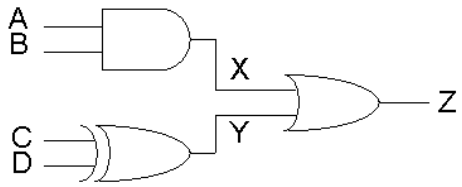
บทที่ 2
องค์ประกอบของภาษา VHDL

2.1 ความรู้เบื้องต้นเกี่ยวกับภาษา VHDL

ก่อนที่จะเข้าสู่รายละเอียดต่างๆของภาษา VHDL ควรที่จะเข้าใจความหมายของสิ่งต่างๆที่เกี่ยวกับภาษา VHDL เสียก่อนดังต่อไปนี้

2.1.1 การทำงานแบบลำดับและแบบขนาน

การทำงานของภาษาโปรแกรมทั่วๆไปเช่นภาษา C จะมีลักษณะเป็นการทำงานจากคำสั่งแรกลงไปสู่คำสั่งสุดท้าย เมื่อทำคำสั่งแรกเสร็จจึงจะทำคำสั่งที่สอง เป็นเช่นนี้จนถึงคำสั่งสุดท้าย ผลลัพธ์ของคำสั่งที่อยู่ก่อนจะมีผลต่อคำสั่งที่อยู่หลัง คือ จะถูกนำไปใช้ในคำสั่งต่อๆไป ลักษณะการทำงานเช่นนี้เรียกว่าเป็นการทำงานแบบลำดับ (Sequential) ส่วนการทำงานอีกชนิดหนึ่งเรียกว่าการทำงานแบบขนาน (Concurrent) ตำแหน่งหรือลำดับที่ของคำสั่งที่อยู่จะไม่มีผลต่อการทำงาน ทุกๆคำสั่งถือว่าดำเนินไปพร้อมๆกัน ลักษณะเช่นนี้เป็นไปตามความเป็นจริงทางฟิสิกส์ เช่นในวงจรดิจิตอลตามรูป 2-1 (ก) ณ เวลาหนึ่งๆสัญญาณ X Y และ Z ต่างก็ทำงานไปพร้อมๆกัน สัญญาณ Z ไม่ได้รอการทำงานของ X และ Y เพียงแต่ค่าของสัญญา X และ Y จะมีผลต่อสัญญาณ Z ณ เวลาถัดไป เมื่อ VHDL เป็นภาษาสำหรับฮาร์ดแวร์ ดังนั้นการทำงานของภาษา VHDL จึงมีคุณสมบัติเป็นแบบขนาน ตัวอย่างคำสั่งของภาษา VHDL ในรูปที่ 2-1 (ข) และ (ค) ซึ่งใช้แทนวงจรดิจิตอลในรูป 2-1 (ก) การเขียนคำสั่งทั้งสองแบบนี้ต่างให้ผลลัพธ์การทำงานเหมือนกัน ถึงแม้ว่าการวางลำดับที่ของคำสั่งต่างกัน ทั้งนี้เพราะว่า VHDL ถือว่าทุกๆคำสั่งต่างทำงานไปพร้อมๆกัน



(ก)

```
X <= A and B;
Y <= C xor D;
Z  <= X or Y;
```

(ข)

```
Z  <= X or Y;
X <= A and B;
Y <= C xor D;
```

(ค)

รูปที่ 2-1 วงจรดิจิตอลและลักษณะคำสั่งของ VHDL

2.1.2 ออบเจ็กต์ (OBJECTS)

คำว่าออบเจ็กต์ในภาษา VHDL หมายถึงองค์ประกอบหนึ่งๆในระบบ ใช้สำหรับเก็บค่าต่างๆ มีอยู่ด้วยกัน 3 แบบ(Class) ได้แก่ ค่าคงที่ (Constant) ตัวแปร(Variable) และ สัญญาณ(Signal)

ค่าคงที่ (Constant) เป็นออบเจ็กต์ประเภทหนึ่งเมื่อกำหนดค่าให้แล้วจะคงค่านั้นตลอดไป ไม่สามารถดัดแปลงหรือแก้ไขได้

ตัวแปร (Variable) เป็นออบเจ็กต์ ที่สามารถกำหนดค่าและเปลี่ยนแปลงค่าได้ตลอดการทำงาน แต่ ณ เวลาหนึ่งๆจะเก็บค่าได้เพียงค่าเดียวเท่านั้น

สัญญาณ (Signal) เป็นออบเจ็กต์ ที่ใช้สำหรับการเชื่อมต่อโมดูลต่างๆเข้าด้วยกัน ถ้าเปรียบเทียบอย่างง่ายๆ Signal ใช้แทนขาของอุปกรณ์ หรือสายไฟที่ใช้เชื่อมต่ออุปกรณ์ต่างๆในวงจรเข้าด้วยกัน ดังนั้นในกรณี Signal ที่เป็นขาของอุปกรณ์ จะต้องมีการระบุทิศทางของ Signal นั้น ส่วนกรณีที่เป็นสายไฟ สำหรับการเชื่อมต่อไม่จำเป็นต้องระบุทิศทางของ Signal

ชื่อ Object การตั้งชื่อ Object ใช้กฎเกณฑ์ดังต่อไปนี้

- เป็นพยัญชนะ ตัวเลข หรือเครื่องหมายขีดเส้นใต้ในภาษาอังกฤษก็ได้แต่ต้องขึ้นต้นด้วยพยัญชนะเสมอ โดยมีความยาวไม่จำกัด
- พยัญชนะตัวเล็กหรือตัวใหญ่มีค่าเท่ากัน
- ห้ามใช้คำสงวนของ VHDL เช่น PORT ENTITY SIGNAL และ PROCESS เป็นต้น

สำหรับรายละเอียดเกี่ยวกับ Object แบบต่างๆ จะกล่าวถึงในภายหลัง

การประกาศใช้ออบเจ็กต์ การที่จะใช้ออบเจ็กต์ใดๆ ในภาษา VHDL ต้องมีการประกาศใช้ก่อน โดยใช้ชุดคำสั่งดังนี้

```
OBJECT_CLASS object_name : TYPE [:= initial_value];
```

OBJECT_CLASS หมายถึง CONSTANT SIGNAL หรือ VARIABLE
Object_name หมายถึงชื่อของออบเจ็กต์
TYPE หมายถึงชนิดของออบเจ็กต์
Initial_value หมายถึงค่าเริ่มต้นของออบเจ็กต์ ซึ่งจะมีหรือไม่มีก็ได้

ตัวอย่างการประกาศสัญญาณ

```
SIGNAL      A_signal : BIT;
SIGNAL      B_signal : BIT := '0';
```

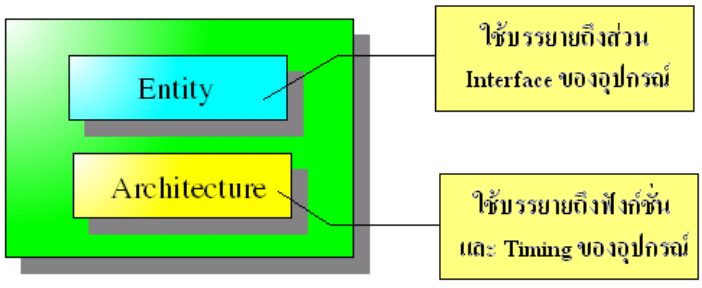
สัญญาณ A_signal ไม่ได้กำหนดค่าเริ่มต้น
แต่ B_signal กำหนดให้เริ่มต้นเป็น '0'

2.2 โครงสร้างของภาษา VHDL

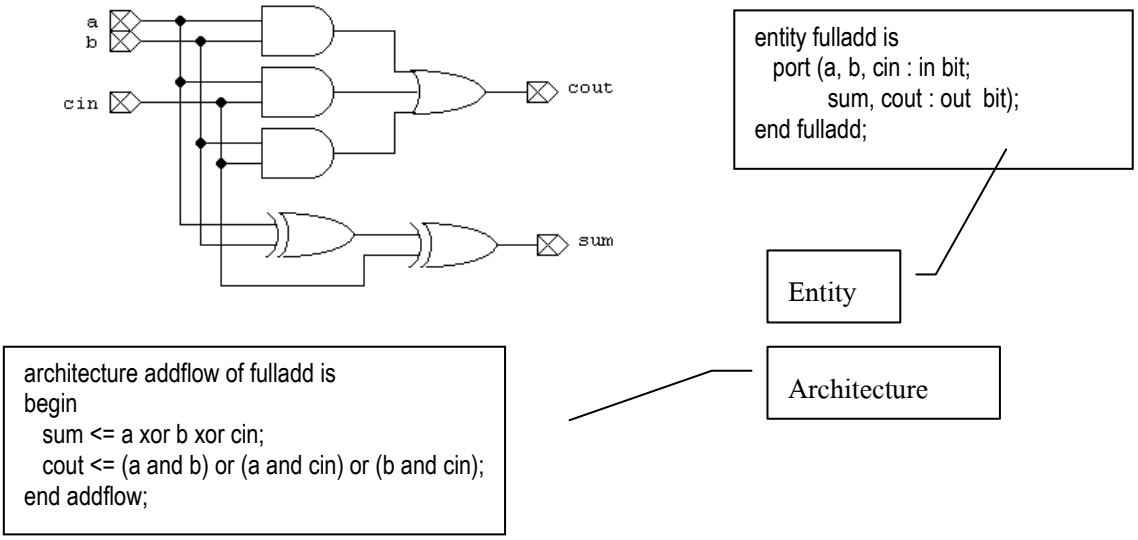
ภาษา VHDL ที่ใช้เพื่อการออกแบบวงจดิจิตอลจะต้องมีองค์ประกอบหลักๆดังนี้

- เอนทิตี (Entity)
- อาร์คิเทคเจอร์ (Architecture)
- แพ็คเกจ (Package)
- คอนฟิคกูเรชั่น (Configuration)

โดยมีส่วนประกอบที่สำคัญ 2 ส่วนคือ Entity และ Architecture



รูปที่ 2-2 โครงสร้างพื้นฐานของภาษา VHDL



รูปที่ 2-3 ลักษณะของ Entity และ Architecture เมื่อเทียบกับโลจิกไดอะแกรม

2.2.1 เอนทิตี (Entity)

เป็นส่วนที่ใช้สำหรับติดต่อกับอุปกรณ์ภายนอกเปรียบเทียบได้กับขาของอุปกรณ์หรือขาของไอซี อุปกรณ์ภายในวงจรหรือโมดูลที่ออกแบบเมื่อต้องการติดต่อกับโลกภายนอกจะอาศัย Entity เป็นส่วนในการส่งผ่านข้อมูล ดังนั้นภายใน Entity ต้องมีส่วนประกอบที่เป็นทางผ่าน หรือส่วนที่ใช้กำหนดค่าพื้นฐานต่างๆ แต่ทั้งนี้ทั้งนั้นการเขียนโมดูลในส่วนของ Entity นี้อาจไม่ต้องมีส่วนประกอบใดๆอยู่เลยก็ ได้ถ้าเป็นการเขียนเพื่อใช้ทดสอบโมดูลอื่น โมดูล VHDL ลักษณะนี้เรียกว่า Testbench รูปแบบการเขียน Entity มีดังนี้

```
-- Entity Design Unit
ENTITY entity_name IS
        GENERIC ( generic_list);
        PORT (port_list);
END entity_name;
```

ความหมายของแต่ละส่วนเป็นดังนี้

ENTITY ... IS .... END .....; ใช้ระบุส่วนที่เป็น Entity

entity_name เป็นชื่อของ Entity การตั้งชื่อใช้กฎเกณฑ์เดียวกับการตั้งชื่อออบเจ็กต์

GENERIC ใช้กำหนดค่าพารามิเตอร์ต่างๆ เช่นค่าพารามิเตอร์เกี่ยวกับเวลาหน่วง ของอุปกรณ์ภายในโมดูลที่ออกแบบ

PORT ใช้กำหนดขาหรือสัญญาณของโมดูลที่จะใช้ติดต่อกับโลกภายนอก พอร์ท (Port) แต่ละพอร์ทต้องประกอบด้วย:

- ชื่อพอร์ท (Name)
- โหมดหรือทิศทางของพอร์ท (Mode)
  - IN : เป็นสัญญาณอินพุท (Input)
  - OUT : เป็นสัญญาณเอาท์พุท (Output)

- INOUT : เป็นสัญญาณสองทิศทาง (Bidirectional)
- BUFFER : ถ้าพิจารณาภายนอกอุปกรณ์มีลักษณะแหมือน OUT และถ้าพิจารณาภายในอุปกรณ์มีลักษณะเป็น INOUT

- Type เป็นแบบหรือชนิดของข้อมูลของพอร์ท ซึ่งภาษา VHDL จะมีแบบมาตรฐานอยู่ เรียกว่า Standard Types ได้แก่
  - BIT มีค่าเป็น 0 หรือ 1
  - BIT_VECTOR เป็นกลุ่มของ BIT
  - BOOLEAN มีค่าเป็นจริง (1) หรือเทจ (0)
  - INTEGERมีค่าเป็นเลขจำนวนเต็ม
  - REAL มีค่าเป็นเลขจำนวนจริง
  - TIME มีค่าเป็นเวลา
  - CHARACTER เป็นตัวอักขระ
  - STRING เป็นกลุ่มของตัวอักขระ

  ทั้งนี้รายละเอียดเกี่ยวกับ Type จะกล่าวในบทต่อไป

เครื่องหมาย ";" ที่ปรากฏอยู่ส่วนท้ายของแต่ละบรรทัดเป็นการบอกว่าจบประโยคคำสั่ง ซึ่งเครื่องหมายนี้สามารถใช้ได้ทุกที่ในการเขียนจบประโยคคำสั่ง

เครื่องหมาย "- -" ใช้ระบุข้อความที่เป็นหมายเหตุ (Comment)

ตัวอย่าง Entity ที่เขียนอยู่ในรูปที่ 2-3 มีความหมายว่าเป็น Entity ที่ชื่อว่า Fulladd ไม่มีส่วนของ GENERIC มีเฉพาะที่เป็นพอร์ท และพอร์ทประกอบด้วยพอร์ท a b และ cin เป็นพอร์ทอินพุท ส่วนพอร์ท sum และ cout เป็นพอร์ทเอาท์พุท พอร์ททั้งหมดนี้เป็นแบบ BIT

ลักษณะการเขียนพอร์ทสามารถเขียนเป็นกลุ่มๆได้ถ้ามีลักษณะเหมือนกันเช่น

```
entity fulladd is
        port (a, b, cin : in bit;
        sum, cout : out  bit);
end fulladd;
```

หรือจะเขียนแยกกันก็ได้เช่น

```
entity fulladd is
        port (   a : in bit;
                 b : in bit;
                 cin : in bit;
        sum : out  bit;
                 cout : out  bit);
end fulladd;
```

พอร์ท a เป็นพอร์ทอินพุทชนิด BIT ส่วนพอร์ท sum เป็นพอร์ทเอาท์พุทชนิด BIT เช่นกัน

ส่วน Entity สำหรับ Testbench ก็เขียนได้ดังนี้

```
entity test_fulladd is
end test_fulladd;
```

2.2.2 อาร์คิเทคเจอร์ (Architecture)

เป็นส่วนที่ใช้สำหรับบรรยายถึงฟังก์ชันการทำงานของโมดูล โดยมีความสัมพันธ์กับ Entity ที่ได้กำหนดไว้ รูปแบบการเขียนเป็นดังนี้

```
ARCHITECTURE arch_name OF entity_name IS
        architecture declarative part
BEGIN
        statements
END arch_name ;
```

ความหมายของแต่ละส่วนเป็นดังนี้

ARCHITECTURE ..... OF ..... IS ..... BEGIN ..... END .....; ใช้ระบุส่วนที่เป็น Architecture

arch_name เป็นชื่อของอาร์คิเทคเจอร์ การตั้งชื่อใช้กฎเกณฑ์เดียวกับการตั้งชื่อออบเจ็กต์

entity_name เป็นชื่อของ Entity ของอาร์คิเทคเจอร์นี้ประกอบอยู่

architecture declarative part เป็นส่วนที่ใช้ประกาศออบเจ็กต์หรือสิ่งต่างๆที่จะนำไปใช้ในโมดูล ซึ่งมีอยู่หลายอย่างได้แก่

- Signal
- Constant
- Type
- Subtype
- Subprogram
- Component

รายละเอียดของแต่ละส่วนนี้จะนำมากล่าวในภายหลัง

Statements เป็นคำสั่งของภาษา VHDL เป็นคำสั่งที่ใช้ทำงานหรือเป็นคำสั่งที่บ่งบอกการเชื่อมต่อของอุปกรณ์ต่างๆเข้าด้วยกัน คำสั่งเหล่านี้มีหลายแบบดังนี้

- **คำสั่งแบบขนาน (Concurrent Statements)** เป็นคำสั่งที่มีการทำงานเป็นแบบขนาน (Concurrent) ภายใน อาร์คิเทคเจอร์ นี้ทุกคำสั่งถือว่าทำงานไปพร้อมๆกันคือเป็นแบบขนาน
- **คำสั่งแบบลำดับ (Sequential Statement)** เป็นคำสั่งที่มีการทำงานเป็นแบบลำดับ แต่การใช้คำสั่งแบบลำดับนี้ต้องอยู่ภายในคำสั่ง PROCESS ดังนี้

```
PROCESS(.....)
BEGIN
        คำสั่งแบบลำดับ (Sequential Statement)
END PROCESS;
```

โดย VHDL ถือว่าหนึ่ง PROCESS คือคำสั่งแบบขนานหนึ่งคำสั่ง

- **การกำหนดค่าสัญญาณ** เป็นการเชื่อมต่อสัญญาณจากสัญญาณหนึ่งส่งให้อีกสัญญาณหนึ่ง
- **การเชื่อมต่ออุปกรณ์** เป็นการต่ออุปกรณ์ต่างๆเข้าด้วยกัน

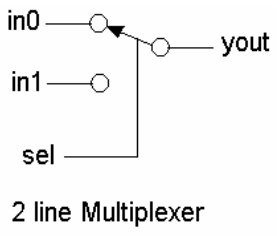
ตัวอย่างการเขียนอาร์คิเทคเจอร์ของโลจิกไดอะแกรมในรูปที่ 2-3 เป็นดังนี้

```
ARCHITECTURE addflow OF fulladd IS
BEGIN
        sum <= a xor b xor cin;
        cout <= (a and b) or (a and cin) or (b and cin);
END addflow;
```

รูปแบบการเขียนอาร์คิเทคเจอร์สามารถเขียนได้หลายรูปแบบดังนี้

แบบอธิบายพฤติกรรม (Behavioral Description)

เป็นรูปแบบการเขียนระดับสูง คล้ายกับการโปรแกรมในภาษาคอมพิวเตอร์ทั่วไป ตัวอย่างตามรูปที่ 2-4 เป็นการเขียนวงจร 2-line Multiplexer แบบ อธิบายพฤติกรรม

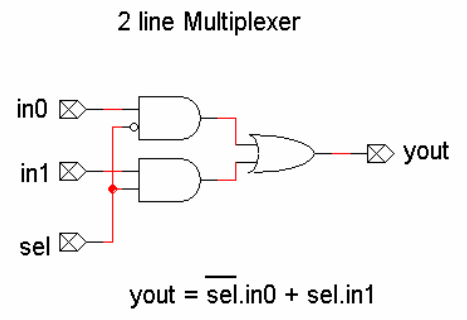



```
ARCHITECTURE behavioral OF Mux2x1 IS
BEGIN
  PROCESS (in0, in1, sel)
        BEGIN
        IF sel = 0 THEN
                yout <= in0;
            ELSE
                yout <= in1;
            END IF;
        END PROCESS;
END behavioral ;
```

รูปที่ 2-4 การเขียนโมดูล VHDL สำหรับวงจร 2-line Multiplexer แบบอธิบายพฤติกรรม

แบบอธิบายการไหลของข้อมูล (Dataflow Description)

คล้ายกับสมการโลจิก ตามตัวอย่างในรูปที่ 2-5 และเขียนเป็นภาษา VHDL ได้ดังนี้

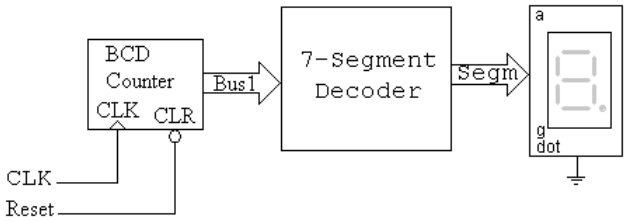



```
ARCHITECTURE dataflow OF Mux2x1 IS
BEGIN
        yout <= ((NOT sel) AND in0) OR (sel AND in1);
END dataflow;
```

รูปที่ 2-5 การเขียนโมดูล VHDL สำหรับวงจร 2-line Multiplexer แบบอธิบายการไหลของข้อมูล

แบบอธิบายโครงสร้าง (Structural Description)

แบบนี้คล้ายกับการนำอุปกรณ์โลจิกมาต่อกันให้ทำงาน เช่นตัวอย่างวงจรนับ 0 ถึง 9 ตามรูปที่ 2-6



รูปที่ 2-6 โลจิกไดอะแกรมวงจรนับ 0 ถึง 9

ตามโลจิกไดอะแกรมในรูปที่ 2-6 มีอุปกรณ์อยู่ 2 ส่วนคือ วงจรนับเลขบีซีดี และวงจรถอดรหัสบีซีดีเป็นแอลอีดี 7 ส่วน ต่อกันตามรูป ซึ่งสามารถเขียนเป็นภาษา VHDL ได้ดังนี้

```
architecture Behavioral of BCD7Segment4Digit is

component BCDCounter is
    Port ( clk : in  STD_LOGIC;
           clr : in  STD_LOGIC;
           y : inout  std_logic_vector (3 downto 0));
end component;
component BCD7Segment  is
   Port ( x : in  STD_LOGIC_VECTOR (3 downto 0);
          y : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

signal bus1 : std_logic_vector (3 downto 0);
signal clk1 : std_logic;

begin
c1: BCDCounter port map(clk,reset,bus1);
          c2: BCD7Segment port map(bus1,segm);
end Behavioral;
```

รูปที่ 2-7 การเขียนโมดูล VHDL สำหรับวงจรนับเลขบีซีดีแบบอธิบายโครงสร้าง

ในตัวอย่างคำว่า component BCDCounter และ component BCD7Segment หมายถึงโมดูลของวงจรนับแบบบีซีดี และโมดูลแปลงรหัสบีซีดีเป็นรหัสของแอลอีดี 7 ส่วน ที่มีการสร้างกันไว้ก่อนแล้ว ส่วนการต่ออุปกรณ์ต่างๆอยู่ในบรรทัดที่มีคำสั่ง PORT MAP (รายละเอียดการใช้ Component อยู่ในหัวข้อที่ 2.4)

แบบผสม (Mixed Model Description)

เป็นการเขียนโดยใช้หลายๆแบบผสมกัน

นอกจากจะสามารถเขียน Architecture ได้หลายๆแบบแล้ว ใน Entity หนึ่งๆยังสามารถมีได้หลายๆ อาร์คิเทคเจอร์ด้วย เมื่อเวลาเรียกใช้ก็สามารถระบุได้ว่าต้องการใช้กับ Architecture ใด

2.2.3 แพ็คเกจ (Package)

ใช้สำหรับเก็บข้อมูลต่างๆที่เป็นประโยชน์ต่อการเขียนรูปแบบบรรยายระบบดิจิตอล เช่น

- โปรแกรมย่อย(Subprogram(
- Type
- Constants
- Signal
- Aliases
- Attributes
- รูปแบบจำลอง (Model (มาตรฐานต่างๆ เช่น อุปกรณ์มาตรฐานของไอซีตระกูล 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 ส่วนต่างๆในPackage Body ต้องมีการกำหนดชื่อไว้ที่ส่วนนี้ ถ้าไม่กำหนด ภายนอกจะไม่สามารถเรียกใช้ได้ แต่ Package อาจจะมีอยู่ใน Package Declaration เพียงอย่างเดียวก็ได้ เช่น Type หรือ Signal ซึ่งภายนอกก็ยังคงเรียกใช้ได้ ในทำนองเดียวกัน Package อาจจะมีเพียง Package body เพียงอย่างเดียวก็ได้ แต่จะไม่สามารถเรียกใช้ได้จากรูปแบบอื่นๆ การเขียน Package declaration มีรูปแบบดังนี้

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

Package_declarative_part หมายถึงส่วนที่ใช้ประกาศต่างๆเช่น

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

ตัวอย่างเช่น การประกาศ TYPE, CONSTANT, COMPONENT และ SIGNAL

```
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 การเขียน Package body มีรูปแบบดังนี้

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

ชื่อของ package_name ที่ใช้ใน package body จะต้องเป็นชื่อเดียวกับชื่อที่กำหนดไว้ใน package declaration ตัวอย่างการเขียน package โดยการกำหนดเป็นโปรแกรมย่อยประเภท FUNCTION

```
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  +b2/(;
        END mean;
END func_avr;
```

2.2.4 คอนฟิคกูเรชั่น (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 XOR ที่มี 2 architecture

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

เมื่อต้อการระบุให้ใช้ architecture beh ให้ใช้ configuration ดังนี้

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

ถ้าไม่มีการระบุด้วย configuration ไว้ Simulator จะใช้ architecturt หน่วยสุดท้ายเป็นแบบจำลอง องค์ประกอบต่างๆที่กล่าวมานี้สามารถสรุปได้ดังนี้

- Entity, architecture และ configuration ประกอบกันเป็นรูปร่างของวงจร
- Package design unit ใช้เป็นที่รวมของฟังก์ชันต่างๆ ที่สามารถเรียกใช้จากแบบจำลองต่างๆได้

2.3 ไลบรารี (Libraries)

เป็นที่เก็บอุปกรณ์ต่างๆ หรือแบบจำลอง ต่างๆ ที่สามารถนำมาใช้งานได้ ในมาตรฐาน IEEE 1076 ได้กำหนดกฎเกณฑ์ของไลบรารี ไว้ว่า ส่วนที่เป็น design library ของ VHDL สามารถนำไปใช้ได้โดยอ้างชื่อของ library นั้นๆ ชื่อของไลบรารี นี้เรียกว่า ชื่อสัญลักษณ์ (Symbolic name) ข้อมูลภายในไลบรารี แบ่งออกเป็น

หน่วยหลัก (Primary units)

- entity declarations
- package declarations
- confiuration specifications

หน่วยรอง (Secondary units)

- architecture bodies
- package bodies

การวิเคราะห์หรือการแปลของภาษา VHDL จะทำหน่วยหลักก่อนแล้วจึงทำหน่วยรอง หลังจากแปลแล้วทั้งหน่วยหลักและหน่วยรองจะถูกเก็บอยู่ใน library เดียวกัน สามารถจำแนกไลบรารีได้เป็น 3 ประเภทได้แก่

- **STANDARD library**

  เป็นไลบรารีมาตรฐาน ใช้ชื่อว่า ***STD*** ภายในไลบรารีประกอบด้วย 2 package คือ
  - **package STANDARD** กำหนดโดย IEEE 1076 ภายในประกอบด้วยการประกาศ ต่างๆ อาทิเช่น VHDL type (REAL, INTEGER, TIME,BIT, BOOLEAN)
  - **package TEXTIO** ภายในประกอบด้วยโปรแกรมย่อยต่างๆ ที่ใช้สำหรับแก้ไขดัดแปลงข้อมูลที่เขียนในรูปรหัส ASCII

- **Working library**

  หมายถึงไลบรารี ที่อยู่ใน working directory ที่กำลังทำงานอยู่ ชื่อไลบรารีนี้จะถูกกำหนดให้ชื่อ ***WORK*** เสมอ model VHDL source file ที่เขียนขึ้นจะถูกแปลแล้วเก็บไว้ในไลบรารีนี้

- **Resource library**

  ทำหน้าที่เป็นที่เก็บข้อมูลเพิ่มเติม สามารถตั้งชื่ออะไรก็ได้ แต่อย่าให้ซ้ำกับชื่อไลบรารีสงวนของ VHDL คือ ***STD*** และ ***WORK***

Visibility

หมายถึงการทำให้หน่วยอื่นๆมองเห็นข้อมูลที่ต้องการเรียกใช้ เช่นการที่จะใช้ข้อมู ลภายใน 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]
```

เช่น

```
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 สำหรับอีกตัวอย่างหนึ่งแสดงให้เห็นการใช้ package และการทำให้มองเห็น package

```
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;
```

2.4 คอมโปเนนต์ (Component)

เมื่อต้องการเรียกใช้อุปกรณ์ที่มีอยู่แล้ว และอุปกรณ์นั้นเขียนอยู่ในรูปของ VHDL จะใช้คำสั่งเกี่ยวกับคอมโปเนนต์ การใช้งานคอมโปเนนต์ มี กใช้คอมขั้นตอน คือ การประกาศคอมโปเนนต์ และการเรีย 2 โปเนนต์

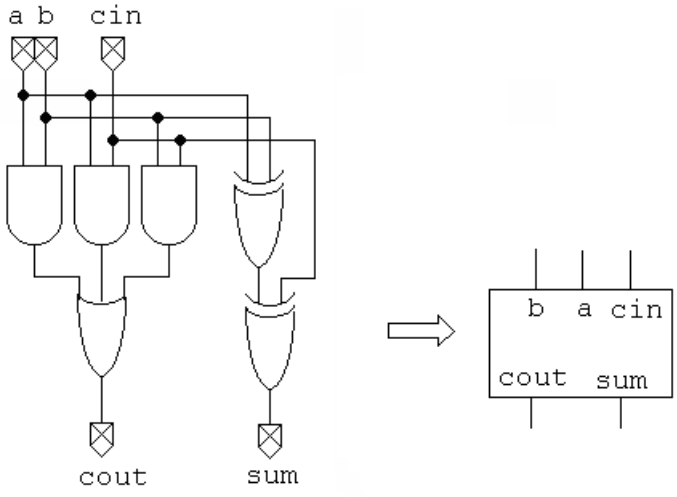
2.4.1 การประกาศคอมโปเนนต์

เป็นการประกาศว่าจะมีคอมโปเนนต์อะไรบ้างที่จะนำมาใช้งาน การประกาศนี้จะทำอยู่ในส่วนของ Architecture โดยอยู่ก่อน คำสัง Begin ของ Architecture และรูปแบบการประกาศเป็นดังนี้

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

จะเห็นได้ว่ารูปแบบการประกาศเหมือนกับการเขียน ENTITY เพียงแต่ใช้คำว่า COMPONENT แทนคำว่า ENTITY และไม่มีคำว่า IS ส่วน Component_name ก็เอามาจากชื่อ Entity นั้นเอง เช่น ถ้าต้องการนำวงจร Full adder จากรูปที่ 2-3 มาทำเป็นคอมโปเนนต์ก็สามารถประกาศได้ดังนี้

```
COMPONENT fulladd
  port (a, b, cin : in bit;
        sum, cout : out  bit);
END COMPONENT ;
```



รูปที่ 2-8 เมื่อมองวงจร Fulladd เป็นคอมโปเนนต์

2.4.2 การเรียกใช้คอมโปเนนต์

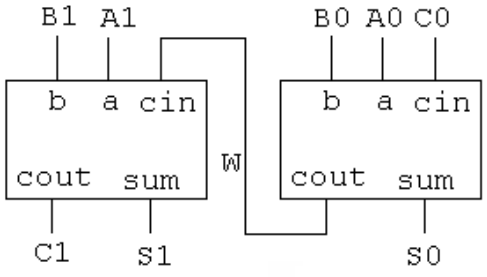
เมื่อได้ประกาศแล้ว คอมโปเนนต์ก็สามารถนำมาใช้งานได้ การใช้งานก็เหมือนกับนำอุปกรณ์อิเล็กทรอนิกส์มาต่อกัน คือนำเอาสัญญาณจากอุปกรณ์ตัวหนึ่งไปต่อเข้ากับสัญญาณอินพุทของอุปกรณ์อีกตัวหนึ่ง การเชื่อมต่อกันของคอมโปเนนต์มี 2 รูปแบบดังนี้

แบบอ้างอิงตำแหน่ง (Port association list)

แบบนี้ให้เขียนสัญญาณที่ต้องการต่อกับพอร์ทตรงลำดับของพอร์ท ตามรูปแบบดังนี้

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

เช่นถ้าต้องการต่อคอมโปเนนต์ Fulladd เข้ากับสัญญาณต่างๆดังรูปที่ 2-8



รูปที่ 2-9 การต่อคอมโปเนนต์ Fulladd 2 ตัวเข้ากับสัญญาณ

```
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;
```

ต่อสัญญาณตรงตามตำแหน่งที่ต้องการ

แบบอ้างอิงชื่อ (Name association list)

แบบนี้ให้จะเขียนสัญญาณเข้ากับพอร์ท ตามรูปแบบดังนี้

```
PORT MAP (pin_name => signal_name
          {, pin_name => signal_name});
```

ตามตัวอย่างในรูปที่ 2-9 สามารถเขียนได้ดังนี้ (แสดงไว้เฉพาะส่วนของ PORT MAP

```
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;
```

2.5 การระบุจำนวน อักขระและสตริง

ถ้าพิจารณาว่าการเขียนภาษา VHDL ก็เหมือนกับการเขียนโปรแกรมทั่วๆไป ดังนั้นการเขียนถึงข้อมูลไม่ว่าจะเป็นตัวเลขหรือตัวอักษรก็ต้องมีกฎเกณฑ์ที่ชัดเจนดังจะได้กล่าวถึงต่อไปนี้

2.5.1 การระบุจำนวน (Numbers)

สามารถจัดเลขหรือจำนวนได้เป็น 2 แบบคือเลขจำนวนจริงกับเลขจำนวนเต็ม การเขียนเลขทั้งสองแบบนี้ใช้เครื่องหมายจุด “.” เป็นตัวบอก ถ้าเป็นเลขจำนวนจริงจะต้องมีจุดแล้วตามด้วยตัวเลข โดยเลขหลังจุดบอกค่าเศษ ส่วนการเขียนเลขจำนวนเต็มไม่ต้องมีจุด นอกจากนี้เพื่อความง่ายในการอ่านอาจใช้เครื่องหมายขีดเส้นใต้เพื่อแบ่งตัวเลขออกเป็นชุดๆ ชุดละ 3 ตัวดังตัวอย่างต่อไปนี้

ตัวอย่างการเขียนเลขฐานสิบ

```
0     1       123_456_789      987E6       --เลขจำนวนเต็ม
0.0   0.5     2.718_28         12.4E-9     --เลขจำนวนจริง
```

ตัวอย่างการเขียนเลขฐานอื่นๆ

```
2#1100_0100#                 -- เลขจำนวนเต็มฐานสอง
2#1.1111_1111_111#E+11       -- เลขจำนวนจริงฐานสอง
16#C4#                       -- เลขจำนวนเต็มฐานสิบหก
16#F.FF#E2                   -- เลขจำนวนจริงฐานสิบหก
```

2.5.2 อักขระ (Characters)

หมายถึงตัวอักษรตามรหัส ASCII เพียงตัวเดียว การเขียนอักขระใช้เครื่องหมาย ‘ ’ เช่น ‘V’ ‘H’ ‘D’ ‘L’ ‘#’ และถ้าตัวอักขระว่างให้เขียนดังนี้ ‘’

2.5.3 สตริง (Strings)

สตริงหมายถึงอักขระหลายๆตัวที่เขียนติดกันรวมถึงอักขระเว้นวรรคด้วย การเขียนสตริงใช้เครื่องหมาย “ ” เช่น

“VHDL” “This is a string” และถ้าเป็นสตริงว่างให้เขียนดังนี้ “”

2.5.4 บิตสตริง (Bit Strings)

หมยถึงกลุ่มของเลขฐานใดๆแต่ต้องกาให้พิจารณาเป็นบิตๆ การเขียนบิตสตริงกับเลขฐานสอง ฐานแปด ฐานสิบหก ให้ใช้ตัวอักษร B O และ X นำหน้าตามลำดับ เช่นตัวอย่างดังต่อไปนี้

```
B"1010110"      --เลขฐานสองมีความยาว 7 บิต
O"126"          --เลขฐานแปด เทียบได้กับเลขฐานสอง          B"001_010_110
X"56"           --เลขฐานสิบหก เทียบได้กับเลขฐานสอง  B"0101_0110"
```

2.6 ออบเจ็กต์ประเภท ค่าคงที่ สัญญาณ และ ตัวแปร

ตามที่ได้กล่าวมาบ้างแล้วในหัวข้อที่ 2.1.2 เกี่ยวกับออบเจ็กต์ สำหรับในหัวข้อนี้จะกล่าวถึงการประกาศค่าโดยละเอียด พร้อมทั้งเปรียบเทียบผลของการทำงานระหว่าสัญญาณกับตัวแปรรวมถึงเมื่อนำไปสังเคราะห์เป็นของจริงจะได้รับผลเช่นไรโครงสร้างของภาษา VHDL ประกอบด้วยส่วนสำคัญ 2 ส่วนคือ Entity และ Architecture

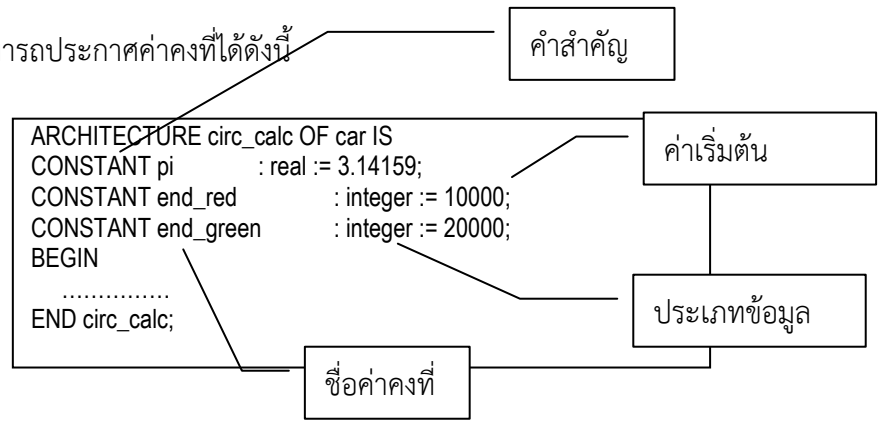
2.6.1 ค่าคงที่ (Constant)

ค่าค่งที่เป็นออบเจ็กต์ที่มีค่าได้เพียงค่าเดียวตลอดทั้งโปรแกรม การประกาศใช้ค่าคงที่ต้องประกาศไว้ใน Architecture และอยู่ก่อน Begin ของ Architecture นั้นๆ

จากรูปแบบการประกาศค่าออบเจ็กต์

```
OBJECT_CLASS object_name : TYPE [:= initial_value];
```

สามารถประกาศค่าคงที่ได้ดังนี้



2.6.2 สัญญาณ (Signal) และ ตัวแปร (Variable)

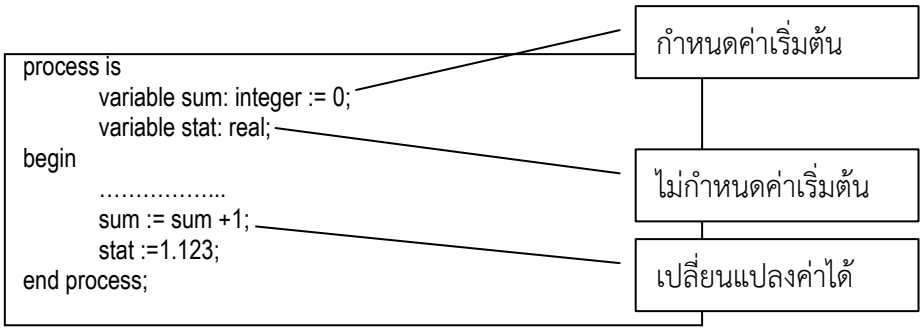
ทั้งสัญญาณและตัวแปรใช้สำหรับส่งผ่านข้อมูลระหว่างอุปกรณ์ แต่สัญญาณจะคล้ายกับสายไฟในวงจรไฟฟ้า ส่วนตัวแปรกับค่าคงที่จะเหมือนกับตัวแปรและค่าคงที่ในโปรแกรมระดับสูงทั่วไป การจำลองการทำงานและการสังเคราะห์วงจรสำหรับสัญญาณและตัวแปรจะให้ผลที่ต่างกัน

ตัวแปร (Variable)

การประกาศและการใช้ตัวแปรมีคุณสมบัติดังนี้

- กระทำภายใน PROCESS
- ใช้ได้ภายใน PROCESS ที่ประกาศไว้เท่านั้น
- การกำหนดค่าเริ่มต้นขึ้นอยู่กับชนิดของข้อมูล
- สามารถเปลี่ยนแปลงค่าได้
- ใช้เครื่องหมาย := สำหรับรับค่า

ตัวอย่างการประกาศเป็นดังนี้

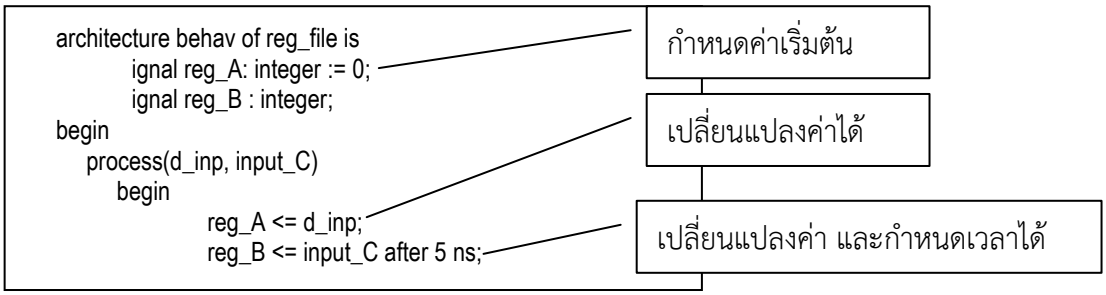


สัญญาณ (Signal)

ตำแหน่งที่ใช้สัญญาณที่สำคัญมี 2 ตำแหน่ง ตำแหน่งแรกอยู่ที่ Entity เป็นพอร์ทอินพุท-เอาท์พุท ส่วนตำแหน่งที่สองถูกกำหนดอยู่ใน Architecture ใช้แทนสายไฟสำหรับการเชื่อมต่ออุปกรณ์ต่างๆเข้าด้วยกัน การประกาศและการใช้สัญญาณมีคุณสมบัติดังนี้

- ประกาศใน Architecture แต่เหนือ Begin ของ Architecture นั้น
- จะกำหนดค่าเริ่มต้นหรือไม่ก็ได้
- สามารถเปลี่ยนแปลงค่าได้
- สามารถกำหนดค่าเวลาการทำงานให้กับสัญญาณได้
- ใช้เครื่องหมาย <= สำหรับรับค่า

ตัวอย่างการประกาศเป็นดังนี้



อีกตัวอย่างหนึ่งแสดงการใช้งานตัวแปรกับสัญญาณร่วมกัน

```
ARCHITECTURE
……..
        SIGNAL a,b: integer;
PROCESS (a) is
        VARIABLE V1,V2:integer;
BEGIN
        V1=a-V2;
        b <= -V1;
        V2 = b+V1+3;
END PROCESS;
```

สังเกตการใช้เครื่องหมายเพื่อรับค่า

การเปรียบเทียบระหว่างสัญญาณกับตัวแปร

เพื่อให้เห็นความแตกต่างระหว่างสัญญาณกับตัวแปร จะเปรียบเทียบให้เห็นได้ดังนี้

- ขอบเขตการใช้งาน

  ตัวแปรมีการทำงานอยู่ภายใน Process

  สัญญาณมีการทำงานอยู่ภายในโมดูลที่ออกแบบ

- การจำลองการทำงาน

  การกำหนดค่าให้กับตัวแปรจะมีผลทันที

  การกำหนดค่าให้กับสัญญาณจะมีผลในรอบการทำงานถัดไป

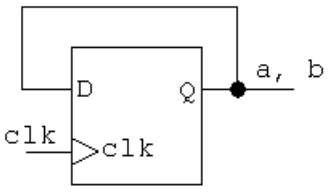
  ถ้าสัญญาณเกิดการเปลี่ยนแปลงค่านั้นจะเกิดเมื่อสิ้นสุด Process

- การสังเคราะห์ (Synthesis) ภายใต้คำสั่งเกี่ยวกับสัญญาณนาฬิกา (clock)

  สัญญาณถ้าอยู่ จะสังเคราะห์เป็นอุปกรณ์ความจำ (memory) ตามรูปที่ 2-10 (ก)

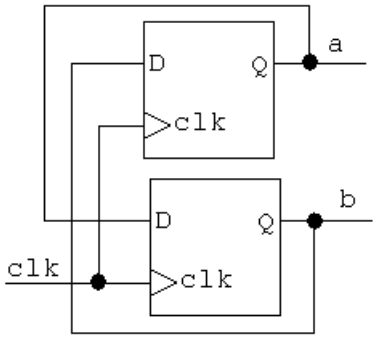
  ตัวแปรสังเคราะห์เป็นสายไฟใน netlist ตามรูปที่ 2-10 (ข)

```
PROCESS (clk)
  VARIBLE a,b: bit;
BEGIN
  IF(clk'EVENT and clk = '1') THEN
        a:= b;
        b:= a;
  END IF;
END PROCESS;
```



รูปที่ 2-10 (ก) ผลการสังเคราะห์ของตัวแปร

```
SIGNAL a, b : bit;
PROCESS (clk)
BEGIN
   IF(clk'event and clk = '1') then
      a <= b;
      b <= a;
   END IF;
END PROCESS;
```



รูปที่ 2-10 (ข) ผลการสังเคราะห์ของสัญญาณ

2.7 ประเภทข้อมูล

จากที่ได้เคยกล่าวไว้แล้วว่าออบเจ็กต์ต่างๆภายใน VHDL ต้องมีการระบุชนิดของข้อมูลของออบเจ็กต์นั้น ทั้งนี้เพราะว่าออบเจ็กต์ที่เป็นประเภทเดียวกันเท่านั้นที่จะสามารถส่งถ่ายข้อมูลด้วยกันได้ หรือถ้าจะมองในแง่การเชื่อมต่อกัน ออบเจ็กต์ต่างชนิดกันไม่สามารถนำมาเชื่อมกันได้ สามารถจำแนกประเภทข้อมูลเป็นกลุ่มใหญ่ๆได้ดังนี้

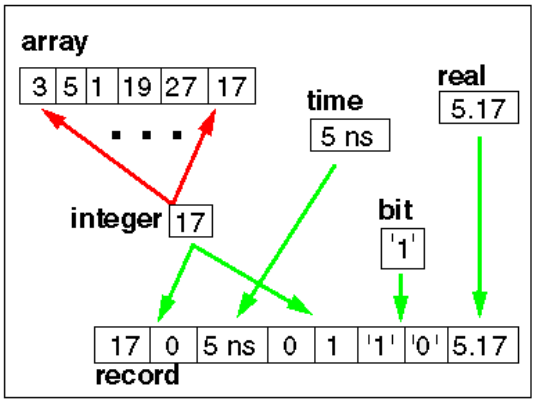
Scalar types:

เป็นข้อมูลที่มีค่าที่แน่นอนค่าหนึ่ง เช่น integer, real, bit, enumerated, และ กายภาพ (physical)

Composite types:

เป็นข้อมูลที่อาจประกอบด้วยค่าหลายค่า เช่น array และ record

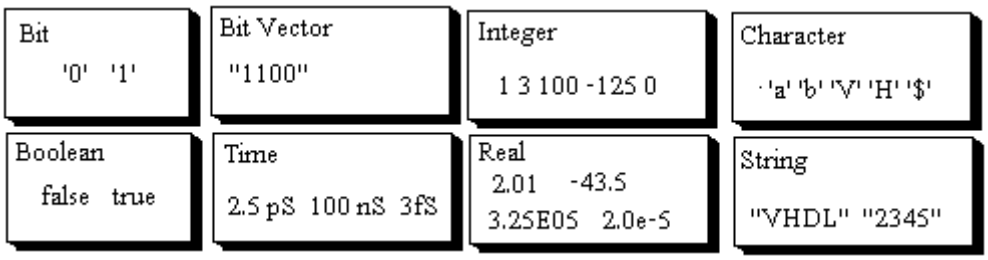
และ Types แบบอื่นๆ: เช่น file access



รูปที่ 2-11 ลักษณะข้อมูลประเภทต่างๆ

2.8 ข้อมูลประเภทมาตรฐาน

ในภาษา VHDL มีข้อมูลอยู่ชุดหนึ่งเรียกว่าข้อมูลประเภทมาตรฐาน เก็บอยู่ในแพกเกจที่เรียกว่า Package STANDARD ข้อมูลประเภทนี้เวลาเรียกใช้ไม่ต้องเขียนลงไปในโมดูลที่ออกแบบ เพราะว่าตัวภาษารู้จักอยู่แล้ว ข้อมูลชุดนี้ได้แก่ BIT BIT_VECTOR TIME INTEGER REAL CHARACTER STRING และ BOOLEAN ข้อมูลแต่ละประเภทมีค่าดังนี้



รูปที่ 2-12 ข้อมูลประเภทมาตรฐาน

BIT มีค่าเป็น 0 หรือ 1 ค่าเริ่มต้นกำหนดไว้ให้เป็น 0

BIT_VECTOR เป็นกลุ่มหรืออะเรย์ของบิต

CHARACTER เป็นตัวอักษร 1 ตัวมีค่าตามตาราง ASCII

STRING เป็นกลุ่มหรืออะเรย์ของ CHARACTER

BOOLEAN มีค่าเป็น จริง(TRUE) กับเท็จ(FALSE)

INTEGER เป็นเลขจำนวนเต็ม มีค่าตามขอบเขตที่กำหนดเช่น 0 ถึง 255

REAL เป็นจำนวนจริง มีค่าตามขอบเขตที่กำหนดมักใช้กับตัวแปร

TIME เป็นข้อมูลประเภทเวลา

2.9 ข้อมูลประเภทเวลา

ข้อมูลประเภทนี้มักใช้งานเป็นเวลาหน่วงของอุปกรณ์ หรือใช้ใน Testbench เพื่อการทดสอบการทำงาน การคูณหรือการหารเวลา จะได้เป็นเวลาเช่นเดิม สามารถกำหนดหน่วยของเวลาได้ตั้งแต่ fs(femtosecond) ps(pico) ns(nano) us(micro) ms(milli) sec(second) min(minute) และ hr(hour) ตัวอย่างการใช้งาน TIME มีดังนี้

```
ARCHITECTURE example OF time_type IS
        SIGNAL clk : BIT;
        CONSTANT priod : TIME := 50 ns;
BEGIN
        y1 <=  (a AND b) after 10 ns;
        clk <= NOT clk after period/2;
END example;
```

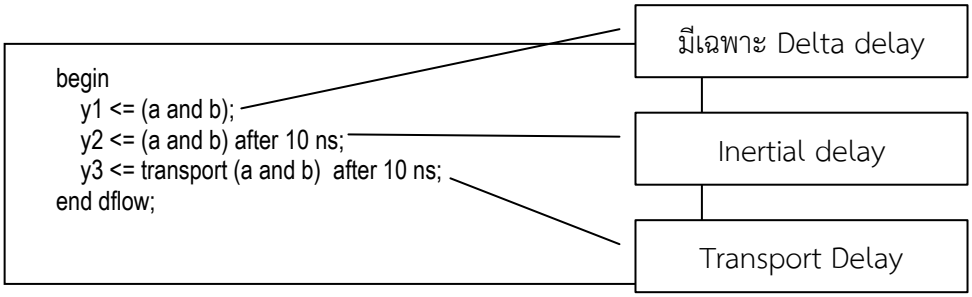
สำหรับเวลาหน่วง (Delay Model) ใน VHDL มีอยู่ด้วยกัน 3 แบบคือ

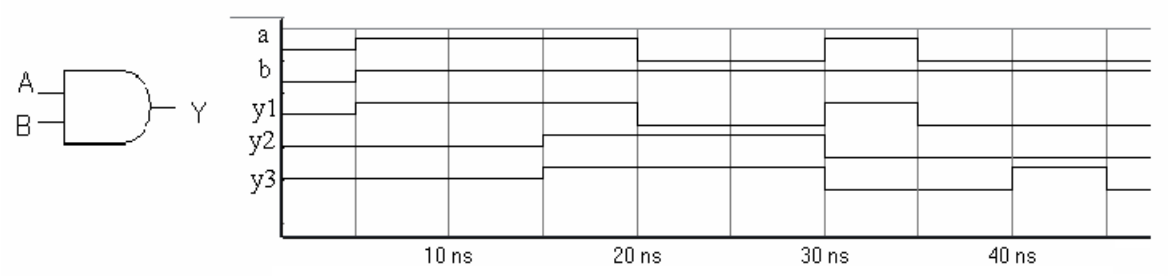
Inertial delay ใช้เป็นค่าหน่วงเวลาของโมเดลในภาษา VHDL สัญญาณ Spikes จะไม่ถูก propagate ถ้าขนาดเล็กกว่า ค่าเวลา ที่ระบุไว้ใน after ใช้เป็นค่า propagation delay time ของอุปกรณ์

Transport Delay เป็นค่าหน่วงเวลาของสัญญาณอินพุทที่จะออกไปยังเอาท์พุททุกค่า

Delta delay เป็นค่าหน่วงเวลาของการทำงานของอุปกรณ์ ในกรณีที่ไม่ได้กำหนด inertial delay

ตัวอย่างที่แสดงให้เห็นถึงความแตกต่างของเวลาหน่วง





รูปที่ 2-13 ค่าเวลาหน่วงแบบต่างๆ

สำหรับความหมายของ Delta Delay ขอให้พิจารณาคำสั่ง Y <= A AND B เมื่อ A และ B มีค่าเป็น 1 และแล้ว A เปลี่ยนจาก 1 ไปเป็น 0 การเปลี่ยนแปลงของ Y เมื่อพิจารณาตามระยะเวลาของ Delta Delay จะมีผลดังนี้

เวลา	A	B	Y	การทำงาน
0	1	1	1	เวลาที่ 0 เมื่อเวลาผ่านมานานแล้ว เอาท์พุทอยู่ในสถานะ Stable
1	0	1	1	เมื่อเวลาผ่านไปที่ 1 อินพุท A เปลี่ยนเป็น 0 ขณะนี้เอาท์พุทยังไม่เปลี่ยน
2	0	1	0	เมื่อเวลาผ่านไปอีก 1 Delta delay อินพุท A ยังเป็น 0 เหมือนเดิม เอาท์พุทเปลี่ยนเป็น 0

2.10 ข้อมูลประเภทอะเรย์

ข้อมูลประเภทอะเรย์ เป็นกลุ่มของข้อมูลประเภท Scalar ตัวอย่างอะเรย์ได้แก่

bit_vector เป็นอะเรย์ของ bit

string เป็นอะเรย์ของ character

std_logic_vector เป็นอะเรย์ของ std_logic

std_ulogic_vector เป็นอะเรย์ของ std_ulogic

(ข้อมูล 2 แบบหลังนี้ เป็นประเภทข้อมูลที่อยู่ใน Package IEEE.std_logic_1164)

นอกจากอะเรย์ที่กำหนดมาในแพกเก็จแล้ว VHDL ยังให้สร้างอะเรย์ใหม่ได้ด้วยคำสั่ง TYPE

```
TYPE type_name IS ARRAY (RANGE) OF element_type;
```

ความหมาย

TYPE..... IS ARRAY .....OF.....; เป็นคำสั่งให้สร้างประเภทข้อมูลอะเรย์ใหม่

type_name เป็นชื่อของประเภทข้อมูล

RANG เป็นขนาดของอะเรย์

Element_type เป็นประเภทข้อมูลของส่วนประกอบอะเรย์นี้

เช่น

```
type MY_BYTE is array (7 downto 0) of STD_ULOGIC;

signal TYPE_BUS : MY_BYTE;
```

ตัวอย่างนี้เป็นการกำหนดประเภทข้อมูลอะเรย์ขึ้นใหม่ชื่อว่า MY_BYTE โดยมีขนาด 8 ตำแหน่ง ส่วน TYPE เป็นสัญญาณที่มีชนิดเป็นอะเรย์แบบ MY_BYTE เพื่อให้เข้าใจถึงการสร้างประเภทข้อมูลแบบใหม่ยิ่งขึ้นลองพิจารณาดูตัวอย่างนี้

```
architecture EXAMPLE of ARRAY is
    type CLOCK_DIGITS is (HOUR10,HOUR1,MINUTES10,MINUTES1);
    type  T_TIME is array (CLOCK_DIGITS) of integer  range 0 to 9;
    signal ALARM_TIME : T_TIME := (0,7,3,0);

begin
    ALARM_TIME(HOUR1)  <= 0;
    ALARM_TIME(HOUR10 to MINUTES10) <= (0,7,0);
end EXAMPLE;
```

ความหมายของแต่ละบรรทัดมีดังนี้

type *CLOCK_DIGITS* ***is*** *(HOUR10,HOUR1,MINUTES10,MINUTES1);*

เป็นการสร้างประเภทของข้อมูลแบบใหม่ชื่อว่า CLOCK_DIGITS โดยมีค่าเป็น HOUR10 HOUR1 MINUTES10 MINUTES1

type *T_TIME* ***is array*** *(CLOCK_DIGITS)* ***of*** *integer range 0 to 9;*

เป็นการสร้างประเภทของข้อมูลแบบอะเรย์แบบใหม่ชื่อว่า T_TIME ซึ่งมีค่าเป็น CLOCK_DIGITS และมีค่าของสมาชิกเป็นเลขจำนวนเต็มตั้งแต่ 0 ถึง 9

signal *ALARM_TIME : T_TIME := (0,7,3,0);*

เป็นการกำหนดสัญญาณที่ชื่อว่า ALARM_TIME เป็นประเภท T_TIME และให้มีค่าดังนี้

ALARM_TIME(HOUR10) = 0

ALARM_TIME(HOUR1) = 7

ALARM_TIME(MINUTES10) = 3

ALARM_TIME(MINUTES1) = 0

นอกจากนี้การกำหนดอะเรย์ยังทำได้หลายๆมิติ (Multidimensional Arrays) เช่นตัวอย่าง

```
architecture EXAMPLE of ARRAY is

    type  INTEGER_VECTOR  is array (1 to 8) of integer;

  -- 1 --
  type  MATRIX_A  is array(1 to 3)  of  INTEGER_VECTOR ;
  -- 2 --
  type  MATRIX_B  is array(1 to 4, 1 to 8)  of  integer ;

   signal MATRIX_3x8 : MATRIX_A;
   signal MATRIX_4x8 : MATIRX_B;

begin
    MATRIX_3x8(3)(5)  <= 10;  --array of array
    MATRIX_4x8(4, 5)   <= 17;  -- 2 dim array
end EXAMPLE;
```

แบบ array of array

แบบ multidimensional array

2.10.1 การกำหนดค่าข้อมูลประเภทอะเรย์

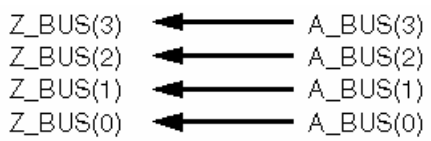
ตัวอย่างการกำหนดค่าให้กับอะเรย์

```
architecture EXAMPLE of ASSIGNMENT is

  signal Z_BUS :    bit_vector (3 downto 0);
  signal BIG_BUS : bit_vector (15 downto 0);

begin

  -- legal assignments:
  Z_BUS(3) <= `1`;
  Z_BUS     <= ``1100``;

  Z_BUS     <=  b`` 1100 ``;
  Z_BUS     <=  x`` c ``;
  Z_BUS     <=  X`` C ``;
  BIG_BUS  <=  B` `0000 _ 0001_0010_0011 ``;

end EXAMPLE;
```

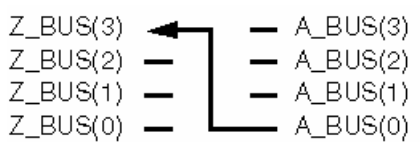
ตัวอย่างการส่งถ่ายข้อมูลระหว่างอะเรย์ โดยกำหนดสัญญาณ

SIGNAL A_BUS, Z_BUS : bit_vector(3 downto 0);

Z_BUS <= A_BUS;



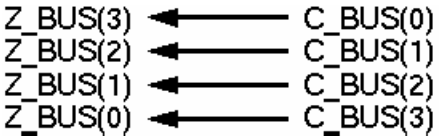
Z_BUS(3) <= A_BUS(0);



ตัวอย่างการส่งถ่ายข้อมูลระหว่างอะเรย์เมื่อกำหนดดรรชนีของอะเรย์ ไม่เหมือนกัน

```
architecture EXAMPLE of ARRAYS is
  signal Z_BUS : bit_vector (3 downto 0);
  signal C_BUS : bit_vector (0 to 3);
begin
  Z_BUS <= C_BUS;
end EXAMPLE;
```

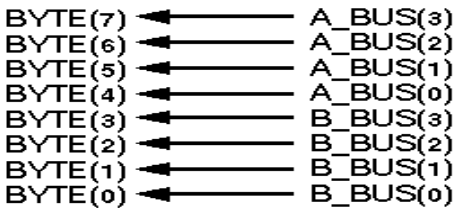
ผลลัพธ์การทำงาน



2.10.2 การรวมข้อมูลแบบ Concatenation

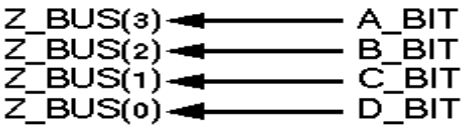
เป็นการรวมข้อมูลขนาดเล็กให้มีขนาดใหญ่ขึ้น โดยใช้เครื่องหมาย Ampersand (&) ดังตัวอย่างนี้

```
architecture EXAMPLE_1 of ONCATENATION is
  signal BYTE            : bit_vector (7 downto 0);
  signal A_BUS, B_BUS : bit_vector (3 downto 0);
begin
  BYTE  <= A_BUS & B_BUS;
end EXAMPLE;
```



ตัวอย่างการรวมข้อมูลอีกแบบ

```
architecture EXAMPLE_2 of CONCATENATION is
  signal Z_BUS : bit_vector (3 downto 0);
  signal A_BIT, B_BIT, C_BIT, D_BIT : bit;
begin

  Z_BUS <= A_BIT & B_BIT & C_BIT & D_BIT;

end EXAMPLE;
```



2.10.3 การรวมข้อมูลแบบ Aggregate และอะเรย์หลายมิติ (Multidimensional Arrays)

เป็นการกำหนดค่าให้กับกลุ่มของข้อมูลประเภทอะเรย์ โดยเขียนได้ 2 แบบ

- แบบสัมพันธ์กับตำแหน่งของอะเรย์

  (value_1,value_2);

- แบบสัมพันธ์กับชื่อของอะเรย์

  (element_1=> value_1, element_2 => value_2);

ตามตัวอย่าง

```
ARCHITECTURE rtl OF ex IS
SIGNAL A, B, C: logic_vector(4 downto 0);
SIGNAL D: logic_vector(0 to 1);
SIGNAL E: logic_vector(0 to 6);
SIGNAL F: logic_vector(0 to 7);
```

```
BEGIN
        A <= ( 2|3 => '1', other => '0');
        B <= (1 => '0', 3 => '1', other => C(1));
        C <= ('0', '1', '1', '0', '1');
        D <= '0' & A(2);
        E <= A(3 downto 0) & B(2 downto 0);
END;
```

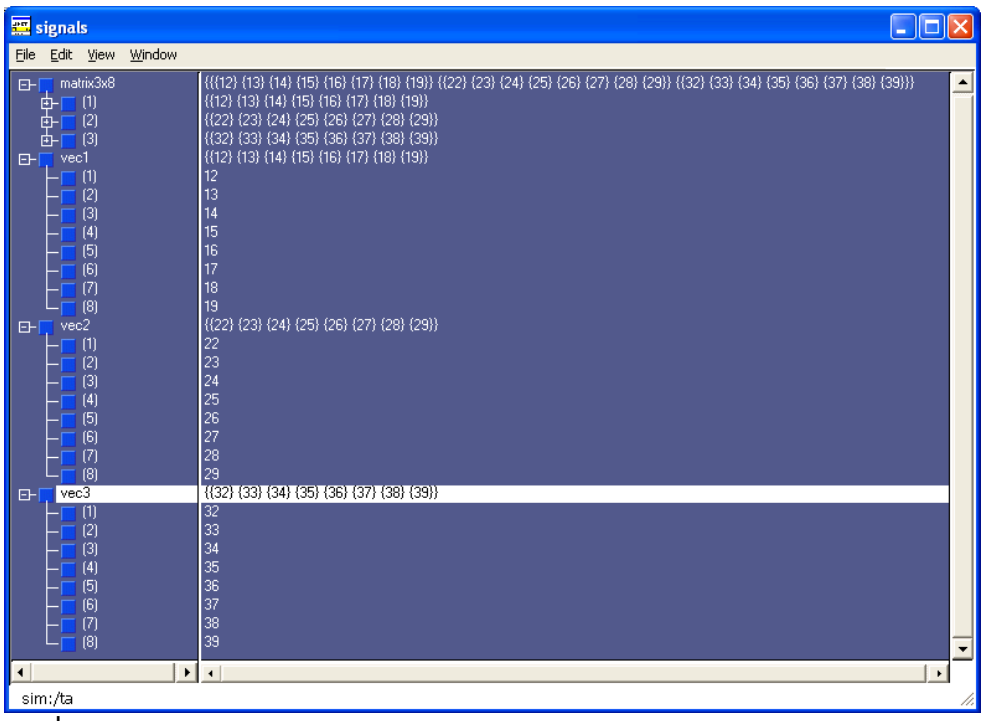
ผลการทำงานได้เป็น

A = 01100 B = 01000 C = 01101 D = 01 E = 1100000

ตัวอย่างที่สองเป็นการกำหนดค่าให้กับอะเรย์แบบ Multidimensional Arrays

```
entity ta is
end ta;
architecture tabeh of ta is
    type INTEGER_VECTOR is array (1 to 8) of integer;
    type MATRIX_A is array(1 to 3) of INTEGER_VECTOR;
       signal MATRIX3x8 : MATRIX_A;
       signal VEC1, VEC2, VEC3 : INTEGER_VECTOR;
begin
   VEC1 <= (12, 13, 14, 15, 16, 17, 18, 19);
   VEC2 <= (22, 23, 24, 25, 26, 27, 28, 29);
   VEC3 <= (32, 33, 34, 35, 36, 37, 38, 39);
   MATRIX3x8 <= (VEC1, VEC2, VEC3);
end tabeh;
```

ผลการทำงานได้ดังนี้



รูปที่ 2-14 ผลการทำงาน การกำหนดค่าให้กับอะเรย์แบบ Multidimensional Arrays

2.10.4 การแบ่งข้อมูลแบบอะเรย์ (Slices of Arrays)

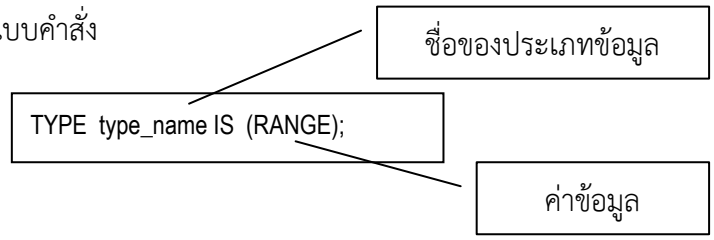
บางครั้งการใช้อะเรย์ก็ต้องการเพียงสมาชิกบางตัวของอะเรย์เท่านั้น ลักษณะนี้เรียกว่า Slide of array มีวิธีการเขียนดังตัวอย่างต่อไปนี้

```
architecture EXAMPLE of SLICES is
  signal BYTE : bit_vector (7 downto 0);
  signal A_BUS, Z_BUS : bit_vector (3 downto 0);
  signal A_BIT : bit;
begin
  BYTE (5 downto 2) <= A_BUS;
  BYTE (5 downto 0) <= A_BUS;              -- wrong

  Z_BUS (1 downto 0) <= `0` & A_BIT;
  Z_BUS              <= BYTE (6 downto 3);
  Z_BUS (0 to 1)     <= `0` & B_BIT;     -- wrong
  A_BIT  <= A_BUS (0);
end EXAMPLE;
```

2.11 ข้อมูลแบบ Enumerated (Enumeration Types)

ประเภทข้อมูลแบบนี้เป็นแบบที่ผู้ใช้กำหนดขึ้นเองเพื่อให้เหมาะสมกับงานที่กำลังทำอยู่ เช่นถ้าต้องการออกแบบระบบควบคุมไฟจราจร ถ้าต้องการความสะดวก การระบุว่าออบเจ็กต์ให้มีค่าเป็น แดงเหลือง เขียว ก็จะทำให้การออกแบบสะดวกขึ้น เพราะว่าเมื่อนำไปสังเคราะห์ VHDL จะแปลงเป็นค่าโลจิกที่เหมาะสมและครอบคลุมให้เอง

รูปแบบคำสั่ง



ตัวอย่างต่อไปนี้เป็นการกำหนดประเภทข้อมูลขึ้นใหม่เรียกว่า T_STATE มีค่าเป็น RESET START EXECUTE และ FINISH ส่วนสัญญาณที่ชื่อว่า CURRENT_STATE และ NEXT_STATE ต่างก็มีข้อมูลเป็นประเภท T_STATE ดังนั้นค่าที่จะป้อนให้สัญญาณทั้งสองนี้ต้องเป็นค่าที่ T_STATE กำหนดไว้คือ

```
architecture EXAMPLE of ENUMERATION is

   type T_STATE is (RESET, START, EXECUTE, FINISH);

  signal CURRENT_STATE, NEXT_STATE : T_STATE ;

begin

  NEXT_STATE    <= CURRENT_STATE;
  CURRENT_STATE <= RESET;

end EXAMPLE;
```

RESET START EXECUTE และ FINISH เมื่อนำไปสังเคราะห์ ค่าเหล่านี้จะถูกเปลี่ยนให้เป็นค่าโลจิกที่เหมาะสมเช่นอาจเป็น 00 01 10 และ 11 ตามลำดับ

อีกตัวอย่างเป็นลักษณะของระบบที่เป็น FSM (Finite State Machine)

```
architecture RTL of TRAFFIC_LIGHT is
  type  T_STATE is  ( INIT,RED,REDYELLOW,GREEN,YELLOW );
  signal STATE, NEXT_STATE : T_STATE;

  signal COUNTER: integer;
  constant END_RED     : integer := 10000;
  constant END_GREEN : integer := 20000;

begin
  LOGIC : process (STATE, COUNTER)
  begin
    NEXT_STATE <= STATE;
    case STATE is
        when  RED               =>
          if COUNTER = END_RED then
             NEXT_STATE <=  REDYELLOW ;
          end if;
        when  REDYELLOW    => -- statements
        when  GREEN          => -- statements
        when  YELLOW        => -- statements
        when  INIT             => -- statements
    end case;
  end process LOGIC;
end RTL;
```

2.12 ข้อมูลแบบ IEEE Standard Logic Type

จากที่ได้กล่าวมาแล้วเกี่ยวกับข้อมูลประเภทมาตรฐาน ที่เป็น BIT จะเห็นได้ว่ามีข้อจำกัดเรื่องที่สามารถให้ได้เพียง 2 ค่า คือ 0 และ 1 แต่ในความเป็นจริงแล้ว ระบบดิจิตอลบางอย่างมีค่าได้มากกว่านี้เช่ นอาจเป็น High Impedance หรือเป็น Unknown ที่ไม่ทราบค่าได้ ทาง IEEE จึงได้กำหนดแพกเก็จข้อมูลขึ้นใหม่เรียกว่า STD_LOGIC_1164 ซึ่งมีค่าได้ 9 ค่าดังนี้

```
type STD_ULOGIC is (
            ` U `,   -- uninitialized. This signal hasn't been set yet.
            ` X `,   -- strong 0 or 1 (= unknown).Impossible to determine this value/result.
            ` 0 `,   -- logic 0
            ` 1 `,   -- logic 1
            ` Z `,   -- high impedance
            ` W `,  -- weak 0 or 1 (= unknown).Weak signal, can't tell if it should be 0 or 1.
            ` L `,   -- weak 0.Weak signal that should probably go to 0.
            ` H `,   -- weak 1.Weak signal that should probably go to 1.
             ` - `,   -- don`t care);
```

จะเห็นได้ว่า STD_LOGIC_1164 นี้ก็เป็น Enumeration type เช่นกัน

ถ้าเทียบกับประเภทข้อมูลแบบมาตรฐานจะเทียบได้เป็น

BIT เทียบกับ STD_ULOGIC

และ BIT_VECTOR เทียบกับ STD_ULOGIC_VECTOR

เวลาเรียกใช้ต้องระบุไลบรารีและแพกเก็จดังนี้

```
LIBRARY      ieee;
USE    ieee.std_logic_1164.ALL;
```

2.12.1 ข้อมูลประเภท Resolved and Unresolved Types

ถึงแม้ข้อมูลประเภท STD_ULOGIC และ STD_ULOGIC_VECTOR จะแก้ปัญหาเรื่องค่าของข้อมูลได้เป็นหลายค่าแล้ว แต่การใช้งานบางกรณีก็ยังไม่สามารถกระทำได้เช่น

```
architecture EXAMPLE of ASSIGNMENT is
  signal A, B, Y: std_ulogic;
begin
  Y <= A;
  Y <= B;
end EXAMPLE;
```

จากโปรแกรมนี้จะเห็นได้ว่าถึงแม้ A B และ Y จะเป็น STD_ULOGIC ซึ่งให้ค่าได้ถึง 9 ค่า แต่การที่ Y ได้รับข้อมูลมาจาก A และ B พร้อมๆกันเช่นนี้ ค่าของ Y ก็ไม่สามารถจะบอกได้อยู่ดี ถ้าต้องการให้ Y มีค่า ต้องใช้ประเภทข้อมูลแบบที่มี Resolution Function ซึ่ง มีชื่อว่า STD_LOGIC และ STD_LOGIC_VECTOR แทน STD_ULOGIC และ STD_ULOGIC_VECTOR ตามลำดับ ลักษณะของตาราง Resolution Function แสดงอยู่ในรูปที่ 3-6 ดังนั้นเมื่อเปลี่ยนเป็น STD_LOGIC ถ้า A มีค่าเป็น 0 และ B มีค่าเป็น High impedance (Z) Y จะมีค่าเป็น 0 ตามที่ Resolve function จัดให้

```
architecture EXAMPLE of ASSIGNMENT is
  signal A, B, Y: std_logic;
begin
  Y <= A;
  Y <= B;
end EXAMPLE;
```

```
  FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
  CONSTANT  resolution_table  : std_logic_table := (
  --     ---------------------------------------------------------------------
  --     U      X      0      1      Z      W      L      H      -
  --     ---------------------------------------------------------------------
       ( 'U',   'U',   'U',   'U',   'U',   'U',   'U',   'U',   'U' ),      -- U
       ( 'U',   'X',   'X',   'X',   'X',   'X',   'X',   'X',   'X' ),     -- X
       ( 'U',   'X',   '0',   ' X ' ,   '0',   '0',   '0',   '0',   'X' ),    -- 0
       ( 'U',   'X',   'X',   '1',   '1',   '1',   '1',   '1',   'X' ),       -- 1
       ( 'U',   'X',   '0',   '1',   'Z',   'W',   'L',   'H',   'X' ),       -- Z
       ( 'U',   'X',   '0',   '1',   'W',   'W',   'W',   'W',   'X' ),      --W
       ( 'U',   'X',   '0',   '1',   'L',   'W',   'L',   'W',   'X' ),     -- L
       ( 'U',   'X',   '0',   '1',   'H',   'W',   'W',   'H',   'X' ),      -- H
       ( 'U',   'X',   'X',   'X',   'X',   'X',   'X',   'X',   'X' )      -- -  );
  VARIABLE result : std_ulogic := 'Z';  -- weakest state default
BEGIN
```

```
  IF    (s'LENGTH = 1) THEN
    RETURN s(s'LOW);
  ELSE
    FOR i IN s'RANGE LOOP
      result := resolution_table(result, s(i));
    END LOOP;
  END IF;
  RETURN result;
END resolved;
```

รูปที่ 2-15 ตัวอย่างตาราง Resolution Function

ตารางที่ 2-1 ตาราง Resolution Function สำหรับ โลจิก AND

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

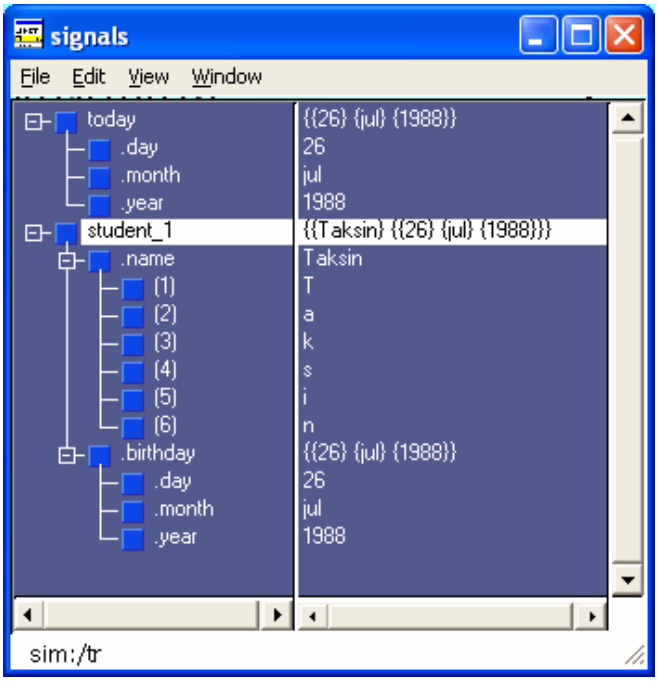
2.13 Records

เป็นชนิดของข้อมูลที่เกิดจากกลุ่มของข้อมูลหลายๆประเภทที่ต่างชนิดกัน ข้อมูลเรคอร์ดนี้จะคล้ายกับเรคอร์ดในภาษาโปรแกรมทั่วไป รูปแบบการกำหนดเรคอร์ดมีดังนี้

```
TYPE identifier IS RECORD
        record_definition
END RECORD
```

ตัวอย่าง

```
entity tr is
end tr;
architecture trbeh of tr is
   type MONTH_NAME is (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC);
   type DATE is record
           DAY: integer range 1 to 31;
           MONTH: MONTH_NAME;
           YEAR: integer range 0 to 4000;
       end record;
   type PERSON is record
         NAME: string (1 to 6);
         BIRTHDAY: DATE;
       end record;
   signal TODAY: DATE;
   signal STUDENT_1: PERSON;
begin
  TODAY     <= (26, JUL, 1988);
  STUDENT_1 <= ("Taksin", TODAY);
end trbeh;
```



รูปที่ 2-16 ผลการทำงานของ Record

2.14 Subtypes

เป็นชนิดของข้อมูลย่อยของข้อมูลหลักที่มีอยู่แล้ว ต้องเป็นชนิดเดียวกับข้อมูลหลัก รูปแบบการกำหนด

```
SUBTYPE subtype_name IS base_type RANGE range_constraint
```

ตัวอย่าง

```
architecture EXAMPLE of SUBTYPES is
        type MY_WORD is array (15 downto 0) of std_logic;
        subtype SUB_WORD is std_logic_vector (15 downto 0);

        subtype MS_BYTE is integer range 15 downto 8;
        subtype LS_BYTE is integer range 7 downto 0;

        signal VECTOR: std_logic_vector(15 downto 0);
        signal SOME_BITS: bit_vector(15 downto 0);
        signal WORD_1: MY_WORD;
        signal WORD_2: SUB_WORD;

begin
        SOME_BITS <= VECTOR; -- wrong
        SOME_BITS <= Convert_to_Bit(VECTOR);
        WORD_1 <= VECTOR; -- wrong
        WORD_1 <= MY_WORD(VECTOR);
        WORD_2 <= VECTOR; -- correct!
        WORD_2(LS_BYTE) <= "11110000";
end EXAMPLE;
```

2.15 Aliases

เป็นการกำหนดชื่อใหม่ให้กับ object ที่มีอยู่แล้ว เพื่อทำให้ง่ายต่อการจัดการ เหมาะกับ Object ที่เป็นแบบผสม ข้อควรระวัง Aliases นี้ใช้ได้กับเครื่องมือที่ใช้สังเคราะห์บางตัวเท่านั้น

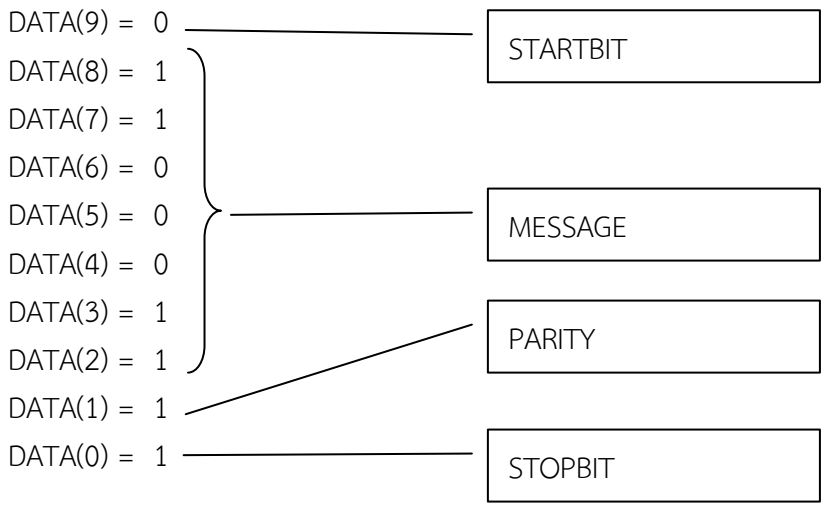
รูปแบบการใช้งาน

```
ALIAS new_name : Object
```

ตัวอย่าง

```
entity tal is
end tal;
architecture EXAMPLE of tal is
    signal DATA : bit_vector(9 downto 0);
    alias STARTBIT : bit is DATA(9) ;
    alias MESSAGE : bit_vector(6 downto 0) is DATA (8 downto 2);
    alias PARITY : bit is DATA(1);
    alias STOPBIT : bit is DATA(0);
    alias REVERSE : bit_vector(1 to 10) is DATA;
--  function calc_parity(data: bit_vector) return bit is
begin
    STARTBIT      <= '0';
    MESSAGE       <= "1100011";
--  PARITY        <= calc_parity(MESSAGE);
    PARITY    <= '1';
    REVERSE(10) <= '1';

end EXAMPLE;
```

ผลการทำงาน



แบบฝึกหัด

2.1 คำสั่งการทำงานแบบขนานต่อไปนี้จะให้ผลลัพธ์การทำงานเป็นอย่างไร ถ้า A = 1 B = 0 C = 1และ X = 1

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

2.2 คำสั่งการทำงานแบบลำดับต่อไปนี้จะให้ผลลัพธ์การทำงานเป็นอย่างไร ถ้า A = 1 B = 0 C = 1และ X = 1

```
PROCESS(A, B, C)
BEGIN
        X <= A and B;
        Y <=  X and C;
END PROCESS;
```

2.3 คำสั่ง GENERIC ใช้ทำอะไรจงอธิบายพร้อมยกตัวอย่างวิธีการใช้งาน

2.4 จงเปรียบเทียบข้อดีและข้อเสียระหว่างการเขียน ARCHITECTURE แบบอธิบายพฤติกรรมกับการเขียนแบบอธิบายการไหลของข้อมูล

2.5 จงยกตัวอย่างวิธีการใช้งาน package และ Library

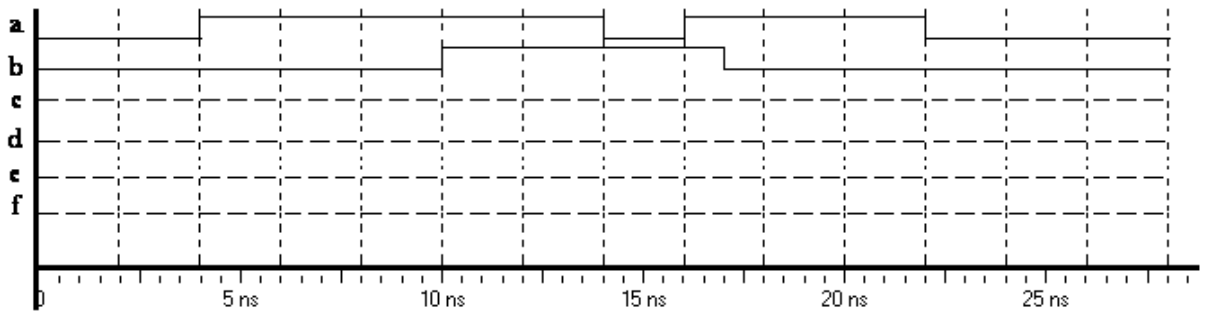
2.6 จงเขียนชนิดของข้อมูลแบบมาตรฐา (Standard Data Types) มา 5 ชนิดพร้อมจงยกตัวอย่างประกอบการอธิบาย

2.7 ข้อมูลประเภทอะเรย์(Arrays) คืออะไร มีลักษณะเป็นอย่างไร และจงยกตัวอย่างประกอบการอธิบาย

2.8 Records ในภาษา VHDL คืออะไร มีลักษณะเป็นอย่างไร และจงยกตัวอย่างประกอบการอธิบาย

2.9 จากคำสั่ง VHDL ต่อไปนี้ จงเขียนรูปคลื่นของ c d e และ f ให้สอดคล้องกับสัญญาณ a และ b ลงในรูปที่

```
begin
   c      <= (a xor b) AFTER 2 NS;
   d      <= (a and b) AFTER 2 NS;
   e      <= TRANSPORT (a xor b) AFTER 4 NS;
   f      <= TRANSPORT (a and b) AFTER 4 NS;
end EXAMPLE;
```



2.10 จากคำสั่ง VHDL ต่อไปนี้ จงแสดงค่าของสัญญาณต่อไปนี้เป็นเลขฐาน 2

```
architecture EXAMPLE of ASSIGNMENT is
signal E, F : bit;
signal A_BUS, B_BUS, C_BUS  :   bit_vector (3 downto 0);
signal D_BUS : bit_vector (0 to 7);
begin
  A_BUS <= X"C";
  B_BUS <=('1', '1', '0', '1');
  C_BUS <= (1=>'1', 3 => '0',others => B_BUS(3));
  D_BUS <= A_BUS(3 downto 0) & B_BUS(2 downto 0) & '0';
  E <= C_BUS(0);
  F <= D_BUS(7);
end EXAMPLE;
```