



VHDL語言入門教學

YiHwa Lai

2003/08/15



大綱

- VHDL語言的基本概念
- 資料型別與資料物件的宣告
- VHDL的敘述語法
- 階層式設計
- 函式、程序與套件
- 問題與討論
- 參考資料





VHDL語言的基本概念

VHDL歷史背景

- VHDL-**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage
- 1983年美國國防部委託IBM、Texas Instrument、Intermetrics負責發展。
- 1987年定義成標準硬體語言，此一標準稱之為**IEEE Std 1076-1987**，又稱為**VHDL 87**。
- 1993年再次更新，更新後之標準稱之為**IEEE Std 1076-1993**，又稱為**VHDL 93**。
- 1996年，IEEE將電路合成的程式標準與規格，加入到VHDL電路設計語言中。

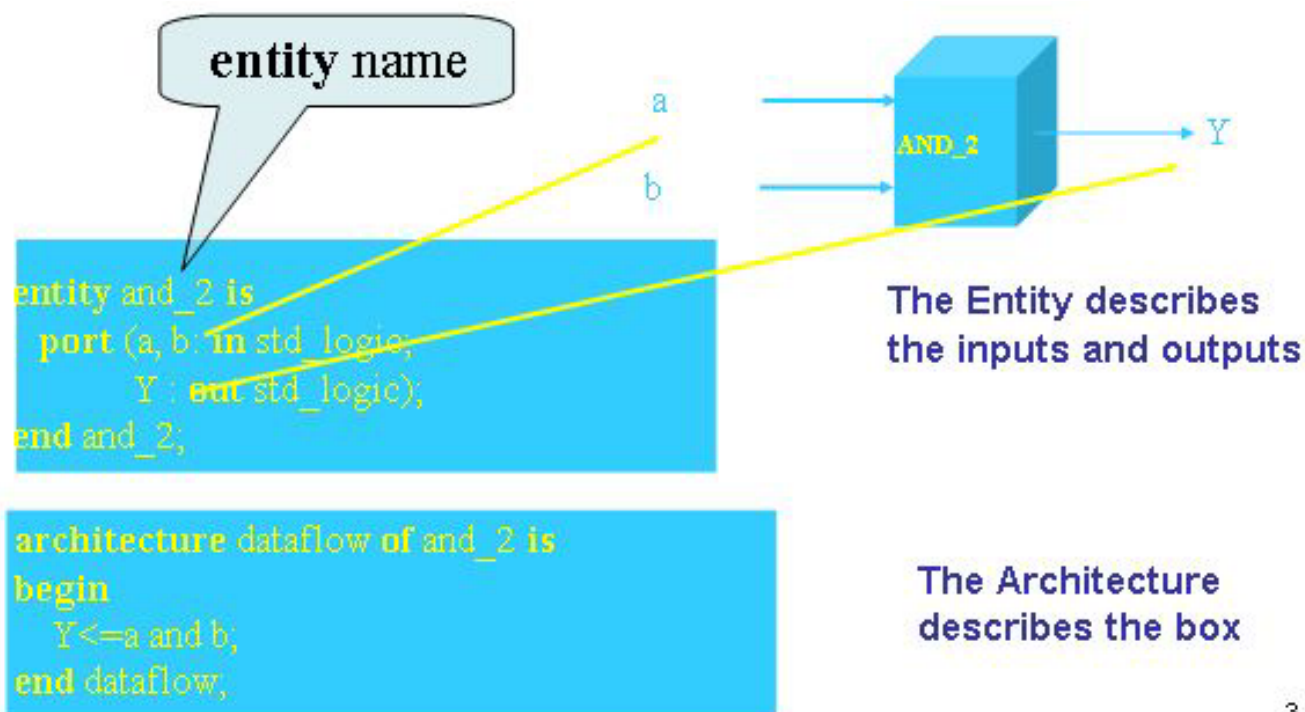


VHDL基本設計觀念

- VHDL語言程式之撰寫，可分為兩個部分：

- 單體(Entity)：電路外觀之描述
- 架構(Architecture)：電路內部功能之描述

Ex：



電路設計中的單體部分

- 單體部分主要分為兩大部分：

- 單體的宣告
- 輸出入埠的描述

- 單體宣告的語法如下：

Entity 單體名稱 is

port(訊號A : 模式 資料型態 ;
訊號B : 模式 資料型態 ;

訊號N : 模式 資料型態) ;

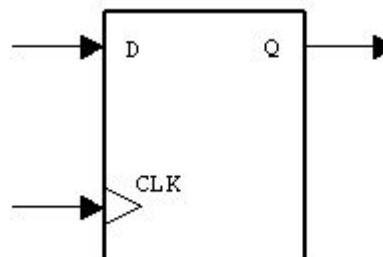
End 單體名稱 ;



電路設計中的單體部分(續)

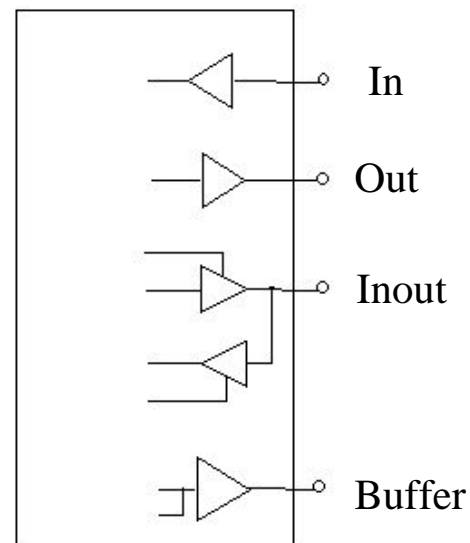
Ex :

```
ENTITY DFF is
PORT(CLK,D: IN STD_LOGIC;
      Q: OUT STD_LOGIC );
END DFF;
```



● 輸出入埠模式

- In：表示該腳位要從外界接收信號
- Out：該腳位將傳送信號到外界
- Inout：可收送雙向模式的Port信號
- Buffer：表示緩衝模式的Port訊號



電路設計中的架構部分

- 電路架構描述部分其語法如下：

Architecture 架構名稱 of 單體名稱 is

{ 架構之宣告區 }

Begin

{ 架構描述程式 }

End 架構名稱 ;

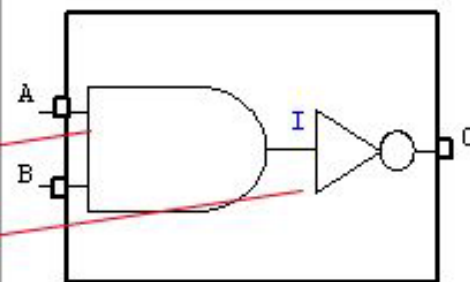


電路設計中的架構部分(續)

- 電路架構描述部分其設計的風格可歸納為三種型式：
 - **結構性描述(Structure Description)**：主要透過元件的宣告與元件的叫用等方式，構成電路中各元件的連線關係。

```

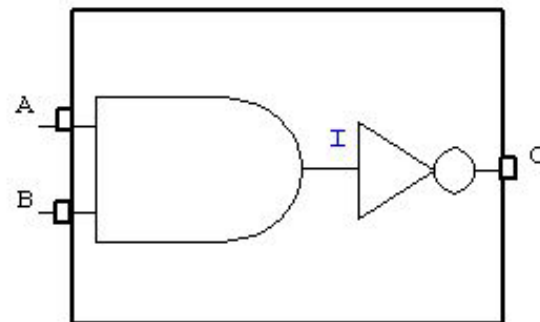
ARCHITECTURE structure OF NAND2 IS
Signal I:BIT;
  component AND_2  --二輸入的NAND元件與其腳位宣告
    port ( I1,I2      : in bit;
           O1         : out bit );
  end component;
  component INVERTER --NOT元件與其腳位宣告
    port ( I1 : in bit;
           O1 : out bit );
  end component;
BEGIN
  Cell1:AND_2 port map(I1=>A, I2=>B, O1=>I);
    --NAND元件腳位之對應連線關係
  Cell2:INVERTER port map(I1=>I, O1=>C);
    --NOT元件腳位之對應連線關係
END structure;
  
```



電路設計中的架構部分(續)

