正確學會 改訂新版 ディジタル回路と Verilog HDL

# Verilog 的16堂課

## 第 12 章

# 同步電路設計

# 本章重點

在設計同步電路設計時, 除了要依照規格表所述來運作, 最好確保在更嚴格的條件下也要能正確動作才行。

- 12.1計數器的串聯連接
- 12.2陷阱修正與測試電路
- 12.3輸入信號的同步化
- 12.4檢查輸入信號的變化
- 11.5輸出信號的突波
- 12.6用 always 來描述組合電路

# 12.1 計數器的串聯連接

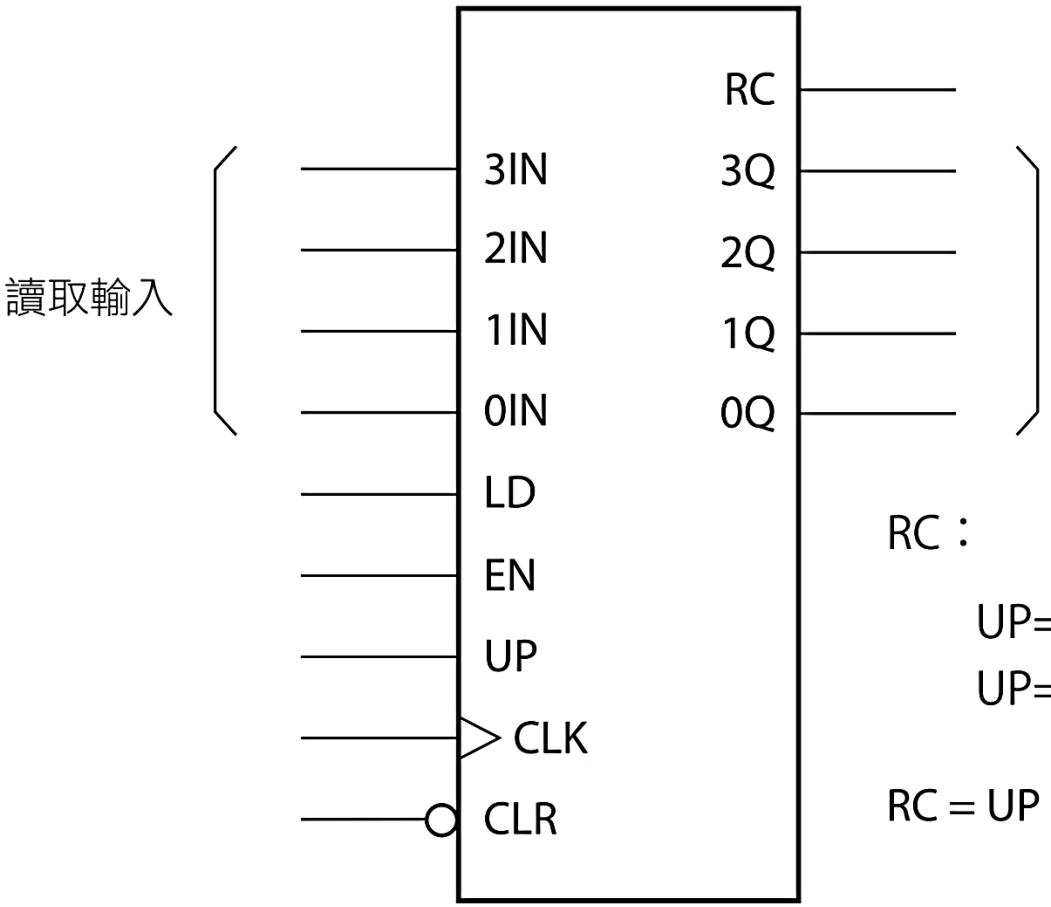
## 12.1.1 「RC」的追加

- 追加對計數器的串聯連接所必須的 RC (RC, Ripple Carry 漣波進位)」
- 當上數計數器的計數值達到最大, 或當下數計數器的計數值達到最小, RC 為 ON
- 以 mod-10 計數器為例：
  - 「UP」為ON, 計數值='9‘ ,「RC」為 ON
  - 「UP」為 OFF, 計數值='0‘ ,「RC」為ON

# 12.1.2 電路符號與串聯連接

- 把 RC 追加到上一章
  「11.5 附有讀取, 致能的上下數計數器」
  - (1)低位元計數器的「LD」~「CLR」已經有說明過了所以省略
  - (2)高位元計數器的「EN」以外, 都跟低位元計數器相通的部分連接
  - (3)高位元計數器的「EN」的連接與低位元計數器不同

- 上述 (3) 為串聯重點
  - (a) 時脈正緣
  - (b)「EN」為 ON
  - (c)「UP」為 ON 且低位元計數器的值為‘9’, 或「UP」為 OFF 且低位元計數器的值為 ‘0’
- 條件 (a) 因為 CLK 接到時脈,要滿足 (b), (c)：
  - 高位元計數器「EN」= 低位元計數器「RC」「EN」




RC： 漣波進位 (Ripple Carry)

UP='1' 的話 計數值='9' 的時候 ON

UP='0' 的話 計數值='0' 的時候 ON

$$RC = UP \cdot Q_3 \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot Q_0 + \overline{UP} \cdot \overline{Q_3} \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0}$$

