บทที่ 3
ประเภทข้อมูลของภาษา VHDL

รศ.ณรงค์ บวบทอง
ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์
มหาวิทยาลัยธรรมศาสตร์

หัวข้อ

- บทนำ
- ออบเจ็กต์ประเภท ค่าคงที่ สัญญาณ และ ตัวแปร
- ประเภทข้อมูล
- ข้อมูลประเภทมาตราฐาน
- ข้อมูลประเภทเวลา
- ข้อมูลประเภทอะเรย์

- ข้อมูลแบบแจกแจง (Enumeration Types)
- ข้อมูลแบบ IEEE Standard Logic Type
- Records
- Subtypes
- Aliases

บทนำ

ถ้าพิจารณาว่าการเขียนภาษา VHDL ก็เหมือนกับการเขียนโปรแกรมทั่วๆไป ดังนั้นการเขียนถึงข้อมูลไม่ว่าจะเป็นตัวเลขหรือตัวอักษรก็ต้องมีกฎเกณฑ์ที่ชัดเจนดังจะได้กล่าวถึงต่อไปปนี้

การระบุจำนวน (Numbers)

เพื่อความง่ายในการอ่าน

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

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

อักขระ (Characters) และสตริง (Strings)

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

สตริงหมายถึงอักขระหลายๆตัวที่เขียนติดกันรวมถึงอักขระเว้นวรรคด้วย การเขียนสตริงใช้เครื่องหมาย “ ” เช่น
“VHDL” “Thi is a string” และถ้าเป็นสตริงว่างให้เขียนดังนี้ “”

บิตสตริง (Bit Strings)

หมยถึงกลุ่มของเลขฐานใดๆแต่ต้องกาให้พิจารณาเป็นบิตๆ

B"1010110" --เลขฐานสองมีความยาว 7 บิต

O"126" --เลขฐานแปด เทียบได้กับเลขฐานสอง B"001_010_110

X"56" --เลขฐานสิบหก เทียบได้กับเลขฐานสอง B"0101_0110"

ออบเจ็กต์ประเภท ค่าคงที่ สัญญาณ และ ตัวแปร

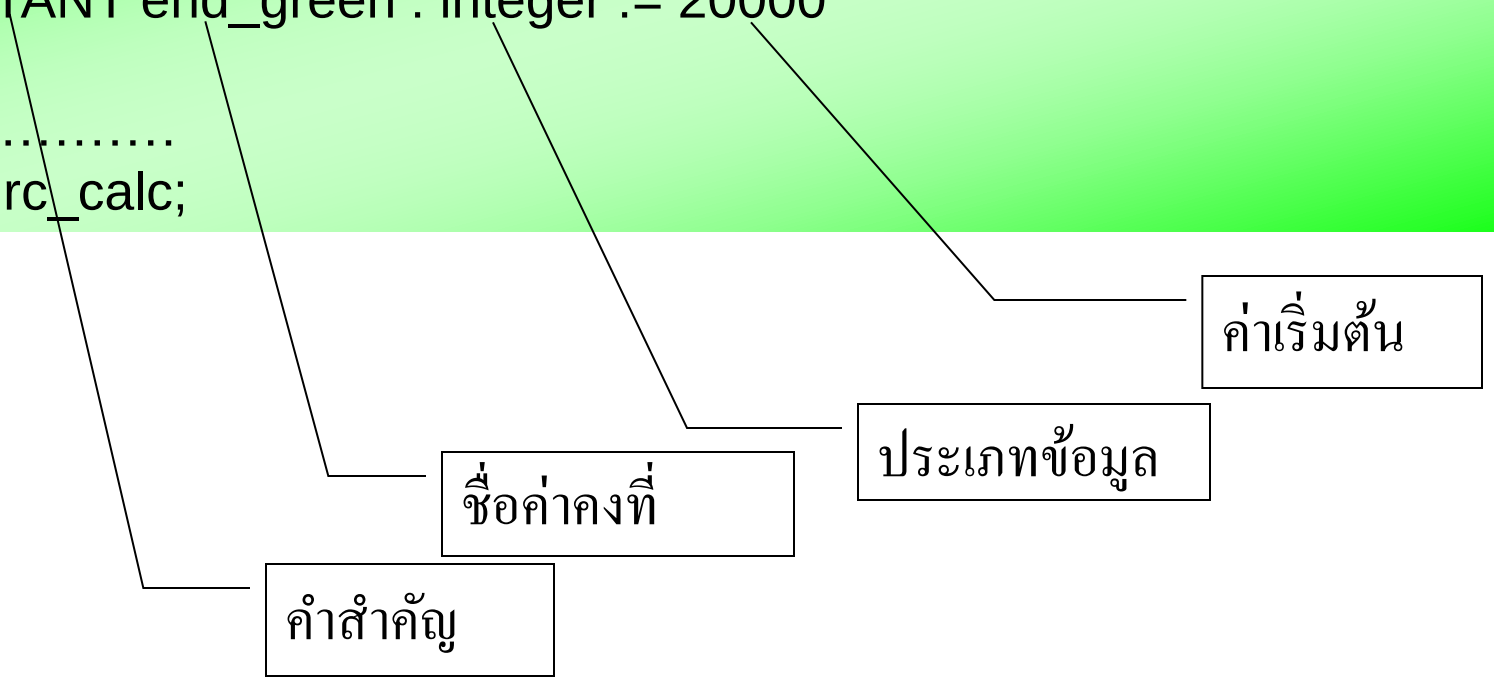
ค่าคงที่ (Constant)

ค่าค่งที่เป็นออบเจ็กต์ที่มีค่าได้เพียงค่าเดียวตลอดทั้งโปรแกรม การประกาศใช้ค่าคงที่ต้องประกาศไว้ใน Architecture และอยู่ก่อน Begin ของ Architecture นั้นๆ

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

ตัวอย่างค่าคงที่

```
ARCHITECTURE circ_calc OF car IS
CONSTANT pi   : real := 3.14159
CONSTANT end_red  : integer := 10000
CONSTANT end_green : integer := 20000
BEGIN
    ……….
END circ_calc;
```



สัญญาณ (Signal) และ ตัวแปร (Variable)

ทั้งสัญญาณและตัวแปรใช้สำหรับส่งผ่านข้อมูลระหว่าอุปกรณ์ แต่สัญญาณจะคล้ายกับสายไฟในวงจรไฟฟ้า ส่วนตัวแปรกับค่าคงที่จะเหมือนกับตัวแปรและค่าคงที่ในโปรแกรมระดับสูงทั่วไป การจำลองการทำงานและการสังเคราะห์วงจรสำหรับสัญญาณและตัวแปรจะให้ผลที่ต่างกัน

ตัวแปร (Variable)

การประกาศและการใช้ตัวแปรมีคุณสมบัติดังนี้

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

```
process is
        variable sum: integer := 0;
        variable stat: real;
begin
        ……..

        sum := sum + 1;
        stat := 1.123;
end process;
```

ไม่กำหนดค่าเริ่มต้น

เปลี่ยนแปลงค่าได้

สัญญาณ (Signal)

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

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

ตัวอย่างการใช้งานตัวแปรกับสัญญาณ

```
architecture behav of reg_file is
   signal reg_A: integer := 0;
   signal reg_B : integer;
begin
   process(d_inp, input_C)

begin
   reg_A <= d_inp;
   reg_B <= input_C after 5 ns;
```

```
ARCHITECTURE
……..
   SIGNAL a,b: integer;
PROCESS (a) is
   VARIABLE v1,v2: integer;
BEGIN
   v1 := a-v2;
   b <= -v1;
   v2 := b+v1*3;
END PROCESS;
```

เปลี่ยนแปลงค่าได้

เปลี่ยนแปลงค่า และกำหนดเวลาได้

สังเกตการใช้เครื่องหมายเพื่อรับค่า

เปรียบเทียบระหว่างสัญญาณกับตัวแปร

ขอบเขตการใช้งาน

ตัวแปรมีการทำงานอยู่ภายใน Process

สัญญาณมีการทำงานอยู่ภายในโมดูลที่ออกแบบ

การจำลองการทำงาน

การกำหนดค่าให้กับตัวแปรจะมีผลทันที

การกำหนดค่าให้กับสัญญาณจะมีผลในรอบการทำงานถัดไป

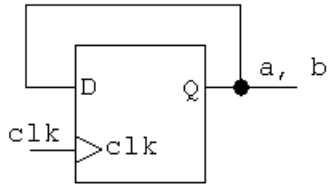
สัญญาณเกิดการเปลี่ยนแปลง ค่าเกิดเมื่อสิ้นสุด Process

การสังเคราะห์ (Synthesis) ภายใต้คำสั่งเกี่ยวกับสัญญาณนาฬิกา (clock)

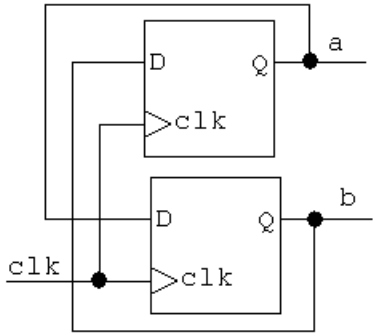
ตัวแปรสังเคราะห์เป็นสายไฟใน netlist

สัญญาณจะสังเคราะห์เป็นอุปกรณ์ความจำ (memory)

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




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



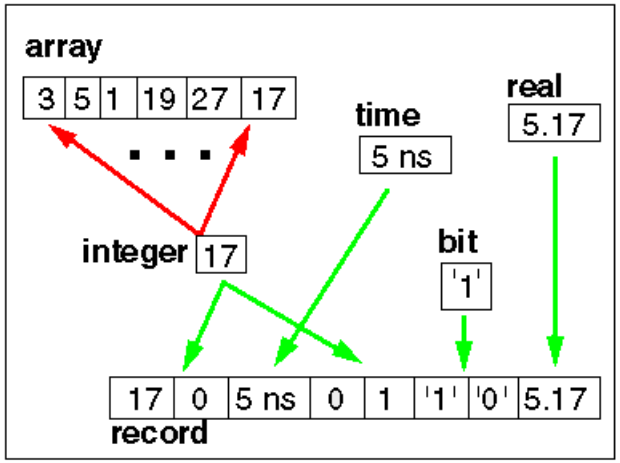
ประเภทข้อมูล

ข้อมูลจำแนกเป็นกลุ่มใหญ่ได้ดังนี้

Scalar types: เป็นข้อมูลที่มีค่าที่แน่นอนค่าหนึ่งๆ เช่น integer, real, bit, enumerated, และ กายภาพ (physical)

Composite types: เป็นข้อมูลที่อาจประกอบด้วยค่าหลายค่า เช่น array และ record

Types แบบอื่นๆ: เช่น file access



ข้อมูลประเภทมาตราฐาน

BIT มีค่าเป็น 0 หรือ 1 ค่าเริ่มต้นกำหนดไว้ให้เป็น 0

BIT_VECTOR เป็นกลุ่มหรืออะเรย์ของบิต

CHARACTER เป็นตัวอักษร 1 ตัวมีค่าตามตาราง ASCII

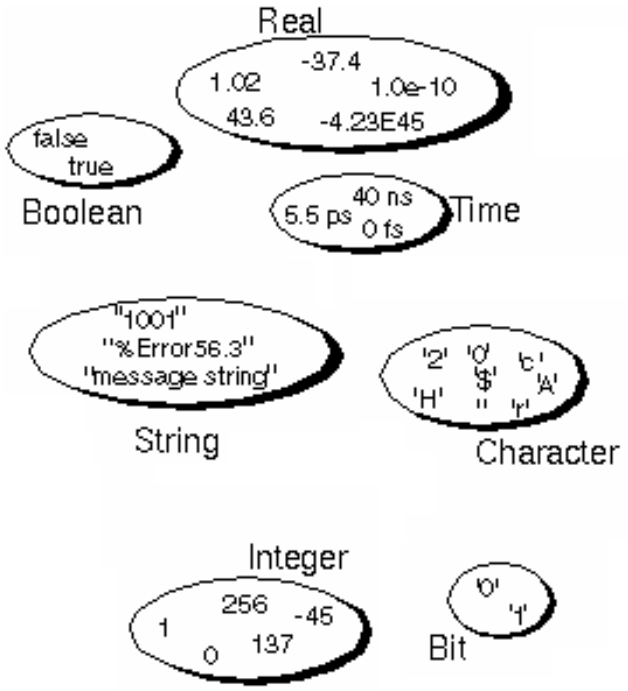
STRING เป็นกลุ่มหรืออะเรย์ของ CHARACTER

BOOLEAN มีค่าเป็น จริง(TRUE) กับเท็จ(FALSE)

INTEGER เป็นเลขจำนวนเต็ม มีค่าตามขอบเขตที่กำหนดเช่น 0 ถึง 255

REAL เป็นจำนวนจริง มีค่าตามขอบเขตที่กำหนดมักใช้กับตัวแปร

TIME เป็นข้อมูลประเภทเวลา



ข้อมูลประเภทเวลา

ใช้งานเป็นเวลาหน่วยของอุปกรณ์ หรือใช้ใน Testbench เพื่อการทดสอบการทำงาน ทำงาน การคูณหรือการหารเวลา จะได้เป็นเวลาเช่นเดิม

หน่วยของเวลาได้ตั้งแต่ fs(femtosecond) ps(pico) ns(nano) us(mico) ms(milli) sec(second) min(minute) และ hr(hour)

```
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;
```

เวลาหน่วง (Delay Model) ใน VHDL มีอยู่ด้วยกัน 3 แบบคือ

Inertial delay ใช้เป็นค่าหน่วงเวลาของโมเดลในภาษา VHDL สัญญาณที่มีขนาดเล็กกว่า ค่าเวลา ที่ระบุไว้ใน after จะไม่ถูก propagate ใช้เป็นค่า propagation delay time ของ อุปกรณ์

Transport Delay เป็นค่าหน่วงเวลาของสัญญาณอินพุทที่จะออกไปยังเอาท์พุททุกค่า

Delta delay เป็นค่าหน่วงเวลาของการทำงานของอุปกรณ์ ในกรณีที่ไม่ได้กำหนด inertial delay

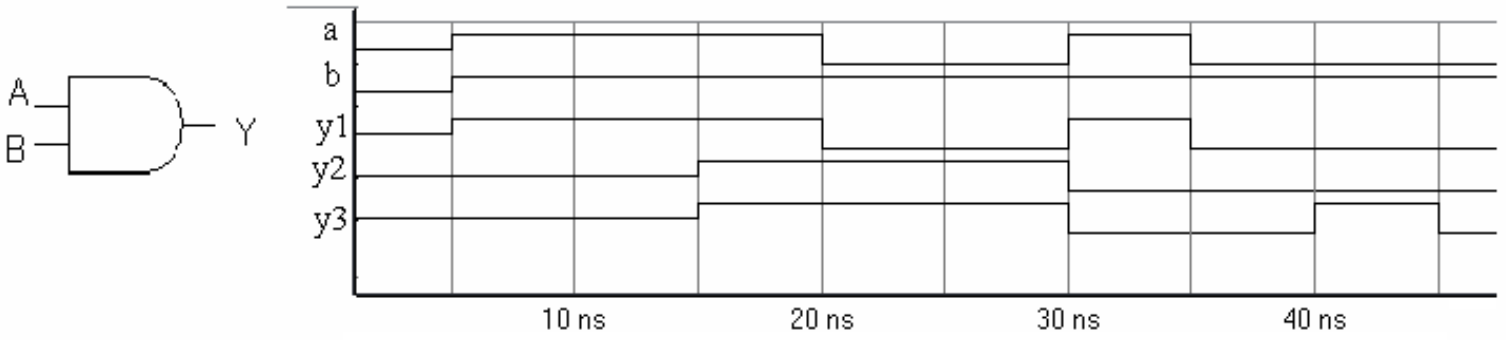
ตัวอย่างที่แสดงให้เห็นถึงความแตกต่างของเวลาหน่วง

```
beginInertial delay
    y1 <= (a and b);
    y2 <= (a and b) after 10 ns;
    y3 <= transport (a and b)  after 10 ns;
end dflow;Transport Delay
```

มีเฉเพาะ Delta delay

Inertial delay

Transport Delay



Delta Delay เมื่อY <= A AND B

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

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)

การสร้างอะเรย์ใหม่ได้ด้วยคำสั่ง TYPE

TYPE type_name IS ARRAY (RANGE) OF element_type;

ความหมาย

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

type_name เป็นชื่อของประเภทช้อมูล

RANG เป็นขนาดของอะเรย์

Element_type เป็นประเภทข้อมูลของส่วนประกอบอะเรย์นี้

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

signal TYPE_BUS : MY_BYTE;
```

Enumeration type

```
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;
```


ALARM_TIME(HOUR10) = 0

ALARM_TIME(HOUR1) = 7

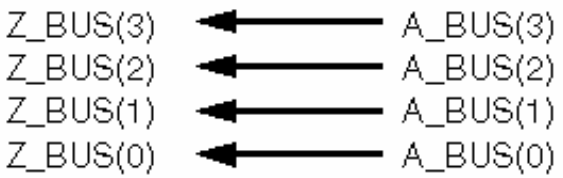
ALARM_TIME(MINUTES10) = 3

ALARM_TIME(MINUTES1) = 0

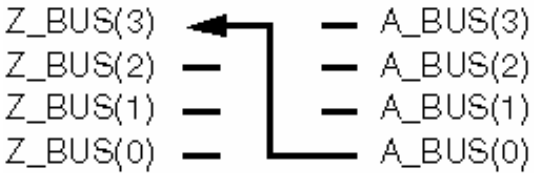
การกำหนดค่าข้อมูลประเภทอะเรย์

```
architecture EXAMPLE of ASSIGNMENT is
   signal Z_BUS,A_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;
```

Z_BUS <= A_BUS;

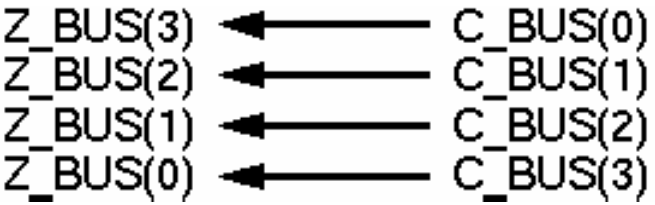


Z_BUS(3) <= A_BUS(0);



ตัวอย่างการส่งถ่ายข้อมูลระหว่างอะเรย์เมื่อกำหนดดรรชนีของอะเรย์ ไม่เหมือนกัน

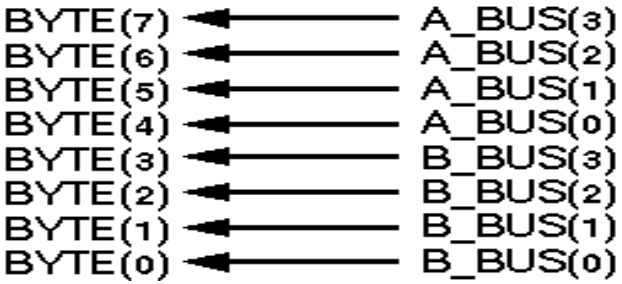
```
architecture EXAMPLE of ARRAYS is
   signal Z_BUS : bit_vector (3 downto 0);
   signal C_BUS : bit_vector (0 to 3);
begin
   Z_BUS <= C_BUS;
end EXAMPLE;
```



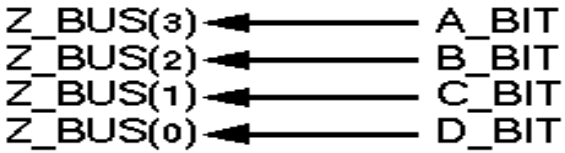
การรวมข้อมูลแบบ Concatenation

การรวมข้อมูลขนาดเล็กให้มีขนาดใหญ่ขึ้น โดยใช้เครื่องหมาย Ampersand (&)

```
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;
```




```
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;
```



การรวมข้อมูลแบบ Aggregate และอะเรย์หลายมิติ (Multidimensional Arrays)

เป็นการกำหนดค่าให้กับกลุ่มของข้อมูลประเภทอะเรย์ โดยเขียนได้ 2 แบบ

- แบบสัมพันธ์กับตำแหน่งของอะเรย์

(value_1,value_2);

- แบบสัมพันธ์กับชื่อของอะเรย์

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

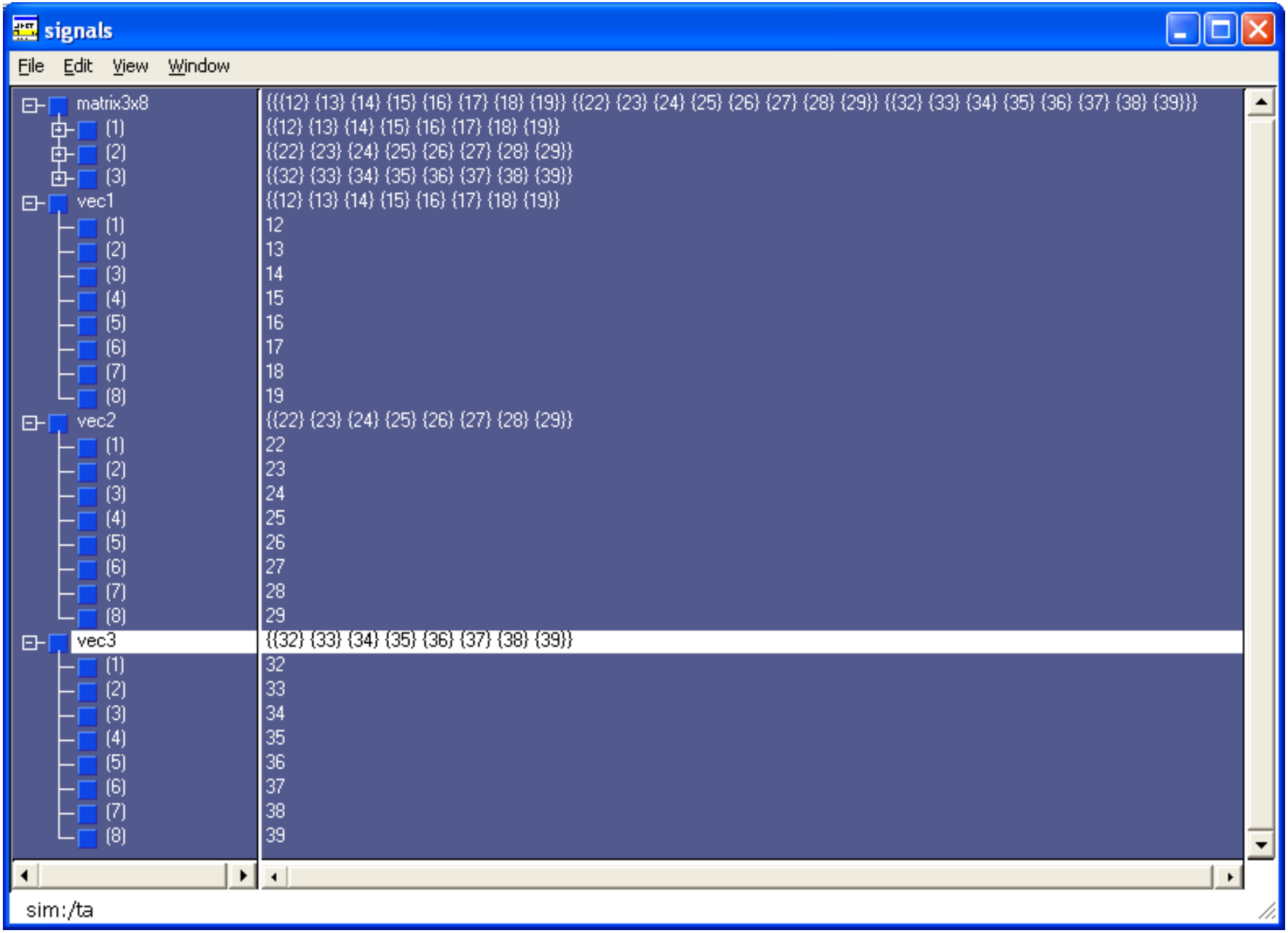
```
architecture dataflow of array_test is
SIGNAL A, B, C: std_logic_vector(4 downto 0);
SIGNAL D: std_logic_vector(0 to 1);
SIGNAL E: std_logic_vector(0 to 6);
begin
        A <= ( 2|3 => '1', others => '0');
        B <= (1 => '0', 3 => '1', others => C(1));
        C <= ('0', '1', '1', '0', '1');
        D <= '0' & A(2);
        E <= A(3 downto 0) & B(2 downto 0);

end dataflow;
```

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

การกำหนดค่าให้กับอะเรย์แบบ 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;
```


signals
File Edit View Window
matrix3x8
(1)
(2)
(3)
vec1
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
vec2
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
vec3
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
{{{12} {13} {14} {15} {16} {17} {18} {19}} {{22} {23} {24} {25} {26} {27} {28} {29}} {{32} {33} {34} {35} {36} {37} {38} {39}}}
{{12} {13} {14} {15} {16} {17} {18} {19}}
{{22} {23} {24} {25} {26} {27} {28} {29}}
{{32} {33} {34} {35} {36} {37} {38} {39}}
{{12} {13} {14} {15} {16} {17} {18} {19}}
12
13
14
15
16
17
18
19
{{22} {23} {24} {25} {26} {27} {28} {29}}
22
23
24
25
26
27
28
29
{{32} {33} {34} {35} {36} {37} {38} {39}}
32
33
34
35
36
37
38
39
sim:/ta

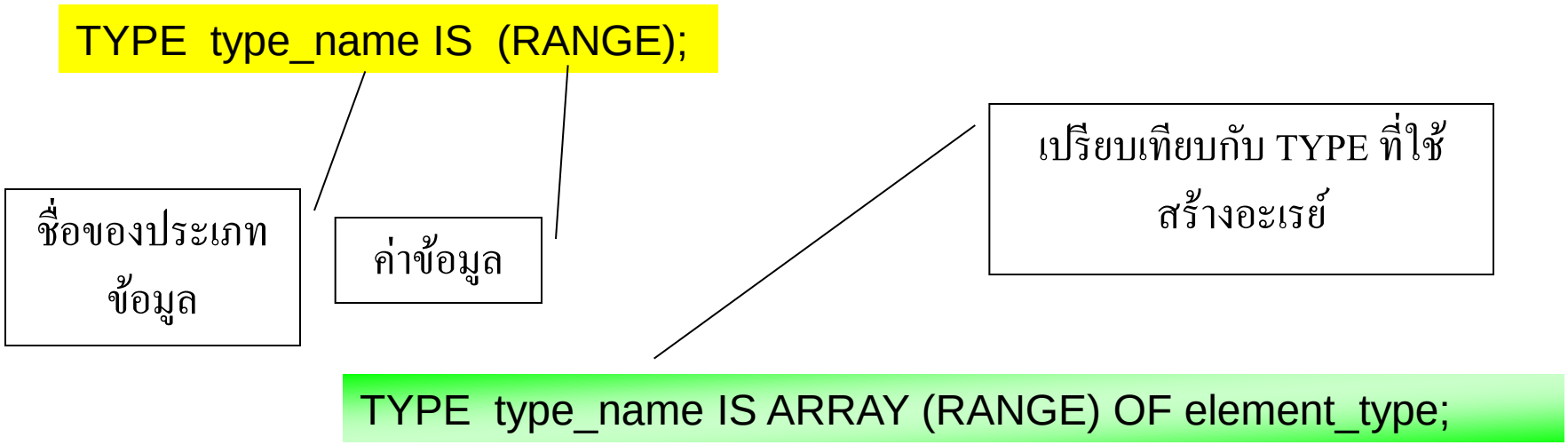
การแบ่งข้อมูลแบบอะเรย์ (Slices of Arrays)

บางครั้งการใช้อะเรย์ก็ต้องการเพียงสมาชิกบางตัวของอะเรย์เท่านั้น ลักษณะนี้เรียกว่า Slide of array

```
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;
```

ข้อมูลแบบแจกแจง (Enumeration Types)

ประเภทข้อมูลแบบนี้เป็นแบบที่ผู้ใช้กำหนดขึ้นเองเพื่อให้เหมาะสมกับงานที่กำลังทำอยู่ เช่นถ้าต้องการออกแบบระบบควบคุมไฟจราจร ถ้าต้องการความสะดวก การระบุว่าออบเจ็กต์ให้มีค่าเป็น แดงเหลือง เขียน ก็จะทำให้การออกแบบสะดวกขึ้น เพราะว่าเมื่อนำไปสังเคราะห์ VHDL จะแปลงเป็นค่าโลจิกที่เหมาะสมและครอบคลุมให้เอง



ตัวอย่าง

```
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 เมื่อนำไปสังเคราะห์ค่าเหล่านี้จะถูกเปลี่ยนให้เป็นค่าโลจิกที่เหมาะสมเช่นอาจเป็น 00 01 10 และ 11 ตามลำดับ

ตัวอย่างเป็นลักษณะของระบบที่เป็น 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;
```

ข้อมูลแบบ IEEE Standard Logic Type

IEEE ได้กำหนดแพกเก็จข้อมูลขึ้นใหม่เรียกว่า STD_LOGIC_1164 ซึ่งมีค่าได้ 9 ค่า ดังนี้

```
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

เปรียบเทียบ BIT กับ STD_ULOGIC

BIT เทียบกับ STD_ULOGIC

BIT_VECTOR เทียบกับ STD_ULOGIC_VECTOR

เวลาเรียกใช้ต้องระบุไลบรารีและแพกเก็จดังนี้

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

ข้อมูลประเภท Resolved and Unresolved Types

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

ต้องใช้ประเภทข้อมูลแบบที่มี Resolution Function ซึ่ง มีชื่อว่า STD_LOGIC และ STD_LOGIC_VECTOR แทน STD_ULOGIC และ STD_ULOGIC_VECTOR ตามลำดับ

Resolution Function

```
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;
```

Records

เป็นชนิดของข้อมูลที่เกิดจากกลุ่มของข้อมูลหลายๆประเภทที่ต่างชนิดกัน ข้อมูลเรคอร์ดนี้จะคล้ายกับเรคอร์ดในภาษาโปรแกรมทั่วไป รูปแบบการกำหนดเรคอร์ดมีดังนี้

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

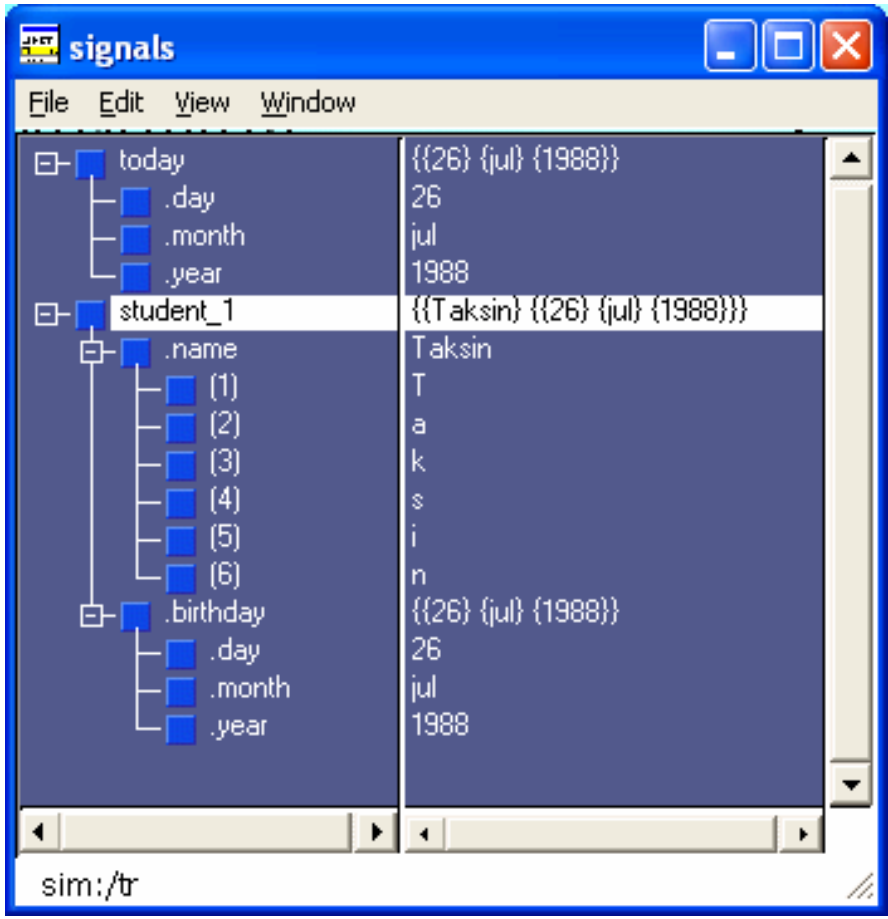
ตัวอย่างและผลการทำงานของ Record

```
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;
```



Subtypes

เป็นชนิดของข้อมูลย่อยของข้อมูลหลักที่มีอยู่แล้ว ต้องเป็นชนิดเดียวกับข้อมูลหลัก

รูปแบบการกำหนด

SUBTYPE subtype_name **IS** base_type **RANGE** range_constraint

```
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;
```

ชื่อแทน (Aliases)

เป็นการกำหนดชื่อใหม่ให้กับ object ที่มีอยู่แล้ว เพื่อทำให้ง่ายต่อการจัดการ เหมาะกับ Object ที่เป็นแบบผสม ข้อควรระวัง Aliases นี้ใช้ได้กับเครื่องมือที่ใช้สังเคราะห์บางตัวเท่านั้น

ALIAS new_name : Object

ภาษา C ใช้ #define เช่น

#define Water_Min PIN_A0

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;
```

DATA(9) = 0 STARTBIT
DATA(8) = 1
DATA(7) = 1
DATA(6) = 0
DATA(5) = 0 MESSAGE
DATA(4) = 0
DATA(3) = 1
DATA(2) = 1
DATA(1) = 1 PARITY
DATA(0) = 1 STOPBIT