- **資料流型式(Data Flow)**：主要是使用VHDL語言中所內建的標準布林函式為主軸，將各訊號間的布林代數關係，以布林方程式來表示之。

```
architecture Dataflow of NAND2 is  
begin  
    C<=A nand B;  
end Dataflow;
```



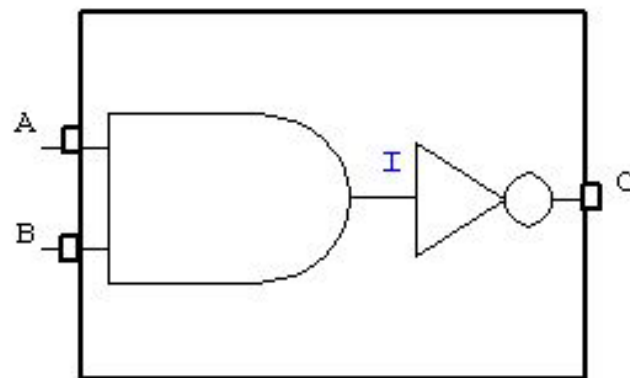
- 布林方程式表示式： $C=(AB)'$ 。



電路設計中的架構部分(續)

- **行為性描述(Behavior Description)**：屬於高階描述方式。主要使用電路特性的行為性描述來設計，如此透過電腦的合成與最佳化，可以加快產品的設計週期，並使設計之電路，非常易於維護。在VHDL語言中的行為性描述電路設計，通常是使用Process的方式達成。

```
architecture behavior of NAND2 is
begin
  process (A,B)
  begin
    if (A='1') and (B='1') then
      C<='0';
    else C<='1';
    end if ;
  end process;
end behavior;
```



10



VHDL程式架構

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity inver_vhdl is
port (   input: in STD_LOGIC;
        output: out STD_LOGIC);
end inver_vhdl;

architecture a of inver_vhdl is
begin
    output <= not input;
end a;
    
```

Use宣告區&標準定義宣告庫

單體宣告區

must refer to entity name

架構宣告區





資料型別與資料物件的宣告

資料的型別

- 邏輯訊號

- 布林代數(Boolean)
- 位元(Bit)
- 標準邏輯(Std_Logic)
- 基本邏輯序列(Bit_Vector)與標準邏輯序列(Std_Logic_Vector)
- 內部訊號宣告(Signal)

- 數值訊號

- 整數(Integer)
- 實數(Real)
- 列舉式(Enumeration)資料型別
- 陣列(Array)資料型別
- 記錄(Record)資料型別



邏輯訊號-Boolean

- 布林代數(Boolean)：

布林代數訊號型式的定義如下：

Type BOOLEAN is (False , True)

上面布林代數的訊號型式包含False和 True兩種狀態：

➤ “False”代表邏輯狀態：’0’

➤ “True”代表邏輯狀態：’1’

【註】Type是VHDL語言中用來宣告訊號型式的指令。

Ex：

```
type boolean is (false, true);
```

```
variable A,B,C: boolean;
```

```
  C := not A
```

```
  C := A and B
```

```
  C := A or B
```



邏輯訊號-Bit

●位元(Bit)

Bit型式是數位邏輯中最基本的邏輯型式，它在VHDL語法中宣告的定義如下：

Type BIT is ('0', '1')

從上面的定義中，我們可以得知BIT的訊號型式共有'0'和'1'兩種類型。

在VHDL中關於BIT型式的邏輯可以適用的運算如下：

邏輯運算	not	and	or	xor	nand	
比較運算	=	/=	<	<=	>	>=
	equal	not equal	less than	less than or equal	greater than	greater than or equal

【註】邏輯運算子“XNOR”僅支援VHDL-93

Ex：

```
type bit is ('0', '1');
```

```
signal x,y,z: bit;
```

```
  x <= '0';
```

```
  y <= '1';
```

```
  z <= x and y;
```



邏輯訊號-Std_Logic

標準邏輯(STD_LOGIC)

在VHDL中標準邏輯訊號型式定義是：

```
Type STD_LOGIC is('X', --Forcing Unknown;浮接不定  
                  '0', --Forcing 0;低電位  
                  '1', --Forcing 1;高電位  
                  'Z', --High Impedance;高阻抗  
                  'W', --Weak Unknown;弱浮接  
                  'L', --Weak 0;弱低電位  
                  'H', --Weak 1;弱高電位  
                  '-', --Don't care;不必理會  
                  );
```

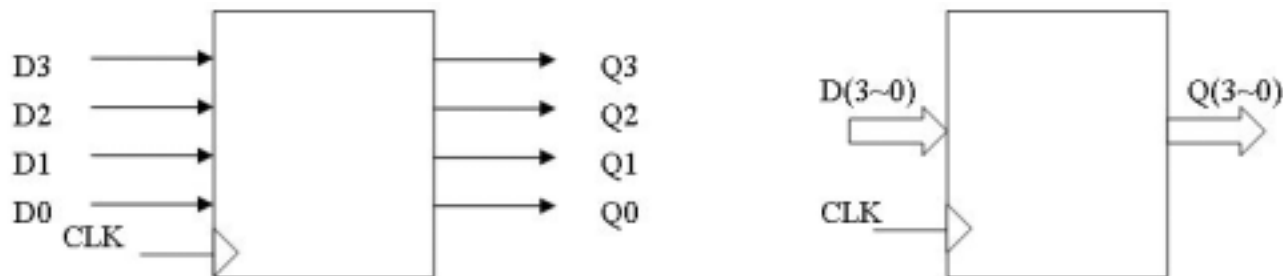
事實上，STD_LOGIC表現的型態就是實際數位電路在輸入或輸出位準所呈現的邏輯特性，它比BIT訊號所能描述的邏輯特性更為真實而且完整；不過在眾多邏輯特性中所謂的Don't care則意指不重要的邏輯，不論是0或1都無所謂，當然在真實的邏輯電路中這種電壓位準是不存在的。



邏輯訊號-Bit_Vector & Std_Logic_Vector

●基本邏輯序列(Bit_Vector)與標準邏輯序列(Std_Logic_Vector)

在數位邏輯電路中，有時候我們會將幾個屬性相同的訊號合成一組代表特定功能的序列訊號，例如微處理機系統中的資料匯流排(Data Bus)或位址匯流排(Address Bus)等。VHDL語法針對這樣的需求也定義了基本邏輯序列(Bit_Vector)與標準邏輯序列 (Std_Logic_Vector) 兩種型態；簡單的說，一群Bit可以構成 Bit_Vector，而一群Std_Logic也可組成Std_Logic_Vector序列。



以上圖為例，我們可以將之以訊號序列型態描述如下：

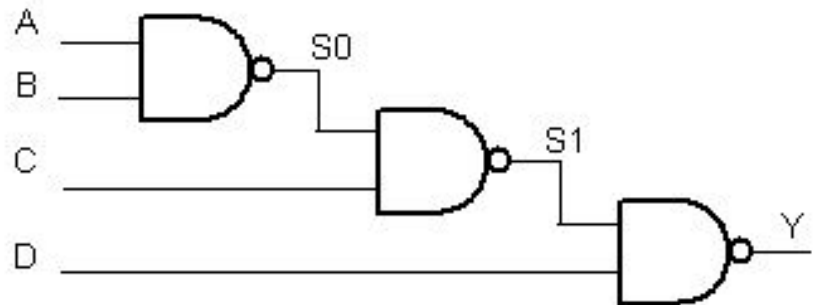
```
Signal D : Std_Logic_Vector(3 downto 0);
Signal Q : Std_Logic_Vector(3 downto 0);
.....
Q<=D;
```



邏輯訊號-Signal

- Signal指令是宣告電路內部自行使用的訊號，因為這類訊號沒有傳送到電路的外部界面，所以通常我們都是在架構(Architecture)中宣告它，而非在VHDL程式的單體(Entity)裡進行宣告。

