# 3장 객체와 자료형 및 연산자

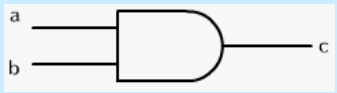


## 제1절 객체(object)와 문장(statement)

▪ 객체(data object)
VHDL에서 값을 갖을 수 있는 것으로 모든 객체는 자료형을 갖음

[예제 3.1] 객체의 종류 및 특징

▪ **signal**(신호) : 외적 변수, 선(wire)으로 표현




```
signal a, b, c : std_logic      -- signal 선언
c <= a and b ;                  -- 값(파형) 대입
```

▪ **variable(변수):내적변수,process 내부에서 유효**

```
variable temp : std_logic ;     -- variable선언
temp := a or b;                 -- 값(파형) 대입
```

▪ **constant(상수) :**
**초기에 선언된 값을 계속 유지하는 상수값 지님**

```
constant p1 : integer := 314 ;
                -- 상수선언과 동시에 상수값 대입
```

# 제1절 객체(object)와 문장(statement)

■ 신호(signal)
- 신호는 VHDL합성시 선(wire)으로 구현되며, 각 부품의 연결에 사용되는 외적변수.

- 객체에 값의 대입
  - "**<=**"의 오른쪽에서 왼쪽으로 대입하는 시간지연요소 대입기호를 사용.
  - signal의 초기값 대입의 경우는 즉시 대입 기호 ":="사용.

- 자료형의 선언
  - bit : 신호의 자료형이 0 혹은 1일 때 단일신호로 사용
  - bit_vector : 다중신호를 의미, 신호의 개수를 나타내는 내림차순 (**downto**) 또는 올림차순(**to**)를 사용


[그림 3.1]버스 다발과 비트의 순서





# 제1절 객체(object)와 문장(statement)

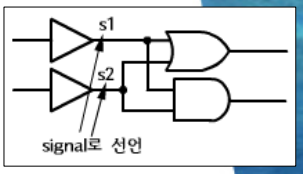
**[예제 3.2] signal의 선언**

```
1  signal  a, b, c : std_logic
      -- a, b, c는 객체의 이름
      -- a, b, c의 객체 종류는 signal이므로 선(wire)으로 구현가능
      -- a, b, c의 자료형이 bit형이므로 '1', '0'의 두 가지 값을 갖음
2  signal count : std_logic vector (3 downto 0) ;
      -- signal count(3), count(2), count(1), count(0) : bit와 동일한 선언
      -- count를 4 비트 bus로 선언
3  signal temp : std_logic vector (3 downto 0) := "1100";
      -- temp를 4 비트 bus로 선언하고 초기값 대입
```

**[예제 3.3] signal의 활용**

<VHDL 구문>

```
1   entity combi_logic is
2     port ( a, b : in std_logic;
3            z, y : out std_logic);
4   end combi_logic;
5   architecture data_flow of combi_logic is
6     signal s1, s2 : std_logic;  -- architecture와 begin사이에 선언
7   begin
8     s1 <= a;              -- 신호 a를 s1으로 전달
9     s2 <= b;              -- 신호 b를 s2로 전달
10    z  <= s1 or s2;       -- "s1 or s2"를 z에 전달
11    y  <= s1 and s2;      -- "s1 and s2"를 y에 전달
12  end data_flow;
```




[그림 3.2] 간단한 조합논리회로

# 제1절 객체(object)와 문장(statement)

■ 변수(variable)

process나 부프로그램에서만 사용되는 내적변수로서 중간연산단계에 주로 이용, 대입기호 즉시 대입기호 ':=' 사용

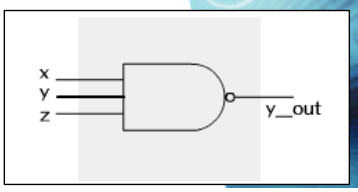
[예제 3.6] 변수 선언방식

```
variable temp1, temp2:std_logic;  -- variable_이름이 temp1, temp2로 정의, 자료형은 bit
        temp1 := '1';          -- :=는 즉시 값이 대입
        temp2 := a or b;       -- a, b 는 signal이고, temp2는 variable
```

