Verilog 硬體描述語言

Verilog HDL

A Guide to Digital Design and Synthesis

IEEE 1364-2001 Compliant

第6章 資料處理模型

6.1 持續指定的描述

- 持續指定(Continuous Assignment)
  - 是在資料處理(Dataflow)層次中對一條接線(net)指定其邏輯值的最基本用法。
- 持續指定的四項特點

1.在指定敘述的左邊，項必須是純量接線(scalar **wire**)的單一條線或向量接線(vector **wire**)，不能是暫存器(**reg**)。

2.持續指定永遠處於活動狀態，一旦敘述右邊的任何運算元值發生變化時左邊指定的線值也會隨之改變。

3.敘述右邊的任何運算元，可以是暫存器(**reg**)、接線(**wire**)或一個函數呼叫。

4.延遲(Delay)是用來控制，當指定敘述需要更動左邊接線值時所需經過的延遲時間，其數值是以單位時間來計量。

6.1 持續指定的描述

範例6-1 持續指定

```
// 持續指定。out、i1 及 i2 皆為接線型態之變數。
assign out = i1 & i2 ;

// 矩陣形式的持續指定，其中變數 addr 為一個 16 位元向量接線，
// addr1_bits 與 addr2_bits 為 16 位元向量暫存器。
assign  addr[15:0] = addr1_bits[15:0] ^ addr2_bits[15:0] ;

//  連結運算運用在持續指定。左手邊是一個純量接線變數，與一個 4 位元
    的向量接線向量變數之連結。
assign   {c_out , sum[3:0]}  = a[3:0] + b[3:0] + c_in;
```

6.1 持續指定的描述

- 隱含式的持續指定(Implicit Continuous Assignment)
  - 每個接線只能有一個隱含式的持續指定，因為一個接線只能被宣告一次。

```
wire out;  //正規式的持續指定
assign out = in1 & in2;
// 隱含式的持續指定，與上面敘述有相同效果
wire out = in1 & in2;
```

- 隱含式的接線宣告(Implicit Net Declaration)
  - 對一條未經宣告的訊號使用隱含式的持續指定時，該訊號會自動被宣告為接線(**wire**)。

```
wire i1, i2;  // 持續指定，out為一個接線
assign out = in1 & in2;
// 訊號out並未被宣告為接線但因其為隱含式的持續指定，
// 模擬器會自動完成此項宣告
```

6.2 延　遲(Delays)

- 延遲是指一個assign的敘述，從右邊值發生變化到相對應左邊值發生變化所需經過的時間。
- Verilog 指定延遲的三種方法
  1. 正規指定延遲(Regular Assignment Delay)
  2. 隱含式指定延遲(Implicit Continuous Assignment Delay)
  3. 接線宣告延遲(Net Declaration Delay)

6.2 延　遲(Delays)

- 正規指定延遲(Regular Assignment Delay)

```
assign #10 out = in1 & in2;  // 在一個持續指定敘述中描述延遲的方法
```

  - 在上例中當in1或是in2值發生變化時，相對應到out變化所需的時間為 10 個時間單位，又稱為慣性延遲(Inertial Delay)。
  - 當輸入突波(Pulse)出現的時間寬度小於延遲時，則輸入引發的變化將不會傳到輸出。

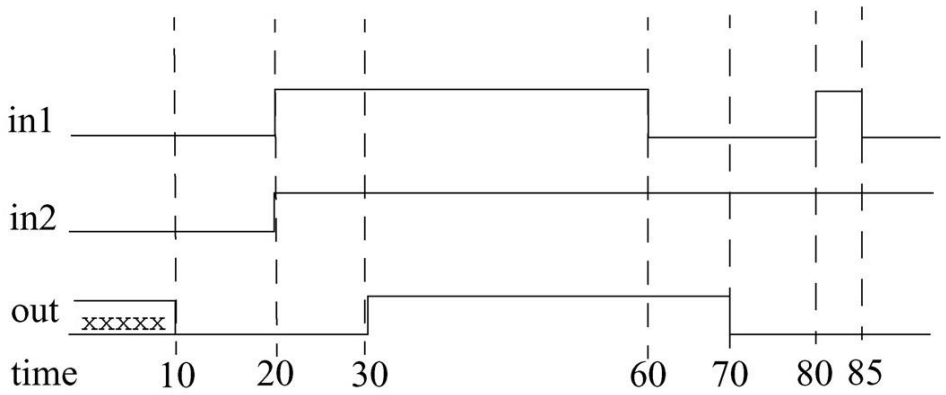


圖6-1　延遲

6.2 延　遲(Delays)

- 正規指定延遲(Regular Assignment Delay)
  - 當 in1 與 in2 在時間等於20時變為邏輯1，相對out值變化延遲了10個時間單位才變為邏輯1。
  - 當in1在時間等於60時變為邏輯0，相對out值變化延遲了10個時間單位到時間70才變為邏輯 0。
  - 當in1在時間等於80時變為邏輯1，並等到時間85時變為0，因為時間間隔為5小於延遲值10，所以相對應的變化沒傳到out 。

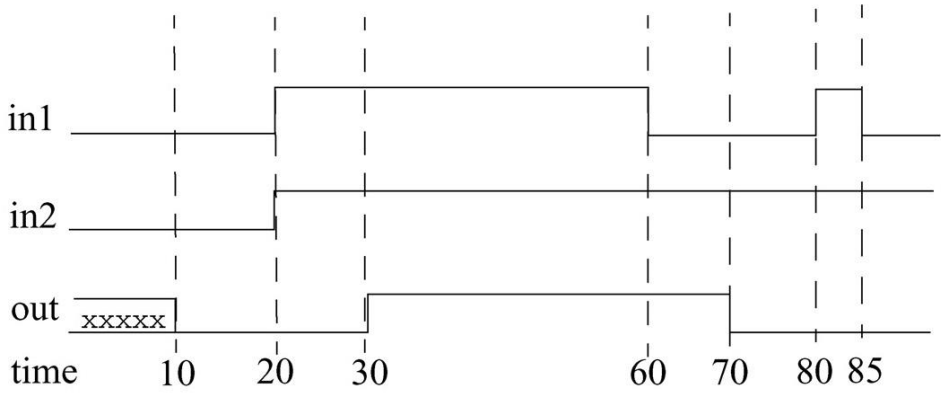


圖6-1　延遲

6.2 延　遲(Delays)

