Lab2 Decimal Counter and LEDs Display

Hong-Jia Yang
Southern University of Science and Technology

Abstract—The correct realization of digital systems depends on the accurate modeling and analysis of circuit structure. This report summaries the process to design a decimal counter using sequential logic circuit. Based on two segments structure method, the design is completed accurately and efficiently. The detail simulation process and main problem in results will be discussed together to analyse the systems from different aspects. Some detailed circuit principle in digital systems will also be explained.

Index Terms—sequential logic circuit, two segments structure, VHDL.

1 Introduction

DIGITAL counter is widely used in digital system. It is not only an independent integrated circuit manufacturing, but also a part of large integrated circuit. The digital counter can not only be used to count the clock pulse, but also be used for frequency division, timing, generating pulse and pulse sequence and digital operation. The instruction address is counted in the controller of the electronic computer, so as to take out the next instruction in sequence, and counter can be used to record the times of addition and subtraction when multiplying and dividing in the arithmetic unit.

Counting is one of the simplest basic operations. The digital counter is composed of basic counting unit and some control gates. And the counting unit contains a series of various triggers with the function of storing information. These triggers include RS trigger, T trigger, D trigger and JK Trigger.

A decimal counter is like a stopwatch. Take a counter from 0 to 1000 as an example. The number of bit in unit increases from 0 to 9. When the bit position reaches 9, it increases the tenth bit by 1 and sets the value of unit bit to 0. Similarly, when all the ten bit and unit bit are 9, the hundred position is 1, and the ten bit and unit bit are cleared. This cycle produces a counting effect.

Detailed code description, module block diagram, simulation circuit diagram and waveform diagram, as well as the problems I encountered in the experiment will be discussed carefully.

In this experiment, we will **continue to learn the basic operation of VIVADO software, and mainly learn to design a sequential logic circuit using a two segments structure, that is, a synchronous section or a synchronous section with asynchronous inputs combined with a combinational section. Also, the structure method will also be used in this lab to help design the system**.

We will simulate a three-digit decimal counter and implement the counter onto the Nexys4 DDR board. The decimal counter should have reset function through a push button and load function through switches. The counting time interval for the counter is 1s and the result should be displayed on 12 LEDs.

We will use Clock Wizard of IP core to generate a 10MHz clock from the 100 MHz crystal oscillator on the board. Then use this clock to generate a 1Hz clock and send to the decimal counter.

Fig. 1. Digital counter

2 Lab Result

2.1 Pre-lab Preparation: Design block diagram

My main method to achieve this design task is firstly generate a 1Hz clock through the IP core and another module called **'pre_counter'**. And this module can generate clock of any frequency smaller than 10MHz through change in the parameters. **Also, in the following simulation I still firstly design the module to generate 1Hz clock and then design the decimal counter.** The block diagram to generate 1s signal is:

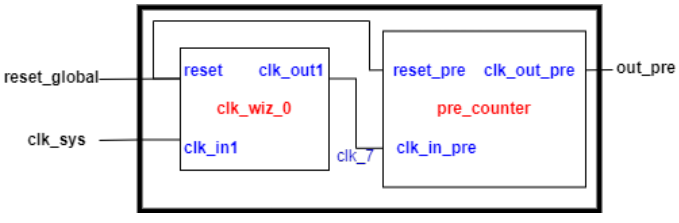


Fig. 2. Generate 1Hz signal

A reset function is connect to all two modules to restart the clock.

From this bock, the diagram for our total decimal counter is:

The 1Hz signal is transferred to decimal counter block and the load inputs are connect directly to the decimal counter block. Also,

*Thanks

- *This work was supported in Department of Electrical and Electronic Engineering in Southern University of Science and Technology with Professor Yajun Yu, and the tutorial assistant and all classmates.*

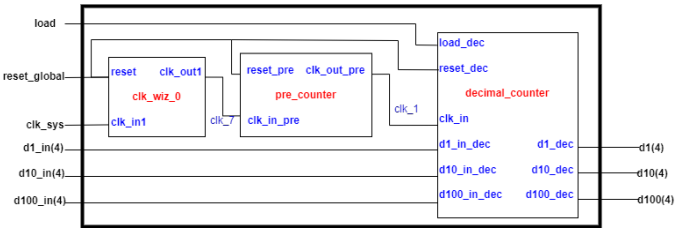


Fig. 3. Block diagram for decimal counter

the reset and load signal are connected to the decimal counter. Finally, the decimal counter output the counting number.

