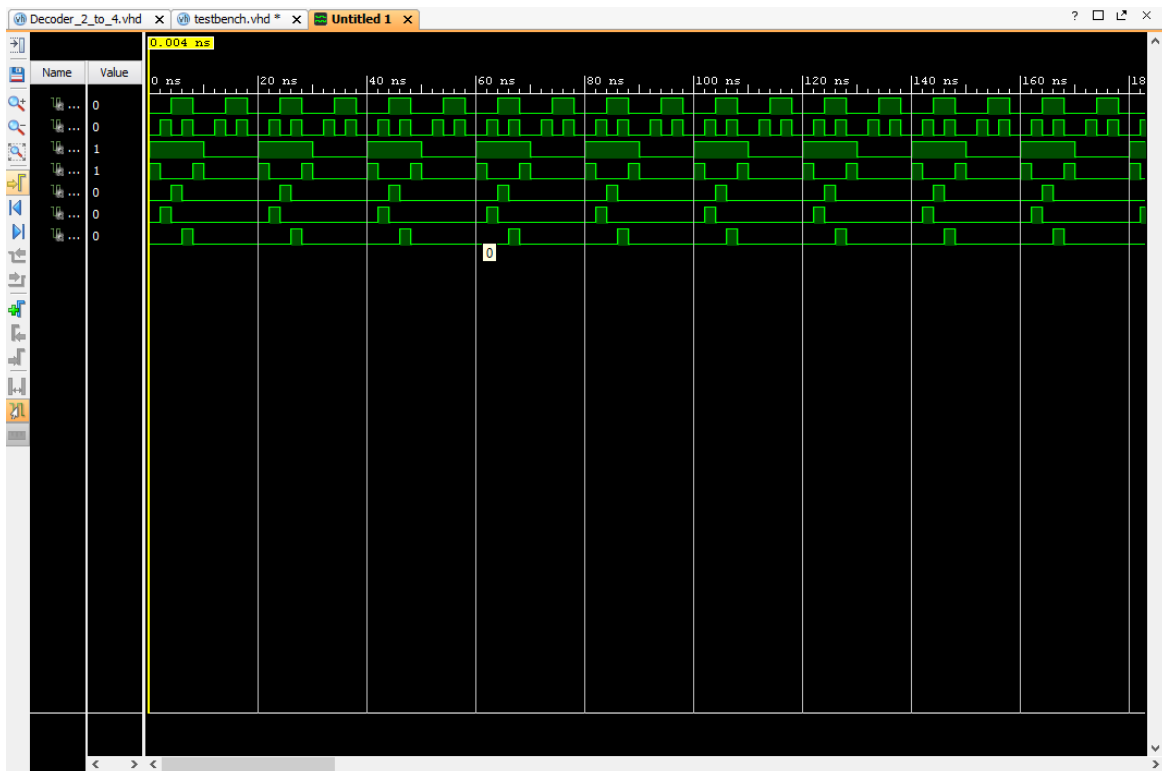
**Figure 2: VHDL code for 2-to-4 decoder  
testbench**

```

17 -- Additional Comments:
18 --
19 -- Aaranan Sathidran
20 -- 101196339
21 -- SYS3320 Lab 2: Top level design for Simple 2-to-4 Decoder component
22 -----
23
24 library IEEE;
25 use IEEE.STD_LOGIC_1164.ALL;
26
27 entity Top_Decoder_2_to_4 is
28     Port ( btn : in STD_LOGIC_VECTOR (3 downto 0);
29           sw : in STD_LOGIC_VECTOR (3 downto 0);
30           led : out STD_LOGIC_VECTOR (3 downto 0));
31 end Top_Decoder_2_to_4;
32
33 architecture Behavioral of Top_Decoder_2_to_4 is
34
35     component Decoder_2_to_4 is
36         Port( w0 : in STD_LOGIC;
37              w1 : in STD_LOGIC;
38              En : in STD_LOGIC;
39              y0 : out STD_LOGIC;
40              y1 : out STD_LOGIC;
41              y2 : out STD_LOGIC;
42              y3 : out STD_LOGIC);
43     end component;
44
45     begin
46
47         D1 : Decoder_2_to_4
48         port map(
49             w0 => btn(0),
50             w1 => btn(1),
51             En => sw(0),
52             y0 => led(0),
53             y1 => led(1),
54             y2 => led(2),
55             y3 => led(3));
56
57     end Behavioral;
58

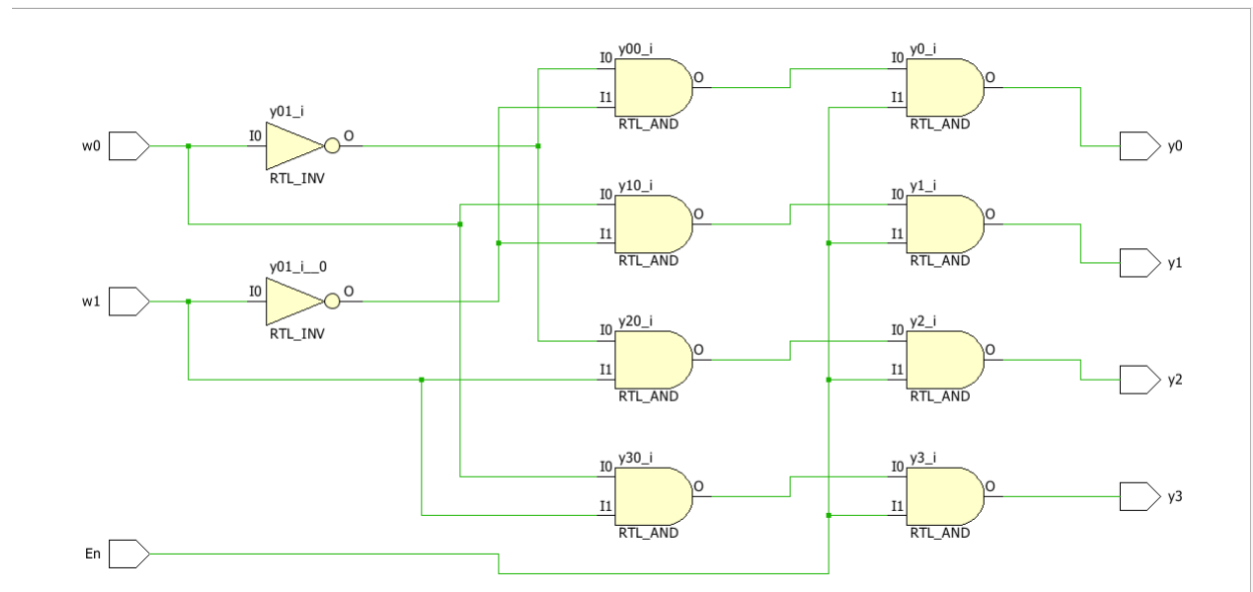
```

