บทที่ 3

ตัวดำเนินการและคำสั่งของภาษา VHDL (VHDL Operators and Statement)

3.1 ตัวดำเนินการและลำดับการทำงาน

ภาษา VHDL จะมีตัวดำเนินการสำหรับกระทำการสิ่งใดๆคล้ายๆกับตัวดำเนินการในภาษาสูงเช่น การบวก ลบ คูณ และ หาร แต่ในภาษา VHDL นั้นตัวดำเนินการเหล่านี้เป็นเครื่องมือช่วยในการสร้างโมเดลในระดับ RTL (Register Transfer Level) ตัวดำเนินการของ VHDL มีตามตารางที่ 4-1

ตารางที่ 3-1 ตัวดำเนินการของภาษา VHDL

	NOT					
Logical	AND	OR	NAND	NOR	XOR	XNOR
Relational	=	/=	<	<=	>=	>
Shift	SLL	SRL	SLA	SRA	ROL	ROR
	+	-				
	*	/	MOD	REM		
Arithmetic	**	ABS				

เช่นเดียวกับภาษาสูงอื่นๆ ที่มีการจัดแบ่งลำดับความสำคัญของตัวดำเนินการ ตัวดำเนินการในตาราง ที่ 4-1 มีการจัดเรียงความสำคัญจากต่ำไปหาสูง โดยเรียงจากบนลงล่างและ ในระดับเดียวกันจะทำงานจากตารางในคอลัมน์ทางซ้ายไปขวา นอกจากลำดับความสำคัญแล้ว จากตารางจะเห็นได้ว่ามีการจัดเรียงตัวดำเนินการเป็นหมวดหมู่ คือ การดำเนินการทางด้านโลจิก การเปรียบเทียบ การเลื่อน และการดำเนินการทางคณิตศาสตร์

3.2 ตัวดำเนินการแบบโลจิก

ตัวกระทำทางด้านโลจิกจะใช้กับข้อมูลประเภท

- BIT
- BIT_VECTOR
- STD_ULOGIC หรือ STD_LOGIC
- STD_ULOGIC_VECTOR หรือ STD_LOGIC_VECTOR

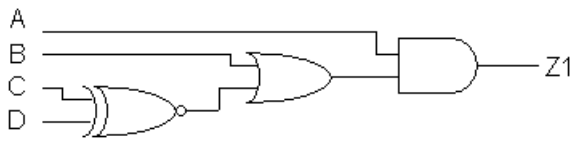
ตัวกระทำทุกตัวต้องเป็นประเภทเดียวกันและมีขนาดเท่ากันผลลัพธ์ที่ได้จะเหมือนกับตัวกระทำทั้งประเภทและขนาด

ตารางที่ 3-2 ตัวดำเนินการแบบโลจิก

ตัวดำเนินการ	ความหมาย	ประเภทข้อมูล	ผลลัพธ์
and	logical and	logical array or boolean	เหมือนตัวกระทำ
or	logical or	logical array or boolean	เหมือนตัวกระทำ
nand	logical complement of and	logical array or boolean	เหมือนตัวกระทำ
nor	logical complement of or	logical array or boolean	เหมือนตัวกระทำ
xor	logical exclusive or	logical array or boolean	เหมือนตัวกระทำ
xnor	logical complement of exclusive or	logical array or boolean	เหมือนตัวกระทำ

ตัวอย่าง

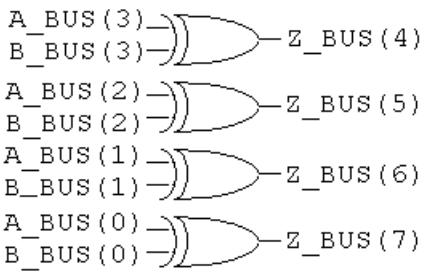
```
entity LOGIC_OP is
  port (A, B, C, D : in   bit;
        Z1:          out bit;
        EQUAL :    out boolean);
end LOGIC_OP;

architecture EXAMPLE of LOGIC_OP is
begin
  Z1 <= A  and (B or (C xnor D)));
  EQUAL <= A xor B;
end EXAMPLE;
```



ผิดเพราะว่าเป็นข้อมูลคนละประเภท
EQUAL เป็น Boolean แต่ A และ B เป็น bit

สำหรับตัวกระทำที่เป็นอะเรย์ เมื่อสังเคราะห์แล้วจะได้เป็นกลุ่มของโลจิกตามตัวอย่างต่อไปนี้

```
architecture EXAMPLE of LOGICAL_OP is
  signal A_BUS, B_BUS : bit_vector (3 downto 0);
  signal Z_BUS :        bit_vector (4 to 7);
begin
  Z_BUS <= A_BUS XOR B_BUS;

end EXAMPLE;
```



สำหรับตัวอย่างนี้ได้แสดงถึงผลการเรียงดรรชนีของอะเรย์แบบต่างๆด้วย

3.3 ตัวดำเนินการสำหรับการเลื่อน (Shift Operators)

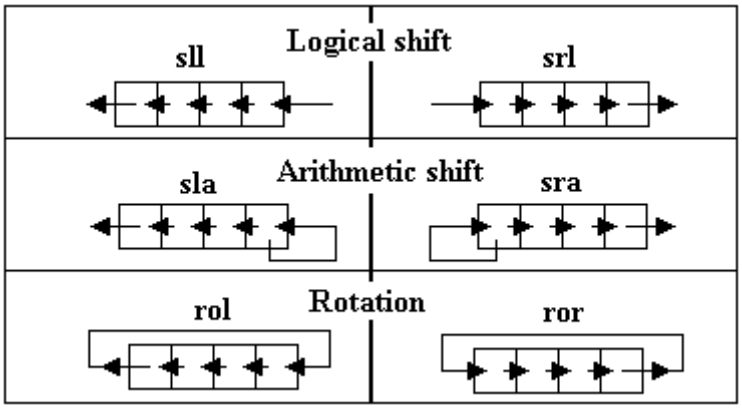
ตัวกระทำสำหรับการเลื่อนจะใช้กับข้อมูลประเภท

- BIT_VECTOR
- BOOLEAN

ทั้งนี้ความยาวของตัวรับและส่งต้องเท่ากัน ผลลัพธ์ที่ได้จะเหมือนกับตัวกระทำทั้งประเภทและขนาด

ตารางที่ 3-3 ตัวดำเนินการสำหรับการเลื่อน

ตัวดำเนินการ	ความหมาย	ประเภทข้อมูล	ผลลัพธ์
sll	shift left logical	logical array sll integer	เหมือนตัวกระทำ
srl	shift right logical	logical array srl integer	เหมือนตัวกระทำ
sla	shift left arithmetic	logical array sla integer	เหมือนตัวกระทำ
sra	shift right arithmetic	logical array sra integer	เหมือนตัวกระทำ
rol	rotate left	logical array rol integer	เหมือนตัวกระทำ
ror	rotate right	logical array ror integer	เหมือนตัวกระทำ



รูปที่ 3-1 การทำงานของการเลื่อนและการหมุนแบบต่างๆ

ตัวอย่าง

```
architecture beh of Logic_shift is
  signal A : bit_vector(3 downto 0) := "1000";
  signal B : bit_vector(3 downto 0) := "1110";
  signal YA, YB, YC : bit_vector(3 downto 0);
begin
  YA <= A SRL 3;
  YB <= B ROL 1;
  YC <= A SLL 2;
end beh;
```

ผลการทำงาน

A = "1000"

B = "1110"

YA = "0001"

YB = "1101

YC = "0000"

3.4 ตัวดำเนินการเปรียบเทียบ (Relational Operators)

ตัวกระทำทางด้านการเปรียบเทียบจะใช้กับข้อมูลประเภทต่างๆดังนี้

- ข้อมูลแบบ Scalar ทุกประเภท
- ข้อมูลแบบอะเรย์ขนาด 1 มิติ
- STD_ULOGIC หรือ STD_LOGIC
- STD_ULOGIC_VECTOR หรือ STD_LOGIC_VECTOR

