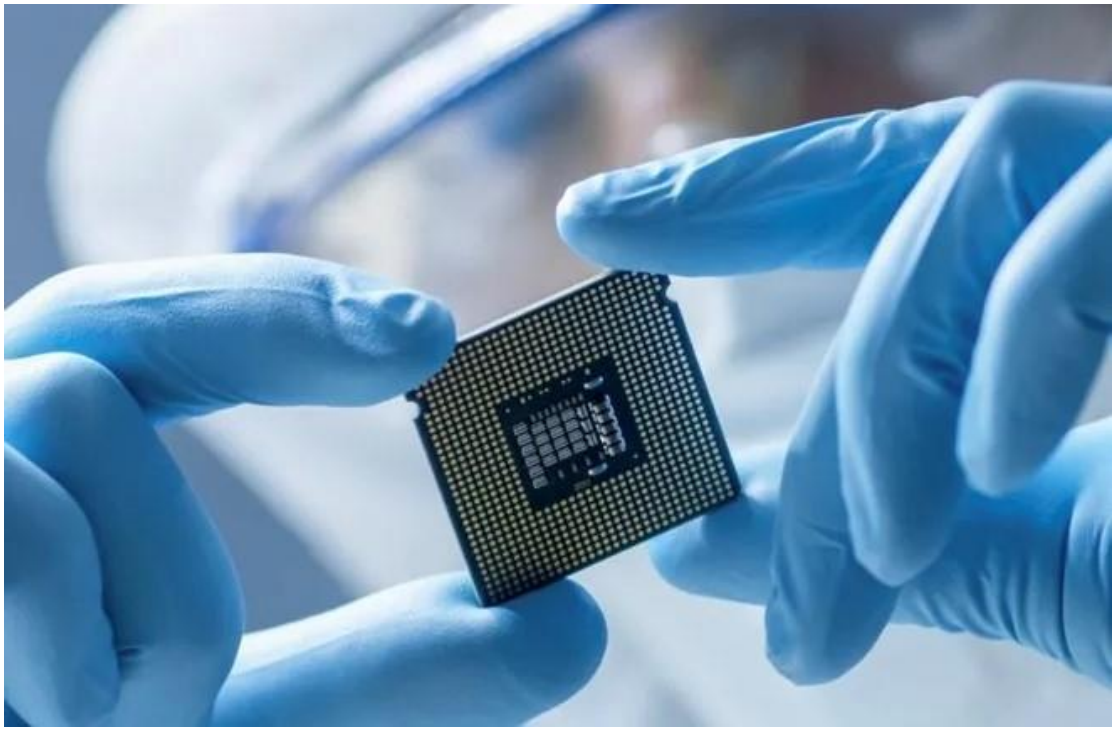


數位系統設計作業

HW11

學號 : 01257027| 姓名 : 林承羿

實作想法：

首先，此項作業沒有跟任何數值輸出有關，要求只有依照指令的模式讓 LED 燈以固定模式閃爍，故以下照片不會出現任何跟數值計算有關的 code，本人為方便聚焦重點，只提供與老師不同的部分，及自己更改的部分。

第一題

程式碼

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

entity Control is
    port(
            Inst31_26:in std_logic_vector(5 downto 0);
            SWImport:out std_logic;
            Jump:out std_logic;
            Halt:out std_logic;
            RegImport:out std_logic;
            RegOutport:out std_logic;
            RegDst:out std_logic;
            Branch:out std_logic;
            MemRead:out std_logic;
            MemtoReg:out std_logic;
            ALUOp:out std_logic_vector(1 downto 0);
            MemWrite:out std_logic;
            ALUSrc:out std_logic;
            RegWrite:out std_logic;
            OUTLED:out std_logic                        -- LED輸出
        );
end Control;
architecture Control of Control is
    signal tmp:std_logic_vector(14 downto 0);           -- 增加一個LED控制
begin

    tmp <= "000000100100010" when inst31_26 = 0 else    --R
           "000000011110000" when inst31_26 = 35 else   --lw
            "000000010001000" when inst31_26 = 43 else   --sw
            "000000000000101" when inst31_26 = 4 else   --beq
            "000010000100011" when inst31_26 = 63 else    --RegImport
            "000001000000011" when inst31_26 = 62 else --RegOutportout
            "000100000000011" when inst31_26 = 61 else --Halt
            "001000000000011" when inst31_26 = 2 else   --Jump
            "010000000100011" when inst31_26 = 60 else   --SWImport
            "100000000000000" when inst31_26 = 26 else   -- LED 控制 26
            "000000000000000";
```

