EIE 2810 Digital System Design Laboratory

Laboratory Report #6

Name: LI, Yutao

Student ID: 121090294

Lab Date: April 21st

The Chinese University of Hong Kong, Shenzhen

Content

Introduction

This laboratory consists of 2 experiments, they are:

- Experiment A: Using VHDL language to construct an 8-input AND Gate on the development board.
- Experiment B: Using VHDL language to construct a user-input down counter on the development board.

1. Experiment A

1.1 Results

1.1.1 Creating a project

As shown in Figure 1, to create a file, first turn on the Vivado software and click “Create Project”.

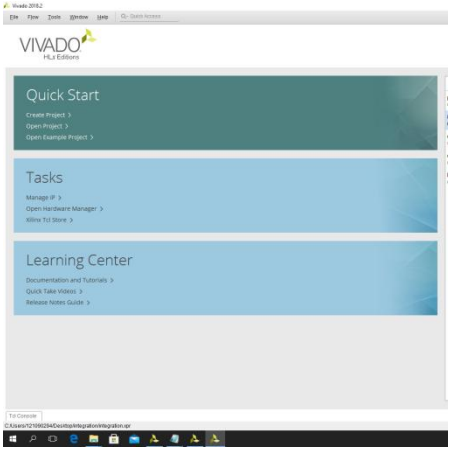

Figure 1 Start Creating a Project

As shown in Figure 2, Figure 3, and Figure 4, click “Next”, name the project, and choose the project type.

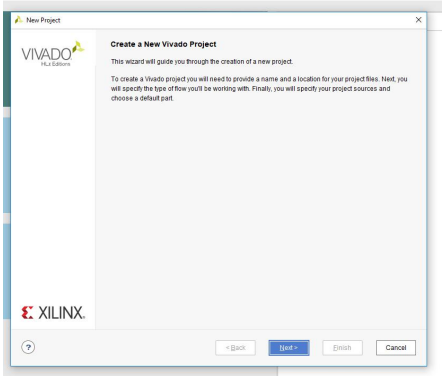

Figure 2 Creating a Project

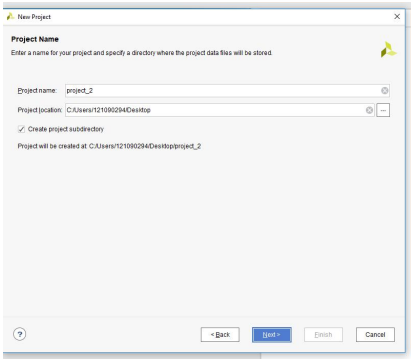

Figure 3 Naming the Project

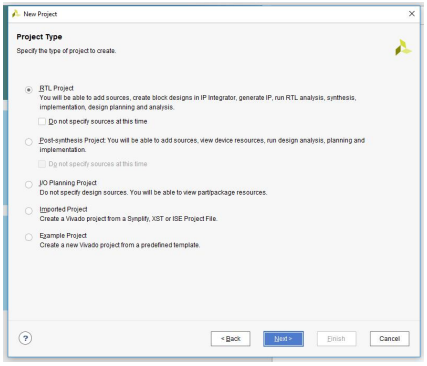

Figure 4 Choosing the Project Type

As shown in Figure 5, choose VHDL language and add a source file to the project.

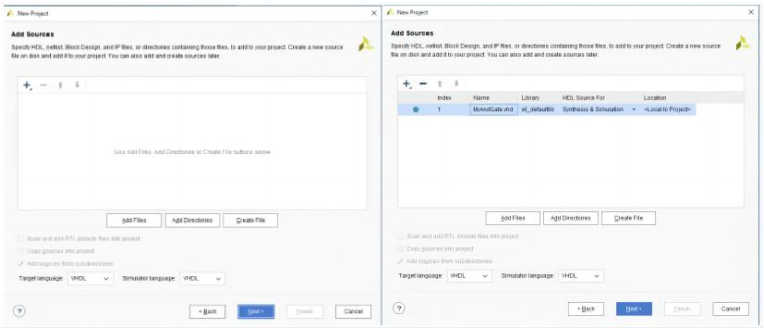

Figure 5 Choosing the Language and Creating Source File

As shown in Figure 6 and Figure 7, neglect adding constraints and choose the target chip. For this lab, the xc7a35tcsg324-1 chip was chosen and utilized.

