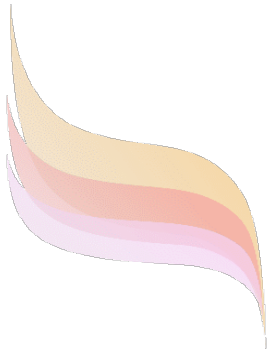
Digital System Design

Lecture 8
Design Examples I

Learning Examples:

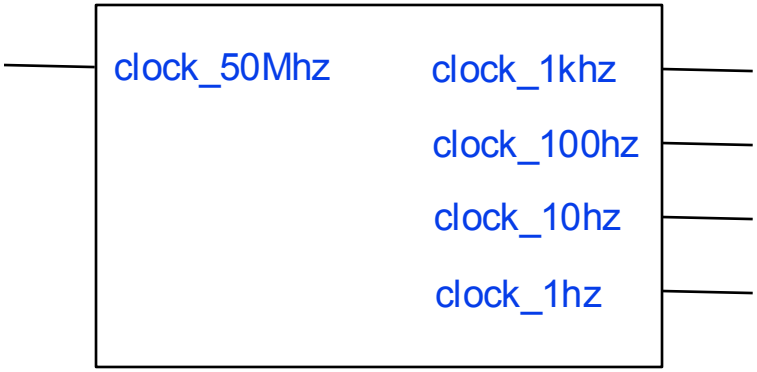
- **Frequency Generator**
  - 24時制電子鐘
- **Debounce Circuit**
- **LED control**
- **8x8 LED Matrix Control**

Frequency Generator



Frequency generator I

- 50Mhz to 1khz, 100hz, 10z, and 1hz



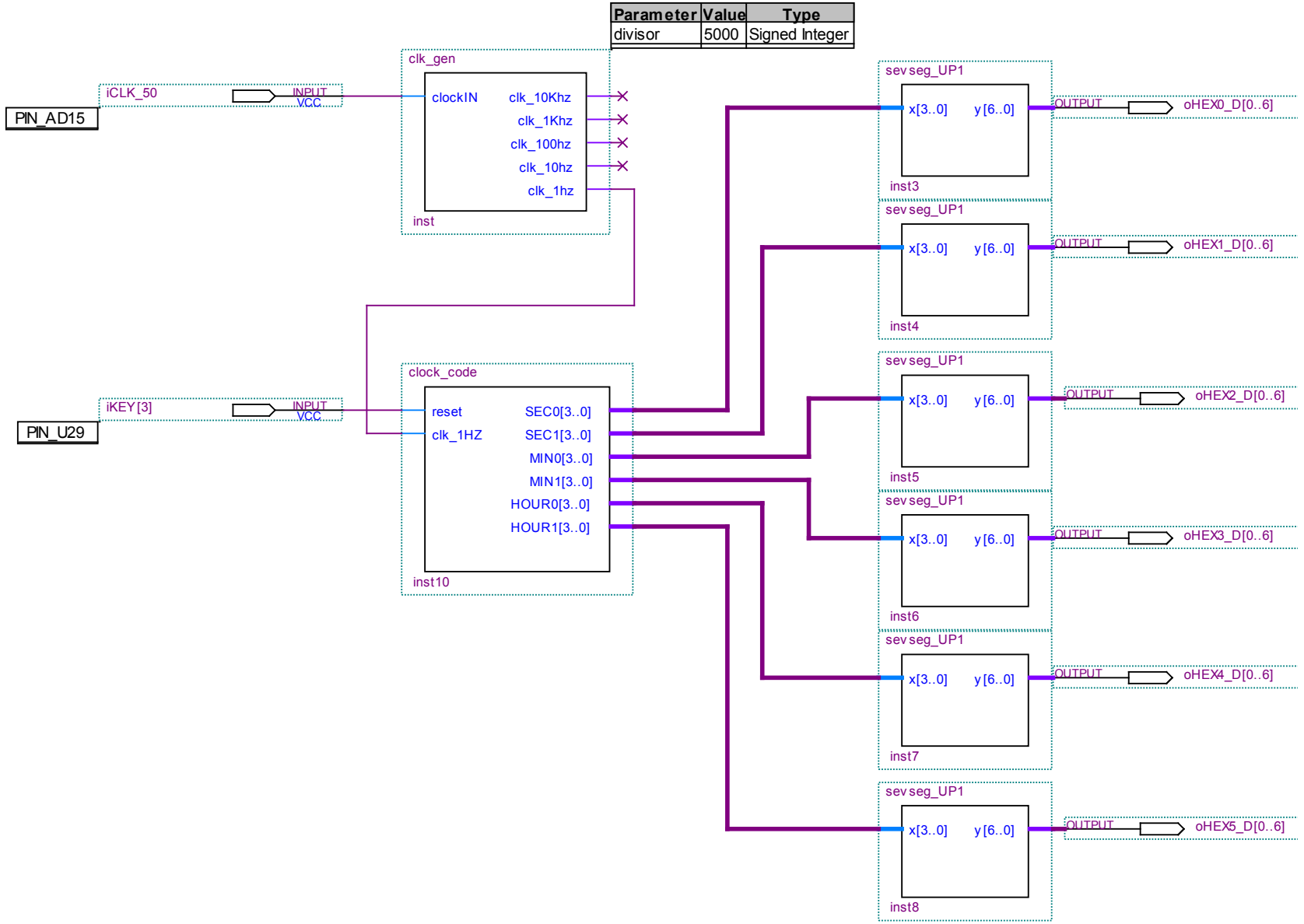

```
entity CLK_DIV is
   port
   (
      clock_50Mhz          : IN  STD_LOGIC;
      clock_1KHz           : OUT STD_LOGIC;
      clock_100Hz          : OUT STD_LOGIC;
      clock_10Hz           : OUT STD_LOGIC;
      clock_1Hz            : OUT STD_LOGIC);
end CLK_DIV;
```

Frequency generator II

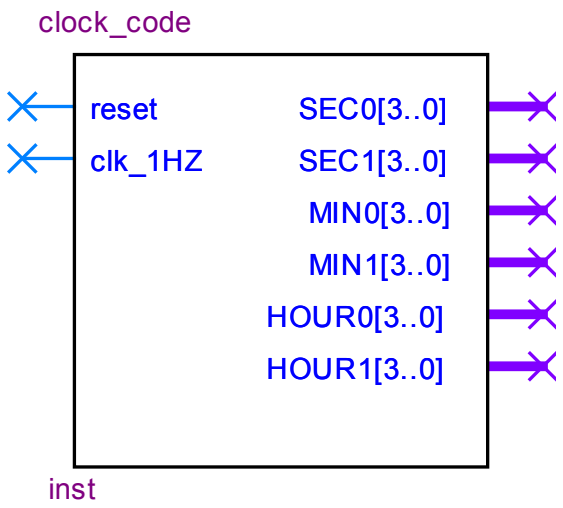
◆ User-specified


```
entity CLK_GEN is
   generic( divisor: integer := 50_000_000 );
   port
   (
      clock_in          : IN  STD_LOGIC;
      clock_out         : OUT STD_LOGIC);
end CLK_GEN;
```

可顯示“時”：“分”：“秒”的24時制電子鐘



VHDL code

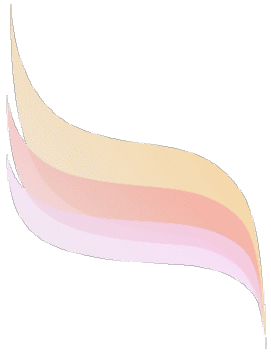



```
ARCHITECTURE a OF clock_code IS
BEGIN

PROCESS (Clk_1HZ, reset)
BEGIN
   IF reset = '0' THEN
      HOUR1<= X"0"; HOUR0<= X"0";
      MIN1 <= X"0"; MIN0 <= X"0";
      SEC1 <= X"0"; SEC0 <= X"0";

   ELSIF clk_1HZ'EVENT AND clk_1HZ = '1' THEN
    -- 個位數“秒”
      IF SEC0 < 9 THEN
         SEC0 <= SEC0 + 1;
      ELSE
     -- 十位數“秒”
         SEC0 <= "0000";
         IF SEC1 < 5 THEN
            SEC1 <= SEC1 + 1;
         ELSE
     -- 個位數“分”
            SEC1 <= "0000";
            IF MIN0 < 9 THEN
               MIN0 <= MIN0 + 1;
            ELSE
     -- 十位數“分”
               MIN0 <= "0000";
               IF MIN1 < 5 THEN
                  MIN1 <= MIN1 + 1;
               ELSE
```

VHDL code

```
-- 個位數“時”
            MIN1 <= "0000";
            IF HOUR0< 9 AND NOT((HOUR1=2) AND (HOUR0=3))THEN
               HOUR0 <= HOUR0 + 1;
            ELSE
--十位數“時”
               IF NOT((HOUR1 = 2) AND (HOUR0 = 3)) THEN
                  HOUR1 <= HOUR1 + 1;
                  HOUR0 <= "0000";
               ELSE
-- 新的一天
                  HOUR1 <= "0000";
                  HOUR0 <= "0000";
               END IF;
            END IF;
         END IF;
      END IF;
   END IF;
END IF;
END PROCESS;
END a;
```

