實驗二

Time-of-day clock

學生: 謝旻錡

學號: 313512078

日期: 2025/03/12

1. 實驗目的

熟悉 latches、flip-flop 及 regist，並以此基礎組合出 counter，並藉由 counter 的功能，實作一個循環 24 小時制的時鐘。

時鐘輸入輸出規格:

Input		Output	
SW_9	Select hour and minute	HEX5~4	Hour
SW_8	Set time (positive edge)	HEX3~2	Minute
$SW_{7\sim0}$	The value of setting time	HEX1~0	Second

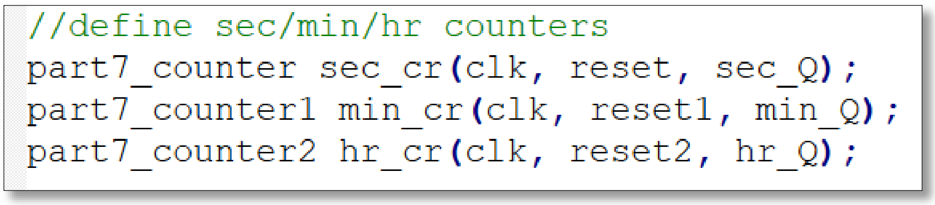
2. 實驗程式碼

STEP.1 我使用 QUARTER 的 LPM (程式碼至於附件)，定義了 second/ minute/ hour 的 counters，並配合 frequency divider 計算 second/ minute/ hour 時間。

STEP.2 計算出的值為二進制，所以需要使用 BCD 來轉換成 10 進制，這邊的 BCD 我使用 Double Dabble 演算法來轉換。

STEP.3 最後接上七段顯示器顯示時間。

● LMP_counters



```
//define sec/min/hr counters
part7_counter sec_cr(clk, reset, sec_Q);
part7_counter1 min_cr(clk, reset1, min_Q);
part7_counter2 hr_cr(clk, reset2, hr_Q);
```

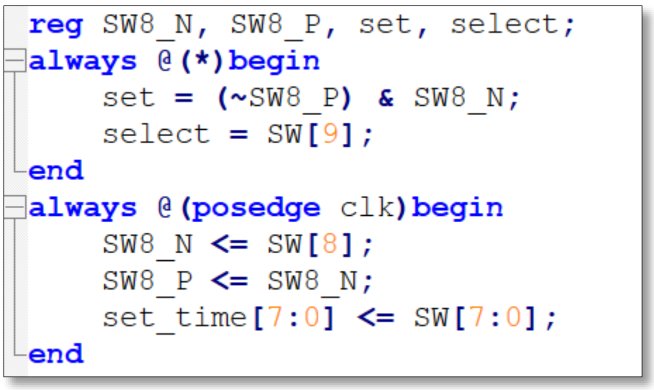
圖 1. counters

變數	內容
clk	時鐘訊號 (50MHz)
reset, reset1, reset2	counter 歸零訊號 (1: rest; 0: nothing)
sec_Q, min_Q, Hr_Q	記數數量

表格 1. Counter 變數

我定義了三個 counter，分別用於 second, minute, hour 的 clk 訊號的記數。每 20ns 記數一次(clk: 50MHz)。

● 時間設定功能



```
reg SW8_N, SW8_P, set, select;
always @(*)begin
    set = (~SW8_P) & SW8_N;
    select = SW[9];
end
always @(posedge clk)begin
    SW8_N <= SW[8];
    SW8_P <= SW8_N;
    set_time[7:0] <= SW[7:0];
end
```

圖 2. 設定時間

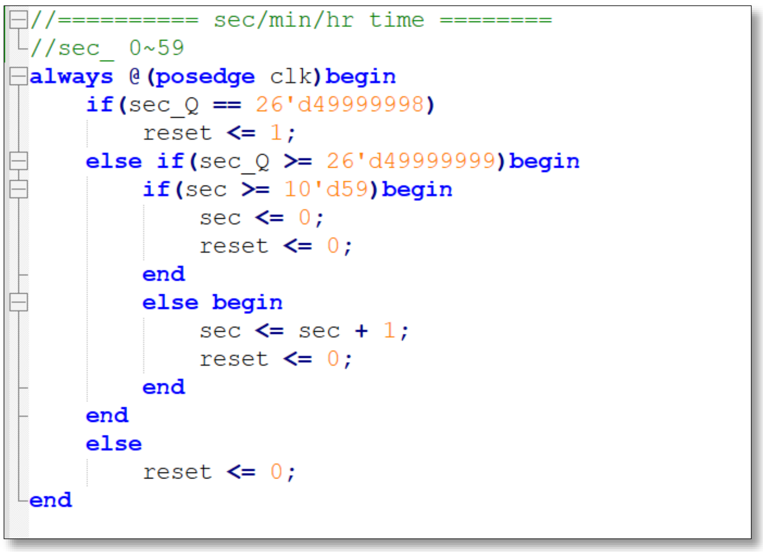
變數	內容
select	選擇設定 hour 或 minute (1: hour; 0: minute)
set	傳入設定時間 (positive edge)
set_time	欲設定的時間
SW8_N, SW8_P	SW8 開關的現在值與上一個 clk 的開關值

表格 2. Time setting 變數

宣告一個 8bit 的 set_time 暫存器，儲存 SW[7:0]開關欲設定的時間。

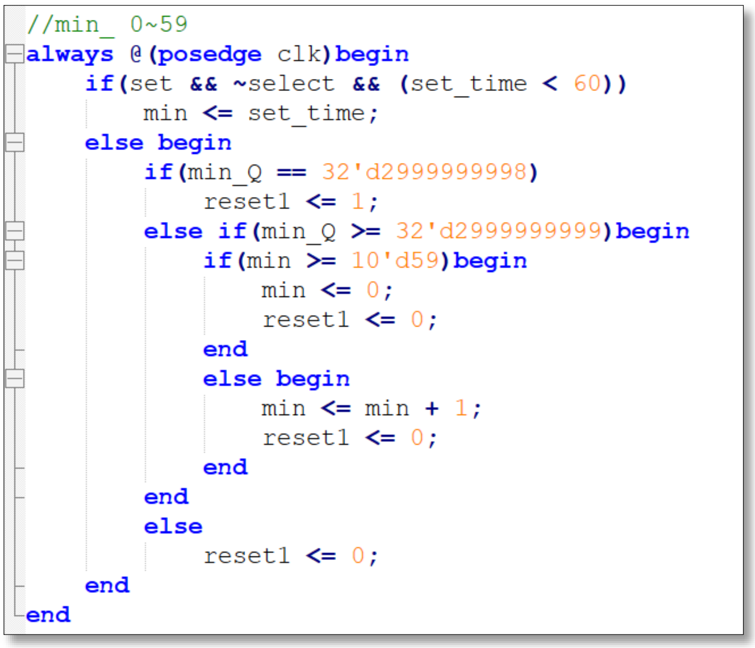
為了使 set 是以正緣觸發的方式設定時間，我寫段邏輯電路 set = (~SW8_P) & $SW8_N$; 。因此當 SW8 為 0 轉至 1 的瞬間，set 的值為 1，以此滿足正緣觸發的條件。

● 計時器 second/ minute/ hour timer



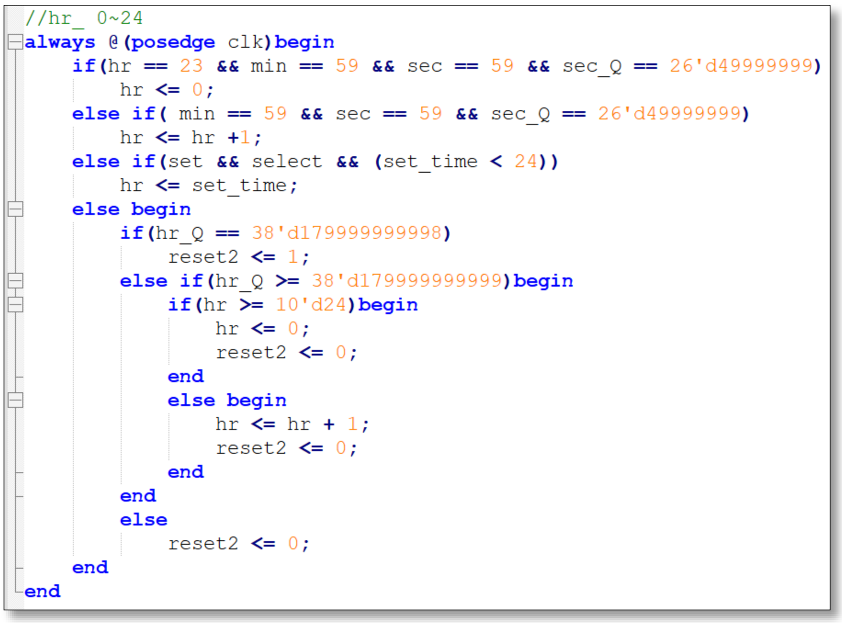
```
//========== sec/min/hr time ========
//sec_ 0~59
always @(posedge clk)begin
    if(sec_Q == 26'd49999998)
        reset <= 1;
    else if(sec_Q >= 26'd49999999)begin
        if(sec >= 10'd59)begin
            sec <= 0;
            reset <= 0;
        end
        else begin
            sec <= sec + 1;
            reset <= 0;
        end
    end
    else
        reset <= 0;
end
```

