

數位邏輯實習

VHDL硬體描述語言(II)

國立台北科技大學
電機工程系
吳昭正

Outline

- 資料型態
- 條件式
- 陣列
- 模組化
- Digital實作
- 資料型態轉換
- 資料運算
- 作業題

Pre-defined VHDL Types

bit	'0' or '1'
boolean	FALSE or TRUE
integer	an integer in the range $-(2^{31} - 1)$ to $+(2^{31} - 1)$ (some implementations support a wider range)
positive	an integer in the range 1 to $2^{31} - 1$ (positive integers)
natural	an integer in the range 0 to $2^{31} - 1$ (positive integers and zero)
real	floating-point number in the range $-1.0\text{E}38$ to $+1.0\text{E}38$
character	any legal VHDL character including upper- and lower case letters, digits, and special characters; each printable character must be enclosed in single quotes, e.g., 'd', '7', '+'
time	an integer with units fs, ps, ns, us, ms, sec, min, or hr

資料型態(2/3)

IEEE Standard Logic

Use of two-valued logic (bits and bit vectors) is generally not adequate for simulation of digital systems. In addition to '0' and '1', values of 'Z' (high-impedance or no connection), 'X' (unknown), and 'U' (uninitialized) are frequently used in digital system simulation. The IEEE standard 1164 defines a **std_logic** type that actually has nine values:

'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'

(We will only use the values 'U', 'X', '0', '1', and 'Z'.)

Section 10.8 (p. 304)

資料型態(3/3)

VHDL Operators

Predefined VHDL operators can be grouped into seven classes:

1. binary logical operators: **and or nand nor xor xnor**
2. relational operators: **= /= < <= > >=**
3. shift operators: **sll srl sla sra rol ror**
4. adding operators: **+ - & (concatenation)**
5. unary sign operators: **+ -**
6. multiplying operators: *** / mod rem**
7. miscellaneous operators: **not abs ****

When parentheses are not used, operators in class 7 have the highest precedence and are applied first, followed by class 6, then class 5, etc.

Section 10.6 (p. 301)

The ‘&’ operator can be used to concatenate two vectors (or an element and a vector, or two elements) to form a longer vector. For example, “010” & ‘1’ is “0101”.

實作範例一(1/4)

Full Adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
entity FullAdder is
```

```
port (
```

```
    X, Y, Cin: in std_logic;
```

```
    Sum, Cout: out std_logic);
```

```
end FullAdder;
```



→ 資料型態

```
architecture Equations of FullAdder is
begin
```

```
    -- concurrent assignment statements
```

```
    Sum <= X xor Y xor Cin;
```

```
    Cout <= (X and Y) or (X and Cin) or (Y and Cin);
```

```
end Equations;
```

→ 布林代數式撰寫法

Conditional Signal Assignments

The general form of a conditional signal assignment statement is

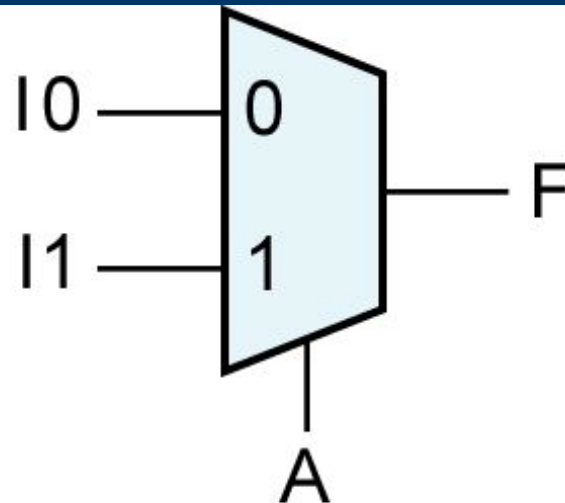
```
signal_name <= expression1 when condition1  
    else expression2 when condition2  
    [else expressionN];
```

This concurrent statement is executed whenever a change occurs in one of the expressions or conditions. If condition1 is true, signal_name is set equal to the value of expression2, etc.

The line in square brackets is optional.

條件式(2/6)

2-to-1 Mux



```
-- conditional signal assignment statement
F <= I0 when A = '0' else I1;
```

Figure 10-5: 2-to-1 Multiplexer

This statement executes whenever A, I0, or I1 changes.

Conditional Signal Assignments

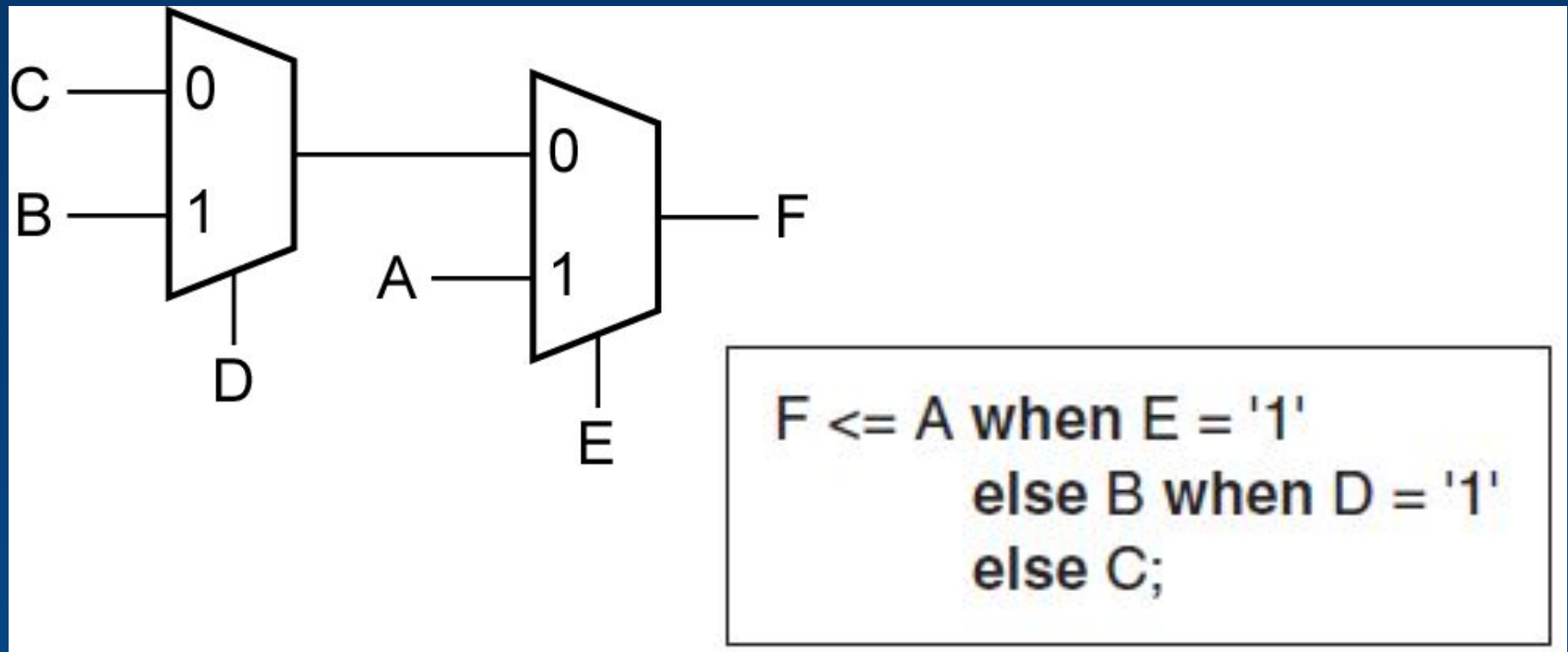


Figure 10-6: Cascaded 2-to-1 MUXes

Conditional Signal Assignments

The logic equation for a 4-to-1 MUX is $F = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$

Thus, one way to model the MUX is with the VHDL statement

```
F <= (not A and not B and I0) or (not A  
and B and I1) or (A and not B and I2) or  
(A and B and I3);
```

Another way to model the 4-to-1 MUX is to use a conditional assignment statement:

```
F <= I0 when A&B = "00"  
else I1 when A&B = "01"  
else I2 when A&B = "10"  
else I3;
```

單引號用於單一數值, 如:'1'或'Z'
雙引號用於多重數值, 如:"00"或"100"

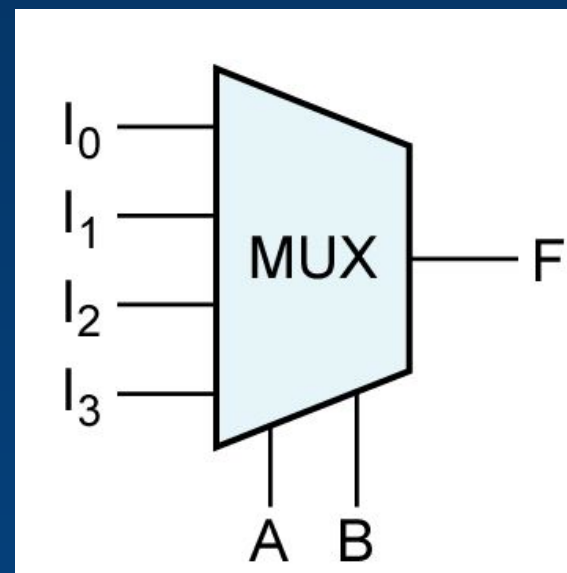


Figure 10-7: 4-to-1 Multiplexer

Conditional Signal Assignments

The general form of a selected signal assignment statement is

with expression_s **select**

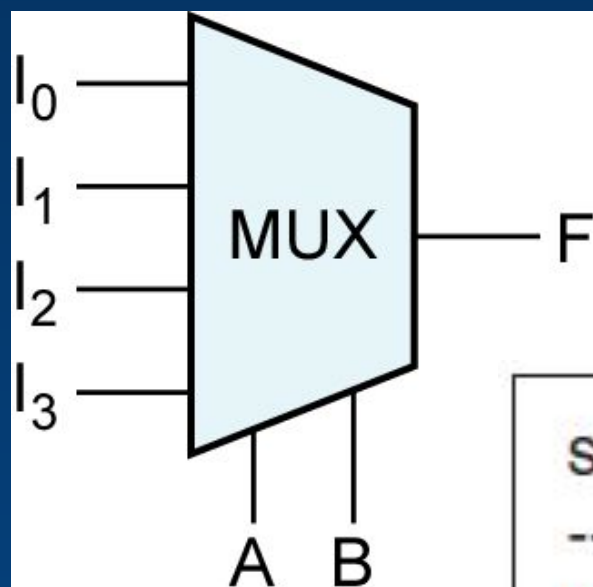
signal_s <= expression1 [**after** delay-time] **when** choice1,

expression2 [**after** delay-time] **when** choice2,

...

[expression_n [**after** delay-time] **when** others];

Conditional Signal Assignments

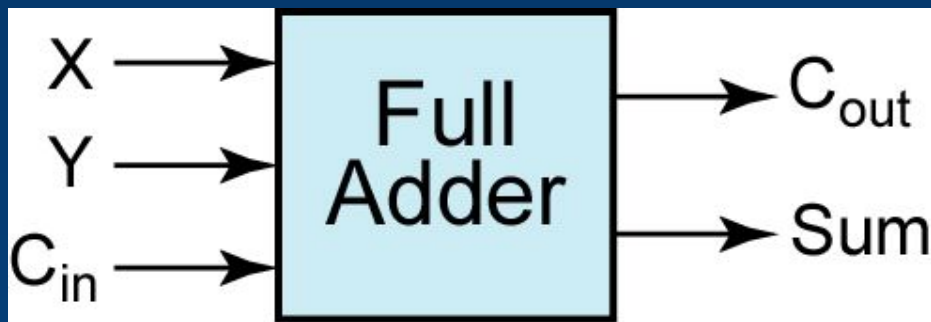


```
sel <= A&B;
-- selected signal assignment statement
with sel select
    F <= I0 when "00",
        I1 when "01",
        I2 when "10",
        I3 when "11";
```

Figure 10-7: 4-to-1 Multiplexer

實作範例一(2/4)

Full Adder



X	Y	C _{in}	C _{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 4-4: Truth Table for a Full Adder

實作範例一(3/4)

Full Adder (when寫法)

architecture Equations of FullAdder is
signal input: std_logic_vector (0 to 2);
begin

input <= X & Y & Cin;

Sum <= '0' when input = "000"

else

'1' when input = "001" else

'1' when input = "010" else

'0' when input = "011" else

'1' when input = "100" else

'0' when input = "101" else

'0' when input = "110" else

'1' when input = "111";

Cout <= '0' when input = "000"
else

'0' when input = "001" else

'0' when input = "010" else

'1' when input = "011" else

'0' when input = "100" else

'1' when input = "101" else

'1' when input = "110" else


'1' when input = "111";

end Equations;

Full Adder (with select寫法)

architecture Equations of FullAdder is
signal input: std_logic_vector (0 to 2);
begin

```
input <= X & Y & Cin;
with input select
Sum <= '0' when "000",
    '1' when "001",
    '1' when "010",
    '0' when "011",
    '1' when "100",
    '0' when "101",
    '0' when "110",
    '1' when others;
```



```
with input select
    Cout <= '0' when "000",
        '0' when "001",
        '0' when "010",
        '1' when "011",
        '0' when "100",
        '1' when "101",
        '1' when "110",
        '1' when others;
```

```
end Equations;
```

4-Bit Adder Modules

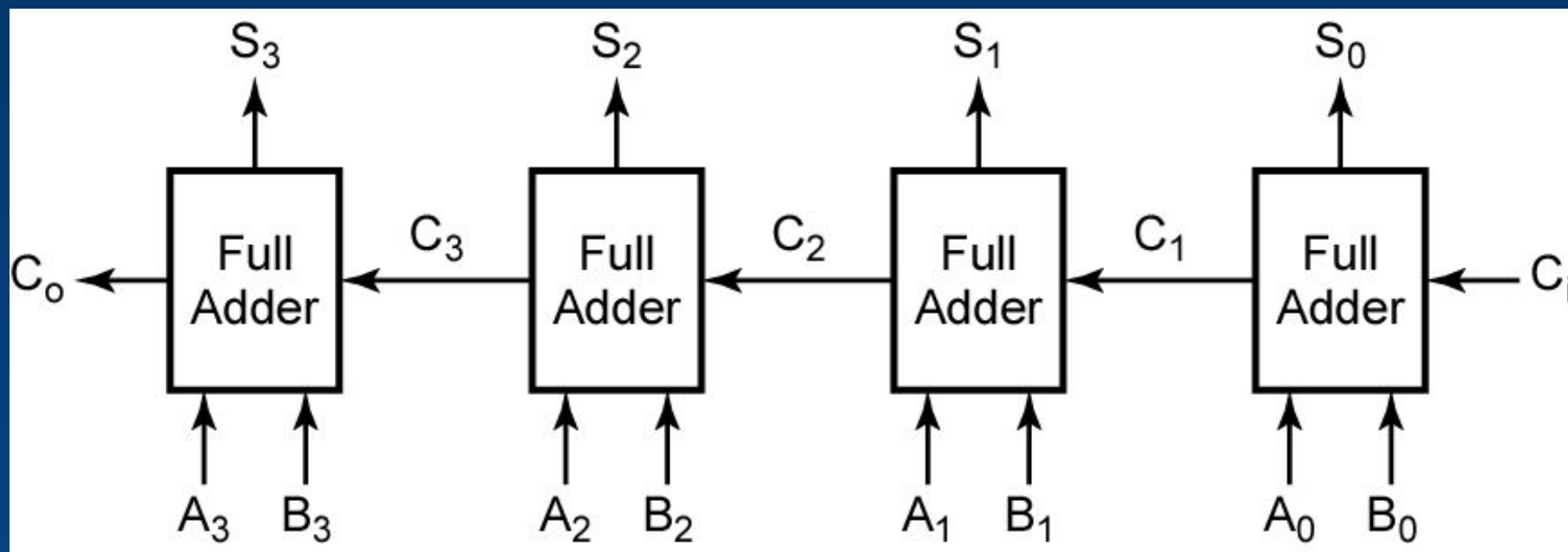


Figure 10-11: 4-Bit Binary Adder

陣列(1/2)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

entity FullAdder is

```
port (
    A0, A1, A2, A3, B0, B1, B2, B3, Ci: in std_logic;
    S0, S1, S2, S3, Co: out std_logic);
```

end FullAdder;

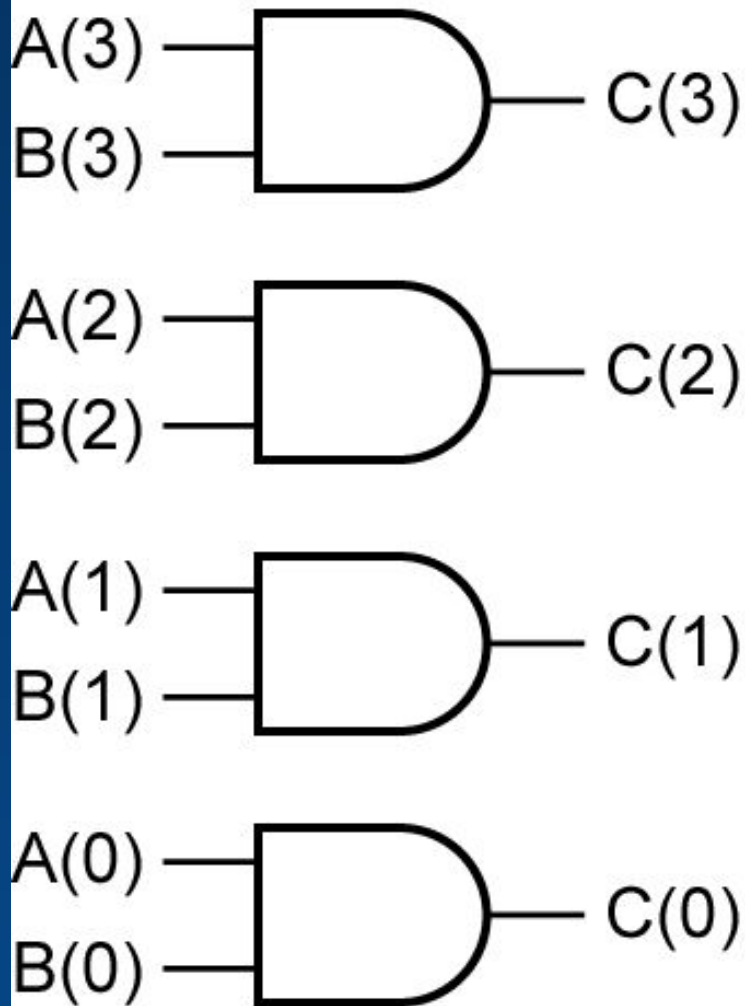


```
port (
    A, B: in std_logic_vector(0 to 3);
    Ci: in std_logic;
    S: out std_logic_vector(0 to 3);
    Co: out std_logic);
```



```
port (
    A, B: in std_logic_vector(3 downto 0);
    Ci: in std_logic;
    S: out std_logic_vector(3 downto 0);
    Co: out std_logic);
```

陣列(2/2)



-- the hard way

```
C(3) <= A(3) and B(3);
C(2) <= A(2) and B(2);
C(1) <= A(1) and B(1);
C(0) <= A(0) and B(0);
```

-- the easy way

```
C <= A and B;
```

Figure 10-4: Array of AND Gates

Components used within the architecture are declared at the beginning of the architecture using a component declaration of the form:

```
component component-name  
  port (list-of-interface-signals-and-their-types);  
end component;
```

The connections to each component used in a circuit are specified by using a component instantiation statement of the form:

```
label: component-name port map (list-of-actual-signals);
```

The list of actual signals must correspond one-to-one to the list of interface signals specified in the component declaration.

4-Bit Adder Modules

```
entity Adder4 is
    port (A, B: in bit_vector(3 downto 0); Ci: in bit; -- Inputs
          S: out bit_vector(3 downto 0); Co: out bit); -- Outputs
end Adder4;

architecture Structure of Adder4 is
    component FullAdder
        port (X, Y, Cin: in bit; -- Inputs
              Cout, Sum: out bit); -- Outputs
    end component;

    signal C: bit_vector(3 downto 1);
begin
    -- instantiate four copies of the FullAdder
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

Figure 10-12:
Structural
Description of a
4-Bit Adder

模組化(3/8)

Full Adder

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;  
  
entity FullAdder4 is  
port (  
    A, B: in std_logic_vector(3 downto 0);  
    Ci: in std_logic;  
    S: out std_logic_vector(3 downto 0);  
    Co: out std_logic);  
end FullAdder4;
```

模組化(4/8)

Full Adder

architecture Structure of FullAdder4 is

component FullAdder

```
port (X, Y, Cin: in std_logic;  
      Cout, Sum: out std_logic);
```

end component;

```
signal C: std_logic_vector(3 downto 1);
```

begin

```
FA0: FullAdder port map(A(0), B(0), Ci, C(1), S(0));
```

```
FA1: FullAdder port map(A(1), B(1), C(1), C(2), S(1));
```

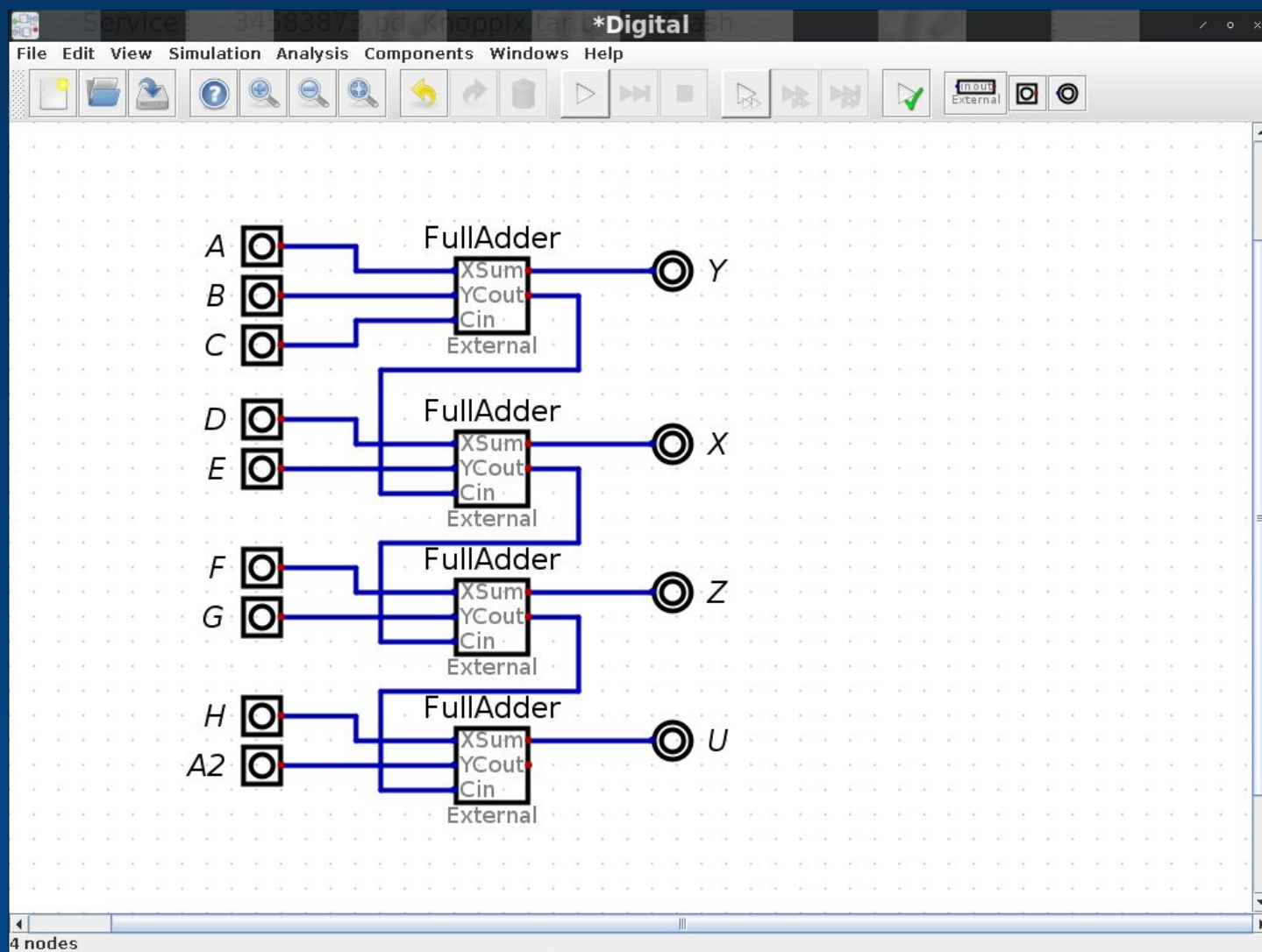
```
FA2: FullAdder port map(A(2), B(2), C(2), C(3), S(2));
```

```
FA3: FullAdder port map(A(3), B(3), C(3), Co, S(3));
```

end Structure;

模組化(5/8)

Digital產生的VHDL



模組化(6/8)

Digital產生的VHDL

```
-- generated by Digital. Don't modify this file!
-- Any changes will be lost if this file is regenerated.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity FullAdder is
  port (
    X, Y, Cin: in std_logic;
    Sum, Cout: out std_logic);
end FullAdder;

architecture Equations of FullAdder is
begin
  -- concurrent assignment statements
  Sum <= X xor Y xor Cin;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;
```


模組化(7/8)

Digital產生的VHDL

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
entity main is
  port (
    A: in std_logic;
    B: in std_logic;
    C: in std_logic;
    D: in std_logic;
    E: in std_logic;
    F: in std_logic;
    G: in std_logic;
    H: in std_logic;
    A2: in std_logic;
    Y: out std_logic;
    X: out std_logic;
    Z: out std_logic;
    U: out std_logic);
end main;

```

architecture Behavioral of main is

```

  signal s0: std_logic;
  signal s1: std_logic;
  signal s2: std_logic;
begin
  gate0: entity work.FullAdder -- FullAdder
    port map (
      X => A,
      Y => B,
      Cin => C,
      Sum => Y,
      Cout => s0);
  gate1: entity work.FullAdder -- FullAdder
    port map (
      X => D,
      Y => E,
      Cin => s0,
      Sum => X,
      Cout => s1);

```

不用宣告Component

模組化(8/8)

Digital產生的VHDL

```
gate2: entity work.FullAdder -- FullAdder
  port map (
    X => F,
    Y => G,
    Cin => s1,
    Sum => Z,
    Cout => s2);
gate3: entity work.FullAdder -- FullAdder
  port map (
    X => H,
    Y => A2,
    Cin => s2,
    Sum => U);
end Behavioral;
```

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity FullAdder is
  port (
    X, Y, Cin: in std_logic;
    Sum, Cout: out std_logic);
end FullAdder;

architecture Equations of FullAdder is
begin
  -- concurrent assignment statements
  Sum <= X xor Y xor Cin;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;
  
```

FullAdder.vhd

第一種做法是每個電路分不同VHDL檔存放
，Quartus與GHDL有支援，**但是Digital+GHDL不支援。**

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity FullAdder4 is
  port (
    A, B: in std_logic_vector(3 downto 0);
    Ci: in std_logic;
    S: out std_logic_vector(3 downto 0);
    Co: out std_logic);
end FullAdder4;

architecture Structure of FullAdder4 is
  component FullAdder
    port (X, Y, Cin: in std_logic;
          Cout, Sum: out std_logic);
  end component;

  signal C: std_logic_vector(3 downto 1);
  begin
    FA0: FullAdder port map(A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map(A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map(A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map(A(3), B(3), C(3), Co, S(3));
  end Structure;
  
```

FullAdder4.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
entity FullAdder is
  port (
    X, Y, Cin: in std_logic;
    Sum, Cout: out std_logic);
end FullAdder;
```

```
architecture Equations of FullAdder is
begin
  -- concurrent assignment statements
  Sum <= X xor Y xor Cin;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin);
end Equations;
```



FullAdder4.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
entity FullAdder4 is
  port (
    A, B: in std_logic_vector(3 downto 0);
    Ci: in std_logic;
    S: out std_logic_vector(3 downto 0);
    Co: out std_logic);
end FullAdder4;
```

```
architecture Structure of FullAdder4 is
  component FullAdder
    port (X, Y, Cin: in std_logic;
          Cout, Sum: out std_logic);
  end component;
```

```
  signal C: std_logic_vector(3 downto 1);
  begin
    FA0: FullAdder port map(A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map(A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map(A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map(A(3), B(3), C(3), Co, S(3));
  end Structure;
```

第二種做法是所有電路都存放在一個VHDL檔，Quartus、GHDL與Digital+GHDL均有支援，但是在Digital+GHDL裡面的前後順序很重要。

Digital實作(3/4)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

entity FullAdder is

檔案前半部分

port (

X, Y, Cin: in std_logic;

Sum, Cout: out std_logic);

end FullAdder;

architecture Equations of FullAdder is
begin

-- concurrent assignment statements

Sum <= X xor Y xor Cin;

Cout <= (X and Y) or (X and Cin) or (Y and Cin);

end Equations;



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

entity FullAdder4 is

檔案後半部分

port (

A, B: in std_logic_vector(3 downto 0);

Ci: in std_logic;

S: out std_logic_vector(3 downto 0);

Co: out std_logic);

end FullAdder4;

architecture Structure of FullAdder4 is

component FullAdder

port (X, Y, Cin: in std_logic;

Cout, Sum: out std_logic);

end component;

signal C: std_logic_vector(3 downto 1);

begin

FA0: FullAdder **port map**(A(0), B(0), Ci, C(1), S(0));

FA1: FullAdder **port map**(A(1), B(1), C(1), C(2), S(1));

FA2: FullAdder **port map**(A(2), B(2), C(2), C(3), S(2));

FA3: FullAdder **port map**(A(3), B(3), C(3), Co, S(3));

end Structure;



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
entity FullAdder4 is
```

```
port (
```

```
  A, B: in std_logic_vector(3 downto 0);
```

```
  Ci: in std_logic;
```

```
  S: out std_logic_vector(3 downto 0);
```

```
  Co: out std_logic);
```

```
end FullAdder4;
```

```
architecture Structure of FullAdder4 is
```

```
  component FullAdder
```

```
    port (X, Y, Cin: in std_logic;
```

```
          Cout, Sum: out std_logic);
```

```
  end component;
```

```
  signal C: std_logic_vector(3 downto 1);
```

```
  begin
```

```
    FA0: FullAdder port map(A(0), B(0), Ci, C(1), S(0));
```

```
    FA1: FullAdder port map(A(1), B(1), C(1), C(2), S(1));
```

```
    FA2: FullAdder port map(A(2), B(2), C(2), C(3), S(2));
```

```
    FA3: FullAdder port map(A(3), B(3), C(3), Co, S(3));
```

```
  end Structure;
```

檔案前半部分



```
LIBRARY ieee;
```

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

```
USE ieee.numeric_std.all;
```

```
entity FullAdder is
```

```
  port (
```

```
    X, Y, Cin: in std_logic;
```

```
    Sum, Cout: out std_logic);
```

```
end FullAdder;
```

```
architecture Equations of FullAdder is
```

```
  begin
```

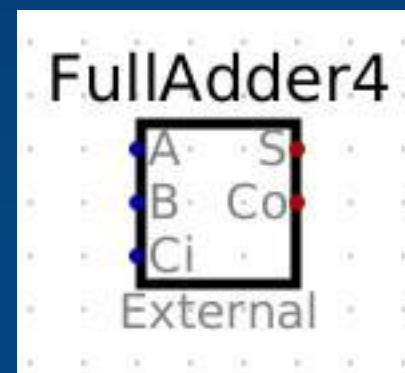
```
    -- concurrent assignment statements
```

```
    Sum <= X xor Y xor Cin;
```

```
    Cout <= (X and Y) or (X and Cin) or (Y and Cin);
```

```
  end Equations;
```

檔案後半部分



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

```
entity FullAdder4_int_v2 is
port (
    A, B: in integer range 0 to 15;
    Ci: in integer range 0 to 1;
    S: out integer range 0 to 15);
end FullAdder4_int_v2;
```

```
architecture Structure of FullAdder4_int_v2 is
begin
    S <= A + B + Ci;
end Structure;
```

VDHL可以用更高階的資料格式來簡化程式，左邊的程式是標準語法，可以在GHDL與Quartus中編譯，但是無法於Digital+GHDL中執行，因為Digital+GHDL無法判定輸入輸出的腳位有幾個bits。

IEEE.**numeric_std** is a library package defined for VHDL. It defines unsigned type and overloaded operators for arithmetic and comparison operations.

Some useful conversion functions in the package include:

TO_INTEGER(A) converts an unsigned vector A to an integer

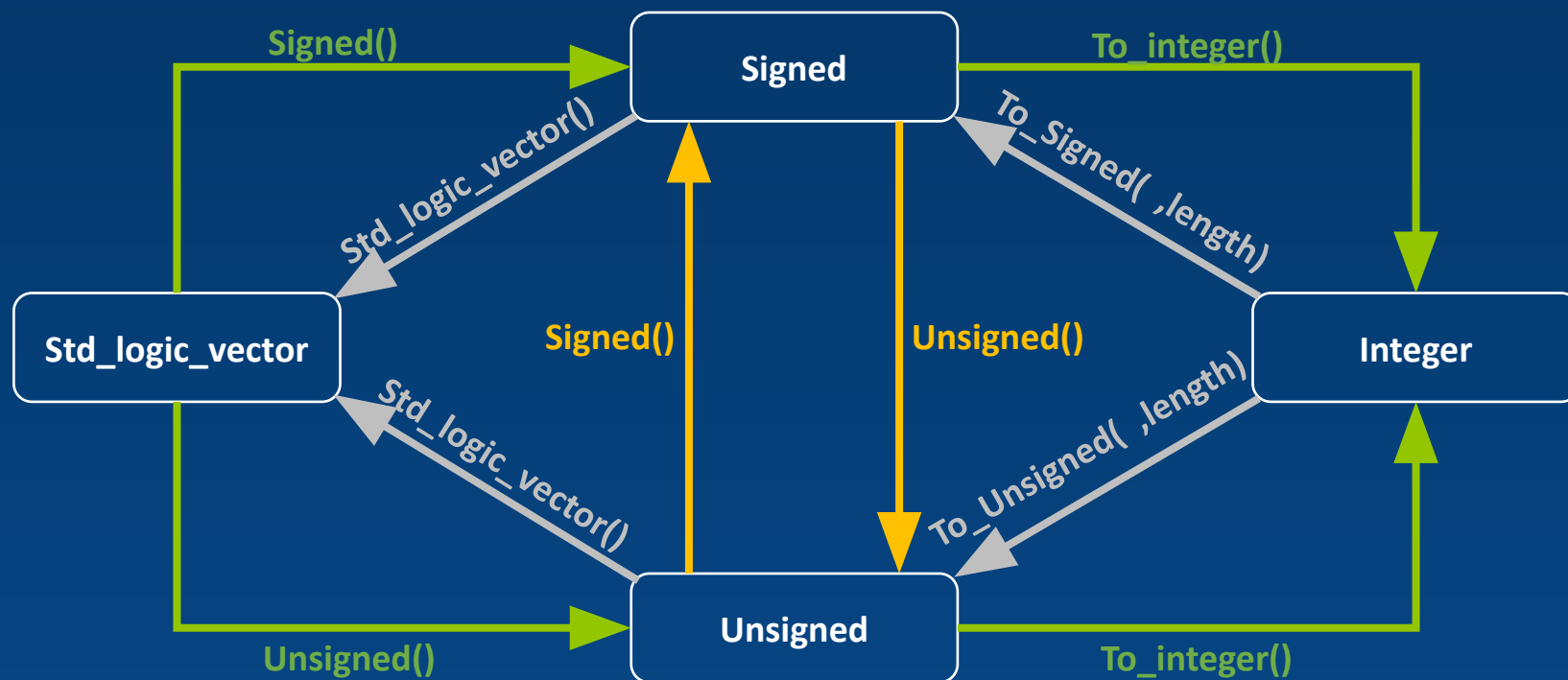
TO_UNSIGNED(B, N) converts an integer to an unsigned vector of length N

The only significant deficiency is that this package does not define an overloaded operator for adding a std_logic bit to an unsigned type. Thus, a statement of the form

$$\text{Sum} \leq \text{A} + \text{B} + \text{carry};$$

is not allowed when carry is of type std_logic.

Conversion diagram



In this part, we will use IEEE.numeric_std package to store numbers instead of the previous registers. From the conversion diagram, we can know that numbers can be converted from std_logic_vector to integer by using the functions: Signed()/Unsigned and To_integer(). For example, if A is defined as a std_logic_vector type, and we expect it to be converted to integer type. First, it is needed to determine that A is a signed or unsigned number. If A is the former, the VHDL is

```
B <= to_integer(signed(A));
```

Where B is a signal, integer type. Otherwise, it is

```
B <= to_integer(unsigned(A));
```

In VHDL, there are some predefined arithmetic operators, like adding and multiplying operators (+, -, &, *, /, rem, mod), relational operators (=, /=, <, <=, >, >=), etc. Except for ‘&’ and relational operators, they can be applied to integer or real numeric operands. The ‘&’ operator can be used to concatenate two vectors (or an element and a vector, or two elements) to form a longer vector. For example, “010” & ‘1’ is “0101”.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity FullAdder4_int_v2 is
port (
    A, B: in integer range 0 to 15;
    S: out integer range 0 to 15);
end FullAdder4_int_v2;
    
```

無法於Digital+GHDL中執行



```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity FullAdder4 is
port (
    A, B: in std_logic_vector(3 downto 0);
    S: out std_logic_vector(3 downto 0);
    Co: out std_logic);
end FullAdder4;
    
```

可以於Digital+GHDL中執行



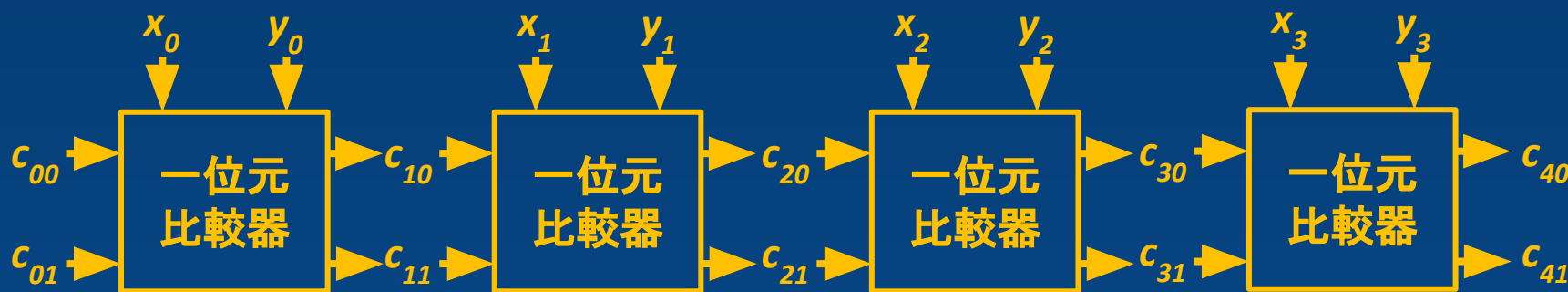
資料運算(4/4)

```
architecture Structure of FullAdder4_int_v2 is
begin
    S <= A + B;
end Structure;
```

```
architecture Structure of FullAdder4 is
signal A_int, B_int, Ci_int, S_int: integer := 0;
signal output: std_logic_vector(4 downto 0);
signal temp: std_logic_vector(1 downto 0);
begin
    A_int <= to_integer(unsigned(A));
    B_int <= to_integer(unsigned(B));
    S_int <= A_int + B_int;
    output <= std_logic_vector(to_unsigned(S_int,5));
    S <= output(3 downto 0);
    Co <= output(4);
end Structure;
```

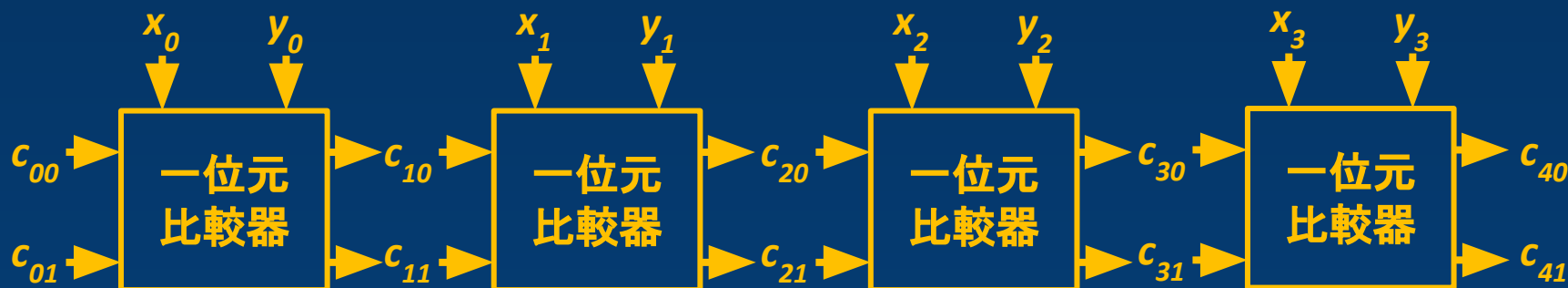
作業實作題目(1/3)

- 作業題1:利用VHDL Component重新實作4位元的比較器, 將兩個四位元的輸入 X 與 Y 進行比較, 並輸出比較結果。
- 1位元比較器請改用VHDL條件式完成。



低位比較到高位比較器

作業實作題目(2/3)



$c_{(i+1)0}$	$c_{(i+1)1}$	所代表的功能
0	0	$x_i = y_i$ 與 $c_{i0} = c_{i1} = 0$ (x 與 y 在 i 位元(包含)之前都一樣)
0	1	$x_i < y_i$ (x 在 i 位元小於 y) $x_i = y_i$ 與 $c_{i0} = 0, c_{i1} = 1$ (x 在 i 位元等於 y , 但在之前的位元中 x 小於 y)
1	0	$x_i > y_i$ (x 在 i 位元大於 y) $x_i = y_i$ 與 $c_{i0} = 1, c_{i1} = 0$ (x 在 i 位元等於 y , 但在之前的位元中 x 大於 y)

作業實作題目(3/3)

- 作業題2:利用投影片31-37頁的資料型態重新設計與擴充加減法器成為加減乘法器。
- 輸入腳位:四位元的A、四位元的B與兩位元的F
- 輸出腳位:八位元的S
- 功能如下:

F = 00	$S_{7\sim0} = A_{3\sim0} + B_{3\sim0}$
F = 01	$S_{7\sim0} = A_{3\sim0} - B_{3\sim0}$
F = 10	$S_{7\sim0} = A_{3\sim0} \times B_{3\sim0}$