```vhdl

	OUTLED <= tmp(14);                                    -- 控制LED
	SWimport <= tmp(13);
	Jump <= tmp(12);
	Halt <= tmp(11);
	RegImport <= tmp(10);
	RegOutport <= tmp(9);

	RegDst <= tmp(8);
	ALUSrc <= tmp(7);
	MemtoReg <= tmp(6);
	RegWrite <= tmp(5);
	MemRead <= tmp(4);
	MemWrite <= tmp(3);
	Branch <= tmp(2);
	ALUOp <= tmp(1 downto 0);

end Control;
```

可以看到其中添加了 OUTLED，作為題目要求的控制信號。

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

entity Instruction_memory is
	port(
			Reset:in std_logic;
			Read_address:in integer range 0 to 63;
			Instruction:out std_logic_vector(31 downto 0)
		);
end Instruction_memory;
architecture Instruction_memory of Instruction_memory is
	type mem_array is array (0 to 63)of std_logic_vector( 31 downto 0 );
	signal memory : mem_array := (
								x"68000000",x"08000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
begin

			Instruction <= memory(Read_address);

			process(Reset)
			begin
				if Reset = '0' then
					memory(0) <= x"68000000";  -- 令LED亮起
					memory(1) <= x"08000000";  -- 跳回第一個指令(LED暗)

					for i in 2 to 63 loop
						memory(i) <= (others => '0');
					end loop;
				end if;
			end process;
```

更改指令，此任務只是讓 LED 閃爍，並不涉及任何來源、目的暫存器更改，不需要存儲任何暫存器，故只需要 31~26 的位置設定為 26(本人設定 OUTLED 的 opcode)即可，兩個指令分別是 LED 亮，LED 暗，除了亮以外做其他事情都是暗的，故可以讓跳回第一個指令同時執行 LED 暗的結果。

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

entity MIPS is
    port(
            clk:in std_logic;
            SWin:in std_logic_vector(9 downto 0);
            Reset:in std_logic;
            PCout:out std_logic_vector(5 downto 0);
            DP:out std_logic_vector(3 downto 0);
            Hex0,Hex1,Hex2,Hex3:out std_logic_vector(6 downto 0);
            OUTLEDone:out std_logic
        );
end MIPS;
architecture MIPS of MIPS is

            component Registers is
                port(
                        clk:in std_logic;
                        Reset:in std_logic;
                        RegWrite:in std_logic;
                        Read_register1:in integer range 0 to 31;
                        Read_register2:in integer range 0 to 31;
                        Write_register:in integer range 0 to 31;
                        Write_data:in std_logic_vector(31 downto 0);
                        Read_data1:out std_logic_vector(31 downto 0);
                        Read_data2:out std_logic_vector(31 downto 0)
                    );
            end component;
```

```
    component ALU is
        port(
                A:in std_logic_vector(31 downto 0);
                B:in std_logic_vector(31 downto 0);
                Operation:in std_logic_vector(3 downto 0);
                ALUresult:out std_logic_vector(31 downto 0);
                Zero:out std_logic

                );
    end component;

    component Data_memory is
        port(
                clk:in std_logic;
                Address:in integer range 0 to 63;
                Write_data:in std_logic_vector(31 downto 0);
                MemWrite:in std_logic;
                MemRead:in std_logic;
                Read_data:out std_logic_vector(31 downto 0)
            );
    end component;

    component Instruction_memory is
        port(
                Reset:in std_logic;
                Read_address:in integer range 0 to 63;
                Instruction:out std_logic_vector(31 downto 0)
            );
    end component;

    component Sign_extend is
        port(
                Inst15_0:in std_logic_vector(15 downto 0);
                Extendout:out std_logic_vector(31 downto 0)
            );
    end component;
```