圖 3. Second timer



```
//min_ 0~59
always @(posedge clk)begin
    if(set && ~select && (set_time < 60))
        min <= set_time;
    else begin
        if(min_Q == 32'd2999999998)
            reset1 <= 1;
        else if(min_Q >= 32'd2999999999)begin
            if(min >= 10'd59)begin
                min <= 0;
                reset1 <= 0;
            end
            else begin
                min <= min + 1;
                reset1 <= 0;
            end
        end
        else
            reset1 <= 0;
    end
end
```

圖 4. Minute timer



```
//hr_ 0~24
always @(posedge clk)begin
    if(hr == 23 && min == 59 && sec == 59 && sec_Q == 26'd49999999)
        hr <= 0;
    else if( min == 59 && sec == 59 && sec_Q == 26'd49999999)
        hr <= hr +1;
    else if(set && select && (set_time < 24))
        hr <= set_time;
    else begin
        if(hr_Q == 38'd179999999998)
            reset2 <= 1;
        else if(hr_Q >= 38'd179999999999)begin
            if(hr >= 10'd24)begin
                hr <= 0;
                reset2 <= 0;
            end
            else begin
                hr <= hr + 1;
                reset2 <= 0;
            end
        end
        else
            reset2 <= 0;
    end
end
```

圖 5. Hour timer

變數	內容
clk	時鐘訊號 (50MHz)
reset, reset1, reset2	counter 歸零訊號 (1: rest; 0: nothing)
sec_Q, min_Q, hr_Q	記數數量
sec, min, hr	計時數值
set_time	欲設定的時間
set	傳入設定時間
select	選擇設定 hour 或 minute (1: hour; 0: minute)

表格 3. Timer 變數

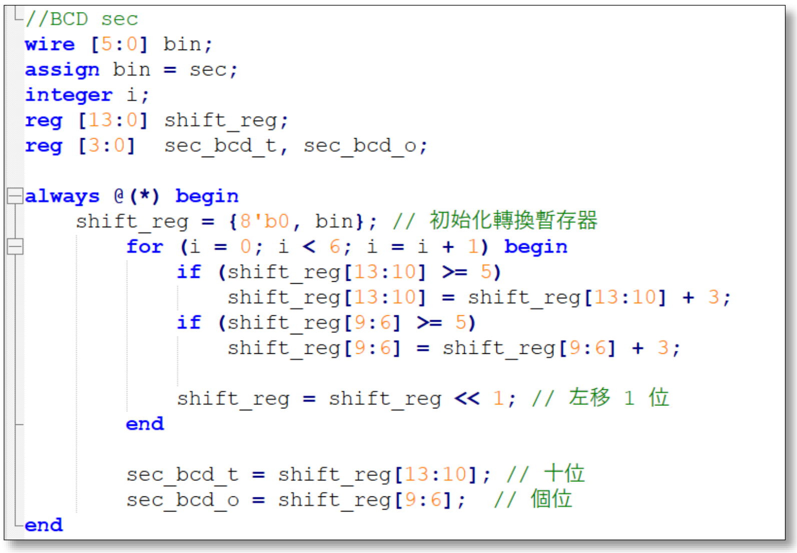
分別設定三個計時器來對 second/ minute/ hour 計時。

Second: 每當 counter 數了5×10^7次，為一秒。如果當 sec 數了 59 秒時，sec 下一秒歸零。

Minute: 每當 counter 數了3×10^9次，為一分鐘。當 min 數了 59 秒，sec 數了 59 秒時，下一秒 min 歸零。

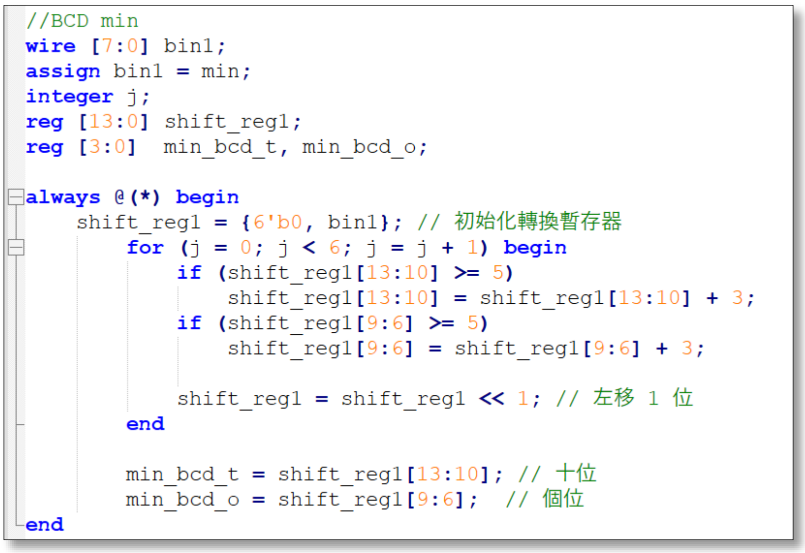
Hour: 每當 counter 數了2×10^{11}次，為一小時。當 hr 數了 23 小時，min 數了 59 秒，sec 數了 59 秒時，下一秒 hr 歸零。

● BCD



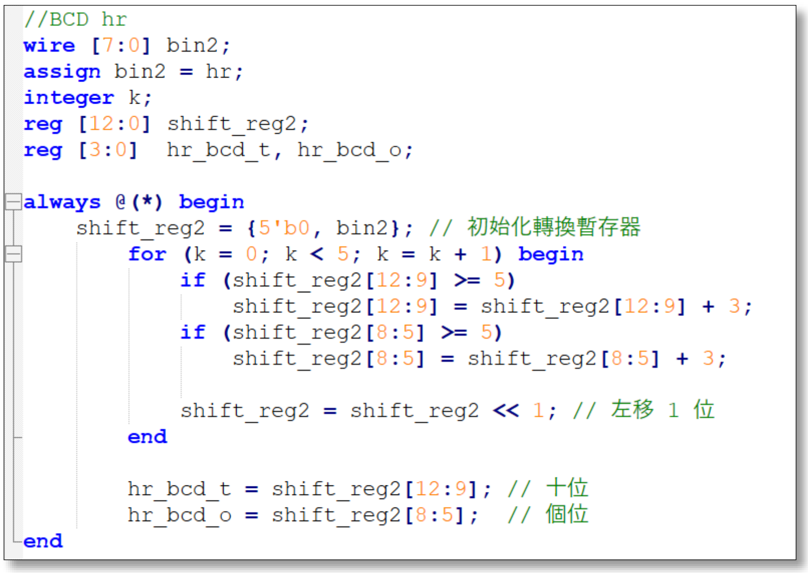
```
//BCD sec
wire [5:0] bin;
assign bin = sec;
integer i;
reg [13:0] shift_reg;
reg [3:0]  sec_bcd_t, sec_bcd_o;

always @(*) begin
    shift_reg = {8'b0, bin}; // 初始化轉換暫存器
        for (i = 0; i < 6; i = i + 1) begin
            if (shift_reg[13:10] >= 5)
                shift_reg[13:10] = shift_reg[13:10] + 3;
            if (shift_reg[9:6] >= 5)
                shift_reg[9:6] = shift_reg[9:6] + 3;

            shift_reg = shift_reg << 1; // 左移 1 位
        end

        sec_bcd_t = shift_reg[13:10]; // 十位
        sec_bcd_o = shift_reg[9:6];  // 個位
end
```