**Figure 3: VHDL code for creating top level for 2-to-4 decoder**

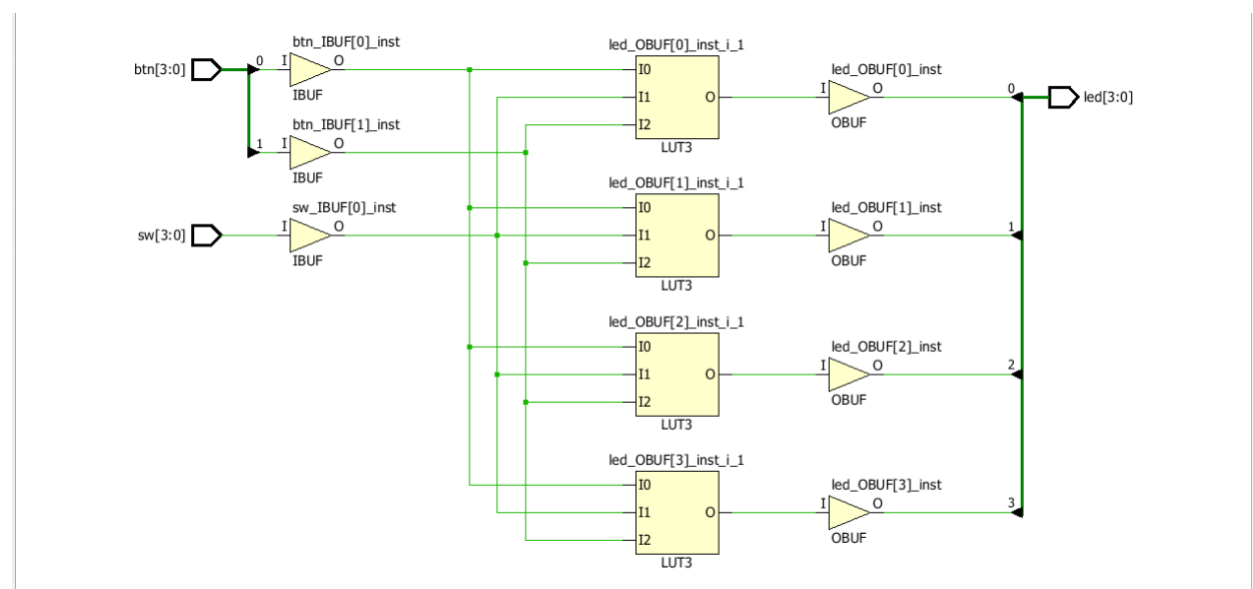


**Figure 4: Simulation timing diagram for 2-to-4 decoder**

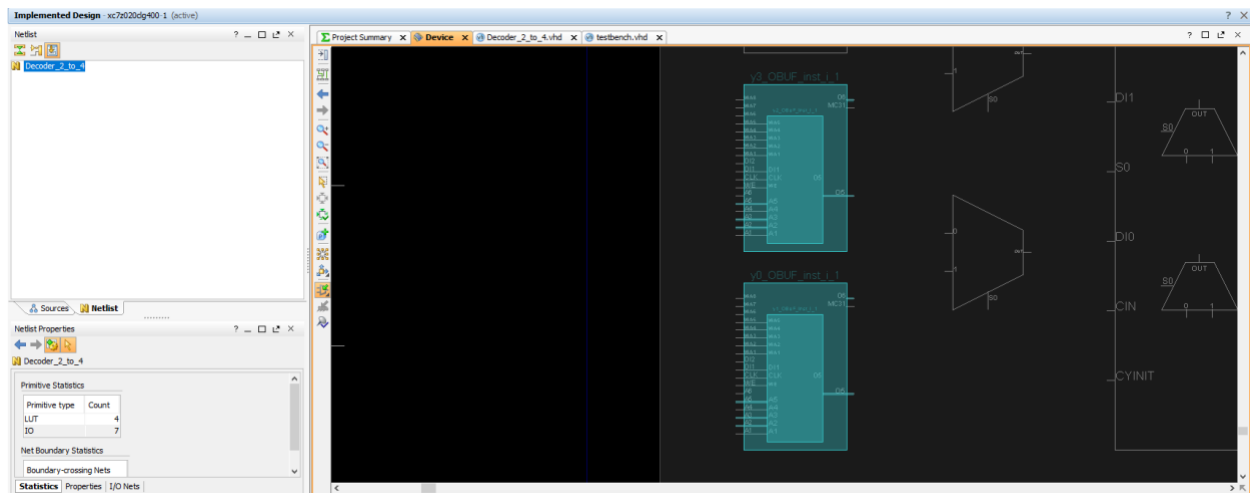
Figure 4 shows that the implemented circuit for the decoder works as expected per the truth table. When the enable bit is 1 and the input is “00” (0–2 ns),  $y_0 = 1$ , when the input is changed to “01” (2–4 ns),  $y_1 = 1$ , when the input is changed to “10” (4–6 ns),  $y_2 = 1$ , and when the input is changed to “11” (6–8 ns),  $y_3 = 1$ . Once the enable bit is changed to 0 (10–20 ns), there is no longer an output for  $y$ .



