**微算機系統**

實驗七

組別： 16

班級、姓名與學號：

資工二 陸詠涵 108580451

資工二 陳佳均 108830035

日期： 2020.12.30

1. 實驗內容：

本次實驗主旨目標

利用四個暫存器和七段顯示器做結合，實作簡易CPU。

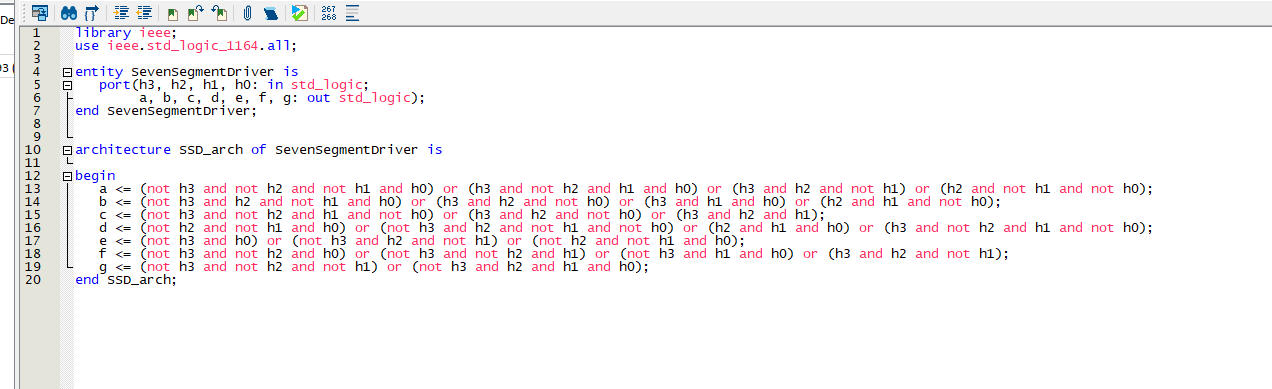
1. 實驗過程及結果：

撰寫你如何完成本次作業的流程方法。

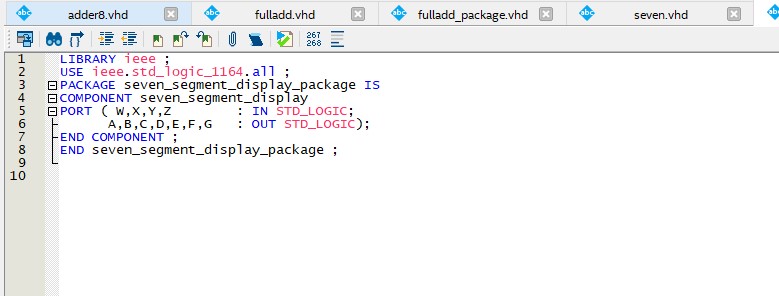
請附上結果圖片(放置於此部分即可)。

實驗步驟:

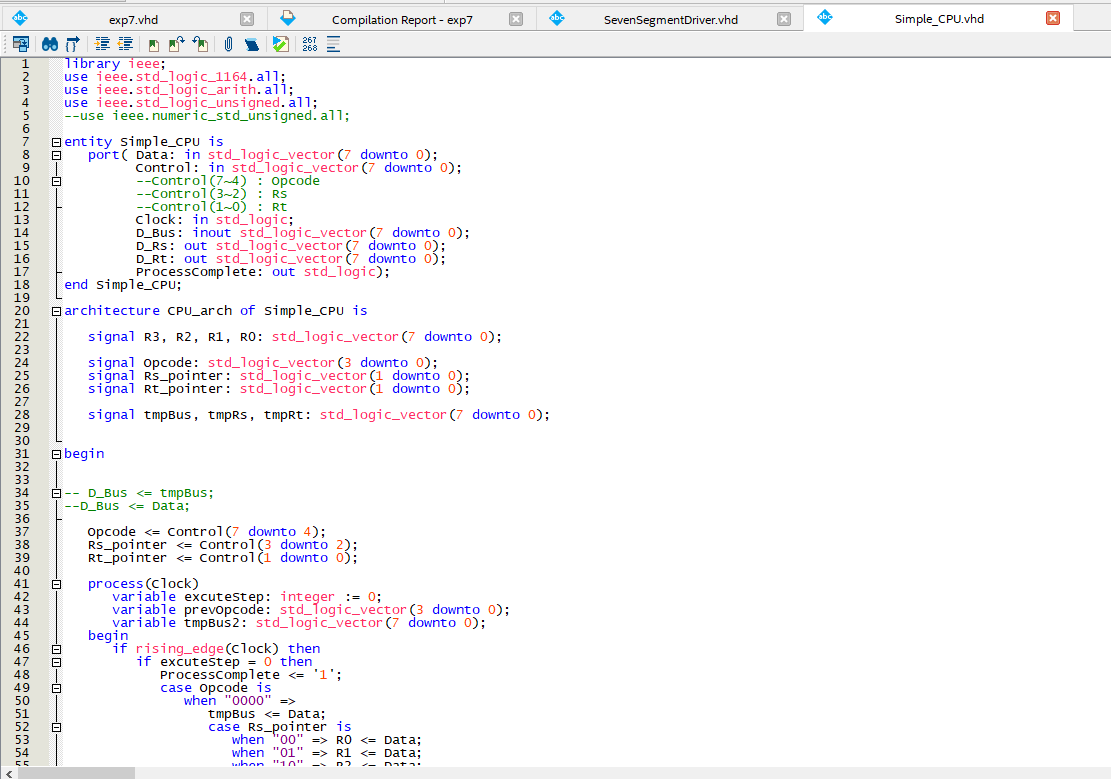
Step1:設計VHDL程式碼

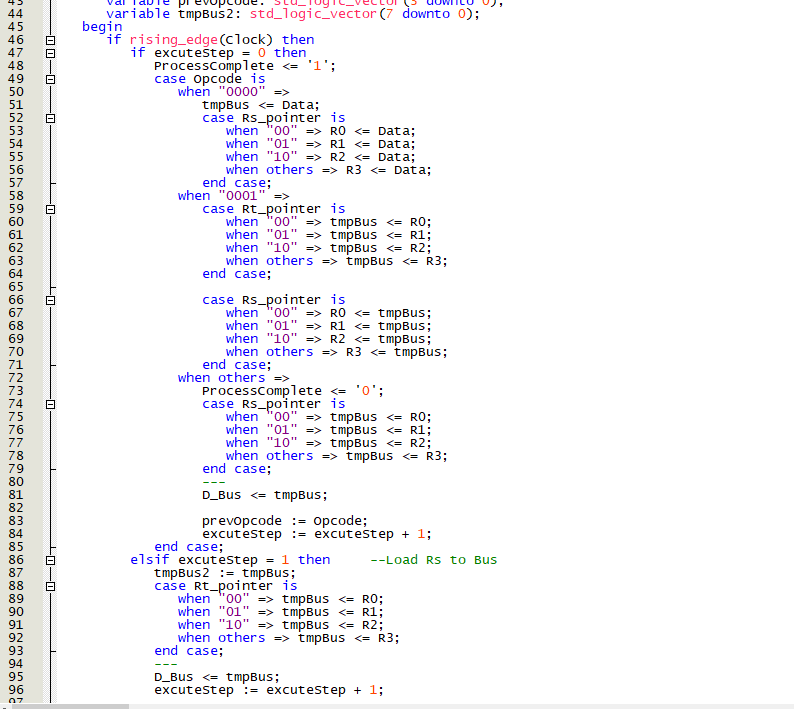


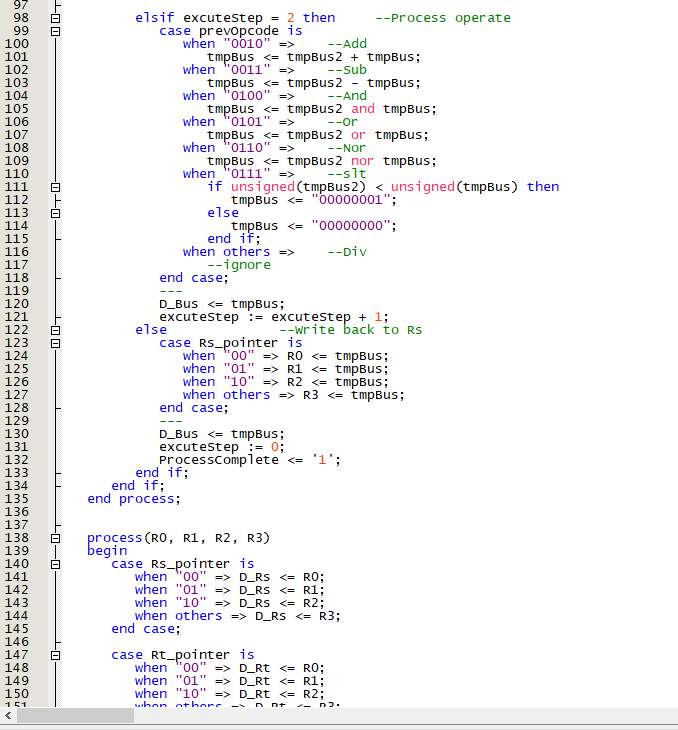
(picture 1)七段顯示器的VHDL程式碼

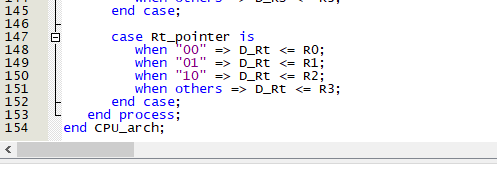


(picture 2)七段顯示器package的VHDL程式碼

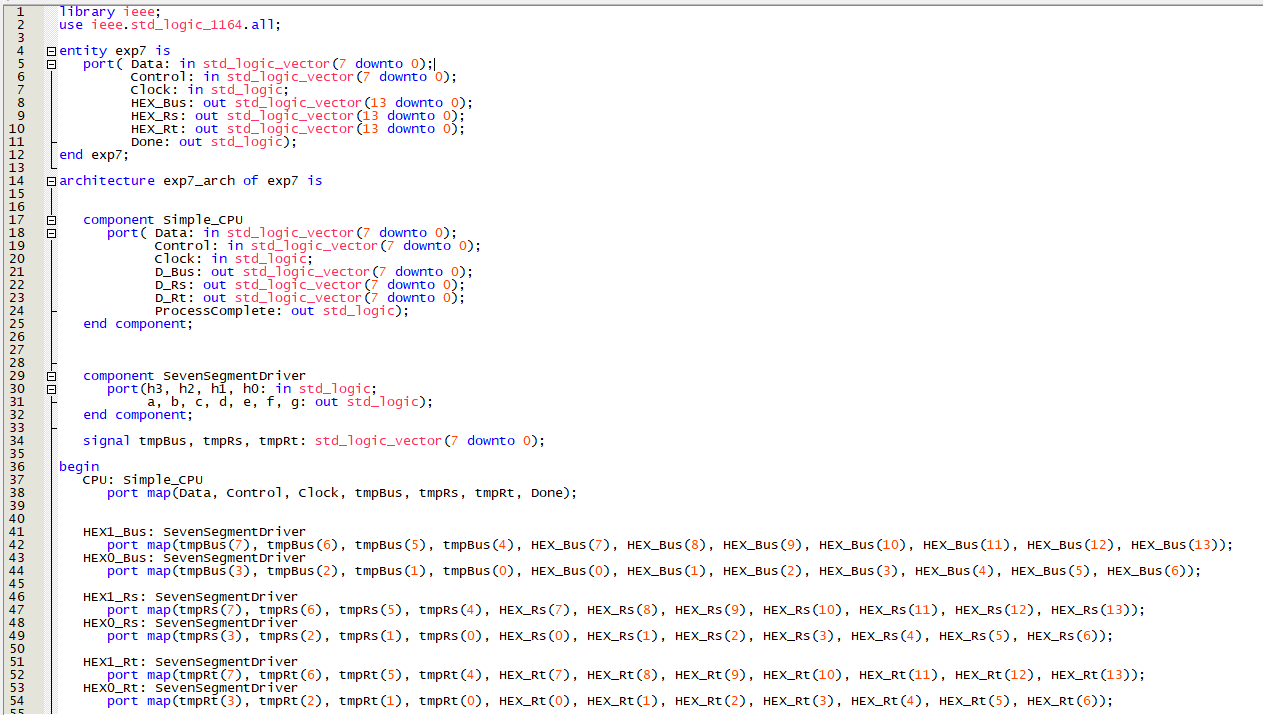


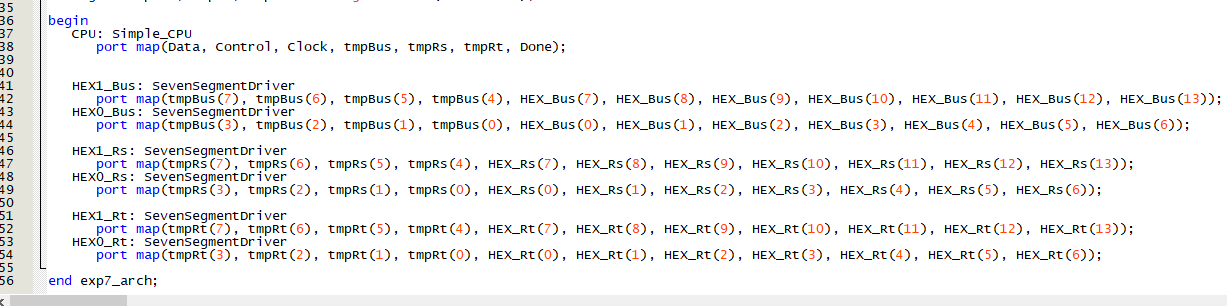






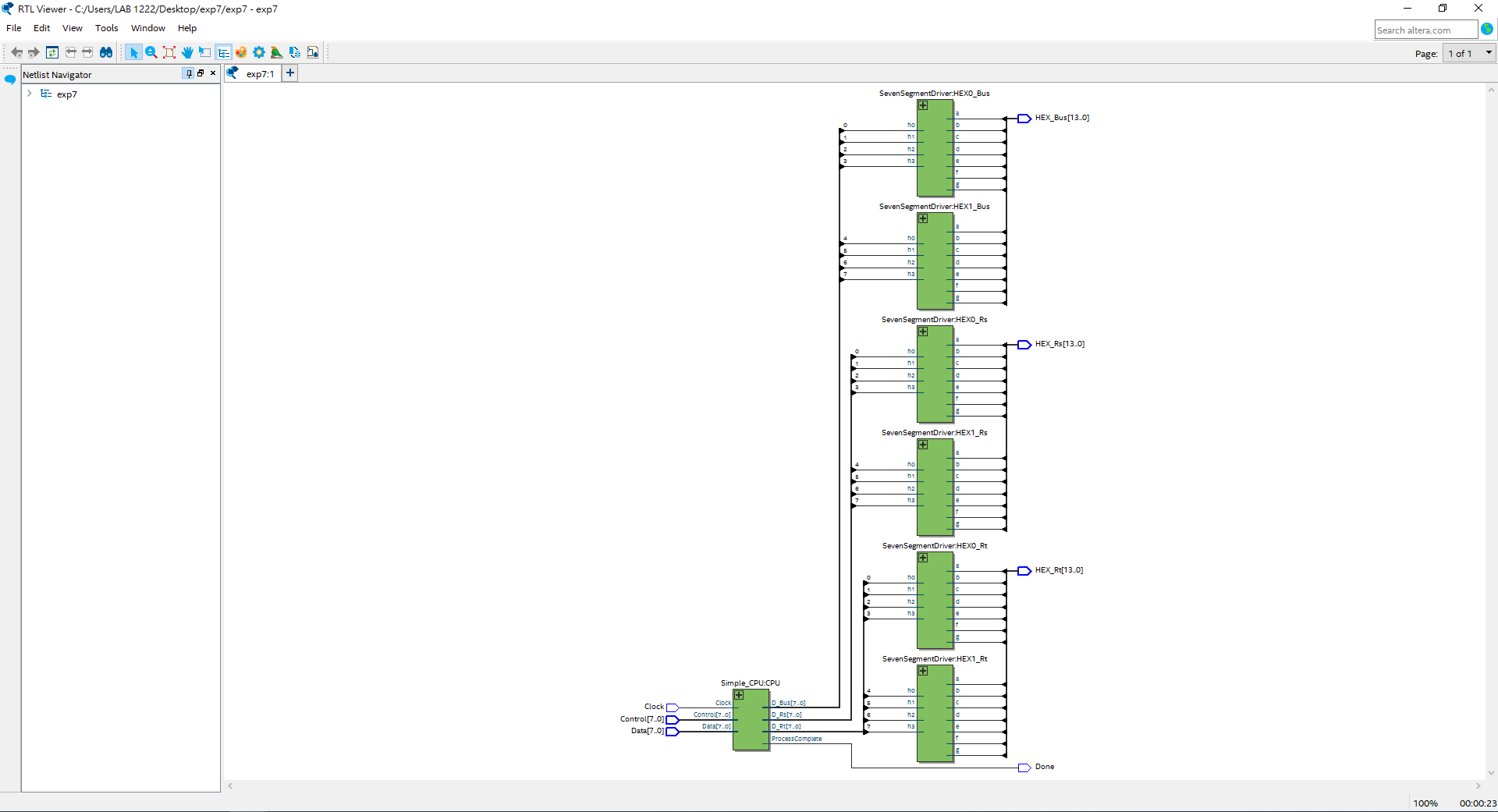
(picture 3)simple CPU程式碼



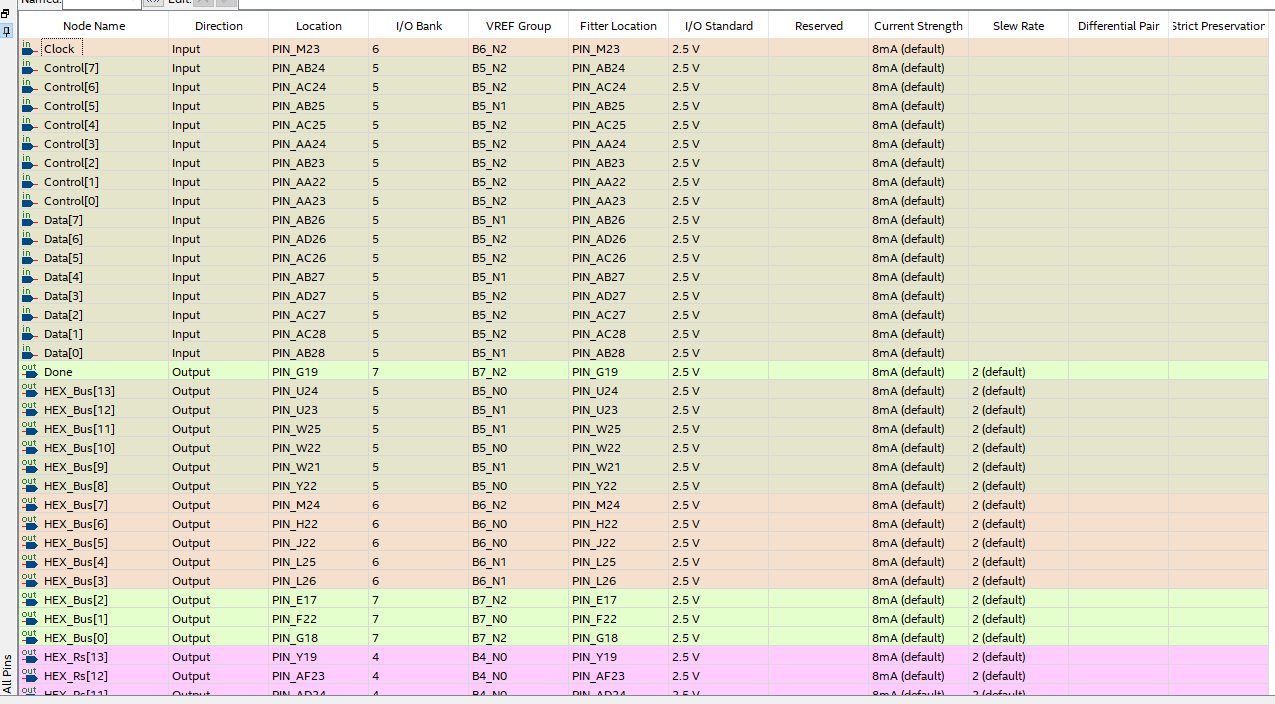


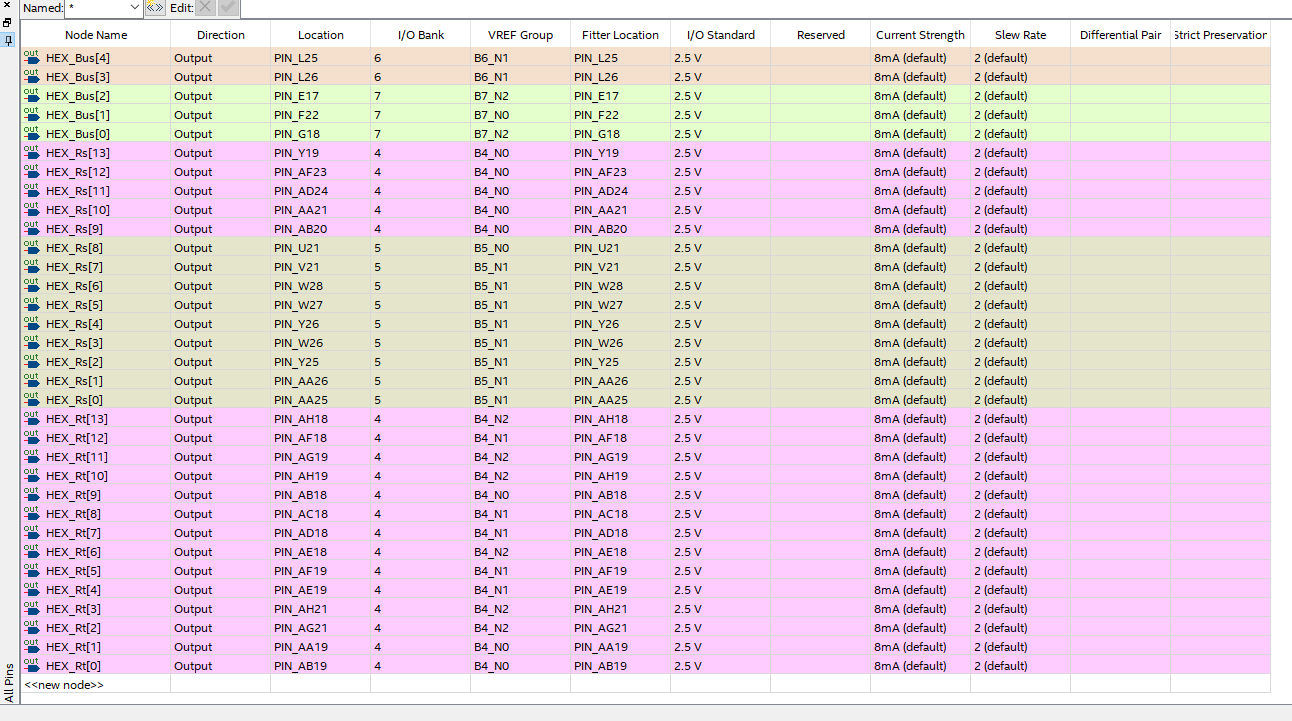
(picture 4)主程式

Step2:接腳位



(picture 5) 邏輯電路圖





(picture 6)所接的腳位

Step3:燒錄到模板中



(picture 7) 結果

1. 程式碼（請調整成最小行高，行高0點）

|  |
| --- |
| 目標一 |
| (程式碼)  **seven\_segment\_display**  library ieee;  use ieee.std\_logic\_1164.all;  entity SevenSegmentDriver is  port(h3, h2, h1, h0: in std\_logic;  a, b, c, d, e, f, g: out std\_logic);  end SevenSegmentDriver;  architecture SSD\_arch of SevenSegmentDriver is  begin  a <= (not h3 and not h2 and not h1 and h0) or (h3 and not h2 and h1 and h0) or (h3 and h2 and not h1) or (h2 and not h1 and not h0);  b <= (not h3 and h2 and not h1 and h0) or (h3 and h2 and not h0) or (h3 and h1 and h0) or (h2 and h1 