ตัวกระทำทุกตัวต้องเป็นประเภทเดียวกันแต่จะมีขนาดเท่ากันหรือไม่ก็ได้ ในกรณีที่เป็นอะเรย์ ถ้า มีขนาดไม่เท่ากัน การเปรียบเทียบจะเปรียบเทียบจากข้อมูลทางซ้ายไปขวาเสมอ ผลลัพธ์ที่ได้จะให้ค่าเป็น จริง กับเท็จ

ตารางที่ 3-4 ตัวดำเนินการสำหรับการเปรียบเทียบ

ตัวดำเนินการ	ความหมาย	ผลลัพธ์
=	เปรียบเทียบ เท่ากับ	boolean
/=	เปรียบเทียบ ไม่เท่ากับ	boolean
<	เปรียบเทียบ น้อยกว่า	boolean
<=	เปรียบเทียบ น้อยกว่าหรือเท่ากับ	boolean
>	เปรียบเทียบ มากกว่า	boolean
>=	เปรียบเทียบ มากกว่าหรือเท่ากับ	boolean

ตัวอย่างการเปรียบเทียบข้อมูลแบบอะเรย์ที่มีความยาวไม่เท่ากัน

```
architecture beh of Vector_relational is
  signal A : bit_vector(7 downto 0) := "00001010";
  signal B : bit_vector(3 downto 0) := "1110";
  signal YA: bit;
begin
  COMPARE: process (A, B)
    begin
      if (A < B) then
        YA <= '1';
      else
        YA <= '0';
    END IF;
  END process;
end beh;
```

เมื่อข้อมูล A = "00001010" และข้อมูล B = "1110" ความยาวไม่เท่ากัน การเปรียบเทียบเหมือนกับจัดเรียงข้อมูลชิดซ้ายก่อนคือจะได้เป็น A = "00001010"
และ B = "11100000"
ดังนั้น A น้อยกว่า B ผลจึงได้ Y = '1'
แต่ทั้งนี้มิได้หมายความว่า B จะกลายเป็น "11100000" B ยังคงค่าเดิมคือ "1110"

ในทำนองเดียวกันถ้า A และ B เป็น Integer ถึงแม้ว่าการกำหนดขนาดไม่เท่ากันก็ยังคงเปรียบเทียบกันได้ ดังตัวอย่าง

```
architecture beh of Int_relational  is
  signal A : integer range 0 to 255 := 12;
  signal B : integer range 0 to 15 := 14;
  signal YA: bit;
begin
  COMPARE: process (A, B)
    begin
      if (A < B) then
        YA <= '1';
      else
        YA <= '0';
      END IF;
    END process;
end beh;
```

ขนาดของ A และ B ไม่เท่ากัน แต่ยังคงเปรียบเทียบกันได้

3.5 ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

เหมือนกับตัวดำเนินการทางคณิตศาสตร์ในภาษาระดับสูงทั่วไป ตัวกระทำจะใช้กับข้อมูลประเภทต่างๆ ได้ดังนี้

- Integer
- Real
- ประเภทกายภาพเช่น Time

ตัวกระทำทุกตัวต้องเป็นประเภทเดียวกัน ผลลัพธ์ที่ได้เป็นข้อมูลประเภทเดียวกัน

ตารางที่ 3-5 ตัวดำเนินการทางคณิตศาสตร์

ตัว ดำเนินการ	ความหมาย	ประเภทข้อมูล	ผลลัพธ์
**	exponentiation	numeric ** integer	numeric
Abs	absolute value	abs numeric	numeric
Not	complement	not logic or boolean	เหมือนตัวกระทำ
*	multiplication	numeric * numeric	numeric
/	division	numeric / numeric	numeric
Mod	modulo	integer mod integer	integer
Rem	remainder	integer rem integer	integer
+	unary plus	+ numeric	numeric
-	unary minus	- numeric	numeric
+	addition	numeric + numeric	numeric
-	subtraction	numeric - numeric	numeric
&	concatenation	array or element & array or element	array

3.5.1 ตัวดำเนินการเกี่ยวกับเศษ (Remainder and Modulo)

REM (Remainder)

ความหมาย A rem B = เศษของ(A/B)

เครื่องหมายเหมือน A เช่น

5 rem 3 = 2

-5 rem 3 = -2

-5 rem –3 = -2

5 rem –3 = 2

MOD (Modulo)

A mod B มี 2 กรณี

<u>กรณีที่1</u> *A และ B เครื่องหมายเหมือนกัน*

A mod B = เศษของ(A/B)

เครื่องหมายเหมือน A หรือ B เช่น

5 mod 3 = 2

-5 mod –3 = -2

<u>กรณีที่2</u> *A และ B เครื่องหมายต่างกัน*

A mod B = เศษของ(BxN – A)

เครื่องหมายเหมือน B และ N คือเลขจำนวน

เต็ม เมื่อคูณกับ B แล้วค่าเท่ากับหรือมากกว่า A

เช่น

-5 mod 3 = +(3*2 - 5) = 1

5 mod –3 = -(3*2 - 5) = -1

3.6 คำสั่งแบบขนาน (Concurrent Statement)

ตามที่ได้กล่าวมาแล้วว่าการทำงานมี 2 แบบ แบบแรกเป็นคำสั่งแบบขนานซึ่งลักษณะการทำงานตรงกับการทำงานทางวงจรไฟฟ้า คือทุกๆ คำสั่งทำงานไปพร้อมๆ กัน ภาษา VHDL มีคำสั่งที่มีการทำงานแบบขนานหลายคำสั่งได้แก่

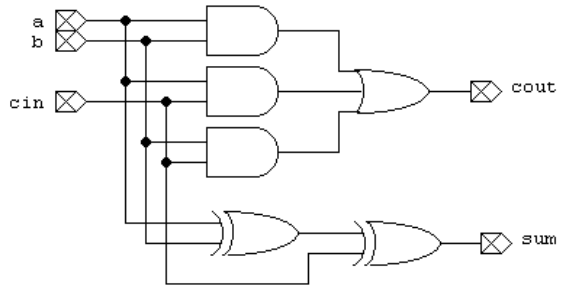
- การกำหนดค่าสัญญาณ (Signal Assignment)
- When Statement
- With Statement
- Process Statement

3.6.1 การกำหนดค่าสัญญาณแบบขนาน (Concuurrent Signal Assignment)

การกำหนดค่าให้กับสัญญาณหรือที่เรียกว่า Signal Assignment เป็นคำสั่งที่มีการทำงานเป็นแบบขนาน สามารถจะเขียนไว้ที่ใดก็ได้ภายใน Architecture รวมถึงการเขียนไว้ใน PROCESS ด้วย รูปแบบคำสั่งเป็นดังนี้

```
Signal <= expression
```

ตัวอย่างการเขียนวงจร FULL ADDER ของวงจรในรูป 3-2 ด้วยการกำหนดค่าสัญญาณแบบขนาน



รูปที่ 3-2 วงจร Full Adder

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fulladd is
  port (a, b, cin : in std_logic;
        sum, cout : out  std_logic);
end fulladd;
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;
```

จากตัวอย่างเมื่อพิจารณารูปแบบของคำสั่ง sum <= (a xor b xor cin); กับ cout <= (a and b) or (a and cin) or (b and cin); การทำงานของสัญญาณ sum กับ cout จะเกิดขึ้นพร้อมกัน ดังนั้นทั้งสองบรรทัดนี้จะเขียนบรรทัดใดก่อนก็ให้ผลเหมือนกัน นอกจากนี้แล้ว ยังสามารถใส่ค่าเวลาหน่วง (Delay time) ของวงจรได้ เช่นถ้าต้องการให้สัญญาณ sum เกิดขึ้นหลังจากการ xor กันแล้ว 5 ns และสัญญาณ cout ก็เช่นกัน สามารถเขียนได้ดังนี้

```
sum <= (a xor b xor cin)  after 5 ns;
cout <= (a and b) or (a and cin) or (b and cin) after 5 ns;
```

3.6.2 การใช้คำสั่ง When

เป็นคำสั่งแบบตรวจสอบเงื่อนไขที่เป็นนิพจน์บูลีน เงื่อนไขของคำสั่งนี้อาจเลื่อมกัน แต่ต้องมีการจัดลำดับความสำคัญ

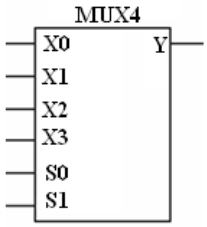
รูปแบบคำสั่ง