My design for the inter structure of the decimal counter is:

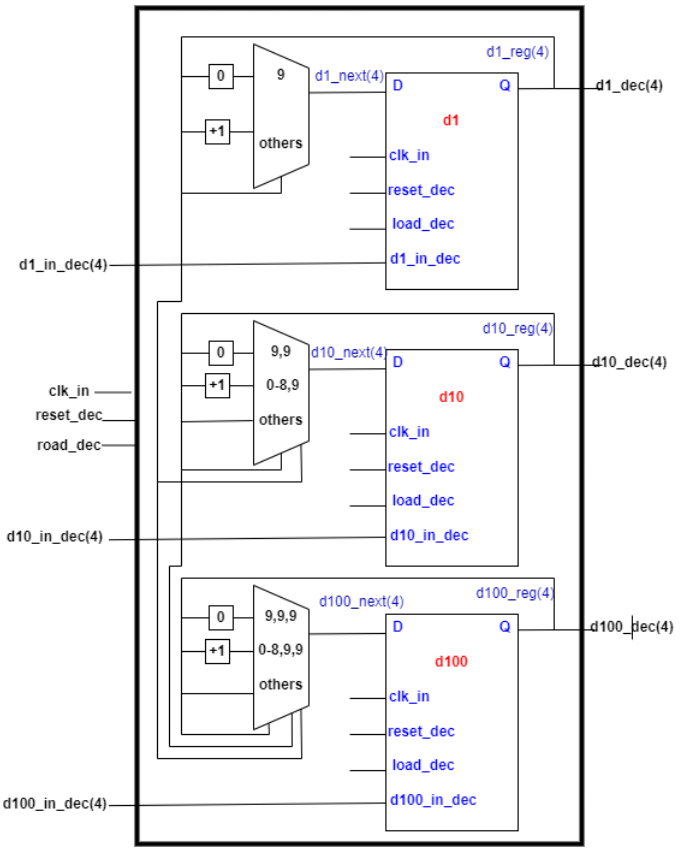


Fig. 4. Inner structure of decimal counter

Using the method of **two segments structure**, we can easily design the sequential logic circuit by creating a combinational module and a flip-flop module. The combinational part send the next value for the flip-flop according to the output of the flip-flops as the rule of decimal counter. And the flip-flop pass the next value to the output as the rising edge of the clock. **My design for the reset and load is that when load is high level, push the reset and load value will be pass to the flip-flops output and when load is low level, push the reset will clear the output**.

3 VHDL Code and Testbench

My code for the pre_counter is:

We use a 24 length vector to count the number, and set its goal value. When it achieve its goal value, output high level to the

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pre_counter is
  port (clk_in_pre, reset_pre: in std_logic;
        clk_out_pre: out std_logic);
end entity pre_counter;

architecture behavior of pre_counter is
 signal counter: std_logic_vector(23 downto 0);

begin
process (clk_in_pre, reset_pre) is
  begin
  if reset_pre = '1' then
    counter <= "000000000000000000000000";
  elsif clk_in_pre'event and clk_in_pre='1' then
      if counter="100110001001011010000000" then --1s
    --if counter="000000000000000001100100" then --10us
    --if counter="000000000000000001111101000" then --100us
        counter<="000000000000000000000000";
        clk_out_pre<='1';
      else
        counter<=counter+'1';
        clk_out_pre<='0';
      end if;
  end if;
end process;
end architecture behavior;
```

decimal_counter. Here we set several several goal value for this pre_counter to our following simulation.

The code for the decimal counter is:

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity decimal_counter is
    port (clk_in, reset_dec,load_dec: in std_logic;
      d1_in_dec,d10_in_dec,d100_in_dec: in std_logic_vector(3 downto 0);
      d1_dec,d10_dec,d100_dec: out std_logic_vector(3 downto 0));
end entity decimal_counter;

architecture concurrent_arch of decimal_counter is
signal d1_reg, d10_reg, d100_reg: std_logic_vector (3 downto 0);
signal d1_next, d10_next, d100_next: std_logic_vector (3 downto 0);

begin
-- register
process (clk_in,reset_dec,load_dec) is
  begin
  if (reset_dec = '1' and load_dec='1') then
    d1_reg <=d1_in_dec;
    d10_reg <=d10_in_dec;
    d100_reg <=d100_in_dec;
  elsif  (reset_dec='1' and load_dec = '0') then
    d1_reg <="0000";
    d10_reg <="0000";
    d100_reg <="0000";
  elsif clk_in'event and clk_in='1' then
    d1_reg <= d1_next;
    d10_reg <= d10_next;
    d100_reg <= d100_next;
  end if;
end process;
-- next-state logic
d1_next <= "0000" when d1_reg = 9 else
           d1_reg+1;
d10_next <= "0000" when (d1_reg = 9 and d10_reg = 9) else
            d10_reg+1 when d1_reg = 9 else
            d10_reg;
d100_next <= "0000" when (d1_reg=9 and d10_reg=9 and d100_reg=9) else
             d100_reg+1 when (d1_reg=9 and d10_reg=9) else
             d100_reg;
-- Output logic
d1_dec <= d1_reg; d10_dec <= d10_reg; d100_dec <= d100_reg;
end architecture concurrent_arch;
```