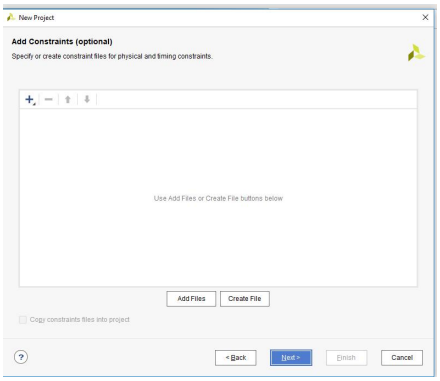

Figure 6 Neglecting Adding Constraints

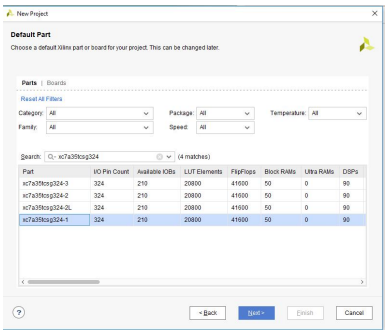

Figure 7 Selecting the Proper Chip for the Experiment

As shown in Figure 8, finish the process of creating the project.

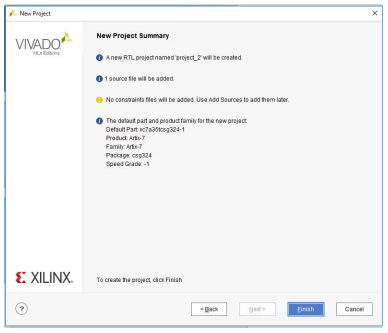

Figure 8 End of Creating the Project

1.1.2 Coding and Combining the Codes to the Development Board

As shown in Figure 9, before we dive into the coding part, let Vivado create an entity for us.

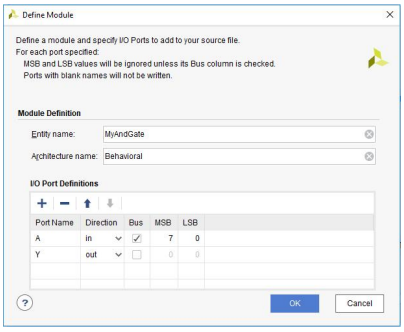

Figure 8 Creating Entity

As shown in Figure 10, double-click the source file in the Design Source to open the coding box. The coding box is shown in Figure 11.

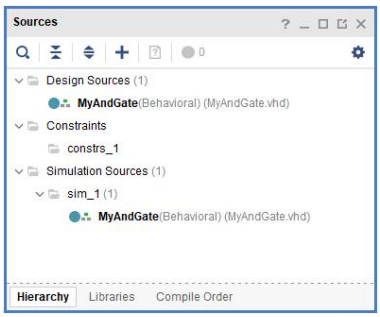

Figure 10 The Source Box

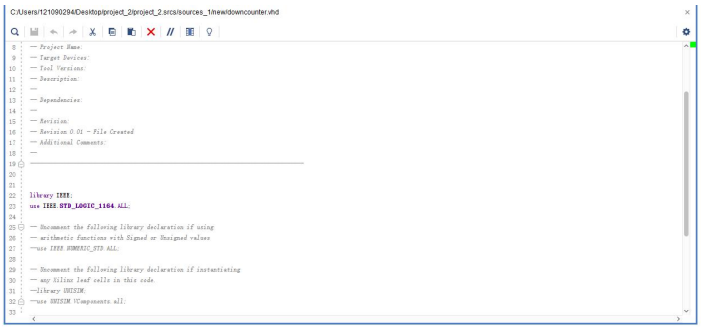

Figure 11 The Coding Box

Edit the architecture of the entity. To realize the 8-input AND Gate, assign the 8 elements in the input to the output with the “and” relationship. The editing process is shown in Figure 12. The detailed codes will be provided in the appendix.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity MyAndGate is
    Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
           Y : out STD_LOGIC);
end MyAndGate;