```
....  
signal S0,S1 : std_logic;  
begin  
    S0 <= A nand B;  
    S1 <= S0 nand C;  
    Y <= S1 nand D;  
end a;
```



數值訊號-Integer Data Type

- VHDL語言中的整數數值範圍：

(- 231 + 1) to (+ 231 - 1)

- Example of Integer Data Type:

Signal A : Integer;

--32位元數值宣告

Signal B,C : Integer range 0 to 7;

-- 3位元數值宣告

Variable INT_S : Integer;

--32位元數值宣告



數值訊號-無號整數(UNSIGNED)序列

- Unsigned指令宣告，它同時具有邏輯和數值的特性，因此既可作邏輯處理又可作數值運算。

Ex:

Signal A : Std_logic;

Signal B,C : Unsigned(3 downto 0);

.....

A<=B(3) and B(2) or B(0) ;

--邏輯處理

C<=B-1;

--數值運算



數值訊號- Real Data Type

- VHDL語言中的實數範圍：

-1.0E38 to 1.0E38

- Example of Real Data Type:

Signal A,B : Real;

.....

A<=3.0 ;

--需要表示成小數點型式

B<=1.5E15;



數值訊號- Enumerated Data Type

- 列舉式資料型別 (Enumeration type) 是一種集合種類的宣告，使用者可以使用一些有意義的名稱，將其定義成集合體的元素。
- Example of Enumeration Data Type:

Type co_state IS (main_green, main_yellow, farm_green, farm_yellow);

Type std_ulogic IS ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

Type week IS (Mon, Tue, Wed, Thu, Fri, Sat, Sun);



數值訊號- Array Data Type

- 陣列資料型別是由同一種資料型別所組成的資料型別。

- Example of Array Data Type:

Type Byte IS Array(7 downto 0) of Bit;

Type Word IS Array(31 Downto 0) of Bit;

- Example (二維的ROM):

Type ROM_table5x4 IS array (0 to 4, 0 to 3) of bit;

Constant ROM_con: ROM_table5x4:=

```
( ('1','0','0','1'),  
  ('0','1','0','1'),  
  ('0','0','1','0'),  
  ('1','0','0','1'),  
  ('0','0','1','1') );
```



數值訊號-Record Data Type

- 記錄資料型別 (Record types) 與陣列資料型別(Array Types)最大的差異是在於記錄資料型別是由兩種或兩種以上資料型別所構成，它可視為族群元素集合體的宣告。
- Example of Record Data Type:

Type MU2 IS Record

Din,Dout: bit_vector(7 downto 0);

S:bit;

End Record;

Signal X,Y:MU2;





資料物件

- 常數 (Constants)
- 訊號 (Signals)
- 變數 (Variables)
- 別名 (Aliases)



資料物件-常數(Constants)

- 在VHDL語言中，我們將固定值宣告成常數，它類以C語言中以#define來設定常數值的作法。
- Example of Constants:

Constant A: Std_logic_vector(3 downto 0):="0011";

Constant Width: Integer:=8;

Constant PI : real := 3.1415926535897 ;

.....



資料物件-訊號(Signals)

- 訊號可以用來宣告所有元件內部的信號線或內接腳位。

- Example of Signals:

Signal A : Std_Logic_vector(4 downto 0);

Signal temp: bit_vector(0 to 3);

.....



資料物件-變數(Variables)

- 變數之處理及設定一定要發生在行為描述程式內，而不可在行為描述程式外，因此變數屬於局部(Local)物件。
- Example of Variables:

Variable temp: Std_logic:= '0';

Variable temp: Std_logic_Vector(3 downto 0);

Variable A,B:Boolean:=False

Variable Name

Data Type

Assignment Statement

Initial Value



資料物件-別名(Aliases)

- 在電路中，若有一條匯流排需要區分成數束不同的子線連接到各個地方去時，我們就可以將原來的母線分開來各自命名，宣告每一束子線並各給其一個別名(Aliases)。
- Example of Aliases:

Signal A_Bus: Std_Logic_Vector(31 downto 0);

Alias Bank1 : Std_Logic_Vector(7 downto 0) IS A_Bus (31 downto 24);

Alias Cal_D : Std_Logic_Vector(15 downto 0) IS A_Bus (23 downto 8);

Alias Rank : Std_Logic_Vector(7 downto 0) IS A_Bus (7 downto 0);



運算子(Operators)

- 邏輯運算子：not and or xor nand xnor
- 關係運算子：= /= < <= > >=
- 算術運算子：+ (加)、- (減)、* (乘)、/ (除)、**(次方)、REM、MOD
- 位移運算子：SLA、SRA、SLL、SRL、ROL、ROR



運算子(Operators)- 邏輯運算子

- Logic operators:

not and or xor nand xnor

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

ENTITY log_ex IS
  PORT( A,B :in std_logic_vector(0 to 3);
        Y  :out std_logic_vector(0 to 3);
        Z  :out std_logic);
end log_ex;
```

```
architecture a of log_ex is
begin
  Y(0) <= A(0) AND B(0) ;
  Y(1) <= A(1) OR  B(1) ;
  Y(2) <= A(2) NOR B(2) ;
  Y(3) <= A(3) XOR B(3) ;
  Z  <= (not A(0)) nand B(0);
end A;
```



運算子(Operators)-關係運算子

- Relational operators:

- = , /= , < , <= , > , >=
- Operands must be of the same type

```
library ieee;  
use ieee.std_logic_1164.all;  
  
ENTITY rel_ex IS  
PORT( A,B,C,D,E,F:in  
      std_logic_vector(2 downto 0);  
      Y: out std_logic);  
end rel_ex;
```

```
architecture behavior of rel_ex is  
begin  
  process(A,B,C,D,E,F)  
  begin  
    if((A=B) or ((C>D) and not(E<=F))) then  
      y<='1';  
    else  
      y<='0';  
    end if;  
  end process;  
end behavior;
```



運算子(Operators)- 算術運算子

- Arith. operators:

➤ + (加)、- (減)、* (乘)、/ (除)、**(次方)、REM(求餘數)、MOD(求商數)

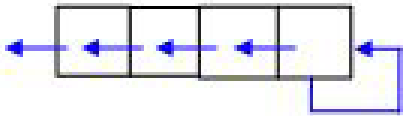
```
library IEEE;  
use IEEE.Std_logic_1164.all;  
use IEEE.Std_logic_unsigned.all;  
  
entity arith_ex is  
port(A,B : in integer range 7 downto 0;  
      C,D : out integer range 64 downto 0;  
      E,F,G : out integer range 7 downto 0);  
end arith_ex;
```

```
architecture a of arith_ex is  
begin  
    C <= A * B;  
    D <= 4**2;  
    E <= 4 mod 2;  
    F <= 6/2;  
    G <= 11 REM 3;  
end a;
```

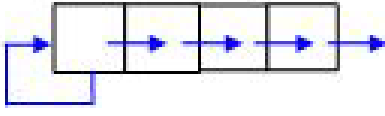


運算子(Operators)- 位移運算子

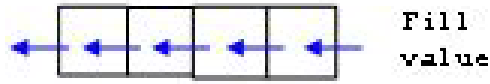
SLA
算術向左位移



SRA
算術向右位移



SLL
邏輯向左位移



SRL
邏輯向右位移



ROL
向左旋轉



ROR
向右旋轉



A="10010101"

A sll 2 為 "01010100"
A srl 3 為 "00010010"
A sla 3 為 "10101111"
A sra 2 為 "11100101"
A rol 3 為 "10101100"
A ror 5 為 "10101100"





VHDL的敘述語法

VHDL的敘述語法

- 在VHDL電路設計語言的架構描述中，可以分成兩種類型的敘述語法：
 - 共時性敘述(Concurrent Statements)
 - 順序性敘述(Sequence Statements)
- 在架構內的每一個程式敘述，均代表著同時發生的性質，但在Process敘述內(或次程式)的程式敘述，則稱之為順序性敘述。
- 在電路架構中的每一個Process敘述或次程式，其均代表著共時性的意義。

Process(...)

Begin

{順序性描述}

End Process;

Process(...)