```
TARGET <= VALUE;
TARGET <= VALUE_1 when CONDITION_1 else
          VALUE_2 when CONDITION_2 else
          . . .
          VALUE_n;
```

ตัวอย่างนี้เป็นการตรวจสอบค่า X ถ้า X เท่ากับ "1111" ให้สัญญาณจาก B แก่สัญญาณ Z_CONC แต่ถ้าไม่ใช้ให้ตรวจสอบว่า X มากกว่า "1000" หรือไม่ ถ้าใช้ให้สัญญาณจาก C แก่ Z_CONC ถ้าไม่ใช้ให้สัญญาณ A แก่ Z_CONC

```
entity CONDITIONAL_ASSIGN is
        port (A, B, C, X : in   bit_vector (3 downto 0);
                Z_CONC : out bit_vector (3 downto 0);
        Z_SEQ    : out bit_vector (3 downto 0));
end CONDITIONAL_ASSIGNMENT;

architecture exam1 of CONDITIONAL_ASSIGN is
begin   Z_CONC <= B  when X = "1111" else
                  C  when X > "1000" else
                  A;
end exam1
```

อีกตัวอย่างเป็นการออกแบบวงจรมัลติเพลกซ์แบบเข้า 4 ออก 1 ตามบล็อกไดอะแกรมในรูปที่ 3-3



รูปที่ 3-3 บล็อกไดอะแกรมของวงจรมัลติเพลกซ์ 4 ทาง

สามารถเขียนเป็นภาษา VHDL ได้ดังนี้

```
entity MUX4 is
        generic (width: integer := 4);
        port ( x : in std_logic_vector (width-1 downto 0);
               s : in std_logic_vector (1 downto 0);
               Y : out std_logic );
end MUX4;
architecture BEH of MUX4 is
begin
        Y <=    X(0) when selector = "00" else
                X(1) when selector = "01" else
                X(2) when selector = "10" else
                X(3);
end BEH;
```

3.6.3 การใช้คำสั่ง With

เป็นคำสั่งแบบตรวจสอบเงื่อนไขที่เป็นนิพจน์บูลีน เหมือนคำสั่ง WHEN แต่เงื่อนไขของคำสั่งต้องไม่เลื่อมกัน และครอบคลุมทุกตัวเลือก โดยเงื่อนไขอาจบอกค่าด้วยวิธีการดังต่อไปนี้ได้

- บอกค่าเดี่ยว
- บอกเป็นย่านหรือกลุ่มของค่า
- สามารถใช้การเลือกค่าได้ด้วย ("|" หมายถึง "or")

- "when others" ใช้เพื่อให้ครอบคลุมทุกตัวเลือก

รูปแบบกำหนดได้ดังนี้

```
with EXPRESSION select
      TARGET <=      VALUE_1 when CHOICE_1,
                     VALUE_2 when CHOICE_2 | CHOICE_3,
                     VALUE_3 when CHOICE_4 to CHOICE_5,
                     . . .
                     VALUE_n when others;
```

ตัวอย่างแรกเป็นการตรวจสอบค่า X เพื่อกำหนดสัญญาณให้ Z_CONC ตามเงื่อนไขดังนี้

ถ้า X เท่ากับ 0 สัญญาณ Z_CONC ได้จาก A

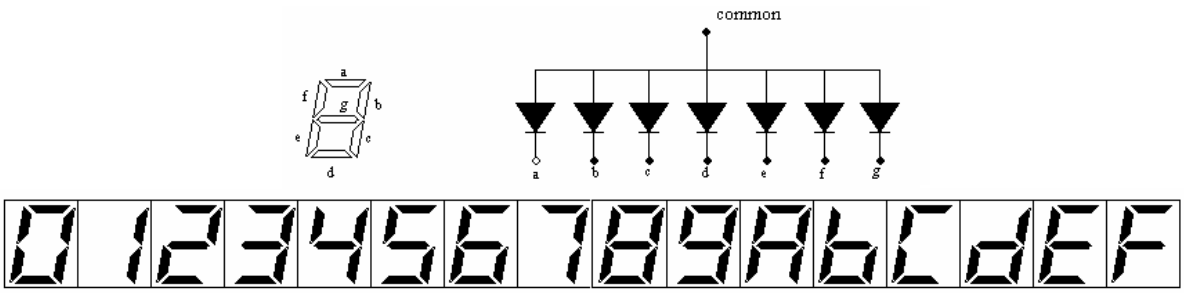
ถ้า X เท่ากับ 7 หรือ 9 สัญญาณ Z_CONC ได้จาก B

ถ้า X เท่ากับ 1 ถึง 5 สัญญาณ Z_CONC ได้จาก C

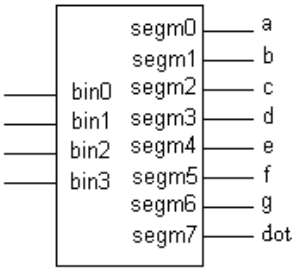
นอกจากนั้นให้สัญญาณ Z_CONC เป็น '0'

```
entity SELECTED_ASSIGNMENT is
        port (A, B, C, X : in   integer range 0 to 15;
        Z_CONC : out integer range 0 to 15;
        Z_SEQ   : out integer range 0 to 15);
end SELECTED_ASSIGNMENT;

architecture EX of SELECTED_ASSIGNMENT is
begin
  -- Concurrent version of selected signal assignment
  with X select
    Z_CONC <= A when 0,
                  B when 7 | 9,
                  C when 1 to 5,
                  0 when others;
end EX
```

ตัวอย่าง วงจรถอดรหัส 7 – segment สำหรับ LED 7 Segment แบบ Common Anode



รูปที่ 3-4 (ก) รูปแบบการแสดงตัวเลข



รูปที่ 3-4 (ข) บล็อกไดอะแกรมของวงจรถอดรหัส

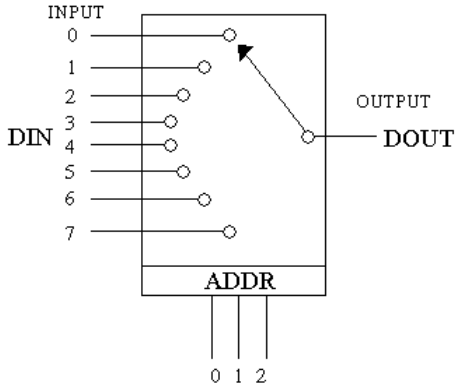
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity segdec is
          port (bin : in std_logic_vector(3 downto 0);
               segm : out std_logic_vector(7 downto 0));
end segdec;

architecture Behavioral of segdec is
begin
  with bin select
   segm <=      "00000011" when "0000",
                "10011111" when "0001",
                "00100101" when "0010",
                "00001101" when "0011",
                "10011001" when "0100",
                "01001001" when "0101",
                "01000001" when "0110",
                "00011111" when "0111",
                "00000001" when "1000",
                "00001001" when "1001",
                "00010001" when "1010",
                "11000001" when "1011",
                "01100011" when "1100",
                "10000101" when "1101",
                "01100001" when "1110",
                "01110001" when others;
end Behavioral;
```

ตัวอย่าง วงจร 8 channel Multiplexer MUX8 เป็นวงจรเลือกสัญญาณจากอินพุทส่งออกทางเอาท์พุท โดย ADDR จะเป็นตัวเลือกให้สัญญาณ DIN ใดออกที่เอาท์พุท DOUT

DIN เป็นบัสอินพุทขนาด 8 บิต

ADDR เป็นบัสควบคุมขนาด 3 บิต

DOUT เป็นสัญญาณเอาท์พุท



ADDR			Output
2	1	0	DOUT
0	0	0	DOUT = DIN 0
0	0	1	DOUT = DIN 1
0	1	0	DOUT = DIN 2
0	1	1	DOUT = DIN 3
1	0	0	DOUT = DIN 4
1	0	1	DOUT = DIN 5
1	1	0	DOUT = DIN 6
1	1	1	DOUT = DIN 7

