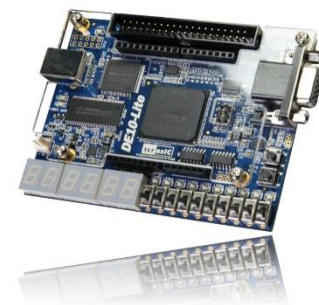


VHDL與組合邏輯電路設計

Outline

- VHDL 介紹
- 單體(Entity)
 - 訊號模式(Mode)
- 架構(Architecture)
- 資料型態(Data Type)
 - 邏輯訊號
 - 數值訊號
- 資料物件(Data Object)
- 運算子(Operator)
- 基本指定敘述「<=」或「:=」
- 條件式指定敘述(Conditional Signal Assignment)
- 選擇式指定敘述(Selected Signal Assignment)
- 自動生成電路



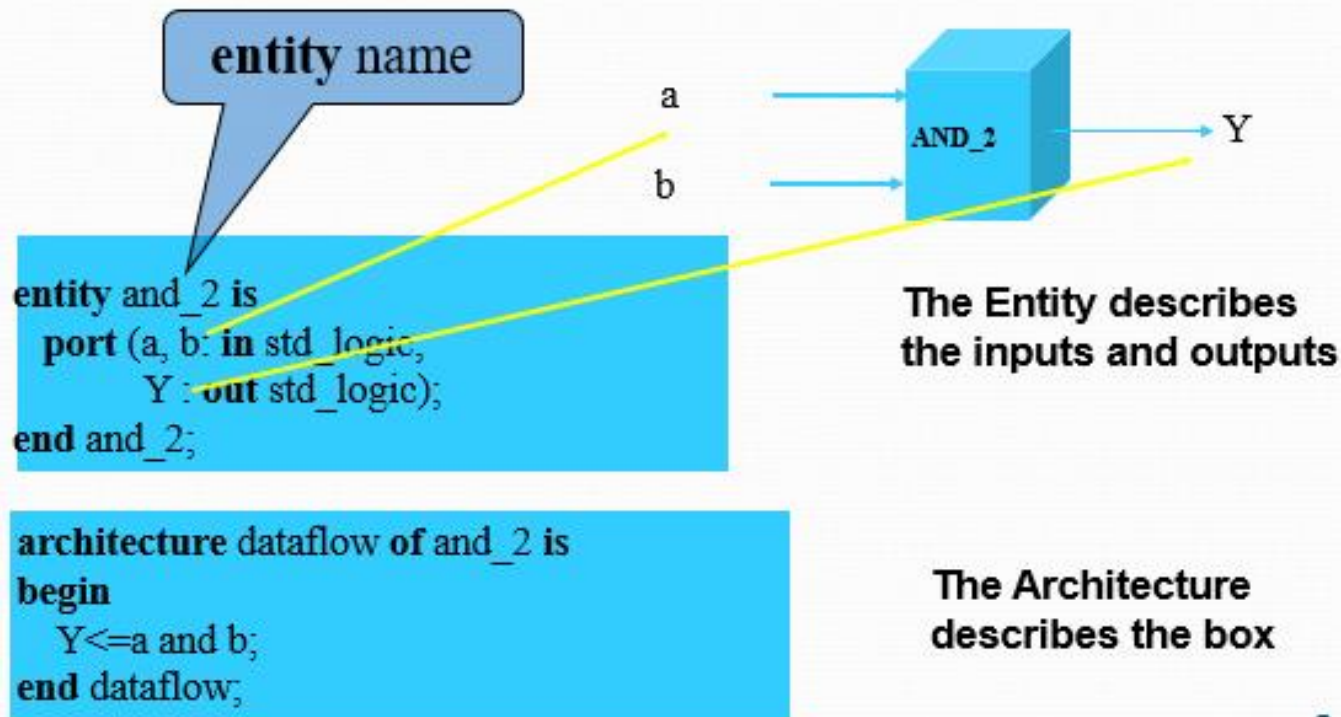
VHDL介紹

- **V**ery **H**igh Speed Integrated Circuit (VHSIC)
Hardware **D**escription **L**anguage
- 1983年美國國防部委託IBM、Texas Instrument、Intermetrics負責發展。
- IEEE Std 1076-1987/1993/2000
- VHDL is a Hardware Description Language, not a programming language.
- 2個主要HDL
 - VHDL
 - Verilog

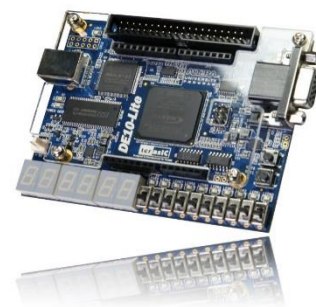


VHDL介紹

- VHDL描述分為兩個部分：
 - 單體(Entity)：電路外觀之描述
 - 架構(Architecture)：電路內部功能之描述



單體 Entity



單體(Entity)

- 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 ;

• • •

• • •

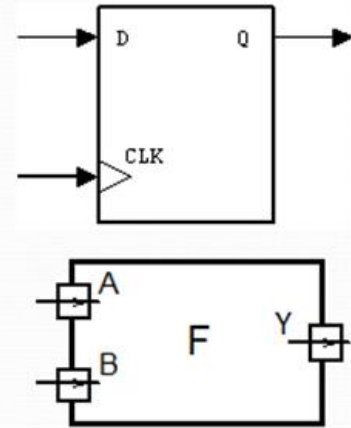
• • •

訊號N : 模式 資料型態) ;

End 單體名稱 ;

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

```
ENTITY NAND2 is
PORT(A,B: IN bit;
      Y : OUT bit );
END NAND2;
```



訊號模式

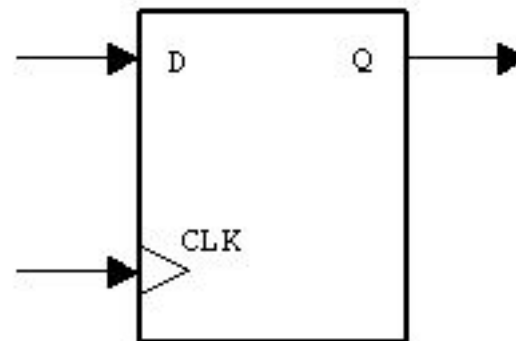
Ex :

ENTITY DFF is

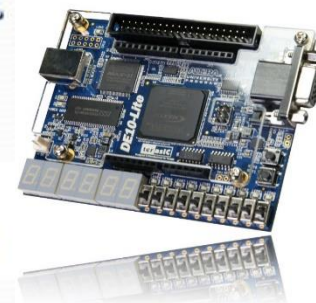
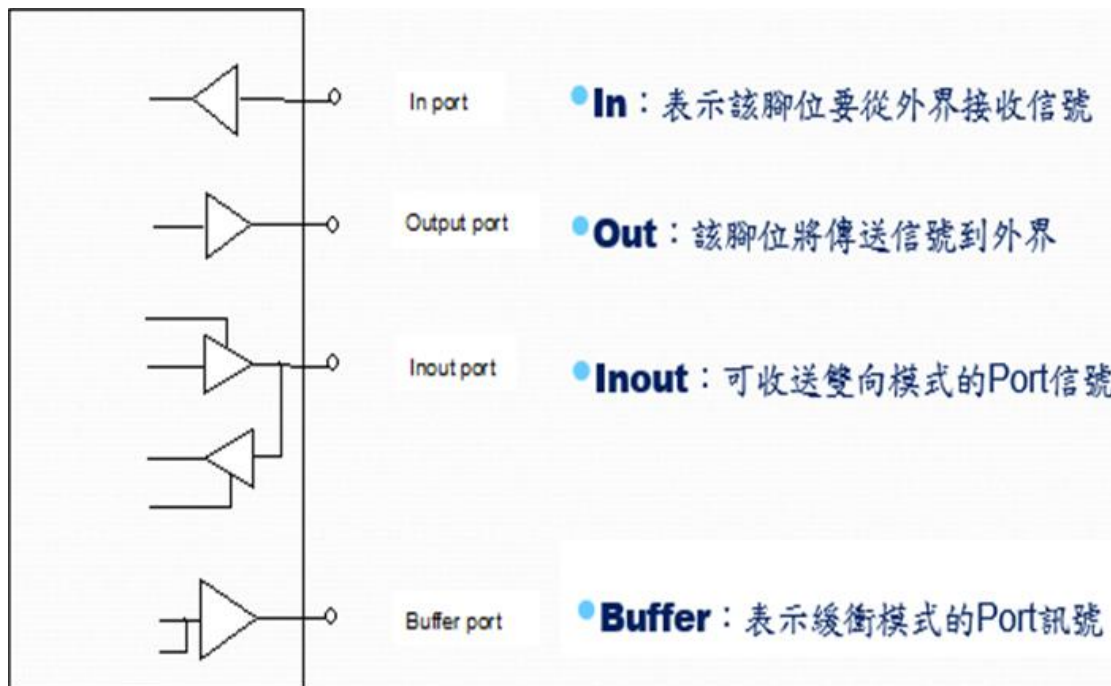
PORT(CLK, D : **IN** STD_LOGIC;