LED Control



LED 單燈右移

功能說明：

使用 CPLD 晶片，設計一個 LED 單燈右移的數位電路。LED 變化如圖 8-2 所示每秒單燈右移一次。

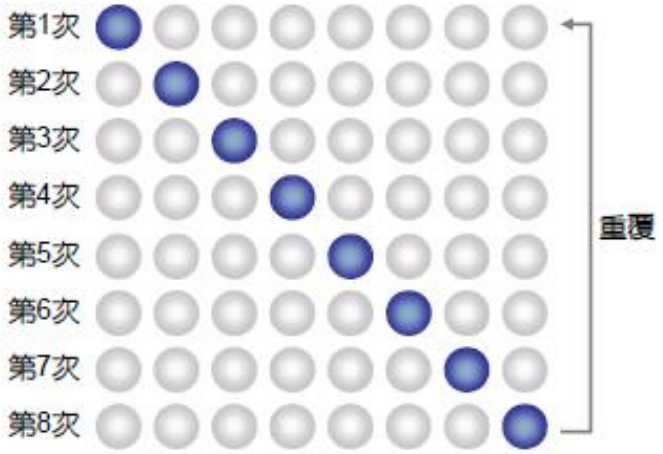


圖8-2　LED 單燈右移的變化情形

Source: CPLD 數位邏輯設計實務

LED 單燈右移　(ror1bit.vhd)

- **電路方塊圖：**

如圖 8-3 所示爲 LED 單燈右移電路方塊圖，包含**除頻電路、LED 右移電路、反相電路**等三個部份。

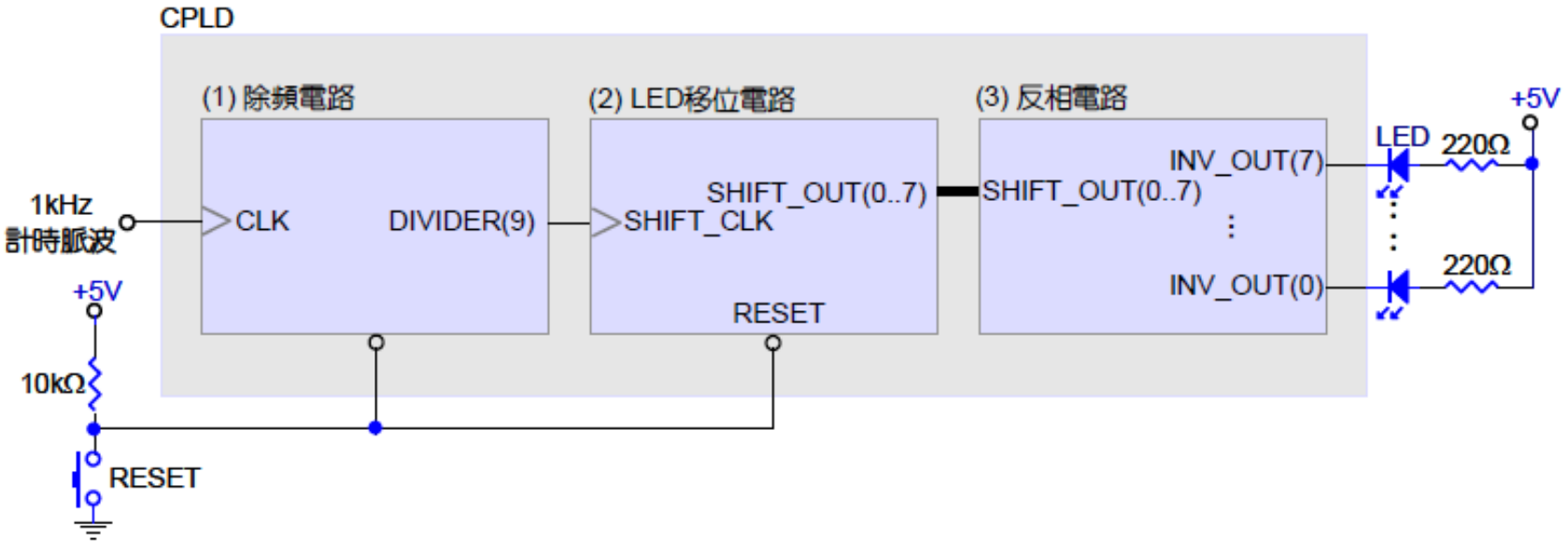


圖8-3　LED 單燈右移電路方塊圖

Source: CPLD 數位邏輯設計實務

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

ENTITY lab71 IS
PORT (
    CLOCK_50      : IN STD_LOGIC;                  -- 輸入時脈
    KEY           : IN STD_LOGIC_VECTOR(1 DOWNTO 0);  -- 按鈕，使用 KEY[1] 作為重設
    LEDG          : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)  -- LED 輸出
    --HEX0        : OUT STD_LOGIC_VECTOR(0 TO 6)      -- 七段顯示器輸出
);
END lab71;

ARCHITECTURE Behavioral OF lab71 IS
    SIGNAL clk_div : INTEGER RANGE 0 TO 49999999 := 0; -- 用於生成 1Hz 的計時器
    SIGNAL led_counter : INTEGER RANGE 0 TO 7 := 0;     -- LED 計數器
BEGIN
    PROCESS (CLOCK_50, KEY)
    BEGIN
        IF KEY(1) = '0' THEN                    -- 當按鈕按下 (有效低) 時重設
            led_counter <= 7;                   -- LED 計數器重置為 7
            LEDG <= "10000000";                 -- 重置 LED 點亮狀態為 LEDG(7)
            clk_div <= 0;                       -- 計時器重置
        ELSIF rising_edge(CLOCK_50) THEN                -- 在時脈的上升沿
            clk_div <= clk_div + 1;             -- 計時器加 1

            IF clk_div = 49999999 THEN              -- 每 50MHz 時脈下 25000000 次，即 1Hz
                clk_div <= 0;                   -- 重置計時器

                -- 使 led_counter 倒數
                IF led_counter = 0 THEN             -- 當計數器為 0 時重置到 7
                    led_counter <= 7;
                ELSE
                    led_counter <= led_counter - 1;   -- 否則減 1
                END IF;

                LEDG <= (others => '0');            -- 先將所有 LED 熄滅
                LEDG(led_counter) <= '1';           -- 點亮當前計數的 LED
            END IF;
        END IF;
    END PROCESS;