and not h0);  c <= (not h3 and not h2 and h1 and not h0) or (h3 and h2 and not h0) or (h3 and h2 and h1);  d <= (not h2 and not h1 and h0) or (not h3 and h2 and not h1 and not h0) or (h2 and h1 and h0) or (h3 and not h2 and h1 and not h0);  e <= (not h3 and h0) or (not h3 and h2 and not h1) or (not h2 and not h1 and h0);  f <= (not h3 and not h2 and h0) or (not h3 and not h2 and h1) or (not h3 and h1 and h0) or (h3 and h2 and not h1);  g <= (not h3 and not h2 and not h1) or (not h3 and h2 and h1 and h0);  end SSD\_arch;  **seven\_segment\_display\_package**  LIBRARY ieee ;  USE ieee.std\_logic\_1164.all ;  PACKAGE seven\_segment\_display\_package IS  COMPONENT seven\_segment\_display  PORT ( W,X,Y,Z : IN STD\_LOGIC;  A,B,C,D,E,F,G : OUT STD\_LOGIC);  END COMPONENT ;  END seven\_segment\_display\_package ;  **Simple\_CPU**  library ieee;  use ieee.std\_logic\_1164.all;  use ieee.std\_logic\_arith.all;  use ieee.std\_logic\_unsigned.all;  --use ieee.numeric\_std\_unsigned.all;  entity Simple\_CPU is  port( Data: in std\_logic\_vector(7 downto 0);  Control: in std\_logic\_vector(7 downto 0);  --Control(7~4) : Opcode  --Control(3~2) : Rs  --Control(1~0) : Rt  Clock: in std\_logic;  D\_Bus: inout std\_logic\_vector(7 downto 0);  D\_Rs: out std\_logic\_vector(7 downto 0);  D\_Rt: out std\_logic\_vector(7 downto 0);  ProcessComplete: out std\_logic);  end Simple\_CPU;  architecture CPU\_arch of Simple\_CPU is  signal R3, R2, R1, R0: std\_logic\_vector(7 downto 0);    signal Opcode: std\_logic\_vector(3 downto 0);  signal Rs\_pointer: std\_logic\_vector(1 downto 0);  signal Rt\_pointer: std\_logic\_vector(1 downto 0);    signal tmpBus, tmpRs, tmpRt: std\_logic\_vector(7 downto 0);      begin  -- D\_Bus <= tmpBus;  --D\_Bus <= Data;    Opcode <= Control(7 downto 4);  Rs\_pointer <= Control(3 downto 2);  Rt\_pointer <= Control(1 downto 0);    process(Clock)  variable excuteStep: integer := 0;  variable prevOpcode: std\_logic\_vector(3 downto 0);  variable tmpBus2: std\_logic\_vector(7 downto 0);  begin  if rising\_edge(Clock) then  if excuteStep = 0 then  ProcessComplete <= '1';  case Opcode is  when "0000" =>  tmpBus <= Data;  case Rs\_pointer is  when "00" => R0 <= Data;  when "01" => R1 <= Data;  when "10" => R2 <= Data;  when others => R3 <= Data;  