We write this code according to the **two segments structure** and our block diagram. We mark the two segments.

Finally, the code for the full structure and testbench are shown below:

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity clk_decimal_counter is
  port (clk_sys,reset_global,load: in std_logic;
        dl_in,dl0_in,dl00_in: in std_logic_vector(3 downto 0);
        dl,dl0,dl00: out std_logic_vector(3 downto 0));
end entity clk_decimal_counter;

architecture structure of clk_decimal_counter is

component pre_counter is
port (clk_in_pre, reset_pre: in std_logic;
clk_out_pre: out std_logic);
end component pre_counter;

component clk_wiz_0
port(clk_out1: out std_logic;reset: in std_logic;clk_in1: in std_logic);
end component;

component decimal_counter is
port (clk_in, reset_dec,load_dec: in std_logic;
dl_in_dec,dl0_in_dec,dl00_in_dec: in std_logic_vector(3 downto 0);
dl_dec,dl0_dec,dl00_dec: out std_logic_vector(3 downto 0));
end component decimal_counter;

signal clk_7,clk_1 : std_logic;

begin
  clkwiz : clk_wiz_0 port map(
         clk_in1 => clk_sys,reset => reset_global,clk_out1 => clk_7);
  precounter: pre_counter port map(
          clk_in_pre=>clk_7, reset_pre=>reset_global,
          clk_out_pre=>clk_1);
  decimalcounter: decimal_counter port map(
          clk_in=>clk_1,reset_dec=>reset_global,load_dec=>load,
          dl_in_dec=>dl_in,dl0_in_dec=>dl0_in,dl00_in_dec=>dl00_in,
          dl_dec=>dl,dl0_dec=>dl0,dl00_dec=>dl00);
end STRUCTURE;
```

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

entity tb_clk_decimal_counter is
end tb_clk_decimal_counter;

architecture behavior of tb_clk_decimal_counter is
component clk_decimal_counter
  port (clk_sys,reset_global,load: in std_logic;
        dl_in,dl0_in,dl00_in: in std_logic_vector(3 downto 0);
        dl,dl0,dl00: out std_logic_vector(3 downto 0));
end component;

signal clk_sys_tb,reset_global_tb,load_tb : std_logic:='0';
SIGNAL dl_in_tb, dl0_in_tb, dl00_in_tb : STD_LOGIC_VECTOR(3 DOWNTO 0) := X"0";
SIGNAL dl_tb, dl0_tb, dl00_tb : STD_LOGIC_VECTOR(3 DOWNTO 0) := X"0";

begin
   uut: clk_decimal_counter PORT MAP ( clk_sys=>clk_sys_tb,reset_global=>reset_global_tb,load=>load_tb,
                                       dl_in=>dl_in_tb,dl0_in=>dl0_in_tb,dl00_in=>dl00_in_tb,
                                       dl=>dl_tb,dl0=>dl0_tb,dl00=>dl00_tb);
   reset_global_tb<='0', '1' after 10000 ns,'0' after 12000 ns;
   dl_in_tb<= X"1";dl0_in_tb<= X"1";dl00_in_tb<= X"9";
   --load_tb<='1';
   clock_gen: process
   constant period : time := 10 ns;
   begin
     clk_sys_tb <= '0';
     wait for period/2;
     clk_sys_tb <= '1';
     wait for period/2;
   end process;

end behavior;
```

In the testbench **we produce a 100M clock by ourselves for in simulation we can not use the system clock**. And set the reset and load signal.