```
70          component Control is
71              port(
72                      Inst31_26:in std_logic_vector(5 downto 0);
73                      SWImport:out std_logic;
74                      Jump:out std_logic;
75                      Halt:out std_logic;
76                      RegImport:out std_logic;
77                      RegOutport:out std_logic;
78                      RegDst:out std_logic;
79                      Branch:out std_logic;
80                      MemRead:out std_logic;
81                      MemtoReg:out std_logic;
82                      ALUOp:out std_logic_vector(1 downto 0);
83                      MemWrite:out std_logic;
84                      ALUSrc:out std_logic;
85                      RegWrite:out std_logic;
86                      OUTLED:out std_logic
87                  );
88          end component;
89
90          component ALUControl is
91              port(
92                      ALUOp:in std_logic_vector(1 downto 0);
93                      Inst5_0:in std_logic_vector(5 downto 0);
94                      Operation:out std_logic_vector(3 downto 0)
95                  );
96          end component;
97
98          component Add_32bit is
99              port(
100                     A:in std_logic_vector(31 downto 0);
101                     B:in std_logic_vector(31 downto 0);
102                     Result:out std_logic_vector(31 downto 0)
103                 );
104         end component;
```

```
105
106         component digits_large is
107         port(
108                 BIN:in integer range 0 to 99999999;
109                 num7: out integer range 0 to 9;
110                 num6: out integer range 0 to 9;
111                 num5: out integer range 0 to 9;
112                 num4: out integer range 0 to 9;
113                 num3: out integer range 0 to 9;
114                 num2: out integer range 0 to 9;
115                 num1: out integer range 0 to 9;
116                 num0: out integer range 0 to 9
117                 );
118
119         end component;
120
121         component decoder_7seg is
122             PORT(
123                     BCD:in std_logic_vector(3 downto 0);
124                     HEX:out std_logic_vector(6 downto 0)
125                     );
126         end component;
127
128
129     type INT_array is Array (0 to 3) of integer range 0 to 9;
130     type INT_array_large is Array (0 to 7) of integer range 0 to 9;
131     type LOGIC_array is Array (0 to 3) of std_logic_vector(3 downto 0);
132     signal num1,num2: INT_array;
133     signal num: INT_array_large;
134     signal P:std_logic_vector(19 downto 0);
135     signal show:LOGIC_array;
136     signal cnt:std_logic_vector(25 downto 0);
```

```
        signal Instruction:std_logic_vector(31 downto 0);
        signal Extendout:std_logic_vector(31 downto 0);
        signal Read_data1:std_logic_vector(31 downto 0);
        signal Read_data2:std_logic_vector(31 downto 0);

        signal ALUResult:std_logic_vector(31 downto 0);
        signal ALU_B:std_logic_vector(31 downto 0);
        signal Operation:std_logic_vector(3 downto 0);
        signal Zero:std_logic;


        signal Read_mem_data:std_logic_vector(31 downto 0);
        signal PC:std_logic_vector(31 downto 0);
        signal SWImport,Jump,Halt,RegImport,RegOutport:std_logic;
        signal Branch,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite, OUTLED:std_logic;
        signal ALUop:std_logic_vector(1 downto 0);

        signal Read_register1,Read_register2,Write_register:integer range 0 to 31;
        signal Write_data:std_logic_vector(31 downto 0);

        signal Add_32bit_A:std_logic_vector(31 downto 0);
        signal Add_32bit_B:std_logic_vector(31 downto 0);
        signal AddResult:std_logic_vector(31 downto 0);
        signal Output:std_logic_vector(31 downto 0);

        signal delay:integer range 0 to 50000;
```

```
begin

--Program Counter
    process(clk,reset)
    begin
        if reset = '0' then
            PC <= (others => '0');
            delay <= 0;
            OUTLEDone <= '1';
        elsif clk'event and clk = '1' then
            if Branch = '1' and Zero = '1' then
                PC <= AddResult;

            -- elsif Jump = '1' and delay = 24999 then --直到暫停的時間結束，PC指到下一個指令
            elsif Jump = '1' and delay = 1 then
            --else
                delay <= 0;
                OUTLEDone <= '1';                   -- 亮
                PC <= PC(31 downto 28)&Instruction(25 downto 0)&"00";
            elsif Halt = '0' then
            --else
                delay <= delay + 1;                 -- 延遲所有過程，讓PC暫停
            end if;