architecture Behavioral of MyAndGate is

begin

    Y <= A(7) and A(6) and A(5) and A(4) and A(3) and A(2) and A(1) and A(0);

end Behavioral;
```

Figure 12 Editing the Architecture of the Entity

As shown in Figure 13, run synthesis for the code.

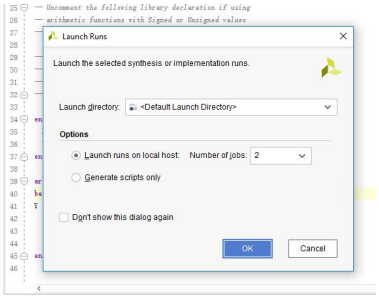

Figure 13 Running Synthesis for the Code

As shown in Figure 14, run implementation after running synthesis.

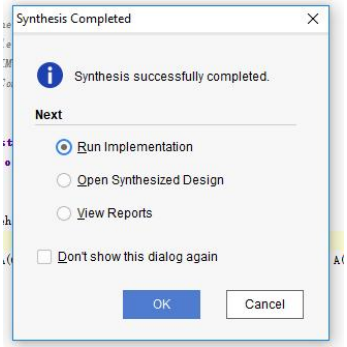

Figure 13 Running Implementation for the Code

After finishing running the implementation, we need to arrange the I/O Pins for the entities. As shown in Figure 14 and Figure 15, to arrange I/O Pins, choose the open implemented design, choose Layout in the toolbar, and choose I/O Planning.

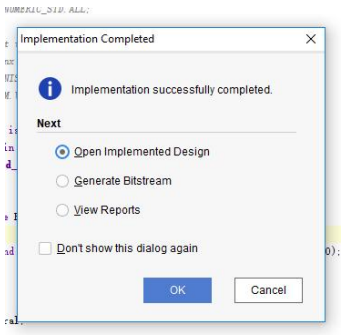

Figure 14 Choosing Open Implemented Design

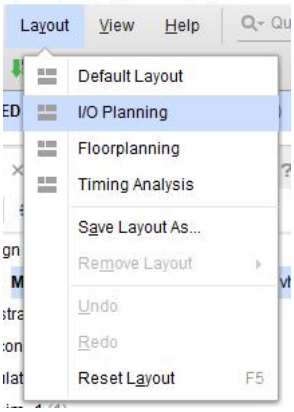

Figure 15 Choosing I/O Planning to Arrange the I/O Pins

After opening the I/O Ports window, modify the I/O Std, and arrange all the input entity elements and output entity elements to the package pins on the development board. To realize the 8-input AND Gate, put the 8 inputs into 8 switches and the output to one of the LEDs on the development board. The pin arrangement operation is shown in Figure 16.

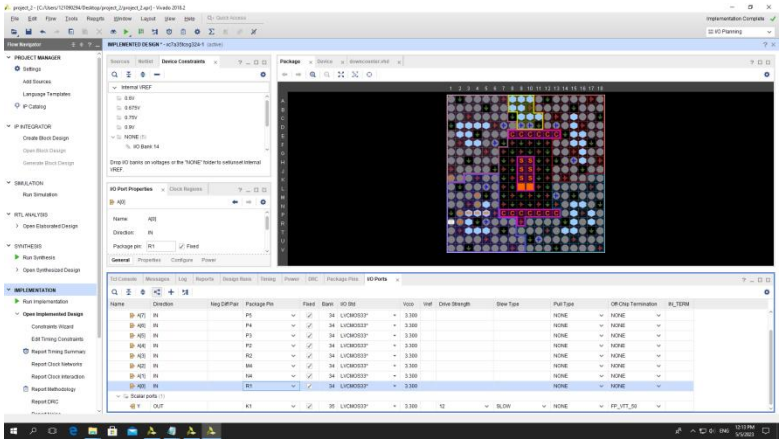

Figure 15 Choosing I/O Planning to Arrange the I/O Pins

