A：完整設計程式的步驟

1. 開新專案(精靈)
   1.1 選擇儲存位置及專案名稱
   1.2 選擇使用晶片
2. 開新檔案(選擇圖形、波形或文字模式)。
3. 儲存新檔案。(建立此檔案及檔案資料夾，以方便管理檔案)。
4. 繪制電路圖或程式。
5. 儲存檔案與編譯(compiler)。
6. 模擬(simulation)電路輸出。(需建立新的波形檔 ＊.scf)
7. 確定 CPLD 晶片型號與設定腳位。
8. 重新編譯(compiler)。
9. 硬體驗證(下載程式至 CPLD 晶片)。

B：實驗板的 I/O 練習

C：零件庫的建立

D：匯流排的使用

第三章 VHDL 設計簡介

目前硬體描述語言主流可分為二類

◆ VHDL (Very High Speed Integrated Circuit Hardware Description Language)

◆ Verilog HDL

兩者皆是國際電機電子工程師協會(IEEE，International Electrical & Electronic Engineering)的標準語言。

VHDL 就是非常高速積體電路的硬體描述語言。這是一項原由美國國防部所支持的研究計畫。為了將電子電路的設計意涵以文件方式保存下來，以便其它人能輕易地了解電路的設計意義。這就是 VHDL 的由來。在 1985 年，美國國防部取得委託研究的第一版語言。隨後，VHDL 語言就轉移給 IEEE，並在 1987 年成為 IEEE1076---1987 標準。1988 年，英國國防部規定所有官方的 ASIC 設計均需以 VHDL 為設計描述語言。所以 VHDL 就逐漸地成為工業界的標準。1993 年，IEEE 將 IEEE1076---1987 標準經過一些增修(新增一些功能、去除模糊部份以及保留往前共容等等)之後，規範了另一個新的 VHDL 標準------IEEE1164。1996 年，IEEE 將電路合成的程式標準與規格加入至 VHDL 電路設計語言中，稱之為 IEEE1076．3 標準。而 Verilog HDL 於 1995 年才成為 IEEE 標準。

◆ AHDL：ALTERA 公司自訂的 HDL 語言，較少人用。

副檔名：AHDL → *.tdf
VHDL → *.vhd
Verilog HDL → *.v

3.1 VHDL 的語法規則

1.識別字(indentifier)只容許使用 26 個英文字母(大小寫相同)、阿拉伯數字及底線” _” ，字數不得超過 32 個字母，底線不能 2 個”_ _”相連，且不能放於最後。
2.識別字的第一個字必須為英文字母。
3.識別字不得使用保留字(reserved word)(如 and all bus else if end entity 等)。
4.註解說明為連續 2 個連接線”—“，可在指令敘述後，也可單獨成為一行。

3.2 VHDL 之基本架構

VHDL 基本架構通常可分為三大部分

library <零件庫名稱>； use <零件庫名稱>.<零件套件名稱>.all； : :	library ieee ; use ieee.std_logic_1164.all ; use ieee.std_logic_unsigned.all ; use ieee.std_logic_arith.all ;
entity <實體檔案名稱>is port(電路腳位名稱：I/O 屬性 資料型態； : :) ; end <實體檔案名稱>；	entity NAND_2 is port (A,B : in std_logic ; Y : out std_logic) ; end NAND_2 ;
architecture <架構名稱> of <實體檔案名稱> is : begin : : 對電路內部特性的描述 : end<架構名稱>；	architecture Arch of NAND_2 is begin Y <= not(A and B) ; --同時性的描述(同一時間執行多個描述) end Arch ;

常見有四類
☆ in
☆ out
☆ buffer
☆ inout

I/O 屬性：	in	例 a	:	in	std_logic;
	out	y	:	out	std_logic;
	buffer	D_ff	:	buffer	std_logic_vector (3 downto 0);
	inout	D_bus	:	inout	std_logic_vector (7 downto 0);

常用資料型態：

A：邏輯訊號

1.布林(Boolean)

只有 flase 與 true 兩種資料型態

例：

```
variable flag : boolean :=fales;
…....
if (flag = true) then
f = ’0’;
…….
```

2.位元(bit)

只有 ’0’ 與 ‘1’ 兩種

3.標準邏輯(std_logic)

主要有 ‘0’ 、 ‘1’ 、 ‘Z’ 、 ‘_’ 四種形態

‘Z’ —高阻抗

‘_’ —不理會

例：

```
a,OE：in      std_logic;
f     ：out   std_logic;
……
if    OE= ‘1’ then
      f <= a;
else
      f <= ‘Z’;
end if;
```