            -- if OUTLED = '1' and delay = 49999 then --直到暫停的時間結束，PC指到下一個指令
            if OUTLED = '1' and delay = 3 then
                delay <= 0;
                OUTLEDone <= '0';                   -- 暗
                PC <= PC +4;
            end if;
        end if;
    end process;

--Instruction Memory

Inst_Memory_part:
Instruction_memory port map (Reset,conv_integer(PC(7 downto 2)),Instruction);
```

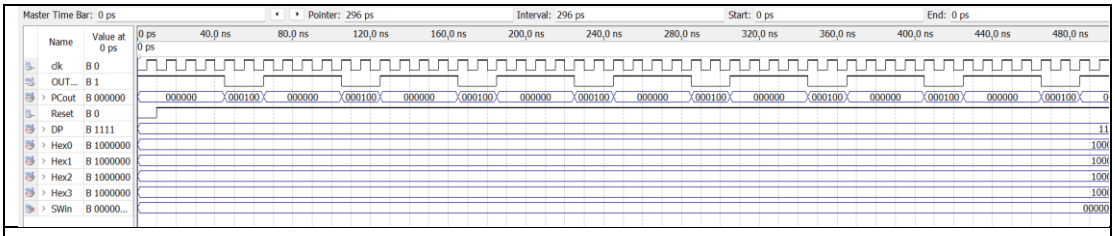
```
203
204
205   --Registers
206   Read_register1 <= conv_integer(Instruction(20 downto 16)) when RegOutport = '1' else
207                     conv_integer(Instruction(25 downto 21));
208
209   Read_register2 <= conv_integer(Instruction(20 downto 16));
210
211   Write_register <= conv_integer(Instruction(20 downto 16)) when RegImport = '1' else  --MUX
212                     conv_integer(Instruction(20 downto 16)) when SWImport = '1' else  --MUX SWIN
213                     conv_integer(Instruction(20 downto 16)) when RegDst = '0' else  --MUX
214                     conv_integer(Instruction(15 downto 11));
215
216   Write_data <=  x"0000"&Instruction(15 downto 0) when RegImport = '1' else
217                  x"0000"&"000000"&SWin when SWImport = '1' else   --"00000000000000000000"
218                  ALUResult when MemtoReg = '0' else
219                  Read_mem_data;
220
221
222
223   Register_part:
224   Registers port map(clk,Reset,RegWrite,Read_register1,Read_register2,Write_register,
225                      Write_data,Read_data1,Read_data2);
226
227
228   --ALU
229   ALU_B <= Read_data2 when ALUSrc = '0' else
230            Extendout;
231
232   ALU_part:
233   ALU port map (Read_data1,ALU_B,Operation,ALUresult,Zero);
234
235
236   --Data Memory
237
238   Data_mem:
239   Data_memory port map (clk,conv_integer(ALUResult(7 downto 2)),Read_data2,MemWrite,MemRead,Read_mem_data);
240
```

```
241   --Control
242
243   Control_part:
244   Control port map (Instruction(31 downto 26),SWImport,Jump,Halt,RegImport,RegOutport,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite, OUTLED);
245
246   --ALUControl
247
248   ALUControl_part:
249   ALUControl port map (ALUOp,Instruction(5 downto 0),Operation);
250
251
252   --Sign_extend
253
254   Sign_extend_part:
255   Sign_extend port map (Instruction(15 downto 0),Extendout);
256
257
258   --Add_32bit
259   Add_32bit_A <= PC + 4;
260   Add_32bit_B <= Extendout(29 downto 0) & "00";
261
262   Add_32bit_part:
263   Add_32bit port map (Add_32bit_A,Add_32bit_B,AddResult);
264
265   PCout <= PC(5 downto 0);
266
267
268
269   ----------------------------------------------------------------------------
270   process(clk,Reset)
271   begin
272       if Reset = '0' then
273           output <= x"00000000";
274       elsif clk'event and clk = '0' then  --fall edge
275           if RegOutport = '1' then
276             Output <= Read_data1;
277           end if;
278       end if;
279   end process;
```

