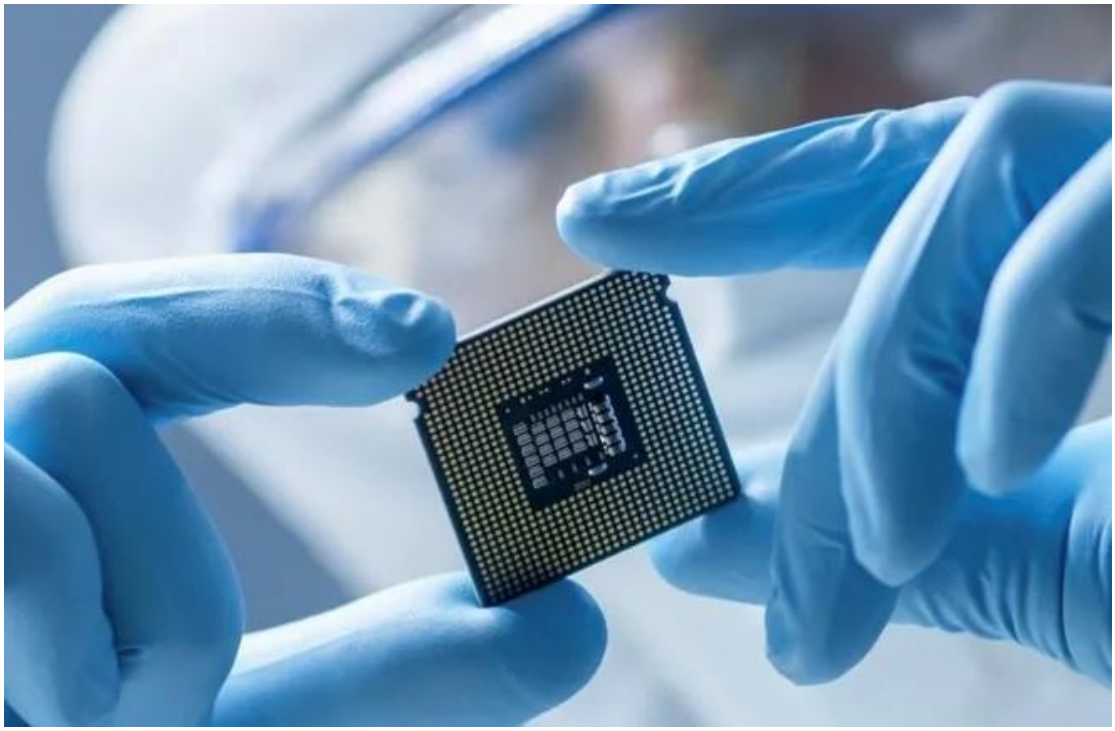


數位系統設計作業

HW2

學號 : 01257027 | 姓名 : 林承羿

第一題

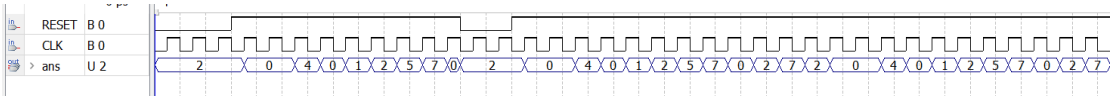
個人出生年+學號: 200401257027

程式碼

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

entity BirSchool is
    port(
        CLK:in std_logic;
        ans:out std_logic_vector(2 downto 0);
        RESET:in std_logic
    );
end BirSchool;
```