圖 6. BCD of second



```
//BCD min
wire [7:0] bin1;
assign bin1 = min;
integer j;
reg [13:0] shift_reg1;
reg [3:0]  min_bcd_t, min_bcd_o;

always @(*) begin
    shift_reg1 = {6'b0, bin1}; // 初始化轉換暫存器
        for (j = 0; j < 6; j = j + 1) begin
            if (shift_reg1[13:10] >= 5)
                shift_reg1[13:10] = shift_reg1[13:10] + 3;
            if (shift_reg1[9:6] >= 5)
                shift_reg1[9:6] = shift_reg1[9:6] + 3;

            shift_reg1 = shift_reg1 << 1; // 左移 1 位
        end

        min_bcd_t = shift_reg1[13:10]; // 十位
        min_bcd_o = shift_reg1[9:6];  // 個位
end
```

圖 7. BCD of minute



```
//BCD hr
wire [7:0] bin2;
assign bin2 = hr;
integer k;
reg [12:0] shift_reg2;
reg [3:0]  hr_bcd_t, hr_bcd_o;

always @(*) begin
    shift_reg2 = {5'b0, bin2}; // 初始化轉換暫存器
        for (k = 0; k < 5; k = k + 1) begin
            if (shift_reg2[12:9] >= 5)
                shift_reg2[12:9] = shift_reg2[12:9] + 3;
            if (shift_reg2[8:5] >= 5)
                shift_reg2[8:5] = shift_reg2[8:5] + 3;

            shift_reg2 = shift_reg2 << 1; // 左移 1 位
        end

        hr_bcd_t = shift_reg2[12:9]; // 十位
        hr_bcd_o = shift_reg2[8:5];  // 個位
end
```

圖 8. BCD of hour

變數	內容
bin, bin1, bin2	儲存 sec/ min/ hr 的暫存器
shift_reg	用於 sec 的 BCD 計算的暫存空間
shift_reg1	用於 min 的 BCD 計算的暫存空間
shift_reg2	用於 hr 的 BCD 計算的暫存空間
sec_bcd_t, sec_bcd_o	秒 轉換後的 10 進制值
min_bcd_t, min_bcd_o	分 轉換後的 10 進制值
hr_bcd_t, hr_bcd_o	時 轉換後的 10 進制值

sec/ min/ hr 裡儲存的時間為二進制形式，因此我藉由使用 double dabble 的演算法，將其轉換為二進制。

Double dabble 演算法:

遞推關係

- 初始條件（當 i=0 時）:

$$D(0) = (0,0,\ldots,0,B)$$

BCD 欄位初始化為 0。

- **主要轉換步驟（第 i 次迭代）：**

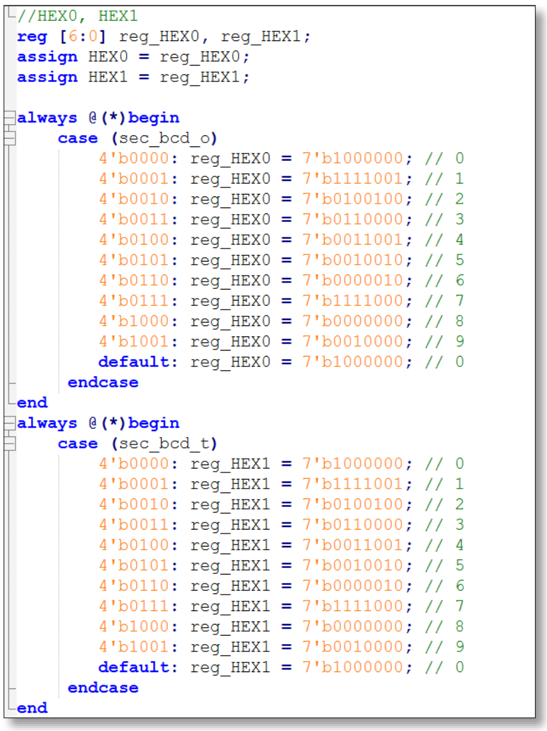
$$若\ Dj(i) \geq 5,\ 則\ Dj(i) = Dj(i) + 3, \forall j \in [0, k-1]$$

然後整體左移一位：

$$D(i+1) = (Dk-1(i) \ll 1, Dk-2(i) \ll 1, \ldots, D0(i) \ll 1, R(i) \ll 1 + bn-1-i)$$

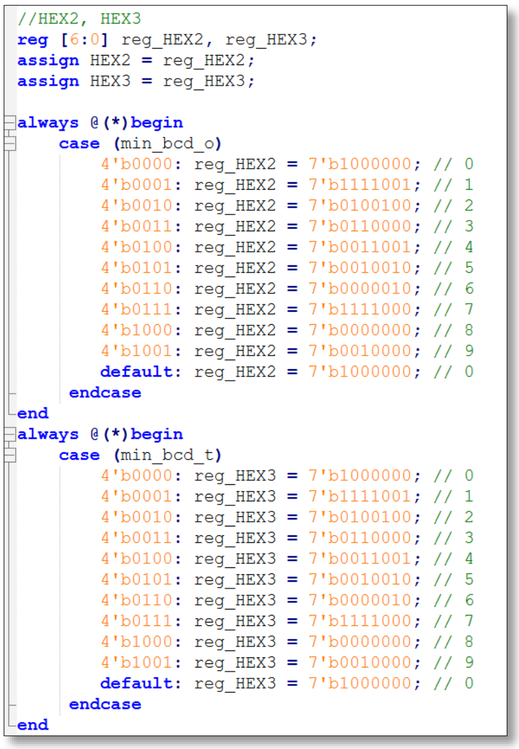
這個過程會持續 n 次，最終 D(n)即為 BCD 的結果。

● 7 段顯示器



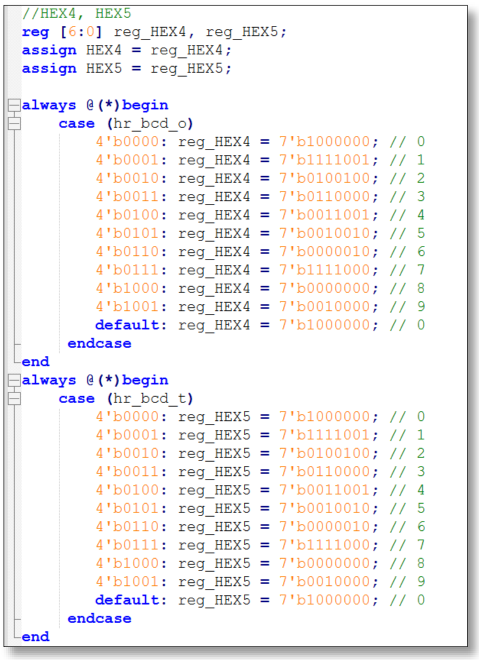
```verilog
//HEX0, HEX1
reg [6:0] reg_HEX0, reg_HEX1;
assign HEX0 = reg_HEX0;
assign HEX1 = reg_HEX1;

always @(*)begin
    case (sec_bcd_o)
        4'b0000: reg_HEX0 = 7'b1000000; // 0
        4'b0001: reg_HEX0 = 7'b1111001; // 1
        4'b0010: reg_HEX0 = 7'b0100100; // 2
        4'b0011: reg_HEX0 = 7'b0110000; // 3
        4'b0100: reg_HEX0 = 7'b0011001; // 4
        4'b0101: reg_HEX0 = 7'b0010010; // 5
        4'b0110: reg_HEX0 = 7'b0000010; // 6
        4'b0111: reg_HEX0 = 7'b1111000; // 7
        4'b1000: reg_HEX0 = 7'b0000000; // 8
        4'b1001: reg_HEX0 = 7'b0010000; // 9
        default: reg_HEX0 = 7'b1000000; // 0
    endcase
end
always @(*)begin
    case (sec_bcd_t)
        4'b0000: reg_HEX1 = 7'b1000000; // 0
        4'b0001: reg_HEX1 = 7'b1111001; // 1
        4'b0010: reg_HEX1 = 7'b0100100; // 2
        4'b0011: reg_HEX1 = 7'b0110000; // 3
        4'b0100: reg_HEX1 = 7'b0011001; // 4
        4'b0101: reg_HEX1 = 7'b0010010; // 5
        4'b0110: reg_HEX1 = 7'b0000010; // 6
        4'b0111: reg_HEX1 = 7'b1111000; // 7
        4'b1000: reg_HEX1 = 7'b0000000; // 8
        4'b1001: reg_HEX1 = 7'b0010000; // 9
        default: reg_HEX1 = 7'b1000000; // 0
    endcase
end
```