After arranging the I/O pins, save the restriction and rerun the synthesis process and

implementation process. The saving procedure is shown in Figure 15.

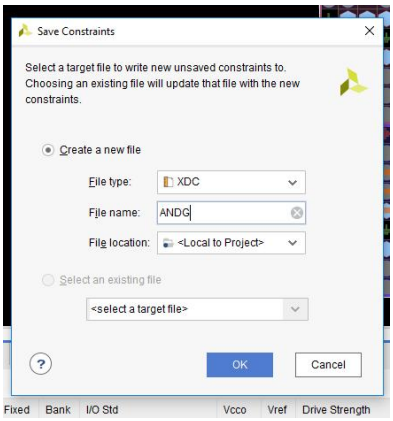

Figure 15 Saving Restrictions

As shown in Figure 16, after implementation rerunning is done, let Vivado generate a bitstream.

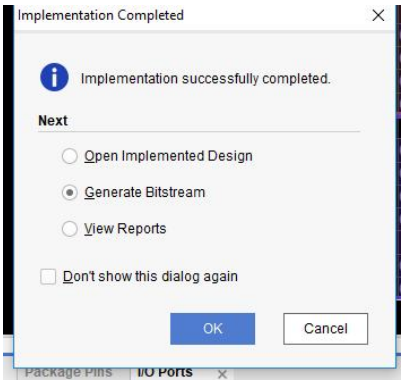

Figure 16 Generating Bitstream

After bitstream generation is done, we can input the codes to the development board. To input the codes to the board, first, choose “Open Hardware Manager”. Then click “Open Target” and click “Auto Connect”. Lastly, click “Program Device” and “Program”. The mentioned inputting process is shown in Figure 17, Figure 18, and Figure 19.

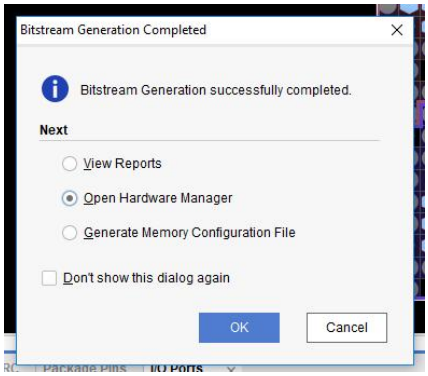

Figure 17 Opening Hardware Manager

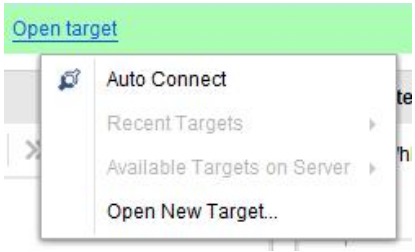

Figure 18 Connecting the Computer and the Development Board

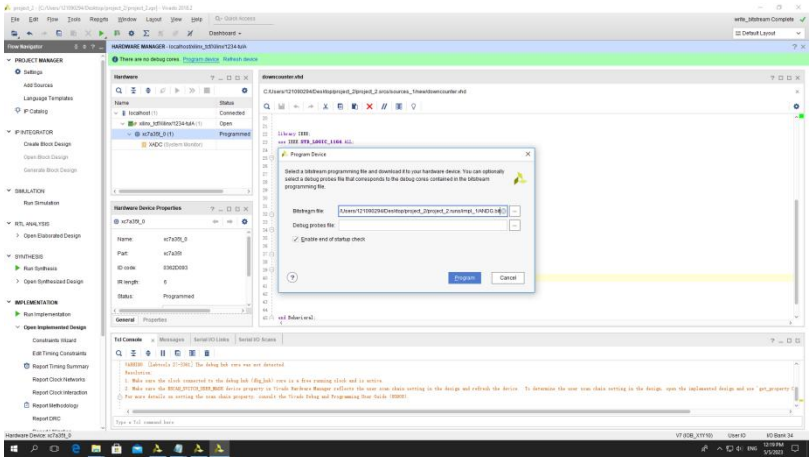

Figure 19 Inputting the Codes to the Development Board

1.1.3 Phenomena Shown on the Development Board

The phenomena shown on the development board are shown in Figure 20.

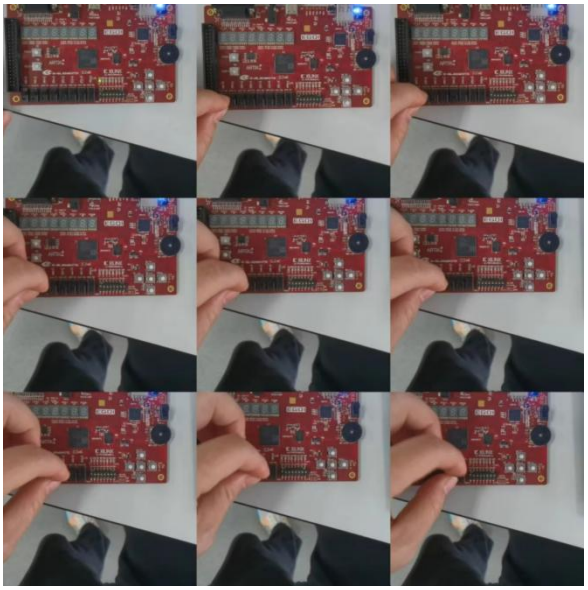

Figure 20 The Luminous Condition With Different Inputs

As shown in Figure 20, when we turn on the switches one by one, only when 8 switches are turned on will the LED be lightened, which goes well with the 8-input AND Gate logic. This means we successfully coded an 8-input AND Gate and combined it to a development board.