Q: **OUT** STD_LOGIC);

END DFF;



Mode
(信號方向)



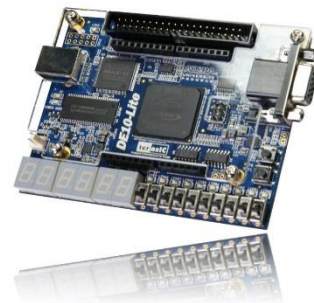
架構

Architecture



架構(Architecture)(1/5)

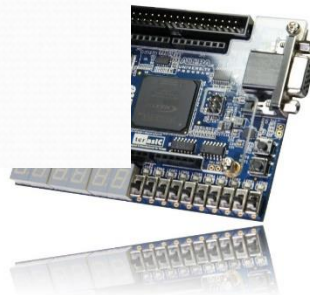
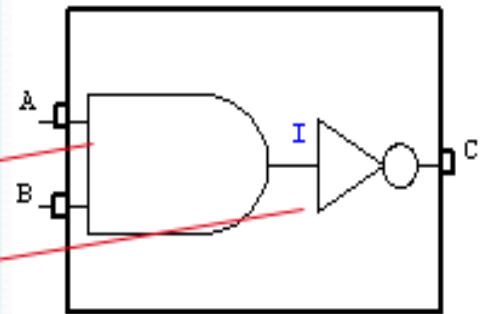
- Architecture的3種描述風格
 - 結構性描述(Structure Description)
 - 資料流描述(Data Flow Description)
 - 行為描述(Behavior Description)
- 電路架構描述部分其語法如下：
 - Architecture** 架構名稱 **of** 單體名稱 **is**
 { 架構之宣告區 }
 - Begin**
 { 架構描述程式 }
 - End** 架構名稱 ;



架構(Architecture)(2/5)

- 結構性描述(Structure Description)

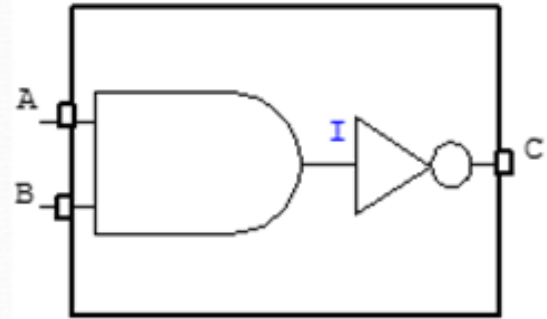
```
ARCHITECTURE structure OF NAND2 IS
Signal I:BIT;
  component AND_2  --二輸入的NAND元件與其腳位宣告
    port ( I1,I2      : in bit;
           O1         : out bit );
  end component;
  component INVERTER --NOT元件與其腳位宣告
    port ( I1 : in bit;
           O1 : out bit );
  end component;
BEGIN
  Cell1:AND_2 port 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;
```



架構(Architecture)(3/5)

- 資料流描述(Data Flow Description)
 - 利用布林方程式來表現各信號之間的布林代數關係
 - 屬於並行(Concurrent)敘述的方式

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



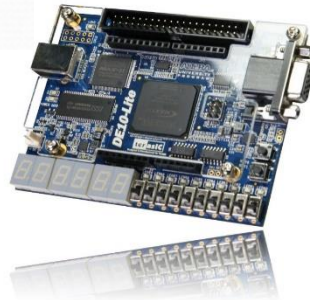
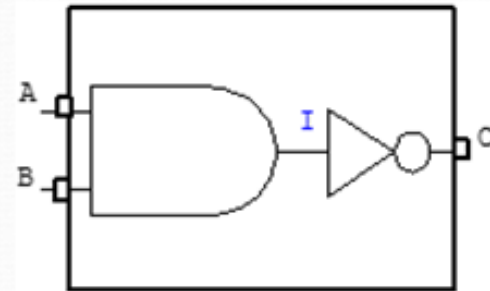
● 布林方程式表示式 : $C = (AB)'$



架構(Architecture)(4/5)

- 行為描述(Behavior Description)
 - 使用Process敘述的方式來完成 (Sequential執行模式)
 - 屬於高階描述方式(High-level Description)

```
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;
```



架構(Architecture)(5/5)

- Example

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
  
entity abc is  
port ( input : in STD_LOGIC;  
       output: out STD_LOGIC);  
end abc;  
  