รูปที่ 3-5 บล็อกไดอะแกรมของวงจรเลือกข้อมูล 8 ทางและตารางการทำงาน

```
---------------------------------------
--    The multiplex 8 bits       --
---------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity mux8 is
          port (din  : in bit_vector(7 downto 0);
                addr : in bit_vector(2 downto 0);
                dout : out bit);
end mux8;
architecture mux8_arc of mux8 is
begin
  with addr select
     dout <= din(0) when B"000",
                 din(1) when B"001",
                 din(2) when B"010",
                 din(3) when B"011",
                 din(4) when B"100",
                 din(5) when B"101",
                 din(6) when B"110",
                 din(7) when B"111";
end mux8_arc;
```

ตัวอย่างสุดท้ายวงจรมัลติเพลกซ์แบบเข้า 4 ออก 1 ตามรูป 3-3 คล้ายกับวงจรในหัวข้อ คำสั่ง WHEN

```
entity MUX4 is
          generic (width: integer := 4);
          port ( x : in std_logic_vector (width-1 downto 0);
                 s : in std_logic_vector (1 downto 0);
                 Y : out std_logic );
end MUX4;

architecture BEH_WITH of MUX4 is
 begin
          with s select
           Y <= X(0) when "00",
           Y <= X(1) when "01",
           Y <= X(2) when "10",
           Y <= X(3) when "11",
end BEH_WHIT;
```

3.7 คำสั่งแบบลำดับ (Sequential Statement)

จากที่ได้กล่าวมาแล้วว่าการทำงานของภาษา VHDL นั้นเป็นการทำงานเหมือนฮาร์ดแวร์ ดังนั้นคำสั่งทุกคำสั่งภายใน Architecture จะถือว่าทำงานไปพร้อมๆกัน ไม่ขึ้นอยู่กับลำดับที่ ที่คำสั่งนั้นอยู่ แต่ถ้าต้องการเขียนคำสั่งเหมือนกับการเขียนโปรแกรมทั่วไปที่มีการทำงานเป็นลำดับก็สามารถทำได้ แต่คำสั่งเหล่านั้นต้องอยู่ในคำสั่ง Process อีกทีหนึ่ง คำสั่งที่มีการทำงานเป็นลำดับในภาษา VHDL มีดังนี้

- คำสั่ง IF
- คำสั่ง CASE
- คำสั่ง FOR Loop
- คำสั่ง WHILE Loop
- คำสั่ง WAIT
- คำสั่ง NULL

3.7.1 คำสั่ง Process (Process Statement)

Process เป็นคำสั่งที่การทำงานเป็นแบบขนาน แต่ที่นำมากล่าวในหัวข้อของคำสั่งแบบลำดับก็เพราะว่าภายในคำสั่ง Process เป็นคำสั่งแบบลำดับเท่านั้น ดังนั้นการจะใช้คำสั่งแบบลำดับคำสั่งใด คำสั่งเหล่านั้นต้องอยู่ภายในคำสั่ง Process

แนวคิดของคำสั่ง Process ก็ได้มาจากการเขียนโปรแกรมทั่วๆไปคือคำสั่งภายใน Process การทำงานจะเป็นแบบลำดับ ใน Architecture หนึ่งสามารถมี Process ได้หลายๆ Process แต่ทุกๆ Process จะทำงานขนานกัน รูปแบบคำสั่ง Process มีดังนี้

```
[Process_name:] PROCESS [(sensitivity_list)]
        [Process_declarative_part]
BEGIN
        Sequential_statements
END PROCESS [process_name];
```

Process_name เป็นชื่อของ Process จะมีหรือไม่มีก็ได้ ถ้าทีต้องตามหลังด้วยโคลอน ":" และถ้ามีหลาย Process ชื่อนี้ต้องไม่ซ้ำกัน

Sensitivity_list เป็นสัญญาณที่มีผลกระทบต่อการทำงานภายใน Process จะมีหรือไม่มีก็ได้

Sequential_statements คำสั่งแบบลำดับที่ใช้ใน Process

ตัวอย่าง Process ที่มี Sensitivity_list

```
Sync_process:  PROCESS (CLK)
BEGIN
        Wait until clk = '0';
        sum <= a XOR b XOR cin;
END PROCESS;
```

ตัวอย่าง Process ที่ไม่มี Sensitivity_list

```
D_Flipflop: PROCESS
BEGIN
        wait until (clk 'EVENT and CLK =  '1');
        Q <= D;
END PROCESS;
```

3.7.2 ตัวแปร (Variables)

ในหัวข้อ 2.6.2 ได้เคยกล่าวถึงตัวแปรและสัญญาณไว้แล้วแต่เนื่องจากการใช้คำสั่งแบบลำดับมักมีการใช้ตัวแปรช่วยในการเขียนโปรแกรมมาก ดังนั้นจึงขอทบทวนเกี่ยวกับตัวแปรและสัญญาณก่อน

ตัวแปรหรือ Variable ใช้ได้เฉพาะภายในคำสั่ง PROCESS เท่านั้นและเมื่อใช้ใน Processs ใด ก็จะมองเห็นได้จาก Process นั้นเท่านั้น จะไม่เห็นจาก Process อื่น ยกเว้น SHARE VARIABLE ซึ่งถูกนิยามขึ้นใหม่ใน VHDL'93 ซึ่งมีวิธีระบุดังนี้

```
SHARE VARIABLE  shareed_variable_name : type
```

การส่งผ่านค่าระหว่า Variable กับ Signal สามารถทำได้ แต่ใช้เค รื่องหมายต่างกัน ถ้า ตัวรับเป็น Variable ใช้เครื่องหมาย := แต่ถ้าตัวรับเป็น Signal ใช้ <= ตามตัวอย่างดังนี้

```
architecture RTL of XYZ is
    signal A_sig, B_sig, C_sig : integer range 0 to 7;
    signal Y_sig, Z_sig :     integer range 0 to 15;
begin
    process (A_sig, B_sig, C_sig)
      variable M_var, N_var : integer range 0 to 7;
    begin
      M_var := A_sig;
      N_var := B_sig;
      Z_sig <= M_var + N_var;
      M_var := C_sig;
      Y_sig <= M_var + N_var;
    end process;
end RTL;
```

3.7.3 การใช้คำสั่ง IF (IF Statement)

เป็นคำสั่งที่ใช้สำหรับเลือกการทำงานตามเงื่อนไขที่ระบุ โดยเงื่อนไขอยู่ในรูปนิพจน์บูลีน คำสั่ง IF มีหลายรูปแบบดังนี้

รูปแบบที่ 1

```
IF condition THEN
  -- sequential statements
END IF;
```

รูปแบบที่ 2

```
IF condition THEN
  -- sequential statements
ELSE
  -- sequential statements
END IF;
```

รูปแบบที่ 3

```
IF condition THEN
  -- sequential statements
ELSIF CONDITION then  -- ระวัง ELSIF ไม่ใช้ ELSE IF
  -- sequential statements
. . .
ELSE
  -- sequential statements
END IF;
```

ตัวอย่าง

```
entity IF_STATEMENT is
  port (A, B, C, X : in   bit_vector (3 downto 0);
        Z          : out bit_vector (3 downto 0);
end IF_STATEMENT;
```

```
architecture EXAMPLE1 of IF_STATEMENT is
begin
  process (A, B, C, X)
  begin
    Z <= A;
    if (X = "1111") then
      Z <= B;
    elsif (X > "1000") then
      Z <= C;
    end if;
  end process;
end EXAMPLE1;
```

```
architecture EXAMPLE2 of IF_STATEMENT is
begin
  process (A, B, C, X)
  begin
    if (X = "1111") then
      Z <= B;
    elsif (X > "1000") then
      Z <= C;
    else
      Z <= A;
    end if;
  end process;
end EXAMPLE2;
```

ตัวอย่างการเขียนวงจร MUX แบบ 4 อินพุท ตามรูป 3-3 โดยใช้คำสั่ง If...Else

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
        port (  a, b, c, d  : in std_logic_vector (7 downto 0);
                sel1, sel2 : in std_logic;
                outmux     : out std_logic_vector (7 downto 0));
end mux4;
architecture behavior of mux4 is
begin
        process (a, b, c, d, sel1, sel2)
        begin
                if (sel1 = '1') then
                        if (sel2 = '1') then
                                outmux <= a;
```