1.2 Questions

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FullAdder is
port( A: in std_logic;
      B: in std_logic;
      Cin: in std_logic;
      S: out std_logic;
      C: out std_logic);
end entity;

architecture adding of FullAdder is
begin
S <= (A xor B) xor Cin;
C <= (A and (B or Cin)) or (Cin and B);
end adding;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity 4BitAdder is
port( BCD1: in std_logic_vector(3 downto 0);
      BCD2: in std_logic_vector(3 downto 0);
      BCD3: out std_logic_vector(3 downto 0);
      carry: out std_logic);
end entity;

architecture Add of 4BitAdder is

component FullAdder is
port( A: in std_logic;
      B: in std_logic;
      Cin: in std_logic;
      S: out std_logic;
      C: out std_logic);
end component;

signal carry1: std_logic:='0';
signal carry2: std_logic:='0';
signal carry3: std_logic:='0';
signal carry4: std_logic:='0';
signal carry5: std_logic:='0';
begin
```

```
Adding0: FullAdder port map (BCD1(0), BCD2(0), carry1, BCD3(0), carry2);
Adding1: FullAdder port map (BCD1(1), BCD2(1), carry2, BCD3(1), carry3);
Adding2: FullAdder port map (BCD1(2), BCD2(2), carry3, BCD3(2), carry4);
Adding3: FullAdder port map (BCD1(3), BCD2(3), carry4, BCD3(3), carry5);
carry <= carry5;

end Add;
```