[예제 3.7] 3입력 NAND 게이트

<VHDL 구문>

```
1   architecture example of nand_system is
2    begin
3     process (x, y, z)
4          variable temp : std_logic; -- variable 선언, variable 위치 process와 begin사이
5            begin
6             temp := '1' ;          -- temp가 즉시 '1'로 바뀜
7             temp := x nand temp ;   -- variable의 즉시 대입, temp=x
8             temp := y nand temp ;   -- temp=x · y
9             temp := z nand temp ;   -- temp=x · y · z
10           y_out <= temp ;  -- signal에 variable의 시간지연 대입, y_out=x*y*z
11    end process ;
12  end example ;
```




[그림 3.4] 엔티티 nand_system





# 제1절 객체(object)와 문장(statement)

■ 신호(signal)와 변수(variable)의 비교

▪ 신호(signal)

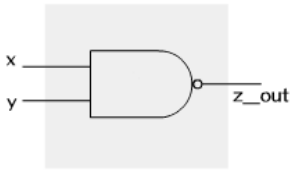
- 외부 변수
- 선언 : architecture와 begin 사이에서 선언, 부프로그램에서 선언, port signal에서 선언
- 합성시 wire로 구현되며, 비교적 간단한 연산이 요구될 때 이용
- signal에 값(파형)이 인가될 때 현재의 변화된 값이 바로 대입되지 않고 process가 끝난 시점에 대입

▪ 변수(variable)

- 내부 변수
- 선언 : process와 begin 사이에서 선언,부프로그램의 parameter에서 선언
- 합성시 선(wire)로 바로 구현되지 않고 연산의 중간단계로 활용
- 복잡한 알고리즘의 구현시 사용되며, 현재의 값만 가짐
- process와 subprogram내에서만 유효, 이 영역을 벗어나면 variable 값이 무효

# 제1절 객체(object)와 문장(statement)

**[예제 3.8]signal과 variable의 사용상 차이점**




[그림 3.5] 두 입력 AND게이트(entity nand2_system)

♪ **signal**이 잘못 사용한 예

<VHDL구문>

```
1  architecture example1 of nand2_system is
2     signal temp : std_logic ; -- signal 선언 (architecture와 begin사이)
3   begin
4       process (x, y)
5         begin
6            temp <= '1' ; -- 현 시점에서는 temp = '0', end process에서 temp ='1'
7            temp <= x nand temp; -- end process에서 temp ='1'
8            temp <= y nand temp; -- end process에서 temp ='1'
9            z_out <= temp;          -- end process에서 z_out = '1'
10       end process ;
11  end example1 ;
```





# 제1절 객체(object)와 문장(statement)

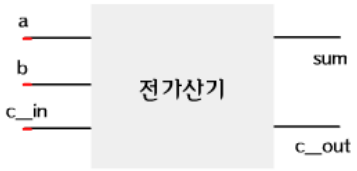
♪ **variable**을 사용한 문제 해결

<VHDL 구문>

```
1  architecture example2 of nand2_system is
2     begin
3       process (x, y)
4          variable temp : std_logic ;   -- variable선언 (process와 begin사이)
5         begin
6            temp := '1' ;                -- temp가 즉시 '1'로 바뀜
7            temp := x nand temp ; -- temp = (x*1)'
8            temp := y nand temp ; -- temp = (x*y)'
9            z_out <= temp ;          -- z_out = (x*y)'
10       end process ;
11  end example2 ;
```

# 제1절 객체(object)와 문장(statement)

[예제 3.9] 전가산기(full adder) 회로




[그림 3.6] 엔티티 full_adder

```
1  entity full_adder is
2        port ( a, b, c_in  : in std_logic ;
3              sum, c_out : out std_logic) ;
4  end full_adder ;
5  architecture example of full_adder is
6  begin
7       process (a, b, c_in)
8        variable temp_sum1,temp_c1,temp_c2,temp_c3 : std_logic; --객체 variable을 선언
9       begin
10            temp_sum1 := a xor b;                  -- 즉시 대입
11            sum      <= c_in xor temp_sum1;
12            temp_c1   := a and b;                  -- 즉시 대입
13            temp_c2   := a and c_in;               -- 즉시 대입
14            temp_c3   := b and c_in;               -- 즉시 대입
15            c_out <= temp_c1 or temp_c2 or temp_c3;
16       end process;
17   end example;
```