圖 9. 7 段顯示器-秒



```
//HEX2, HEX3
reg [6:0] reg_HEX2, reg_HEX3;
assign HEX2 = reg_HEX2;
assign HEX3 = reg_HEX3;

always @(*)begin
    case (min_bcd_o)
        4'b0000: reg_HEX2 = 7'b1000000; // 0
        4'b0001: reg_HEX2 = 7'b1111001; // 1
        4'b0010: reg_HEX2 = 7'b0100100; // 2
        4'b0011: reg_HEX2 = 7'b0110000; // 3
        4'b0100: reg_HEX2 = 7'b0011001; // 4
        4'b0101: reg_HEX2 = 7'b0010010; // 5
        4'b0110: reg_HEX2 = 7'b0000010; // 6
        4'b0111: reg_HEX2 = 7'b1111000; // 7
        4'b1000: reg_HEX2 = 7'b0000000; // 8
        4'b1001: reg_HEX2 = 7'b0010000; // 9
        default: reg_HEX2 = 7'b1000000; // 0
     endcase
end
always @(*)begin
    case (min_bcd_t)
        4'b0000: reg_HEX3 = 7'b1000000; // 0
        4'b0001: reg_HEX3 = 7'b1111001; // 1
        4'b0010: reg_HEX3 = 7'b0100100; // 2
        4'b0011: reg_HEX3 = 7'b0110000; // 3
        4'b0100: reg_HEX3 = 7'b0011001; // 4
        4'b0101: reg_HEX3 = 7'b0010010; // 5
        4'b0110: reg_HEX3 = 7'b0000010; // 6
        4'b0111: reg_HEX3 = 7'b1111000; // 7
        4'b1000: reg_HEX3 = 7'b0000000; // 8
        4'b1001: reg_HEX3 = 7'b0010000; // 9
        default: reg_HEX3 = 7'b1000000; // 0
     endcase
end
```

圖 10. 段顯示器-分



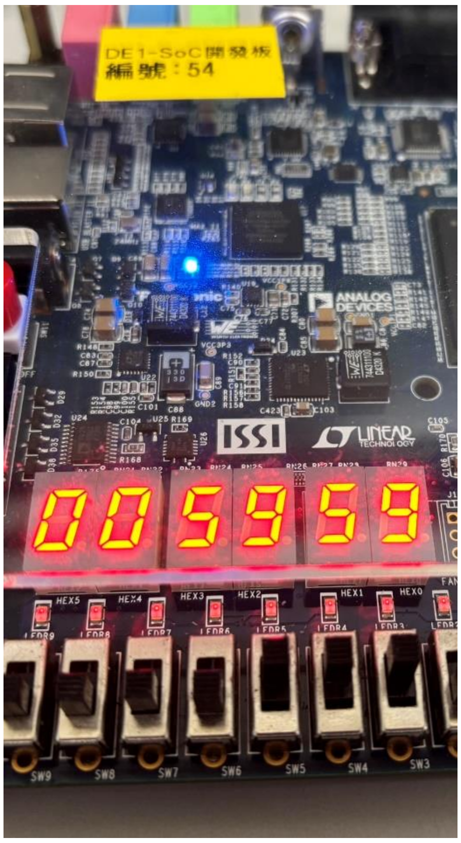
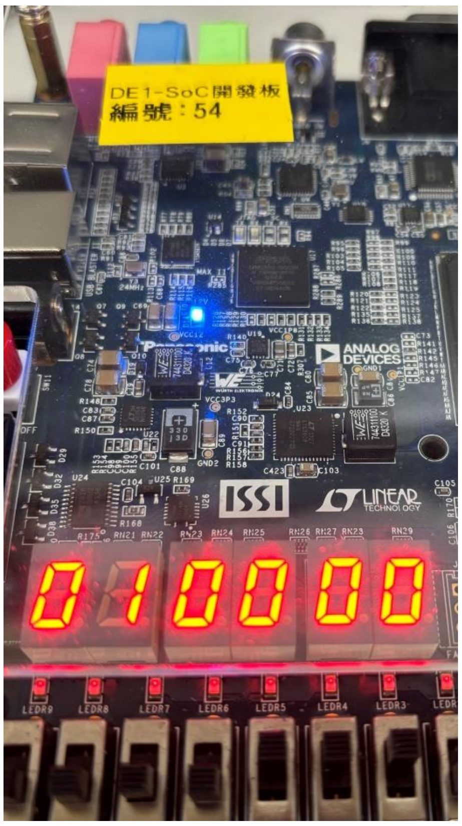
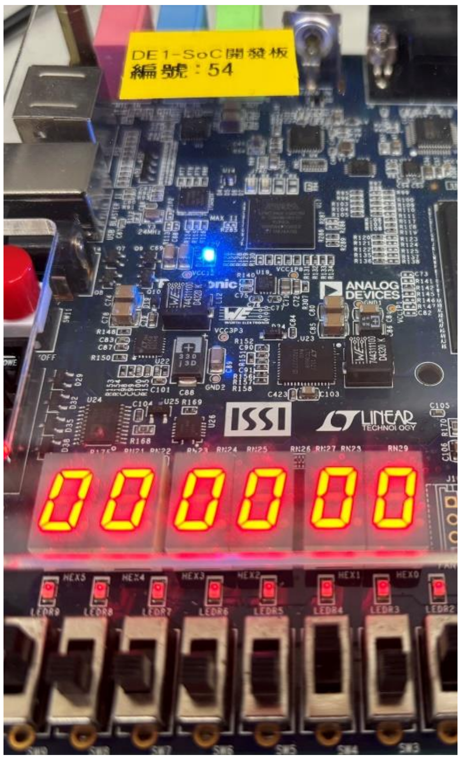
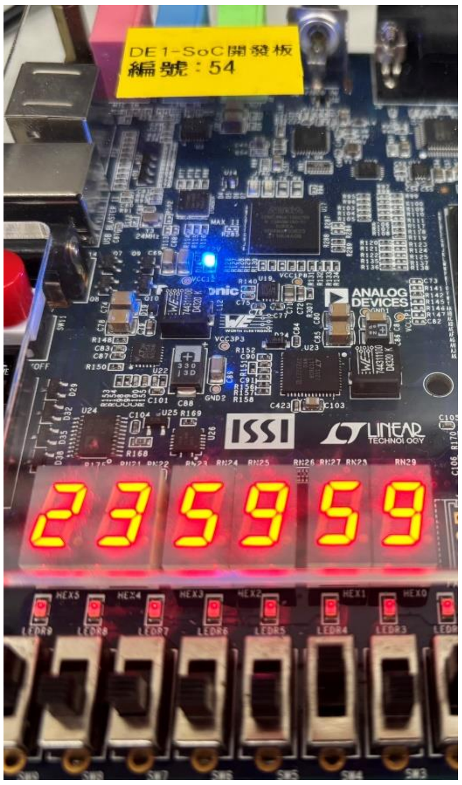
```
//HEX4, HEX5
reg [6:0] reg_HEX4, reg_HEX5;
assign HEX4 = reg_HEX4;
assign HEX5 = reg_HEX5;

always @(*)begin
    case (hr_bcd_o)
        4'b0000: reg_HEX4 = 7'b1000000; // 0
        4'b0001: reg_HEX4 = 7'b1111001; // 1
        4'b0010: reg_HEX4 = 7'b0100100; // 2
        4'b0011: reg_HEX4 = 7'b0110000; // 3
        4'b0100: reg_HEX4 = 7'b0011001; // 4
        4'b0101: reg_HEX4 = 7'b0010010; // 5
        4'b0110: reg_HEX4 = 7'b0000010; // 6
        4'b0111: reg_HEX4 = 7'b1111000; // 7
        4'b1000: reg_HEX4 = 7'b0000000; // 8
        4'b1001: reg_HEX4 = 7'b0010000; // 9
        default: reg_HEX4 = 7'b1000000; // 0
     endcase
end
always @(*)begin
    case (hr_bcd_t)
        4'b0000: reg_HEX5 = 7'b1000000; // 0
        4'b0001: reg_HEX5 = 7'b1111001; // 1
        4'b0010: reg_HEX5 = 7'b0100100; // 2
        4'b0011: reg_HEX5 = 7'b0110000; // 3
        4'b0100: reg_HEX5 = 7'b0011001; // 4
        4'b0101: reg_HEX5 = 7'b0010010; // 5
        4'b0110: reg_HEX5 = 7'b0000010; // 6
        4'b0111: reg_HEX5 = 7'b1111000; // 7
        4'b1000: reg_HEX5 = 7'b0000000; // 8
        4'b1001: reg_HEX5 = 7'b0010000; // 9
        default: reg_HEX5 = 7'b1000000; // 0
     endcase
end
```

