Lab3 Seven-Segment 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 structure. This report summaries the process to achieve seven-segment display and use seven-segment to display a digital counter result. Based on the previous knowledge of two segments structure, structure design, the design is completed accurately and efficiently. The detailed simulation process will be discussed to analyse the systems from different aspects.

Index Terms—seven-segment display, system structure design, VHDL.

1 Introduction

SEVEN segment display is an electronic display device used to display decimal numbers and letters. It can display numbers (0-9), letters (A-F), and other customize characters under control. Seven segment display is widely used in digital clock, electronic instrument, basic calculator and other electronic equipment displaying digital information as an effective means of information feedback and human-computer interaction.

The 7-segment display is composed of 7 stripe LED segments (some 7-segment displays also have decimal point LED), which are arranged together in the shape of ”8”. Different values are generated by different lighting combinations of the seven stripe LEDs. There are two common types of seven segment displays, common anode and common cathode. A common anode 7-segment display connects all the anodes together, while a common cathode 7-segment display connects all the cathodes together.

Fig. 1. Seven-segment digital display tube

This experiment is the first time we use the learned VHDL knowledge to solve a practical case. According to the device principle of on-board peripherals, we use VHDL logic to realize some functions. Specifically, according to the working principle of the seven segment display, we will write the anode driver to realize visual persistence at 1KHz, and write the program of segment display to present the correct characters on the corresponding seven-segment position driven by the anode.

Then we will use the local part simulation method to verify our small modules respectively. For example, the hex-to-7 decoder and anode driver. This will greatly improve the efficiency of our system design. Then, we simulate and verify the whole system to ensure that there is no problem in all levels of simulation. Finally, we carry out on-board verification to realize a system that accepts four 4-bit digital inputs (recorded as A, B, C and D), and presents A, B, C, D, A+B, A-B, C+D and C-D in eight seven segment respectively. Finally, we will evaluate the resource occupation and delay of the system we built.

This is a project with both sequential logic and combinational logic. We will see that the two-segments structure will still play an important role in the process of writing code. At the same time, because of the multi-level structure of the system, the division of levels and the naming of signals are all important issues to be considered. Detailed code description, module block diagram, simulation circuit diagram and waveform diagram will be discussed carefully.

*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.*

2 Pre-lab Preparation

2.1 Principle of seven-segment

As shown in the Nexys user guide, the circuit for the seven segment is:

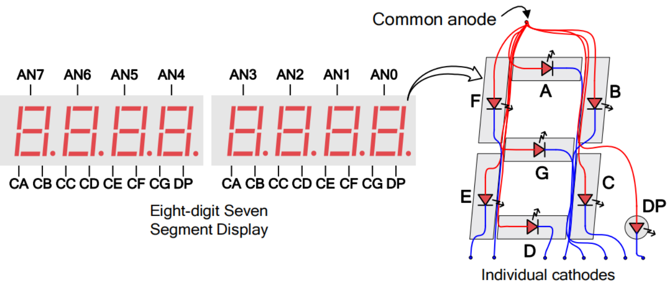


Fig. 2. Circuit for seven segment on Nexys

The seven segment display is a common anode connection mode. At the same time, Nexys connects eight seven segment displays to the same group of cathode control lines. The user lights up the eight seven segment displays by continuously and quickly cycling at the anode, and controls the cathode to display the characters in the correct position when the anode circulates to the target seven segment display. Due to the visual persistence effect, each seven point display should be refreshed continuously at a frequency of 1kHz to 60Hz.

2.2 Block Diagram

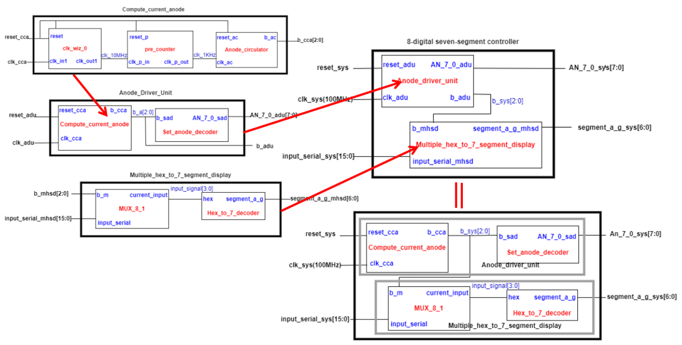


Fig. 3. Block diagram for the system

I design the diagram as follow: **Basically, the 8-digital seven-segmant controller contains two main parts, the Anode_driver_unit and the Multiple_hex_to_7_segment_display**. Anode_driver_unit contains the Compute_current_anode and Set_anode_decoder parts,

Compute_current_anode accept the input 100MHz clk signal ans reset signal, and generate a 3-bit vector indicating which seven-segment should be lighted. And Set_anode_decoder accept this vector and generate a 8-bit signal to control the lighting of anodes.