- 隱含式指定延遲(Implicit Continuous Assignment Delay)
  - 我們可以在宣告一條接線時，同時立即指定這條接線的輸入敘述和輸入到輸出的延遲，如以下範例所示。

```
// 隱含式持續指定描述延遲的方法
wire #10 out = in1 & in2;
// 相同於
wire out;
assign #10 out = in1 & in2;
```

  - 此方法與第一種方法的效果相同。

6.2 延　遲(Delays)

- 接線宣告延遲(Net Declaration Delay)
  - 第三種方法是在宣告一條接線時，只指定這條接線的延遲，這條接線的敘述則用關鍵字assign來指定，如以下範例所示。

```
// 宣告一個 net 變數 out 的延遲時間為 10 個時間單位
wire #10 out;
assign out = in1 & in2;

// 相同於
wire out;
assign #10 out = in1 & in2;
```

  - 接線宣告延遲也可以用在邏輯閘層次的模型中。

6.3 運算式、運算子與運算元

- 資料處理模型藉由不同的運算來描述電路設計，而不是用主要邏輯閘來描述，因此資料處理模型主要由運算式(Expressions)、運算子(Operators)與運算元(Operands)來建立。
- **運算式**
  - 結合運算子與運算元而產生一個結果，如下例所示。

```
//結合運算子與運算元的運算式範例
a^b
addr1[20:17] + addr2[20:17]
In1 | in2
```

6.3 運算式、運算子與運算元

- 運算元
  - 有些指定敘述只能接受某些特殊的運算元。
  - 運算元的資料形態包括:常數(Constants)、整數(Integers)、實數(Real Numbers)、接線(Nets)、暫存器(Registers)、時間(Times)、位元選擇(Bit-Select)(指定向量接線或向量暫存器中的特定位元)、部分向量選擇(Part-Select)(選擇向量接線或向量暫存器中的多個位元)、記憶體(Memories)或是函數的回傳值(Function Calls)等等。

```
integer count, final_count;
final_count = count + 1; // count 是一個整數運算元
real a, b, c;     c = a – b; // a與b是實數
reg [15:0] reg1, reg2; reg [3:0] reg_out;
reg_out = reg1[3:0] ^ reg2[3:0]; // 向量暫存器的一部分
reg ret_value;  ret_value = calculate_parity(A, B); // 函數形態運算元
```

6.3 運算式、運算子與運算元

- 運算子
  - 運算子作用在運算元上以得到想要的結果。

```
d1 && d2  // && 是一個運算子，作用在d1與d2上
!a[0]     // ! 是一個運算子，作用在a[0]上
B1>>1     // >> 是一個運算子，作用在B與1上
```

6.4 運算子的種類

表6-1　運算子的種類與符號

運算子種類	符號	運算功能	所需的運算元數目
算術運算符號	*	乘法	2
	/	除法	2
	+	加法	2
	–	減法	2
	%	取餘數	2
	**	指數	2
邏輯運算符號	!	邏輯上的 “NOT”	1
	&&	邏輯上的 “AND”	2
	||	邏輯上的 “OR”	2
比較運算符號	>	大於	2
	<	小於	2
	>=	大於或等於	2
	<=	小於或等於	2
相等運算符號	==	等於	2
	!=	不等於	2
	===	事件上的等於	2
	!==	事件上的不等於	2
位元運算符號	~	逐位元反相(取 1 的補數)	1
	&	對相對位元作 "AND"	2
	|	對相對位元作 "OR"	2
	^	對相對位元作 "XOR"	2
	^~或~^	對相對位元作 "XNOR"	2

6.4 運算子的種類

表6-1　運算子的種類與符號(續)

運算子種類	符號	運算功能	所需的運算元數目
化簡的位元運算符號	&	化簡的 "AND"	1
	~&	化簡的 "NAND"	1
	|	化簡的 "OR"	1
	~|	化簡的 "NOR"	1
	^	化簡的 "XOR"	1
	^~或~^	化簡的 "XNOR"	1
移位運算符號	>>	向右移位	2
	<<	向左移位	2
	>>>	向右算術移位	2
	<<<	向左算術移位	2
連結運算符號	{}	連結	任意數目皆可
	{{}}	複製	任意數目皆可
條件運算符號	?:	做條件運算	3

6.4 運算子的種類

- 算術運算子(Arithmetic Operators)
  - 算術運算子包含有兩種: 二元(Binary)運算子與一元(Unary)運算子。
- 二元運算子
  - 二元的運算符號有乘(*)、除(/)、加(+)、減(-)、指數運算(**)與取餘數(%)，皆需要兩個運算元。

```
A = 4'b0011;  B = 4'b0100; // A與B是暫存器向量變數
D = 6; E = 4; F = 2; // D、E與F 是整數
A * B; // 將A與B相乘，結果是4'b1100
D/E;  // D除以E，結果是1，小數部分被捨去
A+B; //將A與B相加，結果是4'b0111
B-A; // B減掉A ，結果是4'b0001
F = E**F; // E 的 F 次方結果是16
```

6.4 運算子的種類

- 二元運算子
  - 如果任一個運算元有不確定值(x)的位元時，則其結果為不確定值(x)，如下例所示。

```
in1 = 4'b101x;
in2 = 4'b1010;
sum = in1 + in2 ; // sum 的結果子是 4'bx
```

  - 取餘數運算子為求取兩個運算元相除所得之餘數功用與C語言取餘數運算結果一樣。

```
13 % 3 ; // 結果為 1
16 % 4;  // 結果為 0
-7 % 2;  // 結果為 -1，正負符號要與第一個運算元相同
7 % -2;  // 結果為 +1，正負符號要與第一個運算元相同
```

6.4 運算子的種類

- 一元運算子
  - 如正號(+)與負號(-)可當作一元運算子來用，他們較二元的加減法有較高的優先執行順序。

```
- 4   // 負 4
+ 5   // 正 5
```

  - 要注意的是在Verilog中的負數皆是以二的補數來表示，因此在Verilog中負數的資料型態最好是整數或實數，避免使用<sss>'<base><nnn>型態的負數，因為它們會被轉換成無號的二的補數，經常產生無法預期的結果。

```
// 建議使用整數或實數
-10 / 5    // 結果為 -2
-'d10 / 5  // 相等於(10的二補數) / 5 = (2^32 - 10) / 5
// 其中32代表預設的字元(word)的長度
// 這將會導致一個不正確且不可預期的結果
```