END Behavioral;
```

LED 單燈左右移 (霹靂燈)

- **功能說明：**

使用 CPLD 晶片，設計一個 LED 單燈左右移的數位電路。LED 如圖 8-6 所示，每 0.5 秒單燈左右來回移動。

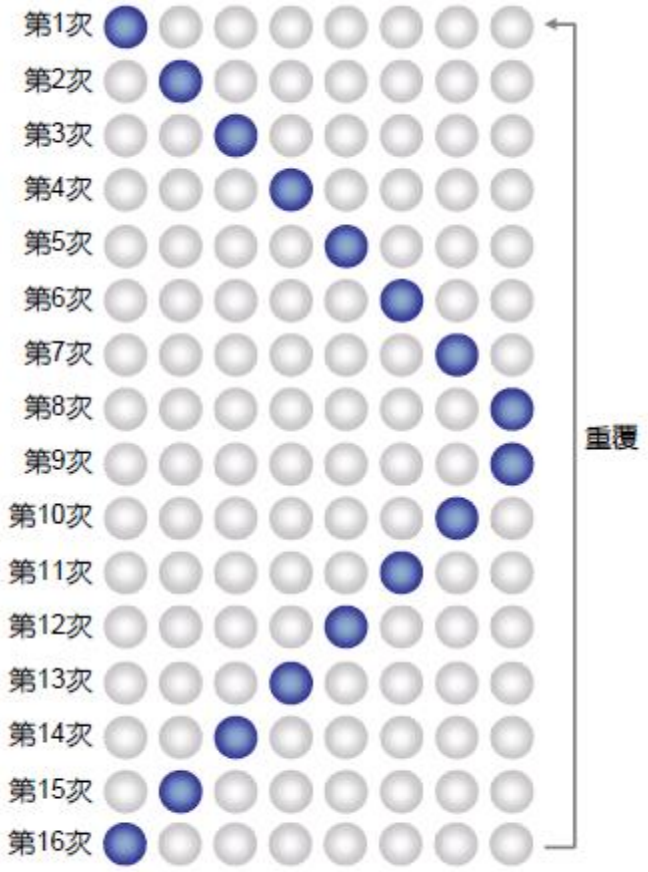


圖8-6　LED 單燈左右移的變化情形

Source: CPLD 數位邏輯設計實務

LED 單燈左右移(霹靂燈) (pili1bit.vhd)

電路方塊圖：

如圖 8-7 所示爲 LED 單燈左右移電路方塊圖，包含**除頻電路、除 16 計數電路、解碼電路**等三個部份。

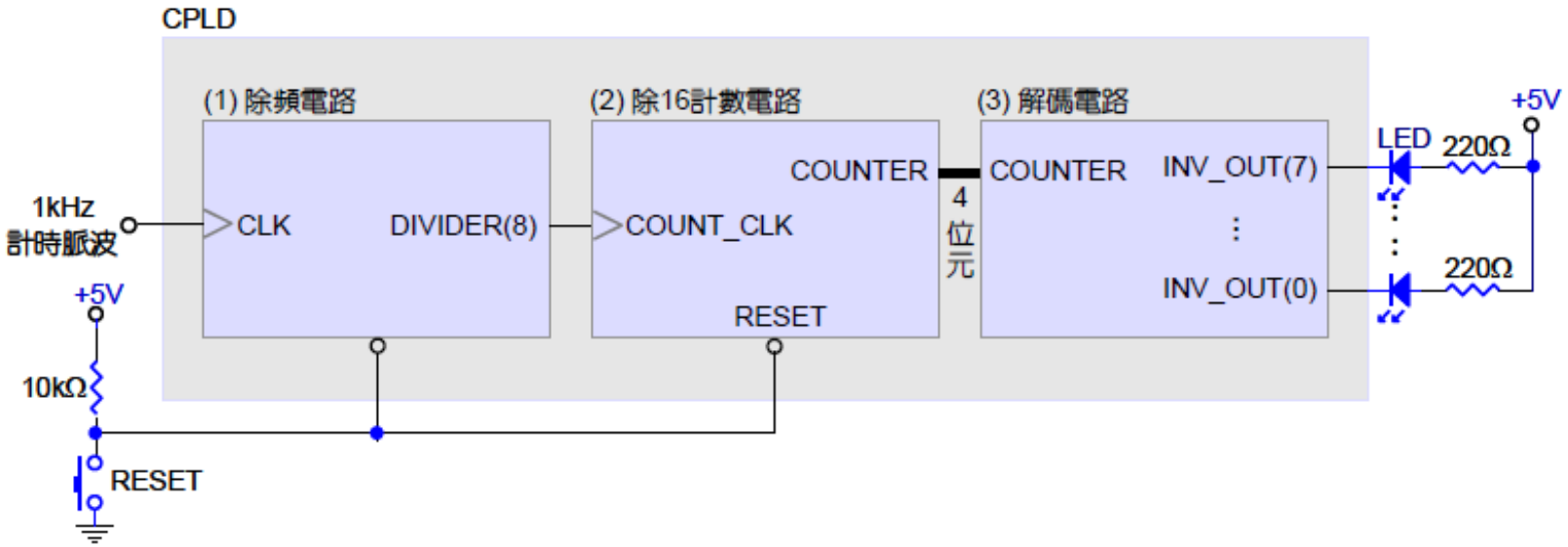


圖8-7　LED 單燈左右移電路方塊圖

Source: CPLD 數位邏輯設計實務

指撥開關控制LED移位速度

- **功能說明：**

使用 CPLD 晶片，設計指撥開關控制 LED 單燈移位速度的數位電路。當指撥開關 S1 切至 ON 位置時，移位脈波爲 1Hz，LED 移位速度最慢；當指撥開關 S2 切至 ON 位置時，移位脈波爲 2Hz；當指撥開關 S3 切至 ON 位置時，移位脈波爲 4Hz…餘依此類推；當指撥開關 S8 切至 ON 位置時，移位脈波爲 128Hz，LED 移位速度最快。

※ 令其中一個按鍵的優先級最高

Maybe 按2个以上令其不亮

Source: CPLD 數位邏輯設計實務

指撥開關控制LED移位速度 (dip8_var_shift.vhd)

電路方塊圖：

如圖 9-10 所示爲指撥開關控制 LED 移位速度電路方塊圖，包含**除頻電路、多工電路、LED 移位電路、緩衝電路**等四個部份。

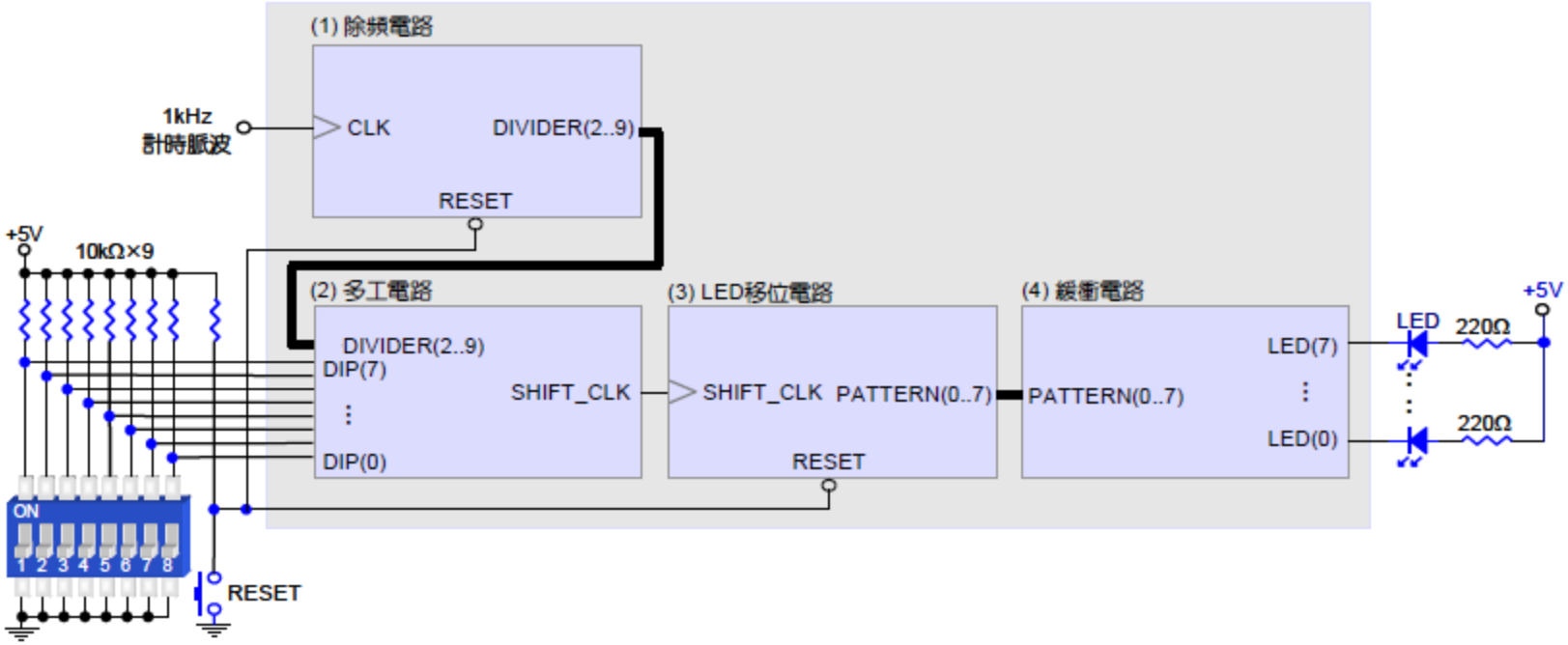


圖9-10　指撥開關控制 LED 移位速度電路方塊圖

Source: CPLD 數位邏輯設計實務

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

entity DIP8_VAR_SHIFT is
	port
	(	RESET,CLK:in std_logic;
		DIP:in std_logic_vector(7 downto 0);
		LED:out std_logic_vector(7 downto 0));
end DIP8_VAR_SHIFT;

architecture arch of DIP8_VAR_SHIFT is
signal DIVIDER:std_logic_vector(9 downto 0);
signal SHIFT_CLK:std_logic;
signal PATTERN:std_logic_vector(7 downto 0);
begin

--divider
process(CLK,RESET)
begin
	if RESET='0' then
		DIVIDER<="0000000000";
	elsif CLK'event and CLK='1' then
		DIVIDER<=DIVIDER+1;
	end if;
end process;

--multiplxer
with DIP select
SHIFT_CLK<=	DIVIDER(2)  when "01111111",
			DIVIDER(3)  when "10111111",
			DIVIDER(4)      when "11011111",
			DIVIDER(5)      when "11101111",
			DIVIDER(6)      when "11110111",
			DIVIDER(7)      when "11111011",
			DIVIDER(8)      when "11111101",
			DIVIDER(9)      when others;

--shift circuit
process(SHIFT_CLK,RESET)
begin
	if RESET='0' then
		PATTERN<="01111111";
	elsif SHIFT_CLK'event and SHIFT_CLK='1' then
		PATTERN<=PATTERN(0)&PATTERN(7 downto 1);
	end if;
end process;

--buffer circuit
LED<=PATTERN;
end arch;
```

指撥開關控制LED不同變化

功能說明：

使用 CPLD 晶片，設計指撥開關控制 LED 產生單燈右移、單燈左移、閃爍及左右來回移等四種變化的數位電路。當指撥開關 S1 切至 ON 位置時，LED 單燈右移；當指撥開關 S2 切至 ON 位置時，LED 單燈左移；當指撥開關 S3 切至 ON 位置時，LED 閃爍；當指撥開關 S4 切至 ON 位置時，LED 左右來回移。

電路方塊圖：

如圖 9-12 所示爲指撥開關控制 LED 變化電路方塊圖，包含**除頻電路**、**LED 右移電路**、**LED 左移電路**、**LED 閃爍電路**、**LED 左右移電路及多工電路**等六個部份。

Source: CPLD 數位邏輯設計實務

指撥開關控制LED不同變化 (dip8_var_led.vhd)