architecture a of abc is  
begin  
    output <= not input;  
end a;
```

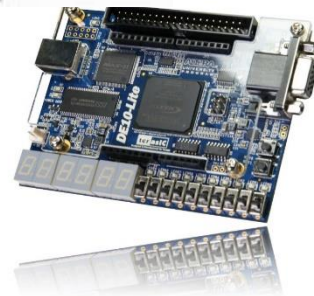
Use宣告區&標準定義宣告庫

單體宣告區

must refer to entity name

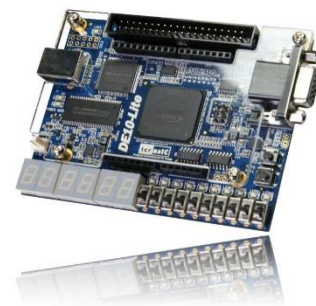
架構宣告區

零件庫	套件檔	敘述
IEEE	std_logic_1164	定義std_logic(標準邏輯)與std_logic_vector(標準邏輯陣列)信號類型，提供基本邏輯運算函數(本單元所需要的功能)與信號類型轉換函數等。



資料型態

Data Type



資料型態-邏輯訊號

- VHDL的資料型態分為邏輯訊號和數值訊號。

資料型態		範圍
邏輯訊號	boolean	true 、 false
	bit	'1' 、 '0 '
	bit_vector	bit 陣列
	std_logic	'U' (大寫)：未初始化 'X' (大寫)：浮接未知 '0'：低電位 '1'：高電位 'Z' (大寫)：高阻抗 'W' (大寫)：弱浮接 'L' (大寫)：弱低電位 'H' (大寫)：弱高電位 '_'：忽略
	std_logic_vector	std_logic 陣列



資料型態-邏輯訊號

- 陣列的遞增與遞減：

A : IN logic_vector(0 to 15) ; -- 遞增

Y : OUT logic_vector(15 to 0) ; -- 遞減

- 邏輯資料的表達方式

A(7) <= '1';

A(7 downto 4) <= "0011";

A <= "1100101100111100";

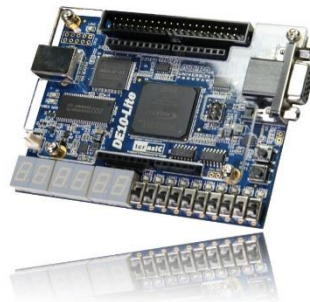
A <= "1100101100111100";

A <= **B**"1100_1011_0011_1100";

A <= **X**"CB3C";

A <= **O**"626";

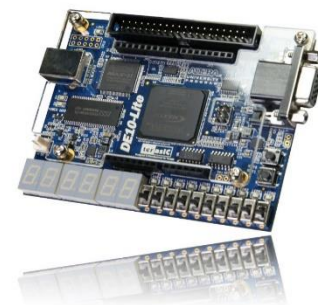
模擬時會以此遞增
或遞減作為訊號預
設順序



資料型態-數值訊號

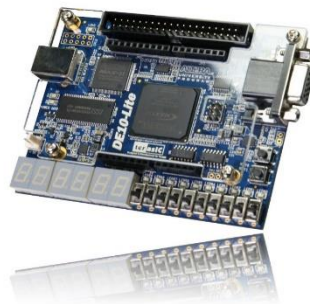
- VHDL的資料型態分為邏輯訊號和數值訊號。
- 數值訊號通常直接使用10進位數字

資料型態		範圍
數值訊號	Integer	範圍： $-(2^{31}-1) \sim (2^{31}-1)$ 例如-2,-1,0,1,2,3 等
	real	範圍： $-1 \times 10^{38} \sim 1 \times 10^{38}$ 例如1.0,-1.0E5
	time	範圍： $-(2^{31}-1) \sim (2^{31}-1)$ 例如1us,7ns,100ps
	character	包括A~Z,a~z,0~9 例如'a','b','2','\$','=' 等
	string	character陣列



資料物件

Data Object



資料物件(1/4)

Constant
一定要設定初始值

資料物件種類	宣告格式與範例
常 數 Constant	Constant 常數名稱: 資料型態 := 初始值; Ex. Constant A: <i>std_logic</i> := '1';
信 號 Signal	Signal 信號名稱: 資料型態(:= 初始值); Signal ci: <i>std_logic</i>;
變 數 Variable	Variable 變數名稱: 資料型態(:= 初始值); Variable W: <i>bit</i>;
檔案 File	File 檔案名稱: text open 操作模式 is "檔名 " File RESULTS: <i>TEXT</i> open <i>WRITE_MODE</i> is "results.txt";



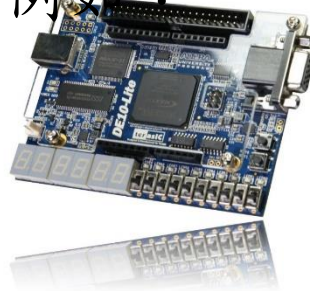
資料物件(2/4)

- 常數(Constant)

Constant是在「**architecture** 與 **begin** 之間」宣告，使用容易解讀(具有代表性但不可為保留字)來代表一個值，如此將提升電路描述的可讀性，且較便於維護與管理。

- 信號(Signal)

Signal是電路內部信號的連接線，可為單線或匯流排(std_logic_vector)。可在實體區(**Entity**)、架構區(**architecture**)、套件(**Package**)的宣告中使用。宣告時，可指定**初始值**(使用「**:=**」為連接符號)；而在描述電路時，也可指定其值(使用「**<=**」為連接符號)，例如：
s1 <= "01101011";



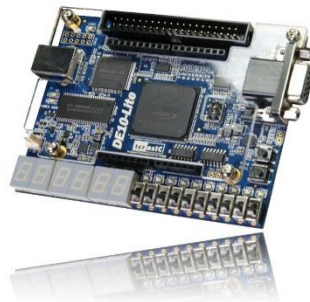
資料物件(3/4)

- 變數 Variable

Variable只能使用在**Process**與副程式(**Function** 或 **Procedure**)等時序性敘述中，其宣告的格式與Signal類似，而有效範圍也侷限在Process或副程式裡。下一頁為Variable 與Signal的比較表。.

- 檔案File

File用來開啟檔案，讀取資料或寫入資料，通常使用於模擬時，撰寫測試平台(Test Bench)。



資料物件(4/4)

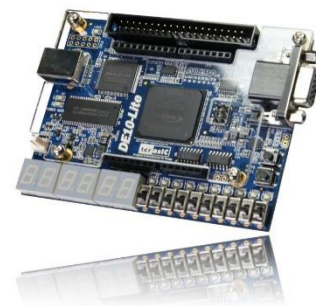
• Signal 與 Variable 之比較

比 較	Signal	Variable
功能	1. 用來儲存或傳遞邏輯值 2. 可被合成為記憶元件或匯流排 3. 可在不同的 Process 之間傳遞	1. 只能用於時序性敘述。 2. 可多次變化，但記憶元件只有一個(中間變化都是組合邏輯產出的中間值)。
宣告位置	宣告於 Architecture 與 Begin 之間	宣告於 Process 與 Begin 之間
宣告範例	Signal A : std_logic;	Variable A : std_logic;
指定方式	信號指定數值以「<=」符號表示信號傳遞	變數指定數值以「:=」表示數值設定
特性	在一次流程中，只會依最後的敘述來決定	在一次流程中，每次經過的敘述都會變化，最後一次敘述是累積下來的結果(就像一般軟體程式一樣，其中間變化都是組合邏輯產出的中間值)
易讀性	較易解讀	不易解讀(無法從中間插入解讀)
資源消耗	可能較少	可能較多(經過的變化皆需一堆組合邏輯電路來支持)



運算子

Operator

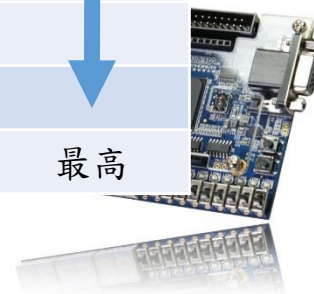


運算子

- VHDL提供邏輯運算子>關係運算子>算術運算子>移位運算子4種運算子。

種 類	運算子	敘述	優先等級
邏輯運算子	and	及運算	最低
	or	或運算	
	nand	反及運算	
	nor	反或運算	
	xor	互斥或運算	
	xnor	互斥反或運算	
關係運算子	=	等於比較	
	/=	不等於比較	
	<	小於比較	
	>	大於比較	
	<=	小於等於比較	
	>=	大於等於比較	

種 類	運算子	敘述	優先等級
算術運算子	+	加運算	
	-	減運算	
	&	連接運算	
	+	正運算	
	-	負運算	
	*	乘運算	
	/	除運算	
	mod	取餘數	
	rem	取餘數	
	**	指數運算	
邏輯運算子	abs	取絕對值	
	not	反向對值	最高



運算子

• 移位運算子

目的訊號 \leftarrow 來源訊號 移位運算子 移位位數;

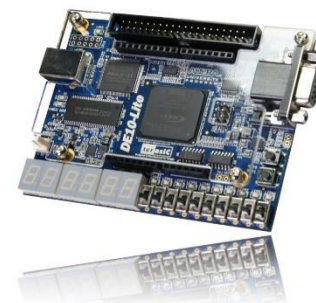
運算子	功能	敘述	
SLL	邏輯左移	資料左移，最右位元補 0	
SRL	邏輯右移	資料右移，最左位元補 0	
SLA	算術左移	資料左移，最右位元補 0	
SRA	算術右移	資料右移，最左位元不變	
ROL	左旋轉	資料左移，最左位元移至最右位元	
ROR	右旋轉	資料右移，最右位元移至最左位元	

ex. :

Y \leftarrow A SLL 2;

若 A="10110010" ，CNT=2，執行後

Y="11001000"



基本指定敘述

- 基本指定敘述是以「 \leftarrow 」或「 $:=$ 」為運算符號，其語法格式如下：

目的信號 \leftarrow 驅動源來源信號或資料；

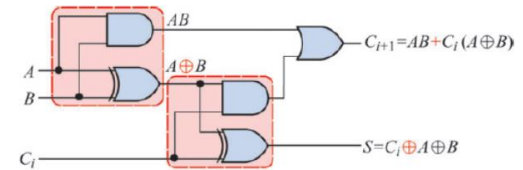
- ex.

$A \leftarrow '1'$ ； -- 將'1'填入A 信號

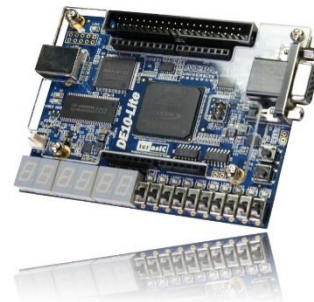
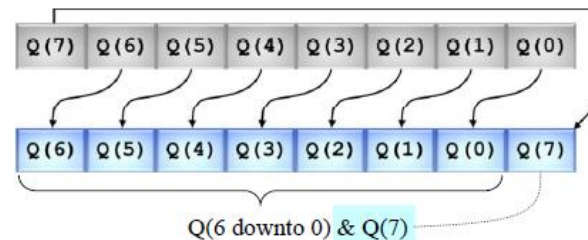
$B \leftarrow \text{not } (C \text{ and } D)$ ； -- 即 $B = C \cdot D$

$\text{sum} \leftarrow A \text{ xor } B \text{ xor } C$ ； -- 和

$\text{carry} \leftarrow (A \text{ and } B) \text{ or } (B \text{ and } C) \text{ or } (A \text{ and } C)$ ； -- 進位



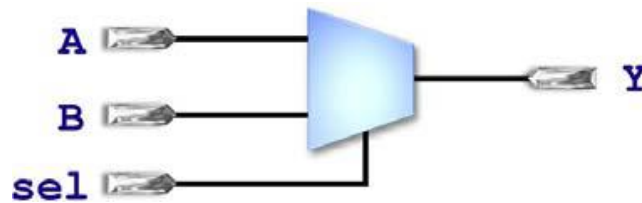
$Q := Q(6 \text{ downto } 0) \& Q(7)$;



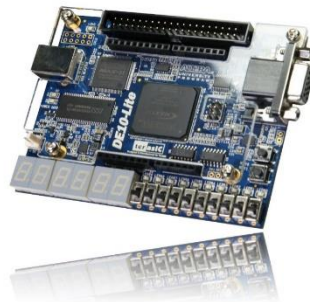
條件式指定敘述(1/3)

- 條件式指定敘述(Conditional Signal Assignment)是在基本指定敘述之中，加入條件判斷，以決定所要指定的值，其敘述語法格式如下：

目的信號 <= 驅動源1 **when** 條件 **else** 驅動源 2;



Y <= A when sel = '1' else B;



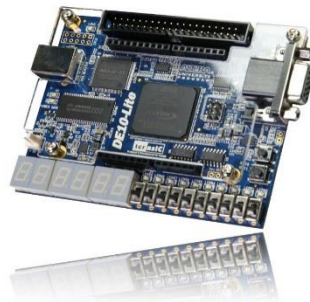
條件式指定敘述(2/3)

- 4 to 1 多工器(資料選擇器)

```
Y <= A when sel = "00" else  
      B when sel = "01" else  
      C when sel = "10" else  
      D;
```

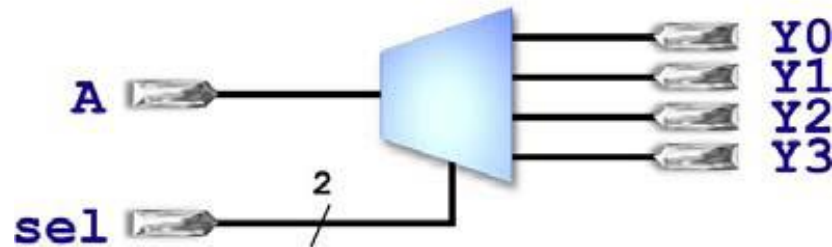
- 使用數值訊號作為判斷條件