使用單引號 ’‘ 來設定單一位元資

4.位元向量(bit_vector)

使用雙引號 “ ” 來設定向量資料

例： A,B ： bit_vector(0 to 8);

C,D,E： bit_vector(7 downto 0);

signal C ： bit_vector(3 downto 0) ：=” 1100”

即 C(3)=’1’ C(2)=’1’ C(1)=’0’ C(0)=’0’

5.標準邏輯向量(std_logic_vector)

標準邏輯向量(std_logic_vector) 與 位元向量(bit_vector)的宣告與使用幾乎相同，只是資料中多了'Z' 與 '_'兩種。由於標準邏輯與標準邏輯向量的資料型態就是電路的真實狀態，所以在程式中幾乎取代了位元與位元向量。

B：數值訊號

主要用到的有

1. 整數(integer)

範圍：-2^{31} 到 $2^{31}-1$

-2147483648 to 2147483647

signal B：integer range 0 to 7; -- 宣告 B 為 3 位元的數值

signal C：integer range -16 to 15; -- 宣告 C 為 5 位元的數值

signal D：integer; -- 宣告 D 為 32 位元的數值

2.無號整數(unsigned)

可做數值運算，也可作邏輯運算

常用運算子：

1.指定運算子

訊號(signal)指定運算子 <=

變數(variable)指定運算子 :=

例：A <= B 當 A 宣告為訊號(signal)時

A := B 當 A 宣告為變數(variable)時

2.關係運算子---其結果是以布林資料型態來表示。"true"或"false"

= 等於

/= 不等於

> 大於

< 小於

>= 大於等於

<= 小於等於

例：if a /= b then ……

3.邏輯運算子

not	**nand**
and	**xor**
or	**xnor**
nor	

注意：not 次序最優先，其餘同時使用時，依由左至右的次序執行

4.算術運算子

+	加法
-	減法
*	乘法
/	除法
**	次方
abs	絕對值
mod	模數
rem	餘數

MAX+plus Ⅱ只支援 ” + 、 - 、 * 、 / ” 四種運算子，且資料型態為整數或位元向量

5.並置運算子 “ & “

例： ‘0’ & ‘1’ & ‘1’ 結果為 “011”
“011” & ” 01” 結果為 “01101”

6.數值運算子

rol ： 向左旋轉
“1001010” rol 2 is “0101010”
ror ： 向右旋轉
“1001010” ror 3 is “0101001”
“1001010” ror -2 is “0101010” 相當於 rol 2
“1001010” rol -2 is “1010010” 相當於 ror 2
sll ： 向左移位
“1001010” sll 2 is “0101000” 移位後空位補’0’
srl ： 向右移位
“1001010” srl 3 is “0001001” 移位後空位補’0’
sla ： 有號數向左移位

"1001010" sla 2 is "0101000" 向左移位後空位補最右位元值'0'

sra : 有號數向右移位

"1001010" sra 3 is "1111001" 向右移位後空位補最左位元值'1'

範例 1：

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
use ieee.std_logic_arith.all ;
--******************************
entity NAND_2 is
  port (     A,B : in std_logic ;
             Y     : out std_logic ) ;
end NAND_2 ;
--******************************
architecture Arch of NAND_2 is
begin
      Y <= not(A and B) ;       --同時性的描述(同一時間執行多個描述)
end Arch    ;
```

--資料流描述

```
--****************************************
architecture Arch of NAND_2 is
begin
      Y <=    A    nand    B ;
end Arch    ;
```

--行為式描述

```
--****************************************
architecture Arch of NAND_2 is
```

```
begin
        process(A,B)
        begin
                if (A='1') and (B='1') then    --循序(在 process 中之描述都是循序執行)
                        Y<='0';
                else
                        Y<='1';
                end if;
        end process;
end    Arch ;
--****************************************
```

範例 2：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
--************************
entity add_4 is
port(
        a,b   :       in     std_logic_vector(3 downto 0);
        s     :       out    std_logic_vector(3 downto 0);
        c     :       out    std_logic);
end add_4;
--*********************************************
architecture adder_4 of add_4 is
signal temp     :       std_logic_vector(4 downto 0);
begin
        temp<=   a+b;
        s      <=   temp(3 downto 0);
        c <= temp(4);
end adder_4;
```