```
                        else
                                outmux <= b;
                        end if;
                else
                        if (sel2 = '1') then
                                outmux <= c;
                        else
                                outmux <= d;
                        end if;
                end if;
        end process;
end behavior;
```

3.7.4 การใช้คำสั่ง CASE (CASE Statement)

การใช้งานคำสั่ง CASE คล้ายกับคำสั่ง IF แต่ expression เพื่อใช้ในการเลือกของคำสั่ง CASE มีความสำคัญเท่ากัน การทำงานของคำสั่ง CASE นี้ คล้ายกับคำสั่ง WITH SELECT ในแบบ ขนาน รูปแบบคำสั่ง มีดังนี้

```
CASE  expression  IS
 WHEN value_1  =>
        -- sequential statements
 WHEN value_2 | value_3  =>
        -- sequential statements
 WHEN value_4 to value_n  =>
        -- sequential statements
 WHEN  others =>
        -- sequential statements
END CASE ;
```

เงื่อนไขการเลือกมีดังนี้

- ค่าของตัวเลือกต้องไม่เลื่อมกัน
- ระบุค่าตัวเลือกเป็นค่าใดค่าหนึ่งที่แน่นอน หรือ
- ระบุค่าตัวเลือกเป็นกลุ่ม โดยใช้คำสั่ง TO
- ระบุค่าตัวเลือกจากค่าใดค่าหนึ่งจากตัวเลือกหลายๆตัว โดยใช้เครื่องหมาย "|" ซึ่งมีความหมายหมายถึง "หรือ"
- ต้องกำหนดค่าตัวเลือกให้ครอบคลุมทุกกรณี โดยอาจใช้คำสั่ง "WHEN OTHERS"

ตัวอย่าง

```
entity CASE_STATEMENT is
  port (A, B, C, X : in   integer range 0 to 15;
        Z          : out integer range 0 to 15;
end CASE_STATEMENT;

architecture EXAMPLE of CASE_STATEMENT is
begin
  process (A, B, C, X)
  begin
    case  X  is
      when  0  =>       Z <= A;
```

```
            when  7  |  9  =>          Z <= B;
            when  1 to 5  =>          Z <= C;
            when others =>            Z <= 0;
          end case;
        end process;
      end EXAMPLE;
```

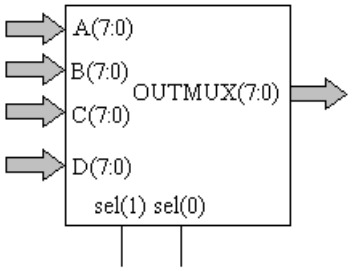
ตัวอย่างการระบุค่าตัวเลือกแบบระบุย่าน

```
entity RANGE_1 is
port (A, B, C, X : in   integer range 0 to 15;
      Z          : out integer range 0 to 15;
end RANGE_1;
architecture EXAMPLE of RANGE_1 is
begin
  process (A, B, C, X)
  begin
    case X is
      when 0 =>
        Z <= A;
      when 7 | 9 =>
        Z <= B;
      when 1 to 5 =>
        Z <= C;
      when others =>
        Z <= 0;
    end case;
  end process;
end EXAMPLE;
```

```
entity RANGE_2 is
port (A, B, C, X : in    bit_vector(3 downto 0);
      Z          : out  bit_vector(3 downto 0);
end RANGE_2;
 architecture EXAMPLE of RANGE_2 is
begin
  process (A, B, C, X)
  begin
    case X is
      when "0000" =>
        Z <= A;
      when "0111" | "1001" =>
        Z <= B;
      when "0001" to "0101" =>    -- ผิด เพราะว่าเป็น BIT
        Z <= C;
      when others =>
        Z <= 0;
    end case;
  end process;
end EXAMPLE;
```

ตัวอย่างการเขียนวงจร MUX แบบ 4 อินพุท อินพุทละ 8 บิต โดยใช้คำสั่ง case

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
        port (a, b, c, d : in std_logic_vector (7 downto 0);
                sel : in std_logic_vector (1 downto 0);
                outmux : out std_logic_vector (7 downto 0));
end mux4;
architecture behavior of mux4 is
begin
        process (a, b, c, d, sel)
        begin
                case sel is
                        when "00" => outmux <= a;
                        when "01" => outmux <= b;
                        when "10" => outmux <= c;
                        when others => outmux <= d; -- case statement must be complete
                end case;
        end process;
end behavior;
```



3.7.5 การใช้คำสั่ง NULL (NULL Statement)

คำสั่ง NULL ในภาษา VHDL หมายถึงไม่กระทำการใดๆ มักใช้ในคำสั่ง CASE เนื่องจากคำสั่ง CASE ต้องมีทางเลือกครบทุกเงื่อนไข แต่ถ้าบางครั้งไม่ต้องการให้ทำอะไรเลยในบางเงื่อนไข ก็ใช้คำสั่ง NULL เข้ามาช่วย เช่น

```
Process(control)
begin
        Motor <= motor_stop;
        case control is
                when forward =>         motor <= motor_forward;
                when reverse  =>        motor <= motor_reverse;
                when others   =>        null;
        end case;
end process;
```

ตามตัวอย่างนี้ถ้า control ไม่ใช่ forward และไม่ใช่ reverse motor จะยังคงมีค่าเท่ากับ motor_stop

3.7.6 การใช้คำสั่ง Attribute ของออบเจ็กต์

การใช้ attribute หมายถึงการอ้างอิงถึงคุณสมบัติต่างๆของวัตถุ เช่นการใช้ 'event ซึ่งเป็น attribute สำหรับตรวจสอบการเปลี่ยนแปลของวัตถุเช่น clk'event ตรวจสอบว่าสัญญาณ clk นี้มีการเปลี่ยนแปลงหรือไม่ attribute มีอยู่หลายคำสั่งมาก ในที่นี้จะยกตัวอย่างที่ใช้งานบ่อยมาดังนี้

‘Left	ให้ค่าตัวชี้อีลิเมนต์(Element) ที่อยู่ซ้ายสุดของออบเจ็กต์
‘Right	ให้ค่าตัวชี้อีลิเมนต์(Element) ที่อยู่ขวาสุดของออบเจ็กต์
‘High	ให้ค่าสูงสุดของจำนวน array
‘Low	ให้ค่าต่ำสุดของจำนวน array
‘Length	ให้ค่าความยาวของชนิดข้อมูล
‘RANGE	ให้ค่าย่านหรือขอบเขตของออบเจ็กต์
‘REVERSE_RANGE	ให้ค่าย่านหรือขอบเขตของออบเจ็กต์แบบกลับทิศทาง
‘EVENT	ให้ค่าเป็นจริงเมื่อออบเจ็กต์มีการเปลี่ยนแปลงค่า

ตัวอย่างการใช้งาน

Signal y_up : std_logic_vector (15 downto 0)

Signal y_dn : std_logic_vector (0 to 15)

ถ้าหา attribute จะได้ค่าดังนี้

```
y_up'left           =       15
y_up'right          =       0
y_up'high           =       15
y_up'low            =       0
y_up'length         =       16
y_up'range          =       15 downto 0
y_up'reverse_range  =       0 to 15
```

```
y_dn'left           =       0
y_dn'right          =       15
y_dn'high           =       15
y_dn'low            =       0
y_dn'length         =       16
y_dn'range          =       0 to 15
y_dn'reverse_range  =       15 downto 0
```

ข้อควรระวังสำหรับการใช้งาน attributes บางตัวสามารถนำไปสังเคราะห์ได้ แต่บางตัวสังเคราะห์ไม่ได้ ทั้งนี้ขึ้นอยู่กับเครื่องมือที่ใช้งานด้วย

3.8 การใช้คำสั่ง Loop

คำสั่งสำหรับให้เกิดการทำงานแบบวนรอบ ในภาษา VHDL มี 2 แบบได้แก่

- FOR loop
- WHILE loop

3.8.1 คำสั่ง FOR Loop

รูปแบบมีดังนี้

```
[loop_label:]
FOR identifier IN discrete_range LOOP
  -- sequential statements
END LOOP  [loop_label] ;
```

Loop_label เป็นชื่อของ loop จะมีหรือไม่มีก็ได้

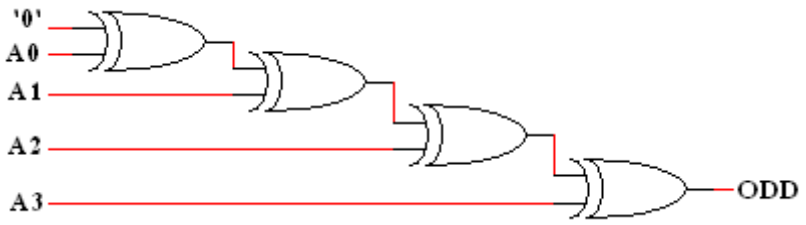
Identifier เป็นพารามิเตอร์ที่ใช้กำหนดจำนวนรอบของการทำงาน ค่านี้ไม่ต้องมีการกำหนดไว้ก่อน แต่จะกำหนดในคำสั่ง FOR เลยว่าต้องการให้มีค่าเท่าไรถึงเท่าไร และในแต่ละรอบของการทำงาน ค่านี้จะเพิ่มขึ้นที่ละ 1 เมื่อเพิ่มจนถึงค่าสูงสุดก็จะหลุดออกมาจากลูป และค่านี้จะเปลี่ยนค่าด้วยการใช้งานอย่างอื่นไม่ได้ อีกทั้งนอกลูปก็มองไม่เห็นด้วย

Discrete_rang ค่าของ Identifier โดยระบุเป็น ย่าน เช่น 0 TO 4

ตัวอย่าง

```
entity FOR_LOOP is
  port (A : in    integer range 0 to 3;
        Z  : out bit_vector (3 downto 0));
end FOR_LOOP;

architecture EXAMPLE of FOR_LOOP is
begin
  process (A)
  begin
    Z <= "0000";
    for  I  in  0 to 3  loop
      if (A = I) then
        Z(I) <= `1`;
      end if;
    end loop;
  end process;
end EXAMPLE;
```

ตัวอย่างการหาค่าพาริตี้ (Parity) ของข้อมูลขนาด 4 บิต โดยใช้คำสั่ง FOR LOOP ถ้าเป็นพาริตี้คี่ ODD จะเท่ากับ 1 การทำงานเป็นไปตามรูปที่ 3-6



รูปที่ 3-6 วงจรคำนวณหาค่าพาริตี้

```
entity PARITY is
     port (A: in   bit_vector (3 downto 0);
           ODD : out bit);
end PARITY;
architecture RTL of PARITY is
begin
    process (A)
      variable TEMP : bit;
    begin
      TEMP := `0`;

      for i in  A`low to A`high  loop
        TEMP := TEMP xor A(i);
      end loop;
      ODD <= TMP;
    end process;
end RTL;
```

ตัวอย่างการใช้ค่า Identifier ของคำสั่ง FOR

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity countzeros is
        port (a : in std_logic_vector (7 downto 0);
                Count : out std_logic_vector (2 downto 0) );
end mux4;
architecture behavior of mux4 is
        signal Count_Aux: std_logic_vector (2 downto 0);
begin
        process (a)
        begin
                Count_Aux <= "000";
                for i in a'range loop
                        if (a[i] = '0') then
                                Count_Aux <= Count_Aux + 1;
                        end if;
                end loop;
                Count <= Count_Aux;
        end process;
end behavior;
```

3.8.2 คำสั่ง While Loop

เป็นคำสั่งวนรอบอีกรูปแบบหนึ่ง การทำงานเหมือนกับคำสั่ง FOR LOOP แต่คำสั่งนี้ใช้สำหรับการจำลองการทำงานเท่านั้น จึงเหมาะสำหรับการเขียนเป็น โมดูลสำหรับทดสอบโมดูลอื่น (Testbench) แต่ใช้สังเคราะห์เป็นของจริงไม่ได้ มีรูปแบบการใช้งานดังนี้

```
[loop_label:]
WHIL condition LOOP
  -- sequential statements
END LOOP  [loop_label] ;
```

Loop_label เป็นชื่อของ loop จะมีหรือไม่มีก็ได้

condition เป็รเงื่อนไขที่ใช้กำหนดการทำงานภายในลูป ถ้าเงื่อนไขเป็นจริงจะทำงานภายในลูปแต่ถ้าเงื่อนไขเป็นเท็จก็จะหลุดออกจากลูป ดังนั้นการกำหนดเงื่อนไขนี้ต้องทำด้วยความระมัดระวัง ต้ องให้มีโอกาสที่เงื่อนไขเป็นเท็จ เพื่อป้องกันมิให้การทำงานไม่หลุดออกจากลูป

ตัวอย่างเปรียบเทียบการใช้ FOR LOOP กับ WHILE LOOP และโปรดสังเกตการใช้ Attribute สำหรับระบุจำนวนรอบ

```
entity CONV_INT is
  port (VECTOR: in   bit_vector(7 downto 0);
        RESULT:  out integer);
end CONV_INT;
```

```
architecture B of CONV_INT is
begin
  process(VECTOR)
    variable TMP: integer;

  begin
    TMP := 0;

    for I in VECTOR'range loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
    end loop;

    RESULT <= TMP;
  end process;
end B;
```

```
architecture C of CONV_INT is
begin
  process(VECTOR)
    variable TMP: integer;
    variable I      : integer;
  begin
    TMP := 0;
    I := VECTOR'high;
    while (I >= VECTOR'low) loop
      if (VECTOR(I)='1') then
        TMP := TMP + 2**I;
      end if;
      I := I - 1;
    end loop;
    RESULT <= TMP;
  end process;
end C;
```

3.9 คำสั่ง WAIT Statement

คำสั่ง WAIT ใช้สำหรับหยุดการทำงานภายใน PROCESS มี 4 รูปแบบดังนี้

- WAIT FOR

- WAIT ON
- WAIT UNTIL
- WAIT

WAIT FOR

สั่งให้หยุดรอเป็นระยะเวลาขนาดหนึ่ง มีรูปแบบดังนี้

```
WAIT FOR specific_time;
```

WAIT ON

สั่งให้หยุดการทำงานเพื่อรอการเกิดเหตุการณ์ที่ระบุไว้ มีรูปแบบดังนี้

```
WAIT ON signal_list;
```

WAIT UNTIL

สั่งให้หยุดการทำงานจนกว่าเงื่อนไขที่ระบุจะเป็นจริง มีรูปแบบดังนี้

```
WAIT UNTIL condition;
```

WAIT

สั่งให้หยุดการทำงาน มีรูปแบบดังนี้

```
WAIT;
```

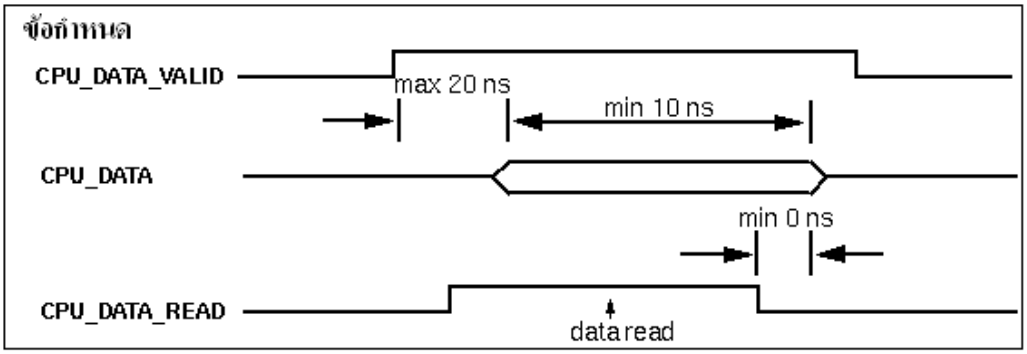
ตัวอย่าง เป็นการสร้างโมเดลของ D-flipflop โดยใช้คำสั่ง Wait On และ Wait Until เพื่อตรวจสอบสัญญาณนาฬิกา

```
entity FF is
    port (D, CLK : in   bit;
          Q      : out bit);
end FF;
```

```
architecture BEH_1 of FF is
begin
  process
  begin
    wait on CLK;
    if (CLK = '1') then
      Q <= D;
    end if;
  end process;
end BEH_1;
```

```
architecture BEH_2 of FF is
begin
  process
  begin
    wait until CLK=`1`;

    Q <= D;

  end process;
end BEH_2;
```

อีกตัวอย่างเป็นการสร้างสัญญาณที่กำหนดคาบเวลาต่างๆ ตามข้อกำหนด เช่นตามรูปที่ 6-2 เป็นไดอะแกรมเวลาสำหรับการอ่านข้อมูลของ CPU



รูปที่ 3-7 คาบเวลาของสัญญาณที่สร้างโดยคำสั่ง wait for

สามารถเขียนเป็น VHDL ได้ดังนี้

```
READ_CPU : process
begin
    wait until CPU_DATA_VALID = `1`;
    CPU_DATA_READ <= `1`;
    wait for 20 ns;
    LOCAL_BUFFER <= CPU_DATA;
    wait for 10 ns;
    CPU_DATA_READ <= `0`;
end process READ_CPU;
```

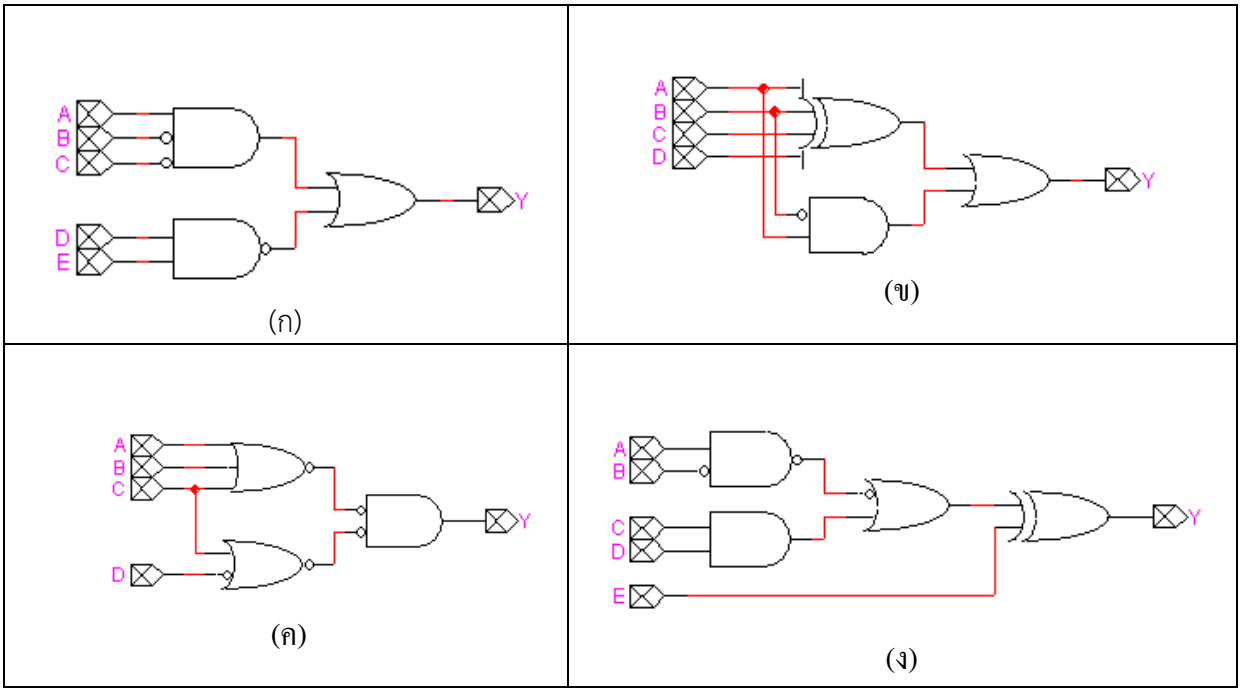
การเขียนค่าเวลานี้ ส่วนใหญ่ใช้กับการเขียน Testbench เพื่อการทดสอบระบบที่ออกแบบขึ้น

แบบฝึกหัด

3.1 จงเขียนเป็นโมเดล VHDL เพื่อทำหน้าที่เป็นโลจิกเกตต่อไปนี้

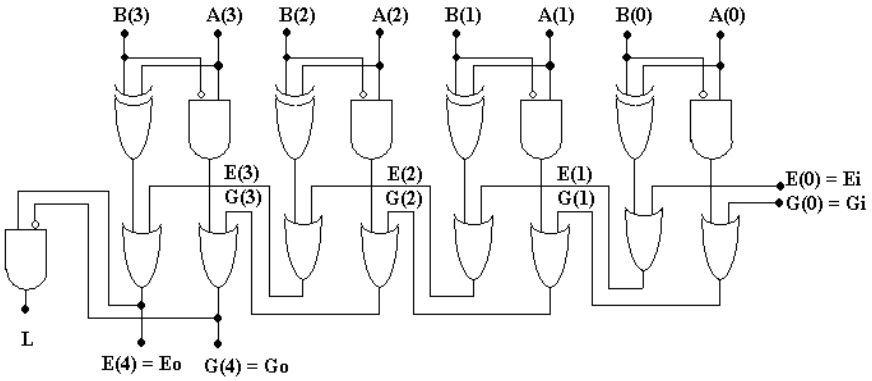
- 3.1.1 เกต AND 4 อินพุท
- 3.1.2 เกต OR 5 อินพุท
- 3.1.3 เกต XOR 4 อินพุท
- 3.1.4 เกต XNOR 4 อินพุท
- 3.1.5 เกต NAND 5 อินพุท

3.2 จากโลจิกไดอะแกรมในรูป 3-8 (ก) ถึง (ง) ต่อไปนี้ จงเขียนเป็นโมเดล VHDL เฉพาะส่วน Architecture



รูปที่ 3-8 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 32.

3.3 จากโลจิกไดอะแกรมในรูปที่ 3-9 จงเขียนโมเดล VHDL



รูปที่ 3-9 โลจิกไดอะแกรมของแบบฝึกหัดข้อ 3.3

3.4 จากโมเดล VHDL ต่อไปนี้ จงเขียนเป็นโลจิกไดอะแกรม

```
library IEEE;
use IEEE.std_logic_1164.all;
entity selector is
   port (a,  b,  s    : in std_logic;
         y : out std_logic);
end selector;
architecture rtl of selector is
begin
   y <= (a and not(s))  or  (b and s) after 10 ns;
end rtl;
```

3.5 จาก VHDL Code ถ้ากำหนดให้สัญญาณ A,B,C,D....H ให้เป็นตามตาราง จงแสดงค่าของสัญญาณ W X Y และ Z

```
library IEEE;
use IEEE.std_logic_1164.all;
entity p1 is
port( A, B, C, D, E, F, G, H : in std_logic;
        W, X, Y, Z : out std_logic);
end p1;
architecture dataflow of p1 is
begin
    W <= A and B;
    X  <= (C and D) or G;
    Y  <= F xor H;
    Z  <= not(E or G);
end  dataflow;
```

A	B	C	D	E	F	G	H	W	X	Y	Z
'U'	'X'	'1'	'0'	'L'	'-'	'H'	'1'	……	……	……	……

3.6 จากตารางการทำงานในตารางที่ จงเขียน 2 และตารางที่1โมเดล VHDL ให้สมบูรณ์

Input				Output	
A	B	C	D	X	Y
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	1
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	0
1	1	0	0	1	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	1	1

Input		Output
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

3.7 จาก VHDL code ต่อไปนี้ จงเขียนตารางการทำงานของระบบที่ VHDL นี้สร้างขึ้น

3.7.1

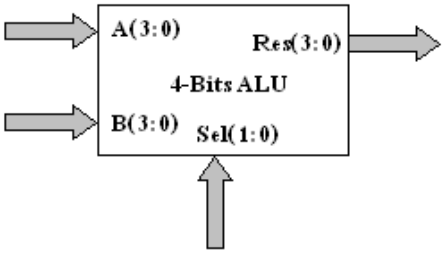
```
entity X_BOX is
        port (A, B, C, D: in BIT;
                X, Y: out BIT);
end X_BOX;

architecture DATA_FLOW of X_BOX is

        signal S, Q, R: BIT_VECTOR(2 downto 0);

begin

        S <= A & B & C;
        Q <= O"5";
        R <= S xor Q;
        with R select
        X <= '0'          when B"000" | B"111",
                not D when B"010" to "110",
                D         when others;
        Y <= A and B xor R(0) when (D/='0' and C='1') else
                A or B xor R(1)   when (D='0' and C/='1') else
                R(2);
end DATA_FLOW;
```

3.7.2

```
entity CONDITIONAL_ASSIGN is
        port (S : in  bit_vector (3 downto 0);
              Q : out bit_vector (3 downto 0));
end CONDITIONAL_ASSIGN;

architecture EX of CONDITIONAL_ASSIGN is
begin
        Q <= "0100"  when s = "0000" else
                "0110" when s = "0100" else
                "1110" when s > "1000" and s < "1010" else
                "1111";
end EX;
```

3.8 จงเขียนโมเดล VHDL ด้วยคำสั่งแบบขนาน เพื่อสร้าง ตัวเปรียบเทียบข้อมูลขนาด 8 บิต โดยมีอินพุทเป็น A และ B (A และ B มีขนาด 8 บิต) เอาท์พุทเป็น Q กำหนดให้ ถ้า A มากกว่า B ให้ Q มีค่าเป็นเป็น “10” ถ้า A น้อยกว่า B ให้ Q มีค่าเป็น “11” และถ้า A เท่ากับ B ให้ Q มีค่าเป็น “00”

3.9 จงเขียนโมเดล VHDL ด้วยคำสั่งแบบขนาน เพื่อสร้าง ALU ขนาด 4 บิต โดยมีบล็อกไดอะแกรม ดังรูปที่ -10



รูปที่ 3-10 บล็อกไดอะแกรมของ ALU ขนาด 4 บิต

กำหนดให้

A และ B เป็นอินพุท ขนาด 4 บิต

Res เป็นเอาท์พุท ขนาด 4 บิต

Sel เป็นอินพุทสำหรับกำหนดการทำงานของ ALU โดยมีการทำงานดังนี้

ถ้า Sel = “00” ให้ Res = A บวก B

ถ้า Sel = “01” ให้ Res = A ลบด้วย B

ถ้า Sel = “10” ให้ Res = A AND B

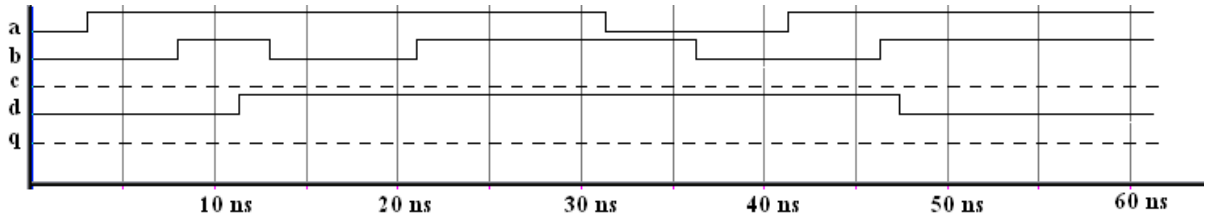
ถ้า Sel = “11” ให้ Res = A OR B

3.10 จาก VHDL Code จงเขียนโลจิกไดอะแกรม และถ้าให้ A = “1101” Q จะมีค่าเท่าไร

```
entity P6_1 is
        port (A : in  bit_vector (3 downto 0);
        Q : out bit_vector (3 downto 0));
end P6_1;
architecture str of P6_1 is
begin
  process(A)
  begin
    for i in 0 to 2 loop
      Q(i) <= A(i) xor A(i+1);
    end loop;
    Q(3) <= A(3);
  end process;
end str;
```

3.11 จาก VHDL Code จงเขียนไดอะแกรมเวลาของ C และ Q ลงในรูปที่ 3-11

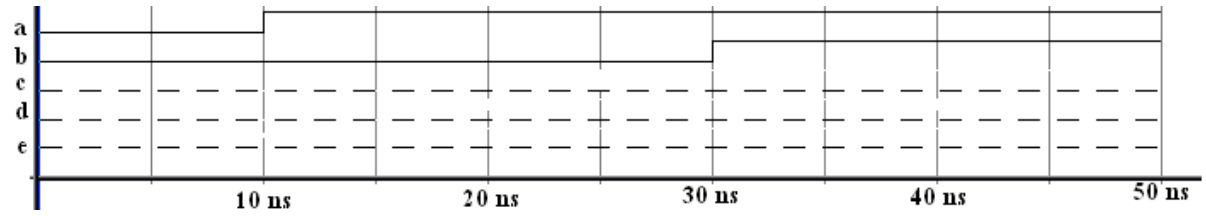
```
entity P6_2 is
        port (A, B, D : in  bit;
        Q : out bit);
end P6_2;
architecture str of P6_2 is
        signal  C : bit;
begin
  P1: process(A,B,C,D)
  begin
    C <= A AND B after 5 ns;
    Q <= C XOR D after 10 ns;
  end process;
end str;
```



รูปที่ 3-11 ไดอะแกรมเวลา

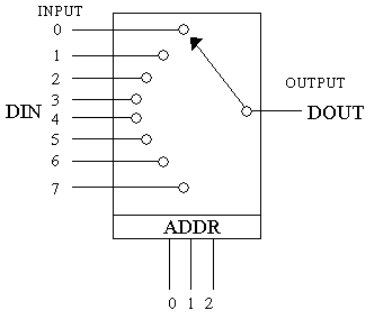
3.12 จาก VHDL Code จงเขียนไดอะแกรมเวลาของ C D และ Q ลงในรูปที่ 3-12

```
entity P6_3 is
          port (A, B: in  bit);
end P6_3;
architecture str of P6_3 is
   signal C, D, E : bit;
begin
  E <= A;
  P1: process
  begin
    wait for 10 ns;
    C <= A OR B after 5 ns;
    D <= A AND not B;
  end process;
end str;
```



รูปที่ 3-12 ไดอะแกรมเวลา

3.13 จากบล็อกไดอะแกรมของมัลติเพล็กเซอร์ ในรูปที่ 3-13 จงเขียน VHDL Code โดยใช้คำสั่งแบบลำดับ



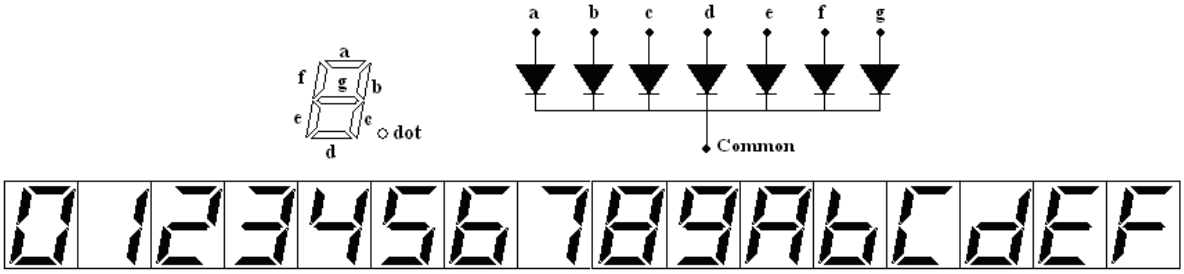
ADDR			Output
2	1	0	DOUT
0	0	0	DOUT = DIN 0
0	0	1	DOUT = DIN 1
0	1	0	DOUT = DIN 2
0	1	1	DOUT = DIN 3
1	0	0	DOUT = DIN 4
1	0	1	DOUT = DIN 5
1	1	0	DOUT = DIN 6
1	1	1	DOUT = DIN 7

รูปที่ 3-13 บล็อกไดอะแกรมของมัลติเพล็กเซอร์ แบบ 8 อินพุท

3.14 จากโจทย์ข้อ 3.8 จงเขียนโมเดล VHDL ด้วยคำสั่งแบบลำดับ

3.15 จากโจทย์ข้อ 3.9 จากจงเขียนโมเดล VHDL ด้วยคำสั่งแบบลำดับ

3.16 จงเขียน VHDL Code ด้วยคำสั่งแบบลำดับ สำหรับวงจรถอดรหัส จากรหัสไบนารีเป็นรหัส 7 – segment แบบ Common cathode



รูปที่ 3-14 การแสดงผลของ 7 – segment แบบ Common cathode