圖 11. 段顯示器-時

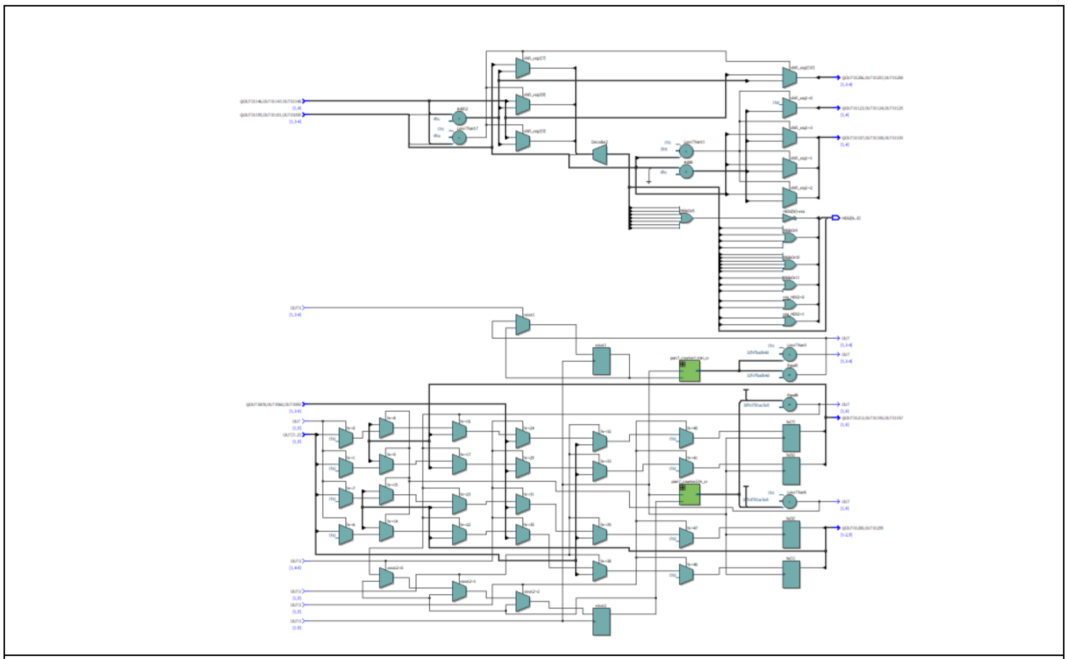
變數	內容
HEX0, HEX1	秒的 10 進制顯示

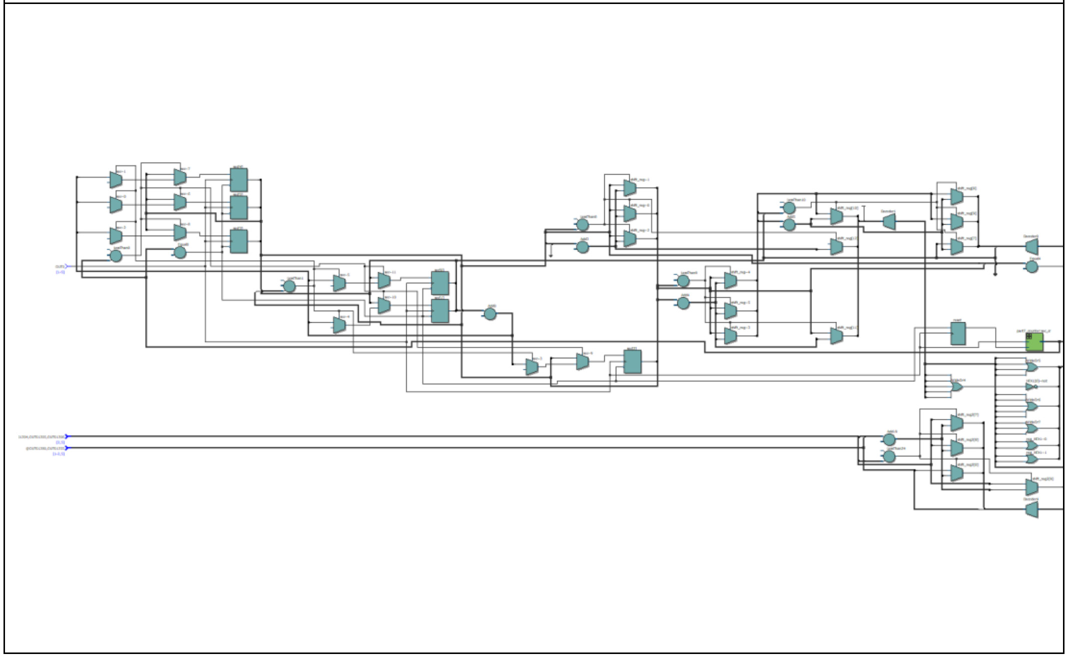
HEX2, HEX3	分的 10 進制顯示
HEX4, HEX5	時的 10 進制顯示

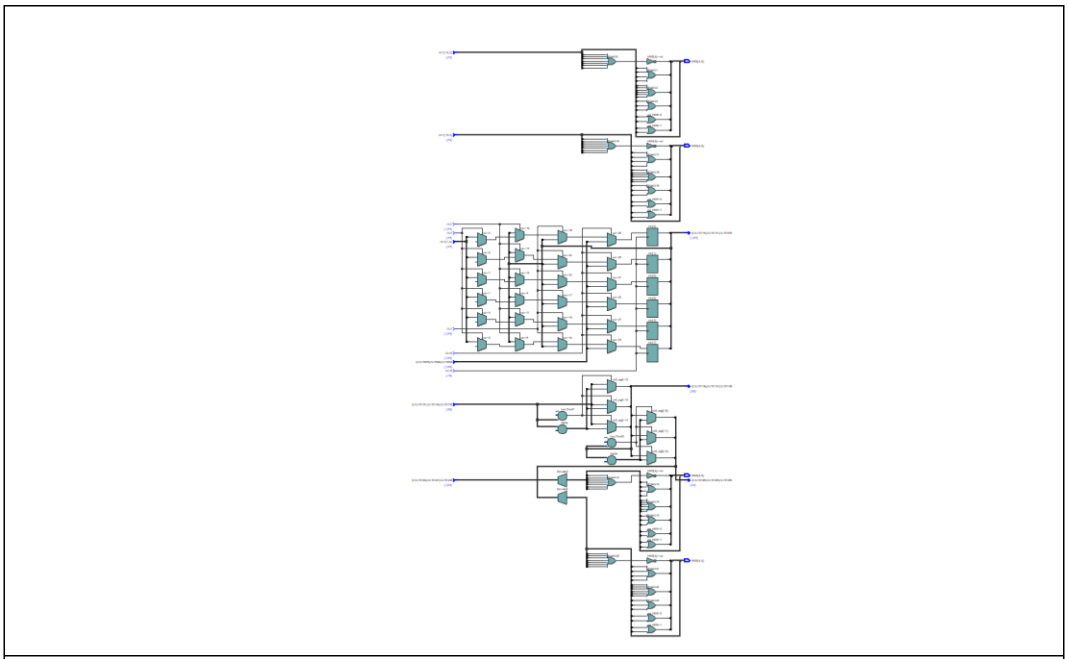
3. 實驗結果照片(optional)

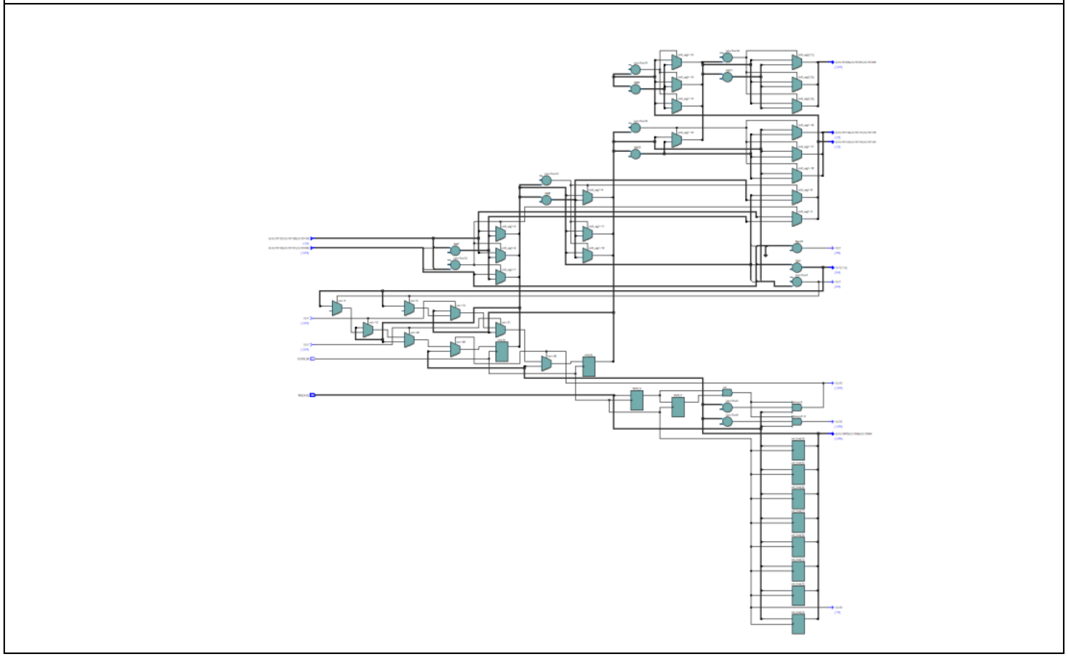
	
	