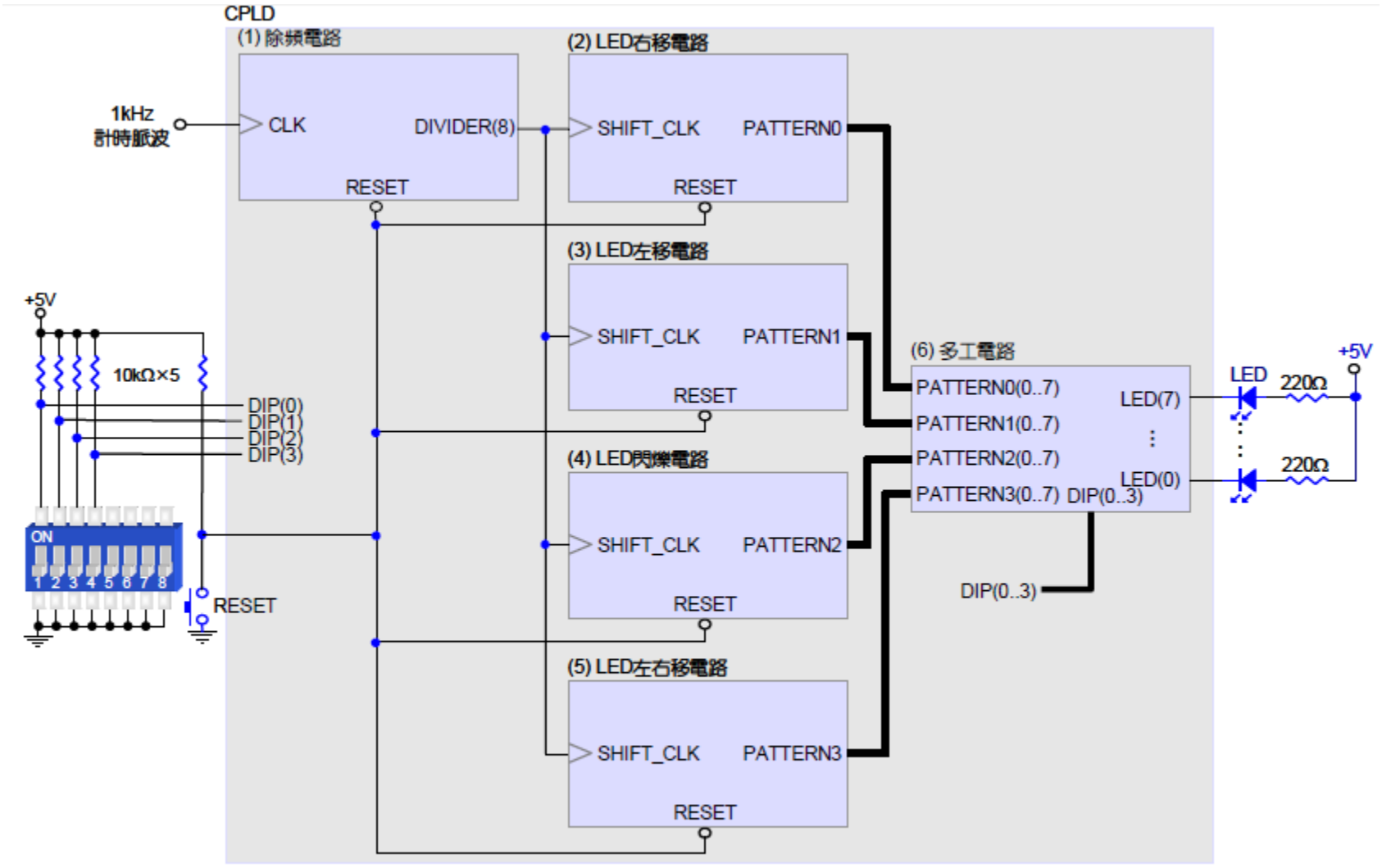


圖9-12　指撥開關控制 LED 變化電路方塊圖

Source: CPLD 數位邏輯設計實務

按鍵開關控制LED移位方向

- **功能說明：**

使用 CPLD 晶片，設計按鍵開關 SW 控制 LED 移位方向的數位電路。按下 RESET 鍵時，執行 LED 單燈右移；每一次按 SW 鍵時，LED 的移位方向會在單燈左移與單燈右移之間改變。

(a) 正脈波型　　(b) 負脈波型

圖9-14　按鍵開關電路

Source: CPLD 數位邏輯設計實務

按鍵開關控制LED移位方向 (tack_sw1_led.vhd)

電路方塊圖：

如圖 9-17 所示爲指撥開關控制 LED 移位方向電路方塊圖，包含**除頻電路、除彈跳電路、方向控制電路、LED 移位電路**等四個部份。

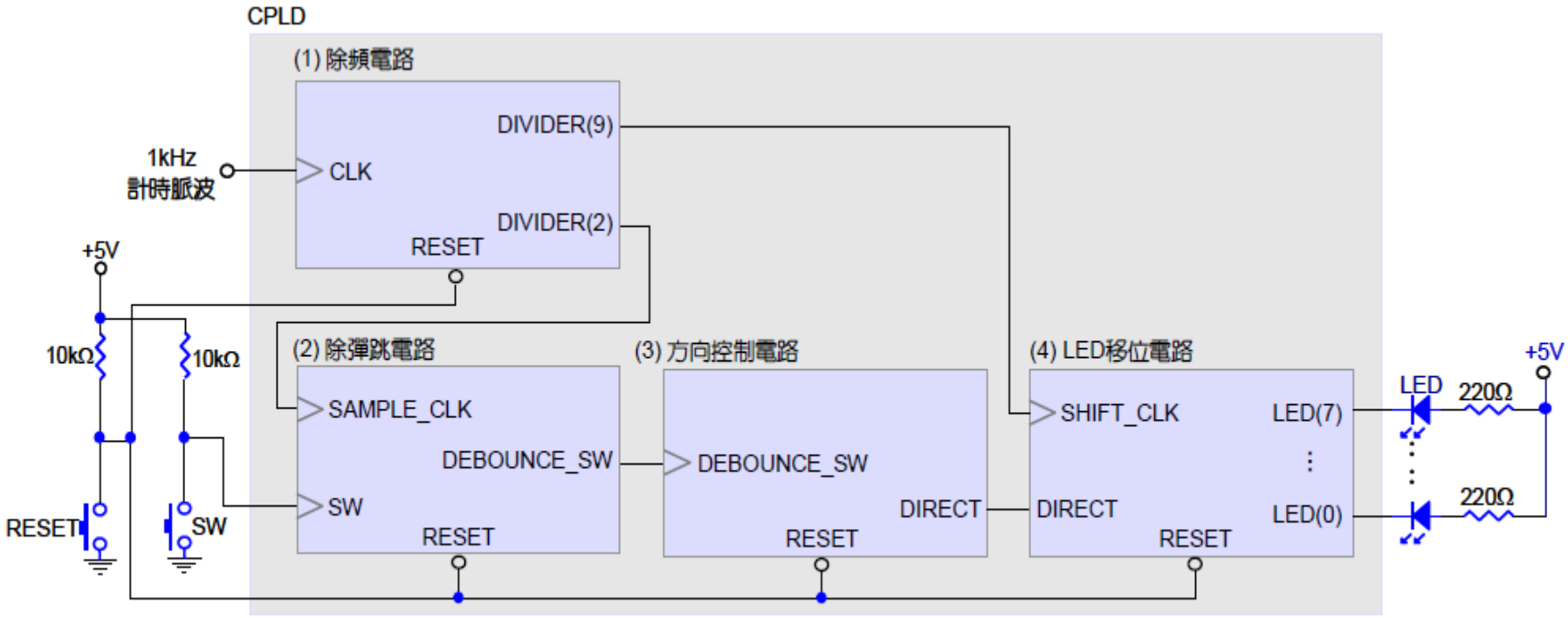


圖9-17　指撥開關控制 LED 移位方向電路方塊圖

Source: CPLD 數位邏輯設計實務

按鍵開關控制LED不同變化

功能說明：

使用 CPLD 晶片，設計使用四個按鍵開關控制 LED 產生單燈右移、單燈左移、同時閃爍及左右來回移等四種變化的數位電路。當按下 SW0 鍵時，執行單燈右移動作，按下 SW1 鍵時，執行單燈左移動作，按下 SW2 鍵時，執行閃爍動作，按下 SW3 鍵時，執行左右來回移動作。

電路方塊圖：

如圖 9-19 所示爲按鍵開關控制 LED 變化電路方塊圖，包含**除頻電路**、**除彈跳電路**、**LED 右移電路**、**LED 左移電路**、**LED 閃爍電路**、**LED 左右移電路**、**多工電路**等七個部份。

Source: CPLD 數位邏輯設計實務

按鍵開關控制LED不同變化 (tack_sw4_led.vhd)