end case;  when "0001" =>  case Rt\_pointer is  when "00" => tmpBus <= R0;  when "01" => tmpBus <= R1;  when "10" => tmpBus <= R2;  when others => tmpBus <= R3;  end case;    case Rs\_pointer is  when "00" => R0 <= tmpBus;  when "01" => R1 <= tmpBus;  when "10" => R2 <= tmpBus;  when others => R3 <= tmpBus;  end case;  when others =>  ProcessComplete <= '0';  case Rs\_pointer is  when "00" => tmpBus <= R0;  when "01" => tmpBus <= R1;  when "10" => tmpBus <= R2;  when others => tmpBus <= R3;  end case;  ---  D\_Bus <= tmpBus;    prevOpcode := Opcode;  excuteStep := excuteStep + 1;  end case;  elsif excuteStep = 1 then --Load Rs to Bus  tmpBus2 := tmpBus;  case Rt\_pointer is  when "00" => tmpBus <= R0;  when "01" => tmpBus <= R1;  when "10" => tmpBus <= R2;  when others => tmpBus <= R3;  end case;  ---  D\_Bus <= tmpBus;  excuteStep := excuteStep + 1;    elsif excuteStep = 2 then --Process operate  case prevOpcode is  when "0010" => --Add  tmpBus <= tmpBus2 + tmpBus;  when "0011" => --Sub  tmpBus <= tmpBus2 - tmpBus;  when "0100" => --And  tmpBus <= tmpBus2 and tmpBus;  when "0101" => --Or  tmpBus <= tmpBus2 or tmpBus;  when "0110" => --Nor  tmpBus <= tmpBus2 nor tmpBus;  when "0111" => --slt  if unsigned(tmpBus2) < unsigned(tmpBus) then  tmpBus <= "00000001";  else  tmpBus <= "00000000";  end if;  when others => --Div  --ignore  end case;  ---  D\_Bus <= tmpBus;  excuteStep := excuteStep + 1;  else --Write back to Rs  case Rs\_pointer is  when "00" => R0 <= tmpBus;  when "01" => R1 <= tmpBus;  when "10" => R2 <= tmpBus;  when others => R3 <= tmpBus;  end case;  ---  D\_Bus <= tmpBus;  excuteStep := 0;  ProcessComplete <= '1';  end if;  end if;  end process;    process(R0, R1, R2, R3)  begin  case Rs\_pointer is  when "00" => D\_Rs <= R0;  when "01" => D\_Rs <= R1;  when "10" => D\_Rs <= R2;  when others => D\_Rs <= R3;  end case;    case Rt\_pointer is  when "00" => D\_Rt <= R0;  when "01" => D\_Rt <= R1;  when "10" => D\_Rt <= R2;  when others => D\_Rt <= R3;  end case;  end process;  end CPU\_arch;  **主程式**  library ieee;  use ieee.std\_logic\_1164.all;  entity exp7 is  port( Data: in std\_logic\_vector(7 downto 0);  Control: in std\_logic\_vector(7 downto 0);  Clock: in std\_logic;  HEX\_Bus: out std\_logic\_vector(13 downto 0);  HEX\_Rs: out