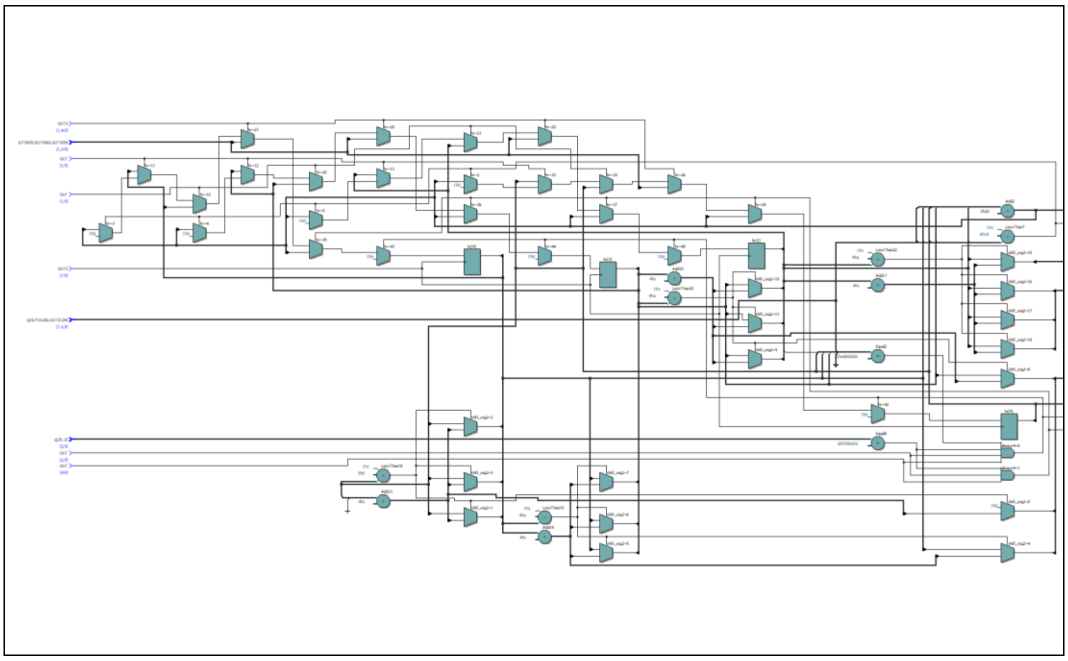
4. RTL 布局(optional)











5. 問題與討論

1. RTL 線路過大，我認為是因為我設計時，使用了三個 counters。可以改成只用一個 counter 來實作，減少電路量。
2. 如果改用一個 counter 製作，我會把 timer 在單獨拉出一個獨立的模組。這樣可以在修改時，簡潔易點。