```

architecture BirSchool of BirSchool is
    signal cnt:std_logic_vector(4 downto 0);
begin
    process(RESET, CLK)                                     --200401257027
    begin
        if RESET = '0' then
            cnt <= "00" & "010";                            --強制回到開頭
        elsif rising_edge(CLK) then
            case cnt is
                when "00" & "010" => cnt <= "00" & "000"; --2>0
                when "00" & "000" => cnt <= "01" & "000"; --0>0
                when "01" & "000" => cnt <= "00" & "100"; --0>4
                when "00" & "100" => cnt <= "10" & "000"; --4>0
                when "10" & "000" => cnt <= "00" & "001"; --0>1
                when "00" & "001" => cnt <= "01" & "010"; --1>2
                when "01" & "010" => cnt <= "00" & "101"; --2>5
                when "00" & "101" => cnt <= "00" & "111"; --5>7
                when "00" & "111" => cnt <= "11" & "000"; --7>0
                when "11" & "000" => cnt <= "10" & "010"; --0>2
                when "10" & "010" => cnt <= "01" & "111"; --2>7
                when "01" & "111" => cnt <= "00" & "010"; --7>2
                when others => cnt <= "00" & "010";         --強制回到開頭
            end case;
        end if;
    end process;
    ans <= cnt(2 downto 0);                                 --更新結果
end BirSchool;
```

波形圖



可以清楚看到 RESET 運作情況，依照預設回到開頭。且遵循自己預設的規則走，設定兩 bit 作為避免 4 個 0 無反單一編碼。

第二題

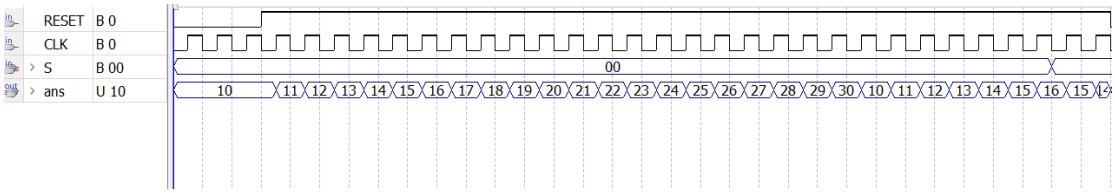
程式碼

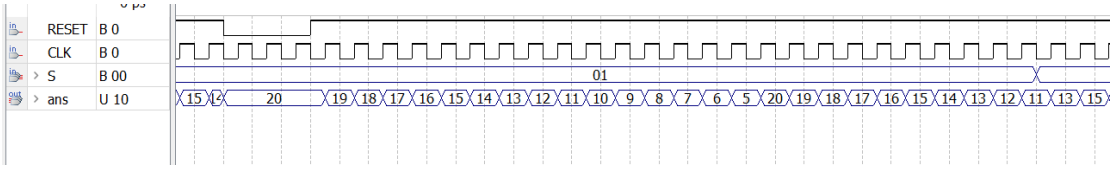
```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity CNT3period is
7       port(
8           CLK:in std_logic;
9           RESET:in std_logic;
10          S:in std_logic_vector(1 downto 0);
11          ans:out std_logic_vector(4 downto 0)
12      );
13  end CNT3period;
14
```

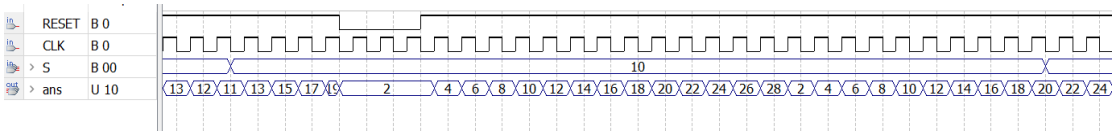
```
architecture CNT3period of CNT3period is
    signal cnt:std_logic_vector(4 downto 0);
begin
    process(CLK, RESET)
    begin
        if RESET='0' then                           --重置當下計數器
            if S="00" then
                cnt <= "01010";
            elsif S="01" then
                cnt <= "10100";
            elsif (S="10" or S="11") then
                cnt <= "00010";
            end if;
        elsif CLK'event and CLK='1' then            --正緣觸發
            if S="00" then                          --"00"為10~30上數1
                if cnt>="11110" then                --大於等於30變成10