**Figure 5: RTL analysis schematic for 2-to-4 decoder**



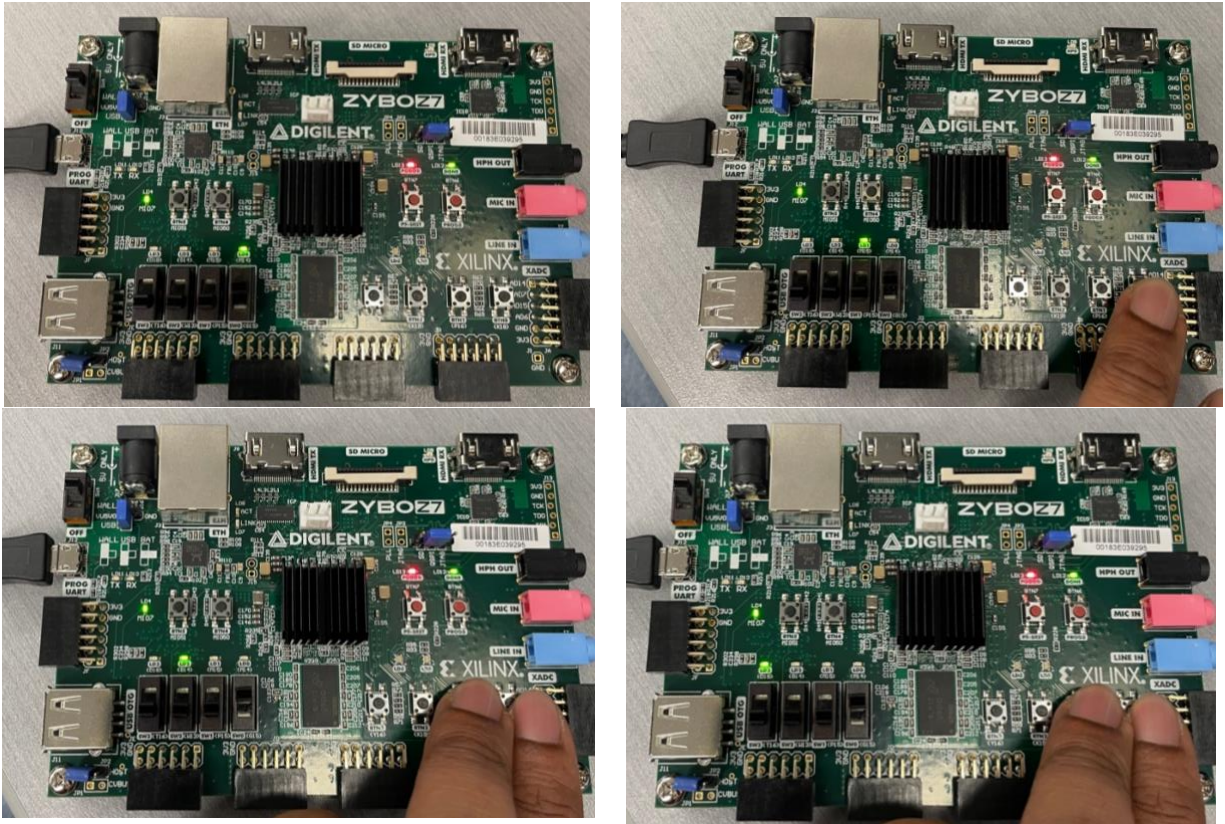
**Figure 6: Synthesis schematic for top level of 2-to-4 decoder**



**Figure 7: Physical implementation of 2-to-4 decoder on FPGA board**

Figure 7 shows that there are 4 lookup tables and 7 I/O blocks in the design. Each LUT is responsible for each y output (y0-y3):