4 Simulation Result

4.1 Behavioral simulation

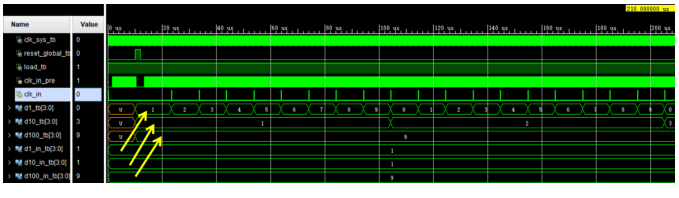


Fig. 5. Behavioral simulation result with 10us counting interval

We simulation the load mode for this can simplify test out our bug. from the behavioral simulation, the simulation is correct, the load numbers are loaded after the reset. And the three clock signal 100M, 10M and 100000Hz are all correct.

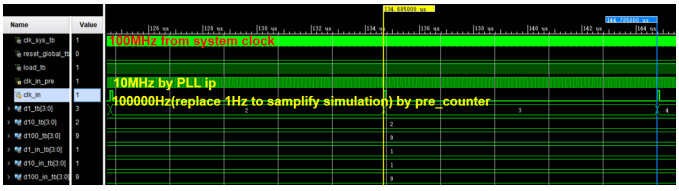


Fig. 6. Behavioral simulation result in detail

4.2 Post-Synthesis Simulation

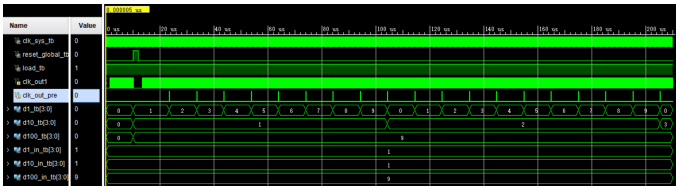


Fig. 7. Post-Synthesis Functional Simulation with 10us counting interval

The functional simulation is also in line with our design. **However, in the timing result we will see some problem**.

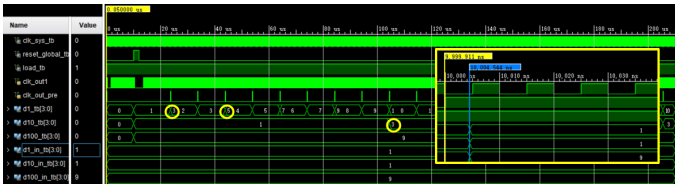


Fig. 8. Post-Synthesis Timing Simulation with 10us counting interval result 1

As we can see, **there are instantaneous change in d1 and d10 when the counting clock arrive. This is due to Competitive adventure in the circuit**. That is, when we add '1' in the combinational circuit, there may be more than one bit change for one output. For example, d1 change from '1' to '2', in binary code, it changes from '0001' to '0010', **if the bit in the third position comes quicker then the last position bit, it firstly changes to '0011', which is '3' and then changes to '0010'**. This is why we see a instantaneous change to '3' here.

To solve this problem, we can use Gray code, each time only one bit changes. For four bit Gray code, start from '0000', '0001', '0011', '0010', '0110', '0111', '0101', '0100', '1100', '1101', '1111', '1110', '1010', '1011', '1001' and finally '1000'.

However, these instantaneous change do not influence our actual result on the board for they are too short to be seen by our eyes.

When we do not load the number just reset to zero and count, in timing simulation we do not see the instantaneous change. This may because our load signal's relevant circuit influence the circuit for d1, d10 output delay time. More detail reason is hard to find so I just record this phenomenon.

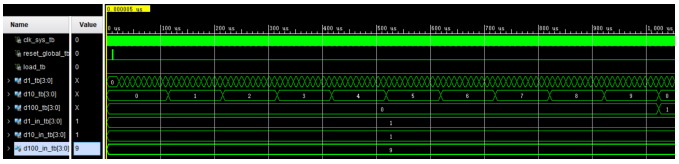
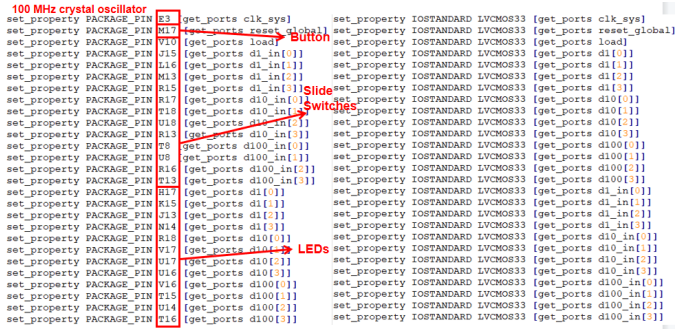