6.4 運算子的種類

- 邏輯運算子(Logical Operators)
  - 邏輯運算子包含有邏輯及(logic-and, &&)、邏輯或(logic-or, ||)與邏輯反(logic-not, !)，其中及(&&)與或(||)為二元運算符號，反(!)為一元運算符號。
  - 邏輯運算結果會產生一個一位元的值其中1代表真(True)、0代表假(False)、x代表不確定(Ambiguous)。
  - 如果運算元不等於0則傳回邏輯1，表示所得結果為真。若等於0則傳回邏輯0，表示所得結果為假。如果運算元中有x或是z的位元時則傳回x，這種情況在模擬器中通常被認定結果為假。
  - 邏輯運算子的運算元可以是一個變數，或者是一個運算式。
  - 建議在邏輯運算中，為了使人容易了解運算的相對關係與先後順序，可以使用括號()。

6.4 運算子的種類

- 邏輯運算子(Logical Operators)
  - 邏輯運算子範例

```
// 邏輯運算
A = 3;  B = 0;
A && B; //結果為 0，相同於(邏輯 1 && 邏輯 0)
A  || B;   //結果為 1，相同於(邏輯 1 || 邏輯 0)
! A;   //結果為 0，相同於not(邏輯 1)
! B;   //結果為 1，相同於not(邏輯 0)

// 不確定值
A = 2'b0x; B = 2'b10;
A && B; //結果為 x，相同於(x && 邏輯 1)
// 運算式
(a == 2) && (b == 3) //若是a == 2且b == 3 則結果為 1
// 若其中一個為非則結果為 0
```

6.4 運算子的種類

- 比較運算子(Rational Operators)
  - 比較運算子有大於(>)、小於(<)、大於或等於(>=)、小於或等於(<=)四種。
  - 比較的結果為真則傳回 1，為假則傳回 0。
  - 當相互比較的兩個運算元中，有一個值當中包含x或z的位元時，則傳回x代表無法比較大小。

```
A = 4;  B = 3;
X = 4'b1010; Y = 4'b1101; Z = 4'b1xxx;
A <= B;   //結果為邏輯 0
A > B;    //結果為邏輯 1
Y >= X;   //結果為邏輯 1
Y < Z;    //結果為 x
```

6.4 運算子的種類

- 相等運算子(Equality Operators)
  - 相等運算子有兩類:邏輯上的相等運算(相等==與不相等!=)與事件上的相等運算(相等===與不相等!==)。
  - 當結果為真則傳回 1，為假則傳回 0。
  - 比較時是將兩個運算元逐一位元的相互比較，假如長度不一時，較短的運算元就填0，事件上的相等運算較嚴格。

表6-2　相等運算子

運算式	說明	可能傳回值
a==b	a跟b相等，若是a、b中有x或是z的值的話則傳回x。	0 , 1, x
a!=b	a跟b不相等，若是a、b中有x或是z的值的話則傳回x。	0 , 1, x
a===b	a跟b相等，包括x和z的比較。	0 , 1
a!==b	a跟b不相等，包括x和z的比較。	0 , 1

6.4 運算子的種類

- 相等運算子(Equality Operators)
  - 相等運算子範例如下

```
A = 4;  B = 3;
X = 4'b1010; Y = 4'b1101;
Z = 4'b1xxz; M = 4'b1xxz; N = 4'b1xxx;
A == B; //結果為邏輯 0
X != Y;  //結果為邏輯 1
X = Z;   //結果為 x
Z === M;   //結果為邏輯 1 (所有對應的位元值包括 x 與 z 皆相同)
Z === N;   //結果為邏輯 0 (最小位元值不相同)
M !== N;   //結果為 1
```

6.4 運算子的種類

- 位元運算子(Bitwise Operators)
  - 位元運算子有反(negation, ~)、及(and, &)、或(or, |)、互斥或(xor, ^)與反互斥或(xor, ~^)五種。其運算是將兩種運算元作相對應的逐一位元邏輯運算，若兩運算元長度不一，則較短運算元的不足部份直接以 0 值補齊。

bitwise and	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

bitwise or	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

bitwise xor	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

bitwise xnor	0	1	x
0	1	0	x
1	0	1	x
x	x	x	x

bitwise negation	result
0	1
1	0
x	x

表6-3　位元運算子的真值表

6.4 運算子的種類

- 位元運算子(Bitwise Operators)
  - 位元運算子的範例如下，與一般位元運算不同，反運算(not)只需要一個運算元。

```
X = 4'b1010; Y = 4'b1101;
Z = 4'b10x1;

~X      //反運算結果為4'b0101
X & Y   //位元及運算結果為4'b1000
X | Y   //位元或運算結果為4'b1111
X ^ Y   //位元互斥或運算結果為4'b0111
X ^ ~Y   //位元反互斥運算結果為4'b1000
X & Z    //結果為4'b10x0
```

6.4 運算子的種類

- 位元運算子(Bitwise Operators)
  - 位元運算與邏輯運算不同之處在於，邏輯運算子係針對兩個運算元的真假狀況做運算，位元運算子則是針對兩個運算元的對應位元做運算。
  - 邏輯運算子只產生邏輯0、1或x值，位元運算子則會視運算元的寬度來產生純量或向量。舉例說明如下

```
X = 4'b1010; Y = 4'b0000;
Z = 4'b10x1;

X | Y    //位元運算結果為4'b1010
X || Y   //邏輯運算結果相同於(1 || 0)，結果為 1
```

6.4 運算子的種類

- 化簡運算子(Reduction Operators)
  - 化簡運算子有及(&)、反及(~&)、或(|)、反或(~|)、互斥或(^)、反互斥或(~^與^~皆可)六種。
  - 化簡運算子與位元運算子相同，但只有一個運算元，其運算是針對單一運算元的所有位元做邏輯上的運算，並輸出一個位元的結果。

```
X = 4'b1010; Y = 4'b0000;
Z = 4'b10x1;

&X     //相同於 1&0&1&0，結果為 1'b0
| X    //相同於 1|0|1|0，結果為 1'b1
^X     //相同於 1^0^1^0，結果為 1'b0
// 化簡運算子中的互斥或與反互斥或運算，可以用來計算一個向量變數
// 的奇偶性質(odd or even parity)
```