```
Y <= A when sel >= 90 else  
      B when sel >= 80 else  
      C when sel >= 70 else  
      D when sel >= 60 else  
      F;
```

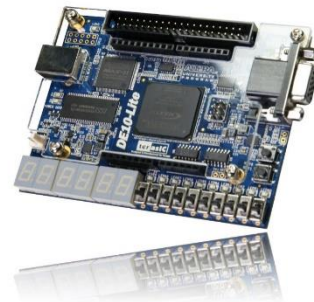


條件式指定敘述(3/3)

- 1 to 4 解多工器



```
Y0 <=A when sel = "00" else 'Z';  
Y1 <=A when sel = "01" else 'Z';  
Y2 <=A when sel = "10" else 'Z';  
Y3 <=A when sel = "11" else 'Z';
```



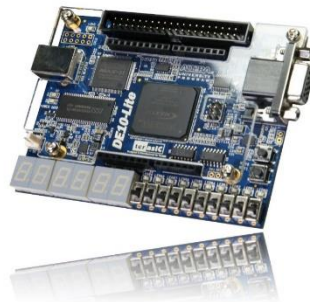
選擇式指定敘述(1/2)

- 選擇式指定敘述(Selected Signal Assignment)與條件式指定敘述之作用類似，但選擇式指定敘述並沒有優先次序問題。其敘述語法格式如下：

With 操作対象 Select

目的信號 <= 驅動源1 When 判斷值1,
 驅動源2 When 判斷值2,
 ⋮

驅動源n When others;

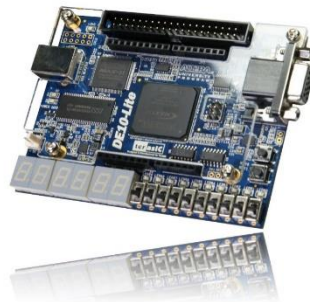


選擇式指定敘述(1/2)

- 2 to 4解碼器



```
With A Select
    Y <= "0001" When "00",
        "0010" When "01",
        "0100" When "10",
        "1000" When others;
```



Component 與 Port-Map 敘述 (1/6)

- Component

- 顧名思義就是零件的意思，使用到的Component都必須放置在同一個專案資料夾內。
- 連結零件的敘述必須在**Architecture**和**Begin**之間。
- 語法格式入下：

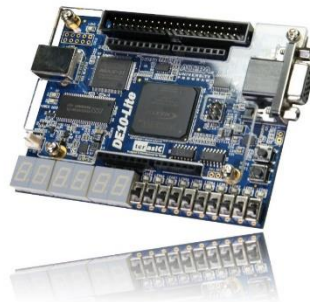
Component 零件名稱

Port(輸出入埠名稱1 : 信號方向資料型態 ;

:

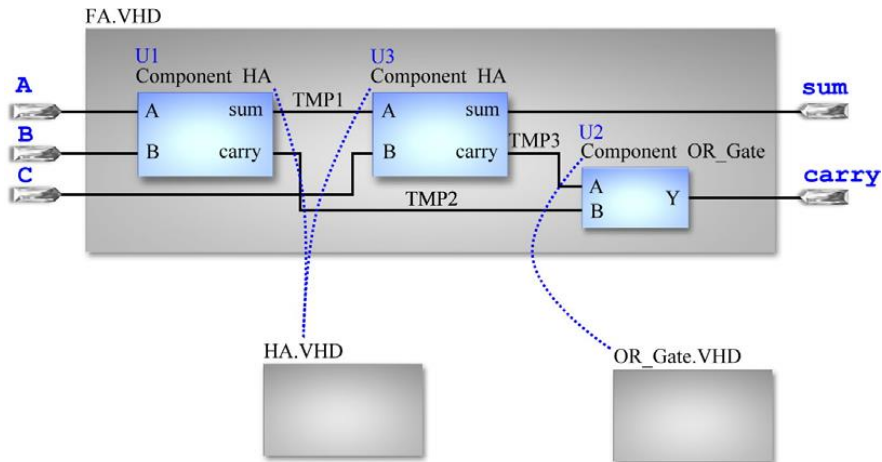
輸出入埠名稱n : 信號方向資料型態);

End Component;



Component 與 Port-Map 敘述 (2/6)

- Component example:



Architecture FA_ARCH of FA is

Component OR_Gate

Port(A,B:in std_logic;

Y:out std_logic);

End Component;

Component HA

Port(A,B:in std_logic;

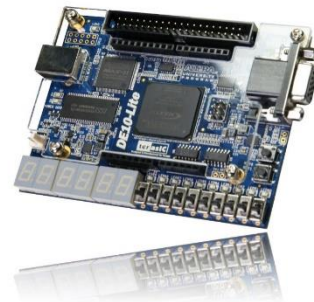
sum,carry:out std_logic);

End Component;

Signal TMP1,TMP2,TMP3:std_logic;

Begin

:

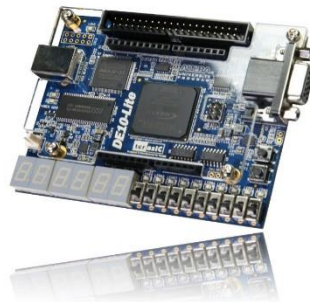


Component 與Port-Map 敘述 (3/6)

- Port-Map

- Port-Map的功能是將Component的接腳與信號做連接，類似電路圖中零件與零件中的連線。
- 語法格式入下：

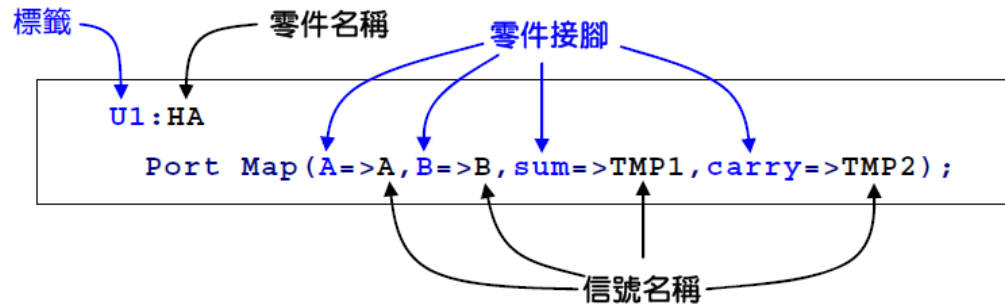
標籤：零件名稱 Port Map(零件接腳1 => 信號名稱1,
零件接腳2 => 信號名稱2,
:
零件接腳n => 信號名稱n);



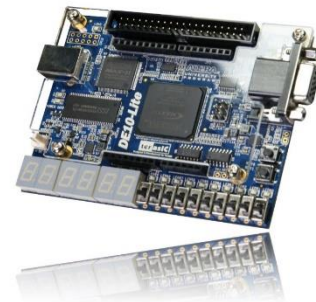
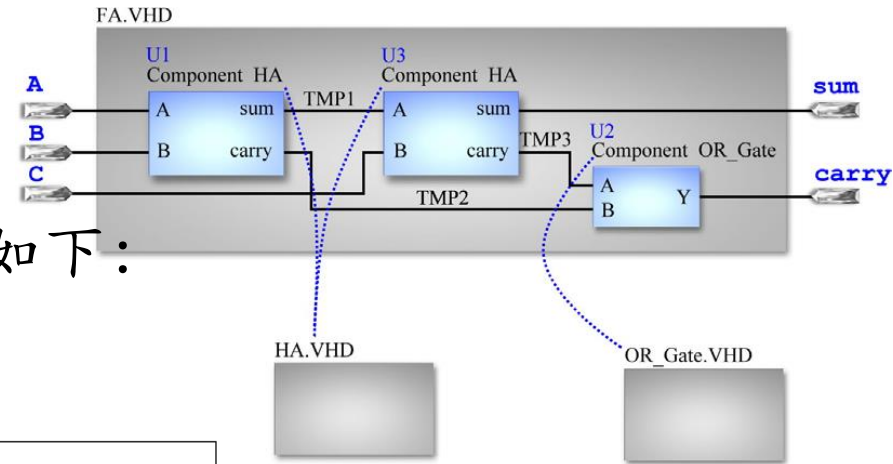
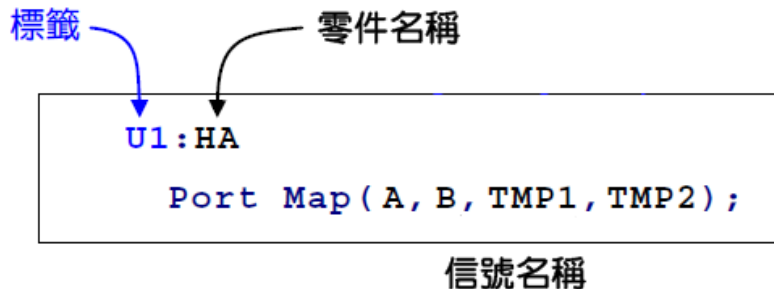
Component 與 Port-Map 敘述 (4/6)

- Port-Map

- 以右圖U1零件為例，連結如下：

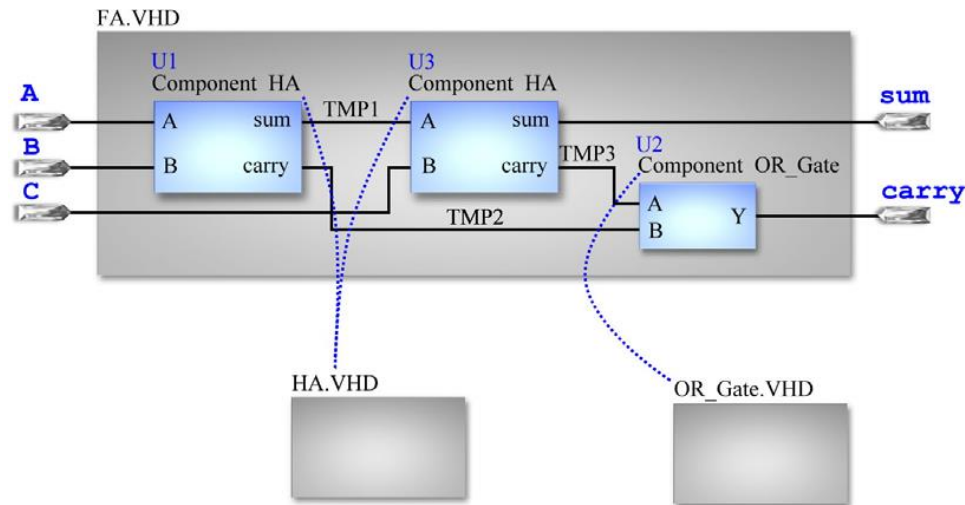


- 亦可僅使用訊號名稱，但使用此方法，訊號順序必須完全對應於該零件的零件接腳順序。



Component 與 Port-Map 敘述 (5/6)

- Port-Map example



Begin

U1:HA

Port Map(A=>A, B=>B, sum=>TMP1, carry=>TMP2);

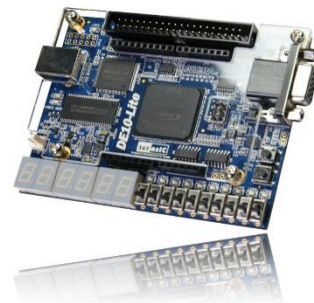
U2:OR_Gate

Port Map(A=>TMP3, B=>TMP2, Y=>carry);

U3:HA

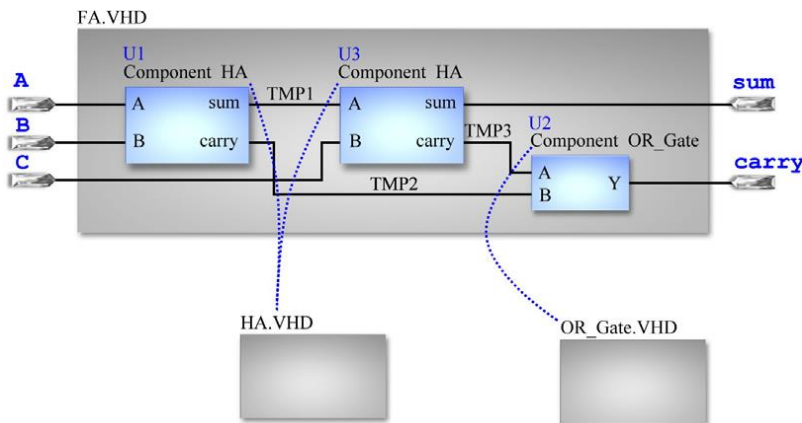
Port Map(A=>TMP1, B=>C, sum=>sum, carry=>TMP3);

End FA_ARCH;



Component 與 Port-Map 敘述 (6/6)

- Component and Port-Map Example:



Architecture FA_ARCH of FA is