6. 附件

/*part7 day clock*/

```
module part7 (SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, CLOCK_50 );

input CLOCK_50;

input [9:0] SW;

output [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;


reg reset, reset1, reset2;

wire clock;

wire [25:0] sec_Q;

wire [31:0] min_Q;

wire [37:0] hr_Q;

reg [5:0] sec;

reg [7:0] min;

reg [7:0] hr;

reg [7:0] set_time; //二進制


assign clk = CLOCK_50;



//define sec/min/hr counters

part7_counter sec_cr(clk, reset, sec_Q);

part7_counter1 min_cr(clk, reset1, min_Q);

part7_counter2 hr_cr(clk, reset2, hr_Q);
```

```
//SW
reg SW8_N, SW8_P, set, select;
always @(*)begin
    set = (~SW8_P) & SW8_N;
    select = SW[9];
end
always @(posedge clk)begin
    SW8_N <= SW[8];
    SW8_P <= SW8_N;
    set_time[7:0] <= SW[7:0];
end


//========== sec/min/hr time ========
//sec_ 0~59
always @(posedge clk)begin
    if(sec_Q == 26'd49999998)
        reset <= 1;
    else if(sec_Q >= 26'd49999999)begin
        if(sec >= 10'd59)begin
            sec <= 0;
            reset <= 0;
        end
        else begin
```

```
                sec <= sec + 1;
                reset <= 0;
            end
        end
        else
            reset <= 0;
    end

    //min_ 0~59
    always @(posedge clk)begin
        if(set && ~select && (set_time < 60))
            min <= set_time;
        else begin
            if(min_Q == 32'd2999999998)
                reset1 <= 1;
            else if(min_Q >= 32'd2999999999)begin
                if(min >= 10'd59)begin
                    min <= 0;
                    reset1 <= 0;
                end
                else begin
                    min <= min + 1;
                    reset1 <= 0;
                end
            end
```

```
                else
                        reset1 <= 0;
        end
end


//hr_ 0~24
always @(posedge clk)begin
        if(hr == 23 && min == 59 && sec == 59 && sec_Q == 26'd49999999)
                hr <= 0;
        else if( min == 59 && sec == 59 && sec_Q == 26'd49999999)
                hr <= hr +1;
        else if(set && select && (set_time < 24))
                hr <= set_time;
        else begin
                if(hr_Q == 38'd179999999998)
                        reset2 <= 1;
                else if(hr_Q >= 38'd179999999999)begin
                        if(hr >= 10'd24)begin
                                hr <= 0;
                                reset2 <= 0;
                        end
                        else begin
                                hr <= hr + 1;
                                reset2 <= 0;
                        end
```

```
            end
            else
                  reset2 <= 0;
      end
end



//==================== BCD =========================
//BCD sec
wire [5:0] bin;
assign bin = sec;
integer i;
reg [13:0] shift_reg;
reg [3:0]    sec_bcd_t, sec_bcd_o;

always @(*) begin
      shift_reg = {8'b0, bin}; // 初始化轉換暫存器
            for (i = 0; i < 6; i = i + 1) begin
                  if (shift_reg[13:10] >= 5)
                        shift_reg[13:10] = shift_reg[13:10] + 3;
                  if (shift_reg[9:6] >= 5)
                        shift_reg[9:6] = shift_reg[9:6] + 3;

                  shift_reg = shift_reg << 1; // 左移 1 位
            end
            sec_bcd_t = shift_reg[13:10]; //  十位
            sec_bcd_o = shift_reg[9:6];    //  個位
end
//BCD min
wire [7:0] bin1;
assign bin1 = min;
integer j;
reg [13:0] shift_reg1;
reg [3:0]    min_bcd_t, min_bcd_o;

always @(*) begin
      shift_reg1 = {6'b0, bin1}; //  初始化轉換暫存器
            for (j = 0; j < 6; j = j + 1) begin
                  if (shift_reg1[13:10] >= 5)
                        shift_reg1[13:10] = shift_reg1[13:10] + 3;
                  if (shift_reg1[9:6] >= 5)
                        shift_reg1[9:6] = shift_reg1[9:6] + 3;

                  shift_reg1 = shift_reg1 << 1; //  左移  1  位
            end

            min_bcd_t = shift_reg1[13:10]; //  十位
            min_bcd_o = shift_reg1[9:6];    //  個位
end
```

```
//BCD hr
wire [7:0] bin2;
assign bin2 = hr;
integer k;
reg [12:0] shift_reg2;
reg [3:0]   hr_bcd_t, hr_bcd_o;

always @(*) begin
        shift_reg2 = {5'b0, bin2}; //  初始化轉換暫存器
                for (k = 0; k < 5; k = k + 1) begin
                        if (shift_reg2[12:9] >= 5)
                                shift_reg2[12:9] = shift_reg2[12:9] + 3;
                        if (shift_reg2[8:5] >= 5)
                                shift_reg2[8:5] = shift_reg2[8:5] + 3;

                        shift_reg2 = shift_reg2 << 1; //  左移  1  位
                end

                hr_bcd_t = shift_reg2[12:9]; //  十位
                hr_bcd_o = shift_reg2[8:5];    //  個位
end

//================ HEX5~0 ========================
//HEX0, HEX1
reg [6:0] reg_HEX0, reg_HEX1;
```

```
assign HEX0 = reg_HEX0;

assign HEX1 = reg_HEX1;


always @(*)begin

        case (sec_bcd_o)

                4'b0000: reg_HEX0 = 7'b1000000; // 0

                4'b0001: reg_HEX0 = 7'b1111001; // 1

                4'b0010: reg_HEX0 = 7'b0100100; // 2

                4'b0011: reg_HEX0 = 7'b0110000; // 3

                4'b0100: reg_HEX0 = 7'b0011001; // 4

                4'b0101: reg_HEX0 = 7'b0010010; // 5

                4'b0110: reg_HEX0 = 7'b0000010; // 6

                4'b0111: reg_HEX0 = 7'b1111000; // 7

                4'b1000: reg_HEX0 = 7'b0000000; // 8

                4'b1001: reg_HEX0 = 7'b0010000; // 9

                default: reg_HEX0 = 7'b1000000; // 0

          endcase

end

always @(*)begin

        case (sec_bcd_t)

                4'b0000: reg_HEX1 = 7'b1000000; // 0

                4'b0001: reg_HEX1 = 7'b1111001; // 1

                4'b0010: reg_HEX1 = 7'b0100100; // 2

                4'b0011: reg_HEX1 = 7'b0110000; // 3

                4'b0100: reg_HEX1 = 7'b0011001; // 4
                4'b0101: reg_HEX1 = 7'b0010010; // 5
                4'b0110: reg_HEX1 = 7'b0000010; // 6
                4'b0111: reg_HEX1 = 7'b1111000; // 7
                4'b1000: reg_HEX1 = 7'b0000000; // 8
                4'b1001: reg_HEX1 = 7'b0010000; // 9
                default: reg_HEX1 = 7'b1000000; // 0
            endcase
end
//HEX2, HEX3
reg [6:0] reg_HEX2, reg_HEX3;
assign HEX2 = reg_HEX2;
assign HEX3 = reg_HEX3;

always @(*)begin
        case (min_bcd_o)
                4'b0000: reg_HEX2 = 7'b1000000; // 0
                4'b0001: reg_HEX2 = 7'b1111001; // 1
                4'b0010: reg_HEX2 = 7'b0100100; // 2
                4'b0011: reg_HEX2 = 7'b0110000; // 3
                4'b0100: reg_HEX2 = 7'b0011001; // 4
                4'b0101: reg_HEX2 = 7'b0010010; // 5
                4'b0110: reg_HEX2 = 7'b0000010; // 6
                4'b0111: reg_HEX2 = 7'b1111000; // 7
                4'b1000: reg_HEX2 = 7'b0000000; // 8
                4'b1001: reg_HEX2 = 7'b0010000; // 9
                default: reg_HEX2 = 7'b1000000; // 0
            endcase
    end
    always @(*)begin
            case (min_bcd_t)
                4'b0000: reg_HEX3 = 7'b1000000; // 0
                4'b0001: reg_HEX3 = 7'b1111001; // 1
                4'b0010: reg_HEX3 = 7'b0100100; // 2
                4'b0011: reg_HEX3 = 7'b0110000; // 3
                4'b0100: reg_HEX3 = 7'b0011001; // 4
                4'b0101: reg_HEX3 = 7'b0010010; // 5
                4'b0110: reg_HEX3 = 7'b0000010; // 6
                4'b0111: reg_HEX3 = 7'b1111000; // 7
                4'b1000: reg_HEX3 = 7'b0000000; // 8
                4'b1001: reg_HEX3 = 7'b0010000; // 9
                default: reg_HEX3 = 7'b1000000; // 0
            endcase
    end
    //HEX4, HEX5
    reg [6:0] reg_HEX4, reg_HEX5;
    assign HEX4 = reg_HEX4;
    assign HEX5 = reg_HEX5;


    always @(*)begin
            case (hr_bcd_o)
```

```
                4'b0000: reg_HEX4 = 7'b1000000; // 0
                4'b0001: reg_HEX4 = 7'b1111001; // 1
                4'b0010: reg_HEX4 = 7'b0100100; // 2
                4'b0011: reg_HEX4 = 7'b0110000; // 3
                4'b0100: reg_HEX4 = 7'b0011001; // 4
                4'b0101: reg_HEX4 = 7'b0010010; // 5
                4'b0110: reg_HEX4 = 7'b0000010; // 6
                4'b0111: reg_HEX4 = 7'b1111000; // 7
                4'b1000: reg_HEX4 = 7'b0000000; // 8
                4'b1001: reg_HEX4 = 7'b0010000; // 9
                default: reg_HEX4 = 7'b1000000; // 0
            endcase
      end
      always @(*)begin
           case (hr_bcd_t)
                4'b0000: reg_HEX5 = 7'b1000000; // 0
                4'b0001: reg_HEX5 = 7'b1111001; // 1
                4'b0010: reg_HEX5 = 7'b0100100; // 2
                4'b0011: reg_HEX5 = 7'b0110000; // 3
                4'b0100: reg_HEX5 = 7'b0011001; // 4
                4'b0101: reg_HEX5 = 7'b0010010; // 5
                4'b0110: reg_HEX5 = 7'b0000010; // 6
                4'b0111: reg_HEX5 = 7'b1111000; // 7
                4'b1000: reg_HEX5 = 7'b0000000; // 8
                4'b1001: reg_HEX5 = 7'b0010000; // 9
                default: reg_HEX5 = 7'b1000000; // 0
            endcase
    end


endmodule
```

part7_counter.v

```
    // megafunction wizard: %LPM_COUNTER%
    // GENERATION: STANDARD
    // VERSION: WM1.0
    // MODULE: LPM_COUNTER

    //
        ============================================================
        ==========
    // File Name: part7_counter.v
    // Megafunction Name(s):
    //          LPM_COUNTER
    //
    // Simulation Library Files(s):
    //          lpm
    //
        ============================================================
        ==========
//
    ********************************************************
    ***
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.0 Build 162 10/23/2013 SJ Full Version
//
    ********************************************************
    ***


//Copyright (C) 1991-2013 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.  Please refer to the
//applicable agreement for further details.
```

```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module part7_counter (
    clock,
    sclr,
    q);

    input     clock;
    input     sclr;
    output        [25:0]    q;

    wire [25:0] sub_wire0;
    wire [25:0] q = sub_wire0[25:0];

    lpm_counter     LPM_COUNTER_component (
                .clock (clock),
                .sclr (sclr),
                .q (sub_wire0),
                .aclr (1'b0),
                .aload (1'b0),
                .aset (1'b0),
                .cin (1'b1),
                .clk_en (1'b1),
                .cnt_en (1'b1),
                .cout (),

                .data ({26{1'b0}}),

                .eq (),

                .sload (1'b0),

                .sset (1'b0),

                .updown (1'b1));

    defparam

      LPM_COUNTER_component.lpm_direction = "UP",

      LPM_COUNTER_component.lpm_port_updown                    =
    "PORT_UNUSED",

      LPM_COUNTER_component.lpm_type = "LPM_COUNTER",

      LPM_COUNTER_component.lpm_width = 26;



endmodule


//
    ============================================================
    ==========

// CNX file retrieval info

//
    ============================================================
    ==========

// Retrieval info: PRIVATE: ACLR NUMERIC "0"

// Retrieval info: PRIVATE: ALOAD NUMERIC "0"

// Retrieval info: PRIVATE: ASET NUMERIC "0"

// Retrieval info: PRIVATE: ASET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: CLK_EN NUMERIC "0"
// Retrieval info: PRIVATE: CNT_EN NUMERIC "0"
// Retrieval info: PRIVATE: CarryIn NUMERIC "0"
// Retrieval info: PRIVATE: CarryOut NUMERIC "0"
// Retrieval info: PRIVATE: Direction NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
    "Cyclone V"
// Retrieval info: PRIVATE: ModulusCounter NUMERIC "0"
// Retrieval info: PRIVATE: ModulusValue NUMERIC "0"
// Retrieval info: PRIVATE: SCLR NUMERIC "1"
// Retrieval info: PRIVATE: SLOAD NUMERIC "0"
// Retrieval info: PRIVATE: SSET NUMERIC "0"
// Retrieval info: PRIVATE: SSET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX
    STRING "0"
// Retrieval info: PRIVATE: nBit NUMERIC "26"
// Retrieval info: PRIVATE: new_diagram STRING "1"
// Retrieval info: LIBRARY: lpm lpm.lpm_components.all
// Retrieval info: CONSTANT: LPM_DIRECTION STRING "UP"
// Retrieval info: CONSTANT: LPM_PORT_UPDOWN STRING
    "PORT_UNUSED"
// Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_COUNTER"
// Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "26"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL "clock"
// Retrieval info: USED_PORT: q 0 0 26 0 OUTPUT NODEFVAL "q[25..0]"
// Retrieval info: USED_PORT: sclr 0 0 0 0 INPUT NODEFVAL "sclr"
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @sclr 0 0 0 0 sclr 0 0 0 0
// Retrieval info: CONNECT: q 0 0 26 0 @q 0 0 26 0
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter_inst.v
    FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter_bb.v TRUE
etrieval info: LIB_FILE: lpm
```