Begin

{順序性描述}

End Process;

訊號指定敘述;

共時性意義



共時性敘述

- 直接設定敘述：使用 “<=” 運算子
- 條件式的訊號設定敘述：when-else
訊號Y <= 訊號 A when (條件1) else
訊號 B when (條件2) else
訊號 C ;
- 選擇式的訊號設定敘述：with-select-when
with 選擇訊號 X select
訊號Y <= 訊號 A when 選擇訊號 X 為 m ,
訊號 B when 選擇訊號 X 為 n ,
:
訊號 Z when others ;



共時性敘述-when...else敘述

● Example：二對四解碼器電路

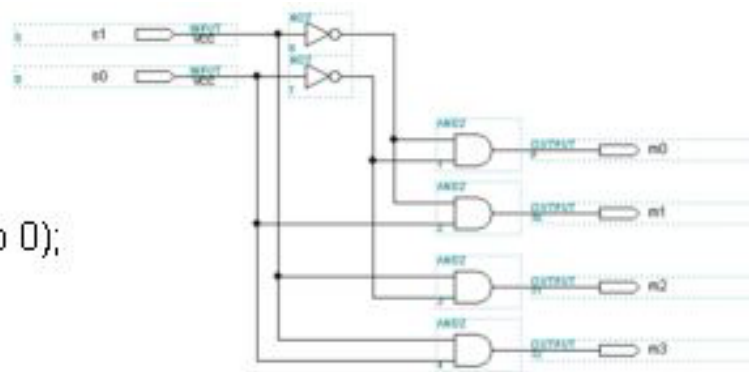
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
ENTITY decoder2_4w IS
PORT (X : IN STD_LOGIC_VECTOR(1 downto 0);
      Y0,Y1,Y2,Y3 : OUT STD_LOGIC);
END decoder2_4w;
```

```
ARCHITECTURE a OF decoder2_4w IS
BEGIN
```

```
  Y0 <= '1' when X="00" else '0';
  Y1 <= '1' when X="01" else '0';
  Y2 <= '1' when X="10" else '0';
  Y3 <= '1' when X="11" else '0';
```

```
END a;
```



輸入		輸出			
X1	X0	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



共時性敘述-with...select...when敘述

● Example：四對二編碼器電路

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity w_s_c42 is
port( X : IN STD_LOGIC_VECTOR(3 downto 0);
      Y : OUT STD_LOGIC_VECTOR(1 downto 0));
end w_s_c42;

architecture a of w_s_c42 is
begin
  With X Select
    Y <= "00" when "0001",
         "01" when "0010",
         "10" when "0100",
         "11" when "1000",
         "00" when others ;
end a;

```

輸入				輸出	
M3	M2	M1	M0	S1	S0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



順序性敘述-Process敘述

- Process敘述是VHDL語言中，用來描述行為化電路模型的主要敘述。其語法如下：

Label1 : PROCESS (Sensitivity List)

變數宣告區

BEGIN

PROCESS主體敘述

END PROCESS Label1;

- 在PROCESS 的程式主體內主要是由順序性敘述，如if...then...else等敘述方法來描述電路的行為化模型。



IF敘述-if...then...end if

- IF敘述的第一種架構為只有IF...Then...End IF，其語法如下：
 if (條件式) then
 命令敘述;
 end if;
- 此種架構主要用途是用來描述記憶元件如正反器、閥鎖器...等。

Ex :

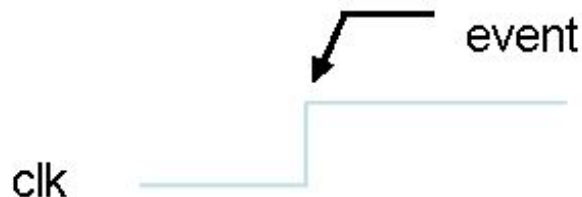
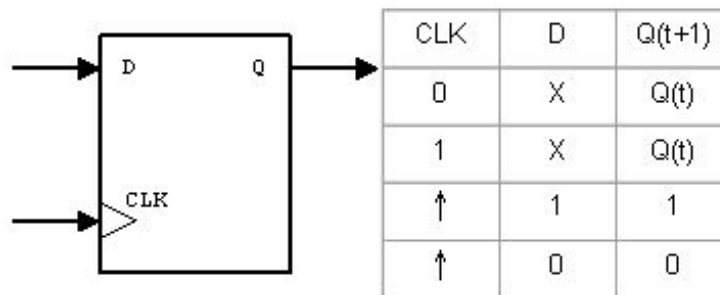
```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY dff_v is
PORT( CLK,D : IN  STD_LOGIC;
      Q      : OUT STD_LOGIC );
END dff_v;
  
```

```

ARCHITECTURE a OF dff_v IS
BEGIN
  PROCESS (CLK)
  BEGIN
    IF CLK'event AND CLK='1' THEN
      Q <= D;
    END IF;
  END PROCESS;
END a;
  
```



IF敘述-if...else...end if

- IF敘述的第二種架構為具有else指令的架構，其語法如下：

if (條件式) then

命令敘述 1;

else

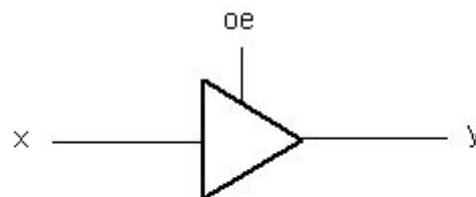
命令敘述 2;

end if;

Ex :
library IEEE;
use IEEE.std_logic_1164.all;

entity tri_gate **is**
port (oe, X : **in** std_logic;
 Y : **out** std_logic);
end tri_gate;

architecture a **of** tri_gate **is**
begin
process (oe, X)
begin
 if oe = '1' **then**
 Y <= X;
 else
 Y <= 'Z';
 end if;
end process;
end a;



三態非反相緩衝器在當Enable=0時，
 輸出是浮接狀態（高阻抗輸出）；
 Enable=1時，輸出和輸入信號相同



IF敘述-if...elsif...else...end if

- IF敘述的第三種架構，具有2個以上的條件式，可以用來描述具有優先順序的選擇邏輯電路，其語法如下：

```
if (條件式 1) then  
    命令敘述 1;  
elsif (條件式 2) then  
    命令敘述 2;  
else  
    命令敘述 3;  
end if;
```



IF敘述-if...elsif...else...end if

Ex :

Library IEEE;

Use ieee.std_logic_1164.all;

Entity MUX41 **IS**

PORT(A,B,C,D:**IN** std_logic;

S:**IN** std_logic_vector(1 downto 0);

X:**OUT** std_logic);

END MUX41;

Architecture A **of** MUX41 **IS**

BEGIN

PROCESS (s, a, b, c, d)

BEGIN

if (s = "00") **then**

X <= a;

elsif (s = "01") **then**

X <= b;

elsif (s = "10") **then**

X <= c;

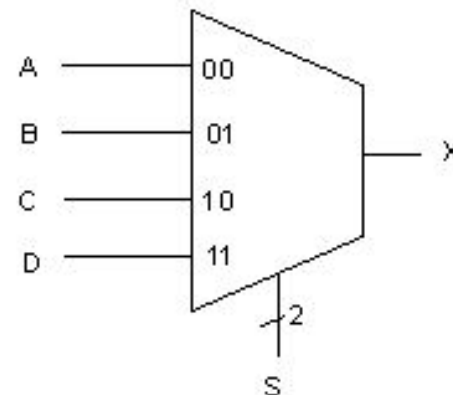
else

X <= d;

end if;

END PROCESS;

END a;



IF敘述-巢狀似敘述

- IF敘述也可以巢狀似敘述，其語法如下：

```
if (條件式 1) then
    if (條件式 2) then
        命令敘述 1;
    else
        命令敘述 2;
    end if;
else
    命令敘述 3;
end if;
```



IF敘述-巢狀似敘述

● Ex :

```
Library IEEE;
Use ieee.std_logic_1164.all;
```

```
Entity MUX41b IS
PORT(A,B,C,D:IN std_logic;
      S:IN std_logic_vector(1 downto 0);
      X:OUT std_logic);
END MUX41b;
```

Architecture A of MUX41b IS

BEGIN

PROCESS (s, a, b, c, d)

BEGIN

if (s = "00") then X <= a;

else

if (s = "01") then X <= b;

else

if (s = "10") then X <= c;

else X <= d;