                    cnt <= "01010";
                elsif cnt<"01001" then              --小於10變成10
                    cnt <= "01010";
                else
                    cnt <= cnt+"00001";
                end if;
            elsif S="01" then                       --"01"為20~5下數1
                if cnt<="00101" then                --小於等於5變成20
                    cnt <= "10100";
                elsif cnt>"10100" then              --大於20變成20
                        cnt <= "10100";
                else
                    cnt <= cnt-"00001";
                end if;
            elsif (S="10" or S="11") then           --"10"和"11"為2~28上數2
                if cnt>="11100" then                --大於等於28變成2
                    cnt <= "00010";
                elsif cnt<"00010" then              --小於2變成2
                    cnt <= "01010";
                else
                    cnt <= cnt+"00010";
                end if;
            end if;
        end if;
    end process;
    ans <= cnt;
end CNT3period;
```

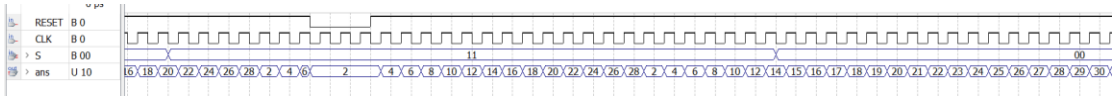
如果上數，理應設定到達上限即回至此區間下限，但考慮到如果從其他區間來到此區間，並不保證上下限於區間內，因此如果沒有在區間內，先設定高於上限即回至下限，低於下限回至下限。下數則是高於上限回至上限，低於下限回至上限。

波形圖

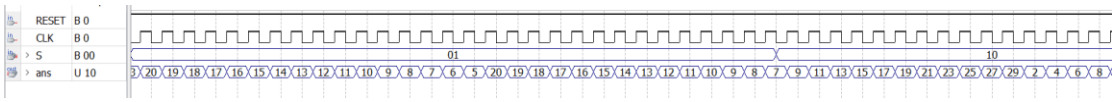




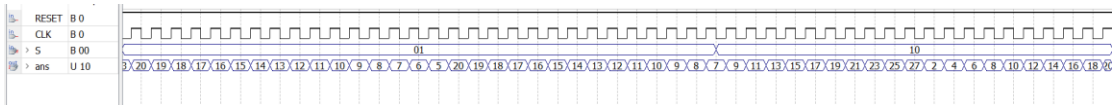




可清楚看到 RESET 作用在不同區間有不同結果，且高於區間設定邊界也確實回到預定的下限、上限。



特例，如果由奇數進入 10、11 區間且未 RESET，即會+2 直到超出 28，因此將 46 行 code 改成 >=”11011”，即偶數 26 會加至 28，奇數最多 27，保證區間正確性。

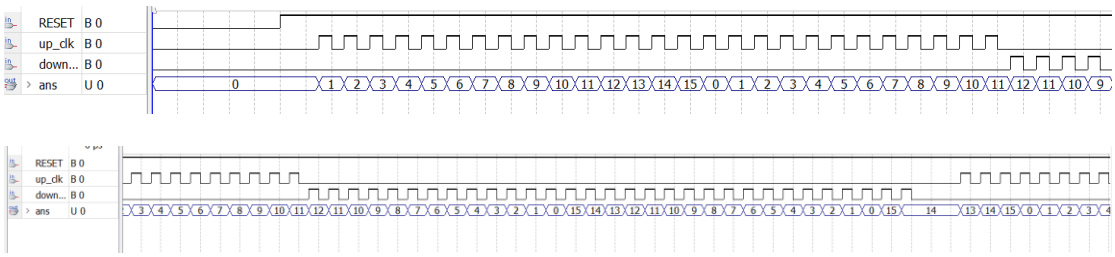


加分題

程式碼

撞牆：

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

entity up_downCNT is
    port(
        up_clk:in std_logic;
        down_clk:in std_logic;
        RESET:in std_logic;
        ans:out std_logic_vector(3 downto 0)
    );
end up_downCNT;

architecture up_downCNT of up_downCNT is
    signal cnt:std_logic_vector(3 downto 0);
    signal clk:std_logic;
     signal last_up:std_logic;
     signal last_down:std_logic;
begin


    clk <= up_clk or down_clk;
    process(RESET, clk)
    begin
        if RESET = '0' then
            cnt <= "0000";
        elsif rising_edge(clk) then
            if down_clk='0' and last_down='1' then      --如果上一次的我跟現在的我不一樣，就動作
                cnt <= cnt - '1';
            else
                cnt <= cnt + '1';
            end if;
            last_up <= up_clk;                          --紀錄上一次正元觸發紀錄
            last_down <= down_clk;
        end if;
    end process;
    ans <= cnt;
end up_downCNT;
```