Fig. 9. Post-Synthesis Timing Simulation with 10us counting interval result 2

4.3 Post-Implementation Simulation

In the constraints file, we use the following setting: We set the system clock port to E3, which is the 100 MHz crystal oscillator. And set reset to a button port, load and load number to switches ports and result to LEDs ports.




```
set_property PACKAGE_PIN E3   [get_ports clk_sys]        set_property IOSTANDARD LVCMOS33 [get_ports clk_sys]
set_property PACKAGE_PIN M17  [get_ports reset_global]   set_property IOSTANDARD LVCMOS33 [get_ports reset_global]
set_property PACKAGE_PIN J15  [get_ports load]           set_property IOSTANDARD LVCMOS33 [get_ports load]
set_property PACKAGE_PIN L16  [get_ports d1_in[0]]       set_property IOSTANDARD LVCMOS33 [get_ports d1[0]]
set_property PACKAGE_PIN M13  [get_ports d1_in[1]]       set_property IOSTANDARD LVCMOS33 [get_ports d1[1]]
set_property PACKAGE_PIN R15  [get_ports d1_in[2]]       set_property IOSTANDARD LVCMOS33 [get_ports d1[2]]
set_property PACKAGE_PIN R17  [get_ports d1_in[3]]       set_property IOSTANDARD LVCMOS33 [get_ports d1[3]]
set_property PACKAGE_PIN T18  [get_ports d10_in[0]]      set_property IOSTANDARD LVCMOS33 [get_ports d10[0]]
set_property PACKAGE_PIN U18  [get_ports d10_in[1]]      set_property IOSTANDARD LVCMOS33 [get_ports d10[1]]
set_property PACKAGE_PIN R13  [get_ports d10_in[2]]      set_property IOSTANDARD LVCMOS33 [get_ports d10[2]]
set_property PACKAGE_PIN T8   [get_ports d10_in[3]]      set_property IOSTANDARD LVCMOS33 [get_ports d10[3]]
set_property PACKAGE_PIN U8   [get_ports d100_in[0]]     set_property IOSTANDARD LVCMOS33 [get_ports d100[0]]
set_property PACKAGE_PIN R16  [get_ports d100_in[1]]     set_property IOSTANDARD LVCMOS33 [get_ports d100[1]]
set_property PACKAGE_PIN T13  [get_ports d100_in[2]]     set_property IOSTANDARD LVCMOS33 [get_ports d100[2]]
set_property PACKAGE_PIN H17  [get_ports d100_in[3]]     set_property IOSTANDARD LVCMOS33 [get_ports d100[3]]
set_property PACKAGE_PIN K15  [get_ports d1[0]]          set_property IOSTANDARD LVCMOS33 [get_ports d1_in[0]]
set_property PACKAGE_PIN J13  [get_ports d1[1]]          set_property IOSTANDARD LVCMOS33 [get_ports d1_in[1]]
set_property PACKAGE_PIN N14  [get_ports d1[2]]          set_property IOSTANDARD LVCMOS33 [get_ports d1_in[2]]
set_property PACKAGE_PIN R18  [get_ports d1[3]]          set_property IOSTANDARD LVCMOS33 [get_ports d1_in[3]]
set_property PACKAGE_PIN V17  [get_ports d10[0]]         set_property IOSTANDARD LVCMOS33 [get_ports d10_in[0]]
set_property PACKAGE_PIN U17  [get_ports d10[1]]         set_property IOSTANDARD LVCMOS33 [get_ports d10_in[1]]
set_property PACKAGE_PIN U16  [get_ports d10[2]]         set_property IOSTANDARD LVCMOS33 [get_ports d10_in[2]]
set_property PACKAGE_PIN V16  [get_ports d10[3]]         set_property IOSTANDARD LVCMOS33 [get_ports d10_in[3]]
set_property PACKAGE_PIN T15  [get_ports d100[0]]        set_property IOSTANDARD LVCMOS33 [get_ports d100_in[0]]
set_property PACKAGE_PIN U14  [get_ports d100[1]]        set_property IOSTANDARD LVCMOS33 [get_ports d100_in[1]]
set_property PACKAGE_PIN T16  [get_ports d100[2]]        set_property IOSTANDARD LVCMOS33 [get_ports d100_in[2]]
set_property PACKAGE_PIN V15  [get_ports d100[3]]        set_property IOSTANDARD LVCMOS33 [get_ports d100_in[3]]
```

