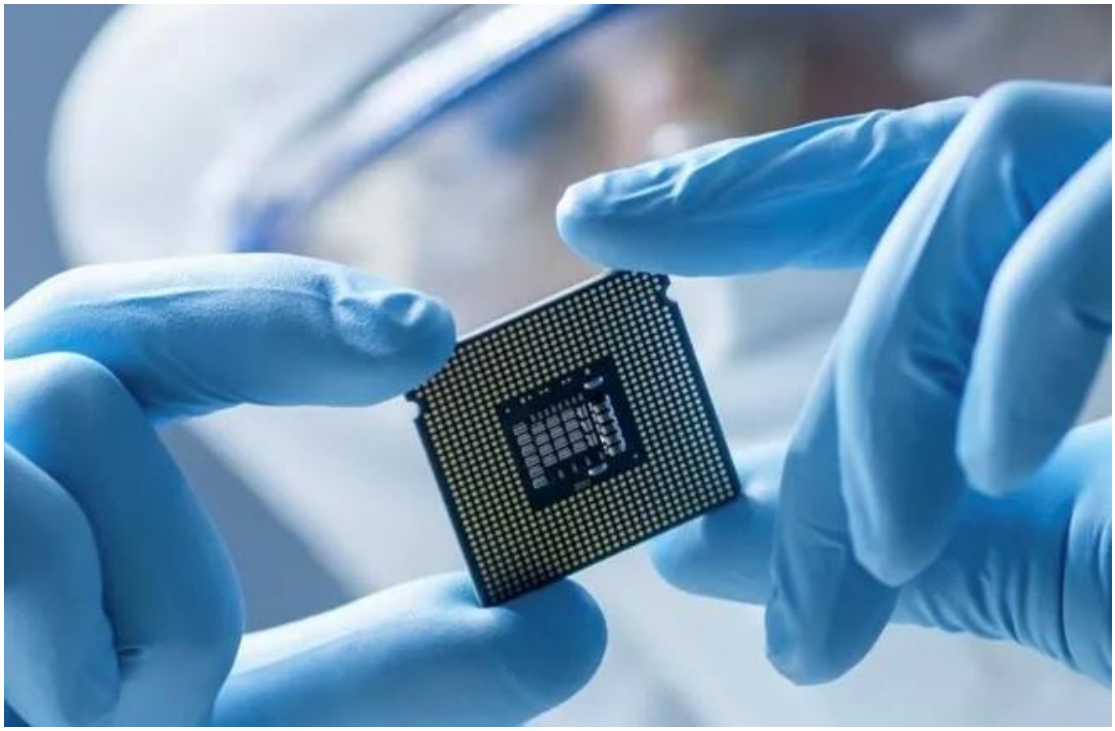


數位系統設計
期末專案

學號 ： 01257027 | 姓名 ： 林承羿

題目：串接計時器(2 台機器)

版本一，1 台機器(0~9999)

程式碼

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

entity upBCD_4bits is
    port(
        clk:in std_logic;                                   -- 時脈
        reset:in std_logic;                                 -- 重置
        stop_pos:in std_logic;                              -- 暫停計數
        casout:out std_logic;                               -- 進位
        Q:out std_logic_vector(3 downto 0)                  -- 結果
    );
end upBCD_4bits;

architecture upBCD_4bits of upBCD_4bits is
    signal cnt:std_logic_vector(3 downto 0):=(others => '0');
    signal flag:std_logic:='0';                             -- 進位(避免最開始的 0)
begin
    process(clk, reset)
    begin
        if(reset = '0') then
            cnt <= x"0";
            flag <= '0';
        elsif stop_pos = '0' then
            cnt <= cnt;
            flag <= flag;
        elsif rising_edge(clk) then
            if(cnt = x"9")then                              -- BCD 計數
                cnt <= x"0";
                flag <= '1';
            else
                cnt <= cnt + '1';
                flag <= '0';
            end if;
        end if;
    end process;
    Q <= cnt;
    process(cnt)
    begin
        if ((cnt = x"0") and (flag = '1')) then             -- 進位指示
            casout <= '1';
        else
            casout <= '0';
        end if;
    end process;
end upBCD_4bits;
```

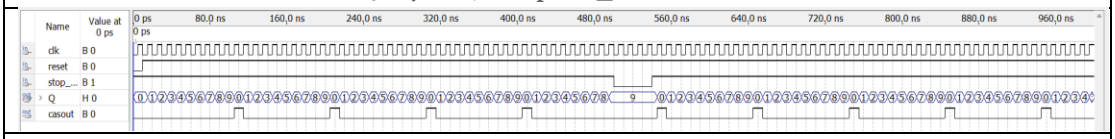
在版本一中，假定每個位數都是 10 進位，故沒有區分個位數、十位數。特別需要注意的是 flag 的運用，因為數到 0 會進位，那最初又從 0 開始數，會一開始就進位，為了排除此問題，設定在開始上數後才可以偵測是否需要進位。

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

entity upBCD_16bits is
    port(
        clk:in std_logic;
        reset:in std_logic;
        stop_pos:in std_logic;
        casout:out std_logic;
        -- look:out std_logic_vector(0 to 2);  -- 觀察進位的flag
        Q:out std_logic_vector(15 downto 0)
    );
end entity;

architecture upBCD_16bits of upBCD_16bits is
    component upBCD_4bits is
    port(
        clk:in std_logic;
        reset:in std_logic;
        stop_pos:in std_logic;
        casout:out std_logic;
        Q:out std_logic_vector(3 downto 0)
    );
    end component;
    signal casout_in:std_logic_vector(0 to 2):=(others => '0');
begin
    upBCD0:upBCD_4bits port map(clk, reset, stop_pos, casout_in(0), Q(3 downto 0));
    upBCD1:upBCD_4bits port map(casout_in(0), reset, stop_pos, casout_in(1), Q(7 downto 4));
    upBCD2:upBCD_4bits port map(casout_in(1), reset, stop_pos, casout_in(2), Q(11 downto 8));
    upBCD3:upBCD_4bits port map(casout_in(2), reset, stop_pos, casout, Q(15 downto 12));

    -- look <= casout_in;
end upBCD_16bits;
```

上圖示串接四個 BCD 個位數計數器的程式碼，比較需要關注的點是本上的時脈來源是上一個進位。

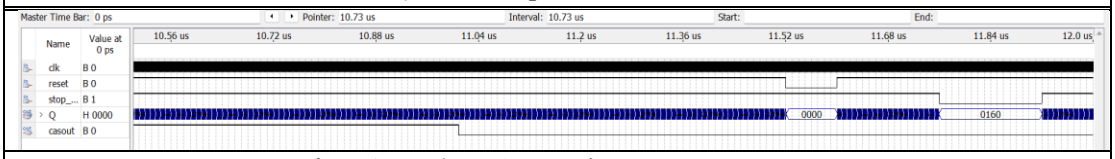
波形圖

參考檔案：upBCD_4bits.vwf

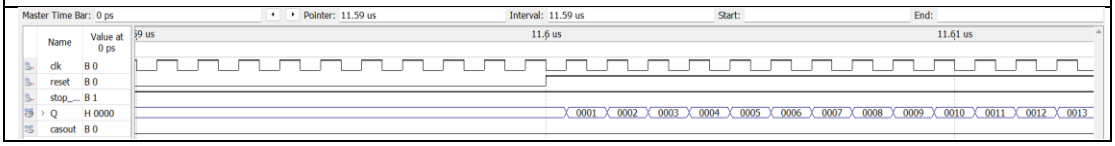


從上圖可以看到兩個重點，第一個是有正確的輸出可以進位的位置，且運作合理。第二個是有正確的歸零、暫停。

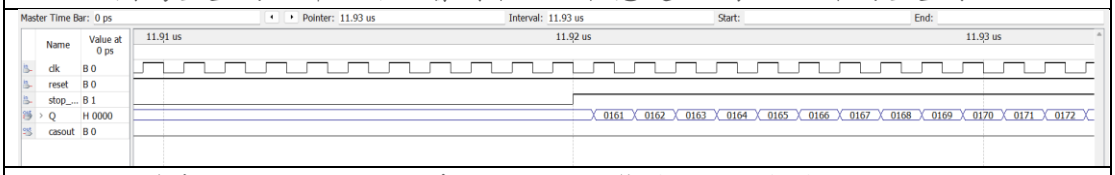
參考檔案：upBCD_16bits.vwf



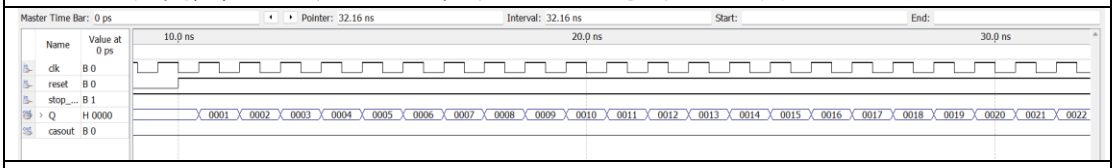
從上圖大致可以看到有正確的重置、暫停。



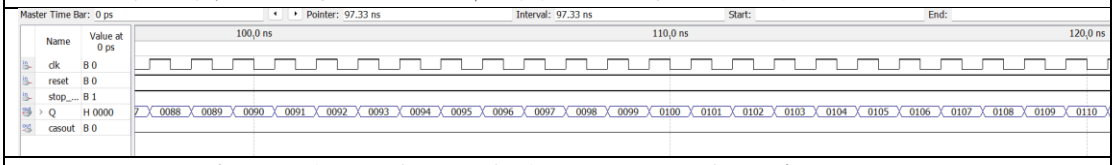
上圖為重置的細部。可以看到下一個狀態是1，表示正確的重置了。



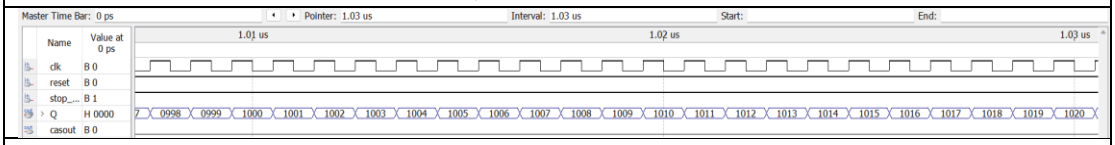
上圖為暫停的細部。可以看到下一個狀態是161，與重置不同。



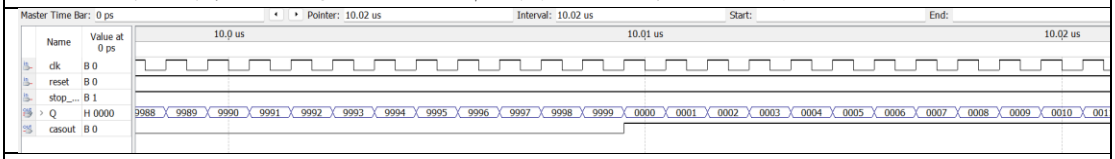
上圖主要表示的是個位數的計數與進位至十位數正確。



上圖主要表示的是十位數的計數與進位至百位數正確。



上圖主要表示的是百位數的計數與進位至千位數正確。



上圖主要表示的是千位數的計數與溢位後正確歸零。

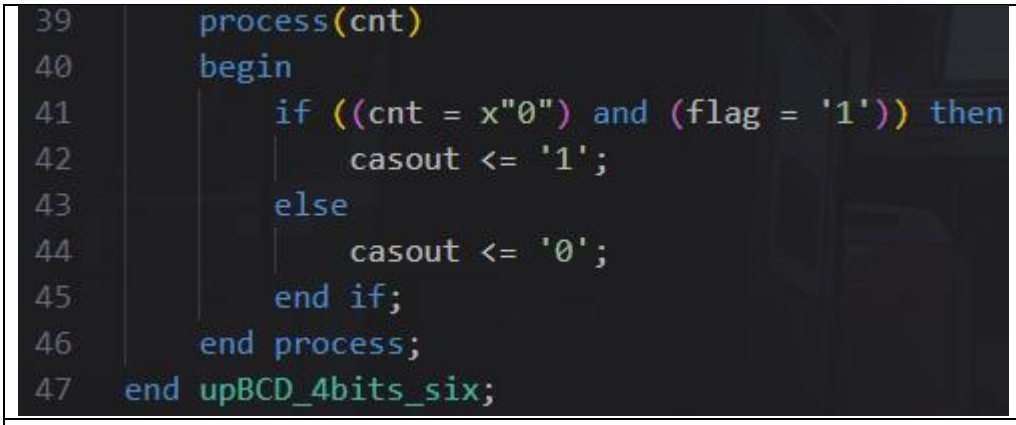
版本二，1 台機器(0~6060)

程式碼

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity upBCD_4bits_ten is
7      port(
8          clk:in std_logic;
9          reset:in std_logic;
10         stop_pos:in std_logic;
11         casout:out std_logic;
12         Q:out std_logic_vector(3 downto 0)
13     );
14 end upBCD_4bits_ten;
15
16 architecture upBCD_4bits_ten of upBCD_4bits_ten is
17     signal cnt:std_logic_vector(3 downto 0):=(others => '0');
18     signal flag:std_logic:='0';
19 begin
20     process(clk, reset)
21     begin
22         if(reset = '0') then
23             cnt <= x"0";
24             flag <= '0';
25         elsif stop_pos = '0' then
26             cnt <= cnt;
27             flag <= flag;
28         elsif rising_edge(clk) then
29             if(cnt = x"9")then
30                 cnt <= x"0";
31                 flag <= '1';
32             else
33                 cnt <= cnt + '1';
34                 flag <= '0';
35             end if;
36         end if;
37     end process;
38     Q <= cnt;
```

```
39      process(cnt)
40      begin
41          if ((cnt = x"0") and (flag = '1')) then
42              casout <= '1';
43          else
44              casout <= '0';
45          end if;
46      end process;
47  end upBCD_4bits_ten;
```

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

entity upBCD_4bits_six is
    port(
        clk:in std_logic;
        reset:in std_logic;
        stop_pos:in std_logic;
        casout:out std_logic;
        Q:out std_logic_vector(3 downto 0)
    );
end upBCD_4bits_six;

architecture upBCD_4bits_six of upBCD_4bits_six is
    signal cnt:std_logic_vector(3 downto 0):=(others => '0');
    signal flag:std_logic:='0';
begin
    process(clk, reset)
    begin
        if(reset = '0') then
            cnt <= x"0";
            flag <= '0';
        elsif stop_pos = '0' then
            cnt <= cnt;
            flag <= flag;
        elsif rising_edge(clk) then
            if(cnt = x"5")then      -- 時鐘的十位數
                cnt <= x"0";
                flag <= '1';
            else
                cnt <= cnt + '1';
                flag <= '0';
            end if;
        end if;
    end process;
    Q <= cnt;
```



```
39      process(cnt)
40      begin
41          if ((cnt = x"0") and (flag = '1')) then
42              casout <= '1';
43          else
44              casout <= '0';
45          end if;
46      end process;
47  end upBCD_4bits_six;
```

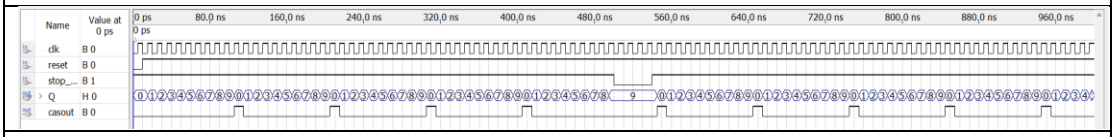
可以從上圖看到版本二為了區分十位數與個位數，分開寫了兩個檔案，最大的區別本人有使用註解標示出來，就單純是數到哪進位而已。

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

entity upBCD_16bits is
    port(
        clk:in std_logic;
        reset:in std_logic;
        stop_pos:in std_logic;
        casout:out std_logic;
        -- look:out std_logic_vector(0 to 2);  -- 觀察進位的flag
        Q:out std_logic_vector(15 downto 0)
    );
end entity;

architecture upBCD_16bits of upBCD_16bits is
    component upBCD_4bits_ten is
    port(
        clk:in std_logic;
        reset:in std_logic;
        stop_pos:in std_logic;
        casout:out std_logic;
        Q:out std_logic_vector(3 downto 0)
    );
    end component;

    component upBCD_4bits_six is
        port(
            clk:in std_logic;
            reset:in std_logic;
            stop_pos:in std_logic;
            casout:out std_logic;
            Q:out std_logic_vector(3 downto 0)
        );
    end component;

    signal casout_in:std_logic_vector(0 to 2):=(others => '0');
begin
    upBCD0:upBCD_4bits_ten port map(clk, reset, stop_pos, casout_in(0), Q(3 downto 0));
    upBCD1:upBCD_4bits_six port map(casout_in(0), reset, stop_pos, casout_in(1), Q(7 downto 4));
    upBCD2:upBCD_4bits_ten port map(casout_in(1), reset, stop_pos, casout_in(2), Q(11 downto 8));
    upBCD3:upBCD_4bits_six port map(casout_in(2), reset, stop_pos, casout, Q(15 downto 12));

    -- look <= casout_in;
end upBCD_16bits;
```

從以上可以看到與版本一的主要差別是多引入了十位數的計數，且在引用的方式上由上往下第二、四個都套用了十位數的計數規則，而非個位數的數到十進位。

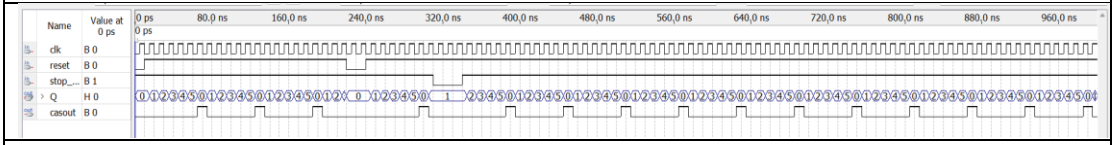
波形圖

參考檔案：upBCD_4bits_ten.vwf



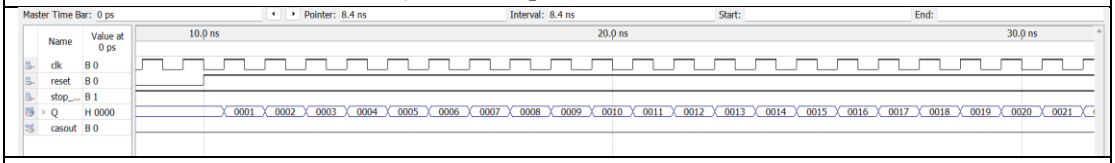
從上圖可以看到兩個重點，第一個是有正確的輸出可以進位的位置，且運作合理。第二個是有正確的歸零、暫停。

參考檔案：upBCD_4bits_six.vwf

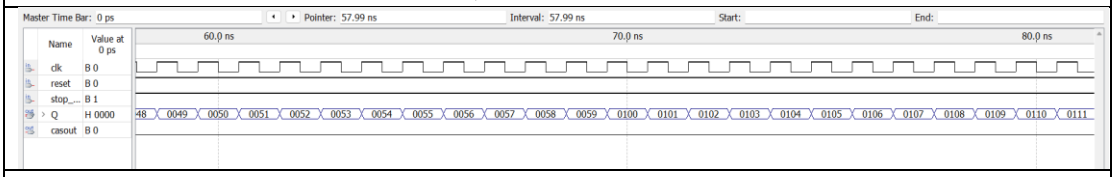


從上圖可以看到兩個重點，第一個是有正確的輸出可以進位的位置，且運作合理。第二個是有正確的歸零、暫停。

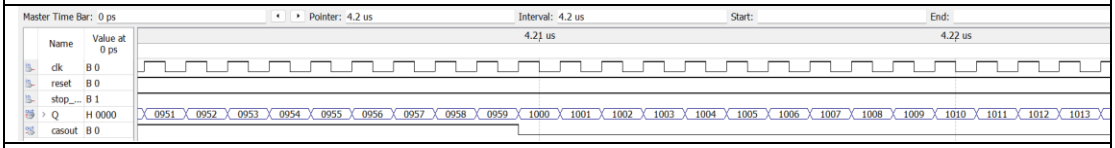
參考檔案：upBCD_16bits.vwf



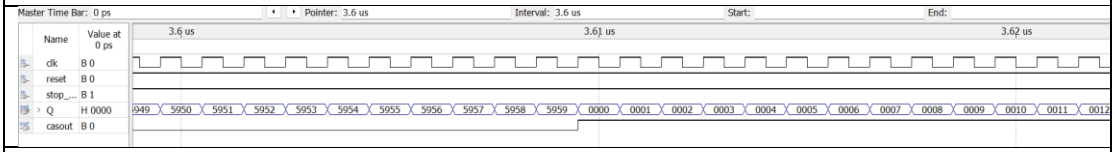
上圖主要表示的是個位數的計數與進位至十位數正確。



上圖主要表示的是十位數的計數與進位至百位數正確。值得注意的結果呈現確實反映出時鐘的運作方式，六十秒後進位變成一分鐘。



上圖主要表示的是百位數的計數與進位至千位數正確。



上圖主要表示的是千位數的計數與分鐘溢位後正確歸零。

共同程式碼

程式碼

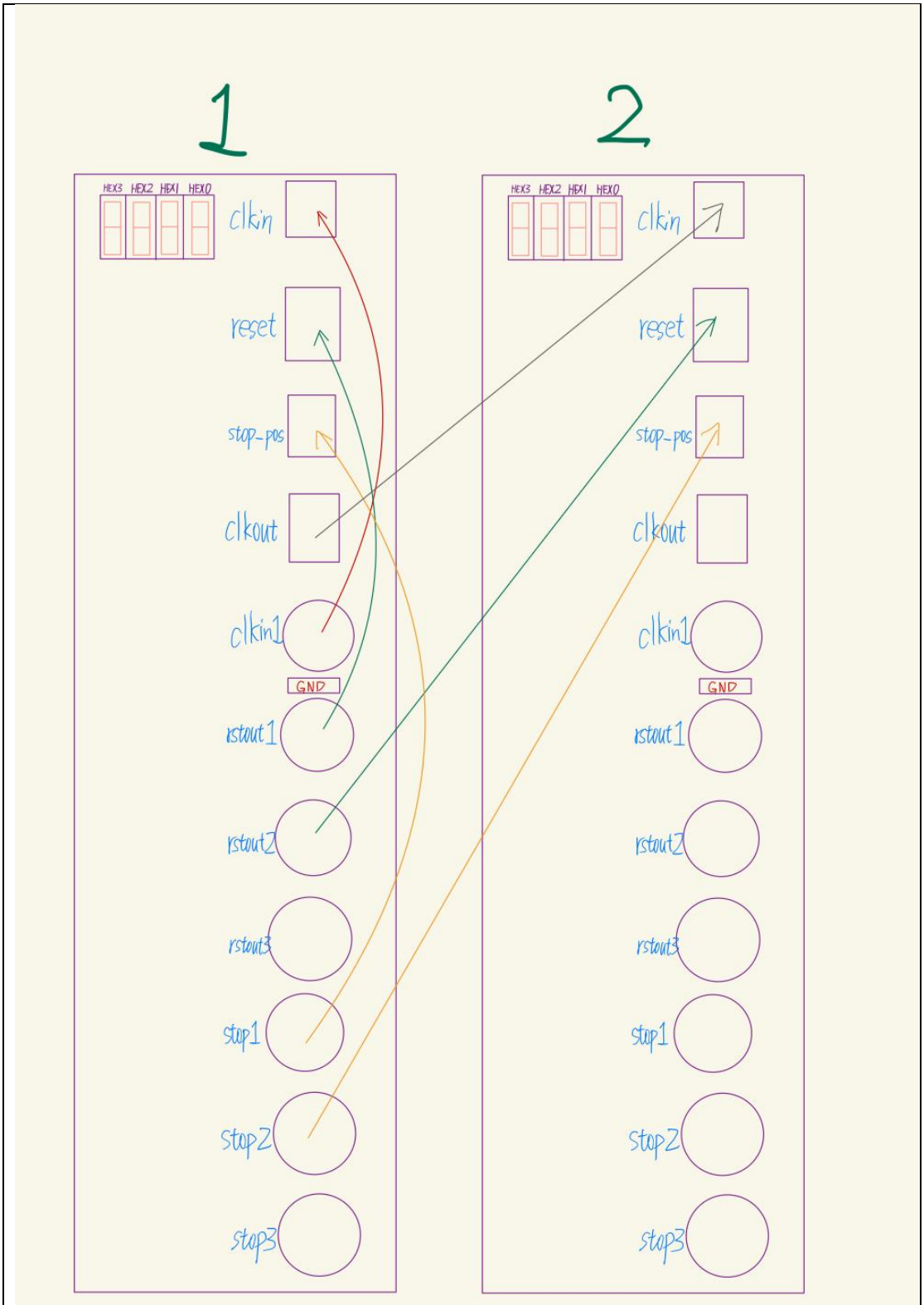
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity decoder_7seg is
    PORT(
            BCD:in std_logic_vector(3 downto 0);
            HEX:out std_logic_vector(6 downto 0)
            );
end decoder_7seg;

architecture decoder_7seg of decoder_7seg is
begin

    HEX <=  "1000000" when BCD = x"0" else
            "1111001" when BCD = x"1" else
            "0100100" when BCD = x"2" else
            "0110000" when BCD = x"3" else
            "0011001" when BCD = x"4" else
            "0010010" when BCD = x"5" else
            "0000010" when BCD = x"6" else
            "1111000" when BCD = x"7" else
            "0000000" when BCD = x"8" else
            "0010000" when BCD = x"9" else
            "1111111";
end decoder_7seg;
```

從 BCD 呼叫的方式中看到傳進來的是四位元，故以十六進制作為判斷依據後輸出對應的七段顯示器結果。

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


entity BCD is
    port(
        clk:in std_logic;               -- 綁定 50M 時脈
        bot:in std_logic;               -- 綁定 bot 0 按鈕(reset)
        botStop:in std_logic;           -- 綁定 bot 1 按鈕(stop)

        clkin:in std_logic;             -- 接收時脈(BCD counter)
        reset:in std_logic;             -- 兩塊板子接收 reset
        stop_pos:in std_logic;          -- 接收 stop 信號
        clkout:out std_logic;           -- 發出進位時脈
        clkin1:out std_logic;           -- 發出除頻結果，讓第一塊板子拉線接收
        stop1, stop2:out std_logic;     -- 接收 botStop，發出 stop 信號，拉線讓兩塊板子暫停
        rstout1, rstout2:out std_logic; -- 接收 bot，發出 reset 信號，拉線讓兩塊板子重置
        Hex0,Hex1,Hex2,Hex3:out std_logic_vector(6 downto 0)
    );
end entity;

architecture BCD of BCD is
    component upBCD_16bits is
        port(
            clk:in std_logic;
            reset:in std_logic;
            stop_pos:in std_logic;
            casout:out std_logic;
            Q:out std_logic_vector(15 downto 0)
        );
    end component;

    component decoder_7seg is
        PORT(
            BCD:in std_logic_vector(3 downto 0);
            HEX:out std_logic_vector(6 downto 0)
        );
    end component;
```

```
41      signal ans:std_logic_vector(15 downto 0);       -- BCD counter 結果
42      signal cnt:integer range 0 to 50000000:=0;      -- 除頻器(範圍 1 秒 -> 1/10000 秒)
43      signal clkCnt:std_logic:='0';                   -- 除頻後 clock
44      signal waste:std_logic:='0';                    -- stop 按下後，啥都不要做就是暫停
45  begin
46      rstout1 <= bot;
47      rstout2 <= bot;
48
49      stop1 <= botStop;
50      stop2 <= botStop;
51
52      process(clk)
53      begin
54          if(rising_edge(clk)) then
55              if (cnt < 4999) then
56                  cnt <= cnt + 1;
57                  clkCnt <= '0';
58              else
59                  clkCnt <= '1';
60                  cnt <= 0;
61              end if;
62          end if;
63      end process;
64      clkin1 <= clkCnt;                               -- 第一塊板子拉線接除頻結果
65                                                      -- 第二塊板子拉線接收第一塊板子的clkout
66
67      upBCD:upBCD_16bits port map(clkin, reset, stop_pos, clkout, ans);
68
69      -- 接七段顯示器
70      decoder0:decoder_7seg port map(ans(3 downto 0), Hex0);
71      decoder1:decoder_7seg port map(ans(7 downto 4), Hex1);
72      decoder2:decoder_7seg port map(ans(11 downto 8), Hex2);
73      decoder3:decoder_7seg port map(ans(15 downto 12), Hex3);
74  end BCD;
```

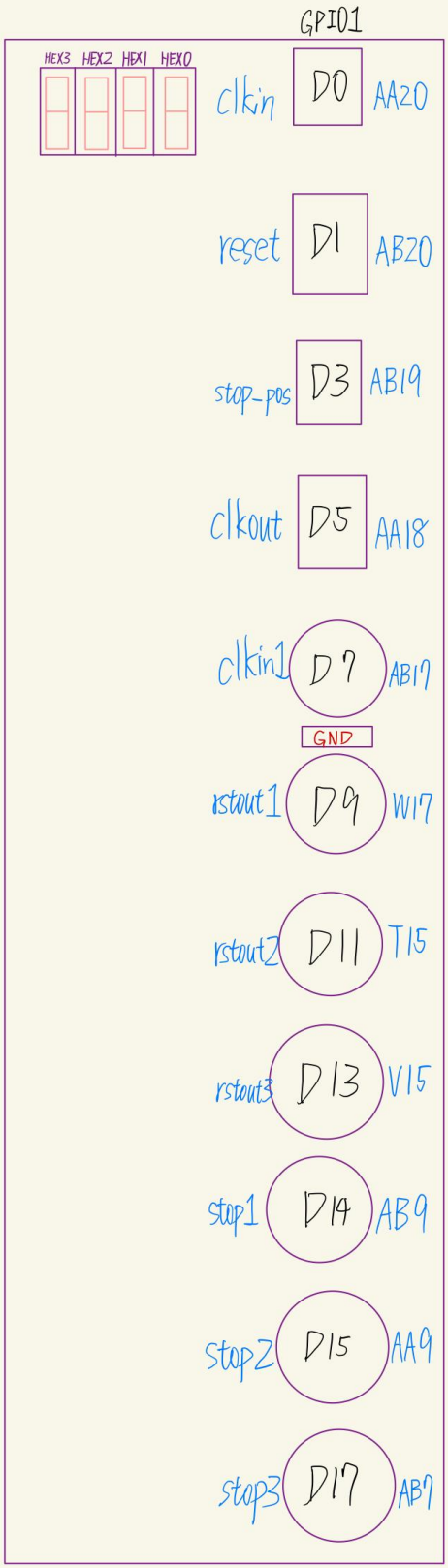
從上圖可以看到為了實現兩塊板子的串接，多設定了許多參數，細部功能及用義在上圖已用註解的方式表明。以下附上本人設計的構想圖。



圖中圓圈可以用寫死的方式來理解、而矩形則是可以想成一個模塊，可以接受任一時脈、重置來源、暫停來源。

此專案的題目主要敘述的是串接兩塊，第一快要是萬分之一秒，這操作需要 50M 的時脈來除頻，但如果在 clkin 中直接寫死，第二塊板子也會吃到萬分之一秒而非是第一塊進位的時脈，因此想法是把除頻結果輸出在 clkin1 中，第一塊板子用這輸出拉近時脈接口 clkin，第二塊也可以利用 clkout 拉進去 clkin，一切都是因為沒有把 clkin 當作時脈來源直接寫死。

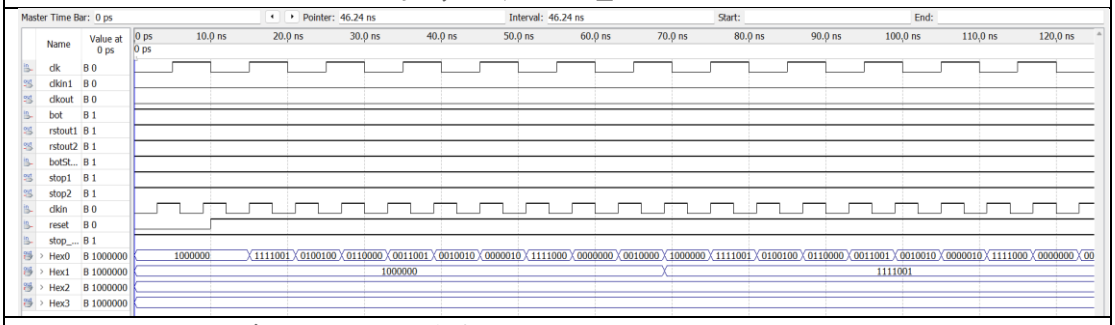
講完時脈，接下來講解 reset，理想是指按一個按鈕就可以同時重置兩塊板子，因此需要把一個信號放大成兩個，所以上圖可以看到假設我的 reset 按鈕是 bot，而 bot 輸出在 rstout1、rstout2、rstout3 中，這三個輸出接口都是相同的信號，等於同時有三個 reset 輸入，最多可提供三塊板子的串接。接下來暫停也與重置的操作相同，為了放大信號，做了相同的事情。



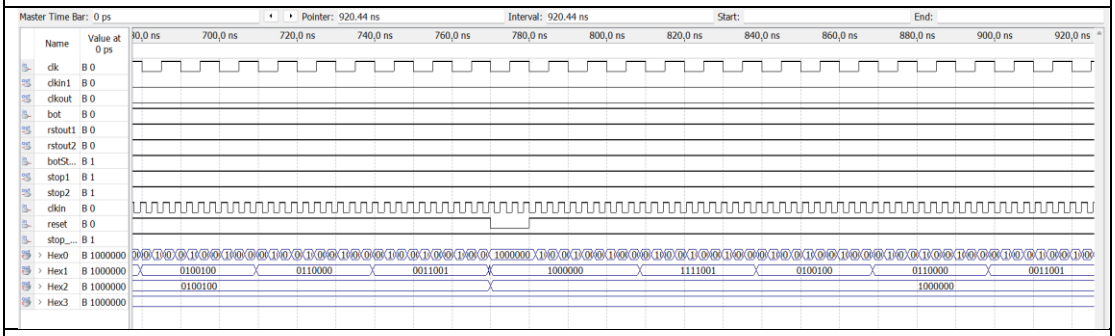
以上是本人在此專案中的接腳對應。

波形圖

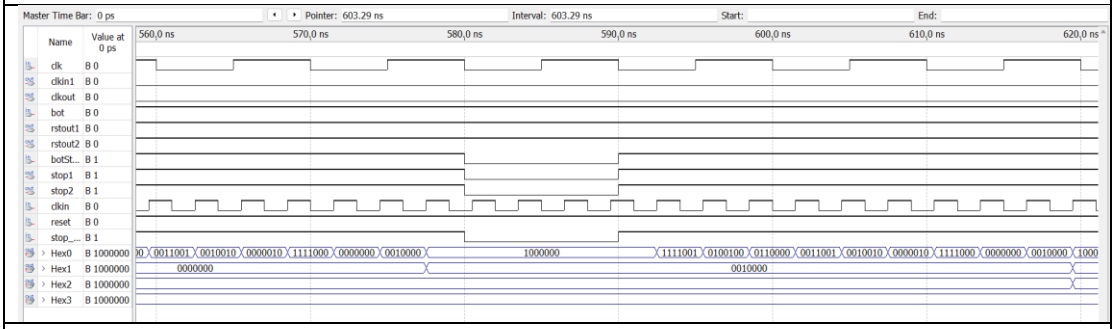
參考檔案：BCD_3ns.vwf



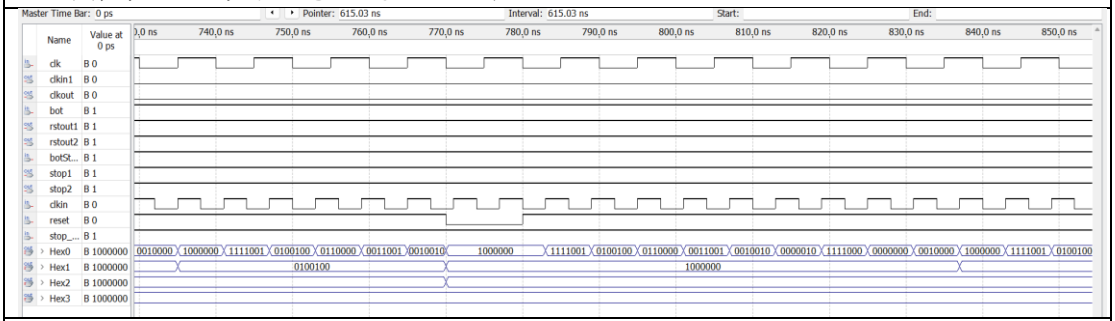
從上圖可以看到零到九都與數字應該對應到的七段顯示器相同。



上圖是在 clkin 設定 3ns 時的 reset，可以從 HEX1 中前後狀態看到有確實的做到歸零。且需要留意的是 reset 在約 770ns 的地方。

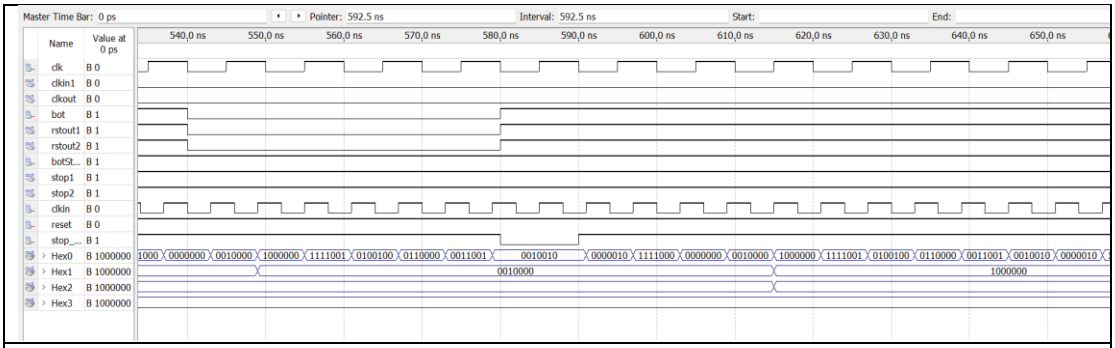


上圖是在 clkin 設定 3ns 時的 stop，可以從 HEX1 中前後狀態看到有確實的做到暫停。且需要留意的是 reset 在約 580ns 的地方。



上圖是在 clkin 設定 6ns 時的 reset，可以從 HEX1 中前後狀態看到有確實的做到歸零。且需要留意的是 reset 在約 770ns 的地方，與 3ns 的相同，因為沒辦法模擬出兩塊板子的行為。

可以看到的是 botStop 的暫停訊號在結果中有確實地被放大成三個訊號，bot 也是如此，故可以正確執行 stop、reset 的行為。



上圖是在 clkin 設定 6ns 時的 reset，可以從 HEX1 中前後狀態看到有確實的做到暫停。且需要留意的是 stop 在約 580ns 的地方，與 3ns 的相同，因為沒辦法模擬出兩塊板子的行為。

可以看到的是 botStop 的暫停訊號在結果中有確實地被放大成三個訊號，bot 也是如此，故可以正確執行 stop、reset 的行為。

額外功能：暫停計數(非歸零)

程式碼

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

entity upBCD_4bits_ten is
    port(
        clk:in std_logic;
        reset:in std_logic;
        stop_pos:in std_logic;
        casout:out std_logic;
        Q:out std_logic_vector(3 downto 0)
    );
end upBCD_4bits_ten;

architecture upBCD_4bits_ten of upBCD_4bits_ten is
    signal cnt:std_logic_vector(3 downto 0):=(others => '0');
    signal flag:std_logic:='0';
begin
    process(clk, reset)
    begin
        if(reset = '0') then
            cnt <= x"0";
            flag <= '0';
        elsif stop_pos = '0' then
            cnt <= cnt;
            flag <= flag;
        elsif rising_edge(clk) then
            if(cnt = x"9")then
                cnt <= x"0";
                flag <= '1';
            else
                cnt <= cnt + '1';
                flag <= '0';
            end if;
        end if;
    end process;
    Q <= cnt;
```

```
39      process(cnt)
40      begin
41          if ((cnt = x"0") and (flag = '1')) then
42              casout <= '1';
43          else
44              casout <= '0';
45          end if;
46      end process;
47  end upBCD_4bits_ten;
```

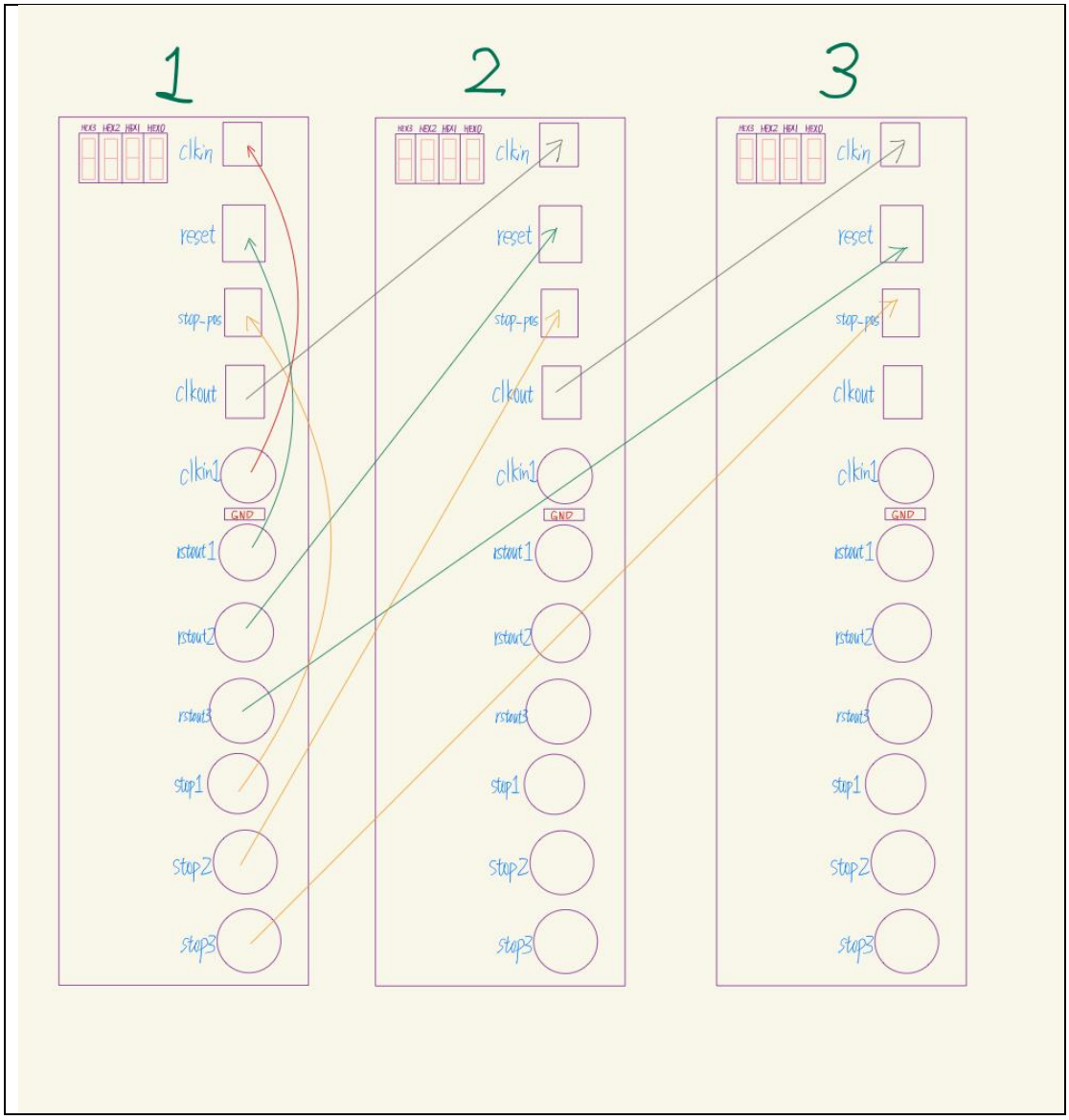
增加的功能是暫停，主要程式碼在 25~27 行，設定當非歸零、非可上數的操作即暫停。而波形圖的呈現已經於上面內容表示完畢。

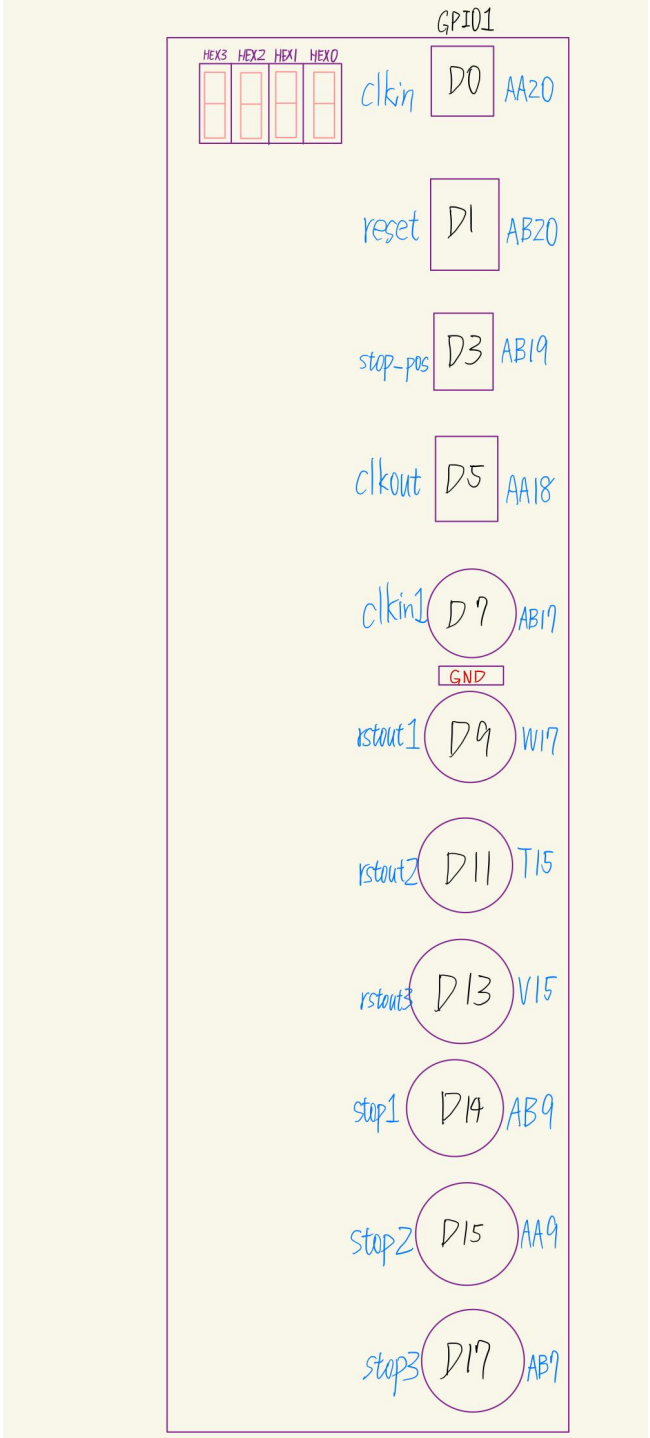
```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7  entity BCD_three is
8      port(
9          clk:in std_logic;                          -- 綁定 50M 時脈
10         bot:in std_logic;                          -- 綁定 bot 0 按鈕(reset)
11         botStop:in std_logic;                      -- 綁定 bot 1 按鈕(stop)
12
13         clkin:in std_logic;                        -- 接收時脈(BCD counter)
14         reset:in std_logic;                        -- 兩塊板子接收 reset
15         stop_pos:in std_logic;                     -- 接收 stop 信號
16         clkout:out std_logic;                      -- 發出進位時脈
17         clkin1:out std_logic;                      -- 發出除頻結果，讓第一塊板子拉線接收
18         stop1, stop2, stop3:out std_logic;         -- 接收 botStop，發出 stop 信號，拉線讓兩塊板子暫停
19         rstout1, rstout2, rstout3:out std_logic;   -- 接收 bot，發出 reset 信號，拉線讓兩塊板子重置
20         Hex0,Hex1,Hex2,Hex3:out std_logic_vector(6 downto 0)
21     );
22 end entity;
23
24 architecture BCD_three of BCD_three is
25     component upBCD_16bits is
26         port(
27             clk:in std_logic;
28             reset:in std_logic;
29             stop_pos:in std_logic;
30             casout:out std_logic;
31             Q:out std_logic_vector(15 downto 0)
32         );
33     end component;
34
35     component decoder_7seg is
36         PORT(
37             BCD:in std_logic_vector(3 downto 0);
38             HEX:out std_logic_vector(6 downto 0)
39         );
40     end component;
```

```
41      signal ans:std_logic_vector(15 downto 0);       -- BCD counter 結果
42      signal cnt:integer range 0 to 50000000:=0;      -- 除頻器(範圍 1 秒 -> 1/10000 秒)
43      signal clkCnt:std_logic:='0';                   -- 除頻後 clock
44      signal waste:std_logic:='0';                    -- stop 按下後，啥都不要做就是暫停
45  begin
46      rstout1 <= bot;
47      rstout2 <= bot;
48      rstout3 <= bot;
49
50      stop1 <= botStop;
51      stop2 <= botStop;
52      stop3 <= botStop;
53
54      process(clk)
55      begin
56          if(rising_edge(clk)) then
57              if (cnt < 4999) then
58                  cnt <= cnt + 1;
59                  clkCnt <= '0';
60              else
61                  clkCnt <= '1';
62                  cnt <= 0;
63              end if;
64          end if;
65      end process;
66      clkin1 <= clkCnt;                               -- 第一塊板子拉線接除頻結果
67                                                      -- 第二塊板子拉線接收第一塊板子的clkout
68
69      upBCD:upBCD_16bits port map(clkin, reset, stop_pos, clkout, ans);
70
71      -- 接七段顯示器
72      decoder0:decoder_7seg port map(ans(3 downto 0), Hex0);
73      decoder1:decoder_7seg port map(ans(7 downto 4), Hex1);
74      decoder2:decoder_7seg port map(ans(11 downto 8), Hex2);
75      decoder3:decoder_7seg port map(ans(15 downto 12), Hex3);
76  end BCD_three;
```

由上圖可以看到與基本題有明顯不同的是第 48 行、52 行，為了實現三塊板子的串接，必要的是多個 reset，因為第三塊板子也要因為我按下一個重置按鈕而作重置的操作。

至於暫停，也許可以不用，想法是如果前兩塊板子都暫停了，意味著已經停止上數，因此第三塊板子的時脈不可能在暫停的區間內有正緣觸發的動作，同理第二塊也是，因此最理想的狀態下是只有第一塊需要，往後都會因為第一塊停止上數而表現出暫停的現象。但為了保證結果的正確，最安全的方式也就是全都接。


1
2
3
HEX3 HEX2 HEX1 HEX0
clkin
reset
stop_pos
clkout
clkin1
GND
rstout1
rstout2
rstout3
stop1
stop2
stop3



圖中圓圈可以用寫死的方式來理解、而矩形則是可以想成一個模塊，可以接受任一時脈、重置來源、暫停來源。並附上本人在此專案中的接腳對應。

心得

透過此專案的撰寫，本人清楚的認知到自己這學期沒有白活，每次熬夜寫的硬體語言沒有還給老師，我還好好的應用到了專案中。也感謝自己每次的堅持，透過一次次基礎題與每次難搞的加分題慢慢累積出了勉強能做出小玩具的程度。