第 8 講 行為描述---迴圈

1. 參考文獻:

[1]Online Verilog Simulator: edaplayground 教學:
https://www.youtube.com/watch?v=f9uwtAax4v0&t=249s

[2]金門大學陳鍾誠老師電子書: http://ccckmit.wikidot.com/ve:main

[3]林灶生，verilog 晶片設計第 6 章，全華書局

2. 基本觀念[3]:

2.1 for 迴圈

```
for(迴圈變數=低值; 迴圈變數＜高值; 迴圈變數=迴圈變數+常數)
    begin
      <敘述區塊>;
    end
```

2.2 while 迴圈

```
while (條件判斷表示式)
    begin
        <敘述區塊>;
    end
```

2.3 forever 迴圈

```
forever
 begin
  ＜敘述區塊＞;
 end
```

2.4 repeat 迴圈

```
repeat (表示式)
    begin
        <敘述區塊>;
    end
```

例題 1 : BCD counter (from 00 ~ 99)

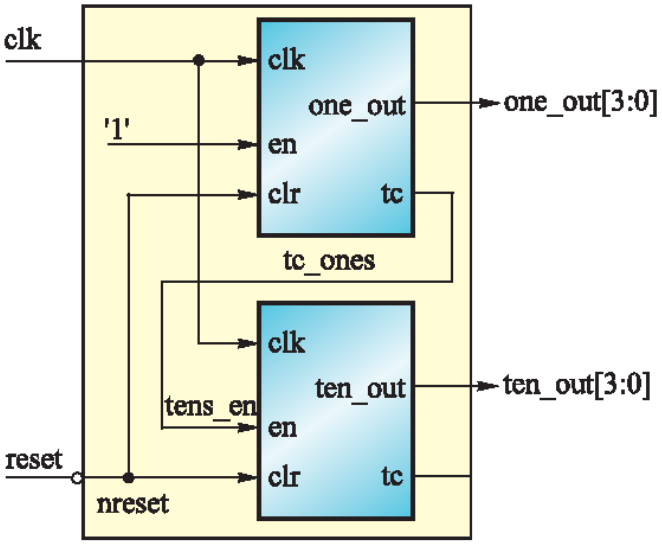

圖 1: BCD counter

```
testbench.sv
SV/Verilog Testbench
  initial
    $monitor($time, " clk=%b, reset=%b, ten_out=%d, one_out=%d", clk,
reset, ten_out, one_out);

  initial
    begin
      clk=0;
      reset=0;
      #35 reset=1;
    end

  initial
    begin
      forever #10 clk=~clk;
    end

  initial #400 $finish;

endmodule
```

```
design.sv
SV/Verilog Design
// Code your design here
module counter(clk, reset, one_out, ten_out);
  input clk, reset;
  output [3:0] one_out, ten_out;

  wire nreset, ones_en, tens_en, tc_ones;
  wire [3:0] one_out, ten_out;

  cnt_10 ONES(ones_en, clk, nreset, tc_ones, one_out);
  cnt_10 TENS(tens_en, clk, nreset, , ten_out);

  assign tens_en = tc_ones;
  assign ones_en = 1'b1;
  assign nreset = ~reset;
endmodule

module cnt_10(ce, clk, clr, tc, qout);
  input ce, clk, clr;
  output tc; //carry signal
  output [3:0] qout;  //count signal
```

```
Log   Share
[2024-04-07 10:12:47 UTC] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
               0, clk=0, reset=0, ten_out= 0, one_out= 0
              10, clk=1, reset=0, ten_out= 0, one_out= 0
              20, clk=0, reset=0, ten_out= 0, one_out= 0
              30, clk=1, reset=0, ten_out= 0, one_out= 0
              35, clk=1, reset=1, ten_out= 0, one_out= 0
              40, clk=0, reset=1, ten_out= 0, one_out= 0
              50, clk=1, reset=1, ten_out= 0, one_out= 1
              60, clk=0, reset=1, ten_out= 0, one_out= 1
              70, clk=1, reset=1, ten_out= 0, one_out= 2
              80, clk=0, reset=1, ten_out= 0, one_out= 2
```