問題：在邊界會有問題，因為要連續兩個才能判定，而邊界上一個有作用時是該動的，與現在不該動的情況確實符合自己設的條件，故看起來像是慢了一拍的結果。開頭並沒有問題，因為開頭符合連續兩個皆處於相同狀態，不會動作。

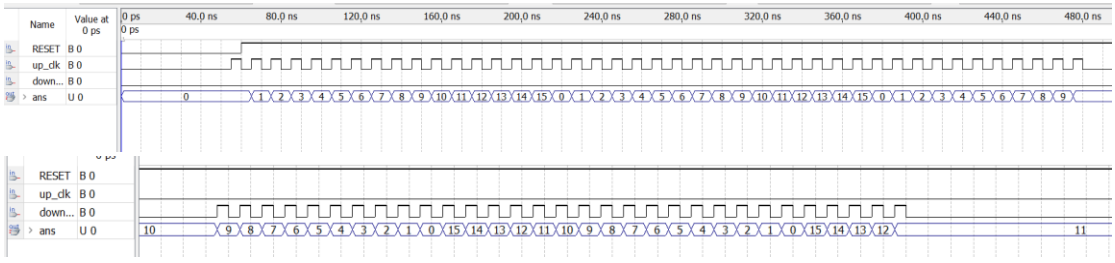
暴力的放棄：

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

entity Nup_down is
    port(
        up_clk:in std_logic;
        down_clk:in std_logic;
        RESET:in std_logic;
        ans:out std_logic_vector(3 downto 0)
    );
end Nup_down;

architecture Nup_down of Nup_down is
    signal cnt:std_logic_vector(3 downto 0);
   signal clk:std_logic;
```

```vhdl
begin
    clk <= (up_clk or down_clk);                    --當作唯一的clock
    process(clk, RESET)
    begin
        if RESET='0' then
            cnt <= (others => '0');                 --重設
        elsif clk'event and clk='1' then            --正緣來時，當需要+1，down_clk必為0
            if down_clk='0' then
                cnt <= cnt+'1';
            else
                case cnt is                         --因為考慮到不只一個條件需要考慮，在未知情況下使用暴力枚舉
                    when x"F" => cnt <= x"E";
                    when x"E" => cnt <= x"D";
                    when x"D" => cnt <= x"C";
                    when x"C" => cnt <= x"B";
                    when x"B" => cnt <= x"A";
                    when x"A" => cnt <= x"9";
                    when x"9" => cnt <= x"8";
                    when x"8" => cnt <= x"7";
                    when x"7" => cnt <= x"6";
                    when x"6" => cnt <= x"5";
                    when x"5" => cnt <= x"4";
                    when x"4" => cnt <= x"3";
                    when x"3" => cnt <= x"2";
                    when x"2" => cnt <= x"1";
                    when x"1" => cnt <= x"0";
                    when x"0" => cnt <= x"F";
                    when others => cnt <= X"0";     --只是為了符合文法
                end case;
            end if;
        end if;
    end process;
    ans <= cnt;                                     --輸出答案
end Nup_down;
```

不講道理，毫無擴充性可言，但用時間換結果看起來也不錯，when others 不應該出現，因為已經列舉出在 16 bits 下所有可能，但為了文法：)

波形圖



這作法只保證了上數、下數單一邊符合假設，故上數對了，下數即會出錯，因此在保證單邊滿足情況下，另一半使用暴力枚舉。也確實看到非常有用，與事實假設符合。

心得

邏輯死去，只有暴力才是一切的正解。感謝加分題讓我卡了快 24 小時，教授考不考慮贊助一下我住進豪華版 ICU。

使用兩個時鐘信號來控制加減操作是比較罕見的設計，因為一般的計數器通常只依賴一個時鐘信號進行運行。這裡要實現的是根據兩個不同的時鐘信號進行加法和減法操作，這使得邊沿檢測和時序控制變得更加複雜。

這些 VHDL 程式碼反映了硬體描述語言設計的複雜性和挑戰性，特別是當涉及多個時鐘訊號的控制時。編寫這段程式碼加深了我對邏輯設計的理解，尤其是對於時序和邊沿檢測的精確控制的需求。這類型的設計也強調了模擬和測試的重要性，這是確保硬體電路在真實環境中正常工作的關鍵。