Fig. 10. Constraints file

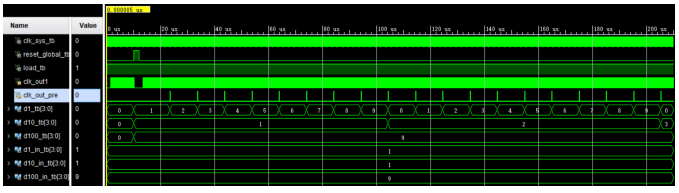


Fig. 11. Post-Implementation Functional Simulation with 10us counting interval

In post-implementation simulation, we still see the same result that the functional simulation is in line with our design and instantaneous change occur in timing simulation. But this change still do not influence our actual result.

Also, **we can find that the time delay after the clock rising goes to decimal_counter in implementation is 8.956ns, which in synthesis is 4.633ns. The delay in implementation is nearly twice longer than synthesis, this time adding is due to the layout and routing**.

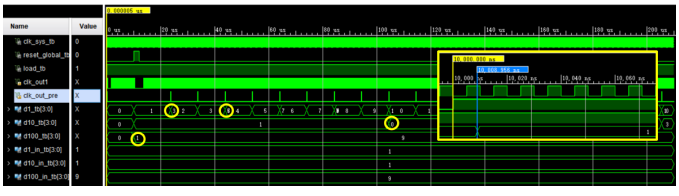


Fig. 12. Post-Implementation Timingl Simulation with 10us counting interval

The schematic is very similar to our block diagram, which verify our code's correctness.

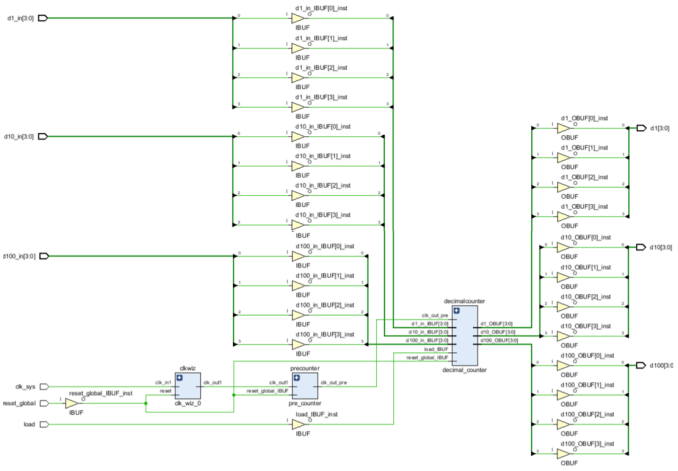


Fig. 13. Schematic

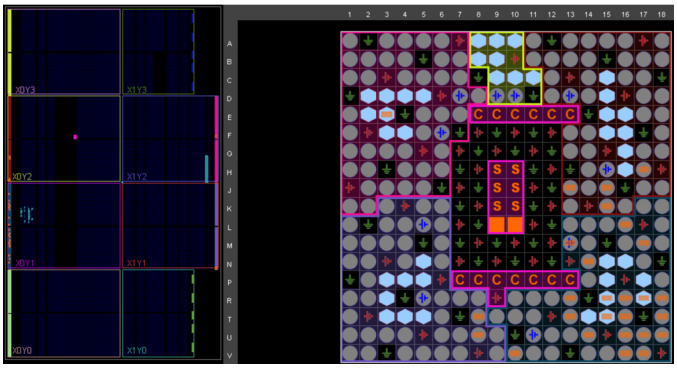
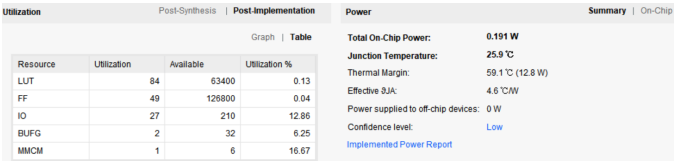


Fig. 14. Device and Package

The device, package result are shown below:
The report is shown below:



Resource	Utilization	Available	Utilization %
LUT	84	63400	0.13
FF	49	126800	0.04
IO	27	210	12.86
BUFG	2	32	6.25
MMCM	1	6	16.67

Total On-Chip Power:	0.191 W
Junction Temperature:	25.9 °C
Thermal Margin:	59.1 °C (12.8 W)
Effective θJA:	4.6 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Fig. 15. Report

We can find that our design takes little resources on board.

