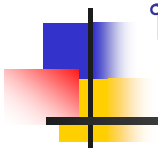
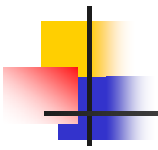


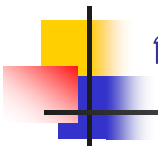
พื้นฐานภาษา VHDL



- การเขียนภาษา VHDL เบื้องต้น

  ปกติภาษา VHDL เป็นภาษาลักษณะเป็นแบบ Case insensitive หมายถึง ไม่มีความแตกต่างกันในการเขียนตัวพิมพ์เล็กแล้วพิมพ์ใหญ่ ไม่ว่าจะเขียนแบบไหนก็มีความหมายเหมือนกัน

  Entity = ENTITY = entity = eNtitY



การทำงานแบบขนาน

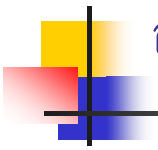
- ARCHITECTURE example OF my_design IS

  BEGIN

  a <= b ; - - line no .1

  b <= c ; - - line no .2

  END example ;



ออบเจ็กต์ในภาษา VHDL

- CONSTANT เมื่อกำหนดค่าเริ่มต้นให้แล้วจะคงค่านั้นไว้ตลอด
- SIGNAL สัญญาณสามารถรับค่าได้เพียงค่าเดียวเท่านั้นในขณะเวลาหนึ่ง
- VARIABLE สามารถกำหนดค่าใดๆให้ได้และสามารถที่จะเปลี่ยนแปลงค่าได้ตลอดเวลาของการจำลองการทำงาน

การประกาศออบเจ็กต์

Object_class identifier : TYPE := initial_value

- Object_class ได้แก่ CONSTANT , SIGNAL หรือ VARIABLE
- การตั้งชื่อ เป็นไปตามกฎของภาษา VHDL
- TYPE การกำหนดประเภทออบเจ็กต์ที่ประกาศนั้นๆ

การตั้งชื่อ ออบเจ็กต์มีหลักเกณฑ์ต่อไปนี้

- ชื่อประกอบไปด้วยตัวหนังสือที่ใช้ในภาษาอังกฤษ
  - พยัญชนะ A - Z , a - z
  - ตัวเลข 0,1,2,3,4,5,6,7,8,9
  - เครื่องหมายขีดเส้นใต้ ‘_’
- ชื่อจะต้องขึ้นต้นด้วยพยัญชนะเสมอ
- ชื่อประกอบด้วยพยัญชนะได้ไม่จำกัด
- การใช้เครื่องหมายขีดเส้นใต้ต้องนำหน้าพยัญชนะหรือตามหลัง
- พยัญชนะตัวเล็กตัวใหญ่มีความหมายเหมือนกัน

ชนิดของออบเจ็กต์

- Boolean คือกลุ่ม ตรรกะ ได้แก่ FALSE และ TURE
- BIT คื่อ กลุ่มของค่า ‘1’ และ ‘0’
- INTEGER คือกลุ่มจำนวนเต็มมีค่า -214748348 ถึง 214748347
- CHARACTER คือกลุ่มของค่า พยัญชนะ ‘A’ - ‘Z’, ‘a’ - ‘z’
- TIMEได้แก่หน่วยเวลา
- SEVERITY LEVEL คือกลุ่มค่าการแจ้งเตือนความรุนแรง

องค์ประกอบของภาษา VHDL

Architecture

```
ARCHITECTURE arch_name OF entity_name Is
signal_declarations
type_ declarations
subtype_ declarations
subprogram_ declarations
component_ declarations
BEGIN
- - < concurrent_statements >
```

PROCESS_statements

concurrent_signal_assignments

component_instantiation_statements

Generate_Statements

END PROCESS;

END arch_name;

ประเภทข้อมูลในภาษา VHDL

ค่าคงที่ (CONSTANT)

- เป็นการกำหนดค่าที่ประกาศไว้แล้ว จะไม่สามารถเปลี่ยนแปลงค่าได้ เมื่อนำไปใช้ในการเขียนบรรยายพฤติกรรมวงจร โดยมีรูปแบบการเขียนดังนี้
- CONSTANT constant_name : type_spec := value ;

ประเภทข้อมูลในภาษา VHDL

ตัวอย่าง การประกาศค่าคงที่รูปแบบต่างๆ

CONSTANT vdd : REAL := -4 . 5 ;

CONSTANT cycle : TIME := 100 ns ;

CONSTANT pi : REAL := 3.14 ;

CONSTANT five : BIT_VECTOR := “0101”;

การประกาศค่าคงที่จะถูกประกาศไว้ในส่วนต้นของ Architecture, หรือในส่วน Process declaration ซึ่งจะมีรูปแบบการใช้งานต่างกันออกไปขึ้นอยู่กับว่าประกาศไว้ที่ใด

ประเภทข้อมูลในภาษา VHDL

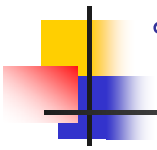
สัญญาณ

สัญญาณ (SIGNAL) มีหน้าที่ส่งผ่านข้อมูลระหว่าง Concurrent statement ภายใน Architecture และใช้ใน Sequential statement ภายในคำสั่ง PROCESS เพื่อกำหนดพฤติกรรมการทำงานของวงจร มีรูปแบบการเขียนดังนี้

SIGNAL signal_name : type_spec;

หรือบางครั้งสามารถกำหนดค่าเริ่มต้นให้กับสัญญาณที่ประกาศได้ดังนี้

SIGNAL signal_name : type_spec := initial_value



ประเภทข้อมูลในภาษา VHDL

ตัวอย่าง การประกาศสัญญาณในรูปแบบต่างๆ

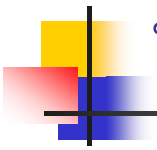
```
SIGNAL a_bus, b_bus, z_bus : BIT_VECTOR(3 DOWNTO 0);

SIGNAL a_BIT, b_BIT, c_BIT, d_BIT : BIT ;

SIGNAL byte : BIT_VECTOR(0 TO 7) ;

SIGNAL count : INTEGER RANGE 1 TO 50 ;

SIGNAL ground : BIT := '0' ;
```

ประเภทข้อมูลในภาษา VHDL

การกำหนดค่าให้กับสัญญาณ (Signal assignment)

การกำหนดค่าให้สัญญาณ จะต้องกระทำระหว่างชนิดของข้อมูลสัญญาณประเภทเดียวกันเท่านั้นถึงจะ Assign ค่าให้กันได้

ตัวอย่าง การกำหนดค่าสัญญาณ (Signal assignment)

```
SIGNAL z_bus : BIT_VECTOR (3 DOWNTO 0) ;
SIGNAL c_bus : BIT_VECTOR (1 TO 4) ;
. . .
- - Legal
z_bus (3 DOWNTO 2) <= “00”;
c_bus (2 TO 4) <= z_bus (3 DOWNTO 1) ;  ✓
- - Illegal
Z_bus (0 TO 1) <= “11” ;  ✗
```


จะต้องแก้ไขให้ถูกต้อง จะต้องแก้ไขดังนี้

```
z_bus (1 DOWNTO 0) <= “11” ;  ✓
```

เนื่องจากประกาศสัญญาณ z_bus ในลักษณะทิศทางของบัสเป็นแบบ DOWNTO เมื่อนำไปใช้งานในโมเดล ไม่สามารถที่จะกลับทิศทางของสัญญาณดังกล่าวได้

การประกาศนามแฝง

Alias เป็นคำสั่งที่ใช้สำหรับสร้างชื่อใหม่ให้กับวัตถุ เพื่อทำให้โค้ดที่เขียนอ่านได้เข้าใจยิ่งขึ้น มีรูปแบบการเขียนดังนี้

```
ALIAS alias_name : alias_type IS object_name;
```

ตัวอย่าง การใช้งานคำสั่ง ALIAS ในการกำหนดชื่อใหม่ให้กับบิตบางบิตของสัญญาณ count

```
SIGNAL count : BIT_VECTOR (7 DOWNTO 0);
- - An alias is an alternative name for an existing object
- - (signal, variable or constant). It doesn't define a new object
ALIAS sign_bit : BIT IS count (7) ;
ALIAS lsb_bit : BIT IS count (0) ;
ALIAS nibble_l : BIT_VECTOR (3 DOWNTO 0) IS count (3 DOWNTO 0) ;
ALIAS nibble_h : BIT_VECTOR (3 DOWNTO 0) IS count (7 DOWNTO 4) ;
```


ตัวแปร

VARIABLE ในภาษา VHDL จะเป็นตัวแปรในลักษณะแบบ Local object ที่ประกาศในส่วนที่เป็น Sequential body เท่านั้น การประกาศ VARIABLE มีรายละเอียดดังนี้

```
VARIABLE variable _ neme: varible _ type;
```

หรือถ้าต้องการกำหนดค่าเริ่มต้นให้กับ VARIABLE ได้

```
VARIABLE variable _ neme: varible _ type:= inintial_value;
```

ตัวอย่าง การประกาศ VARIABLE ในรูปแบบต่าง ๆ ภายในคำสั่ง PROCESS

```
PROCESS(…)
--Variable declaration within PROCESS statement only!!
VARIABLE index : INTEGER RANGE 1 TO 50 ;
VARIABLE memory : BIT_VECTOR(0 TO 7);
VARIABLE cycle_time : TIME RANGE 10 ns TO 50 ns : = 10
   ns BEGIN
```


ตัวอย่าง แสดงการถ่ายโอนค่าสัญญาณ (Signal assignment) ระหว่างวัตถุต่างชนิดกัน

```
SIGNAL abus, bbus : BIT_VECTOR(3 DOWNTO 0);
SIGNAL int_a, int_b : INTEGER;
SIGNAL real_a, real_b : REAL;
. . .
BEGIN
   abus <= bbus; ✓
   abus <= int_a; ✗
   real_a <= real_b; ✓
   int_b <= real_b; ✗
```

ข้อมูลประเภทกายภาพ

ข้อมูลกายภาพจะถูกประกาศใน package STANDARD

```
TYPE TIME IS RANGE -2147483647 TO 2147483647
unit
        fs;                     -- femtoseconds
        ps = 1000 fs;           -- femtoseconds
        ns = 1000 ps;           -- nanoseconds
        us = 1000 ns;           -- microseconds
        ms = 1000 us;           -- miliseconds
        sec = 1000 ms;          -- seconds
        min = 60 sec;           -- minutes
        hr = 60 min;            -- hours
END  units;
```

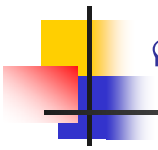

ข้อมูลประเภทอะเรย์

ในภาษา VHDL สามารถใช้งานข้อมูลประเภทอะเรย์โดยที่ข้อมูลประเภทอะเรย์ จะเป็นกลุ่มของชนิดข้อมูล ประเภท Scalar ดังแสดงในตัวอย่างต่อไปนี้

```
TYPE std_ulogic_vector IS ARRAY (NATURAL RANGE<>) OF std_ulogic;
```

หรือเป็นข้อมูลประเภท STRING หรือ BIT_VECTOR ก็ตามใน padkage STANDARD

```
TYPE string IS ARRAY (POSITIVE RANGE <>) OF CHARACTER;
TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE<>)OF BIT;
```



ตัวอย่าง การสร้างชนิดข้อมูลที่ผู้ใช้กำหนดขึ้นเอง

```
TYPE ram IS ARRAY (0 TO 31) OF INTEGER RANGE 0 TO 255 ;
TYPE my_type IS ARRAY (3 DOWNTO 0) OF std_logic;
TYPE my_int IS ARRAY RANGE 0 TO 15 ;
```


ตัวอย่าง การประกาศข้อมูลประเภทอะเรย์แบบมี Index position ที่แตกต่างกันโดยความหมายของตำแหน่งในกลุ่มของข้อมูลอาจหมายถึงนัยสำคัญ ของบิตต่างๆในระบบบัสของสัญญาณที่ประกาศใช้ในโมเดล

```
TYPE word IS ARRAY (3 DOWNTO 0) OF std_logic;
TYPE inv_word IS ARRAY (0 TO 3) OF std_logic;

SIGNAL a_bus : inv_word;
SIGNAL b_bus : word;
```

การรวมข้อมูลแบบ Aggregates

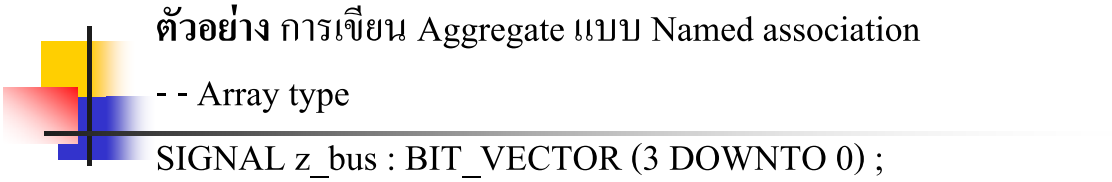
เป็นรูปแบบการกำหนดค่าให้กับกลุ่มของข้อมูลประเภทอะเรย์ โดยมีรูปแบบการเขียนดังนี้

(value_1, value_2, . . .)

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

ตัวอย่าง การเขียน Aggregate แบบ Positional association

```
- - Array type
SIGNAL z_bus : BIT_VECTOR (3 DOWNTO 0) ;
- - Scalar types
SIGNAL a_bit, b_bit, c_bit, d_bit : BIT ;
. . .
        - - Positional  association
        z_bus  <= (a_bit, b_bit, c_bit, d_bit) ;
```

ตัวอย่าง การเขียน Aggregate แบบ Named association

```
- - Array type
SIGNAL z_bus : BIT_VECTOR (3 DOWNTO 0) ;
- - Scalar types
SIGNAL a_bit, b_bit, c_bit, d_bit  :   BIT ;
. . .
        - - Named  association
        z_bus  <=  (   2  =>  b_bit ,
                       1  =>  c_bit ,
                       3  =>  a_bit ,
                       0  =>  d_bit  ) ;
```

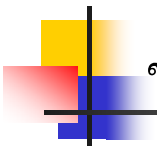
ตัวอย่าง การเขียนรูปแบบ Slices of Arrays

```
SIGNAL a_bus : std_logic_vector (7 DOWNTO 0) ;
SIGNAL b_bus : std_logic_vector (0 TO 7);
BEGIN
        - - Legal
        a_bus(4) <= '1' ; ✓
        a_bus(3 DOWNTO 0) <= "1010" ;
        b_bus(2) <= a_bus(1) ;
        b_bus(3 TO 7) <= "11100" ;
        - - Illegal
        a_bus (0  TO 1)  <= "00" ; ✗
        b_bus (3 DOWNTO 2)  <= "10" ;
END ;
```


การรวมข้อมูลแบบ Concatenation

ในภาษา VHDL มีการใช้เครื่องหมาย Ampersand (&) โดยใช้เชื่อมต่อข้อมูลขนาดเล็กให้มีขนาดใหญ่ พิจารณาจากตัวอย่างต่อไปนี้

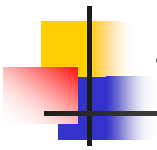
```
SIGNAL z_bus : std_logic_vector (7 DOWNTO 0) ;
SIGNAL a_nible, b_nible : std_logic_vector (3 DOWNTO 0) ;
SIGNAL a_bit, b_bit, c_bit, d_bit,  :  std_logic ;
BEGIN
        z_bus  <= "0000" & a_nible ;
        z_bus  <= a_nible & b_nible ;
        z_bus  <= a_bit & b_bit & c_bit & d_bit & "1010";
. . .
```



ข้อมูลแบบ Enumerated

เป็นประเภทข้อมูลที่ประกอบด้วย ตัวหนังสือหรือชื่อต่างๆที่สามารถกำหนดขึ้นมาเอง โดยสมาชิกของข้อมูลประเภทนี้จะไม่มีความหมายทางกายภาพเป็นเพียงตัวอักษรหรือตัวหนังสือที่ใช้แทนความหมายที่กำหนดขึ้นมาเองโดยรูปแบบดังนี้

TYPE type_name IS type_definition ;



ประเภทข้อมูลย่อย (Subtypes)

Subtype เป็นชนิดของข้อมูลย่อยของข้อมูลหลัก (Type) โดยมีรูปแบบการเขียน ดังนี้

SUBTYPE subtype_name IS base_type RANGE

range_constraint ;

ประเภทข้อมูลแบบประสม (Composite Data type)

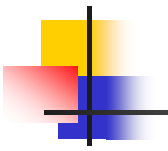
Composite type เป็นชนิดข้อมูลที่ประกอบขึ้นจากชนิดข้อมูลประเภทอะเรย์ หรือกลุ่มของข้อมูลประเภทต่างๆ โดยสามารถแบ่งเป็น 2 ประเภท ได้แก่

1. Arrays of array
2. Records

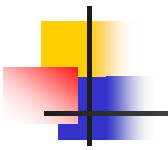
ตัวดำเนินการในภาษา VHDL

ภาษา VHDL จะมีชุดของ Operator ที่ใช้สำหรับการเปรียบเทียบ การกระทำการทางคณิตศาสตร์หรือการกระทำการทางด้านตรรกะ โดย Operator จะเปรียบเสมือน Toolkit ที่ใช้สำหรับสร้าง RTL Model ขึ้นมา โดยชุดของ Operator ทั้งหมดในภาษา VHDL มีดังนี้

NOT	inversion
AND	and function
NAND	not-and function



OR	or function
NOR	not-or function
XOR	exclusive-or function
XNOR	exclusive-nor function
=	equality
/=	inequality
>=	greater-than or equal
>	greater-than
<=	less-then or equal
<	less-than
SLL	shift-left logical



SRL	shift-right logical
SLA	shift-left arithmetic
SRA	shift-right arithmetic
ROL	rotate left
ROR	rotate right
+	addition
-	minus sign
*	multiplication
/	division
MOD	modulo arithmetic
REM	remainder after division
**	exponentiation
ABS	absolute value
&	concatenation

ลำดับการทำงานของ Operator

- การแบ่งกลุ่มของ Operator สามารถแบ่งออกเป็นกลุ่มและมีลำดับการทำงานของ Operator จากสูงไปต่ำดังนี้

กลุ่มพิเศษ

* *, ABS, NOT

กลุ่มการคูณ

*, /, MOD, REM

กลุ่มเครื่องหมาย

+, -

กลุ่มการบวก

+, -, &

กลุ่มการเลื่อนบิต

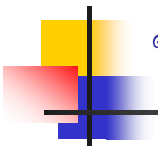
SLL, SRL, SLA, SRA, ROL, ROR

กลุ่มความสัมพันธ์

=, /=, <, <=, >, >=

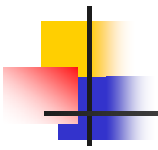
กลุ่มตรรกะ

AND, OR, NAND, NOR, XOR, XNOR



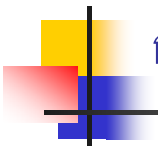
ชนิดข้อมูลที่สามารถใช้ตัวดำเนินการ Boolean

- BIT
- BIT_VECTOR
- STD_LOGIC
- STD_LOGIC_VECTOR



ตัวอย่าง การใช้งาน Boolean

```
- - And  gate  2   Input
ENTITY   and2   IS
    PORT   ( a_bit, b_bit  :  IN  BIT ;
                 z_bit  :  OUT  BIT  );
END   and2;
ARCHITECTURE and_operator  IF  and2  IS
BEGIN
          z_bit  <=  a_bit  AND  b_bit ;
END  and_operator ;
```



การสังเคราะห์วงจรจากตัวดำเนินการ Logical

NOT a <=> not a

a AND b <=> a and b

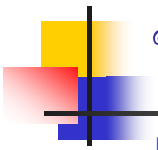
a NAND b <=> not a or not b

a OR b <=> a or b

a NOR b <=> not a and not b

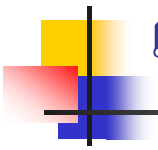
a XOR b <=> (not a and b) or (a and not b)

a XNOR b <=> (a and b) or (not a and b)



ชนิดข้อมูลและเงื่อนไขการใช้งาน Relational

- ข้อมูลที่มีรูปแบบเป็น Scalar ทุกประเภท
- ข้อมูลทุกชนิดที่เป็น Array ขนาด 1 มิติ
  - การใช้งาน relational Operator มีเงื่อนไขดังนี้
    - Operand ที่นำมาเปรียบเทียบจะต้องเป็นข้อมูลชนิดเดียวกัน
    - กรณีที่ข้อมูลมาเปรียบเทียบกันเป็น Array ข้อมูลไม่จำเป็นที่จะต้องมีขนาดความยาวเท่ากัน
    - การเปรียบเทียบจะจัดเรียงข้อมูลที่จะนำมาเปรียบเทียบชิดทางด้านซ้ายเสมอ
    - ข้อมูลที่นำมาเปรียบเทียบกัน จะไม่คำนึงถึงความหมายทางคณิตศาสตร์



แบบ Array

- สำหรับวงจร Array จะต้องมี Array มาเปรียบเทียบกัน จะต้องมีขนาดความยาวเท่ากันมิเช่นนั้นโปรแกรมสังเคราะห์วงจรจะไม่สร้างเป็นวงจรเปรียบเทียบขึ้น โดยจะสร้างในลักษณะเป็น Logic False จาก Boolean แทน

Shifting

- Shifting Operators จะเป็น Operator ที่เพิ่มขึ้นมาใน VHDL'93 โดยจะสามารถทำกับ Array , Boolean และ Bit ดังนั้นถ้าหากต้องการใช้งาน Operators ประเภทนี้ จึงต้องควรตรวจสอบซอร์ฟแวร์สำหรับจำลองการทำงานและซอร์ฟแวร์สำหรับสังเคราะห์วงจรเสียก่อนว่าสามารถใช้ได้หรือไม่ Shift Operators ประกอบด้วย

  SLL -- Shift left logical

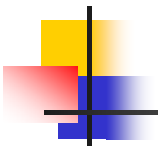
  SRL -- Shift right logical

  SLA -- Shift left arcthmetic

  SRA -- Shift right arcthmetic

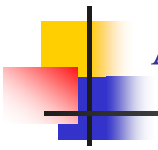
  ROL -- Rotate left

  ROR -- Rotate right



- ชนิดข้อมูลที่สามารถใช้ตัว Shift Operators คือชนิดข้อมูล bit หรือ Boolean ที่เป็น array ขนาด 1 มิติ
- ตัวอย่างการใช้ Logical Shift Left

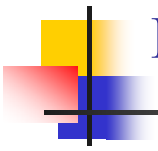
```
Z <= a SLL 1;
```

Arithmetic Shift

- การทำ Arithmetic Shift จะมีลักษณะแตกต่างจากการทำ Logical Shift ที่ Arithmetic Shift จะแทนค่าบิตที่ Shift เข้าไปใหม่ด้วยบิตสุดท้ายของด้านที่มีการ Shift
- Arithmetic Shift Left จะเป็นการเลื่อนบิตไปทางด้านซ้ายมือโดยจะแทนที่บิตที่เลื่อนเข้าไปใหม่ด้วย ค่าที่อยู่ใน LSB เช่น

```
Z <= a SLA 1;
```

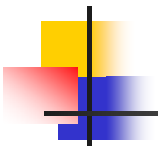


Rotate

การหมุนจะเป็นการเลื่อนบิตข้อมูล โดยจะนำข้อมูลที่ถูกเลื่อนออกไป ไปต่อท้ายของชุดข้อมูลที่มีการเลื่อน เช่น

Rotate Left จะเลื่อนบิตไปทางด้านซ้ายมือแล้วนำข้อมูลที่ถูกเลื่อนออกไปทางด้านซ้ายมือสุด (MSB) ไปแทนที่บิตข้อมูลทางด้านขวาสุด (LSB) เช่น

```
Z  <=  a  ROL  2;
```

- ชนิดของข้อมูลและเงื่อนไขการใช้งาน Arithmetic Operators
  - INTEGER
  - REAL

  เงื่อนไขการใช้งาน Arithmetic Operators คือ
    - Operand จะต้องเป็นข้อมูลชนิดเดียวกัน
    - ผลลัพธ์จะมีชนิดข้อมูลเป็นชนิดเดียวกันกับ Operand