std\_logic\_vector(13 downto 0);  HEX\_Rt: out std\_logic\_vector(13 downto 0);  Done: out std\_logic);  end exp7;  architecture exp7\_arch of exp7 is  component Simple\_CPU  port( Data: in std\_logic\_vector(7 downto 0);  Control: in std\_logic\_vector(7 downto 0);  Clock: in std\_logic;  D\_Bus: out std\_logic\_vector(7 downto 0);  D\_Rs: out std\_logic\_vector(7 downto 0);  D\_Rt: out std\_logic\_vector(7 downto 0);  ProcessComplete: out std\_logic);  end component;      component SevenSegmentDriver  port(h3, h2, h1, h0: in std\_logic;  a, b, c, d, e, f, g: out std\_logic);  end component;    signal tmpBus, tmpRs, tmpRt: std\_logic\_vector(7 downto 0);    begin  CPU: Simple\_CPU  port map(Data, Control, Clock, tmpBus, tmpRs, tmpRt, Done);      HEX1\_Bus: SevenSegmentDriver  port map(tmpBus(7), tmpBus(6), tmpBus(5), tmpBus(4), HEX\_Bus(7), HEX\_Bus(8), HEX\_Bus(9), HEX\_Bus(10), HEX\_Bus(11), HEX\_Bus(12), HEX\_Bus(13));  HEX0\_Bus: SevenSegmentDriver  port map(tmpBus(3), tmpBus(2), tmpBus(1), tmpBus(0), HEX\_Bus(0), HEX\_Bus(1), HEX\_Bus(2), HEX\_Bus(3), HEX\_Bus(4), HEX\_Bus(5), HEX\_Bus(6));  HEX1\_Rs: SevenSegmentDriver  