▲圖 12.1 附有讀取, 致能的 mode-10 上下數計數器的電路符號

- 透過高位元跟低位元計數器的 RC 做 AND, 也可以實現顯示計數值=「99」或 '0' 的信號

# 12.1.3 mode-100 計數器的 Verilog HDL 描述

程式 12.2 mode-100 計數器的 Verilog HDL 描述 LST12_2.V

```
/*       CNT100          */
module   CNT100          (RESET_B,CLK,LOAD,EN,UP,IN,Q,CNT_99);
input    RESET_B, CLK, LOAD, EN, UP;
input    [7:0] IN;
output   CNT_99;
output   [7:0] Q;
wire     LOW_CNT_9, HIGH_CNT_9;
         LD_EN_UDCNT10  LOW_CNT (RESET_B, CLK,LOAD, EN,
                                 UP, IN[3:0], Q[3:0], LOW_CNT_9),
*1                      HIGH_CNT(RESET_B, CLK, LOAD, EN & LOW_CNT_9 ,
                                 UP, IN[7:4], Q[7:4], HIGH_CNT_9);
*2       assign         CNT_99 = LOW_CNT_9 & HIGH_CNT_9;
endmodule
```

- 把下層模組「 z LD_EN_UDCNT10」用簡碼名「LOW_CNT」,「HIGH_CNT」呼叫出來
- 透過「 LOW_CNT」,「HIGH_CNT」的「RC」的 AND 來做成「CNT_99」

程式 12.3　mode-100 計數器的 Verilog HDL 描述 (附有 RC 的 mode-10 上下數計數器)　LST12_2.V

```
/*          LD_EN_UDCNT10   */
module      LD_EN_UDCNT10   ( RESET_B, CLK, LOAD, EN, UP,  IN,  Q,  RC );
input       RESET_B, CLK, LOAD, EN, UP;
input       [3:0] IN;
output      [3:0] Q;
output      RC;
reg         [3:0] Q;
      *3    assign  RC = UP & ( Q ==9 ) | ~UP & ( Q == 0 );
            always  @( posedge CLK or negedge RESET_B  )
                    if      ( !RESET_B )
                            Q <= 0;
                    else if     ( LOAD )
                            Q <= IN;
```

- 「RC」信號是當遞增動作的「計數值」='9'，或者遞減動作的「計數值」='0' 的時候為 ON

# 12.1.5 mode-256 計數器的 Verilog HDL 描述

- 用兩個 mode-16 計數器的話就可以構成 mode-256

程式 12.4 mode-256 計數器的 Verilog HDL 描述 LST12_4.V

```
/*       CNT256          */
module   CNT256          (RESET_B, CLK, LOAD, EN, UP, IN, Q, CNT_255);
input    RESET_B, CLK, LOAD, EN, UP;
input    [7:0] IN;
output   CNT_255;
output   [7:0] Q;
wire     LOW_CNT_15, HIGH_CNT_15;
         LD_EN_UDCNT16  LOW_CNT (RESET_B, CLK, LOAD, EN,
                                 UP, IN[3:0], Q[3:0], LOW_CNT_15),
                        HIGH_CNT (RESET_B, CLK, LOAD, EN & LOW_CNT_15,
                                 UP, IN[7:4], Q[7:4], HIGH_CNT_15);
         assign         CNT_255 = LOW_CNT_15 & HIGH_CNT_15;
endmodule
```

程式 12.5　mode-256 計數器的 Verilog HDL 描述 (附有 RC 的 mode-10 上下數計數器)　LST12_5.V

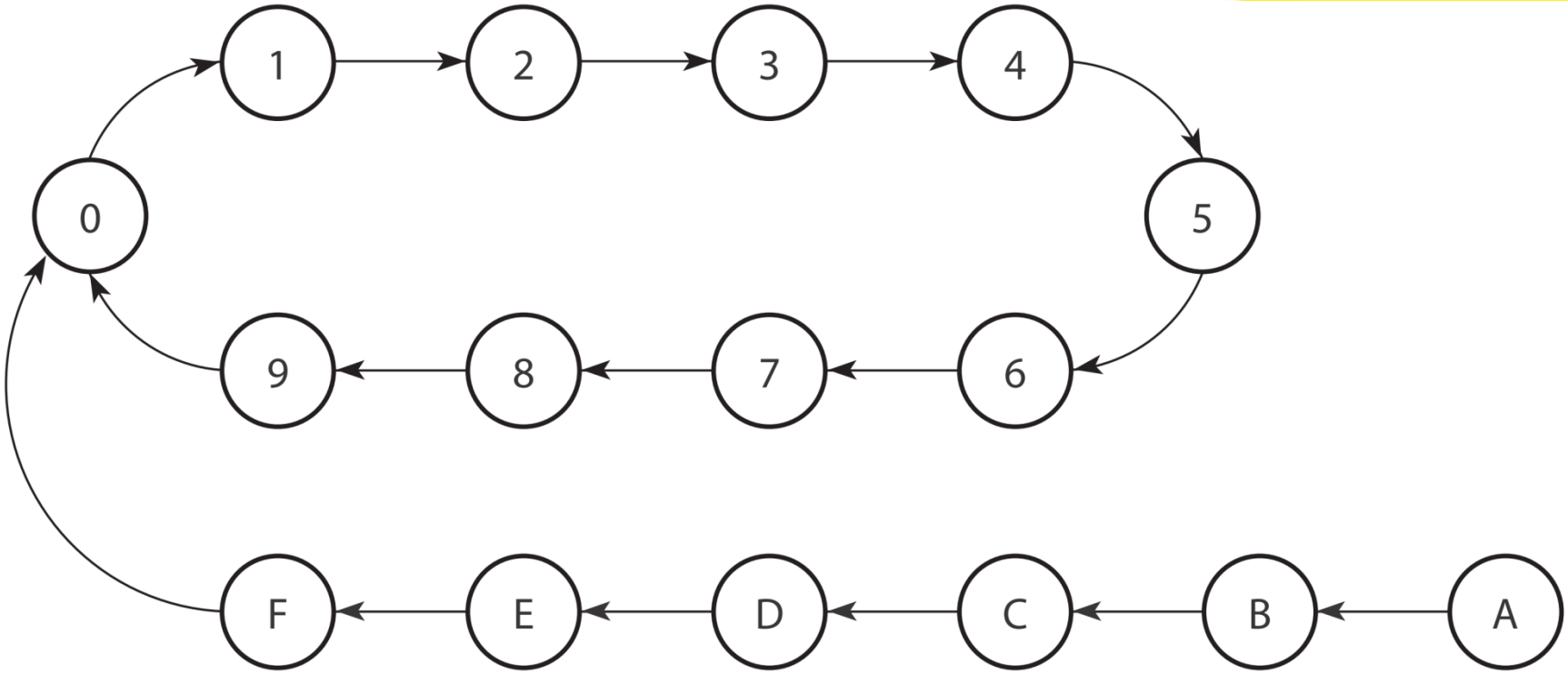
```
/*        LD_EN_UDCNT16  */
module    LD_EN_UDCNT16  ( RESET_B, CLK, LOAD, EN, UP, IN, Q, RC );
input     RESET_B, CLK, LOAD, EN, UP;
input     [3:0] IN;
output    RC;
output    [3:0] Q;
reg       [3:0] Q;
          assign    RC = UP & ( Q == 15 ) | ~UP & ( Q == 0 );
          always    @( posedge CLK or negedge RESET_B  )
                    if   ( !RESET_B )
                         Q <= 0;
                    else if  ( LOAD )
                         Q <= IN;
                    else if  ( EN )
                         if  ( UP )        // COUNT UP
                            Q <= Q + 1;
                         else              // COUNT DOWN
                            Q <= Q - 1;
endmodule
```