- **design.sv 程式:**

```
module counter(clk, reset, one_out, ten_out);
   input clk, reset;
   output [3:0] one_out, ten_out;

   wire nreset, ones_en, tens_en, tc_ones;
   wire [3:0] one_out, ten_out;

   cnt_10 ONES(ones_en, clk, nreset, tc_ones, one_out);
   cnt_10 TENS(tens_en, clk, nreset, , ten_out);

   assign tens_en = tc_ones;
   assign ones_en = 1'b1;
   assign nreset = ~reset;
endmodule

module cnt_10(ce, clk, clr, tc, qout);
   input ce, clk, clr;
   output tc; //carry signal
   output [3:0] qout;    //count signal

   reg [3:0] count;

   always@(posedge clk or posedge clr)
      begin
         if (clr)
            count <=4'b0;
         else
            if(ce)
               if(count==4'h9)
                  count <= 4'h0;
            else
               count <= count+1;
      end

   assign qout = count;
   assign tc = (count==4'h9); //carry signal
endmodule
```

- **testbench 程式:**

```
module testbench();
   reg clk, reset;
   wire [3:0] one_out, ten_out;

   counter U1(clk, reset, one_out, ten_out);

   initial
      $monitor($time, '', clk=%b, reset=%b, ten_out=%d, one_out=%d'', clk, reset,
ten_out, one_out);

   initial
      begin
         clk=0;
         reset=0;
         #35 reset=1;
      end

   initial
      begin
         forever #10 clk=~clk;
      end

   initial #400 $finish;

endmodule
```

- 執行後結果

```
[2024-04-07 10:12:47 UTC] iverilog '-Wall' design.sv testbench.sv  &&
unbuffer vvp a.out
                0, clk=0, reset=0, ten_out= 0, one_out= 0
               10, clk=1, reset=0, ten_out= 0, one_out= 0
               20, clk=0, reset=0, ten_out= 0, one_out= 0
               30, clk=1, reset=0, ten_out= 0, one_out= 0
               35, clk=1, reset=1, ten_out= 0, one_out= 0
               40, clk=0, reset=1, ten_out= 0, one_out= 0
               50, clk=1, reset=1, ten_out= 0, one_out= 1
               60, clk=0, reset=1, ten_out= 0, one_out= 1
```

```
 70, clk=1, reset=1, ten_out= 0, one_out= 2
 80, clk=0, reset=1, ten_out= 0, one_out= 2
 90, clk=1, reset=1, ten_out= 0, one_out= 3
100, clk=0, reset=1, ten_out= 0, one_out= 3
110, clk=1, reset=1, ten_out= 0, one_out= 4
120, clk=0, reset=1, ten_out= 0, one_out= 4
130, clk=1, reset=1, ten_out= 0, one_out= 5
140, clk=0, reset=1, ten_out= 0, one_out= 5
150, clk=1, reset=1, ten_out= 0, one_out= 6
160, clk=0, reset=1, ten_out= 0, one_out= 6
170, clk=1, reset=1, ten_out= 0, one_out= 7
180, clk=0, reset=1, ten_out= 0, one_out= 7
190, clk=1, reset=1, ten_out= 0, one_out= 8
200, clk=0, reset=1, ten_out= 0, one_out= 8
210, clk=1, reset=1, ten_out= 0, one_out= 9
220, clk=0, reset=1, ten_out= 0, one_out= 9
230, clk=1, reset=1, ten_out= 1, one_out= 0
240, clk=0, reset=1, ten_out= 1, one_out= 0
250, clk=1, reset=1, ten_out= 1, one_out= 1
260, clk=0, reset=1, ten_out= 1, one_out= 1
270, clk=1, reset=1, ten_out= 1, one_out= 2
280, clk=0, reset=1, ten_out= 1, one_out= 2
290, clk=1, reset=1, ten_out= 1, one_out= 3
300, clk=0, reset=1, ten_out= 1, one_out= 3
310, clk=1, reset=1, ten_out= 1, one_out= 4
320, clk=0, reset=1, ten_out= 1, one_out= 4
330, clk=1, reset=1, ten_out= 1, one_out= 5
340, clk=0, reset=1, ten_out= 1, one_out= 5
350, clk=1, reset=1, ten_out= 1, one_out= 6
360, clk=0, reset=1, ten_out= 1, one_out= 6
370, clk=1, reset=1, ten_out= 1, one_out= 7
380, clk=0, reset=1, ten_out= 1, one_out= 7
390, clk=1, reset=1, ten_out= 1, one_out= 8
400, clk=0, reset=1, ten_out= 1, one_out= 8
Done
```