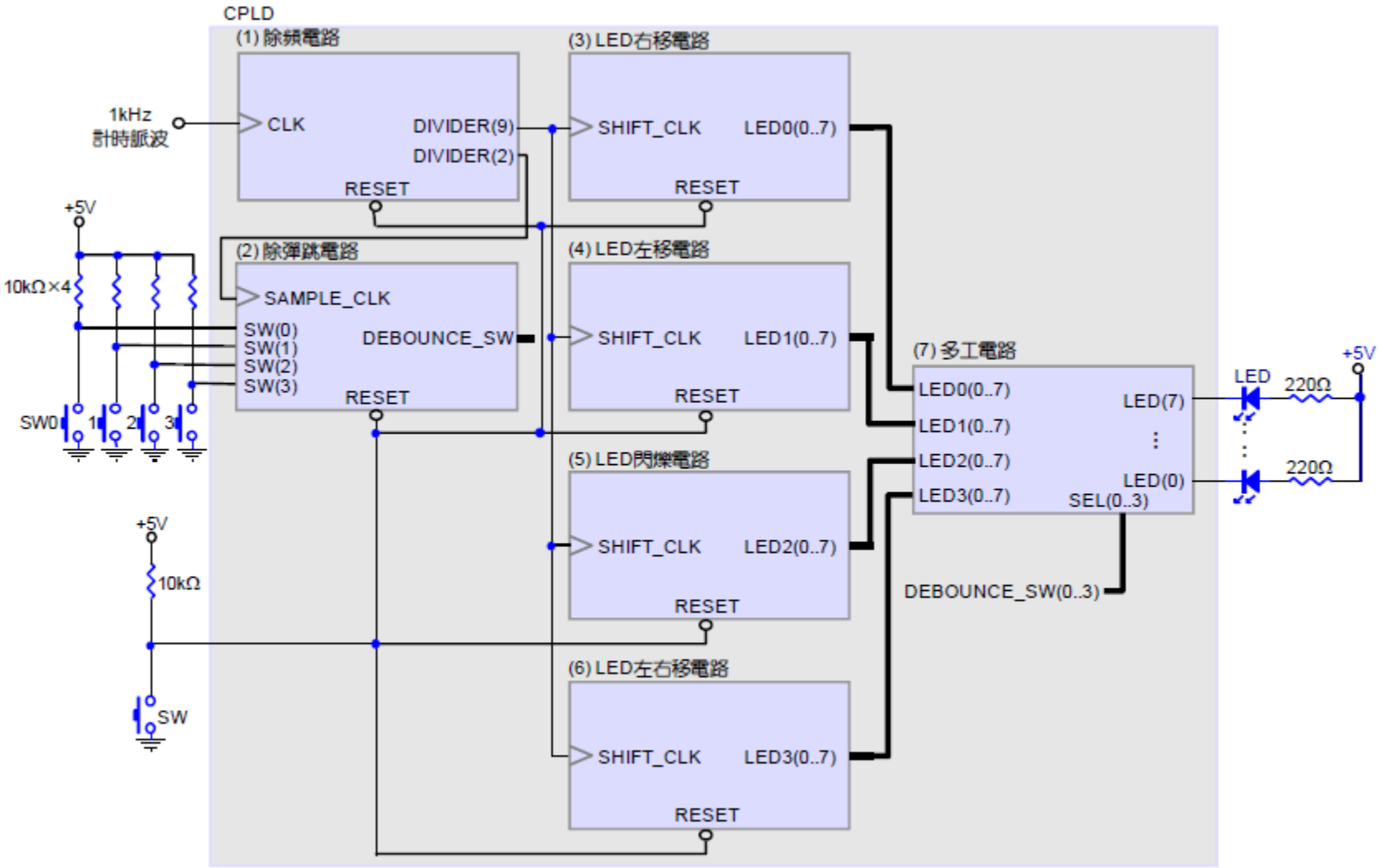
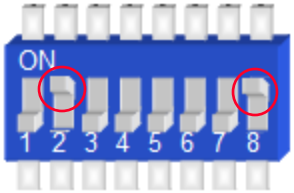


圖9-19　按鍵開關控制 LED 變化電路方塊圖

Source: CPLD 數位邏輯設計實務

LEDs Flashing (dip8flash.vhd)



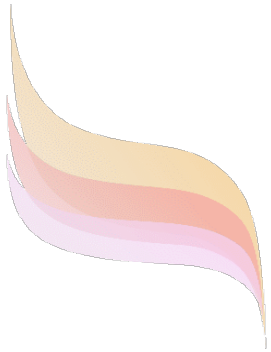
暗 閃 暗 暗 暗 暗 暗 閃

⇨ Modified for DE0

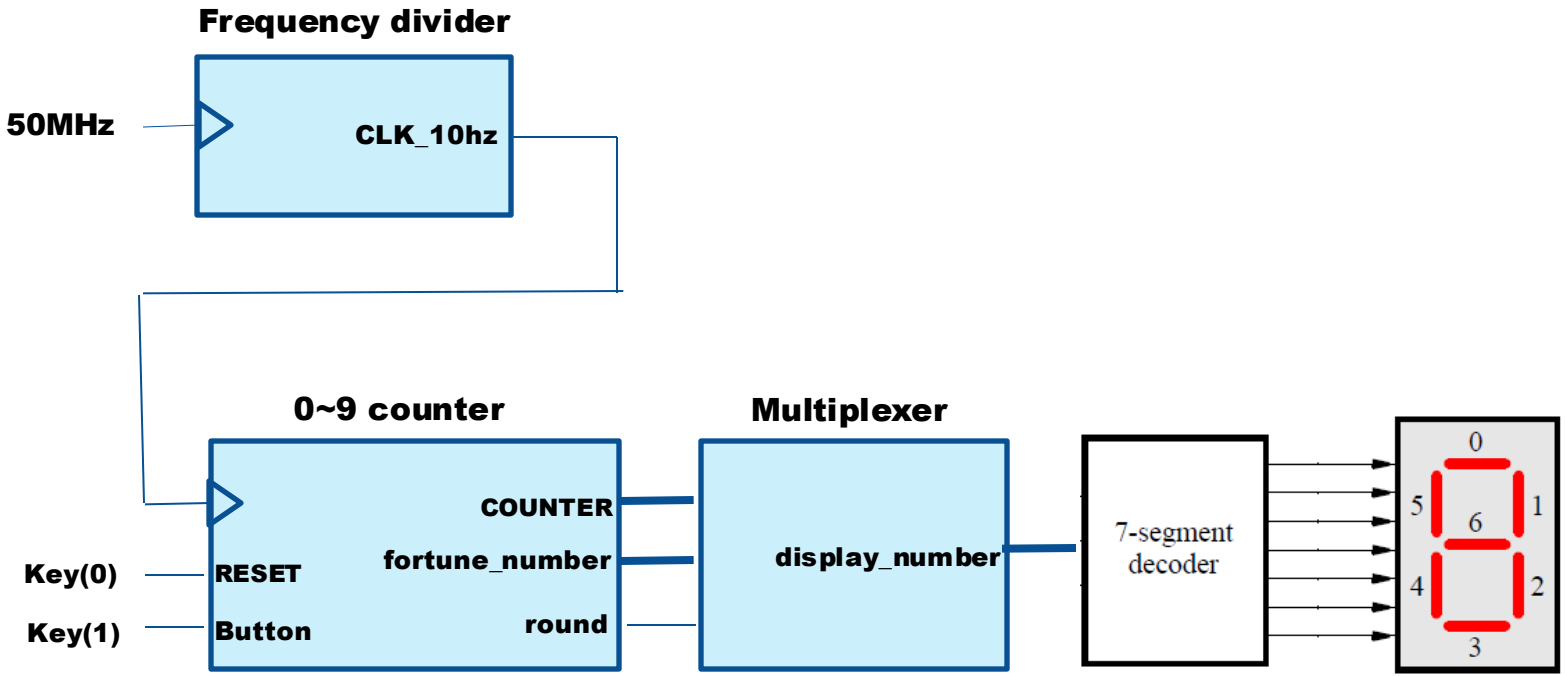
```
entity DIP8FLASH is
   port
   (  CLOCK_50:in std_logic;
      KEY:in std_logic_vector(2 downto 0);
      SW:in std_logic_vector(7 downto 0);
      LEDG:out std_logic_vector(7 downto 0));
end DIP8FLASH;
```


Source: CPLD 數位邏輯設計實務

Wheel of Fortune

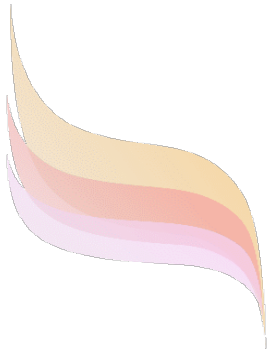


Wheel of Fortune (count_0_9.vhd)

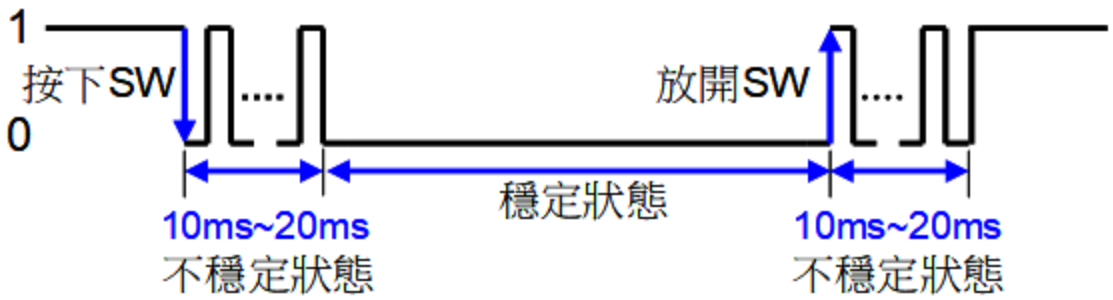


- 按下Key(0), 7-segment 會從數字0開始,快速循環顯示0 ~ 9.
- 在0 ~ 9的數字顯示過程中, 如果按下Key(1), 7-segment 會顯示”幸運數字(固定不變).
- 再按一次Key(1)的話, 7-segment 會重新快速循環顯示0 ~ 9.

Debounce Circuit



Button Debouncing



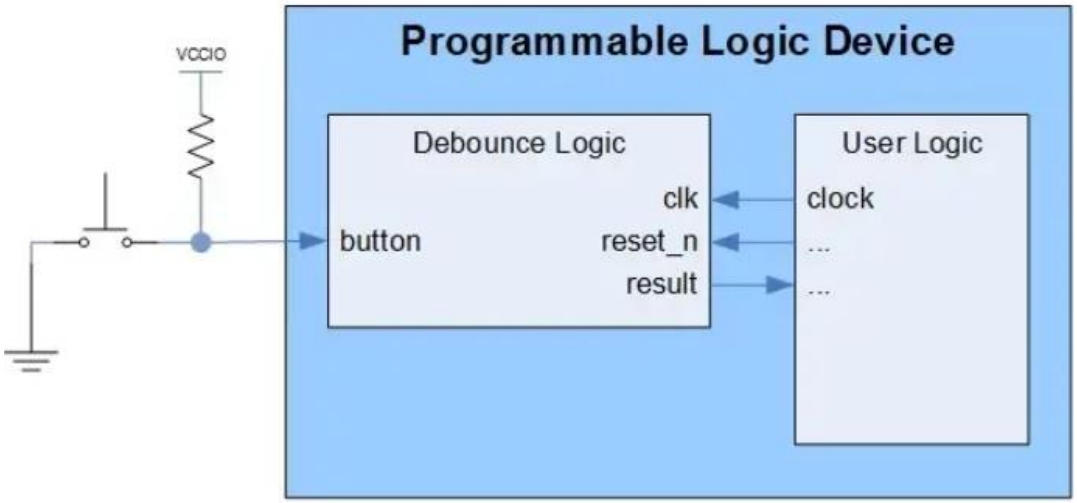
- 作法一: 由於一般彈跳時間約在10ms~20ms間，可直接採用較慢的取樣頻率, i.e. >= 20ms, 取樣(sample)按鍵值.