2. Experiment B

2.1 Design

After creating a new file just as what we've done in Experiment A, Experiment B is then designed to be done in 3 steps.

2.1.1 Clock Division

The first step is clock division. We need to work out 1Hz, 2Hz, 4Hz, and 8Hz clocks respectively, and connect them to 4 LEDs on the development board correspondingly. The development board provides a 100MHz clock with I/O Pin P17. To utilize this clock to provide the target clocks, a counter element was used. Let the counter increase with the rising edge of the 100MHz clock, every time there is a rising edge, the counter increases by 1. When this counter element reaches 6250000, let the 8Hz clock signal flip itself once. Similarly, we can derive a 4Hz signal by flipping the signal once every time there is a rising edge of the 8Hz signal, we can derive a 2Hz signal by flipping the signal once every time there is a rising edge of the 4Hz signal, and we can derive 1Hz signal by flip the signal once every time there is a rising edge of the 2Hz signal. The detailed codes will be provided in the appendix.

2.1.2 Combining the 7-segment Display, Switches, And LEDs

The second step is combining the users' inputs (8 switches) to the 7-segment display and LEDs. We first connect the switches to the LEDs and a button that controls loading. When we press the button, the development board should load what we input in the switches to the corresponding LEDs and light them up. Then add the 7-segment display to this system. According to the Ego_UserManual_v2.2 file, the luminous condition for the first and second 7-segment display is controlled by the same source input but two separate switches. To make these two 7-display work at the same time, a special clock is generated to make two switches close and open in turn. To make the ten not be lightened when the ten equals 0000_2, a when condition is added to control the switch in this special case. Finally, assign the users' inputs (8 switches) to the first two of the 7-segment displays with the special clock and the load button. When we press the load button, the development board should load what we input to the corresponding LEDs and light them up, the development board will also translate the BCD code that the user inputs to decimal numbers and show the decimal number on the first two 7-segment displays. The detailed codes will be provided in the appendix.

2.1.3 Add the Count Down Logic to the System

The third step is to add the count-down logic to the system. To realize this, the next state conditions are set to the users' inputs. The next state map is shown in Table 1 and Table 2. Table 1 shows the next state relationship of the unit while Table 2 shows the next state relationship of the ten.

Table 1

Q^n				Q^{n+1} (The ten is not 0_{10})				Q^{n+1} (The ten is 0_{10})			
1	0	0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	1	1	1	0	1	1	1
0	1	1	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1	0	0	0	0

Table 2

Q^n				Q^{n+1}			
1	0	0	1	1	0	0	0
1	0	0	0	0	1	1	1
0	1	1	1	0	1	1	0
0	1	1	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0

Let the unit change with the rising edge of the 1Hz clock. If the ten is not 0_{10}, when the unit goes to 0_{10}, the ten will decrease by 1 and reset the unit to 9_{10}. If the ten is 0_{10}, when the unit goes to 0_{10}, the unit will remain. After setting the mentioned next state logic to the development board, the down counter is worked out. The detailed codes will be provided in the appendix.

2.2 Results

The phenomena that appeared on the development board are shown in Figure 21 and Figure 22.