# 12.2 陷阱修正與測試電路

- 驗證在更嚴格的條件底下也要能正確運作的動作稱為**陷阱修正**, 為了確認其動作而追加的電路則稱為**測試電路**。

# 12.2.1 陷阱修正

- 陷阱修正指「電路受到雜訊之類的影響而產生預料外的狀態的時候, 把它修正回原來正常的狀態」的意思
- 以同步 mode-10 上數計數器為例,現在狀態對於 'A'~'F' 所定義的下一個狀態的動作不明
- 若要進行陷阱修正, 可改成：
  - (a) 計數值 'A'~'F' 的下一個狀態馬上回到 '0'
  - (b) 計數值 'A' 以上的下一個狀態依序從 'B' → 'C'→……'F' → '0'




▲ 圖 12.5　附有陷阱修正的 mode-10 上數計數器的狀態圖

- 製作出特性表求邏輯式，就可以完成電路設計

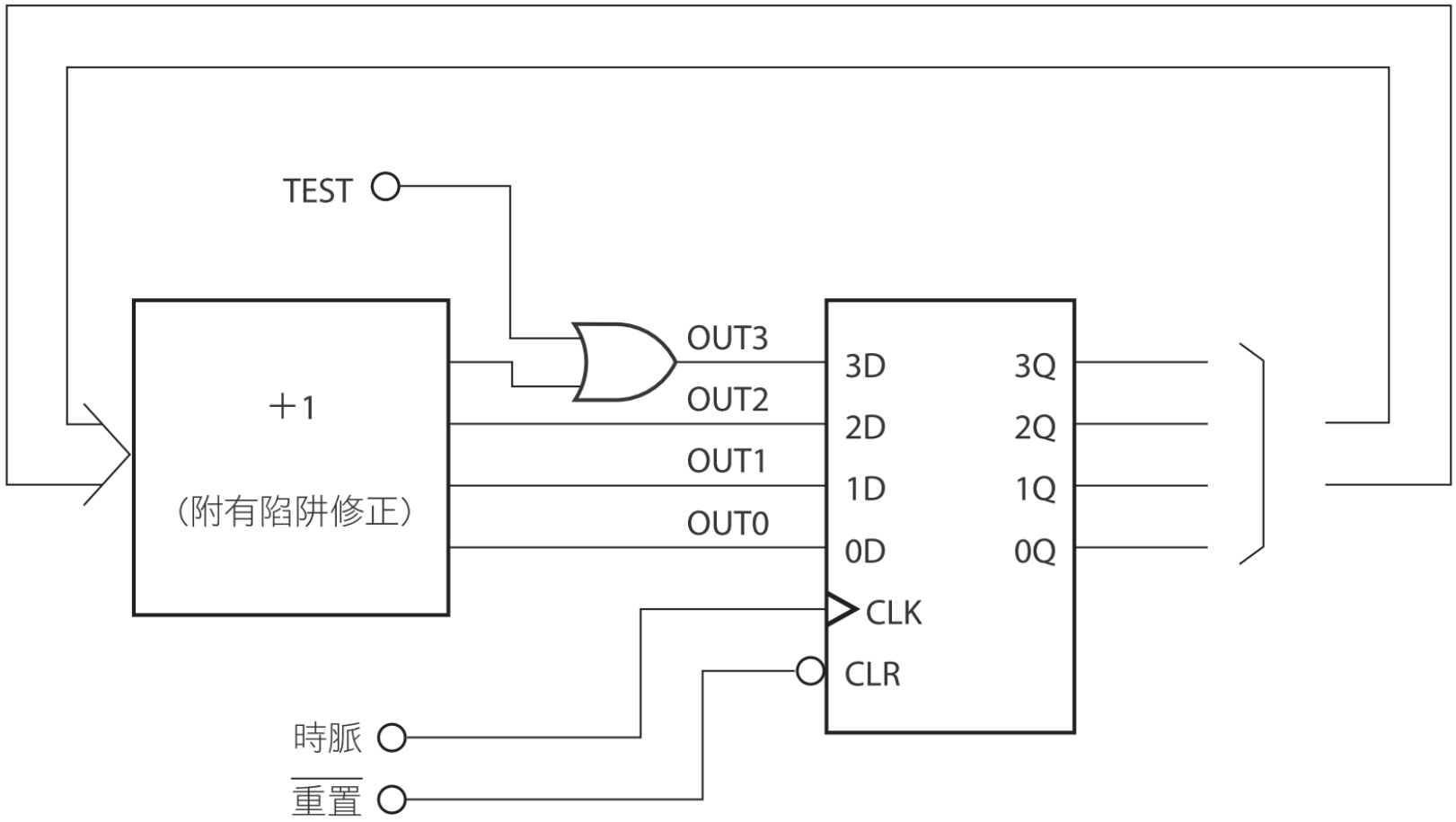
- 陷阱修正的目的不是防止電路的誤動作, 而是從誤動作中復原。
- 陷阱修正並不能讓誤動作發生的機率降低。
- 陷阱修正是對於那些有意外會發生的電路所做,像mode-16計數器這種不會有意外狀態的序向電路就不需要了。

# 12.2.2 陷阱修正的 Verilog HDL 描述

- 陷阱修正後的 mode-10 上數計數器的 Verilog HDL 描述, 跟之前在第 7 章的同步 mode-10 上數計數器的描述 (程式 7.2) 一樣
- 因為先前 Verilog 描述將 '9' 以外的數值就做遞增的動作, 和此處的邏輯式等效

# 12.2.3 追加測試電路

- 模擬 mode-10 上數計數器的動作時, 若只依據特性表讓輸出從 '0'~'9' 變化,會漏掉陷阱修正的動作
- 「為了確認陷阱修正的動作而設定計數值為 '0'~'9' 以外的電路」稱為測試電路
- 此電路可透過「+1 電路」的最高位元與「TEST」的 OR 來實現




▲ 圖 12.7　　附有測試電路的 mode-10 上數計數器

# 12.2.4 考慮測試電路的 Verilog HDL

程式 12.8　附有陷阱修正, 測試電路的 mode-10 上數計數器的 Verilog HDL 描述　LST12_8.V

```
/*              TRAP_CNT10      */
module          TRAP_CNT10      ( RESET_B, CLK, TEST, Q );
input           RESET_B, CLK, TEST;
output          [3:0] Q;
reg             [3:0] Q;
wire            [3:0] OUT;
            *2  assign    OUT = { TEST, 3'b000} | INC_OUT ( Q );
                always    @( posedge CLK or negedge RESET_B )
                          if ( !RESET_B )
                                Q <= 0;
                          else
                                Q <= OUT;

function        [3:0] INC_OUT;                  ┐
input           [3:0] Q;                        │
                if ( Q == 9 )                   │
                          INC_OUT = 0;          │ *1
                else                            │
                          INC_OUT = Q + 1;      │
endfunction                                     ┘

endmodule
```

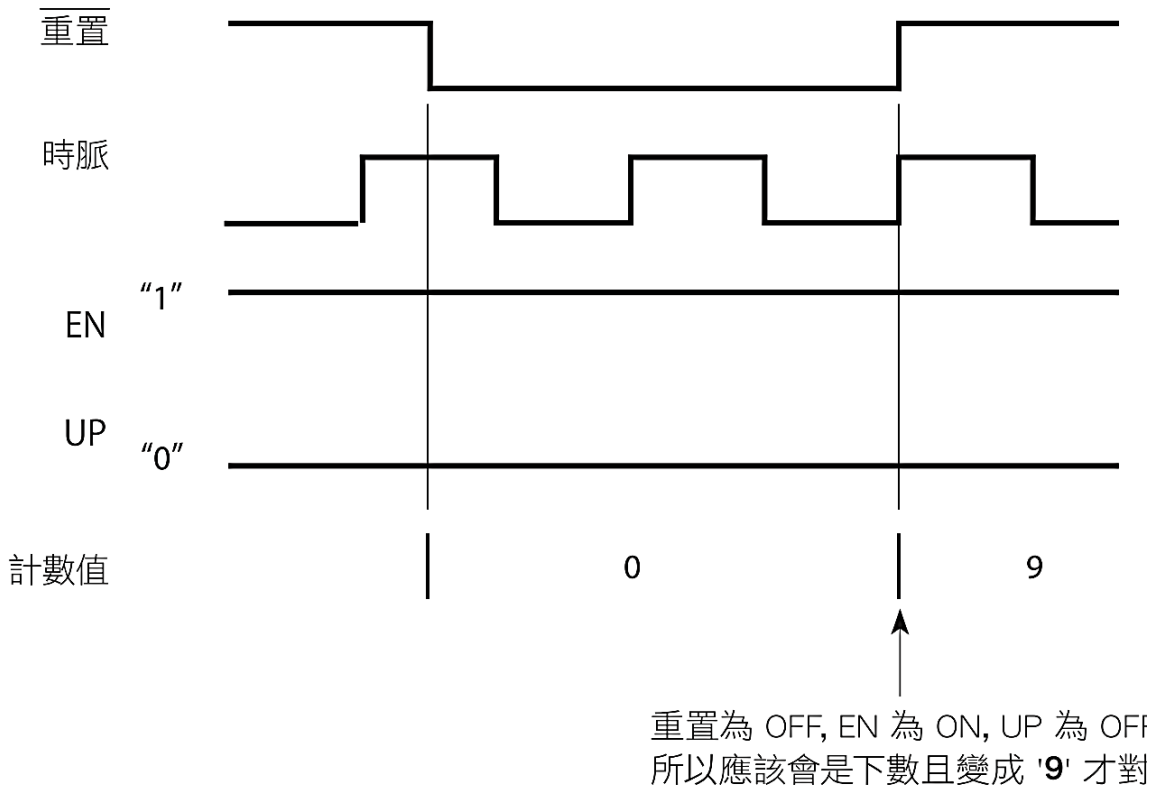
- 「 +1 電路」的輸出要與「TEST」做 OR, 所以暫存器跟「+1 電路」的描述要分開
- 要連到 D 的信號是「 +1 電路」的最高位元輸出跟「TEST」做 OR 之後的「OUT」

# 12.3 輸入信號的同步化

- 同步電路的外部輸入跟系統時脈不同步的時候，這些外部輸入不能一次影響電路中 2 位元以上的正反器

# 12.3.1 非同步重置信號

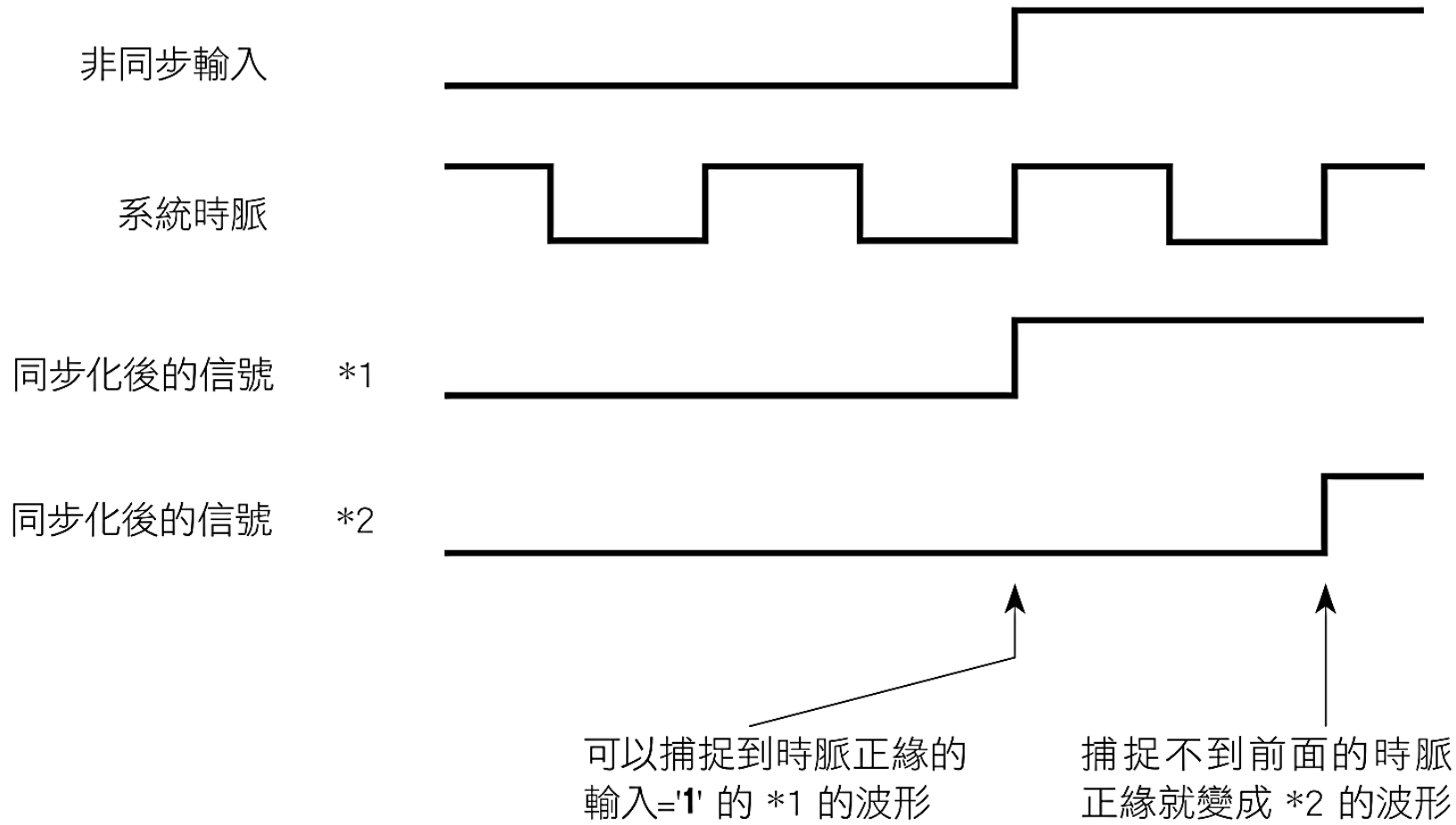
- 以附有致能的 mode-10 上數計數器來做檢驗,問題點在於「重置」OFF 的時機點




▲ 圖 12.9　非同步重置信號造成的誤動作

- 解決方法是把 CLK 與系統時脈連接的正反器的 D 與重置連接, 輸出則跟各正反器的 R (或者暫存器的 CLR) 連接

(b)　假設設定時間, 保持時間之中非同步的輸入有變化



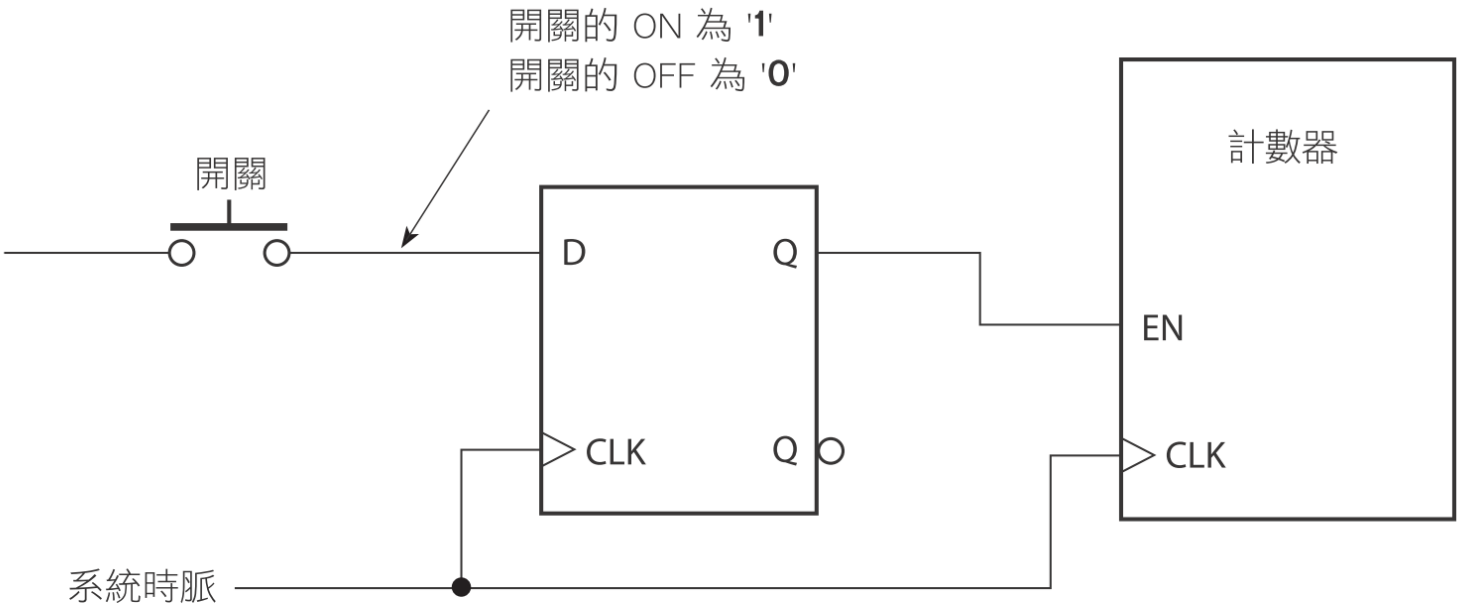

▲圖 12.10　輸入的同步化

# 12.3.2 同步重置信號

- 同步重置方式，是把重置與正反器的 D 或者 J, K 連接，當時脈正緣的時候就會清除正反器
- 同步電路的外部輸入跟系統時脈不同步的時候，這些外部輸入不能一次影響電路中 2 位元以上的正反器

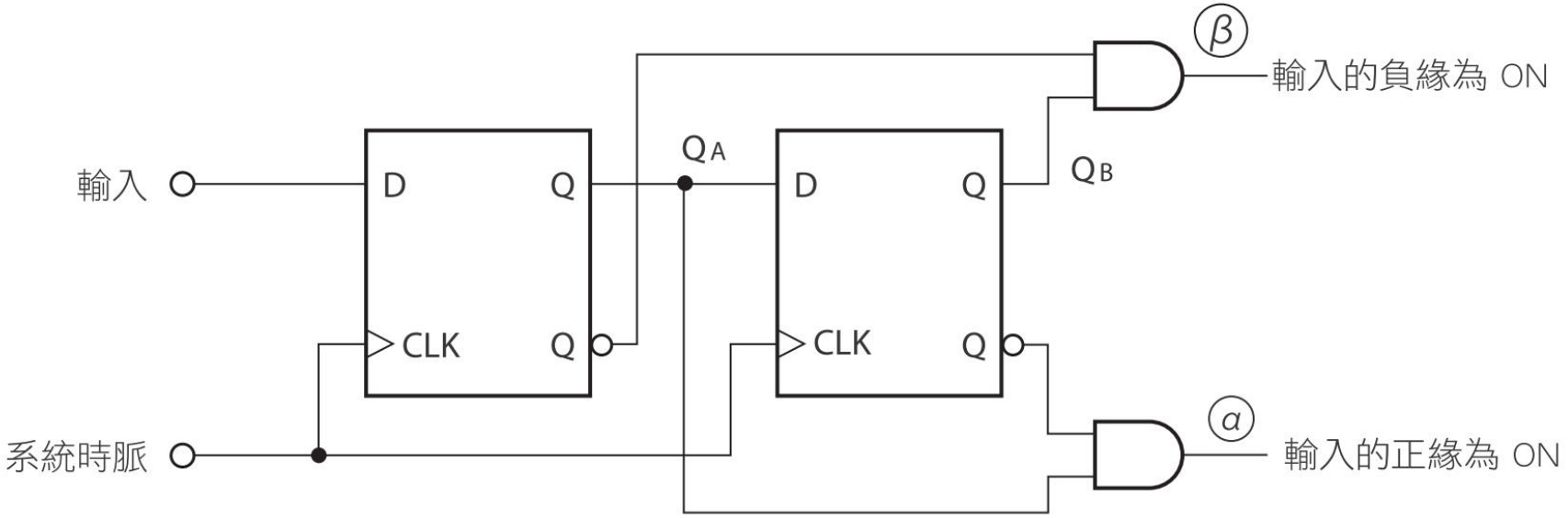
# 12.4 檢查輸入信號的變化

- 規格1：設置一個開關,
  - 為 ON 則每次系統時脈變化就上數,
  - 為 OFF計數值就不變化的計數器
  - 開關電路的細節可省略,計數值最大為'9'




▲ 圖 12.11　根據開關的操作來控制計數器

- 規格2：設置一個開關
  - 為 ON 則上數 '1‘
  - 為 OFF 也一樣上數
  - 開關電路的細節與計數的最大值跟前例一樣
- 用兩個輸入跟系統時脈同步化後的 D 型正反器相連接，把變化透過正反器的輸出「$Q_A$」，「$Q_B$」的時間差來表示




- α：可檢查出輸入的正緣與系統時脈的同步
- β：可檢查出輸入的負緣與系統時脈的同步

# 12.4.2 檢查電路的 Verilog HDL 描述

- 模組「SW_DEF」用 always 來描述 $Q_A$ 與 $Q_B$，用 assign 來描述 α, β

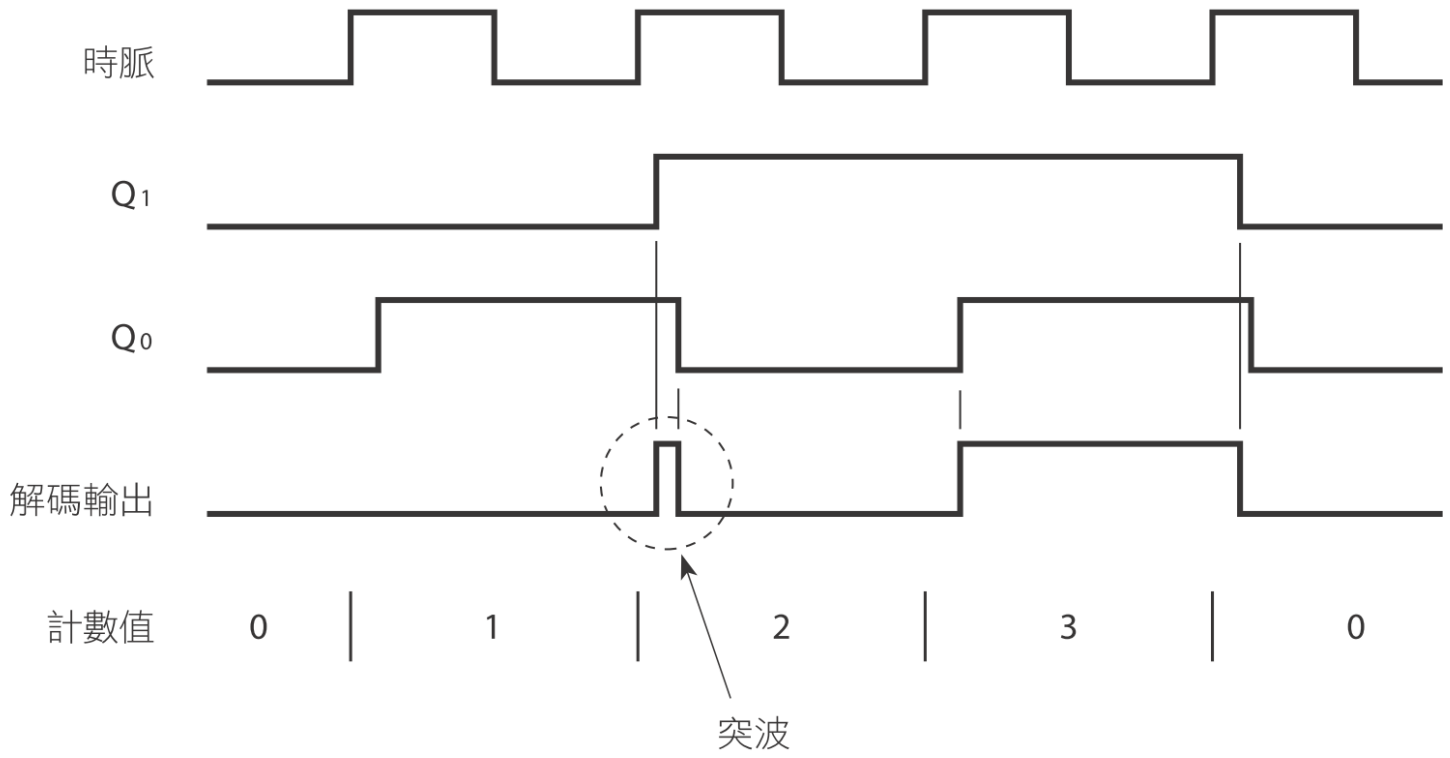
程式 12.10 檢查輸入變化的電路與計數器的 Verilog HDL 描述 LST12_10V

```
/*        SW_DEF    */
module    SW_DEF    ( RESET_B, CLK, SW_INPUT, Q );
input     RESET_B, CLK, SW_INPUT;
output    [3:0] Q;
reg       QA, QB;
wire      ALFA, BETA;

          always    @( posedge CLK or negedge RESET_B  )
                    if        ( !RESET_B )
                              begin
                                        QA <= 0;
                                        QB <= 0;
                              end
                    else
                              begin
                                        QA <= SW_INPUT;
                                        QB <= QA;
                              end

          assign    ALFA  = ~QB &  QA;
          assign    BETA  =  QB & ~QA;

          EN_CNT10 EN_CNT10     ( RESET_B, CLK, ALFA | BETA, Q );
endmodule
```