- 作法二: 採用較快的取樣頻率, i.e. <= 5ms，先讀取一次，隔一段時間後，再讀取一次。如果持續 n 次 (i.e., n=5)讀到的狀態都一樣，就視為按鍵處於穩定狀態，否則就是 bouncing。
  - An n-bit shift register can be used.
  - Or, the circuit on the following pages will be used.

(Note: DE0 的 pushbuttons 都已 debounce 處理過.)

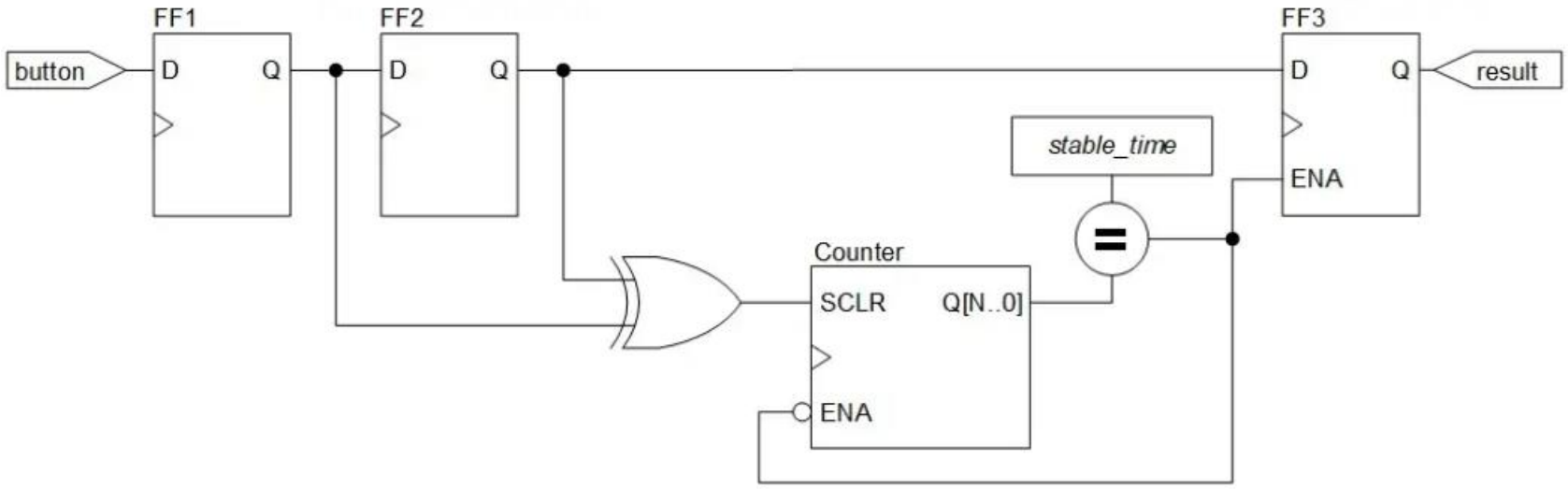
Debounce Logic Circuit

- The debounce component presented here is a simple digital logic circuit that addresses this temporary ambiguity (a common task when interfacing FPGAs or CPLDs with pushbuttons or other switches).



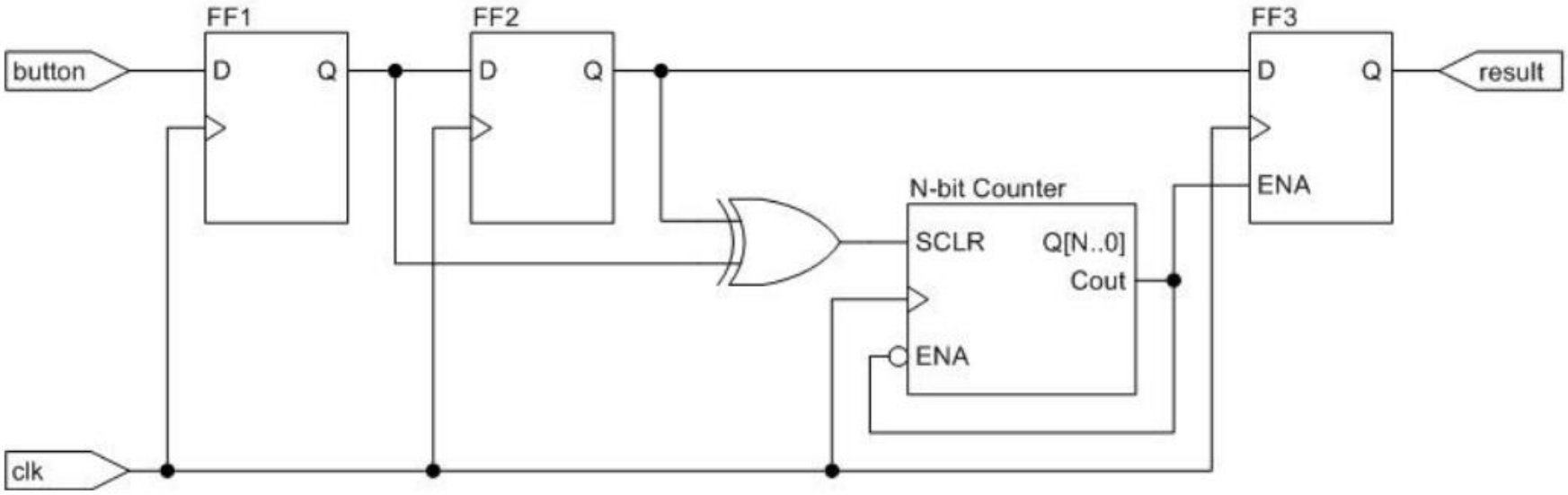
Debounce Logic Circuit

- The debounce circuit continuously clocks the button's logic level into FF1 and subsequently into FF2. So, FF1 and FF2 always store the last two logic levels of the button.
- When these two values remain identical for a specified time, then FF3 is enabled, and the stable value is clocked through to the result output.
- If the button's level changes, the values of FF1and FF2 differ for a clock cycle, **clearing the counter** via the XOR gate.
- If the button's level is unchanging (i.e. if FF1 and FF2 are the same logic level), then the XOR gate releases the counter's synchronous clear, and the counter begins to count.



Debounce Logic Circuit

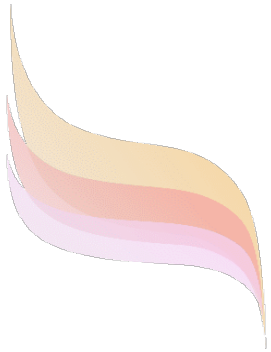
- N-bit Counter is used to determine the time required to validate the button's stability.
- When the counter increments to the point that its carry out bit is asserted, it disables itself from incrementing further and enables the output register FF3.
- The circuit remains in this state until a different button value is clocked into FF1, clearing the counter via the XOR gate.



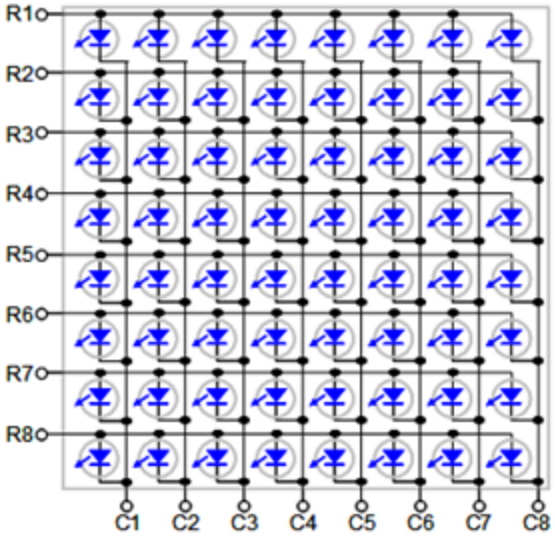
VHDL for Debounce Logic Circuit

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY debounce IS
  GENERIC(
    counter_size  :  INTEGER := 19); --counter size (19 bits gives 10.5ms with 50MHz clock)
  PORT(
    clk     : IN  STD_LOGIC;  --input clock
    button  : IN  STD_LOGIC;  --input signal to be debounced
    result  : OUT STD_LOGIC); --debounced signal
END debounce;

ARCHITECTURE logic OF debounce IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
  SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS => '0'); --counter
BEGIN                                                                              output

  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN                  --reset counter because input is changing
        counter_out <= (OTHERS => '0');
      ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                        --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