According to the previous lab digital counter, we should avoid using “combinational logic with registers” to generate clock signal especially when the frequency are high. So we still use Clock Wizard of IP core to generate a 10MHz signal from board 100MHz crystal oscillator. Then we use a pre_counter to generate the 1KHz signal from 10MHz signal.

Multiple_hex_to_7_segment_display accept the 16-bit input as the values of A, B, C and D. It contains MUX_8_1 and Hex_to_7_decoder, MUX_8_1 pick out the signal to be displayed from the input according to the 3-bit vector from Compute_current_anode. Hex_to_7_decoder transfer the picked 4-bit signal into the lighting combination of LED.

2.3 Truth Table

The combinational logic of this system mainly contains the case statements. And the truth table for Hex_to_7_decoder and Set_anode_decoder are shown as follow:

hex values	input”hex”	a	b	c	d	e	f	g	output”segment a-g”
0	0000	0	0	0	0	0	0	1	0000001
1	0001	1	0	0	1	1	1	1	1001111
2	0010	0	0	1	0	0	1	0	0010010
3	0011	0	0	0	0	1	1	0	0000110
4	0100	1	0	0	1	1	0	0	1001100
5	0101	0	1	0	0	1	0	0	0100100
6	0110	0	1	0	0	0	0	0	0100000
7	0111	0	0	0	1	1	1	1	0001111
8	1000	0	0	0	0	0	0	0	0000000
9	1001	0	0	0	0	1	0	0	0000100
A	1010	0	0	0	1	0	0	0	0001000
b	1011	1	1	0	0	0	0	0	1100000
C	1100	0	1	1	0	0	0	1	0110001
d	1101	1	0	0	0	0	1	0	1000010
E	1110	0	1	1	0	0	0	0	0110000
F	1111	0	1	1	1	0	0	0	0111000

Fig. 4. Truth table for Hex_to_7_decoder

For Hex_to_7_decoder the value for output sgement is determined by the shape of the seven-segment.

The lighting seven-segment location has the value 0.

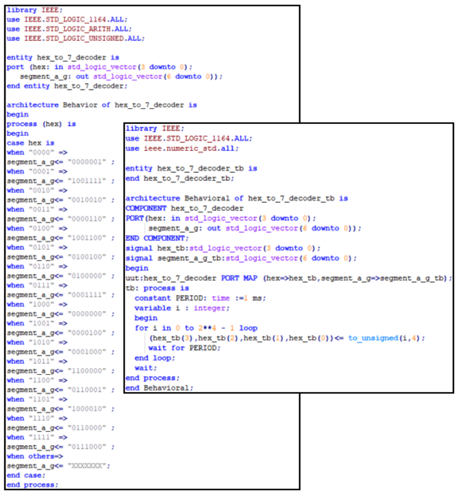
current segment	input ”current”	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	output ”AN_7_0”
AN0	000	1	1	1	1	1	1	1	0	11111110
AN1	001	1	1	1	1	1	1	0	1	11111101
AN2	010	1	1	1	1	1	0	1	1	11111011
AN3	011	1	1	1	1	0	1	1	1	11110111
AN4	100	1	1	1	0	1	1	1	1	11101111
AN5	101	1	1	0	1	1	1	1	1	11011111
AN6	110	1	0	1	1	1	1	1	1	10111111
AN7	111	0	1	1	1	1	1	1	1	01111111

Fig. 5. truth table for Set_anode_decoder

2.4 VHDL code and Local Part Simulation

2.4.1 Hex_to_7_decoder

We complete the code according to the truth table.



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hex_to_7_decoder is
port (hex: in std_logic_vector(3 downto 0);
   segment_a_g: out std_logic_vector(6 downto 0));
end entity hex_to_7_decoder;

architecture Behavior of hex_to_7_decoder is
begin
process (hex) is
begin
case hex is
when "0000" =>
segment_a_g<= "0000001" ;
when "0001" =>
segment_a_g<= "1001111" ;
when "0010" =>
segment_a_g<= "0010010" ;
when "0011" =>
segment_a_g<= "0000110" ;
when "0100" =>
segment_a_g<= "1001100" ;
when "0101" =>
segment_a_g<= "0100100" ;
when "0110" =>
segment_a_g<= "0100000" ;
when "0111" =>
segment_a_g<= "0001111" ;
when "1000" =>
segment_a_g<= "0000000" ;
when "1001" =>
segment_a_g<= "0000100" ;
when "1010" =>
segment_a_g<= "0001000" ;
when "1011" =>
segment_a_g<= "1100000" ;
when "1100" =>
segment_a_g<= "0110001" ;
when "1101" =>
segment_a_g<= "1000010" ;
when "1110" =>
segment_a_g<= "0110000" ;
when "1111" =>
segment_a_g<= "0111000" ;
when others=>
segment_a_g<= "XXXXXXX";
end case;
end process;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity hex_to_7_decoder_tb is
end hex_to_7_decoder_tb;