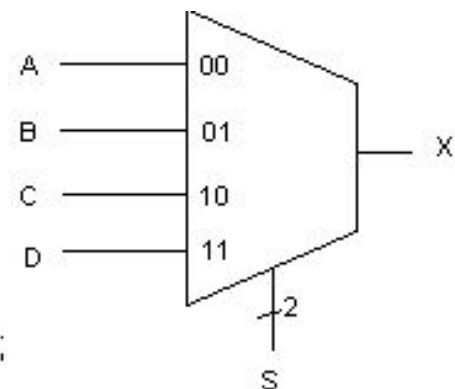
end if;

end if;

end if;

END PROCESS;

END a;



Wait敘述

- Wait Until 條件式

Ex : Wait Until CLK'event and CLK='1' ;

- Wait On 訊號

Ex : Wait On a,b

- Wait For 時間表示式

Ex : Wait For 20ns ;

Ex : Wait For (a*(b+c));



Case敘述-case...when敘述

- Case...when條件敘述指令，可以用來描述一個或一組特定的選擇訊號對於一些電路所做成的選擇邏輯，其語法如下：

Case 選擇訊號 IS

When 選擇訊號1 =>
敘述命令1;

When 選擇訊號2 =>
敘述命令2;

:

When Others =>
敘述命令N;

End Case;



Case敘述-case...when敘述

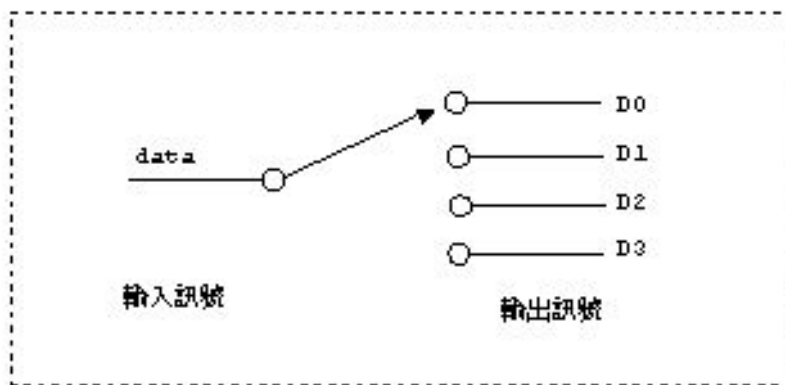
- Ex：一對四的解多工器設計

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

ENTITY demux1to4_case IS
PORT ( data : IN STD_LOGIC;
      S   : IN STD_LOGIC_VECTOR(1 downto 0);
      D0,D1,D2,D3 : OUT STD_LOGIC);
END demux1to4_case;

```



```

ARCHITECTURE a OF demux1to4_case IS
BEGIN
  process(S,data)
  Begin
    D0<='0';D1<='0';
    D2<='0';D3<='0';
    Case S IS
      When "00"=>
        D0<=data;
      When "01"=>
        D1<=data;
      When "10"=>
        D2<=data;
      When Others=>
        D3<=data;
    End Case;
  End Process;
END a;

```



LOOP敘述

- VHDL語言中使用LOOP敘述，來描述重複性的電路操作特性。
- LOOP敘述有三種寫法，分別為：

- for...loop

- 以遞減的方式從開始值執行到結束值為止，其語法如下：

for I in 開始值 downto 結束值 loop

{命令敘述}

end loop;

- 以遞增的方式從開始值執行到結束值為止，其語法如下：

for I in 開始值 to 結束值 loop

{命令敘述}

end loop;

- while...loop

- 其語法如下：

while (條件式) loop

{命令敘述}

end loop;



LOOP敘述

➤ 單純的loop敘述

- 單純的loop敘述會一直反覆的執行，直到exit或next的敘述出現，其條件為真時，才會跳出loop迴圈敘述。其語法如下：

標題名稱：loop

{順序性敘述}

end loop 標題名稱;

- next敘述用來中斷這一次的迴圈運算，而重頭開始迴圈敘述的運算，其語法如下：

next 標題名稱 when 條件式;

- exit敘述用來中斷整個迴圈敘述，程式流程直接跑到迴圈敘述的結尾處，其語法如下：

exit 標題名稱 when 條件式;



LOOP敘述-for...loop敘述 & while...loop敘述

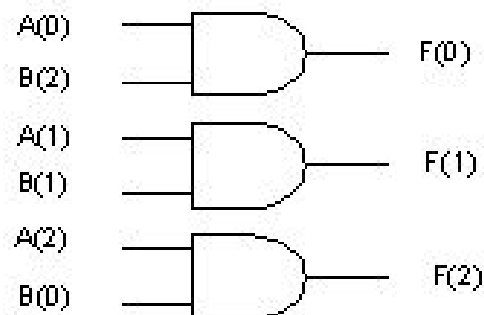
● for...loop

Ex :

```

Process (A, B)
Begin
  LP1: for I in 0 to 2 Loop
    F(I) <= A(I) AND B(2-I);
  end Loop LP1;
end Process;

```



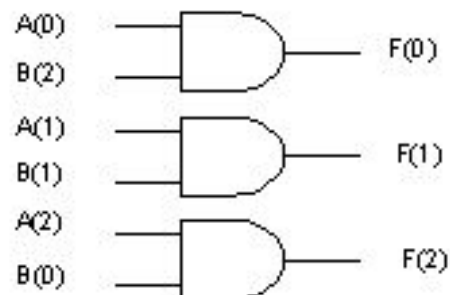
● while...loop

Ex :

```

Process (A, B)
variable J : integer;
Begin
  J := 0;
  LP2: while (J < 3) Loop
    F(J) <= A(J) AND B(2-J);
    J := J + 1;
  end Loop LP2;
end process;

```

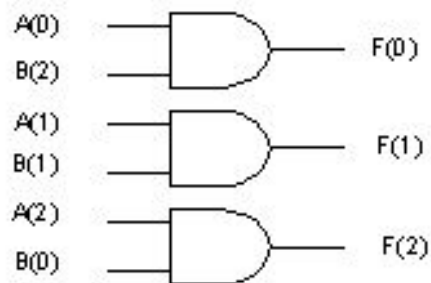


LOOP敘述-單純的loop敘述

- loop指令與exit指令一起使用時，Ex：

```

Process (A, B)
variable K : integer;
Begin
  K := 0;
  LP3: Loop
    F(J) <= A(J) AND B(2-J);
    K := K + 1;
    exit LP3 when (K > 2);
  END Loop LP3;
end Process;
    
```



- loop指令與next指令一起使用時，Ex：

```

LOOP_A:
  FOR I IN (1 TO 10) LOOP
    ... 命令敘述

  LOOP_B:
    FOR J IN (1 TO 10) LOOP
      ... 命令敘述
      NEXT LOOP_A WHEN (J=1);
      ....
    END LOOP LOOP_B;
    ....
  END LOOP LOOP_A;
    
```



階層式設計

方塊(Block)敘述

- Block主要是將同一電路中某一功能的電路以方塊敘述劃分起來，形成一個獨立的電路模組，最後將這些獨立模組組合起來構成我們的電路，模組化的設計方式可以使得系統的維護性和偵錯性大為提高。
- Block方塊敘述的語法如下：

方塊名稱：Block

資料物件宣告區

Begin

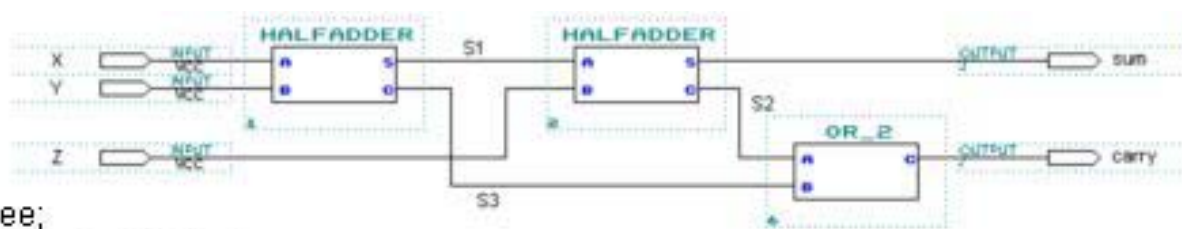
方塊內容程式

END Block 方塊名稱



方塊(Block)敘述

Ex :