port map(tmpRs(7), tmpRs(6), tmpRs(5), tmpRs(4), HEX\_Rs(7), HEX\_Rs(8), HEX\_Rs(9), HEX\_Rs(10), HEX\_Rs(11), HEX\_Rs(12), HEX\_Rs(13));  HEX0\_Rs: SevenSegmentDriver  port map(tmpRs(3), tmpRs(2), tmpRs(1), tmpRs(0), HEX\_Rs(0), HEX\_Rs(1), HEX\_Rs(2), HEX\_Rs(3), HEX\_Rs(4), HEX\_Rs(5), HEX\_Rs(6));    HEX1\_Rt: SevenSegmentDriver  port map(tmpRt(7), tmpRt(6), tmpRt(5), tmpRt(4), HEX\_Rt(7), HEX\_Rt(8), HEX\_Rt(9), HEX\_Rt(10), HEX\_Rt(11), HEX\_Rt(12), HEX\_Rt(13));  HEX0\_Rt: SevenSegmentDriver  port map(tmpRt(3), tmpRt(2), tmpRt(1), tmpRt(0), HEX\_Rt(0), HEX\_Rt(1), HEX\_Rt(2), HEX\_Rt(3), HEX\_Rt(4), HEX\_Rt(5), HEX\_Rt(6));    end exp7\_arch; |
| 目標二 |
| (程式碼) |

1. 實驗心得：

每個人的心得報告至少150字以上，有關於此實驗所遇到的難題，解決方法或是對於實驗過程的分析等。（每名組員都要寫）

108590451陸詠涵的心得:

這次實驗，我們透過不同的op code來執行操作。一開始，我們組其實不太了解為何要使用到decoder，但是，再和其他同學的討論下，了解到所有的運作。所以之後，我們就很快地設計出了load,move,add等語法。隨然，我們在實作燒錄時，move顯示出了的數字有問題。但在後來檢查時，發現是七段顯示器寫錯。所以，改回來之後，就成功了。至於除法器，因為時間來不及，所以就沒做完，只能之後再嘗試看看。

108830035陳佳均的心得：

這次因為有其他同學一起研究要如何作業，所以整體感覺還不錯。比較特別的是，當我們在做move的功能時，出來的數字都是反的，非常的奇葩。好像是因為程式寫反的關係，反正之後重寫就好了。當中除法器的部分，因為後來比較趕就來不及做了，有點可惜，因為我們那時還為了把除法器做出來而壓縮到這次的時間，沒能運用出來，偏可惜。