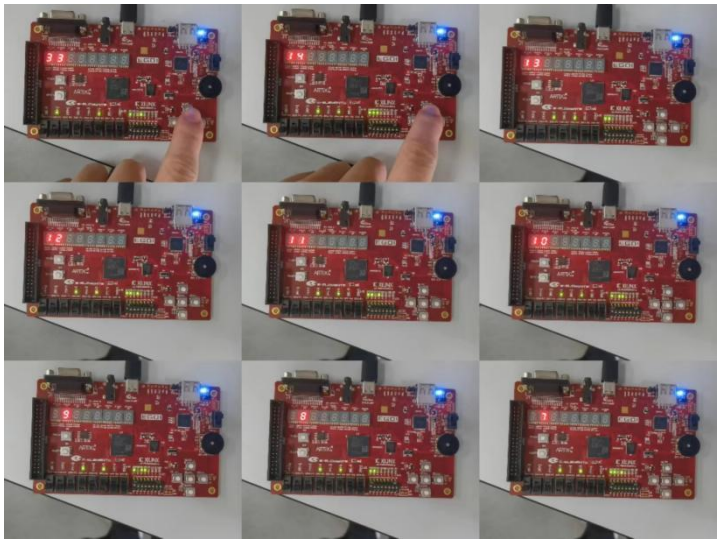

Figure 21 The Down Counter Operating

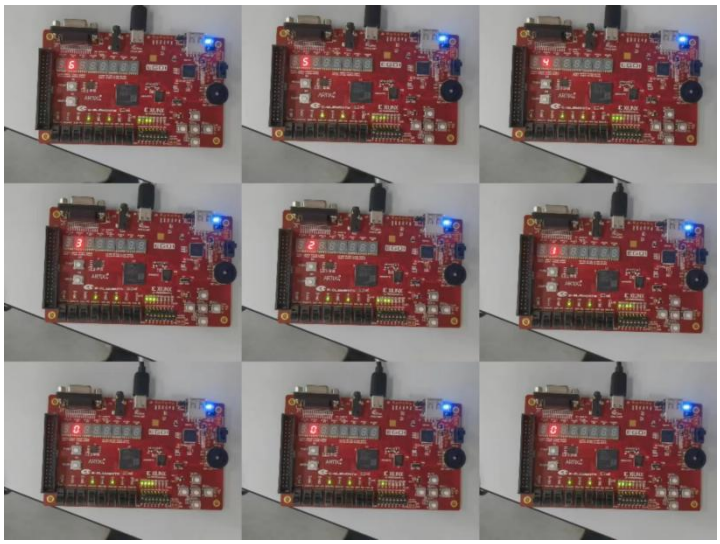

Figure 22 The Down Counter Operating

As shown in Figure 21, the development board will not load the users' input until we press the load button. As shown in Figure 21 and Figure 22, after we load the users' input into the system and release the load button, the 7-segment displays will begin counting down. As shown in Figure 22, when the number falls to 9, there will not exist a display in the ten. As shown in Figure 22, the display will stay at 0 after the whole number counts down to 0_{10}. All of these phenomena go well with the design, which means we successfully coded a down counter and loaded it into the development board.

3. Conclusion

In this lab, we used VHDL language to construct an 8-input AND Gate and a down counter, then we loaded these codes to a development board to verify the function. From the lab, we know:

1) The basic syntax and structure of VHDL language.
2) The way to connect a development board to the computer using Vivado.
3) How to realize a project step by step.

4. Appendix

4.1 The Codes for the 8-Input AND Gate

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MyAndGate is
    Port( A: in STD_LOGIC_VECTOR(7 downto 0);
          Y: out STD_LOGIC);
end MyAndGate;

architecture Behavioral of MyAndGate is

begin

    Y <= A(7) and A(6) and A(5) and A(4) and A(3) and A(2) and A(1) and A(0);

end Behavioral;
```

4.2 The Codes for the Down Counter

```
-- Company:
-- Engineer:
--
-- Create Date: 04/21/2023 09:55:08 AM
-- Design Name:
-- Module Name: counter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity divider is
port (ini_clk, rst: in std_logic;
      l_b: in std_logic;    --load button
      ManualInp: in std_logic_vector(7 downto 0);
      LEDs: out std_logic_vector(7 downto 0);
      SevenSeg: out std_logic_vector(6 downto 0);
      new_clk: out std_logic_vector(3 downto 0);
      t_u: out std_logic_vector(1 downto 0));    --ten or unit
end divider;

architecture Dividing of divider is

signal divide1: std_logic:='0';
signal divide2: std_logic:='0';
signal divide4: std_logic:='0';
signal divide8: std_logic:='0';
signal count: integer:=1;    --used in 8Hz
signal count1: integer:=1;    --used in special clock
signal switch: std_logic_vector(7 downto 0):="00000000";
signal tens: std_logic_vector(3 downto 0):="0000";
signal ones: std_logic_vector(3 downto 0):="0000";
signal tens_apply: std_logic_vector(3 downto 0):="0000";
signal ones_apply: std_logic_vector(3 downto 0):="0000";
signal spc_clk: std_logic:='0';
signal seg_ten: std_logic_vector(6 downto 0):="0000000";
signal seg_one: std_logic_vector(6 downto 0):="0000000";

begin

process (ini_clk, rst)
begin
if (rst='1') then
      count <= 1;
elsif (ini_clk'event and ini_clk ='1') then
      count <= count+1;
      if (count = 6250000) then
                  divide8 <= NOT divide8;
            count <= 1;
      end if;
end if;
end process;

process (divide8)
begin
```

```
if (divide8'event and divide8 ='1') then
    divide4 <= NOT divide4;
end if;
end process;

process(divide4)
begin
if (divide4'event and divide4 ='1') then
    divide2 <= NOT divide2;
end if;
end process;

process(divide2)
begin
if (divide2'event and divide2 ='1') then
    divide1 <= NOT divide1;
end if;
end process;

new_clk(3) <= divide8;
new_clk(2) <= divide4;
new_clk(1) <= divide2;
new_clk(0) <= divide1;