```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SUM_b IS
  PORT (X,Y,Z : IN STD_LOGIC;
        sum,carry: OUT STD_LOGIC);

```

```

END SUM_b;
ARCHITECTURE a OF SUM_b IS
  SIGNAL S1,S2,S3: STD_LOGIC;
BEGIN

```

```

  Blk_halfadder1: BLOCK
    --第一個半加器電路模組
  BEGIN
    S1<=X Xor Y;
    S3<=X And Y;
  END BLOCK Blk_halfadder1;

```

```

  Blk_halfadder2: BLOCK
    --第二個半加器電路模組
  BEGIN
    sum<=S1 Xor Z;
    S2<=S1 And Z;
  END BLOCK Blk_halfadder2;

```

```

  Blk_or2: BLOCK
    --OR閘電路模組
  BEGIN
    carry <= S2 or S3;
  END BLOCK Blk_or2;
END a;

```



Component與port map

- Component的功能能夠協助我們作元件資料庫的設計，它與port map結合可以讓我們利用現有的component像堆積木一般累積出複雜的電路。
- Port Map腳位設定的方式如下：

- 位置對應表示式 (must match the port order)

FA1: full_adder PORT MAP (Cin, a0, b0, S0, t1);

- 名稱對應表示式: signal => port_name

FA1: full_adder PORT MAP (Cin=>x, a0=>y, b0=>z, S0=>Sum, t1=>Carry);



Component與port map

Ex : 利用四對一多工器的component建立十六對一多工器

➤ 四對一多工器描述(component)

library IEEE;

use IEEE.STD_LOGIC_1164.all;

ENTITY mux4_1 IS

PORT (D0,D1,D2,D3 ,S1,S0 : IN STD_LOGIC; Y : OUT STD_LOGIC);

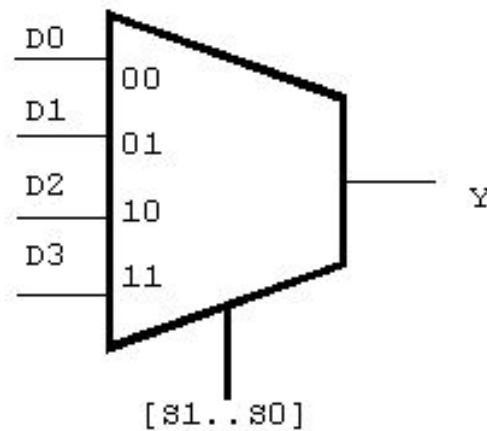
END mux4_1 ;

ARCHITECTURE a OF mux4_1 IS

BEGIN

Y<=(D0 and (not S1)and (not S0))
 or (D1 and (not S1)and S0)
 or (D2 and S1 and (not S0))
 or (D3 and S1 and S0) ;

END a;



Component與port map

➤ 十六對一多工器描述(component)

ARCHITECTURE a OF mux16to1c IS

component mux4_1

port (d0,d1,d2,d3, S1,S0 :**in** std_logic; Y:**out** std_logic);
end component;

signal m :std_logic_vector(0 to 3);

BEGIN

mux1:mux4_1 **port map**(d(0),d(1),d(2),d(3),S(1),S(0),m(0));
 mux2:mux4_1 **port map**(d(4),d(5),d(6),d(7),S(1),S(0),m(1));
 mux3:mux4_1 **port map**(d(8),d(9),d(10),d(11),S(1),S(0),m(2));
 mux4:mux4_1 **port map**(d(12),d(13),d(14),d(15),S(1),S(0),m(3));
 mux5:mux4_1 **port map**(m(0),m(1),m(2),m(3),S(3),S(2),Y);

END a;

Component instance #1 called mux1



Generic和Generic map的使用

- Generic在VHDL語言中，係用來定義元件的參數，如此，VHDL中的元件就成了參數化的元件，而元件 Generic的設定，可以透過Generic Map的方式，將所欲設定的參數數值，傳給參數化元件。
- VHDL語言中的Generic必須在元件的單體宣告區域內來宣告之。

Ex：以一個具有Generic的元件RREG來完成一個8位元的暫存器

➤ RREG部分程式

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.ALL;
```

```
ENTITY rreg IS
```

```
  GENERIC ( Size: INTEGER := 2);
```

```
  PORT ( CLK :IN STD_LOGIC;
```

```
        nRST :IN STD_LOGIC;
```

```
        D:IN STD_LOGIC_VECTOR( Size - 1 DOWNT0 0 );
```

```
        Q:OUT STD_LOGIC_VECTOR( Size - 1 DOWNT0 0 ));
```

```
END rreg;
```

default value



Generic和Generic map的使用

➤ 8位元暫存器部分程式

ARCHITECTURE a OF reg8 IS

COMPONENT rreg

GENERIC (Size: INTEGER := 2);

PORT (CLK :IN STD_LOGIC;

nRST :IN STD_LOGIC;

D :IN STD_LOGIC_VECTOR(Size - 1 DOWNT0 0);

Q :OUT STD_LOGIC_VECTOR(Size - 1 DOWNT0 0));

END COMPONENT;

BEGIN

U1: rreg GENERIC MAP (8)

PORT MAP (CLK => CLK,

nRST => nRST,

D => D,

Q=> Q);

END a;



For-Generate敘述

- VHDL程式提供了For-Generate敘述來描述一些具有重覆性特性的電路，它雖然以迴圈的形式來撰寫，但本身卻是屬於並行敘述指令之一，不可放在Process指令敘述中使用。

- 其格式如下：

標記名稱：For I in 開始值 to 結束值 Generate

.....

END Generate;

Ex:

G1:For I in 3 downto 0 Generate

SUM(I)<=A(I) xor B(I) xor Carry(I);

Carry(I+1)<=A(I) and B(I) and Carry(I);

END Generate G1;



For-Generate敘述

- 以Generate 敘述設計一四位元加法器

ARCHITECTURE a OF full_add4 IS

component FULL_ADD

port(SA,SB,SCin:in bit;

Scout,SUM :out bit);

end component;

signal CARRY:bit_vector(4 downto 0);

BEGIN

CARRY(0)<=cin;

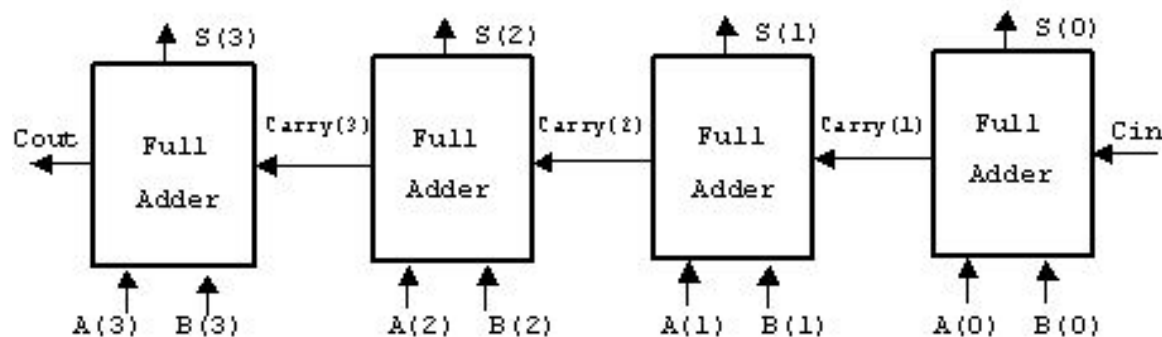
G1:for I in 3 downto 0 generate

FA:FULL_ADD port map (CARRY(I),A(I),B(I),CARRY(I+1),SUM(I));

end generate G1;

cout<=CARRY(4);

END a;





函式、程序與套件

函式(Functions)