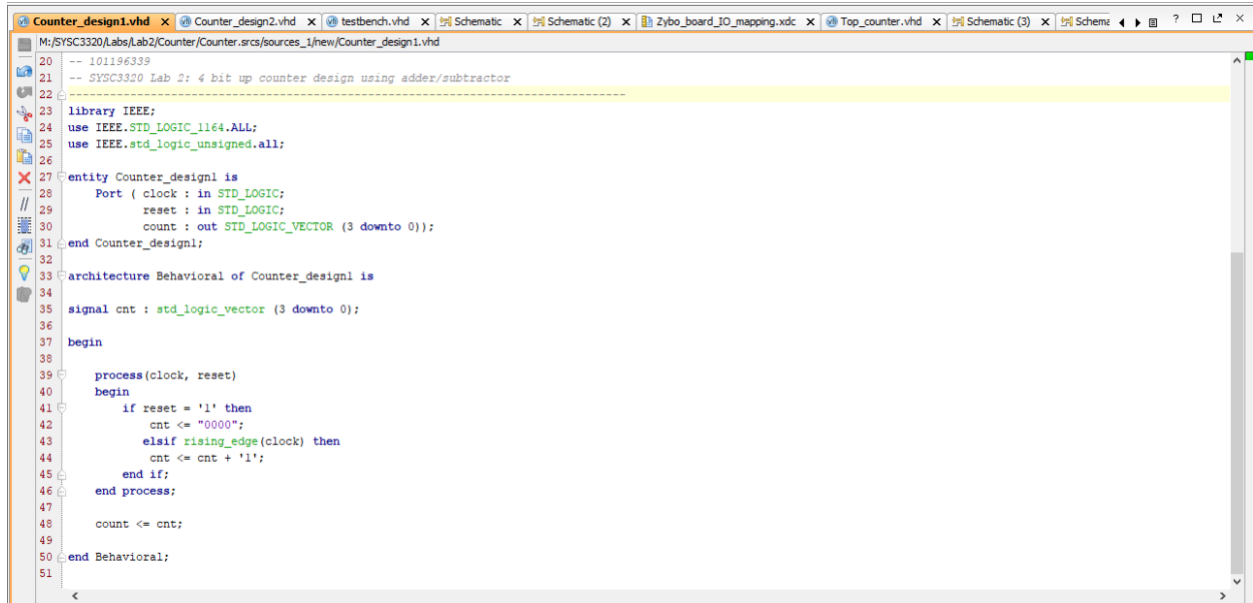
W0	W1	Y0	W0	W1	Y1	W0	W1	Y2	W0	W1	Y3
0	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	0	0	1	0
1	0	0	1	0	0	1	0	1	1	0	0
1	1	0	1	1	0	1	1	0	1	1	1



**Figure 8: Hardware showing LED output of each input combination for 2-to-4 decoder implemented on FPGA**

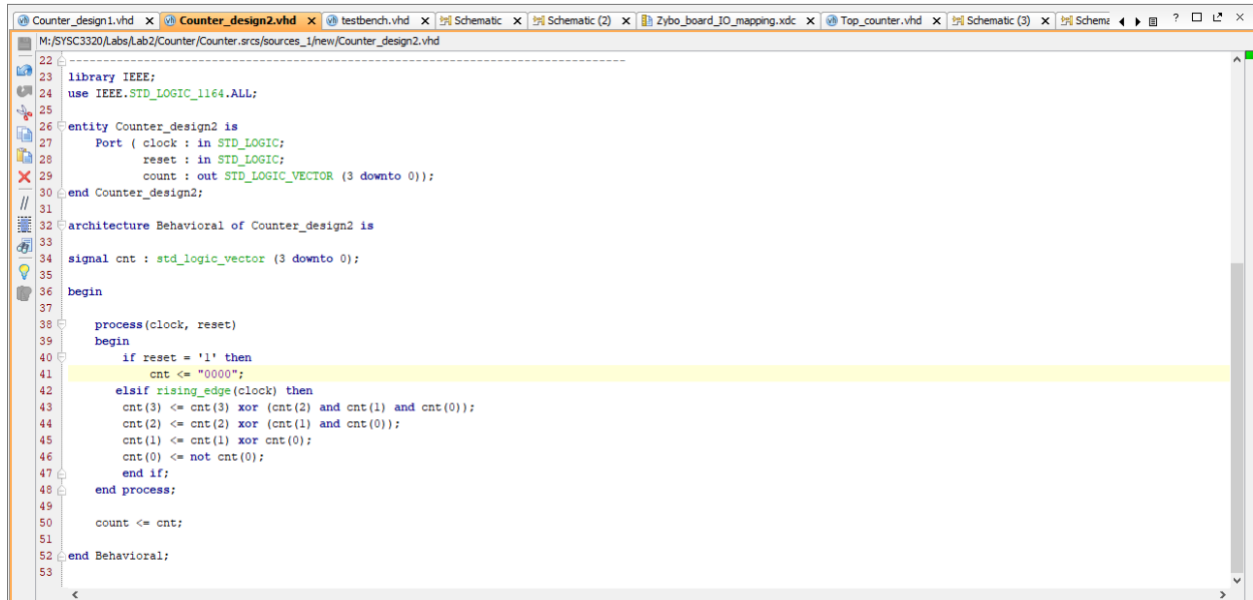
Like Figure 4, Figure 8 shows that the circuit behaves as expected per the truth table. When the enable switch is on and the input is “00” LED 0 is on, when the input is changed to “01” LED 1 is on, when the input is changed to “10” LED 2 is on, and when the input is changed to “11” LED 3 is on. Once the enable switch is off, none of the LEDs will turn on.

## 4-bit up-counter:

A screenshot of a VHDL code editor showing the implementation of a 4-bit up-counter using an adder/subtractor. The code is for Counter\_design1.vhd. It includes a library IEEE, uses IEEE.STD\_LOGIC\_1164.ALL and IEEE.std\_logic\_unsigned.all. The entity Counter\_design1 has ports for clock (STD\_LOGIC), reset (STD\_LOGIC), and count (STD\_LOGIC\_VECTOR (3 downto 0)). The architecture Behavioral of Counter\_design1 is shown, with a signal cnt of type std\_logic\_vector (3 downto 0). The process block for clock and reset shows a reset condition where cnt is set to "0000", and a rising edge condition where cnt is incremented by 1. The count output is assigned to cnt.

```
20 -- 101196339
21 -- SYSC3320 Lab 2: 4 bit up counter design using adder/subtractor
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25 use IEEE.std_logic_unsigned.all;
26
27 entity Counter_design1 is
28     Port ( clock : in STD_LOGIC;
29           reset : in STD_LOGIC;
30           count : out STD_LOGIC_VECTOR (3 downto 0));
31 end Counter_design1;
32
33 architecture Behavioral of Counter_design1 is
34     signal cnt : std_logic_vector (3 downto 0);
35
36 begin
37
38     process(clock, reset)
39     begin
40         if reset = '1' then
41             cnt <= "0000";
42         elsif rising_edge(clock) then
43             cnt <= cnt + '1';
44         end if;
45     end process;
46
47     count <= cnt;
48
49 end Behavioral;
50
51
```

