4. Operators

Operators

	not					
logical	and	or	nand	nor	xor	xnor
relational	=	/=	<	<=	>=	>
shift	sll	srl	sla	sra	rol	ror
	+	-				
	*	/	mod	rem		
arithmetic	**	abs				

sorted in order of increasing precedence (top->down)

New operators: xnor, shift operators

ตัวดำเนินการแบบโลจิก

Logical Operators

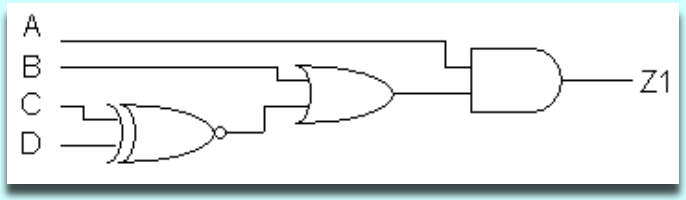

```
entity LOGIC_OP is
   port (A, B, C, D : in   bit;
         Z1:          out bit;
         EQUAL :      out boolean);
end LOGIC_OP;

architecture EXAMPLE of LOGIC_OP is
begin

   Z1 <= A  and (B or (not C xor D)));

   EQUAL <= A xor B;                -- wrong

end EXAMPLE;
```



ตัวกระทำทางด้านโลจิกจะใช้กับข้อมูลประเภท

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

ตัวกระทำทุกตัวต้องเป็นประเภทเดียวกันและมีขนาดเท่ากันผลลัพธ์ที่ได้จะเหมือนกับตัวกระทำทั้งประเภทและขนาด

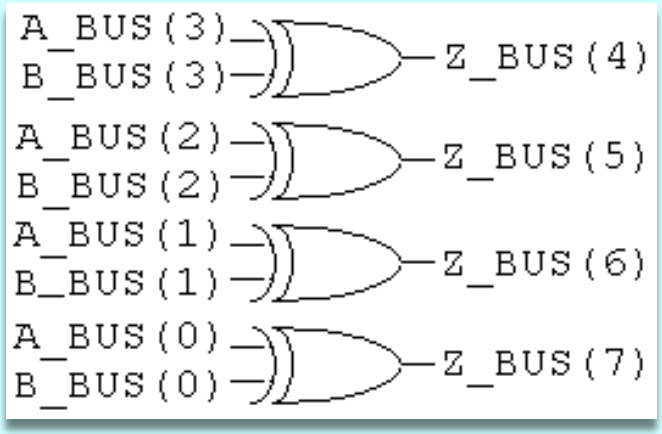
- Priority
  - not (top priority)
  - and, or, nand, nor, xor, xnor (equal priority)
- Predefined for
  - bit, bit_vector
  - boolean
- Data types have to match

Logical Operations with Arrays

สำหรับตัวกระทำที่เป็นอะเรย์ เมื่อสังเคราะห์แล้วจะได้เป็นกลุ่มของโลจิก

- Operands of the same length and type
- Assignment via the position of the elements (according to range definition)

```
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 and B_BUS;
end EXAMPLE;
```