part7_counter1.v

```
    / megafunction wizard: %LPM_COUNTER%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: LPM_COUNTER


//
    ====================================================
    ==========
// File Name: part7_counter1.v
// Megafunction Name(s):
//            LPM_COUNTER
//
// Simulation Library Files(s):
//            lpm
```

```
//
    =========================================================
    ===========

//
    *******************************************************
    ***

// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!

//

// 13.1.0 Build 162 10/23/2013 SJ Full Version

//
    *******************************************************
    ***



//Copyright (C) 1991-2013 Altera Corporation

//Your use of Altera Corporation's design tools, logic functions

//and other software and tools, and its AMPP partner logic

//functions, and any output files from any of the foregoing

//(including device programming or simulation files), and any

//associated documentation or information are expressly subject

//to the terms and conditions of the Altera Program License

//Subscription Agreement, Altera MegaCore Function License

//Agreement, or other applicable license agreement, including,

//without limitation, that your use is for the sole purpose of

//programming logic devices manufactured by Altera and sold by

//Altera or its authorized distributors.    Please refer to the

//applicable agreement for further details.
```

```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module part7_counter1 (
    clock,
    sclr,
    q);

    input    clock;
    input    sclr;
    output      [31:0]   q;

    wire [31:0] sub_wire0;
    wire [31:0] q = sub_wire0[31:0];

    lpm_counter    LPM_COUNTER_component (
                .clock (clock),
                .sclr (sclr),
                .q (sub_wire0),
                .aclr (1'b0),
                .aload (1'b0),
                .aset (1'b0),
                .cin (1'b1),
                .clk_en (1'b1),

                .cnt_en (1'b1),

                .cout (),

                .data ({32{1'b0}}),

                .eq (),

                .sload (1'b0),

                .sset (1'b0),

                .updown (1'b1));

    defparam

      LPM_COUNTER_component.lpm_direction = "UP",

      LPM_COUNTER_component.lpm_port_updown                    =
    "PORT_UNUSED",

      LPM_COUNTER_component.lpm_type = "LPM_COUNTER",

      LPM_COUNTER_component.lpm_width = 32;



endmodule


//
    =====================================================
    ==========

// CNX file retrieval info

//
    =====================================================
    ==========

// Retrieval info: PRIVATE: ACLR NUMERIC "0"

// Retrieval info: PRIVATE: ALOAD NUMERIC "0"
// Retrieval info: PRIVATE: ASET NUMERIC "0"
// Retrieval info: PRIVATE: ASET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: CLK_EN NUMERIC "0"
// Retrieval info: PRIVATE: CNT_EN NUMERIC "0"
// Retrieval info: PRIVATE: CarryIn NUMERIC "0"
// Retrieval info: PRIVATE: CarryOut NUMERIC "0"
// Retrieval info: PRIVATE: Direction NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
    "Cyclone V"
// Retrieval info: PRIVATE: ModulusCounter NUMERIC "0"
// Retrieval info: PRIVATE: ModulusValue NUMERIC "0"
// Retrieval info: PRIVATE: SCLR NUMERIC "1"
// Retrieval info: PRIVATE: SLOAD NUMERIC "0"
// Retrieval info: PRIVATE: SSET NUMERIC "0"
// Retrieval info: PRIVATE: SSET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX
    STRING "0"
// Retrieval info: PRIVATE: nBit NUMERIC "32"
// Retrieval info: PRIVATE: new_diagram STRING "1"
// Retrieval info: LIBRARY: lpm lpm.lpm_components.all
// Retrieval info: CONSTANT: LPM_DIRECTION STRING "UP"
// Retrieval info: CONSTANT: LPM_PORT_UPDOWN STRING
    "PORT_UNUSED"
// Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_COUNTER"
// Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "32"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL "clock"
// Retrieval info: USED_PORT: q 0 0 32 0 OUTPUT NODEFVAL "q[31..0]"
// Retrieval info: USED_PORT: sclr 0 0 0 0 INPUT NODEFVAL "sclr"
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @sclr 0 0 0 0 sclr 0 0 0 0
// Retrieval info: CONNECT: q 0 0 32 0 @q 0 0 32 0
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter1.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter1.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter1.cmp
    FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter1.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter1_inst.v
    FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter1_bb.v
    TRUE
etrieval info: LIB_FILE: lpm
```

part7_counter2.v

```
    // megafunction wizard: %LPM_COUNTER%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: LPM_COUNTER


//
    ====================================================
    ==========
// File Name: part7_counter2.v
// Megafunction Name(s):
```

```
//		LPM_COUNTER
//
// Simulation Library Files(s):
//		lpm
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.0 Build 162 10/23/2013 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2013 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.    Please refer to the
//applicable agreement for further details.



// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module part7_counter2 (
    clock,
    sclr,
    q);

    input    clock;
    input    sclr;
    output      [31:0]    q;

    wire [31:0] sub_wire0;
    wire [31:0] q = sub_wire0[31:0];

    lpm_counter     LPM_COUNTER_component (
                .clock (clock),
                .sclr (sclr),
                .q (sub_wire0),
                .aclr (1'b0),
                .aload (1'b0),
                .aset (1'b0),
                .cin (1'b1),
                .clk_en (1'b1),
                .cnt_en (1'b1),
                .cout (),
                .data ({32{1'b0}}),
                .eq (),
                .sload (1'b0),
                .sset (1'b0),
                .updown (1'b1));
    defparam
      LPM_COUNTER_component.lpm_direction = "UP",
      LPM_COUNTER_component.lpm_port_updown                    =
    "PORT_UNUSED",
      LPM_COUNTER_component.lpm_type = "LPM_COUNTER",
      LPM_COUNTER_component.lpm_width = 32;



endmodule


//
    ====================================================
    ==========

// CNX file retrieval info
```

```
//
    =======================================================
    ===========
// Retrieval info: PRIVATE: ACLR NUMERIC "0"
// Retrieval info: PRIVATE: ALOAD NUMERIC "0"
// Retrieval info: PRIVATE: ASET NUMERIC "0"
// Retrieval info: PRIVATE: ASET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: CLK_EN NUMERIC "0"
// Retrieval info: PRIVATE: CNT_EN NUMERIC "0"
// Retrieval info: PRIVATE: CarryIn NUMERIC "0"
// Retrieval info: PRIVATE: CarryOut NUMERIC "0"
// Retrieval info: PRIVATE: Direction NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING
    "Cyclone V"
// Retrieval info: PRIVATE: ModulusCounter NUMERIC "0"
// Retrieval info: PRIVATE: ModulusValue NUMERIC "0"
// Retrieval info: PRIVATE: SCLR NUMERIC "1"
// Retrieval info: PRIVATE: SLOAD NUMERIC "0"
// Retrieval info: PRIVATE: SSET NUMERIC "0"
// Retrieval info: PRIVATE: SSET_ALL1 NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX
    STRING "0"
// Retrieval info: PRIVATE: nBit NUMERIC "32"
// Retrieval info: PRIVATE: new_diagram STRING "1"
// Retrieval info: LIBRARY: lpm lpm.lpm_components.all
// Retrieval info: CONSTANT: LPM_DIRECTION STRING "UP"
// Retrieval info: CONSTANT: LPM_PORT_UPDOWN STRING "PORT_UNUSED"
// Retrieval info: CONSTANT: LPM_TYPE STRING "LPM_COUNTER"
// Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "32"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL "clock"
// Retrieval info: USED_PORT: q 0 0 32 0 OUTPUT NODEFVAL "q[31..0]"
// Retrieval info: USED_PORT: sclr 0 0 0 0 INPUT NODEFVAL "sclr"
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @sclr 0 0 0 0 sclr 0 0 0 0
// Retrieval info: CONNECT: q 0 0 32 0 @q 0 0 32 0
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter2.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter2.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter2.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter2.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter2_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL part7_counter2_bb.v TRUE
etrieval info: LIB_FILE: lpm
```