Figure 9: VHDL code for counter design 1 (using adder/subtractor)

A screenshot of a VHDL code editor showing the implementation of a 4-bit up-counter using flip flops. The code is for Counter\_design2.vhd. It includes a library IEEE, uses IEEE.STD\_LOGIC\_1164.ALL. The entity Counter\_design2 has ports for clock (STD\_LOGIC), reset (STD\_LOGIC), and count (STD\_LOGIC\_VECTOR (3 downto 0)). The architecture Behavioral of Counter\_design2 is shown, with a signal cnt of type std\_logic\_vector (3 downto 0). The process block for clock and reset shows a reset condition where cnt is set to "0000", and a rising edge condition where cnt is updated using XOR operations to increment by 1. The count output is assigned to cnt.

```
22
23 library IEEE;
24 use IEEE.STD_LOGIC_1164.ALL;
25
26 entity Counter_design2 is
27     Port ( clock : in STD_LOGIC;
28           reset : in STD_LOGIC;
29           count : out STD_LOGIC_VECTOR (3 downto 0));
30 end Counter_design2;
31
32 architecture Behavioral of Counter_design2 is
33     signal cnt : std_logic_vector (3 downto 0);
34
35 begin
36
37     process(clock, reset)
38     begin
39         if reset = '1' then
40             cnt <= "0000";
41         elsif rising_edge(clock) then
42             cnt(3) <= cnt(3) xor (cnt(2) and cnt(1) and cnt(0));
43             cnt(2) <= cnt(2) xor (cnt(1) and cnt(0));
44             cnt(1) <= cnt(1) xor cnt(0);
45             cnt(0) <= not cnt(0);
46         end if;
47     end process;
48
49     count <= cnt;
50
51 end Behavioral;
52
53
```

Figure 10: VHDL code for counter design 2 (using flip flops)



```

21  -- SYS3320 Lab 2: testbench for 4 bit up counter design
22  -----
23  library IEEE;
24  use IEEE.STD_LOGIC_1164.ALL;
25
26  entity testbench is
27  -- Port ( );
28  end testbench;
29
30  architecture Behavioral of testbench is
31  signal Inclock: std_logic := '0';
32  signal Inreset: std_logic := '0';
33  signal Outcount1: std_logic_vector (3 downto 0);
34  signal Outcount2: std_logic_vector (3 downto 0);
35
36  component Counter_design1 is
37  Port ( clock : in STD_LOGIC;
38        reset : in STD_LOGIC;
39        count : out STD_LOGIC_VECTOR (3 downto 0));
40  end component;
41
42  component Counter_design2 is
43  Port ( clock : in STD_LOGIC;
44        reset : in STD_LOGIC;
45        count : out STD_LOGIC_VECTOR (3 downto 0));
46  end component;
47
48  begin
49  C1: Counter_design1 port map(clock => Inclock, reset => Inreset, count => Outcount1);
50  C2: Counter_design2 port map(clock => Inclock, reset => Inreset, count => Outcount2);
51
52  process begin
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71

```

**Figure 11: VHDL code for counter testbench**

```

Counter_design1.vhd x Counter_design2.vhd x testbench.vhd x Schematic x Schematic (2) x Zybo_board_IO_mapping.xdc x Top_counter.vhd * x Schematic (3) x Schen
M:/SYS3320/Labs/Lab2/Counter/Counter.srcs/sources_1/new/Top_counter.vhd
-- 101196339
-- SYS3320 Lab 2: Top level for 4 bit up counter design
-----
23
24
25 library IEEE;
26 use IEEE.STD_LOGIC_1164.ALL;
27
28 entity Top_counter is
29   Port ( btn : in STD_LOGIC_VECTOR (3 downto 0);
30         led : out STD_LOGIC_VECTOR (3 downto 0));
31 end Top_counter;
32
33 architecture Behavioral of Top_counter is
34   component Counter_design1 is
35     Port ( clock : in STD_LOGIC;
36           reset : in STD_LOGIC;
37           count : out STD_LOGIC_VECTOR (3 downto 0));
38   end component;
39
40   component Counter_design2 is
41     Port ( clock : in STD_LOGIC;
42           reset : in STD_LOGIC;
43           count : out STD_LOGIC_VECTOR (3 downto 0));
44   end component;
45
46 begin
47   C1: Counter_design1 port map(clock => btn(0), reset => btn(1), count => led);
48   C2: Counter_design2 port map(clock => btn(0), reset => btn(1), count => led);
49
50 end Behavioral;
51