# 제1절 객체(object)와 문장(statement)

■ 상수(constant)
초기에 선언한 상수의 값을 유지하는데 사용하며, 고정된 값을 갖음

[형식 3.1] 상수의 선언 방식

**constant** 상수_이름 : 자료형[:= 초기값] ;  --상수선언과 초기화

[예제 3.10] 상수의 활용

```
constant delay : time :=5ns        -- constant 선언
constant size : integer :=516;     -- constant 선언, 초기화
```

■ 문장(statements)
- VHDL의 기본 문장
- 선언문(declaration statement)
  설계내에서 사용할 수 있는 상수, 자료형, 객체, 부프로그램 등을 정의하는데 사용
  - 상수(constants) : 리터럴(literal) 숫자, 스트링(strings)
  - 자료형(types) : 레코드(records) 배열(array)
  - 객체(objects) : 신호(signal), 변수(variable), 부품(component)
  - 부프로그램 : 함수(function), 프로시저(procedure)

# 제1절 객체(object)와 문장(statement)

▪ 병행문(concurrent statement)
각 문장이 다른 문장과 병렬로 수행, 하드웨어의 내부동작 또는 구조를 나타내는데 사용, 문장들이 기술된 순서와 관계없이 동시에 수행되는 문장

♂ 병행문의 종류
① 프로세스문(process statement)
② 블록문(block statement)
③ 프로시저 호출문(procedure call statement)
④ 부품 개체화(component instantiation)
⑤ 생성문(generate statement)

[예제 3.11] 병행문의 활용

```
1  architecture data_flow of concurr_ent is
2      signal a, b : std_logic;      --a, b가 신호로 선언
3    begin
4      a <= in3 and in4;    -- and 논리
5      b <= in5 or in6;     -- or 논리
6      out1 <= in1 xor a    -- xor의 출력 논리
7      out2 <= in2 xor b;   -- xor의 출력 논리
8  end data_flow;
```





# 제1절 객체(object)와 문장(statement)

▪ 순차문(sequential statement)
문장들이 쓰여진 순서대로 수행되며, 프로세스(process)문, 부프로그램 내에서 사용

▪ 순차문의 종류
♂ 변수 대입문(variable assignment statement)
♂ 신호 대입문(signal assignment statement)
♂ 프로시저와 함수 호출(procedure and function calls)
♂ **if**, **case**, **loop**, **next**, **exit**, **return**, **wait**문

[예제 3.12] 순차문의 활용

```
1  architecture data_flow of sequent_ial is
2    begin
3     process
4      begin
5       wait until clk;
6       if (accelerator = '1') then
7         case speed is
8            when stop => speed <= slow;
9            when slow => speed <= medium;
10           when medium => speed <= fast;
11           when fast => speed <= slow;
12        end case;
13      end if;
14     end process;
15 end data_flow;
```

# 제2절 자료형(data types)

■ 자료형의 명시

▪ **자료형**

VHDL에서 **객체**는 특정 **자료형(data type)**으로 **선언**되고 **자료형의 데이터**를 가지며, 거의 **무한한 종류의 자료형을 사용**

[형식 3.2] 객체 선언시 자료형의 명시

```
signal      x, y  :  std_logic
variable     kk   :  std_logic_vector ( 3 downto 0)
constant     ss   :  boolean
```