```
281  ----------------------------------------------------------------------------------
282
283              process(clk)
284              begin
285                  if clk'event and clk = '1' then
286                      cnt <= cnt + 1;
287                  end if;
288              end process;
289
290
291  U1:digits_large port map(Conv_integer(Output),num(7),num(6),num(5),num(4),num(3),num(2),num(1),num(0));
292
293
294  show(0) <= conv_std_logic_vector(num(0),4)  when  cnt(25) = '0'  else
295                conv_std_logic_vector(num(4),4);
296
297  show(1) <= conv_std_logic_vector(num(1),4)  when  cnt(25) = '0' else
298                conv_std_logic_vector(num(5),4);
299
300  show(2) <= conv_std_logic_vector(num(2),4)  when  cnt(25) = '0'  else
301                conv_std_logic_vector(num(6),4);
302
303  show(3) <= conv_std_logic_vector(num(3),4)  when  cnt(25) = '0'  else
304                conv_std_logic_vector(num(7),4);
305
306
307
308
309  HEX0_part:decoder_7seg port map(show(0),HEX0);
310  HEX1_part:decoder_7seg port map(show(1),HEX1);
311  HEX2_part:decoder_7seg port map(show(2),HEX2);
312  HEX3_part:decoder_7seg port map(show(3),HEX3);
313
314  DP<= "0000" when cnt(25) = '1' else
315       "1111";
316
317  end MIPS;
```

主要更改的部分是 Program Counter 部分，目的是為了延長 PC 被執行的時間，故不可以讓每次時脈執行時都 PC+4，而是讓一個 delay 信號計數到指定時間再執行。至於 for 迴圈沒有實作的理由是，迴圈一定要做到與時脈相關才會延遲計數否則迴圈跑的步數再多也很快就執行完了，又這行為應該是每次時脈觸發都要做，故捨棄 for 迴圈。

可以看到其中註解了兩行延長為要求時間的 code，是為了呈現結果才這麼做。



為了清楚看到確實有延長，而非照時脈，故意設定最小的延長不與時脈同步。而結果也照著要求按比例縮小，從結果圖也可以觀察到確實指令只有執行 0、4 循環，符合要求，且 OUTLED 的信號也確實影響著結果。

加分題

程式碼

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

entity Instruction_memory is
	port(
			Reset:in std_logic;
			Read_address:in integer range 0 to 63;
			Instruction:out std_logic_vector(31 downto 0)
		);
end Instruction_memory;
architecture Instruction_memory of Instruction_memory is
	type mem_array is array (0 to 63)of std_logic_vector( 31 downto 0 );
	signal memory : mem_array := (
								x"68000000",x"68000000",x"08000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
								x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
begin

			Instruction <= memory(Read_address);

			process(Reset)
			begin
				if Reset = '0' then
					memory(0) <= x"68000000";  -- 令第一個LED亮起
					memory(1) <= x"68000000";  -- 令第二個LED亮起
					memory(2) <= x"08000000";  -- 令第三個LED亮起，並跳回第一個指令

					for i in 3 to 63 loop
						memory(i) <= (others => '0');
					end loop;
				end if;
			end process;

```

指令只有三個，與基本題不同的是，沒有 LED 暗的，只要設定每次亮的位置不同即可，故第一個指令亮最左邊，第二個亮中間，最後一個亮最右邊，並跳回第一個指令。

```vhdl

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

entity MIPS is
    port(
            clk:in std_logic;
            SWin:in std_logic_vector(9 downto 0);
            Reset:in std_logic;
            PCout:out std_logic_vector(5 downto 0);
            DP:out std_logic_vector(3 downto 0);
            Hex0,Hex1,Hex2,Hex3:out std_logic_vector(6 downto 0);
            OUTLED:out std_logic_vector(2 downto 0)
        );
end MIPS;
architecture MIPS of MIPS is

            component Registers is
                port(
                        clk:in std_logic;
                        Reset:in std_logic;
                        RegWrite:in std_logic;
                        Read_register1:in integer range 0 to 31;
                        Read_register2:in integer range 0 to 31;
                        Write_register:in integer range 0 to 31;
                        Write_data:in std_logic_vector(31 downto 0);
                        Read_data1:out std_logic_vector(31 downto 0);
                        Read_data2:out std_logic_vector(31 downto 0)
                    );
            end component;