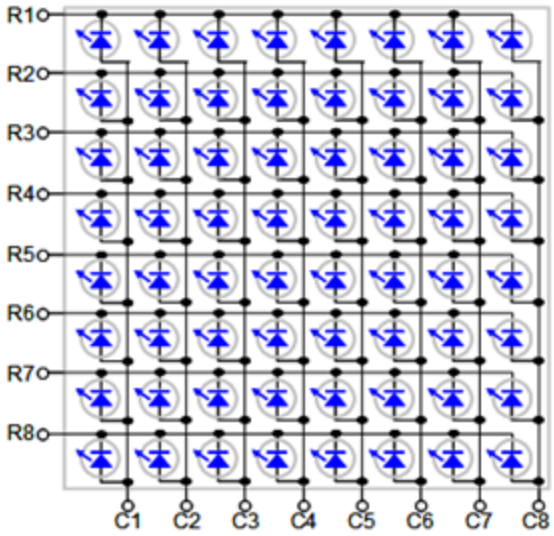
8x8 LED Matrix Control



8x8 LED Matrix

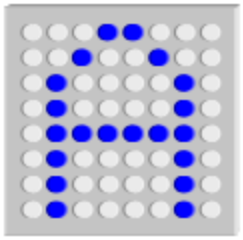


(a) 共陰型(CC)



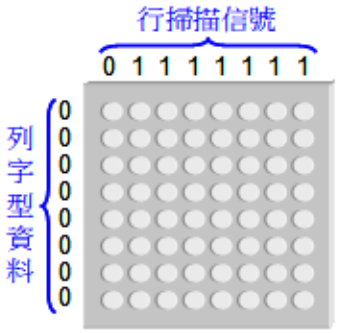
(b) 共陽型(CA)

8x8 LED Matrix – static display

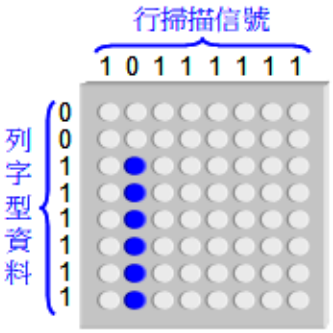


(以行掃描為例做說明)

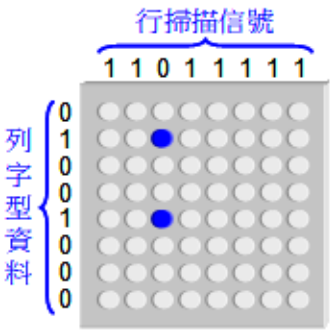
<多工掃描>



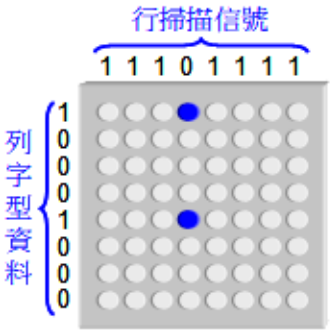
(a) 第 1 次掃描



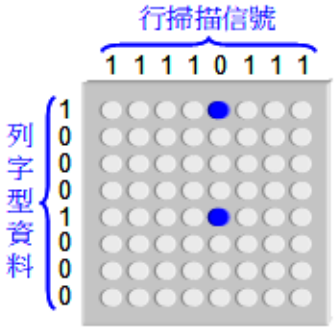
(b) 第 2 次掃描



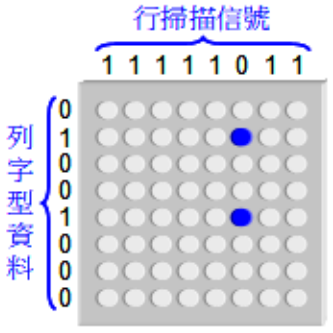
(c) 第 3 次掃描



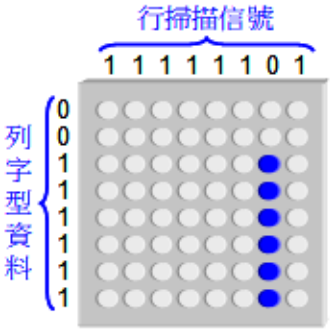
(d) 第 4 次掃描



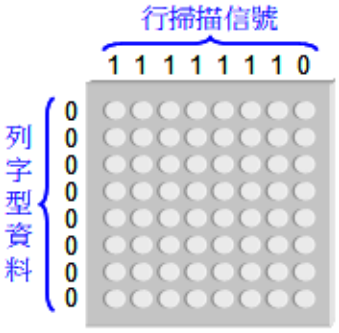
(e) 第 5 次掃描



(f) 第 6 次掃描



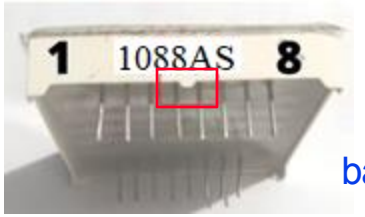
(g) 第 7 次掃描



(h) 第 8 次掃描

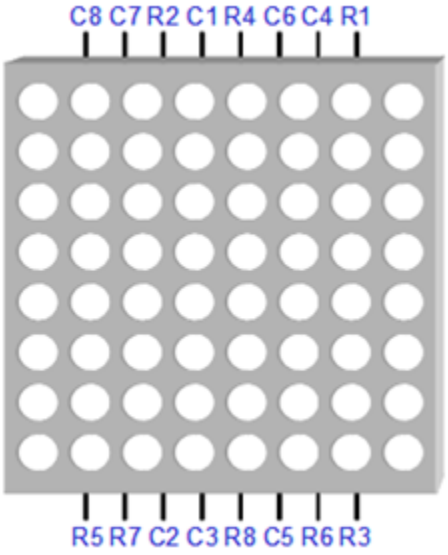
8x8 LED Matrix

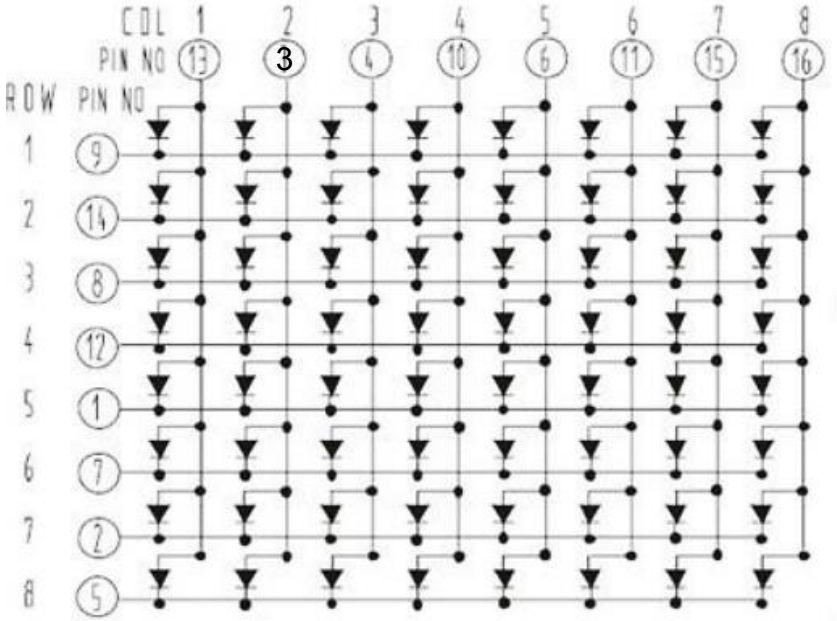
- 實驗所用為 1088AS



front-side

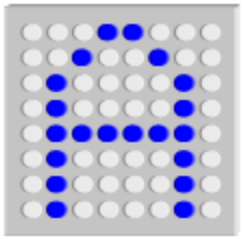
back-side





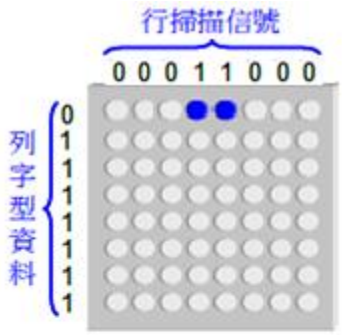
8x8 LED Matrix – static display

⇨ led8x8static.vhd

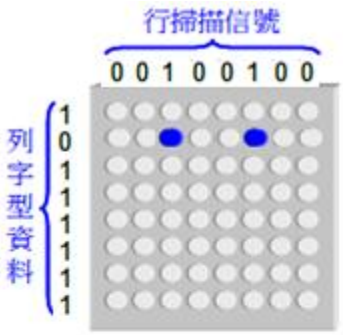


(以列掃描為例做說明)