- 函式(Functions)的宣告語法如下：

Function 函式名稱 (輸入參數：資料型別) Return 輸出參數型別；

- 函式(Functions)的主體內容語法如下：

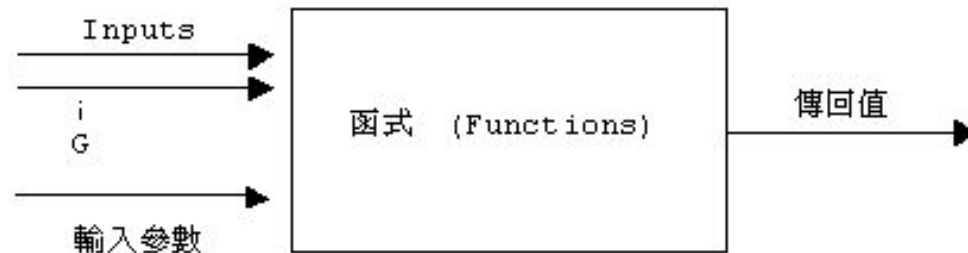
Function 函式名稱 (輸入參數：資料型別) Return 輸出參數型別 IS
函式內的區域變數(Local Variable)宣告區

Begin

函式內的主體內容區

Return 輸出之參數名稱；

END 函式名稱；



函式(Functions)

Ex:函式(Functions)的宣告語法如下：

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
```

```
ENTITY function_calls IS
```

```
    PORT ( A,B,C,D,E: IN STD_LOGIC;
```

```
           Y1,Y2,Y3 : OUT STD_LOGIC);
```

```
END function_calls;
```

```
ARCHITECTURE a OF function_calls IS
```

```
    function Fn1(F1,F2,F3,F4:std_logic) return std_logic is
```

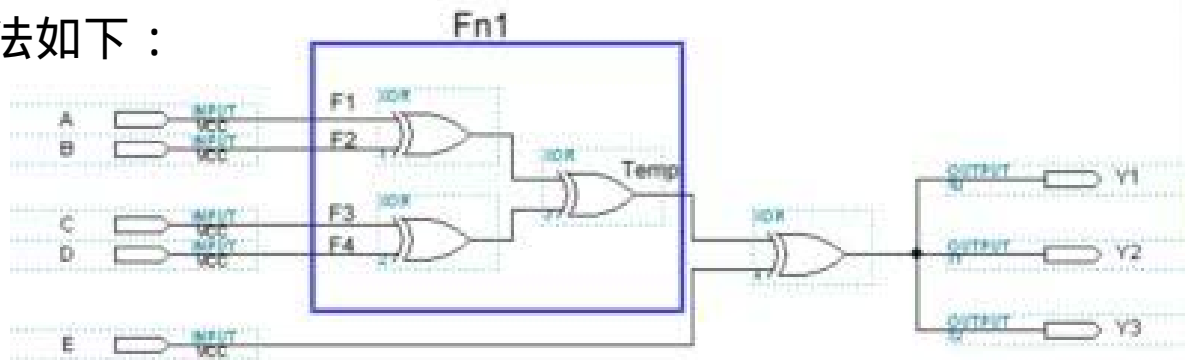
```
        variable temp:std_logic;
```

```
    begin
```

```
        temp:=(F1 xor F2) xor (F3 xor F4);
```

```
        Return temp;
```

```
    end function Fn1;
```



函式(Functions)

BEGIN

process(A,B,C,D,E)

begin

Y1<=Fn1(A,B,C,D) xor E;

--呼叫Fn1函式(使用位置對應表示法)

Y2<=Fn1(F3=>C,F4=>D,F1=>A,F2=>B) xor E;

--呼叫Fn1函式(使用名稱對應表示法)

Y3<=E xor Fn1(A,B,F4=>D,F3=>C);

--呼叫Fn1函式(位置與名稱對應混合法)

end process;

end a;



程序(Procedures)

- 程序(Procedures)的傳回值則可以不限於一個(與函式最大的不同)
- 在VHDL語言中，程序(Procedure)的宣告語法如下：

```
Procedure 程序名稱 ( Signal 訊號A:    資料型別;  
                    Signal 訊號B:    資料型別;  
                    .....  
                    Signal 訊號E:    資料型別;  
                    Signal 訊號M: OUT 資料型別;  
                    Signal 訊號N: OUT 資料型別 ) IS
```

Begin

程序的主體內容

END Procedure;



程序(Procedures)

Ex:

ENTITY adder4 IS

PORT (a,b:in std_logic_vector(3 downto 0);

cin:in std_logic;

sum:out std_logic_vector(3 downto 0);

cout:out std_logic);

END adder4;

ARCHITECTURE structural OF adder4 IS

procedure full_adder(a,b,c:in std_logic;

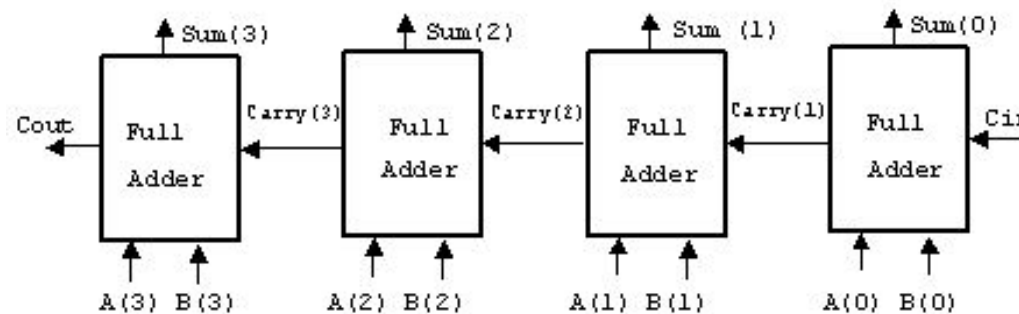
sum,cout:out std_logic) IS

begin

sum:=a xor b xor c;

cout:=(a and b) or (a and c) or (b and c);

end ;



程序(Procedures)

```
BEGIN
process(a,b,cin)
variable result:std_logic_vector(3 downto 0);
variable carry:std_logic;
begin
    full_adder(a(0),b(0),cin,result(0),carry);
        --呼叫一位元全加法器程序
    full_adder(a(1),b(1),carry,result(1),carry);
        --呼叫一位元全加法器程序
    full_adder(a(2),b(2),carry,result(2),carry);
        --呼叫一位元全加法器程序
    full_adder(a(3),b(3),carry,result(3),carry);
        --呼叫一位元全加法器程序

    sum<=result;
    cout<=carry;
end process;
END structural;
```



套件(Packages)

- Package是一種設計單元，它用來宣告一些VHDL中可使用的物件，讓使用者可以將VHDL中的Functions、Procedures及資料型別的定義用一個Package包裝起來，而凡是在套件(Package)內所定義和宣告的東西就變成是公開的，可以在全程式中使用。其語法如下：

```
Package 套件名稱 IS                                --套件宣告部份
    套件宣告部份
END 套件名稱;
Package Body 套件名稱 IS                            --套件主體
    套件主體之內容
END 套件名稱;
```



套件(Packages)

- 當我們的VHDL程式需要使用某一個套件裡所定義的函式或程序時，其語法如下：

USE 目錄名稱.套件名稱.指定的項目；

- 最簡單的例子就是在VHDL程式設計中一定會用使到的：

USE ieee.std_logic_1164.all

其意即為使用IEEE目錄下的std_logic_1164套件內所有的副程式與資料型別。

- 若我們設計的VHDL程式需要用到自己所建立的my_package套件時，語法如下：

USE work.my_package.all

上述的語法說明我們打算使用目前工作目錄(Work Directory)下的my_package套件裡所有的副程式與資料型別。



套件(Packages)

Ex:

```
LIBRARY ieee;
```

```
USE ieee.std_logic_1164.all;
```

```
PACKAGE parity_package IS
```

```
    FUNCTION ODD_par8 (DI: STD_LOGIC_VECTOR(7 DOWNT0 0) ) RETURN STD_LOGIC;
```

```
    FUNCTION ODD_par81 (DI: BIT_VECTOR(7 DOWNT0 0)) RETURN BIT;
```

```
END parity_package;
```

```
PACKAGE BODY parity_package IS
```

```
--Function # 1
```

```
FUNCTION ODD_par8 (DI: STD_LOGIC_VECTOR(7 DOWNT0 0))
```

```
RETURN STD_LOGIC IS
```

```
    VARIABLE temp: STD_LOGIC := '0';
```

```
BEGIN
```

```
    FOR K IN 7 DOWNT0 0 LOOP
```

```
        temp := DI(K) XOR temp;
```

```
    END LOOP;
```

```
    RETURN temp;
```

```
END ODD_par8;
```



套件(Packages)

--Function # 2

```
FUNCTION ODD_par81 (DI: BIT_VECTOR(7 DOWNT0 0))  
RETURN BIT IS  
    VARIABLE temp: BIT := '0';  
BEGIN  
    FOR K IN 7 DOWNT0 0 LOOP  
        temp := DI(K) XOR temp;  
    END LOOP;  
    RETURN temp;  
END ODD_par8;  
  