```

```
34          component ALU is
35              port(
36                      A:in std_logic_vector(31 downto 0);
37                      B:in std_logic_vector(31 downto 0);
38                      Operation:in std_logic_vector(3 downto 0);
39                      ALUresult:out std_logic_vector(31 downto 0);
40                      Zero:out std_logic
41
42                      );
43          end component;
44
45          component Data_memory is
46              port(
47                      clk:in std_logic;
48                      Address:in integer range 0 to 63;
49                      Write_data:in std_logic_vector(31 downto 0);
50                      MemWrite:in std_logic;
51                      MemRead:in std_logic;
52                      Read_data:out std_logic_vector(31 downto 0)
53                  );
54          end component;
55
56          component Instruction_memory is
57              port(
58                      Reset:in std_logic;
59                      Read_address:in integer range 0 to 63;
60                      Instruction:out std_logic_vector(31 downto 0)
61                  );
62          end component;
63
64          component Sign_extend is
65              port(
66                      Inst15_0:in std_logic_vector(15 downto 0);
67                      Extendout:out std_logic_vector(31 downto 0)
68                  );
69          end component;
70
```

```
71          component Control is
72              port(
73                      Inst31_26:in std_logic_vector(5 downto 0);
74                      SWImport:out std_logic;
75                      Jump:out std_logic;
76                      Halt:out std_logic;
77                      RegImport:out std_logic;
78                      RegOutport:out std_logic;
79                      RegDst:out std_logic;
80                      Branch:out std_logic;
81                      MemRead:out std_logic;
82                      MemtoReg:out std_logic;
83                      ALUOp:out std_logic_vector(1 downto 0);
84                      MemWrite:out std_logic;
85                      ALUSrc:out std_logic;
86                      RegWrite:out std_logic;
87                      OUTLED:out std_logic
88                  );
89          end component;
90
91          component ALUControl is
92              port(
93                      ALUOp:in std_logic_vector(1 downto 0);
94                      Inst5_0:in std_logic_vector(5 downto 0);
95                      Operation:out std_logic_vector(3 downto 0)
96                  );
97          end component;
98
99          component Add_32bit is
100             port(
101                     A:in std_logic_vector(31 downto 0);
102                     B:in std_logic_vector(31 downto 0);
103                     Result:out std_logic_vector(31 downto 0)
104                 );
105         end component;
106
```

```
107             component digits_large is
108             port(
109                     BIN:in integer range 0 to 99999999;
110                     num7: out integer range 0 to 9;
111                     num6: out integer range 0 to 9;
112                     num5: out integer range 0 to 9;
113                     num4: out integer range 0 to 9;
114                     num3: out integer range 0 to 9;
115                     num2: out integer range 0 to 9;
116                     num1: out integer range 0 to 9;
117                     num0: out integer range 0 to 9
118                     );
119
120             end component;
121
122             component decoder_7seg is
123                 PORT(
124                         BCD:in std_logic_vector(3 downto 0);
125                         HEX:out std_logic_vector(6 downto 0)
126                         );
127             end component;
128
129
130         type INT_array is Array (0 to 3) of integer range 0 to 9;
131         type INT_array_large is Array (0 to 7) of integer range 0 to 9;