6.4 運算子的種類

- 移位運算子(Shift Operators)
  - 移位運算子包含有右移(>>)與左移(<<)，以及算術運算的右移(>>>)與算術運算的左移(<<<)。
  - 一般的左右移位是將一個向量運算元的所有位元，整體往右或是往左移動特定的位元數，其中因移位所空出的位元則是補 0。
  - 算術運算的左右移則會視被移動的值來決定空出位元補0或1。

```
X = 4'b1101;
Y = X >> 1;  //向右移一位元並在最高位元填0，結果為4'b0110
Y = X << 1;  //向左移一位元並在最低位元填0，結果為4'b1000
Y = X << 2;  //向左移兩位元得Y值結果為4'b0100
integer a, b, c; // 宣告有號整數
a = 0;
b = -10;  // 111…10110 二補數的二進位表示法
c = a + (b >>> 3)    //結果為十進位 -2 ，使用算術移位運算
```

6.4 運算子的種類

- 連結運算子(Concatenation Operators)
  - 連結運算子({, })可以將不同的運算元連結成單一個運算元，其中欲連結的運算元之位元數必須要清楚，否則無法進行連結。
  - 連結運算子用一個大括號包起來，中間欲連結的項目彼此以逗號分開。其中的運算元可以是單一條線、單一個暫存器，向量的接線、向量的暫存器，或是只是向量接線或向量暫存器的一部份。

```
A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;
Z = 4'b10x1;

Y = { B, C }      // Y值結果為 4'b0010
Y = { A, B, C, D, 3'b001 }     // Y值結果為 11'b10010110001
Y = { A, B[0], C[1] }      // Y值結果為 3'b101
```

6.4 運算子的種類

- 複製運算子(Replication Operators)
  - 複製運算子是將一個運算元作指定次數的複製。

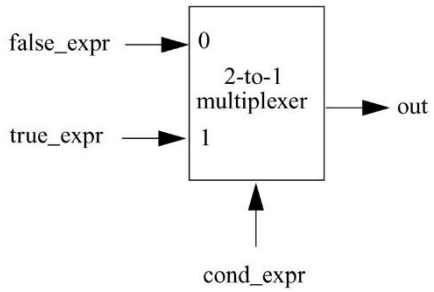
```
reg A;
reg [1:0] B, C;
reg [2:0] D;
A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;

Y = { 4{A} };              // Y值為 4'b1111
Y = { 4{A}. 2{B} };        // Y值為  8'b11110000
Y = {4{A}, 2{B}. D }       // Y值為 11'b11110000110
```

6.4 運算子的種類

- 條件運算子(Conditional Operators)
  - 條件運算子(? :) 要有三個運算元，其格式如下:

    條件運算式 ? 真值運算式 : 假值運算式 ;
  - 當條件運算的結果為 x (不確定值)時，真值與假值運算式的兩個結果會被拿來做逐位的比較。當兩個結果某一個位元相異時，最後輸出的相對位元值為x值，反之若相同時則輸出該位元值。
  - 功能等同一個多工器

```
// 三態緩衝器
assign  addr_bus = enable ? addr_out : 36'bz;
// 二對一多工器
assign out = control ? in1 : in0;
```



  - 巢狀架構

```
assign  out = (A == 3) ? (control ? x : y) : (control ? m : n);
```

6.4 運算子的種類

- 運算子優先順序(Operator Precedence)
  - 若運算式中未使用括號標示特定優先順序，Verilog運算子會依下表來決定:

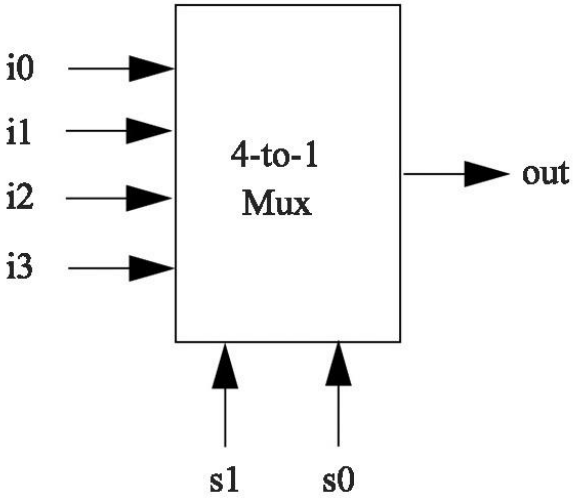
種類	運算子	優先順序
一元運算 乘法、除法、取餘數	+ − ! ~ * / %	最高
加法、減法 移位	+ − << >>	
比較運算 相等運算	< <= > >= == != === !==	
化簡運算	&，~& ^ ^~ |， ~|	
邏輯運算	&& ||	
條件運算	?:	最低

表6-4 運算子的優先順序

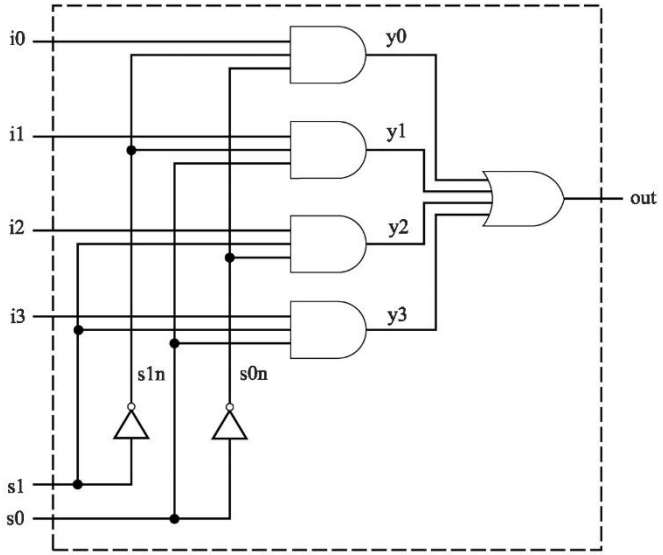
6.4 例 題(Examples)

- 範例6-2 以邏輯方程式的方式設計的四對一多工器
  - **邏輯方程式**:以邏輯指定與邏輯方程式來描述所需邏輯運算

布林函數　out = s1‘ • s0’ • I0 + s1’ • s0 • I1 +
s1 • s0’ • I2 + s1 • s0 • I3;



s1	s0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3



6.4 例 題(Examples)

