Package

ใช้สำหรับเก็บข้อมูลต่างๆที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิตอล เช่น

- โปรแกรมย่อย(Subprogram)
- Type
- Constants
- Signal
- Aliases
- Attributes
- รูปแบบจำลอง (Model) มาตราฐานต่างๆ เช่น อุปกรณ์ มาตราฐานของไอซี ตระกูล 74xx
- Disconnection Specification

ข้อมูลเหล่านี้สามารถนำไปใช้ได้โดย Entity design unit Architecture design unit หรือ Package design unit อื่นๆ โดยปกติ Package จะแบ่งเป็น 2 ส่วนคือ

Package Declaration

Package Body

Package Declaration

เป็นส่วนที่ใช้กำหนดชื่อ (Identifier) ของสิ่งที่ประกาศอยู่ภายใน Package สิ่งต่างๆใน Package Body ต้องมีการกำหนดชื่อไว้ที่ส่วนนี้ ถ้าไม่กำหนด ภายนอกจะไม่สามารถเรียกใช้ได้ แต่ Package อาจจะมีอยู่ใน Package Declaration เพียงอย่างเดียวก็ได้ เช่น Type หรือ Signal ซึ่งภายนอกก็ยังคงเรียกใช้ได้ ในทำนองเดียวกัน Package อาจจะมีเพียง Package body เพียงอย่างเดียวก็ได้ แต่จะไม่สามารถเรียกใช้ได้จากรูปแบบอื่นๆ

Package Declaration

รูปแบบการเขียน Package declaration

```
PACKAGE package_name IS
        Package_declarative_part
END package_name;
```

Package_declarative_part หมายถึงส่วนที่ใช้ประกาศต่างๆเช่นส่วนประกาศกำหนดโปรแกรมย่อย

- Type declaration
- Subtype declarations
- Object declarations (signals, constants)
- Alias declarations
- Attribute specificqations
- Component specfications
- Disconnection specification

ตัวอย่าง

```
PACKAGE example IS
        TYPE cd IS ('C', 'D');
        CONSTANT pi IS: REAL := 3.14159;
        COMPONENT ttl_7400 IS
                PORT( a, b: IN BIT;
                      c: OUT BIT);
        END COMPONENT;
        SIGNAL sg_reset : BIT;
END example;
```

Package Body

Package Body ถูกใช้ในกรณีที่ Package declaration ประกาศชื่อเป็นโปรแกรมย่อย หรือ deferred constant

รูปแบบการเขียน Package body

```
PACKAGE BODY package_name IS
                declarative_part
END package_name;
```

package_name จะต้องเป็นชื่อเดียวกับชื่อที่กำหนดไว้ใน package declaration

ตัวอย่างการเขียน package

```
PACKAGE func_avr IS
        FUNCTION mean(a,b : real) RETURN REAL;
END func_avr;
PACKAGE BODY func_avr IS
        FUNCTION mean(a, b: REAL) RETURN REAL IS
        BEGIN
                RETURN (a + b)/2;
        END mean;
END func_avr;
```

ตัวอย่างการเขียน package

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

```
package lfsr_pkg is

    function many_to_one_fb (DATA, TAPS :std_logic_vector) return std_logic_vector;
    function one_to_many_fb (DATA, TAPS :std_logic_vector) return std_logic_vector;

end;
```

```
package body lfsr_pkg is

    function many_to_one_fb (DATA, TAPS :std_logic_vector) return std_logic_vector is
        variable xor_taps :std_logic;
    begin
       ..................
       ..................
    end function;

    function one_to_many_fb (DATA, TAPS :std_logic_vector) return std_logic_vector is
        variable xor_taps  :std_logic;
        variable all_0s    :std_logic;
    begin
       ..................
       ..................
    end function;
end package body;
```

Configuration Design Unit

จากที่กล่าวมาแล้วว่า Entity design unit หนึ่งๆ อาจจะมี Architecture ได้หลายหน่วย ดังนั้นการจำลองการทำงานจะต้องมีการระบุว่าจะใช้ architecture อันไหนให้ Simulator ทราบ โดยใช้ CONFIGURATION ประกอบ entity กับ architecture design unit รูปแบบการเขียน configuration เป็นดังนี้

```
CONFIGURATION identifier OF entity_name IS
        configuration_declarative_part
END;
```

ตัวอย่าง แบบจำลอง ของเกต AND ที่มี 2 architecture

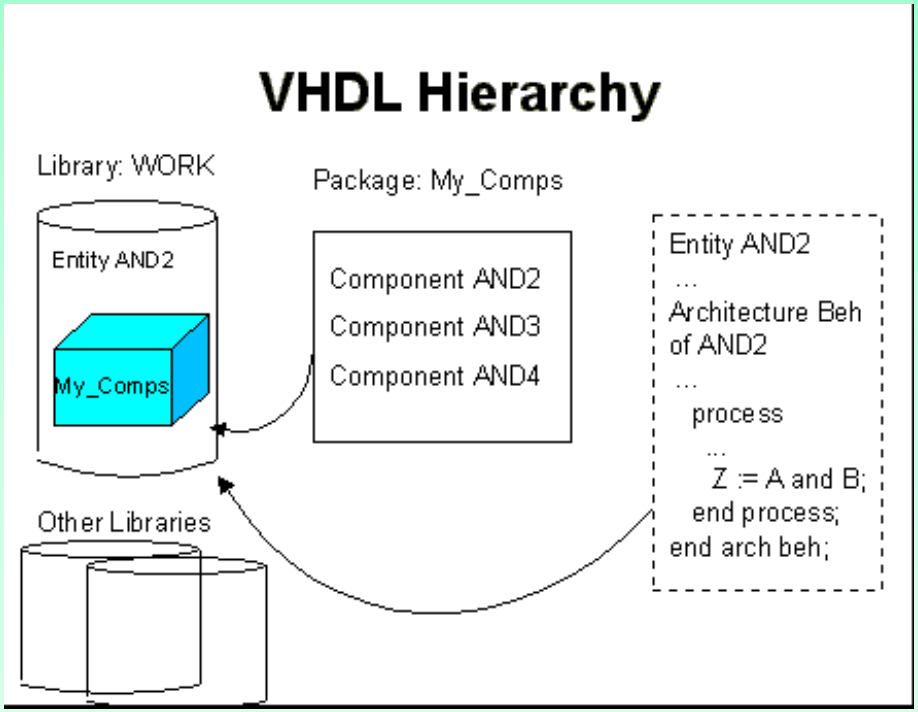
```
ENTITY and2 IS
        PORT (a,b :IN BIT;
        c : OUT BIT);
END and2;
ARCHITECTURE adatflow OF and2 IS
BEGIN
        c <= a and b;
END dataflow;
--
ARCHITECTURE beh OF and2 IS
BEGIN
        PROCESS (a, b)
        BEGIN
                IF (a = '1' and b = '1') THEN
                        c <= '1';
                ELSE
                        c <= '0';
                END IF;
        END PROCESS;
END beh;
```

เมื่อต้องการระบุให้ใช้ architecture beh ให้ใช้ configuration ดังนี้

```
CONFIGURATION beh_arc OF and2 IS
        FOR beh
        END FOR;
END beh_arc;
```

ไลบรารี (Libraries)

เป็นที่เก็บอุปกรณ์ต่างๆ หรือแบบจำลอง ต่างๆ ที่สามารถนำมาใช้งานได้ ในมาตราฐาน IEEE 1076 ได้กำหนดกฎเกณฑ์ของ library ไว้ว่า ส่วนที่เป็น design library ของ VHDL สามารถนำไปใช้ได้โดยอ้างชื่อของ library นั้นๆ ชื่อของ library นี้เรียกว่า ชื่อสัญลักษณ์ (Symbolic name)



ข้อมูลภายใน library

หน่วยหลัก (Primary units)

entity declarations

package declarations

confiuration specifications

หน่วยรอง (Secondary units)

architecture bodies

package bodies

การวิเคราะห์หรือการแปลของภาษา VHDL จะทำหน่วยหลักก่อนแล้วจึงทำหน่วยรอง หลังจากแปลแล้วทั้งหน่วยหลักและหน่วยรองจะถูกเก็บอยู่ใน library เดียวกัน

library มี 3 ประเภทได้แก่

STANDARD library

เป็น library มาตราฐาน ใช้ชื่อว่า ***STD*** ภายใน library ประกอบด้วย 2 package คือ

package STANDARD กำหนดโดย IEEE 1076 ภายในประกอบด้วยการประกาศ ต่างๆ อาทิ เช่น VHDL type (REAL, INTEGER, TIME,BIT, BOOLEAN)

package TEXTIO ภายในประกอบด้วยโปรแกรมย่อยต่างๆ ที่ใช้สำหรับแก้ไขดัดแปลงข้อมูลที่เขียนในรูปรหัส ASCII

Working library

หมายถึง library ที่อยู่ใน working directory ที่กำลังทำงานอยู่ ชื่อ library นี้จะถูกกำหนดให้ชื่อ ***WORK*** เสมอ model VHDL source file ที่เขียนขึ้นจะถูกแปลแล้วเก็บไว้ใน library นี้

Resource library

ทำหน้าที่เป็นที่เก็บข้อมูลเพิ่มเติม สามารถตั้งชื่ออะไรก็ได้ แต่อย่าให้ซ้ำกับชื่อ library สงวนของ VHDL คือ ***STD*** และ ***WORK***

Visibility

หมายถึงการทำให้หน่วยอื่นๆมองเห็นข้อมูลที่ต้องการเรียกใช้ เช่นการที่จะใช้ข้อมูลภายใน package ต้องทำให้ package นั้นถูกมองเห็นเสียก่อน โดยใช้คำสั่ง LIBRARY และคำสั่ง USE โดย คำสั่ง LIBRARY ใช้ระบุชื่อ library ส่วนคำสั่ง USE ใช้ระบุชื่อ package ที่อยู่ใน library นั้น รูปแบบการใช้เป็นดังนี้

LIBRARY library_name;

USE library_name.package_name[.element_of _package หรือ . **ALL**]

ตัวอย่าง

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL
ENTITY test is
:
END test;
```

คำสั่ง **ALL** ในตัวอย่างนี้หมายถึงให้ทุกๆสิ่งที่อยู่ใน package ชื่อ std_logic_1164 ใน library ชื่อ ieee มองเห็นได้สำหรับ entity ชื่อ test และส่วนที่เป็นหน่วยรอง

ตัวอย่าง(ต่อ)

```
package c_cnt is
        type n_num is array (3 downto 0) of integer range 0 to 15;
end c_cnt;
use work.c_cnt.all;
entity ct4 is
  port (dout : out n_num;
        clk, reset : in bit);
end ct4;
```

สำหรับอีกตัวอย่างหนึ่ง แสดงให้เห็นการใช้ package และการทำให้มองเห็น package

คอมโปเนนต์ (Component)

เมื่อต้องการเรียกใช้อุปกรณ์ที่มีอยู่แล้ว และอุปกรณ์นั้นเขียนอยู่ในรูปของ VHDL จะใช้คำสั่งเกี่ยวกับ คอมโปเนนต์ การใช้งานคอมโปเนนต์ มี 2 ขั้นตอน คือ การประกาศคอมโปเนนต์ และการเรียกใช้คอมโปเนนต์

การประกาศคอมโปเนนต์

เป็นการประกาศว่าจะมีคอมโปเนนต์อะไรบ้างที่จะนำมาใช้งาน การประกาศนี้จะทำอยู่ในส่วนของ Architecture โดยอยู่ก่อน คำสัง Begin ของ Architecture และรูปแบบการประกาศเป็นดังนี้

```
COMPONENT  Component_name
        GENERIC (Generic_list);
        PORT (Port_list);
END COMPONENT;
```

เหมือนกับการเขียน ENTITY

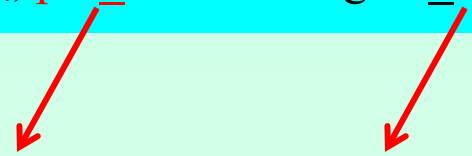
การเรียกใช้คอมโปเนนต์

แบบอ้างอิงตำแหน่ง (Port association list)

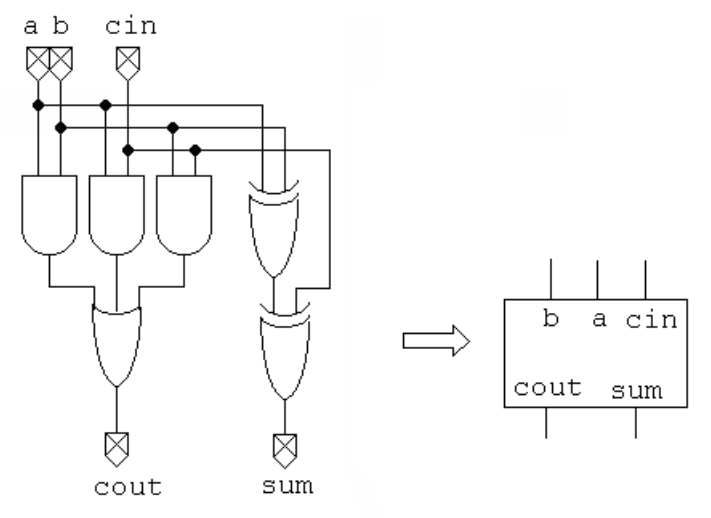
Label: Component_name PORT MAP (signal_name {, signal_name});

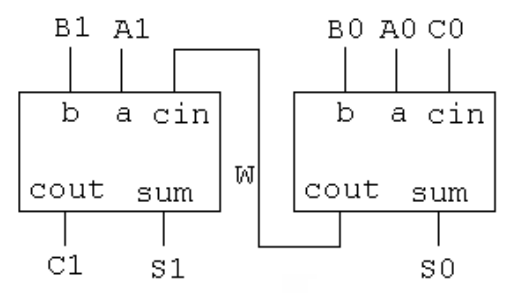
แบบอ้างอิงชื่อ (Name association list)

Label: Component_name PORT MAP (pin_name => signal_name
{, pin_name => signal_name});



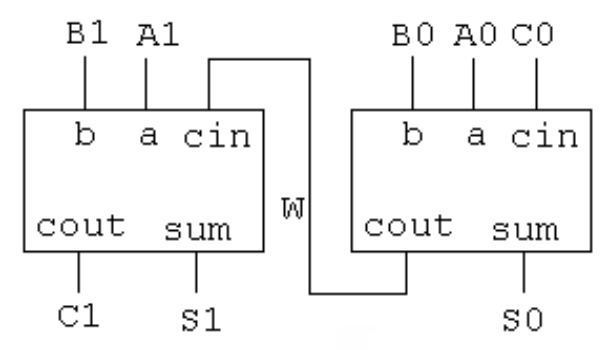
ตัวอย่างแบบอ้างอิงตำแหน่ง






```
ARCHITECTURE struc OF ripple
-- ประกาศคอมโปเนนต์
COMPONENT fulladd
 port (a, b, cin : in bit;
       sum, cout : out  bit);
END COMPONENT
-- ประกาศสัญญาณ
Signal A0, A1, B0, B1, C0, C1,S0, S1, W  : BIT ;
BEGIN
   U1: fulladd PORT MAP ( A0, B0, C0, S0, W);
   U2: fulladd PORT MAP (A1, B1, W,  S1, C1);
END stru;
```

ตัวอย่างแบบอ้างอิงชื่อ




```
BEGIN
        U1: fulladd PORT MAP ( a  => A0,
                               b => B0,
                               cin => C0,
                               sum =>  S0,
                               cout => W);
        U2: fulladd PORT MAP ( a  => A1,
                               b => B1,
                               cin => W,
                               sum =>  S1,
                               cout => C1);
END stru;
```