```
Component OR_Gate
Port( A,B:in std_logic;
      Y:out std_logic);
End Component;
```

```
Component HA
Port( A,B:in std_logic;
      sum,carry:out std_logic);
End Component;
```

```
Signal TMP1,TMP2,TMP3:std_logic;
```

Begin

```
U1:HA
```

```
Port Map(A=>A, B=>B, sum=>TMP1, carry=>TMP2);
```

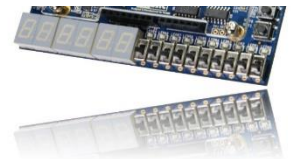
```
U2:OR_Gate
```

```
Port Map(A=>TMP3, B=>TMP2, Y=>carry);
```

```
U3:HA
```

```
Port Map(A=>TMP1, B=>C, sum=>sum, carry=>TMP3);
```

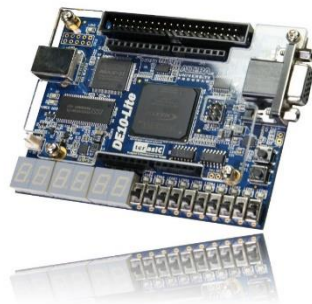
End FA_ARCH;



自動生成電路

For-Generate

If Generate



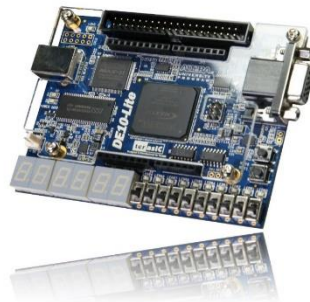
自動生成電路(1/7)

- For-Generate敘述(1/3)
 - 自動產生電路。
 - 適合處理重複性較高的電路。
 - 可使用蜂巢結構
- 語法格式如下：

標籤 1: **For** 迴圈變數 **in** 起始值 **to/downto**
結束值 **Generate**

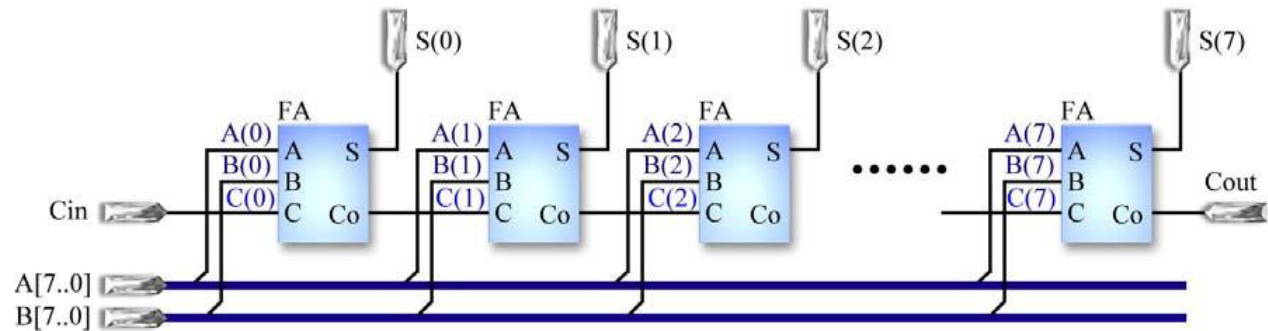
標籤 2: 自我產生電路之描述;
:

End Generate;



自動生成電路(2/7)

- For-Generate敘述(2/3)



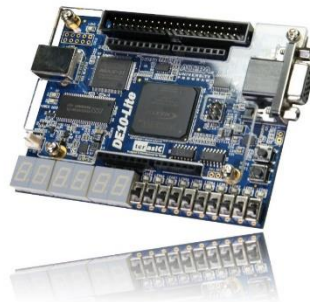
$C(0) \leq Cin;$

ADDER_GEN: For I in 0 to 7 Generate

U: FA Port Map(A(I), B(I), C(I), S(I), C(I+1));

End Generate;

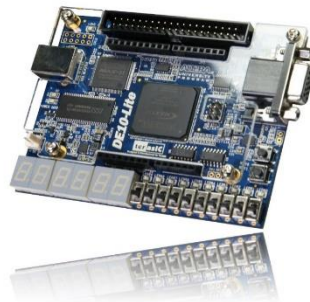
$Cout \leq C(8);$



自動生成電路(3/7)

- For-Generate敘述(3/3)
 - 編譯後，將會產生下列動作:

```
U0: FA Port Map (A(0), B(0), C(0), S(0), C(1));  
U1: FA Port Map (A(1), B(1), C(1), S(1), C(2));  
U2: FA Port Map (A(2), B(2), C(2), S(2), C(3));  
U3: FA Port Map (A(3), B(3), C(3), S(3), C(4));  
U4: FA Port Map (A(4), B(4), C(4), S(4), C(5));  
U5: FA Port Map (A(5), B(5), C(5), S(5), C(6));  
U6: FA Port Map (A(6), B(6), C(6), S(6), C(7));  
U7: FA Port Map (A(7), B(7), C(7), S(7), C(8));
```



自動生成電路(4/7)

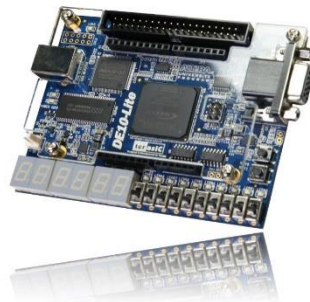
- If-Generate敘述(1/4)
 - 結合If關鍵字，使電路形成具有判斷功能的自我產生敘述。
 - 此條件判斷的功能只是在編譯時，做為產生電路描述的條件，並非建構時序性電路。
- 語法格式如下：

標籤 1 : **If** 判斷條件 **Generate**

標籤 2 : 自我產生電路之描述；

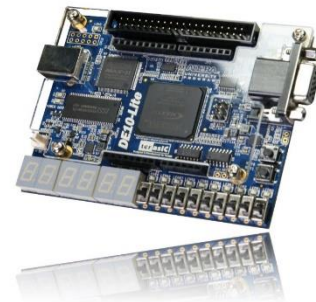
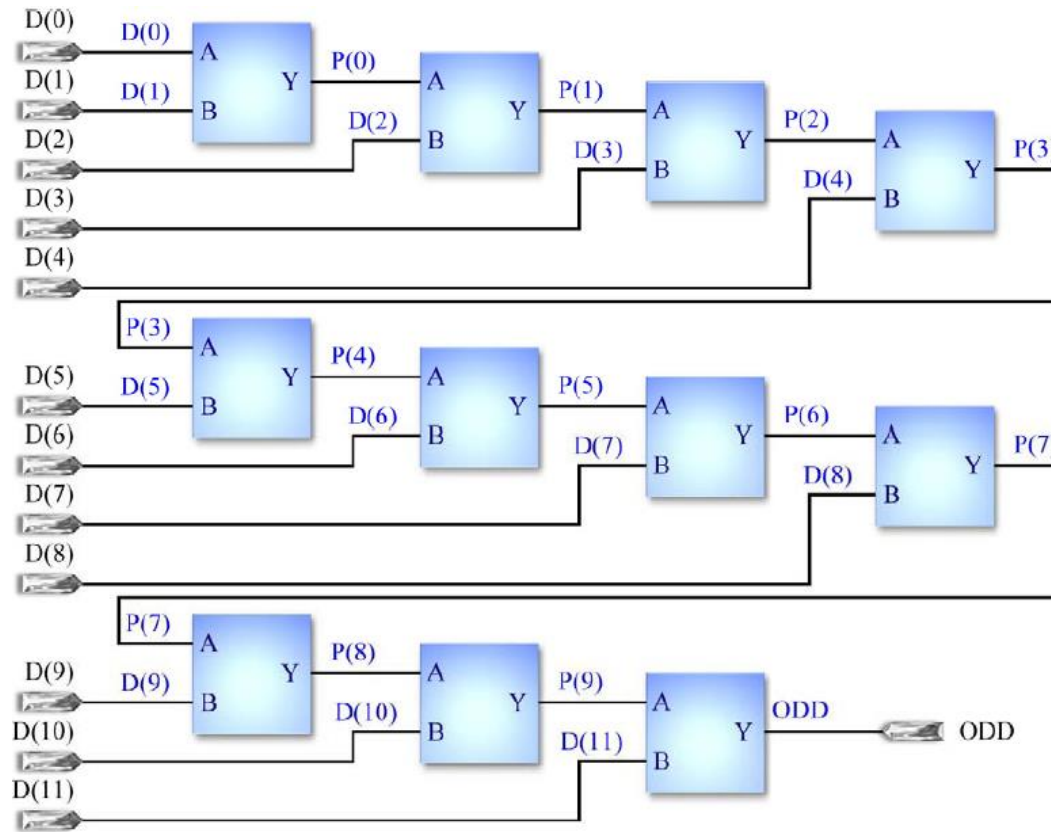
:

End Generate;



自動生成電路(5/7)

- If-Generate敘述(2/4)
 - 12bit 偶同位產生電路(1/3)



自動生成電路(6/7)

- If-Generate敘述(3/4)
 - 12bit 偶同位產生電路(2/3)

```
Library IEEE;  
Use IEEE.std_logic_1164.all;  
  
Entity Parity12 is  
    Port( D:in std_logic_vector(0 to 11);  
          ODD:out std_logic );  
    Constant width: integer :=12;  
End Parity12;
```

```
Architecture Parity_ARCH of Parity12 is  
    Component XOR2  
        Port( A,B:in std_logic; Y:out std_logic);  
    End Component;  
    Signal P: std_logic_vector(0 to width-2);  
Begin  
    GEN:For I in 0 to (width-2) Generate  
        G0: If I = 0 Generate  
            U:XOR2 Port Map (D(0), D(1), P(0)) ;  
        End generate G0 ;  
        G1: If I > 0 and I < (width -2) Generate  
            U:XOR2 Port Map (P(I-1), D(I+1), P(I)) ;  
        End Generate G1 ;  
        G2: If I = (width -2) Generate  
            U:XOR2 Port Map (P(I-1), D(I+1), ODD) ;  
        End Generate G2 ;  
    End Generate GEN;  
End Parity_ARCH;
```

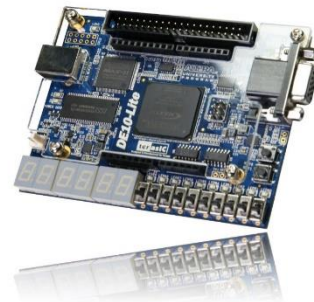


自動生成電路(7/7)

- If-Generate敘述(4/4)
 - 12bit 偶同位產生電路(3/3)
 - 編譯後，將會產生下列動作:

```
U0: XOR2 Port Map(D(0), D(1), P(0)); -- l=0
U1: XOR2 Port Map(P(0), D(2), P(1)); -- l=1
U2: XOR2 Port Map(P(1), D(3), P(2)); -- l=2
U3: XOR2 Port Map(P(2), D(4), P(3)); -- l=3
U4: XOR2 Port Map(P(3), D(5), P(4)); -- l=4
U5: XOR2 Port Map(P(4), D(6), P(5)); -- l=5
U6: XOR2 Port Map(P(5), D(7), P(6)); -- l=6
U7: XOR2 Port Map(P(6), D(8), P(7)); -- l=7
U8: XOR2 Port Map(P(7), D(9), P(8)); -- l=8
U9: XOR2 Port Map(P(8), D(10), P(9)); -- l=9
U10: XOR2 Port Map(P(9), D(11), ODD); -- l=10
```

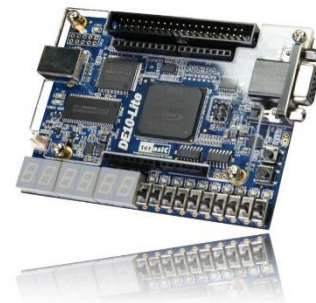
只需修改width，
即可改變偶同
位的位元數



補充:

- 套件檔

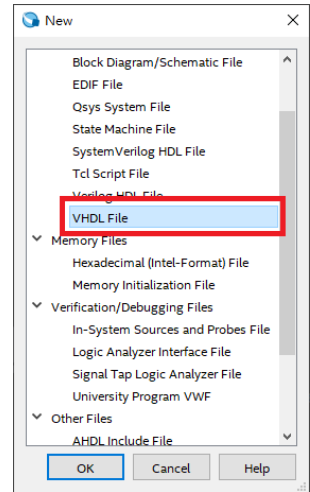
File	Package	Library	Contents
maxplus2.vhd	maxplus2	altera	Altera primitives supported by VHDL.
megacore.vhd	megacore	altera	Pretested megafunctions consisting of several different design files.
altera_mf_components.vhd	altera_mf_components	altera_mf	Altera megafunctions supported by VHDL.
std1164.vhd	std_logic_1164	ieee	Standard for describing interconnection data types for VHDL modeling, and the STD_LOGIC and STD_LOGIC_VECTOR types.
lpm_pack.vhd	lpm_components	lpm	LPM megafunctions supported by VHDL.
syn_arit.vhd	std_logic_arith	ieee	SIGNED and UNSIGNED types, arithmetic and relational functions for use with SIGNED and UNSIGNED types, and the conversion functions CONV_INTEGER, CONV_SIGNED, CONV_UNSIGNED, and CONV_STD_LOGIC_VECTOR.
syn_sign.vhd	std_logic_signed	ieee	Functions that allow the Quartus II software to use STD_LOGIC_VECTOR types as if they are SIGNED types.
syn_unsi.vhd	std_logic_unsigned	ieee	Functions that allow the Quartus II software to use STD_LOGIC_VECTOR types as if they are UNSIGNED types.
numeric_std.vhd	numeric_std	ieee	Functions defined in <i>IEEE Standard VHDL Synthesis Packages</i> (IEEE Std 1076.3-1997).
numeric_bit.vhd	numeric_bit	ieee	Functions defined in <i>IEEE Standard VHDL Synthesis Packages</i> (IEEE Std 1076.3-1997).



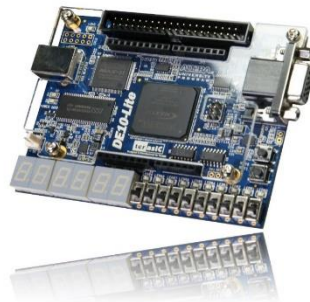
隨堂練習

- 請使用 VHDL 完成下列電路，並完成紀錄，包括 **VHDL Source Code**、**模擬波形圖**。

- a) 三線對八線解碼器(包含致能控制)
- b) 八進位對三進位編碼器(包含致能控制)
- c) 四對一多工器
- d) 一對四解多工器

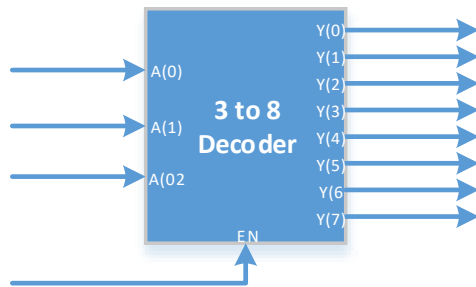


- 本次實驗完成後需助教確認正確，全部完成後，將專案與報告壓縮上傳EE-Class。
- Lecture7_組別XX.ZIP



隨堂練習(一)

- 三對八解碼器(包含致能控制)(1/2)



輸入				輸出							
EN	A(2)	A(1)	A(0)	Y(7)	Y(6)	Y(5)	Y(4)	Y(3)	Y(2)	Y(1)	Y(0)
0	任意	任意	任意	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

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

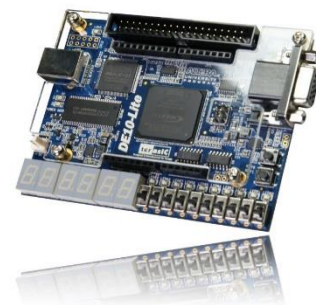
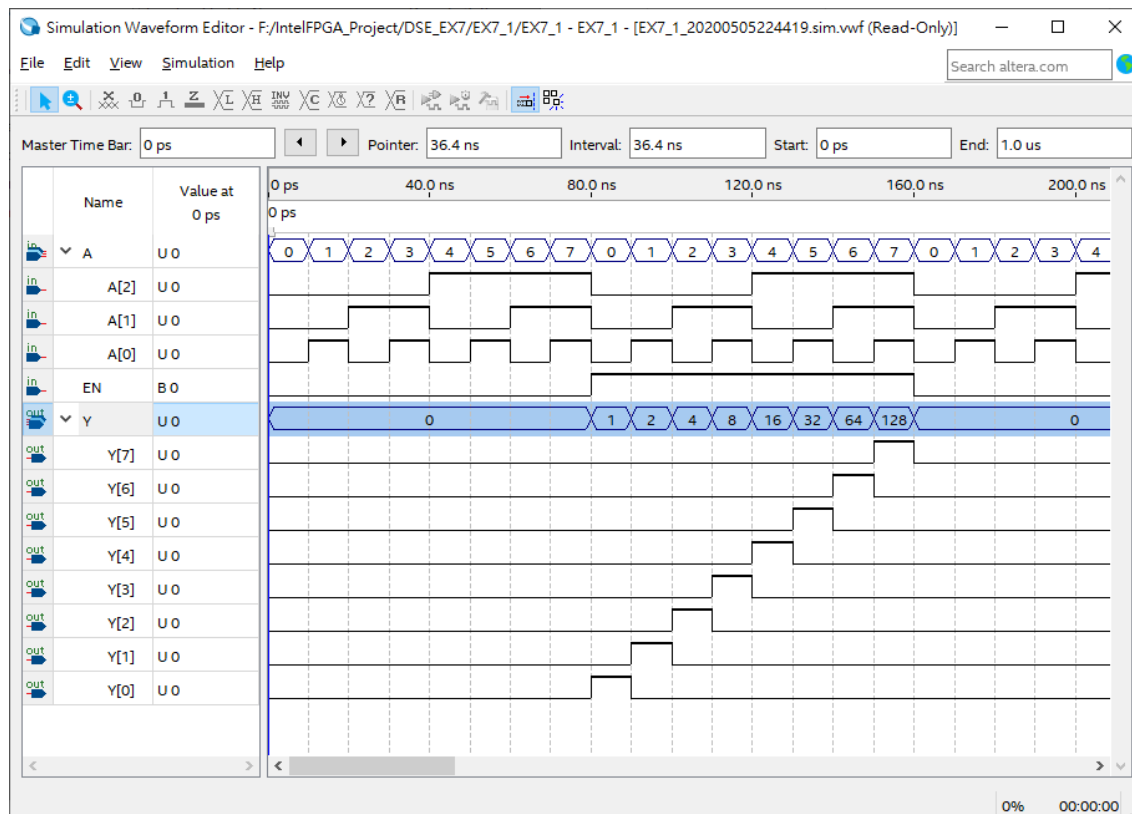
Entity Decoder3_8 is
  Port( EN:in std_logic ;
        A :in std_logic_vector(2 downto 0);
        Y:out std_logic_vector(7 downto 0));
End Decoder3_8;

Architecture ARCH of Decoder3_8 is
  signal TMP: std_logic_vector(3 downto 0);
Begin
  TMP <= EN & A; -- 將EN 與A 連接為4 位元資料
  with TMP select
    Y <= "10000000" when "1111",
          "01000000" when "1110",
          "00100000" when "1101",
          "00010000" when "1100",
          "00001000" when "1011",
          "00000100" when "1010",
          "00000010" when "1001",
          "00000001" when "1000",
          "00000000" when others;
End ARCH;
```



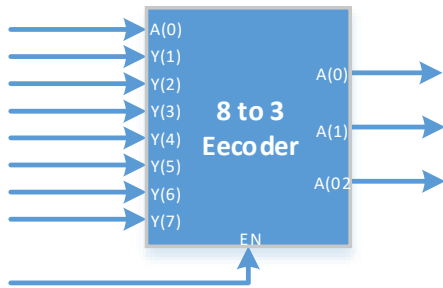
隨堂練習(一)

- 三對八解碼器(包含致能控制)(2/2)



隨堂練習(二)

• 八對三編碼器(1/2)



輸 入									輸 出		
EN	A(7)	A(6)	A(5)	A(4)	A(3)	A(2)	A(1)	A(0)	Y(2)	Y(1)	Y(0)
0	任意	任意	任意	任意	任意	任意	任意	任意	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	0	0	1	0
1	0	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	1	0	1
1	0	1	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	0	0	1	1	1

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

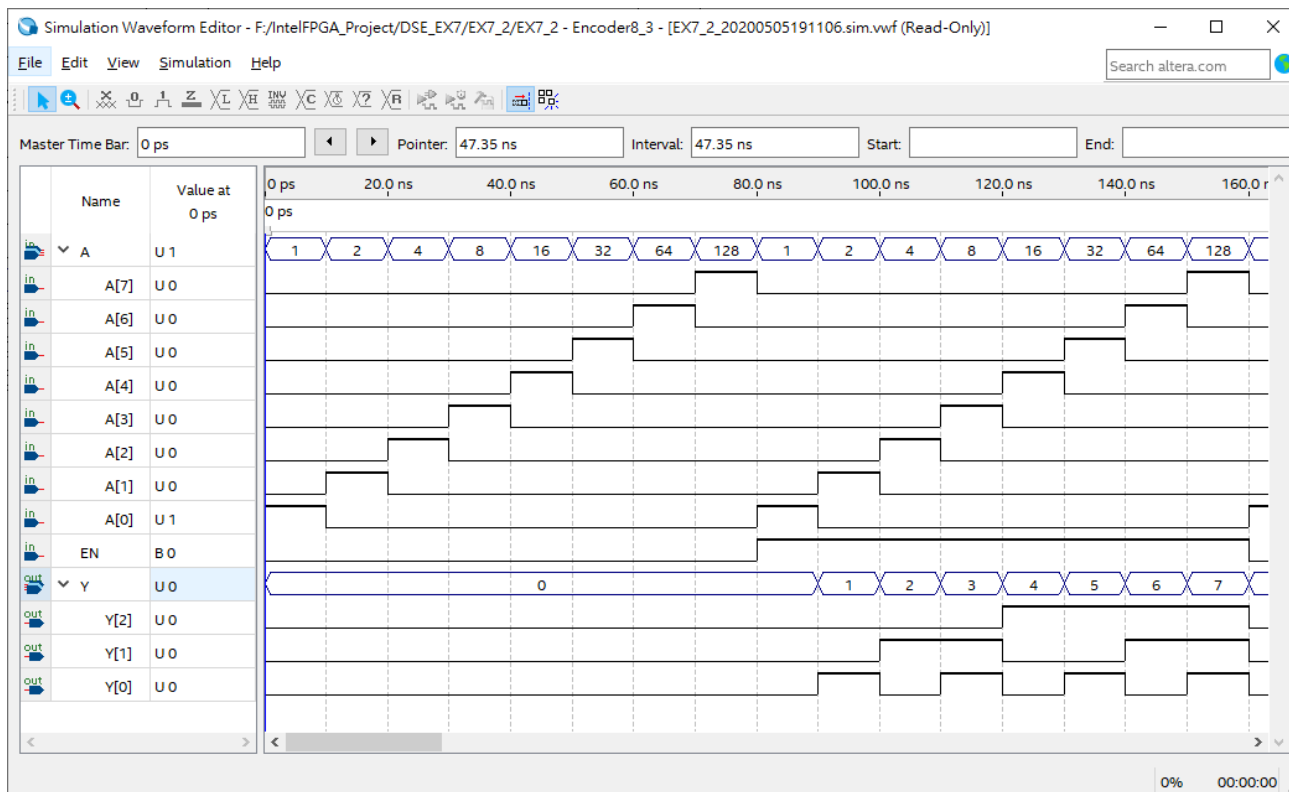
Entity Encoder8_3 is
    Port( EN:in std_logic ;
          A :in std_logic_vector(7 downto 0);
          Y:out std_logic_vector(2 downto 0) );
End Encoder8_3;