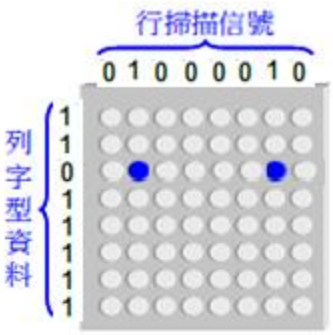
<多工掃描>



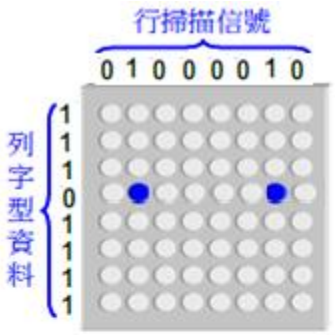
(a) 第 1 次掃描



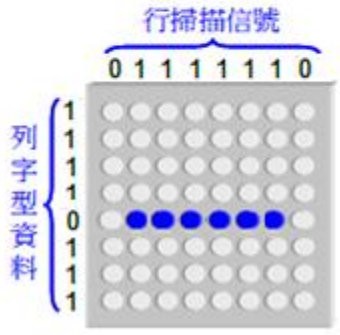
(b) 第 2 次掃描



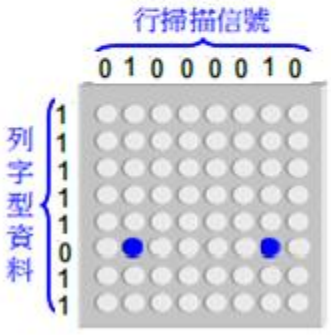
(c) 第 3 次掃描



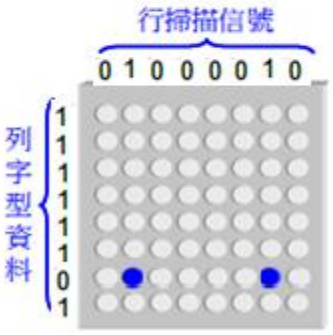
(d) 第 4 次掃描



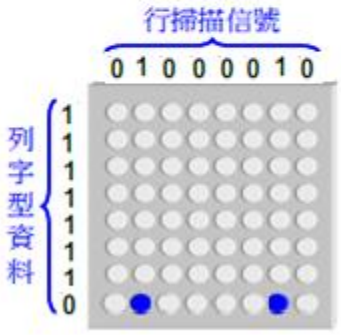
(e) 第 5 次掃描



(f) 第 6 次掃描

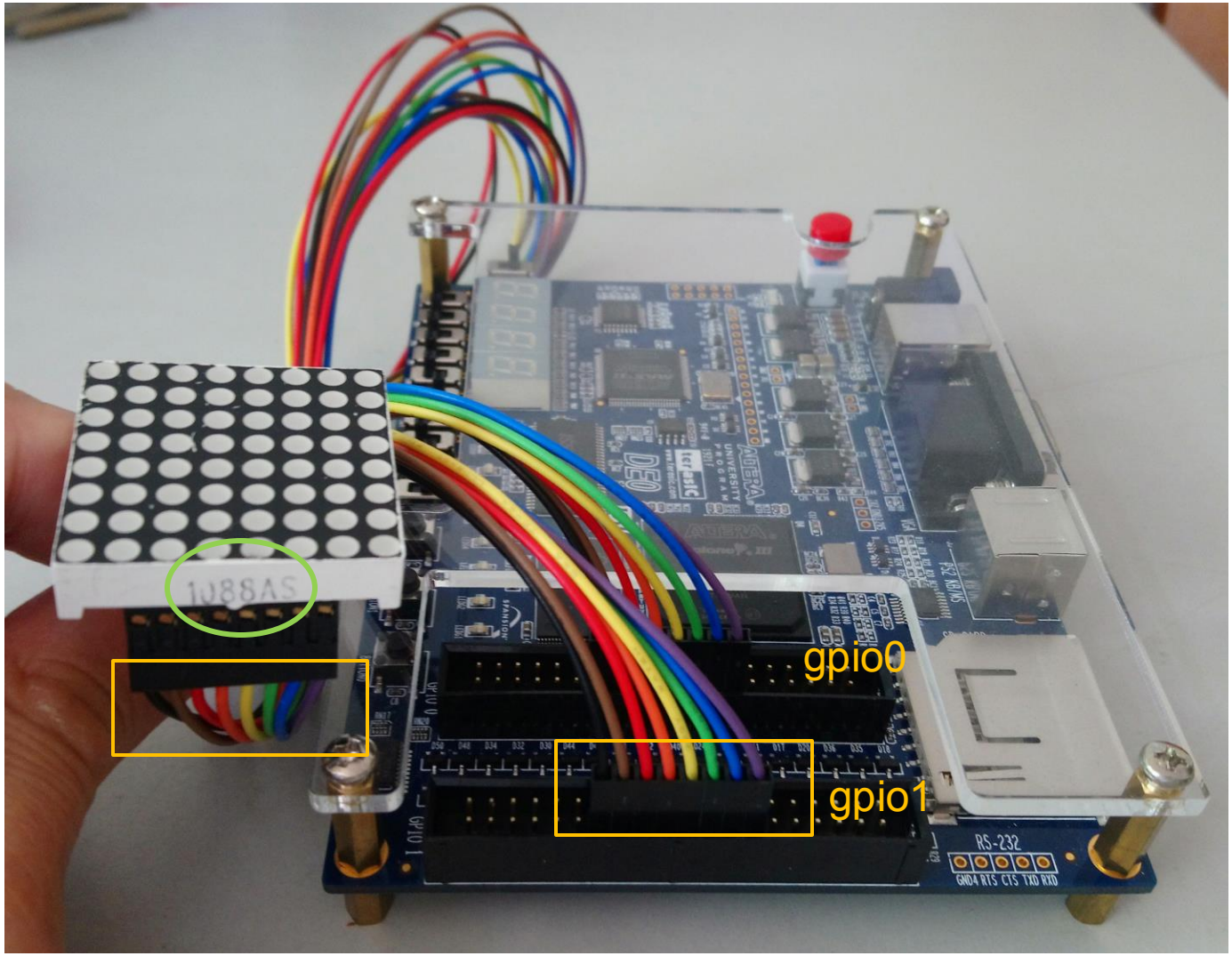


(g) 第 7 次掃描

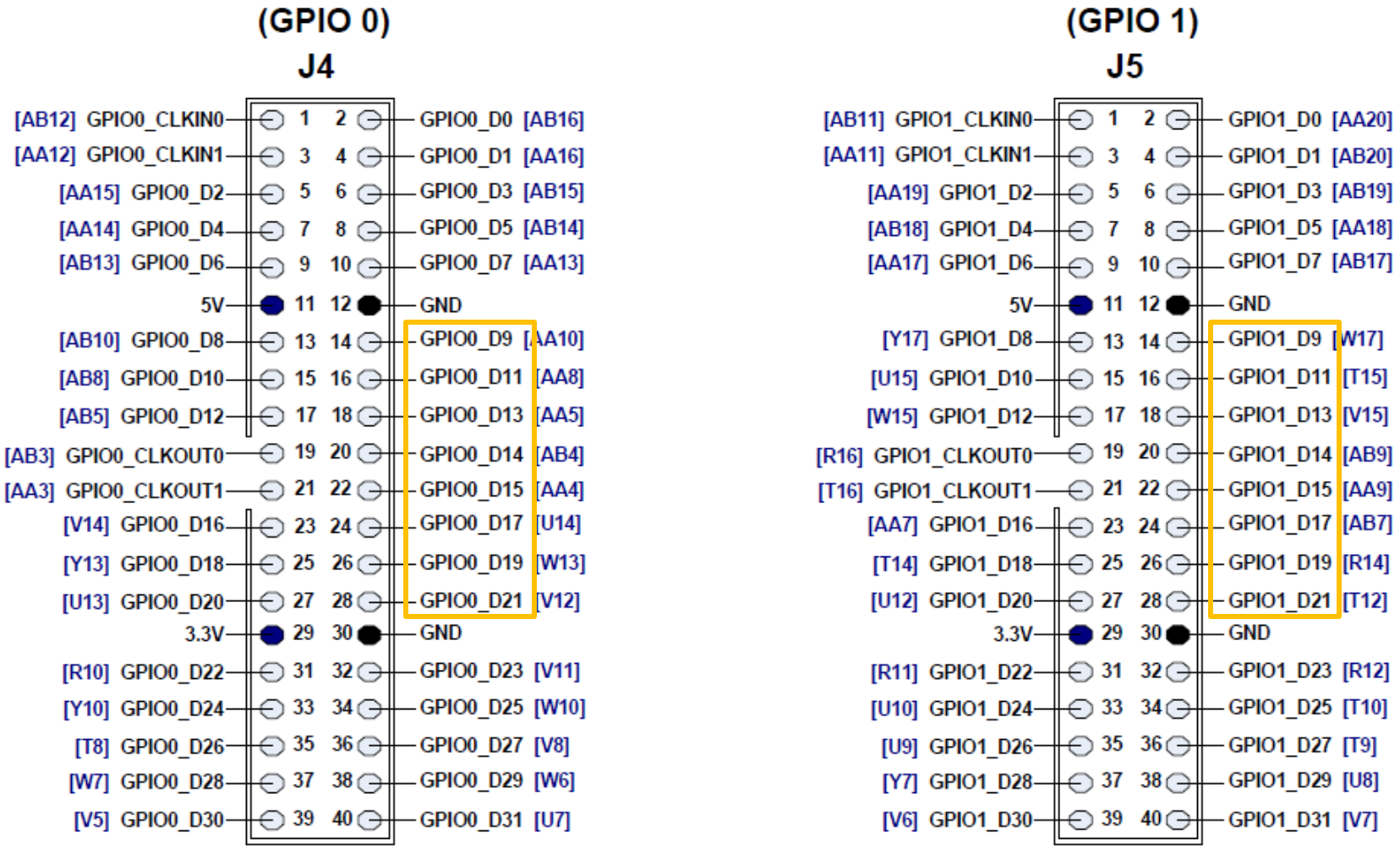


(h) 第 8 次掃描

8x8 LED Matrix – connect to DE0 GPIOs



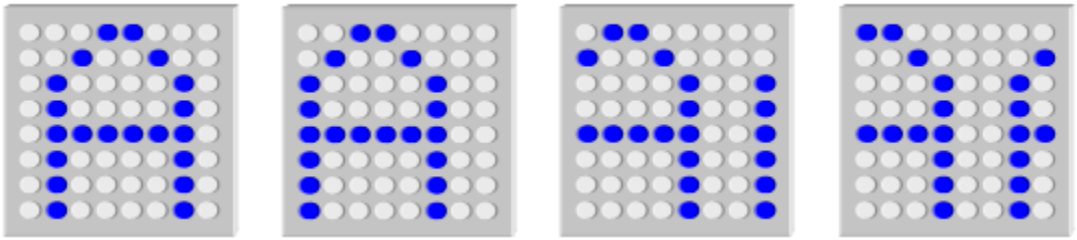
8x8 LED Matrix – connect to DE0 GPIOs



8x8 LED Matrix- dynamic display

⇨ led8x8shift.vhd

左移



右移