# 12.5 輸出信號的突波

- 同步電路的組合電路的輸出含有像鬍子一樣細的脈波,由於訊號受到干擾產生雜訊, 稱為「突波」

# 12.5.1 發生突波的狀況



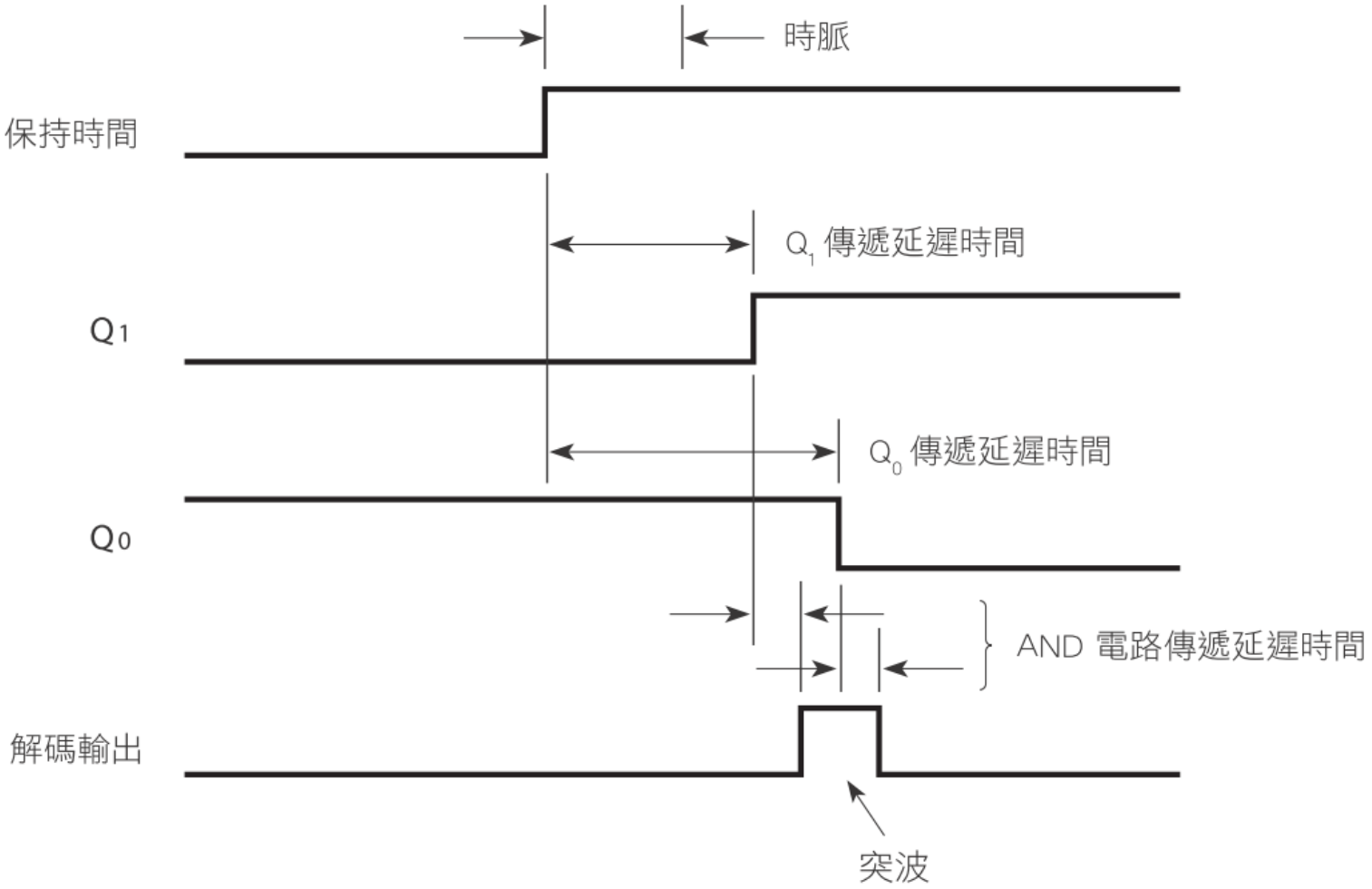

$Q_1$ 的傳遞延遲時間 < $Q_0$ 的傳遞延遲時間就會發生突波

▲圖 12.16　突波的發生

- 突波是根據電路的延遲時間所產生的
- 突波有可能因為同時變化的 2 位元以上的信號做 AND 或 OR 而發生
- 組合電路的輸出會因上述原因造成突波, 但是同步電路就沒這個問題

# 12.5.2 防突波設計

- 同步電路有以下特性, 所以不用擔心突波問題
  - 正反器的傳遞延遲時間 > 保持時間




- 保持時間中, 解碼輸出不會發生突波
- 同步電路是沒有突波問題的

# 12.6 用 always 來描述組合電路

- 組合電路除了用 assign 與 function 來描述, 也可以用 always 來描述

考量到 HDL 的可讀性, 還是用 assign 來描述組合電路, 用 always 來描述序向電路比較適當

程式 12.11　用 always 描述解碼器　LST12_11.V

```
/*        DEC_3TO8 */
module    DEC_3TO8 ( IN, OUT );
input     [2:0] IN;
output    [7:0] OUT;
reg       [7:0] OUT; ←*3
          always   @( IN ) ←*1
                   case( IN )
                            0: OUT = 8'b0000_0001;
                            1: OUT = 8'b0000_0010;
                            2: OUT = 8'b0000_0100;
                            3: OUT = 8'b0000_1000;
                            4: OUT = 8'b0001_0000;
                            5: OUT = 8'b0010_0000;
                            6: OUT = 8'b0100_0000;
                            7: OUT = 8'b1000_0000;
                   endcase
endmodule
```

*2

- 1.用 always 的事件式來列舉出所有的輸入
- 2.把所有可能的輸入都描述出對應的狀態
- 3.輸出即使是組合電路也用 reg 來宣告