architecture Behavioral of hex_to_7_decoder_tb is
COMPONENT hex_to_7_decoder
PORT(hex: in std_logic_vector(3 downto 0);
     segment_a_g: out std_logic_vector(6 downto 0));
END COMPONENT;
signal hex_tb:std_logic_vector(3 downto 0);
signal segment_a_g_tb:std_logic_vector(6 downto 0);
begin
uut:hex_to_7_decoder PORT MAP (hex=>hex_tb,segment_a_g=>segment_a_g_tb);
tb: process is
  constant PERIOD: time :=1 ms;
  variable i : integer;
  begin
  for i in 0 to 2**4 - 1 loop
     (hex_tb(3),hex_tb(2),hex_tb(1),hex_tb(0))<= to_unsigned(i,4);
     wait for PERIOD;
  end loop;
  wait;
end process;
end Behavioral;
```

Fig. 6. Hex_to_7_decoder and its testbench

We generate the different level simulation for the Hex_to_7_decoder. Results are shown in Fig 7.

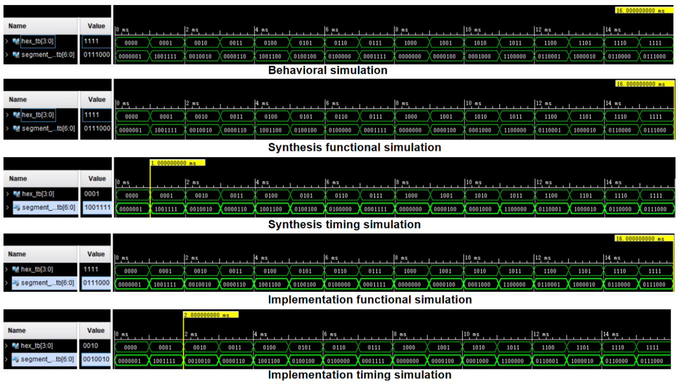

Behavioral simulation
Synthesis functional simulation
Synthesis timing simulation
Implementation functional simulation
Implementation timing simulation

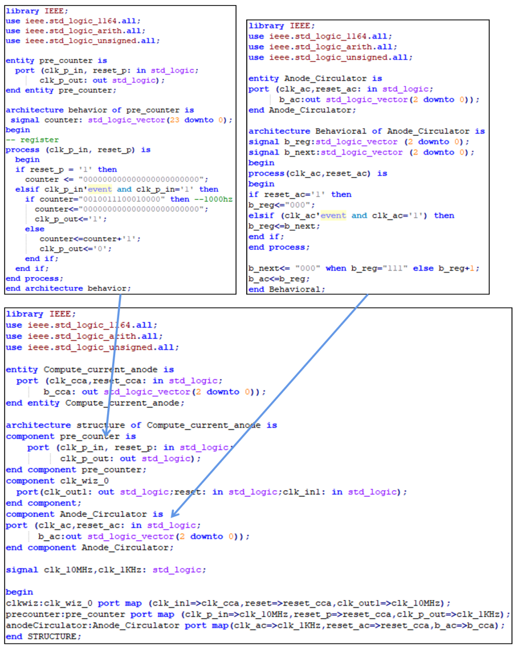
Fig. 7. Hex_to_7_decoder simulation results

From the result, for each condition input, the component generate true results. And from the timing simulation result, the output do not have a obvious delay for the circuit is relatively simple.

2.4.2 Anode_driver_unit

The code for the Compute_current_anode is shown in Fig 8: It contains the clk_wiz_0, pre_counter and Anode_circulator.

Pre_counter use a signal to record the input 10MHz clk, and each 10000 input clk generate a clk output. Anode_circulator use a reg to achieve the loop of 3-bit signal b indicating which seven-segment to be lighted.



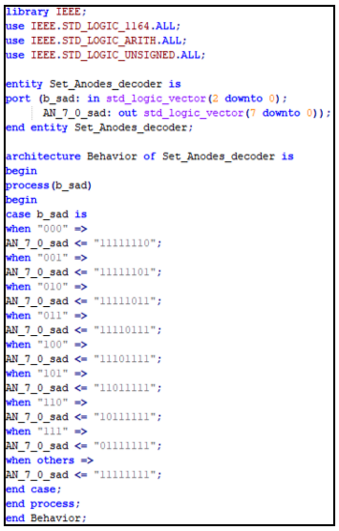
```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity pre_counter is
  port (clk_p_in, reset_p: in std_logic;
       clk_p_out: out std_logic);
end entity pre_counter;

architecture behavior of pre_counter is
 signal counter: std_logic_vector(23 downto 0);
begin
-- register
process (clk_p_in, reset_p) is
  begin
  if reset_p = '1' then
    counter <= "000000000000000000000000";
  elsif clk_p_in'event and clk_p_in='1' then
    if counter="0010011100010000" then --1000hz
      counter<="000000000000000000000000";
      clk_p_out<='1';
    else
       counter<=counter+'1';
       clk_p_out<='0';
    end if;
  end if;
end process;
end architecture behavior;
```

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

entity Anode_Circulator is
port (clk_ac,reset_ac: in std_logic;
      b_ac:out std_logic_vector(2 downto 0));
end Anode_Circulator;

architecture Behavioral of Anode_Circulator is
signal b_reg:std_logic_vector (2 downto 0);
signal b_next:std_logic_vector (2 downto 0);
begin
process(clk_ac,reset_ac) is
begin
if reset_ac='1' then
b_reg<="000";
elsif (clk_ac'event and clk_ac='1') then
b_reg<=b_next;
end if;
end process;

b_next<= "000" when b_reg="111" else b_reg+1;
b_ac<=b_reg;
end Behavioral;
```