3. 作業 8-1: 4 bit BCD adder

$$\begin{array}{rr} & A_3A_2A_1A_0 \\ + & B_3B_2B_1B_0 \\ \hline & C_4S_3S_2S_1S_0 \end{array}$$

當上述運算式之和大於 01001(=9)時，必須作修正 “0110”加至其中，以產生正確的 BCD。當上述運算修正如下:

C4	S3	S2	S1	S0	BCD 碼
0	1	0	1	0	(10)
0	1	0	1	1	(11)
0	1	1	0	0	(12)
0	1	1	0	1	(13)
0	1	1	1	0	(14)
0	1	1	1	1	(15)
1	0	0	0	0	(16)
1	0	0	0	1	(17)
1	0	0	1	0	(18)

修正信號 F = C4 + S3(S2 + S1) 加入 S3, S2, S1, S0 作為修正。

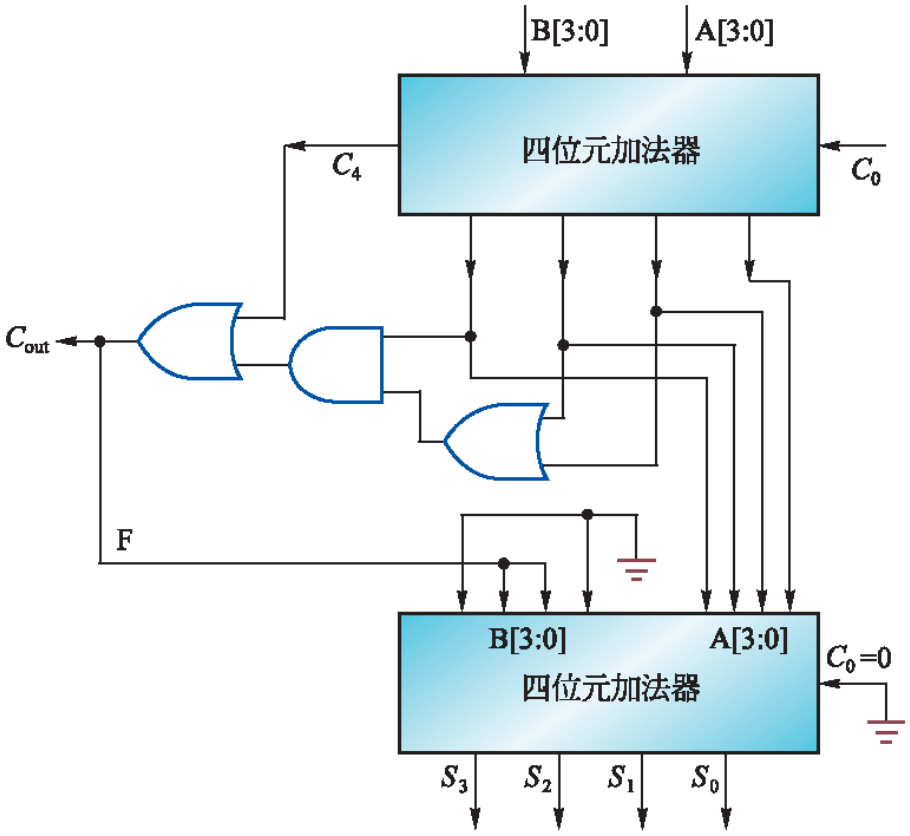

圖 1　4-bit BCD adder 方塊圖

- **design.sv 程式:**

```
module BCDadder4(S, Cout, Cin, A, B);
    output [3:0] S;
    output Cout;
    input Cin;
    input [3:0] A, B;

    wire [3:0] S_tmp;
    wire C4;
    reg [3:0] B_mod;
    reg F;

    adder4 BINADD(S_tmp, C4, Cin, A, B);
    adder4 MODADD(S, , 1'b0, S_tmp, B_mod);

    always@(Cin or A or B or C4 or S_tmp)
        begin
```

```
        F = (C4 | (S_tmp[3] & (S_tmp[2] | S_tmp[1])));
        B_mod = {1'b0, F, F, 1'b0} ; //modified code
      end
   assign Cout = F;

endmodule

module adder4(S, Cout, Cin, A, B);
   output [3:0] S;
   output Cout;
   input    Cin;
   input [3:0] A,B;

   assign {Cout, S} = A + B+ Cin;

endmodule
```

(1) 請設計 testbencht 程式，執行後將結果貼到作業報告中。