5 On Board Verify

And finally, we set the frequency to 1Hz and generate the bitstream and program to the board, the decimal counter result is correct. The result is shown in Fig16.

Note that in simulation we could not use 1s for it takes too long to see one count, but in actual test we can use it directly.

6 Conclusion: Problem in This Lab

Although the structure for the decimal counter is very simple, I still met many problem in this lab and found them hard to debug. I write down several main problems here to summary this lab.

Problem1: The instantaneous change in Timing simulation result. This has been explained in the Post-Synthesis Simulation

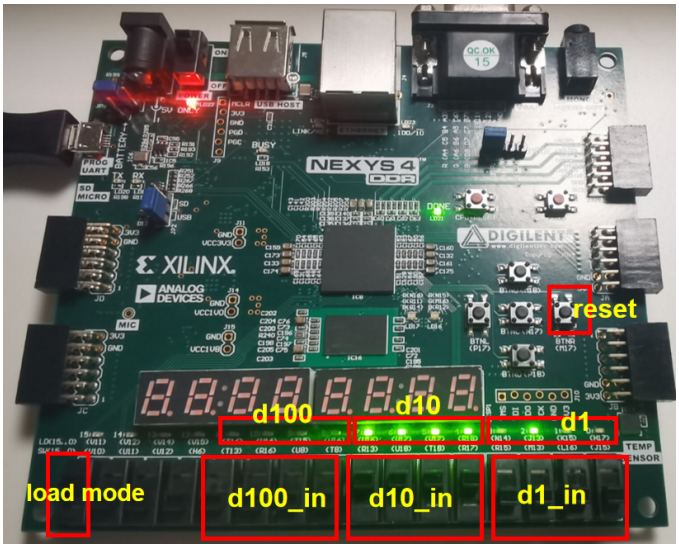


Fig. 16. Test on board(note that the load mode is '1' and after the reset, the load number is given to the output LEDs)

part. Although it is easy to understand, I find it hard to be thought of without experience in Digital system design.

Problem2: this is problem of thinking method in debug When I firstly get the simulation result of timing result shown as follow, the d1 change to 1 with a really short time and change to 2, all the even value has this problem. **A good way to debuug is find the root of the problem**. I quickly realize that this may be caused by the 100000Hz signal from the pre_counter, so I find this clock signal and show it in waveform. Inevitably, it has a instantaneous strange pulse. Then I find the counter in the pre_counter to see its value and find it change several times near the 10M clock change position. And this is also caused by the reason in Problem1. The 24 length number once add '1', there are many bits change and this will cause a mess.

From this debug method, we approached the truth step by step and solve the problem through theoretical analysis and project experience.

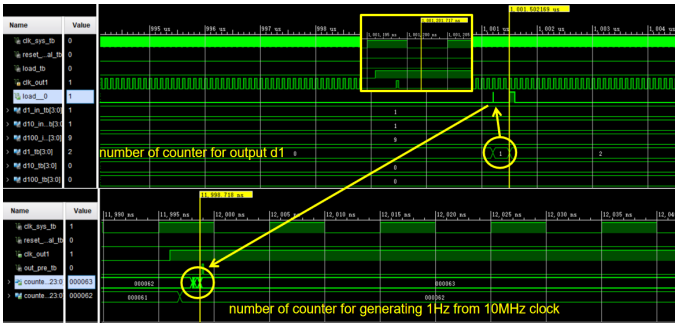


Fig. 17. Problem met in the simulation

Problem3: Different VHDL code, although with the same logic, may generate different simulation result due to the synthesis in VIVADO. In this lab, for the pre_counter, I tried totally five versions of codes to generate the true 1Hz clock. Although these versions code have the same logic, the difference in the if statements' condition or the bits for the count number(24 bits is finally used). **As we can see, it is hard for us to predict the synthesis result, what we should do is try to use standard code and carefully analyze the results**.

In general, from this lab we learned how to use VIVADO simulation for the Sequential logic circuit–a decimal counter. Two segments structure is very useful in the model structure code to achieve the task. Also, we learned the ip core usage, some simulation skill in choosing time interval.

We also learned how to quickly locate the problems and solve them. This is really valuable for our following experience and project ability.