END parity_package;
```



套件(Packages)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.parity_package.all;

ENTITY PAR_EX IS
    PORT ( D_IN1: IN STD_LOGIC_VECTOR(7 DOWNT0 0);
          D_IN2: IN BIT_VECTOR(7 DOWNT0 0);
          PAR_OUT1: OUT STD_LOGIC;
          PAR_OUT2: OUT BIT);
END PAR_EX;

ARCHITECTURE arch OF PAR_EX IS
BEGIN
    PAR_OUT1 <= ODD_par8(D_IN1); --Require Function # 1;
    PAR_OUT2 <= ODD_par81(D_IN2); --Require Function # 2;
END arch;
```





問題與討論

問題與討論

問題：如何呼叫Verilog程式？

建議：使用component與port map指令，來呼叫Verilog程式。

Ex:

--(verilog 程式)--	--(VHDL程式)--	begin
module add(a,b,c);	library ieee;	S1:add port map(x,y,sum);
input a;	use ieee.std_logic_1164.all;	end adder_arch;
input b;	entity adder is	
output [1:0]c;	port(x: std_logic;	
wire [1:0]c;	y: std_logic;	
	sum: std_logic_vector(1downto 0));	
assign c=a+b;	end adder;	
	architecture adder_arch of adder is	
endmodule;	component add is	
	port(a: std_logic;	
	b: std_logic;	
	c: std_logic_vector (1 downto 0));	
	end component;	



問題與討論

問題：如何簡化VHDL程式？

建議：利用Package設計方法。將常用的資料型別、元件、函式及程序，放入自訂的套件中。

Ex：

➤ 未使用Package時

```
ENTITY PAR_EX IS
```

```
    PORT ( D_IN1: IN STD_LOGIC_VECTOR(7 DOWNT0 0);
```

```
          D_IN2: IN BIT_VECTOR(7 DOWNT0 0);
```

```
          PAR_OUT1: OUT STD_LOGIC;
```

```
          PAR_OUT2: OUT BIT);
```

```
END PAR_EX;
```

```
ARCHITECTURE arch OF PAR_EX IS
```

```
--Function # 1
```

```
FUNCTION ODD_par8 (DI: STD_LOGIC_VECTOR(7 DOWNT0 0))
```

```
RETURN STD_LOGIC IS
```

```
    VARIABLE temp: STD_LOGIC := '0';
```

```
BEGIN
```

```
    FOR K IN 7 DOWNT0 0 LOOP
```

```
        temp := DI(K) XOR temp;
```

```
    END LOOP;
```

```
    RETURN temp;
```

```
END ODD_par8
```



問題與討論

--Function # 2

FUNCTION ODD_par81 (DI: BIT_VECTOR(7 DOWNT0 0))

RETURN BIT IS

VARIABLE temp: BIT := '0';

BEGIN

FOR K IN 7 DOWNT0 0 LOOP

temp := DI(K) XOR temp;

END LOOP;

RETURN temp;

END ODD_par8;

BEGIN

PAR_OUT1 <= ODD_par8(D_IN1); --Require Function # 1;

PAR_OUT2 <= ODD_par81(D_IN2); --Require Function # 2;

END arch;



問題與討論

➤ 使用Package時

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.parity_package.all;
ENTITY PAR_EX IS
    PORT ( D_IN1: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          D_IN2: IN BIT_VECTOR(7 DOWNTO 0);
          PAR_OUT1: OUT STD_LOGIC;
          PAR_OUT2: OUT BIT);
END PAR_EX;

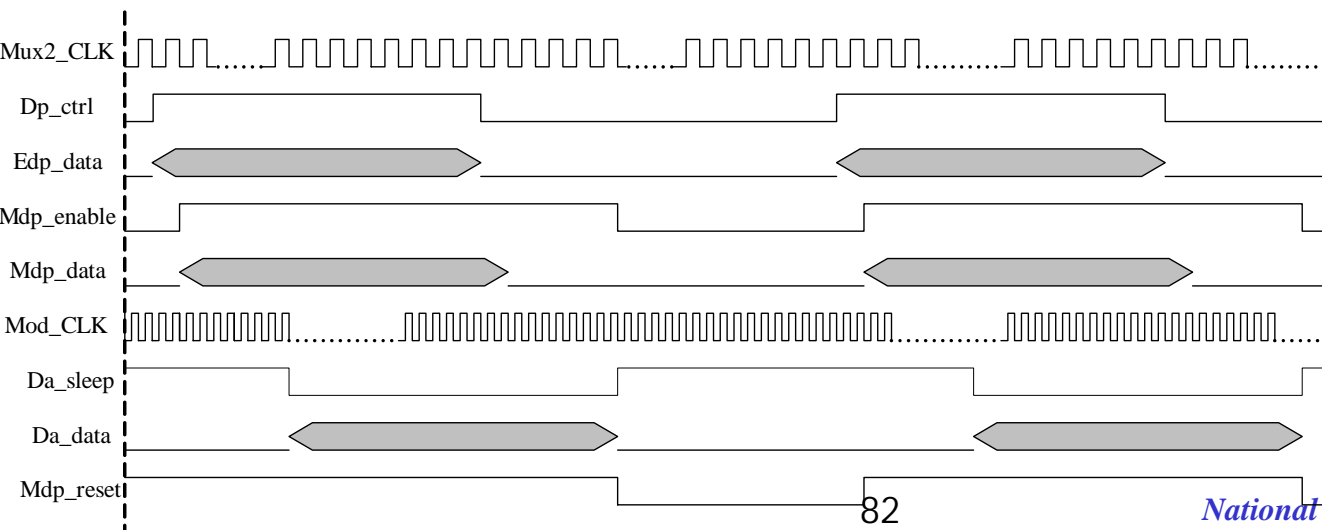
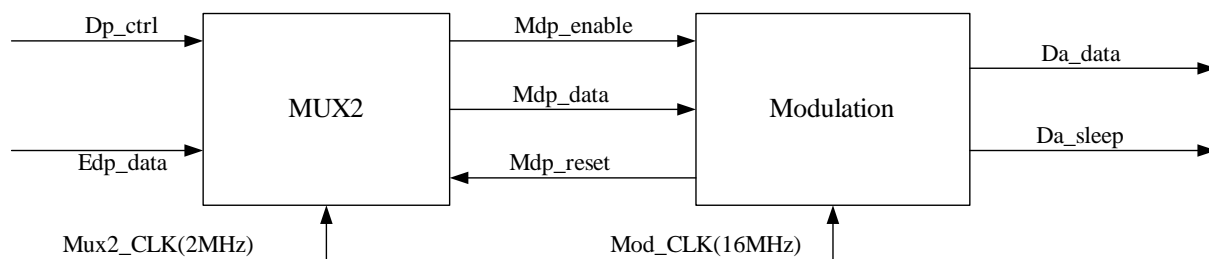
ARCHITECTURE arch OF PAR_EX IS
BEGIN
    PAR_OUT1 <= ODD_par8(D_IN1); --Require Function # 1;
    PAR_OUT2 <= ODD_par81(D_IN2); --Require Function # 2;
END arch;
```



問題與討論

問題：如何整合2個以上功能方塊？

建議：於撰寫VHDL程式之前，先規劃功能方塊間彼此的控制訊號及資料型態，並繪製時序圖再進行程式撰寫。





參考資料

參考資料

- 參考書籍：

- 林傳生，使用VHDL電路設計語言之數位電路設計，儒林
- 黃文吉，VHDL基本程式寫作及應用，儒林

- 參考網站：

- www.vhdl.org
- www.xilinx.com
- www.altera.com