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

entity Compute_current_anode is
  port (clk_cca,reset_cca: in std_logic;
       b_cca: out std_logic_vector(2 downto 0));
end entity Compute_current_anode;

architecture structure of Compute_current_anode is
component pre_counter is
    port (clk_p_in, reset_p: in std_logic;
          clk_p_out: 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 Anode_Circulator is
port (clk_ac,reset_ac: in std_logic;
     b_ac:out std_logic_vector(2 downto 0));
end component Anode_Circulator;

signal clk_10MHz,clk_1KHz: std_logic;

begin
clkwiz:clk_wiz_0 port map (clk_in1=>clk_cca,reset=>reset_cca,clk_out1=>clk_10MHz);
precounter:pre_counter port map (clk_p_in=>clk_10MHz,reset_p=>reset_cca,clk_p_out=>clk_1KHz);
anodeCirculator:Anode_Circulator port map(clk_ac=>clk_1KHz,reset_ac=>reset_cca,b_ac=>b_cca);
end STRUCTURE;
```

Fig. 8. Compute_current_anode VHDL code

And the code for Set_anode_decoder is in Fig 9. We complete this part according to the truth table.



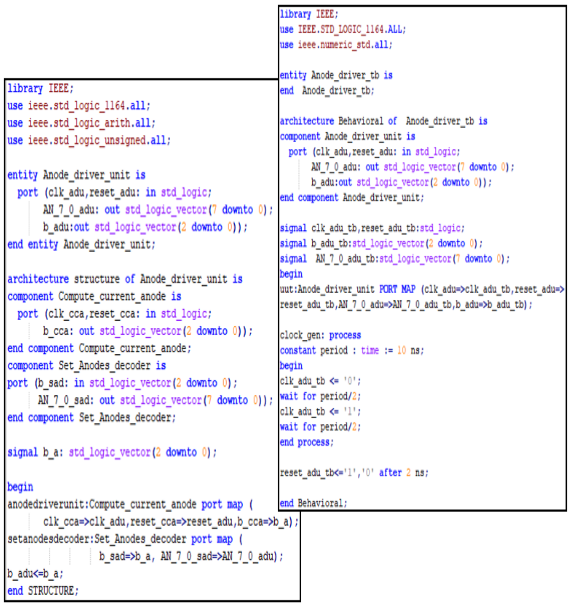
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Set_Anodes_decoder is
port (b_sad: in std_logic_vector(2 downto 0);
     AN_7_0_sad: out std_logic_vector(7 downto 0));
end entity Set_Anodes_decoder;

architecture Behavior of Set_Anodes_decoder is
begin
process(b_sad)
begin
case b_sad is
when "000" =>
AN_7_0_sad <= "11111110";
when "001" =>
AN_7_0_sad <= "11111101";
when "010" =>
AN_7_0_sad <= "11111011";
when "011" =>
AN_7_0_sad <= "11110111";
when "100" =>
AN_7_0_sad <= "11101111";
when "101" =>
AN_7_0_sad <= "11011111";
when "110" =>
AN_7_0_sad <= "10111111";
when "111" =>
AN_7_0_sad <= "01111111";
when others =>
AN_7_0_sad <= "11111111";
end case;
end process;
end Behavior;
```

Fig. 9. Set_anode_decoder VHDL code

The code for Anode_Driver_Unit and testbench is in Fig 10. We use the structure VHDL and combine Set_anode_decoder and Compute_current_anode together.

The simulation results are shown in Fig 11.

From the result, we run 9ms and find that the lighting order of the seven-segment is applied according to our setting. The timing simulation result shows small delay for the output.