- 範例6-2 以邏輯方程式的方式設計的四對一多工器

```
// 運用資料處理模型來描寫一個四對一的多工器
// 與運用邏輯閘的方式作比較
model mux4_to_1(out, i0, i1, i2, i3, s1, s0);

// 宣告輸出與輸入埠
output out;
input i0, i1, i2, i3;
input s1, s0;

// out 輸出訊號的邏輯方程式
assign out =  (~s1 & ~s0 & i0)|
              (~s1 & s0 & i1)|
              (s1 & ~s0 & i2)|
              (s1 & s0 & i3);

endmodule
```

6.4 例 題(Examples)

- 範例6-2 以邏輯方程式的方式設計的四對一多工器
  - 模擬程式(觸發模組) stimulus.v

```
module stimulus;
reg IN0, IN1, IN2, IN3;  // Declare variables to be connected to inputs
reg S1, S0;
wire OUTPUT; // Declare output wire
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0); // Instantiate the multiplexer
initial begin
   IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;  // set input lines
   #1 $display("IN0= %b, IN1= %b, IN2= %b, IN3= %b\n",IN0,IN1,IN2,IN3);
   S1 = 0; S0 = 0; // choose IN0
   #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
   S1 = 0; S0 = 1; // choose IN1
   #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
   S1 = 1; S0 = 0;  // choose IN2
   #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
   S1 = 1; S0 = 1; // choose IN3
   #1 $display("S1 = %b, S0 = %b, OUTPUT = %b \n", S1, S0, OUTPUT);
end

endmodule
```

6.4 例 題(Examples)

- 範例6-3 用條件運算子來設計的四對一多工器
  - 比邏輯方程式表示更簡潔

```
// 運用資料處理模型來描寫一個四對一的多工器
// 與運用邏輯閘的方式作比較
module multiplexer4_to_1(out, i0, i1, i2, i3, s1, s0);

// 宣告輸出與輸入埠
output out;
input i0, i1, i2, i3;
input s1, s0;

// 利用條件運算子來描述四對一多工器
assign out = s1 ?(s0 ? i3 : i2):(s0 ?i1 : i0);

endmodule
```

6.4 例 題(Examples)

- 範例6-4 四位元加法器(4-bit Full Adder)
  - 運用加法與連結運算子來設計

```
// 運用資料處理模型來描述一個四位元的加法器
module fulladder4(sum, c_out, a, b, c_in);

// 宣告輸出與輸入埠列
output [3:0] sum;
outputc_out;
input [3:0] a, b;
input c_in;

// 定義全加器的方程式
assign { c_out, sum } = a + b + c_in;

endmodule
```

6.4 例 題(Examples)

- 範例6-5 四位元前瞻進位全加器
  - 漣波進位加法器在計算每個位元的和(sum)前要先知道每個位元的相對進位值(Carry)。因此最高位元的和(sum)在一個n位元的加法器中，需要2n個邏輯閘階層的運算時間才算得出來。為了改善進位的延遲造成漣波進位加法器速度的瓶頸，有人提出了前瞻進位(Carry Lookahead)的方法來改善進位問題。

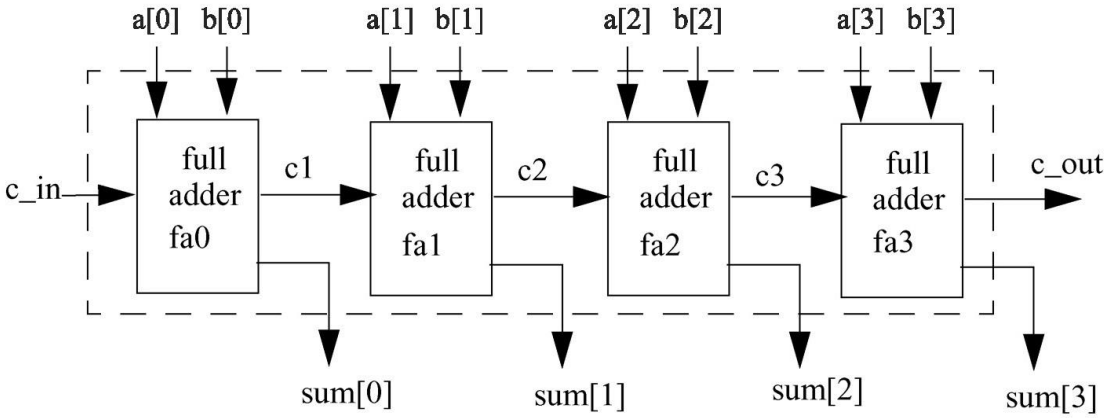
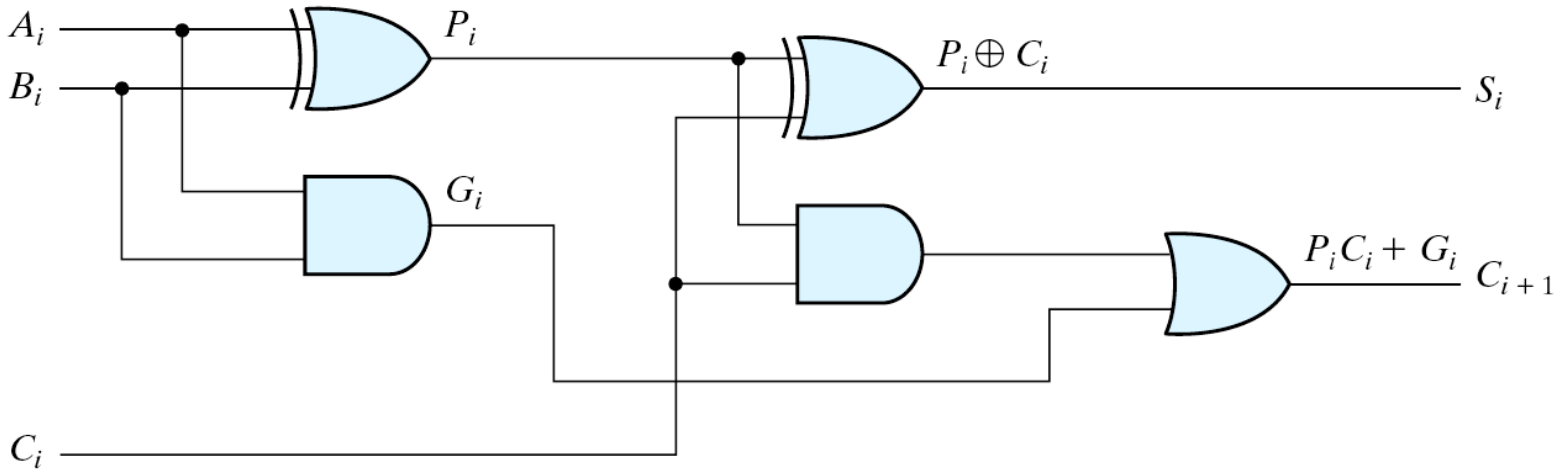


圖5-7 四位元全加器

6.4 例 題(Examples)

- 四位元前瞻進位全加器原理
- *Pi*: carry propagate
- *Gi*: carry generate
- These two signals are common to all full adders and depend only on input augend and addend bits.