132         type LOGIC_array is Array (0 to 3) of std_logic_vector(3 downto 0);
133         signal num1,num2: INT_array;
134         signal num: INT_array_large;
135         signal P:std_logic_vector(19 downto 0);
136         signal show:LOGIC_array;
137         signal cnt:std_logic_vector(25 downto 0);
138
139         signal Instruction:std_logic_vector(31 downto 0);
140         signal Extendout:std_logic_vector(31 downto 0);
141         signal Read_data1:std_logic_vector(31 downto 0);
142         signal Read_data2:std_logic_vector(31 downto 0);
143
```

```
144         signal ALUResult:std_logic_vector(31 downto 0);
145         signal ALU_B:std_logic_vector(31 downto 0);
146         signal Operation:std_logic_vector(3 downto 0);
147         signal Zero:std_logic;
148
149
150         signal Read_mem_data:std_logic_vector(31 downto 0);
151         signal PC:std_logic_vector(31 downto 0);
152         signal SWImport,Jump,Halt,RegImport,RegOutport:std_logic;
153         signal Branch,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite, OUTLEDsig:std_logic;
154         signal ALUop:std_logic_vector(1 downto 0);
155
156         signal Read_register1,Read_register2,Write_register:integer range 0 to 31;
157         signal Write_data:std_logic_vector(31 downto 0);
158
159         signal Add_32bit_A:std_logic_vector(31 downto 0);
160         signal Add_32bit_B:std_logic_vector(31 downto 0);
161         signal AddResult:std_logic_vector(31 downto 0);
162         signal Output:std_logic_vector(31 downto 0);
163
164         signal delay:integer range 0 to 150000;
165         signal flag0, flag1, flag2:std_logic := '0';
166
```

```
167 begin
168 --Program Counter
169     process(clk,reset)
170     begin
171         if reset = '0' then
172             PC <= (others => '0');
173             delay <= 0;
174             flag2 <= '1';
175             OUTLED(2) <= '1';                           -- 一開始亮最左一個LED
176         elsif clk'event and clk = '1' then
177             if Branch = '1' and Zero = '1' then
178                 PC <= AddResult;
179
180             -- elsif Jump = '1' and flag2 = '1' and delay = 24999 then --直到暫停的時間結束，PC指到下一個指令
181             elsif Jump = '1' and flag0 = '1' and delay = 1 then
182             --else
183                 delay <= 0;                             -- 歸零重新計算
184                 flag0 <= '0';                           -- 最右不亮
185                 flag2 <= '1';                           -- 最左亮起
186                 OUTLED(2) <= '1';                       -- 開啟最左
187                 OUTLED(0) <= '0';                       -- 關掉最右
188                 PC <= PC(31 downto 28)&Instruction(25 downto 0)&"00";
189             elsif Halt = '0' then
190             --else
191                 delay <= delay + 1;                     -- 延遲所有過程，讓PC暫停
192             end if;
193
194             -- if OUTLEDsig = '1' and flag0 = '1' and delay = 149999 then   --直到暫停的時間結束，PC指到下一個指令
195             if OUTLEDsig = '1' and flag2 = '1' and delay = 11 then
196                 delay <= 0;
197                 OUTLED(2) <= '0';                       -- 關掉最左
198                 OUTLED(1) <= '1';                       -- 開啟中間
199                 flag2 <= '0';                           -- 最左不亮
200                 flag1 <= '1';                           -- 中間亮起
201                 PC <= PC +4;
202             -- elsif OUTLEDsig = '1' and flag1 = '1' and delay = 149999 then --直到暫停的時間結束，PC指到下一個指令
203             elsif OUTLEDsig = '1' and flag1 = '1' and delay = 11 then
204                 delay <= 0;
205                 OUTLED(1) <= '0';                       -- 關掉中間
206                 OUTLED(0) <= '1';                       -- 開啟最右
207                 flag1 <= '0';                           -- 中間不亮
208                 flag0 <= '1';                           -- 最右亮起
209                 PC <= PC +4;
210             end if;
211         end if;
212     end process;
213
```

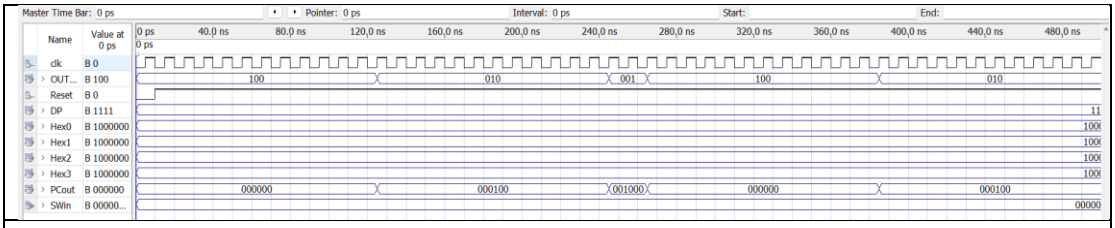
```
215  --Instruction Memory
216
217  Inst_Memory_part:
218  Instruction_memory port map (Reset,conv_integer(PC(7 downto 2)),Instruction);
219
220
221
222  --Registers
223  Read_register1 <= conv_integer(Instruction(20 downto 16)) when RegOutport = '1' else
224                    conv_integer(Instruction(25 downto 21));
225
226  Read_register2 <= conv_integer(Instruction(20 downto 16));
227
228  Write_register <= conv_integer(Instruction(20 downto 16)) when RegImport = '1' else  --MUX
229                    conv_integer(Instruction(20 downto 16)) when SWImport = '1' else  --MUX SWIN
230                    conv_integer(Instruction(20 downto 16)) when RegDst = '0' else  --MUX
231                    conv_integer(Instruction(15 downto 11));
232
233  Write_data <=  x"0000"&Instruction(15 downto 0) when RegImport = '1' else
234                 x"0000"&"000000"&SWin when SWImport = '1' else   --"00000000000000000000"
235                 ALUResult when MemtoReg = '0' else
236                 Read_mem_data;
237
238
239
240  Register_part:
241  Registers port map(clk,Reset,RegWrite,Read_register1,Read_register2,Write_register,
242                     Write_data,Read_data1,Read_data2);
243
244
245  --ALU
246  ALU_B <= Read_data2 when ALUSrc = '0' else
247           Extendout;
248
249  ALU_part:
250  ALU port map (Read_data1,ALU_B,Operation,ALUresult,Zero);
```

```
253  --Data Memory
254
255  Data_mem:
256  Data_memory port map (clk,conv_integer(ALUResult(7 downto 2)),Read_data2,MemWrite,MemRead,Read_mem_data);
257
258  --Control
259
260  Control_part:
261  Control port map (Instruction(31 downto 26),SWImport,Jump,Halt,RegImport,RegOutport,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite, OUTLEDsig);
262
263  --ALUControl
264
265  ALUControl_part:
266  ALUControl port map (ALUOp,Instruction(5 downto 0),Operation);
267
268
269  --Sign_extend
270
271  Sign_extend_part:
272  Sign_extend port map (Instruction(15 downto 0),Extendout);
273
274
275  --Add_32bit
276  Add_32bit_A <= PC + 4;
277  Add_32bit_B <= Extendout(29 downto 0) & "00";
278
279  Add_32bit_part:
280  Add_32bit port map (Add_32bit_A,Add_32bit_B,AddResult);
281
282  PCout <= PC(5 downto 0);
```

```
process(clk,Reset)
begin
    if Reset = '0' then
        output <= x"00000000";
    elsif clk'event and clk = '0' then  --fall edge
        if RegOutport = '1' then
          Output <= Read_data1;
        end if;
    end if;
end process;

            process(clk)
            begin
                if clk'event and clk = '1' then
                    cnt <= cnt + 1;
                end if;
            end process;


U1:digits_large port map(Conv_integer(Output),num(7),num(6),num(5),num(4),num(3),num(2),num(1),num(0));


show(0) <= conv_std_logic_vector(num(0),4)  when  cnt(25) = '0'  else
           conv_std_logic_vector(num(4),4);

show(1) <= conv_std_logic_vector(num(1),4)  when  cnt(25) = '0' else
           conv_std_logic_vector(num(5),4);

show(2) <= conv_std_logic_vector(num(2),4)  when  cnt(25) = '0'  else
           conv_std_logic_vector(num(6),4);

show(3) <= conv_std_logic_vector(num(3),4)  when  cnt(25) = '0'  else
           conv_std_logic_vector(num(7),4);
```

加分題改最多的依然是 Program Counter 的部分，opcode 一樣延用 26，那這指令只知道要亮，所以添加三個 flag 控制信號確認延長時間到了後該關掉哪個 LED，並使哪個 LED 點亮。

與基本題相同，為了呈現出結果的正確性，將要求的時間註解，並按比例縮小呈現出結果。



可以清楚看到圖中呈現出 LED 亮的相對位置，並看到 PC 確實延長了接收下一道指令的時間。從 PCout 中可以看到指令確實只有 0、4、8 再輪流執行，符合題目要求。

心得

透過此次的實習，我更了解到了如何運用 VHLD 使硬體部件照的我的意願行動，並知悉自己也是有能力設計出生活中常見例子的，如題目要求的紅綠燈，我真正了解到自己原來離生活沒有差距如此之大。