Architecture ARCH of Encoder8_3 is
Begin
    Y <= "111" when (EN and A(7)) = '1' Else
         "110" when (EN and A(6)) = '1' Else
         "101" when (EN and A(5)) = '1' Else
         "100" when (EN and A(4)) = '1' Else
         "011" when (EN and A(3)) = '1' Else
         "010" when (EN and A(2)) = '1' Else
         "001" when (EN and A(1)) = '1' Else
         "000";
End ARCH;
```



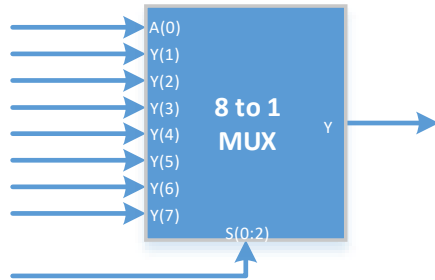
隨堂練習(二)

- 八對三編碼器(1/2)



隨堂練習(三)

- 八對一多工器(1/2)

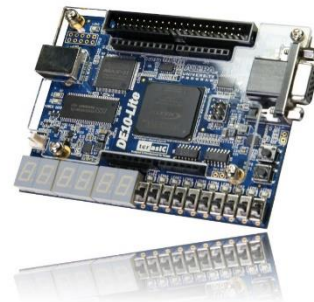


選擇信號			輸出
S(2)	S(1)	S(0)	Y
0	0	0	A(0)
0	0	1	A(1)
0	1	0	A(2)
0	1	1	A(3)
1	0	0	A(4)
1	0	1	A(5)
1	1	0	A(6)
1	1	1	A(7)

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

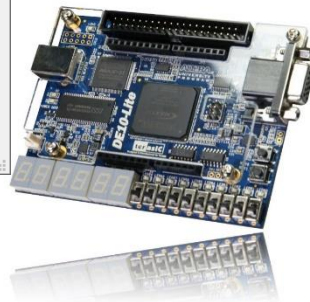
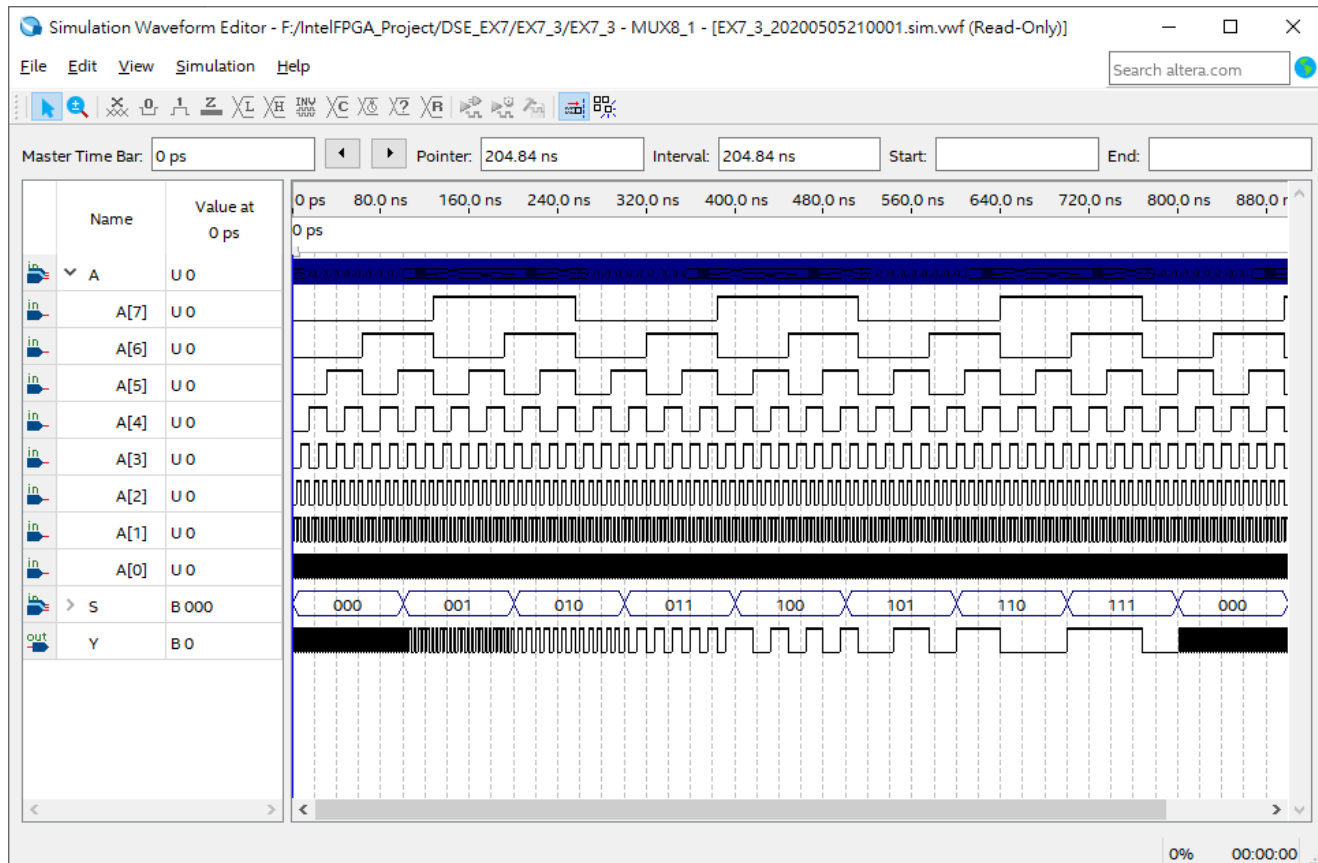
Entity MUX8_1 is
    Port(A:in std_logic_vector(7 downto 0) ;
          S:in std_logic_vector(2 downto 0);
          Y:out std_logic );
End MUX8_1;

Architecture ARCH of MUX8_1 is
Begin
    with S select
        Y <= A(0) when "000",
              A(1) when "001",
              A(2) when "010",
              A(3) when "011",
              A(4) when "100",
              A(5) when "101",
              A(6) when "110",
              A(7) when others;
End ARCH;
```



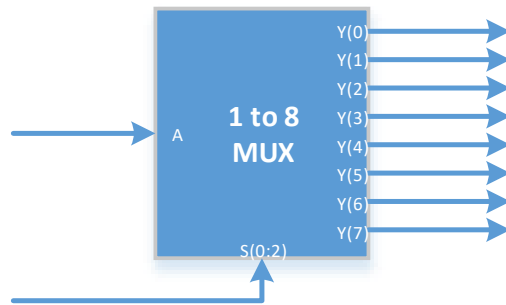
隨堂練習(三)

- 八對一多工器(2/2)



隨堂練習(四)

- 一對八解多工器(1/2)

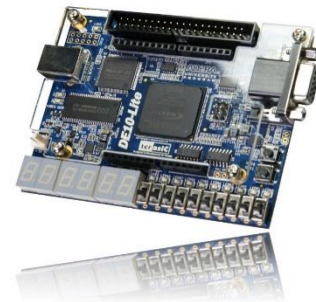


選擇信號			輸 出							
S(2)	S(1)	S(0)	Y(7)	Y(6)	Y(5)	Y(4)	Y(3)	Y(2)	Y(1)	Y(0)
0	0	0	Z	Z	Z	Z	Z	Z	Z	A(0)
0	0	1	Z	Z	Z	Z	Z	Z	A(1)	Z
0	1	0	Z	Z	Z	Z	Z	A(2)	Z	Z
0	1	1	Z	Z	Z	Z	A(3)	Z	Z	Z
1	0	0	Z	Z	Z	A(4)	Z	Z	Z	Z
1	0	1	Z	Z	A(5)	Z	Z	Z	Z	Z
1	1	0	Z	A(6)	Z	Z	Z	Z	Z	Z
1	1	1	A(7)	Z	Z	Z	Z	Z	Z	Z

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

Entity DeMUX1_8 is
Port( A:in std_logic;
      S:in std_logic_vector(2 downto 0);
      Y:out std_logic_vector(7 downto 0) );
End DeMUX1_8;

Architecture ARCH of DeMUX1_8 is
Begin
  Y(0) <= A when S="000" Else 'Z';
  Y(1) <= A when S="001" Else 'Z';
  Y(2) <= A when S="010" Else 'Z';
  Y(3) <= A when S="011" Else 'Z';
  Y(4) <= A when S="100" Else 'Z';
  Y(5) <= A when S="101" Else 'Z';
  Y(6) <= A when S="110" Else 'Z';
  Y(7) <= A when S="111" Else 'Z';
End ARCH;
```



隨堂練習(四)

- 一對八解多工器(2/2)