6.4 例 題(Examples)

- **Reduce the carry propagation delay**
- employ faster gates
- look-ahead carry (more complex mechanism, yet faster)
- carry propagate: $P_i = A_i \oplus B_i$
- carry generate: $G_i = A_i B_i$
- **Output sum: $S_i = P_i \oplus C_i$**
- **Output carry: $C_{i+1} = G_i + P_i C_i$**
- $C_1 = G_0 + P_0 C_0$
- $C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$
  $= G_1 + P_1 G_0 + P_1 P_0 C_0$
- $C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

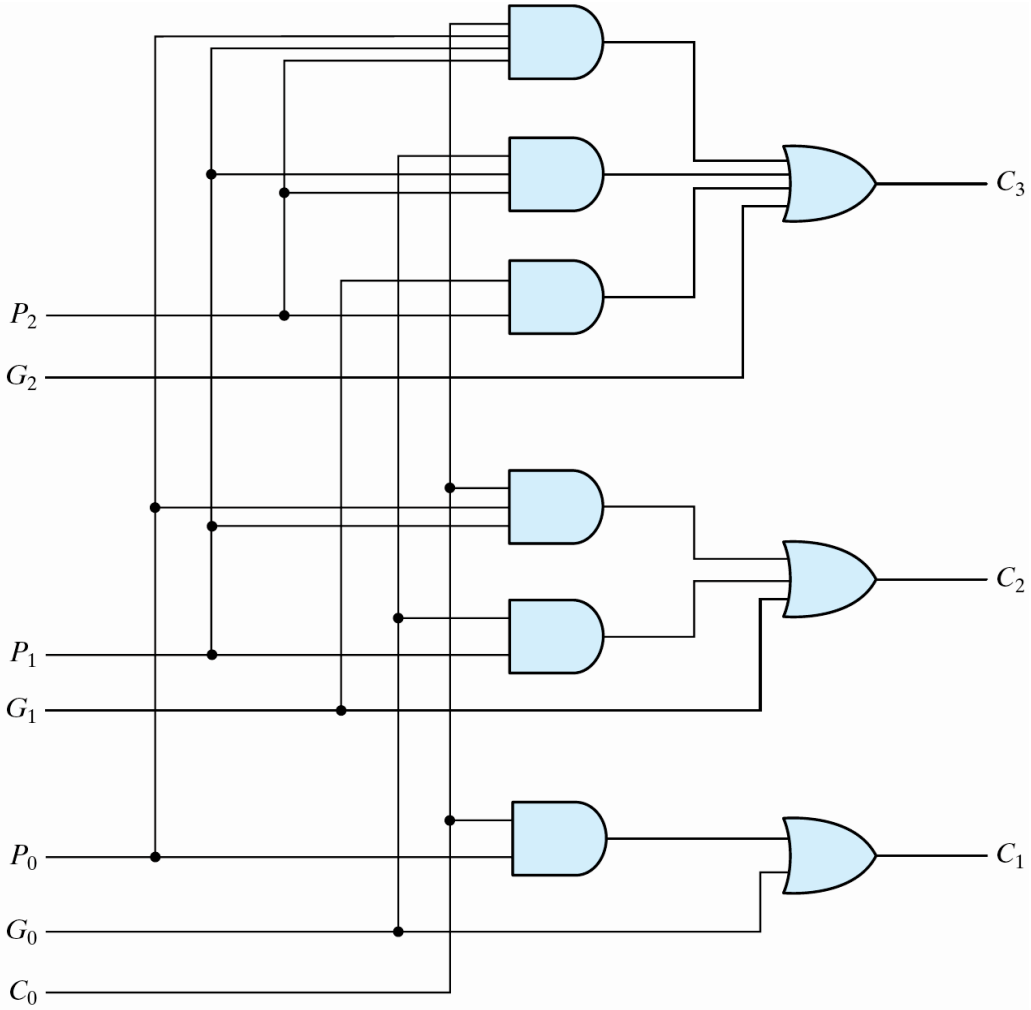
6.4 例 題(Examples)

- Logic diagram

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$



6.4 例 題(Examples)