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

entity Anode_driver_unit is
  port (clk_adu,reset_adu: in std_logic;
       AN_7_0_adu: out std_logic_vector(7 downto 0);
       b_adu:out std_logic_vector(2 downto 0));
end entity Anode_driver_unit;

architecture structure of Anode_driver_unit is
component Compute_current_anode is
  port (clk_cca,reset_cca: in std_logic;
       b_cca: out std_logic_vector(2 downto 0));
end component Compute_current_anode;
component Set_Anodes_decoder is
port (b_sad: in std_logic_vector(2 downto 0);
     AN_7_0_sad: out std_logic_vector(7 downto 0));
end component Set_Anodes_decoder;

signal b_a: std_logic_vector(2 downto 0);

begin
anodedriverunit:Compute_current_anode port map (
       clk_cca=>clk_adu,reset_cca=>reset_adu,b_cca=>b_a);
setanodesdecoder:Set_Anodes_decoder port map (
                  b_sad=>b_a, AN_7_0_sad=>AN_7_0_adu);
b_adu<=b_a;
end STRUCTURE;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity Anode_driver_tb is
end  Anode_driver_tb;

architecture Behavioral of  Anode_driver_tb is
component Anode_driver_unit is
  port (clk_adu,reset_adu: in std_logic;
       AN_7_0_adu: out std_logic_vector(7 downto 0);
       b_adu:out std_logic_vector(2 downto 0));
end component Anode_driver_unit;

signal clk_adu_tb,reset_adu_tb:std_logic;
signal b_adu_tb:std_logic_vector(2 downto 0);
signal  AN_7_0_adu_tb:std_logic_vector(7 downto 0);
begin
uut:Anode_driver_unit PORT MAP (clk_adu=>clk_adu_tb,reset_adu=>
reset_adu_tb,AN_7_0_adu=>AN_7_0_adu_tb,b_adu=>b_adu_tb);

clock_gen: process
constant period : time := 10 ns;
begin
clk_adu_tb <= '0';
wait for period/2;
clk_adu_tb <= '1';
wait for period/2;
end process;

reset_adu_tb<='1','0' after 2 ns;

end Behavioral;
```

Fig. 10. Anode_Driver_Unit and testbench

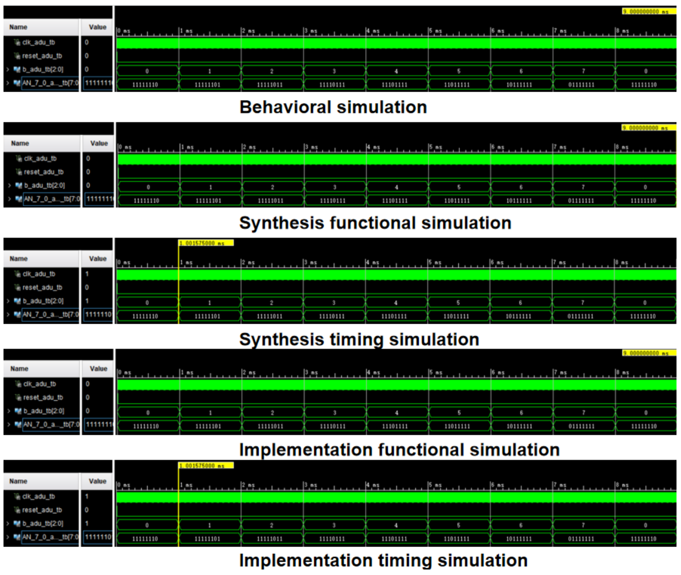


Fig. 11. Anode_Driver_Unit simulation result

2.4.3 Code for the final system:8-digital seven-segment controller

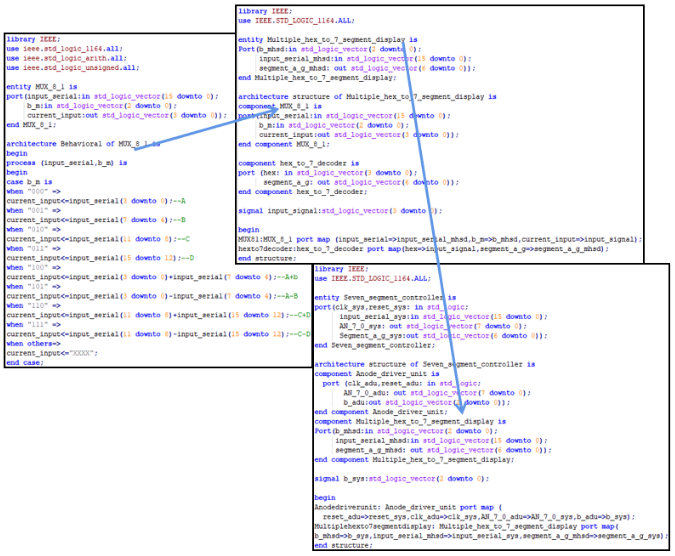
Finally, using the above local design, we complete the structure design of the full system, as shown in Fig 12. For the MUX_8_1 part, we cut input serial into signal A, B, C and D then generate the 8 seven-segment value of A, B, C, D, A+B, A-B, C+D, C-D. Pass the right signal to the Hex_to_7_decoder according to the 3-bit signal b.

3 Simulation Result

3.1 Behavioral simulation

The testbench code for the full system is shown in Fig13.

We set input serial as ”1010110010010111”, that is, A is ”0111”(7), B is ”1001”(9), C is ”1100”(C), D is ”1010”(A). In the calculation, we ignore the effects of sign, underflow and overflow.



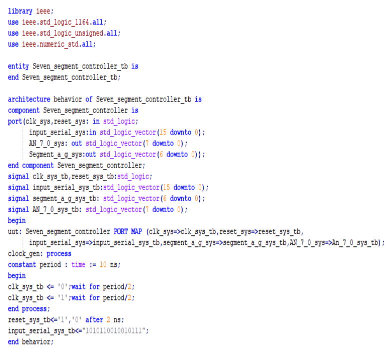
```vhdl
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX_8_1 is
port(input_serial:in std_logic_vector(15 downto 0);
     b_m:in std_logic_vector(2 downto 0);
     current_input:out std_logic_vector(3 downto 0));
end MUX_8_1;

architecture Behavioral of MUX_8_1 is
begin
process (input_serial,b_m) is
begin
case b_m is
when "000" =>
current_input<=input_serial(3 downto 0);--A
when "001" =>
current_input<=input_serial(7 downto 4);--B
when "010" =>
current_input<=input_serial(11 downto 8);--C
when "011" =>
current_input<=input_serial(15 downto 12);--D
when "100" =>
current_input<=input_serial(3 downto 0)+input_serial(7 downto 4);--A+b
when "101" =>
current_input<=input_serial(3 downto 0)-input_serial(7 downto 4);--A-B
when "110" =>
current_input<=input_serial(11 downto 8)+input_serial(15 downto 12);--C+D
when "111" =>
current_input<=input_serial(11 downto 8)-input_serial(15 downto 12);--C-D
when others=>
current_input<="XXXX";
end case;
```

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

entity Multiple_hex_to_7_segment_display is
Port(b_mhsd:in std_logic_vector(2 downto 0);
     input_serial_mhsd:in std_logic_vector(15 downto 0);
     segment_a_g_mhsd: out std_logic_vector(6 downto 0));
end Multiple_hex_to_7_segment_display;

architecture structure of Multiple_hex_to_7_segment_display is
component MUX_8_1 is
port(input_serial:in std_logic_vector(15 downto 0);
     b_m:in std_logic_vector(2 downto 0);
     current_input:out std_logic_vector(3 downto 0));
end component MUX_8_1;

component hex_to_7_decoder is
port (hex: in std_logic_vector(3 downto 0);
      segment_a_g: out std_logic_vector(6 downto 0));
end component hex_to_7_decoder;

signal input_signal:std_logic_vector(3 downto 0);

begin
MUX81:MUX_8_1 port map (input_serial=>input_serial_mhsd,b_m=>b_mhsd,current_input=>input_signal);
hexto7decoder:hex_to_7_decoder port map(hex=>input_signal,segment_a_g=>segment_a_g_mhsd);
end structure;
```

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

entity Seven_segment_controller is
port(clk_sys,reset_sys: in std_logic;
     input_serial_sys:in std_logic_vector(15 downto 0);
     AN_7_0_sys: out std_logic_vector(7 downto 0);
     Segment_a_g_sys:out std_logic_vector(6 downto 0));
end Seven_segment_controller;

architecture structure of Seven_segment_controller is
component Anode_driver_unit is
  port (clk_adu,reset_adu: in std_logic;
        AN_7_0_adu: out std_logic_vector(7 downto 0);
        b_adu:out std_logic_vector(2 downto 0));
end component Anode_driver_unit;
component Multiple_hex_to_7_segment_display is
Port(b_mhsd:in std_logic_vector(2 downto 0);
     input_serial_mhsd:in std_logic_vector(15 downto 0);
     segment_a_g_mhsd: out std_logic_vector(6 downto 0));
end component Multiple_hex_to_7_segment_display;

signal b_sys:std_logic_vector(2 downto 0);

begin
Anodedriverunit: Anode_driver_unit port map (
  reset_adu=>reset_sys,clk_adu=>clk_sys,AN_7_0_adu=>AN_7_0_sys,b_adu=>b_sys);
Multiplehexto7segmentdisplay: Multiple_hex_to_7_segment_display port map(
b_mhsd=>b_sys,input_serial_mhsd=>input_serial_sys,segment_a_g_mhsd=>segment_a_g_sys);
end structure;
```

Fig. 12. Seven_segment_controller VHDL code

So A+B is ”0000”(0), A-B is ”1110”(E), C+D is ”0110”(6) and C-D is “0010”(2).



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

entity Seven_segment_controller_tb is
end Seven_segment_controller_tb;

architecture behavior of Seven_segment_controller_tb is
component Seven_segment_controller is
port(clk_sys,reset_sys: in std_logic;
     input_serial_sys:in std_logic_vector(15 downto 0);
     AN_7_0_sys: out std_logic_vector(7 downto 0);
     Segment_a_g_sys:out std_logic_vector(6 downto 0));
end component Seven_segment_controller;
signal clk_sys_tb,reset_sys_tb:std_logic;
signal input_serial_sys_tb:std_logic_vector(15 downto 0);
signal segment_a_g_sys_tb: std_logic_vector(6 downto 0);
signal AN_7_0_sys_tb: std_logic_vector(7 downto 0);
begin
uut: Seven_segment_controller PORT MAP (clk_sys=>clk_sys_tb,reset_sys=>reset_sys_tb,
     input_serial_sys=>input_serial_sys_tb,segment_a_g_sys=>segment_a_g_sys_tb,AN_7_0_sys=>An_7_0_sys_tb);
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;
reset_sys_tb<='1','0' after 2 ns;
input_serial_sys_tb<="1010110010010111";
end behavior;
```

Fig. 13. Testbench code

The behavioral simulation is shown as follow: We can find that the output AN_7_0 and segment_a_g get the right value the same as our derivation. And they change value every millisecond.

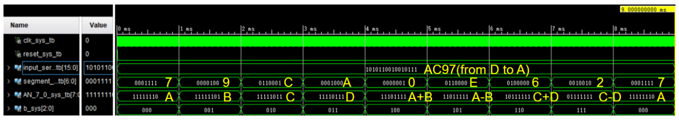


Fig. 14. Behavioral simulation

3.2 Post-Synthesis Simulation

Then we run the post-synthesis simulation, the functional and timing result are shown as follow:

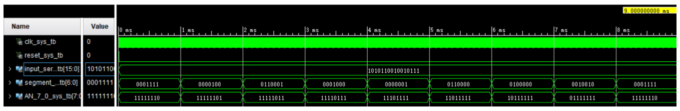


Fig. 15. Post-Synthesis functional Simulation

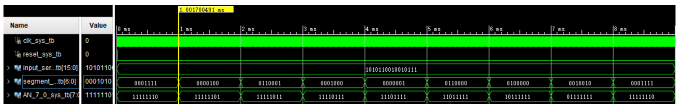


Fig. 16. Post-Synthesis timing Simulation

In the timing result, we find that about 0.0017ms delay for each period. This delay is due to the gate delay and is not obvious in actual observation.

3.3 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, input serial to switches ports and anode AN_7_0 and segment_a_g result to seven-segment ports.

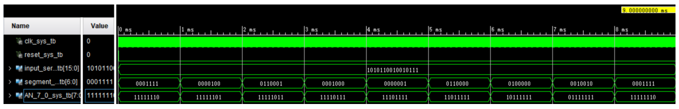


Fig. 17. Post-Implementation functional simulation

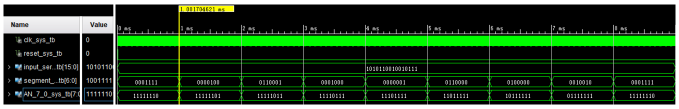


Fig. 18. Post-Implementation timing simulation

From the implementation result, our system work corrctly and the timing simulation delay is a little larger than the synthesis timing result for the layout and wiring.

The schematic for the implementation simulation is shown in Fig 19.

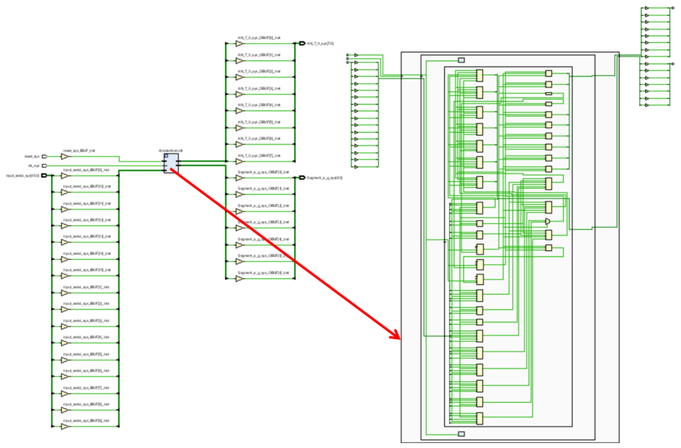
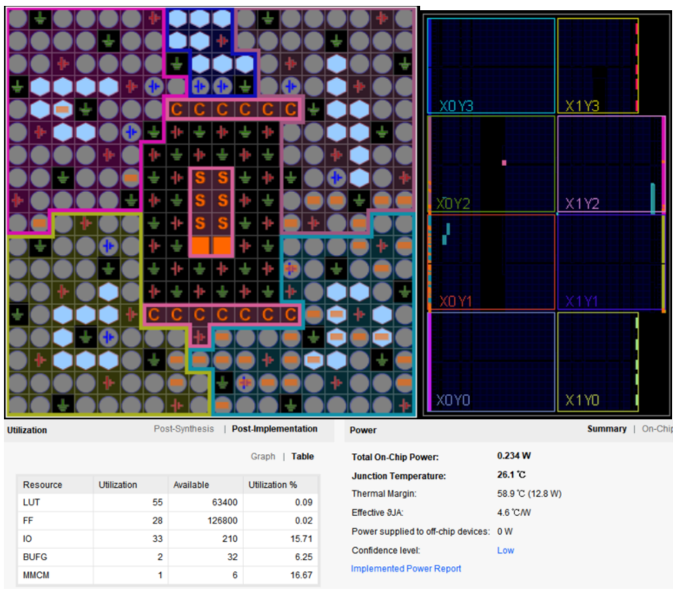


Fig. 19. Schematic

We can see a clear structure of the input and output, for the main part, we show the internal circuit, We can clearly see the hierarchical structure and parts of the circuit.

The device, package and resources are shown in Fig 20.

We can find our design takes little resources.



Utilization   Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	55	63400	0.09
FF	28	126800	0.02
IO	33	210	15.71
BUFG	2	32	6.25
MMCM	1	6	16.67

Power   **Summary** | On-Chip

Total On-Chip Power:	**0.234 W**
Junction Temperature:	**26.1 ℃**
Thermal Margin:	58.9 ℃ (12.8 W)
Effective ϑJA:	4.6 ℃/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Fig. 20. Package, device and resources

4 On Board Verify

And finally, we generate the bitstream and program to the board, the actual result is correct with our derivation. The result is shown in Fig21. According to the user guide of Nexys, we write the constraints file to set the IO ports.

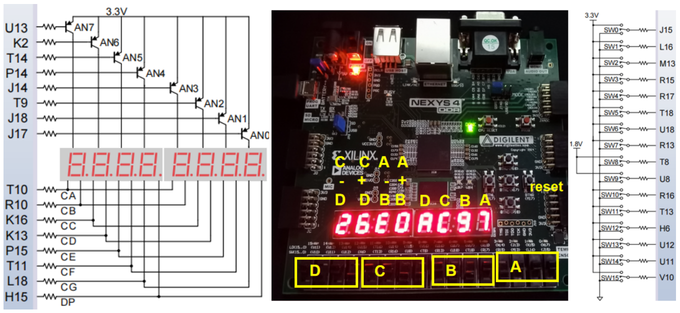


Fig. 21. Test on board

5 Extra Design

In the extra design, I design a digital counter with seven-segment display. Combined this lab and last lab vhdl structure together.

The block diagram for the system is shown in Fig 22. Clk_wiz_0 generate 10MHz clk from system 100MHz clk and pass the clk signal to two different pre_counter. For the digital counter, the pre_counter generate 1Hz clk and for seven-segments the pre_counter genertae 1KHz clk. Then decimal counter pass the output signal d1, d10 and d100 to the seven-segment display input. For the seven-segment other positions, we set them 0. Finally, we do the simulation, and match the input and output signal to the IO ports.

I set the load function to load the number from switches and a reset button to clear the counter. From the actual testing, my design work correctly.

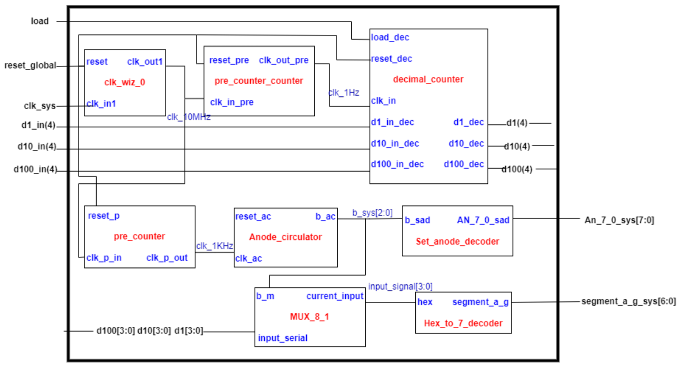


Fig. 22. Block diagram for digital counter

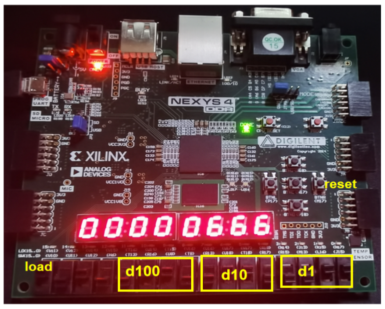


Fig. 23. Counter: test on board

6 Conclusion

In general, from this lab we mainly learned:

(1)How to use VIVADO do simulation for local part of the full system, that is, create different testbench file for different local block.

(2)How to design the circuit to achieve the function fo a hardware peripheral. To understand the principle of the hardware and write VHDL code to achieve.

(3)For a relatively complex digital system, how to group the components and design the layer structure. This part cost me the most time to finish for their are too many signals in the system and naming them is hard. We should find a suitable way to name these signals and write structure in an efficient way. **For example, in this design I use many suffix in the signals' names to mark which components the signal belongs to**.

Also, I met many bugs in this lab and solve them finally. One bug is that for the constraints file, it is case sensitive.

In a word, in this experiment, we combine the learned VHDL knowledge with the actual hardware principle, and use VHDL sequential, combinational logic circuit and two-segment method, structure design to successfully realize the display of seven-segment and display the counting output of a decimal counter.