ตัวดำเนินการสำหรับการเลื่อน

Shift Operators

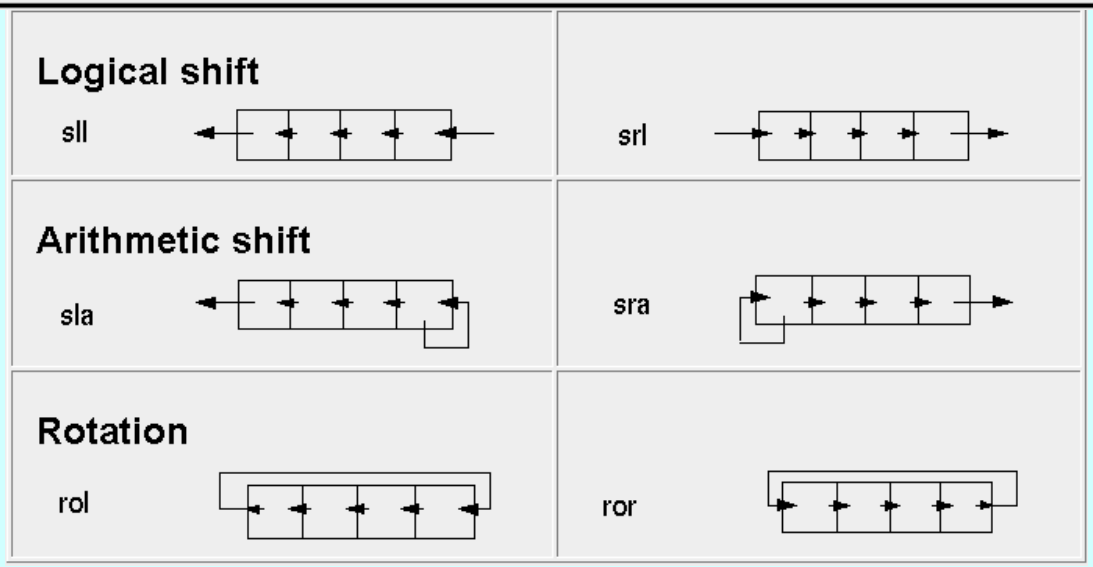
ตัวกระทำสำหรับการเลื่อนจะใช้กับข้อมูลประเภท
BIT_VECTOR
BOOLEAN
ทั้งนี้ความยาวของตัวรับและส่งต้องเท่ากัน ผลลัพธ์ที่ได้จะเหมือนกับตัวกระทำทั้งประเภทและขนาด

```
signal A_BUS, B_BUS, Z_BUS : bit_vector (3 downto 0);

Z_BUS <= A_BUS sll 2;
Z_BUS <= B_BUS sra 1;
Z_BUS <= A_BUS ror 3;
```

'93

At the end, the first value of the type is used for filling up



ตัวอย่าง ตัวดำเนินการสำหรับการเลื่อน


```
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" ได้ YA = "0001"
B = "1110" ได้ YB = "1101
A = "1000" ได้ YC = "0000"

Relational Operators

ตัวกระทำทางด้านการเปรียบเทียบจะใช้กับข้อมูลประเภทต่างๆดังนี้

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

ตัวกระทำทุกตัวต้องเป็นประเภทเดียวกันแต่จะมีขนาดเท่ากันหรือไม่ก็ได้
ในกรณีที่เป็นอะเรย์ ถ้ามีขนาดไม่เท่ากัน การเปรียบเทียบจะเปรียบเทียบจากข้อมูลทางซ้ายไปขวาเสมอ ผลลัพธ์ที่ได้จะให้ค่าเป็น จริงกับเท็จ

ตัวอย่าง Relational Operators


```
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"

Relational Operators


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity int_relational is

end int_relational;
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

Arithmetic Operators

+ addition	- substraction
* multiplication	** exponentiation
/ division	mod modulo
abs absolute value	rem remainder

```
signal A, B, C:    integer;
signal RESULT: integer;

RESULT<= -A + B * C;
```

เหมือนกับตัวดำเนินการทางคณิตศาสตร์ในภาษาระดับสูงทั่วไป ตัวกระทำจะใช้กับข้อมูลประเภทต่างๆ ได้ดังนี้

- Integer
- Real
- ประเภทกายภาพเช่น Time

ตัวกระทำทุกตัวต้องเป็นประเภทเดียวกัน ผลลัพธ์ที่ได้เป็นข้อมูลประเภทเดียวกัน

ตัวดำเนินการเกี่ยวกับเศษ
Remainder and Modulo

Remainder

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
```

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 คือเลขจำนวนเต็ม เมื่อคูณกับ B แล้วค่าเท่ากับหรือมากกว่า A

เช่น

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

5 mod –3 = -(3*2 - 5) = -1