- 4-bit Adder with Carry Lookahead

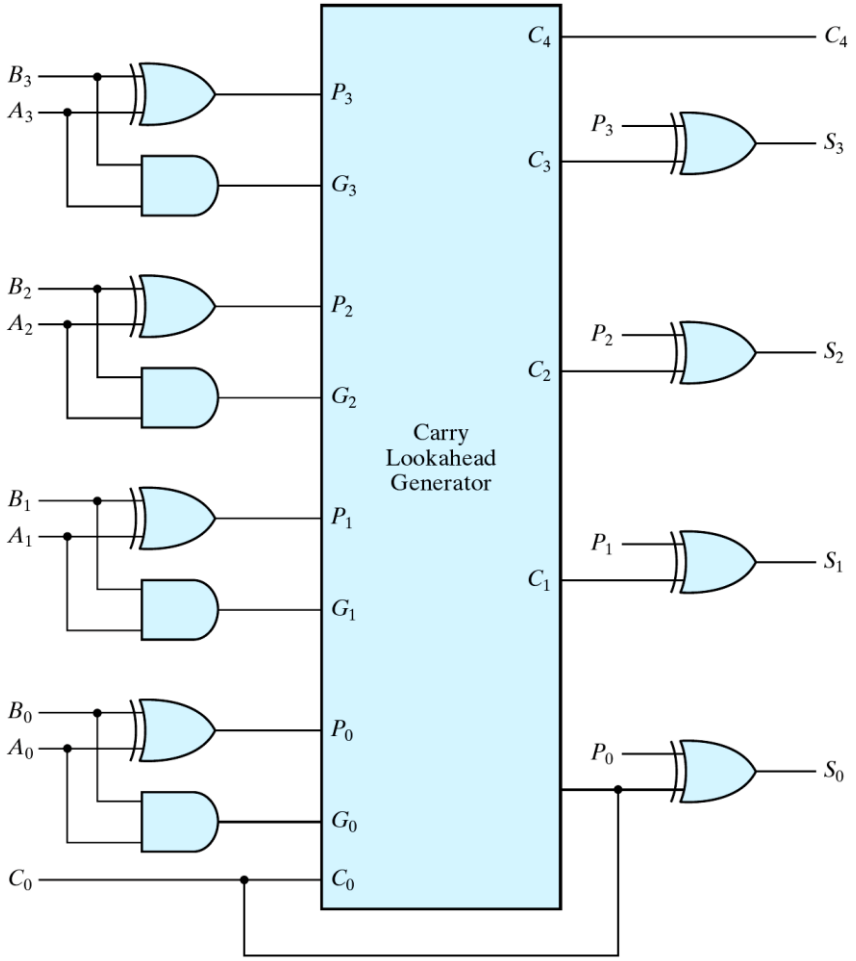
Carry propagate: $P_i = A_i \oplus B_i$

Carry generate: $G_i = A_i B_i$

Output sum: $S_i = P_i \oplus C_i$

Output carry:

$C_{i+1} = G_i + P_i C_i$



6.4 例 題(Examples)

- 範例6-5 四位元前瞻進位全加器

```
module fulladd4(sum, c_out, a, b, c_in);
// 輸出與輸入
output[3:0]sum;
output  c_out;
input[3:0]  a, b;
input  c_in;

// 內部接線
wire p0, g0, p1, g1, p2, g2, p3, g3;
wire c4, c3, c2, c1;

// 計算每階段的 p 訊號
assign p0 = a[0] ^ b[0],
       p1 = a[1] ^ b[1],
       p2 = a[2] ^ b[2],
       p3 = a[3] ^ b[3];

// 計算每階段的 g 訊號
assign g0 = a[0] & b[0],
       g1 = a[1] & b[1],
       g2 = a[2] & b[2],
       g3 = a[3] & b[3];

// 計算每階段的進位值
// 注意方程式中的 c0 訊號等於 c_in 訊號
assign c1 = g0 |(p0 & c_in),
       c2 = g1 |(p1 & g0)|(p1 & p0 & c_in),
       c3 = g2 |(p2 & g1)|(p2 & p1 & g0)|(p2 & p1 & p0 &
                c_in),
      c4 = g3 |(p3 & g2)|(p3 & p2 & g1)|(p3 & p2 & p1 &
                g0)|(p3 & p2 & p1 & p0 & c_in);
// 計算和
assign sum[0] = p0 ^ c_in,
       sum[1] = p1 ^ c1,
       sum[2] = p2 ^ c2,
       sum[3] = p3 ^ c3;
// 指定輸出的進位值
assign c_out = c4;

endmodule
```

6.4 例 題(Examples)

- 漣波進位計數器
  - 我們在這裡要利用負緣觸發(negative-triggerd)正反器，設計一個四位元的漣波進位計數器，設計方式採由上而下(topdown)的模式。
  - 最上層區塊圖

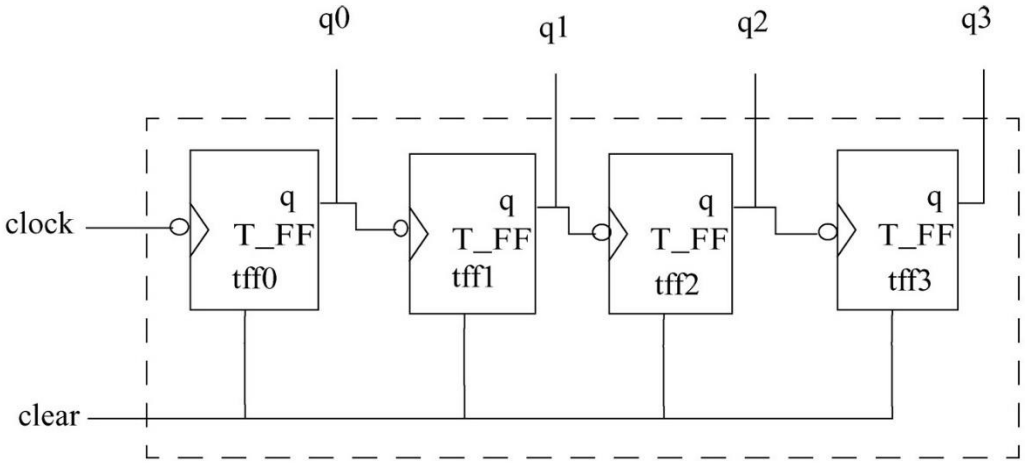


圖6-2　四位元的漣波進位計數器

6.4 例 題(Examples)

- 範例6-6 漣波進位計數器
  - 最上層Verilog程式碼

```
// 漣波進位計數器
module counter(Q , clock , clear);

// 輸出與輸入埠
output [3:0] Q;
input clock, clear;

// 引用 T 型正反器模組
T_FF tff0(Q[0], clock, clear);
T_FF tff1(Q[1], Q[0], clear);
T_FF tff2(Q[2], Q[1], clear);
T_FF tff3(Q[3], Q[2], clear);

endmodule
```

6.4 例 題(Examples)