--clock division finish

--assign the user input to the system

process(switch)
begin
if (rst='1') then
    switch <="00000000";
elsif (ini_clk'event and ini_clk='1') then
    --separate the pressed condition and not pressed condition
    if (l_b='1') then
        switch <= ManualInp;
    else
        switch <= switch
    end if;
end if;
end process;

--special conditions of the BCD
```

```
process(switch)
begin
if (switch(7 downto 4) > "1001") then
      tens <= "1001";
else
      tens <= switch(7 downto 4);
end if;
end process;

process(switch)
begin
if (switch(3 downto 0) > "1001") then
      ones <= "1001";
else
      ones <= switch(3 downto 0);
end if;
end process;

--assign the signal to the LEDs

LEDs(0) <= tens(0);
LEDs(1) <= tens(1);
LEDs(2) <= tens(2);
LEDs(3) <= tens(3);
LEDs(4) <= ones(0);
LEDs(5) <= ones(1);
LEDs(6) <= ones(2);
LEDs(7) <= ones(3);


--generate a special clock for the ten and the unit

process (ini_clk, rst)
begin
if (rst='1') then
      count1 <= 1;
elsif (ini_clk'event and ini_clk ='1') then
      count1 <= count1+1;
      if (count1 = 250000) then
            spc_clk <= NOT spc_clk;
            count1 <= 1;
      end if;
end if;
end process;
```

```
t_u <= "10" when (spc_clk='1') else "01";

--translate BCD to 7-segment

process(tens_apply)
begin
case tens_apply is
when "0000" => seg_ten <="0000000";    --do not let it appear 0 when the ten equals 0
when "0001" => seg_ten <="0110000";
when "0010" => seg_ten <="1101101";
when "0011" => seg_ten <="1111001";
when "0100" => seg_ten <="0110011";
when "0101" => seg_ten <="1011011";
when "0110" => seg_ten <="1011111";
when "0111" => seg_ten <="1110000";
when "1000" => seg_ten <="1111111";
when "1001" => seg_ten <="1111011";
end case;
end process;

process(ones_apply)
begin
case ones_apply is
when "0000" => seg_one <="1111110";
when "0001" => seg_one <="0110000";
when "0010" => seg_one <="1101101";
when "0011" => seg_one <="1111001";
when "0100" => seg_one <="0110011";
when "0101" => seg_one <="1011011";
when "0110" => seg_one <="1011111";
when "0111" => seg_one <="1110000";
when "1000" => seg_one <="1111111";
when "1001" => seg_one <="1111011";
end case;
end process;

--arrange the output case

SevenSeg <= seg_ten when (spc_clk='1') else seg_one;

--the count down logic

process(divide1, rst)
```

```
begin
if (rst = '1') then
    tens_apply <= "0000";
elsif (divide1'event and divide1 = '1') then
    if (l_b = '1') then
        tens_apply <= tens;
    elsif (tens_apply = "0000") then
        tens_apply <= "0000";
    elsif (ones_apply = "0000") then
        tens_apply <= tens_apply - "0001";
    else
        tens_apply <= tens_apply;
    end if;
end if;
end process;

process (divide1, rst)
begin
if (rst = '1') then
    ones_apply <= "0000";
elsif (divide1'event and divide1 = '1') then
    if (l_b = '1') then
        ones_apply <= ones;
    elsif (ones_apply = "0000") then
        if (tens_apply = "0000") then
            ones_apply <= "0000";
        else
            ones_apply <= "1001";
        end if;
    else
        ones_apply <= ones_apply - "0001";
    end if;
end if;
end process;

end Dividing;
```