```

Figure 12: VHDL code for top level of counter

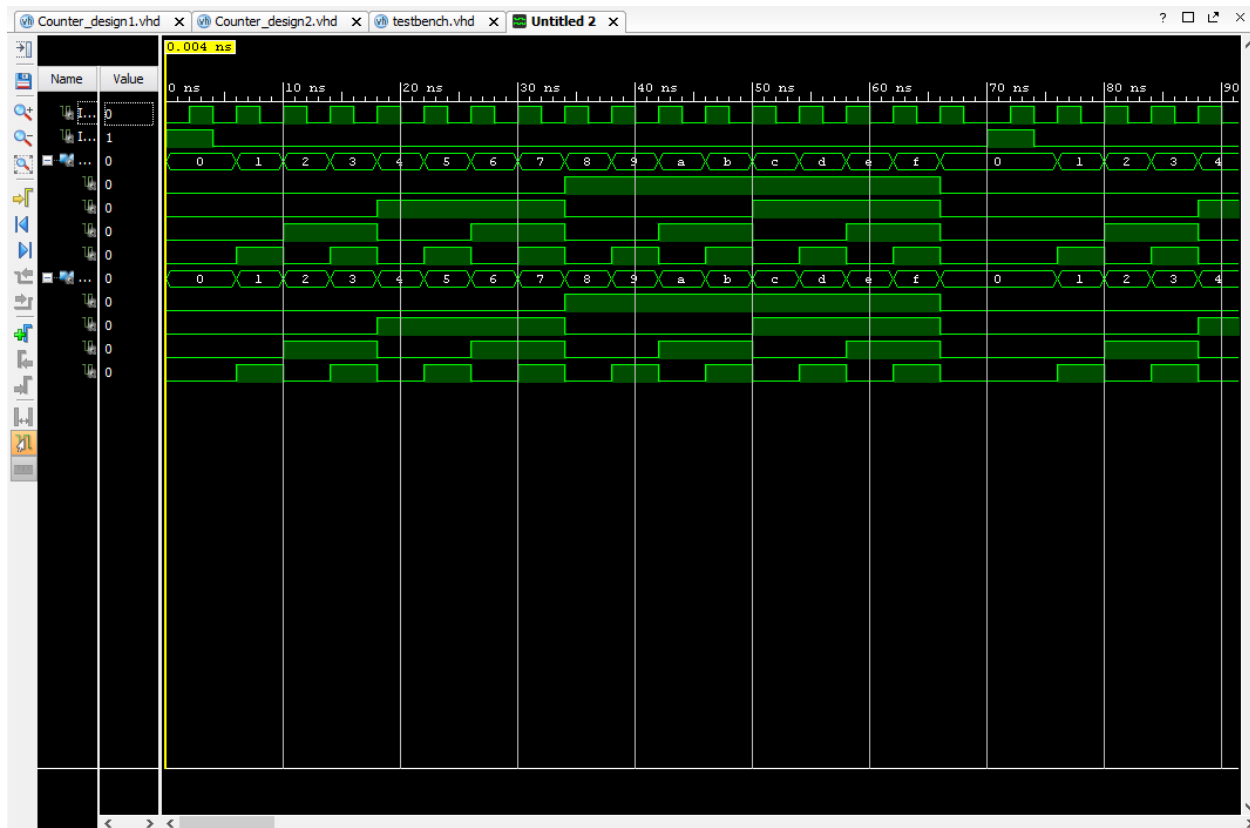
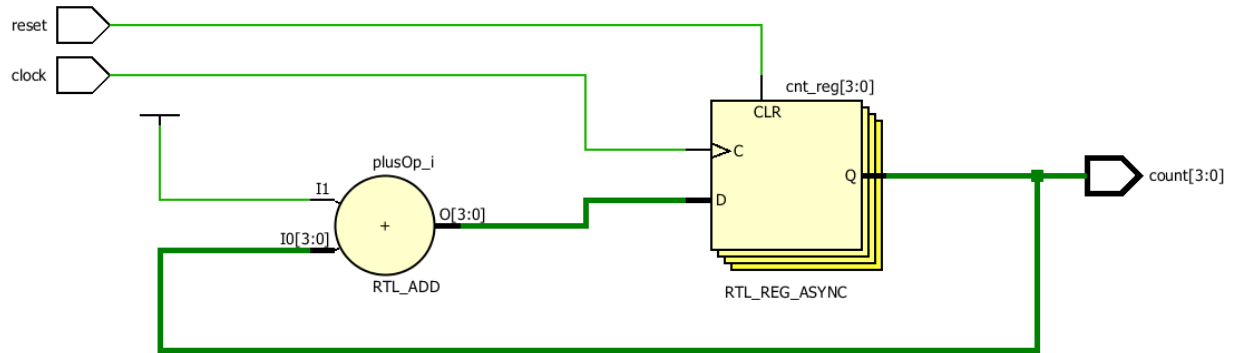
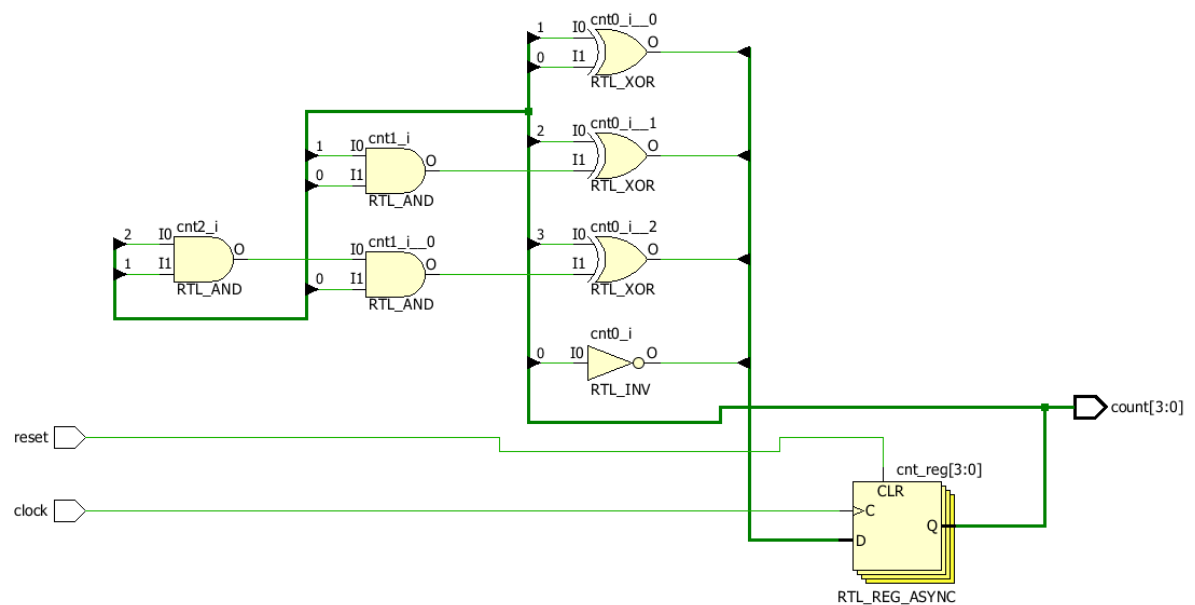


Figure 13: Simulation timing diagram for counter

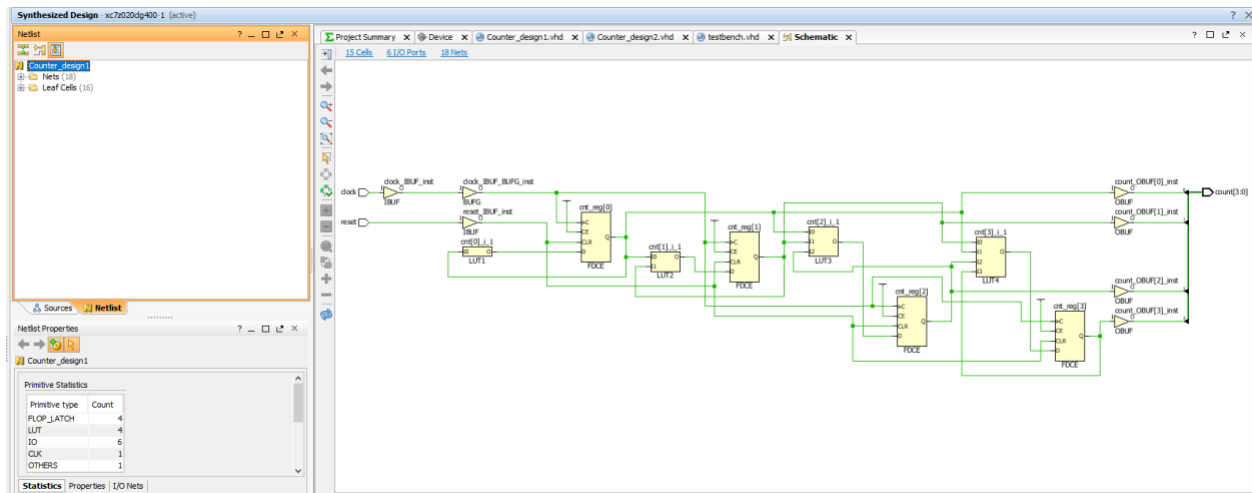
As shown in Figure 13, both designs for the counter implement the correct design. Every time the clock signal is on the rising edge, the count increments until it reaches 15 and resets.



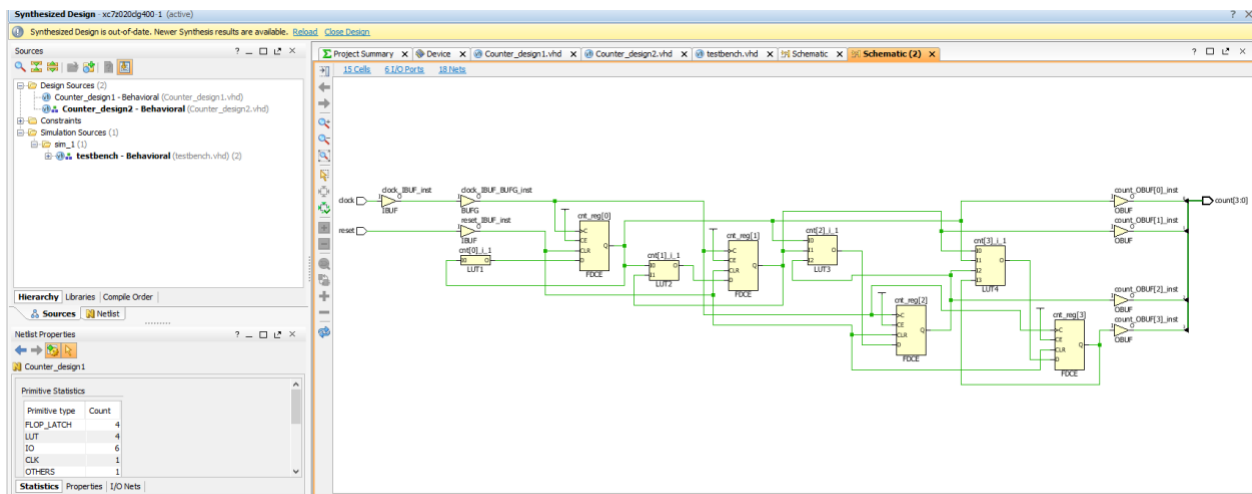
**Figure 14: RTL analysis schematic for counter design 1 (using adder/subtractor)**



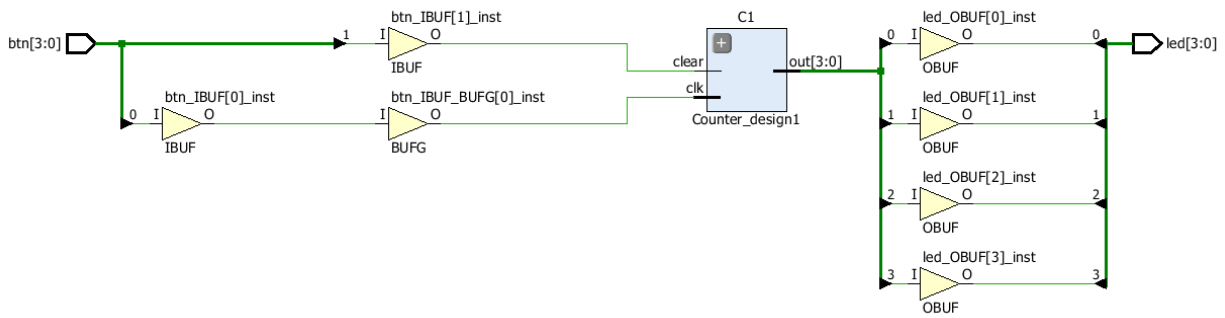
**Figure 15: RTL analysis schematic for counter design 2 (using flip flops)**



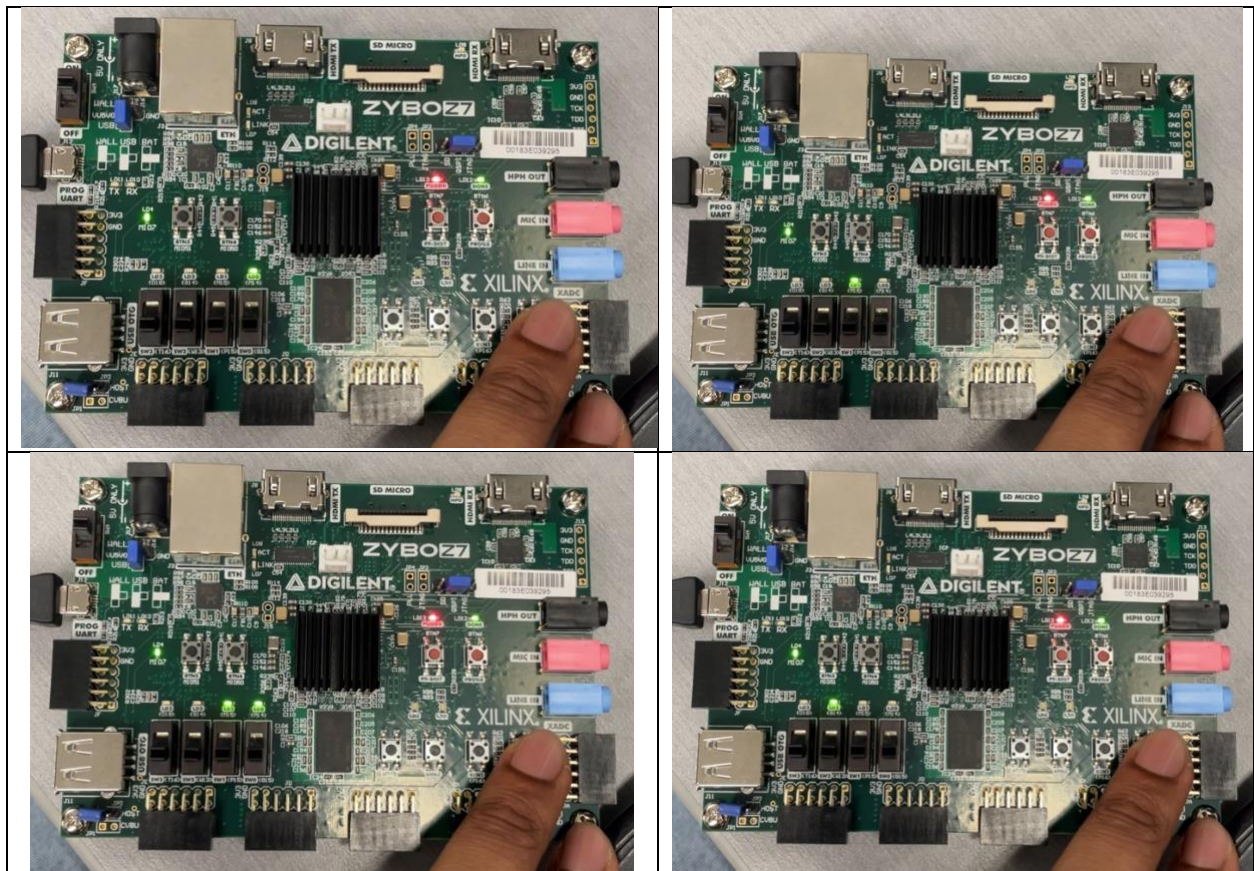
**Figure 16: Synthesized design schematic for counter design 1 (using adder/subtractor)**



**Figure 17: Synthesized design schematic for counter design 2 (using flip flops)**



**Figure 18: Example of synthesized design schematic for top level of counter design (design 1)**



**Figure 19: Hardware showing LED output for binary inputs 1 to 4 for physical implementation of counter on FPGA board**

Figure 19 shows that the circuit implementation on the board correctly displays the correct binary value when incrementing the counter from 1-4. When the counter is incremented using the button, the LEDs display “0001”, “0010”, “0011” and “0100”.