- 範例6-6 漣波進位計數器
  - 中間層與最下層區塊圖

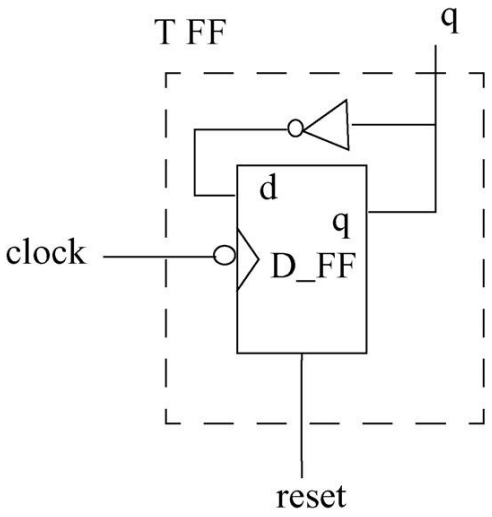


圖6-3 T型正反器

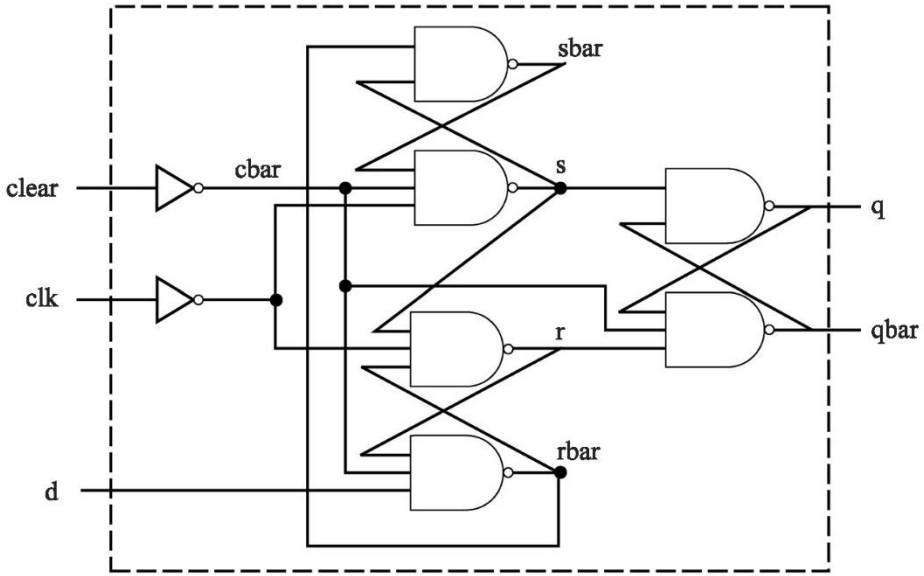


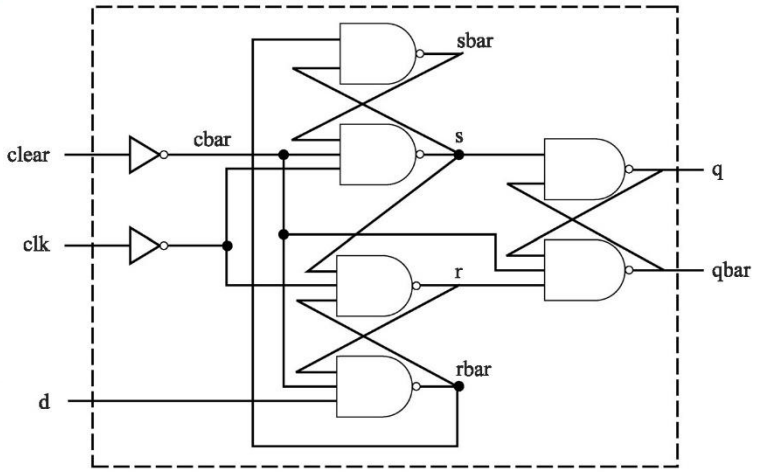
圖6-4 有clear訊號且負緣觸發的D型正反器

6.4 例 題(Examples)

- 範例6-7 T型正反器
  - Verilog程式碼

```
// 負緣觸發的 T 型正反器，並在每個時脈(clk)週期反相一次
module T_FF(q, clk, clear);

// 輸出與輸入埠
output q;
input clk, clear;

// 引用負緣觸發的 D 型正反器
// 並經輸出訊號 q 負迴授到輸入
// 注意到因為沒有用到 qbar 所以 qbar 浮接
edge_dff ff1(q ,~q, clk, clear);

endmodule
```

6.4 例 題(Examples)



- 範例6-8 負緣觸發D型正反器
  - Verilog程式碼

```
// 負緣觸發的 D 型正反器
module edge_dff(q, qbar, d, clk, clear);
// 輸出與輸入埠
output q, qbar;
input d, clk, clear;
// 內部變數
wire s, sbar, r, rbar, cbar;
// 運用資料處理敘述
// 建立訊號 clear 的互補訊號 cbar
assign cbar =~clear;
// 輸入閂 ; 其中閂是位準敏感，一個邊緣敏感的正反器是用三個 SR 閂組成。
assign sbar = ~(rbar & s),
       s = ~(sbar & cbar & ~clk),
       r = ~(rbar & ~clk & s),
       rbar = ~(r & cbar & d);

// 輸出閂
assign q = ~(s & qbar),
       qbar = ~(q & r & cbar);

endmodule
```

6.4 例 題(Examples)

- 範例6-9 漣波進位計數器的觸發模組

```
// 最高階層的觸發模組
module stimulus;

// 宣告觸發輸入訊號
reg CLOCK , CLEAR ;
wire [3:0] Q ;

initial
      $monitor($time," Count Q= %b Clear= %b",Q[3:0],CLEAR);

// 引用設計的計數器模組
counter c1(Q , CLOCK, CLEAR);

// 模擬 Clear 訊號
initial
begin
        CLEAR = 1'b1 ;
        #34 CLEAR = 1'b0 ;
        #200 CLEAR = 1'b1 ;
        #50 CLEAR = 1'b0 ;
end
// 設定 CLOCK 訊號每十個時間單位反相一次
initial
begin
        CLOCK = 1'b0;
        forever #10 CLOCK =~CLOCK;
end

// 在時間 400 的時候結束模擬
initial
begin
        #400 $finish ;
end

endmodule
```

6.4 例 題(Examples)

- 範例6-9 漣波進位計數器的觸發模組
  - 模擬結果如下:

```
  0 Count Q = 0000 Clear= 1
 34 Count Q = 0000 Clear= 0
 40 Count Q = 0001 Clear= 0
 60 Count Q = 0010 Clear= 0
 80 Count Q = 0011 Clear= 0
100 Count Q = 0100 Clear= 0
120 Count Q = 0101 Clear= 0
140 Count Q = 0110 Clear= 0
160 Count Q = 0111 Clear= 0
180 Count Q = 1000 Clear= 0
200 Count Q = 1001 Clear= 0
220 Count Q = 1010 Clear= 0
234 Count Q = 0000 Clear= 1
284 Count Q = 0000 Clear= 0
300 Count Q = 0001 Clear= 0
320 Count Q = 0010 Clear= 0
340 Count Q = 0011 Clear= 0
360 Count Q = 0100 Clear= 0
380 Count Q = 0101 Clear= 0
```

6.4 例 題(Examples)

- 範例6-9 漣波進位計數器的觸發模組
  - 模擬結果波形圖如下: