Introduction to VHDL

Pierre Chen

2006-10-04
CSIE, NCU

History of VHDL

– 1980年代初：VHSIC（Very High Speed Integrated Circuit）的計劃以gate level的方式描述電路。

– 1982年：VHSIC硬體描述語言(VHSIC Hardware Description Language），簡稱VHDL。

– 1987年： VHDL成爲IEEE標準(IEEE 1076) 。

– 1988年：美國國防部規定所有官方的ASIC設計均要以VHDL爲其硬體描述語言，自此之後VHDL也漸漸成爲業界間流通的一種標準。

– 1994 ： IEEE發表新版 VHDL Standard 1164

– 1996 ：結合電路合成的程式標準規格，發表IEEE 1164.3

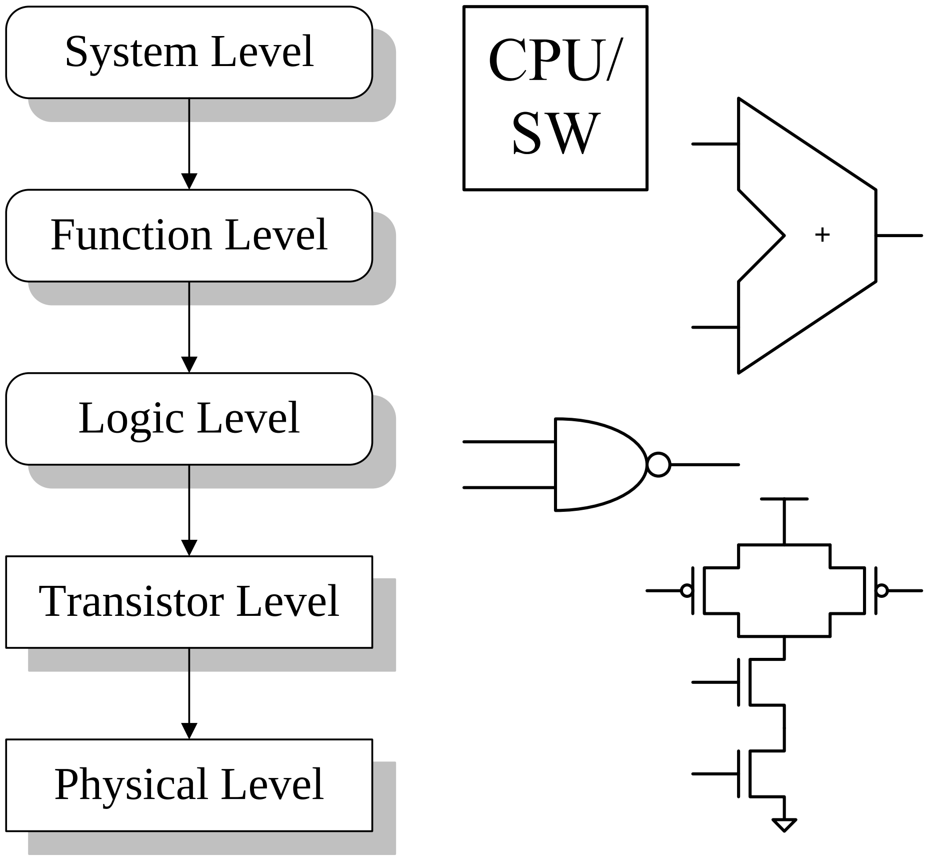
–現在：VHDL已經成爲「電子設計自動化」（EDA）工程的共通語言；

–未來：透過VHDL ，設計電子工業的「矽智產」（Silicon IP）。

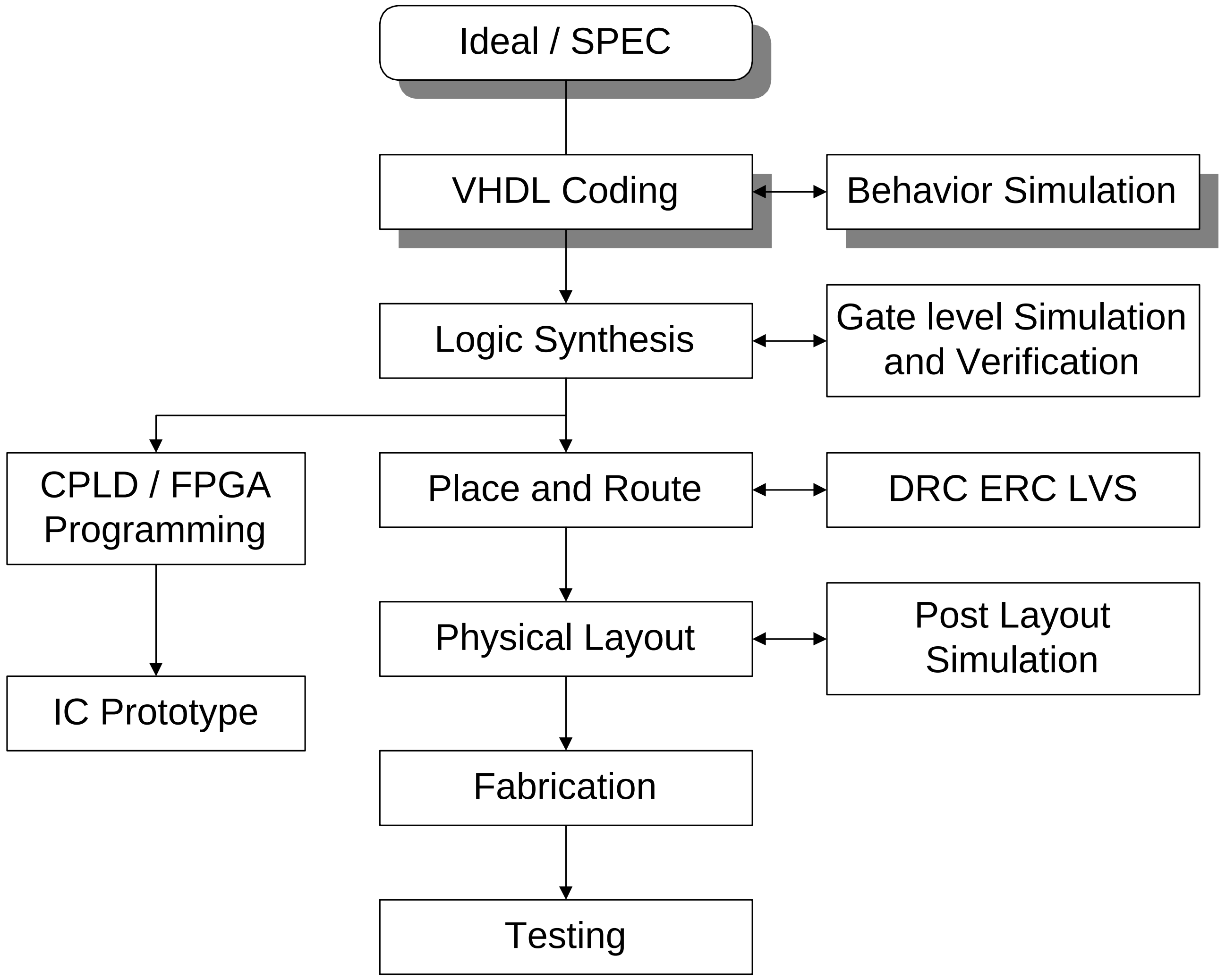
Introduction to VHDL

- Structure Model VHDL code
  - Entity
  - Architecture
  - Component
  - Configuration

Top-down IC Design Flow



VHDL Design Flow

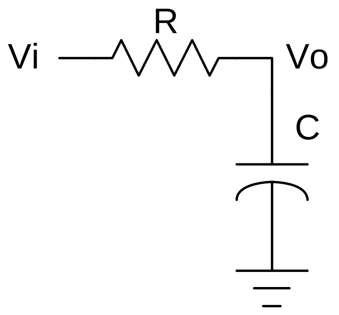


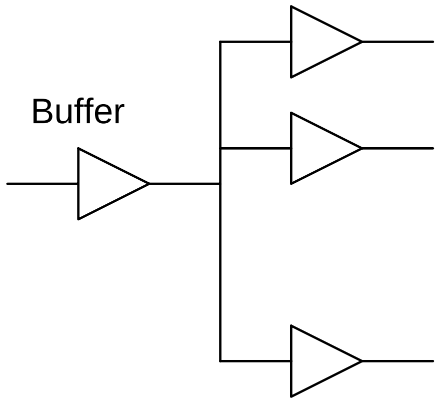
Timing Delay

- Ideal delay time = 0
  - for functional level
- Real delay time

  delay factors :
  - technology

    (TTL, CMOS, GaAs)
  - RC delay
  - fan-out





Simulating Strategy

- Time driven
  - Monitor value at every time period
  - High precision
  - SPICE, etc

- Even driven
  - Record when value change (event)
  - Less simulation time and memory
  - Behavior simulator :

    VHDL, Verilog

Concurrent and Sequential

- Concurrent
  - Each statement execute at the same time

  (logic circuits)
- Sequential
  - Statements execute in series

  (programming languages as C, FORTAN)

- Statement

```
A = B;
B = C;
```

- assume : A=1,B=2,C=3

⇨ concurrent result

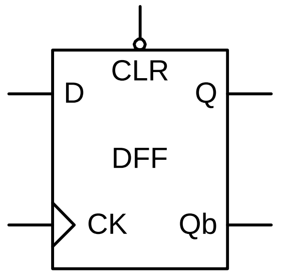
```
A = B = C= 3
```

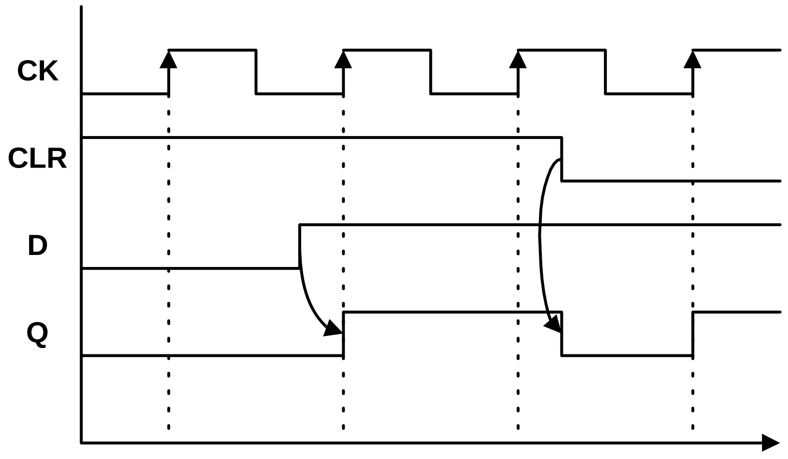
⇨sequential result

```
A = B = 2
B = C = 3
```

Synchronous and Asynchronous

- 同步的基準通常指 CLOCK
- Synchronous signal change value when clock assert.



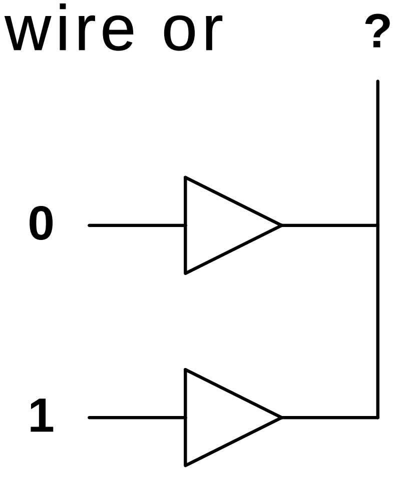


Logic Values and Multi-driven

- 9 value of IEEE std 1164 -1993 :

```
type STD_LOGIC is (
   `U' -- Uninitialized
   `X' -- Forcing Unknown
   `0' -- Forcing Low
   `1' -- Forcing High
   `Z' -- High Impedance
   `W' -- Weak Unknown
   `L' -- Weak Low
   `H' -- Weak High
   `-' -- Don't Care
   );
```

- Multi-driven problem
  - wire and
  - wire or



HDL Design Modeling

- Structure

  description of COMPONENT and WIRE

- Data-Flow

  concurrent signal assignment (select, conditional)

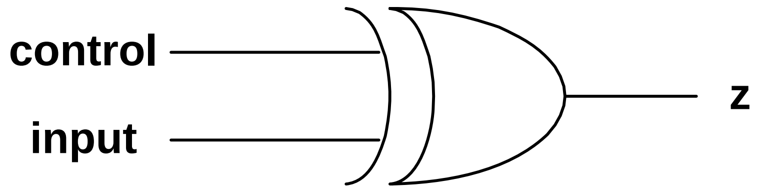
- Behavior

  Sequential signal assignment (high level language)

- Mix of above three

Traditional Design

<u>規格(SPEC) :</u>
當control=1時, z = input
否則 z = inv(input)

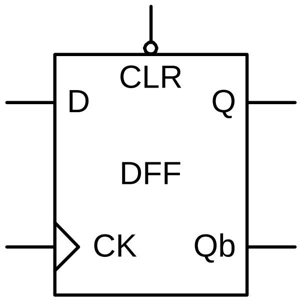


```
IF (control = ‘1’ ) THEN
   z <= input;
ELSE
   z <= NOT ( input );
ENDIF;
```

Entity

- 描述 I/O port 的規格
- 語法:

- Example: D Flip-Flop



```
ENTITY entity_name IS
      PORT ( … );

END entity_name ;
```

```
ENTITY dff IS
  PORT(D   :IN STD_LOGIC;
       CK  :IN STD_LOGIC;
       CLR:IN STD_LOGIC;
       Q   :OUT STD_LOGIC;
       Qb  :OUT STD_LOGIC);
END dff;
-- This is a comment
```

Architecture

- 描述內部電路
- 一個 Entity 可存在多個Architecture
- 語法:

```
ARCHITECTURE a_name OF e_name IS
  -- signals,variables declaration
BEGIN
  -- statements
END a_name ;
```

Architecture - Example : dff

```
ARCHITECTURE behavior OF dff IS
BEGIN
  PROCESS
  BEGING
    IF(CLR = ‘1’) THEN
      Q <= ‘0’;
    ELSE CK’EVENT AND CK=‘1’ THEN
      Q <= D;
    ENDIF;
    WAIT ON CLR, CK;
  END PROCESS;
END behavior;
```

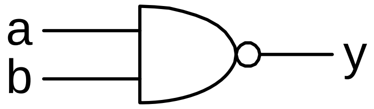
Component

- 宣告元件(logic gate)的名稱與 I/O port
- 語法:

```
COMPONENT comp_name
  PORT ( … );

END COMPONENT ;
```

- Example: nand2



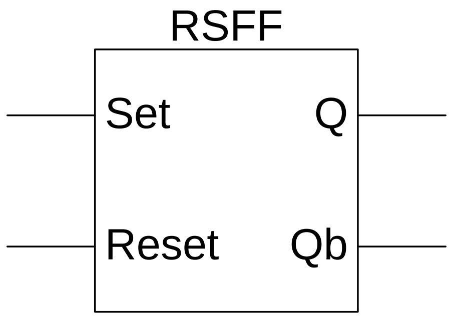
```
COMPONENT nand2
   PORT(a,b   :IN STD_LOGIC;
        y     :OUT STD_LOGIC);
END COMPONENT;
```

Configuration

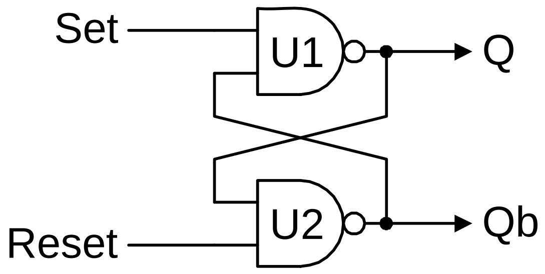
- 決定使用那一個Architecture
- 語法:

```
CONFIGURATION c_name OF e_name IS
   -- configurations
END c_name ;
```

Example : R-S Latch



Symbol



Schematic

```
ENTITY rsff IS
  PORT(set,reset:IN BIT;
       q,qb:BUFFER BIT);
END rsff;
ARCHITECTURE netlist OF rsff IS
  COMPONENT nand2
    PORT(a,b:IN BIT;
         c:OUT BIT);
  END COMPONENT;
BEGIN
  U1:nand2 PORT MAP(set,qb,q);
  U2:nand2 PORT MAP(reset,q,qb);
END netlist;
```

Port Mapping

- Mapping with order (pre-example)
- Mapping with **name association**

  Example:

```
U1:nand2 PORT MAP(a=>set ,b=>qb, c=>q);
U2:nand2 PORT MAP(c=>qb, a=>reset, b=>q);
```

VHDL Operator

- **Comparison Operator**
- **Logic Declaration**
- **Arithmetic Operator**
- **Bitwise Operator**

Logic Declaration

AND

OR

NOT

NOR

NAND

XOR

XNOR

邏輯運算子可以應用在三種資料型態：BIT, BOOLEAN 和STD_LOGIC

Logic Operators

- **Logic operators(**應用在三種資料型態：BIT, BOOLEAN 和STD_LOGIC**):**

  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;
```

Relational operators

- **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;
```

Arithmetic Operator

所有Synthetizer都支援

“+” **:** 加法

“-” **:** 減法

有些Synthetizer支援

“ * ” : 乘法

“ABS” : 求絕對值

編譯時設定數值運算時使用

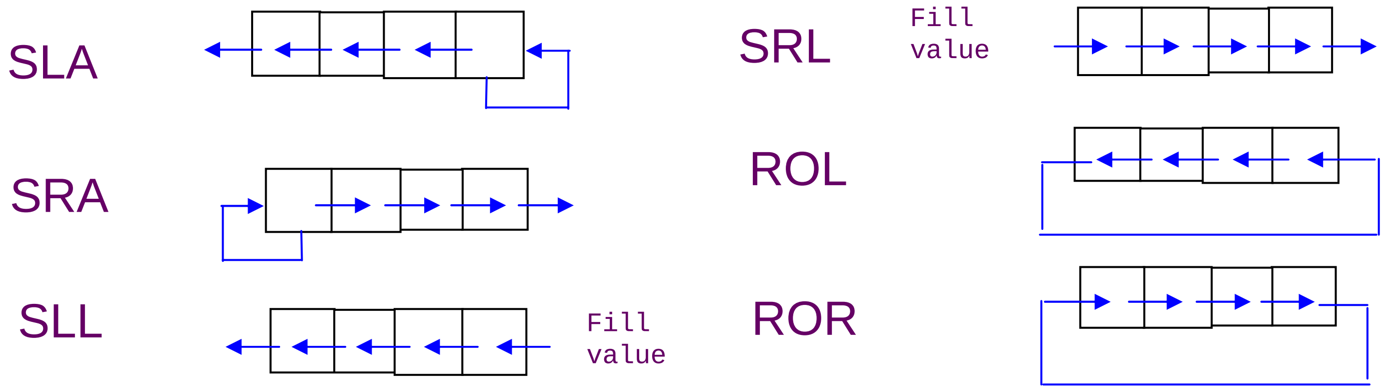
“/” : 除法

“” :** 次方

“MOD” : 求模數

“REM” : 求餘數

Bitwise Operator



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”

Arithmetic Operators

- **Arith. operators:**

  + (加) - (減) * (乘) / (除) **(次方)

```
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;
```

Data Flow Modeling

- Concurrent statements
- Signal Assignment
  - Simple
  - Conditional
  - Select
- Simulation Deltas
- Generic
- Block

Concurrent Statement

- 在ARCHITECTURE中所有陳述都是平行的
- Example：

```
ARCHITECTUE statements OF example IS
BEGIN
  y <= a AND b;      --signal assignment
  PROCESS(a, b, c); --process
  BEGIN
    --inside process is sequential
  END
  U1 : rsff PORT MAP(…);   --component
END statements;
```

Signal Assignment

- Let a = b

```
a <= b;
```

- Delay

```
a <= b AFTER 10ns;
```

- Logic Relation

```
c <= a AND b;
```

- Primitive Logic Operators

  AND OR NAND NOR

  NOT XOR

Conditional Signal Assignment

- 語法：

Signal assignment **WHEN** *condition* **ELSE**
the other value/signal **[WHEN…ELSE…]**

  - Example：

```
sel <= 0 WHEN a = ‘0’ AND b = ‘0’ ELSE
       1 WHEN a = ‘1’ AND b = ‘0’ ELSE
       2 WHEN a = ‘0’ AND b = ‘1’ ELSE
       3 WHEN a = ‘1’ AND b = ‘1’ ELSE
       4;
```

Select Signal Assignment

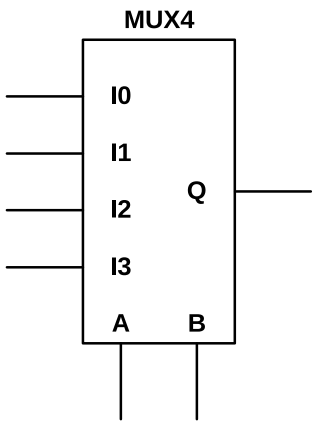
- 語法：

WITH *expression* **SELECT**
signal assignment **WHEN** *expression value*,
the other assignment **WHEN** *other expression value* **;**

- Example:

```
WITH sel SELECT
q <=  i0  WHEN 0,
      i1  WHEN 1,
      i2  WHEN 2,
      i4  WHEN 3,
      ‘X’ WHEN OTHERS;
```

Example - 4-1 MUX



Mux function table

A	B	Q
0	0	I0
0	1	I1
1	0	I2
1	1	I3

```
USE ieee.std_logic_1164.all

ENTITY mux4 IS
  PORT (i0,i1,i2,i3,a,b :IN STD_LOGIC);
        q :OUT STD_LOGIC);
END mux4

ARCHOITECTUR OF mux4 IS
  SIGNAL sel :INTEGER;
BEGIN
  WITH sel SELECT
  q <=  i0  WHEN 0,
        i1  WHEN 1,
        i2  WHEN 2,
        i4  WHEN 3,
        ‘X’ WHEN OTHERS;
  sel <= 0 WHEN a = ‘0’ AND b = ‘0’ ELSE
         1 WHEN a = ‘1’ AND b = ‘0’ ELSE
         2 WHEN a = ‘0’ AND b = ‘1’ ELSE
         3 WHEN a = ‘1’ AND b = ‘1’ ELSE
         4;
END;
```

Example - Decoder

2 to 4 decoder table

addr	word
00	1110
01	1101
10	1011
11	0111

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

ENTITY decoder IS
  PORT( addr :IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        word :OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END decoder;

ARCHITECTURE df OF decoder IS
BEGIN
  word <= "1110" WHEN addr = "00" ELSE
          "1101" WHEN addr = "01" ELSE
          "1011" WHEN addr = "10" ELSE
          "0111" WHEN addr = "11" ELSE
          "1111" ;
END df;
```

Driver Problem

- Bad Example :

```
ARCHITECTURE bad OF mux IS
BEGIN
  q <= i0 WHEN a =‘0’ AND b = ‘0’ ELSE ‘0’;
  q <= i1 WHEN a =‘0’ AND b = ‘1’ ELSE ‘0’;
  q <= i2 WHEN a =‘1’ AND b = ‘0’ ELSE ‘0’;
  q <= i3 WHEN a =‘1’ AND b = ‘1’ ELSE ‘0’;
END bad;
```

- Better Example :

```
ARCHITECTURE better OF mux IS
BEGIN
  q <=  i0 WHEN a =‘0’ AND b = ‘0’ ELSE
        i1 WHEN a =‘0’ AND b = ‘1’ ELSE
        i2 WHEN a =‘1’ AND b = ‘0’ ELSE
        i3 WHEN a =‘1’ AND b = ‘1’ ELSE
        ‘x’; -- unknown
END bad;
```

Generic

- Generic is design parameter such as
  - time delay
  - bus width

- Pass information to an instance of entity

- Example

```
ENTITY and2 IS
  GENERIC(rise,fall :TINE;
          load      :INTEGER);
  PORT(a,b :IN BIT;
       c   :OUT BIT);
END and2;

ARCHITECTURE load_dependent OF and2 IS
  SIGNAL internal :BIT;
BEGIN
  internal <= a AND b;
  c <= internal AFTER (rise+(load*2ns))
    WHEN internal = ‘1’
    ELSE internal AFTER (fall+(load*3ns));
END load_dependent;
```

Generic

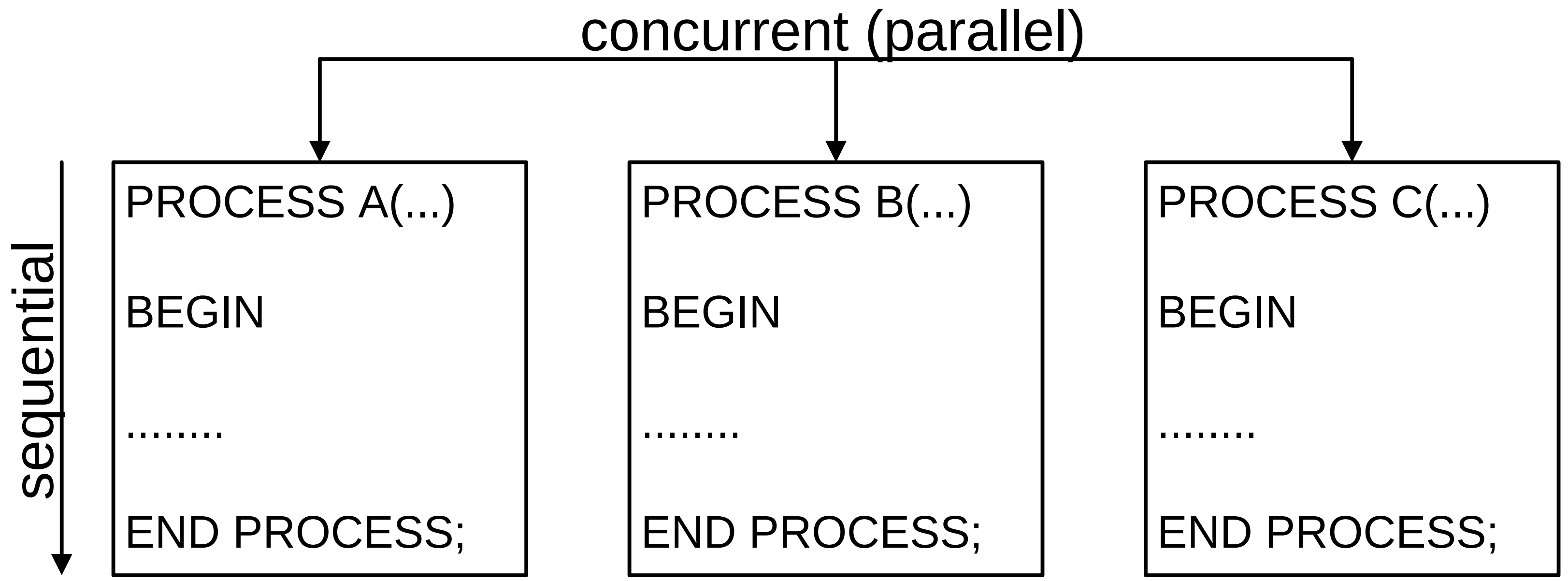
```
USE WORK.std_logic_1164.all
ENTITY test IS
  GENERIC(rise, fall :TIME; load :INTEGER);
  PORT(ina, inb, inc, ind : IN STD_LOGIC;
       out1, out2         : OUT STD_LOGIC);
END test;

ARCHITECTURE test_arch OF test IS
  COMPONENT and2
    GENERIC(rise,fall :TINE; load :INTEGER);
    PORT(a,b :IN BIT;
         c   :OUT BIT);
BEGIN
  U1 :and2     GENERIC MAP(10ns, 20ns, 3)
               PORT MAP (ina, inb, out1);
  U2 :and2     GENERIC MAP(9ns, 11ns, 5)
               PORT MAP (inc, ind, out2);
END test_arch;
```

Sequential Processing

- Process Statements
- Sequential Statements
  - IF-THEN-ELSE
  - CASE
  - LOOP
  - ASSERT
  - WAIT

Process Statements



Process Statements 語法

```
[ process_label：] PROCESS( sensitivity_list )
    { process_declaration }
BEGIN
    { sequential_statements }
END PROCESS [ process_label ];
```

Example :

```
comb：PROCESS（a, b, c）
  VARIABLE temp：STD_LOGIC;
BEGIN
  temp：=NOT（a AND b）;
  IF（c = ‘1’）  THEN
  y <= ‘0’;
  ENDIF;
END PROCESS comb;
```

Process Statements 語法

- 語法說明
  - process_label：流程的名稱標記，可有可無。
  - sensitivity_list：使程式進入流程的觸發信號。一旦觸發信號改變時，即執行一次這個流程，否則流程不會被執行。
  - process_declaration：宣告流程內部使用的信號或變數。
  - sequential_statement；順序式的陳述。

Sequential Statement

- IF-THEN-ELSE
- CASE
- LOOP
- ASSERT
- WAIT

IF-THEN-ELSE

語法:

```
IF  condition  THEN
  sequential_statements
{ ELSIF condition THEN
  sequential_conditions }
[ ELSE
  sequential_statements ]
END IF;
```

- Example (1) 2-1 Mux

```
PROCESS（sel, i0,i1)
      BEGIN
      IF（sel=’0’）   THEN
            y <= i0;
      ELSE
            y <= i1;
      END IF;
END PROCESS;
```

IF-THEN-ELSE

- Example (2)

```
PROCESS（a, b）
  BEGIN
  IF（a = ‘0’）THEN
    y <= ‘0’;
  ELSE
    y <= ‘b’;
  END IF;
END PROCESS;
```

- Example (3) non-full branch

```
PROCESS（i0, i1, a, b)
BEGIN
  IF（a = ‘0’ AND b = ‘1’）THEN
    y <= i0;
  ELSE IF （a=’1’ AND b=’0’）THEN
    y <=i1;
  END IF;
END PROCESSS;
```

CASE Statement

- 語法

```
CASE expression IS
  WHEN choice
{ sequential_statements }
  WHEN choice
{sequential_statements}
  WHEN OTHERS
{sequential_statements}
END CASE ;
```

- Example 4-1 Mux

```
CASE sel IS
  WHEN “00”
    Z <= I0 ;
  WHEN “01”
    Z <= I1 ;
  WHEN “10”
    Z <= I2 ;
  WHEN “11”
    Z <= I3 ;
  WHEN OTHERS
    Z <= ‘X’ ;
END CASE ;
```

LOOP Statements

- WHILE LOOP
- FOR LOOP

- 語法

- Iteration_scheme :

  WHEN (*condition*) LOOP

  FOR *i* IN *range* LOOP

```
[loop_label：] [iteration_scheme] LOOP
  sequential_statements
END LOOP [loop_label]；
```

LOOP Statements

- Example (1)

```
FOR i IN 10 DOWNTO 1  LOOP
  i_squared(i)：= i * i；
END LOOP；
```

- Example (2)

```
WHILE ( day = weekday ) LOOP
  day := get_next_day(day);
END LOOP;
```

LOOP Statements

- Example (3)

```
PROCESS ( clk )
  TYPE day_of_week IS ( sun, mon, tue, wed, thur, fri, sat);
BEGIN
  FOR i IN day_of_week LOOP
    IF i = sat THEN
      son <= mow_lawn;
    ELSIF i = sun THEN
      church <= family;
    ELSE
      dad <= go_to_work;
    ENDIF;
  END LOOP;
END PROCESS;
```

NEXT and EXIT

- NEXT 相當於 C 語言中的 continue, 使迴圈繼續執行

- Example

```
FOR i IN 0 TO 255 LOOP
  IF (done (i) = TRUE) THEN
    NEXT；
  ELSE
    done (i)：= TRUE；
  END IF；
  q <= a(i) AND b(i)；
END LOOP；
```

NEXT and EXIT

- EXIT 相當於 C 語言中的 break, 跳離迴圈,執行迴圈以後的陳述

- Example

```
FOR i IN 0 TO 255 LOOP
  IF (int_a <= 0) THEN
  -- less than or – equal to
    EXIT;
  ELSE
    int_a：int_a - 1；
    q(i) <= 3.1416/REAL(int_a*i);
  END IF；
END LOOP；
  y<=q；
```

NEXT and EXIT

- 利用label 跳離不同迴圈

```
PROCESS (a,b)
BEGIN
  first_loop：FOR i IN 0 TO 100 LOOP
    second_loop：FOR j IN 1 TO 10 LOOP
      .........
      EXIT second_loop；--exit the second loop only
      ...........
      EXIT first_loop；--exit the first loop only
    END LOOP；
    y<=a；
  END LOOP；
  y<=b；
END PROCESS
```

WAIT Statement

- 功能： 等待某信號改變，或爲某個値 等待時間過去
- 目地： 描述暫存器 (latch,flip-flop)

WAIT ON	signals changes
WAIT UNTIL	an expression is true
WAIT FOR	a specific amount of time

WAIT Statement

- Example (1) Positive edge-trigger D flip-flop

```
PROCCESS
BEGIN
   WAIT UNTIL clock =‘1’ AND clock’EVENT
      q <= d;
END PROCCESS
```

- Example (2) Asynchronous reset D flip-flop

```
PROCCESS
BEGIN
   IF (reset = ‘1’) THEN
      q <= 0;
   ELSE IF clock’EVENT AND clock=‘1’ THEN
      q <= d;
   END IF;
   WAIT ON reset, clock;
END PROCCESS
```

WAIT Statement

- Example (3) Synchronous reset D flip-flop

```
PROCCESS
BEGIN
  WAIT UNTIL clock=‘1’ AND clock’EVENT
  IF (reset = ‘1’) THEN
    q <= 0;
  ELSE
    q <= d;
  END IF;
END PROCCESS
```

Sensitivity list ?

WAIT Statement

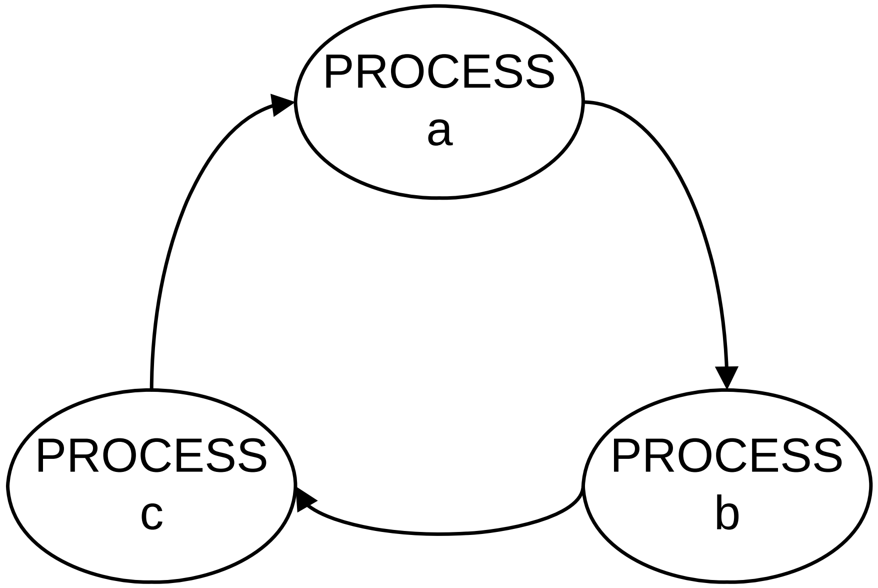
- Multiple WAIT conditions

```
WAIT ON nmi,interrupt UNTIL ((nmi = TRUE) or
                (interrupt = TRUE)) FOR 5 usec;
```

等待 nmi 或 interrupt 的信號變化,
且其中之一的值爲 TRUE
或者過 5 usec 再後繼續執行

WAIT Statement

- Dead-lock
  - waiting signal each other

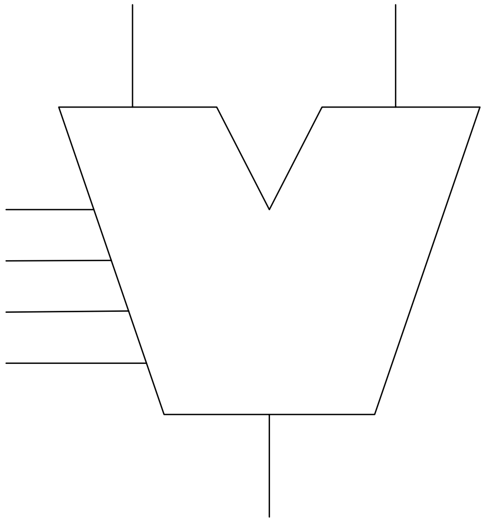


WAIT Statement

- **Sensitivity list versus WAIT Statement**

當PROCESS中含有WAIT等待的陳述時,就不能再有sensitivity list觸發信號,否則會造成互相等待的情形

Demonstrative Example



f3	f2	f1	f0	Function
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	0
0	1	0	0	
0	1	0	1	
0	1	1	0	A⊕B
0	1	1	1	
1	0	0	0	+B
1	0	0	1	
1	0	1	0	B
1	0	1	1	AB
1	1	0	0	1
1	1	0	1	
1	1	1	0	A+B
1	1	1	1	A

ALU.VHD