객체의 종류 객체명 자료형

▪ 미리 정의된 VHDL의 자료형

- 부울(boolean)형: 두 값을 갖는 열거형으로 논리연산과 관계연산의 결과는 boolean값이 됨
- 비트(bit)형: bit형은 두 값 '0'과 '1'을 갖는 열거형으로 논리연산의 결과 값도 bit값이 됨.
- 비트-_벡터(bit_vector)형: bit_vector형은 bit값의 배열을 표현하는데 사용
- 문자(character)형: 문자의 열거형으로 문자는 단일 인용부호(' ') 안에 표현하는데 사용.
- 정수(integer)형: 정수형은 양수와 음수를 표현하는데 사용.
- natural 형: natural형은 정수형의 부자료형으로 0과 양수를 표현하는데 사용
- positive형: 정수형의 부자료형으로 양수 즉, 0과 음수가 아닌 수의 표현에 사용
- 문자열(string)형: 문자의 배열을 나타내며, 문자열의 값은 이중 인용부호에 표현
- 실수(real)형: 실수를 표현하는데 사용
- 시간(time): 시뮬레이션을 위해 사용되는 시간의 값을 표현하는데 사용.

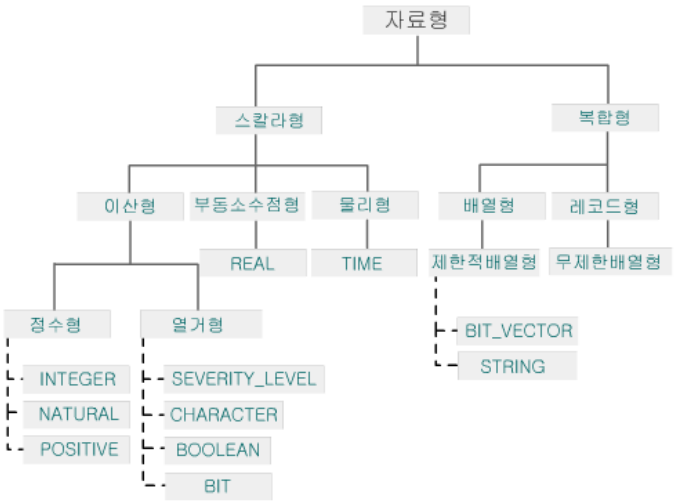




# 제2절 자료형(data types)

■ 자료형의 분류

▪ 스칼라형(scalar type), 복합형(composite type)




[그림 3.7] 자료형의 분류

## 제2절 자료형(data types)

■ 스칼라형(scalar type)
- 정수형(integer type)

⅋ 특정 정수 범위 내의 값을 갖는 자료형을 정의

[예제 3.13] 정수형의 선언

```
type byte is range -128 to 127;
type bit_position is range 14 downto 0;
```

[예제 3.14] 미리 정의된 정수형

```
type integer is range -2147483647 to 2147483647; -- integer는
                                                 -- 미리 정의
```

- 실수형(real type or floating point type)

⅋ 일정한 범위 내의 실수 값을 가짐

[예제 3.15] 실수형의 사용 예

```
type real is range -1.0E38 to 1.0E38;  -- real은 미리 정의된 것
type norm is range 0.0 to 1.0;
```





## 제2절 자료형(data types)

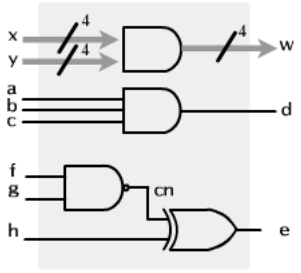
■ 자료형의 특징
- 정수형(integer type)
  정수형은 기본적인 산술연산자에 대한 지원이 가능, 시뮬레이션 시 메모리의 효율적 사용이 가능.
- 비트형(bit type)
  비트형은 VHDL 언어 자체에 적합하나, 기본적인 산술연산의 지원이 어렵고, 3상태, 연결논리 등의 지원이 안되는 단점
- std_ulogic과 std_ulogic_vector형
  미지상태(unknown), 3상태(tristate) 등의 로직상태를 표현하는 모델링이 가능하나, 분해형이 아님
- **std_logic과 std_logic_vector형**
  **로직의 다양한 형태의 표현이 가능, 가장 권장하는 자료의 형태**
- IEEE 1076.3 unsigned와 signed형
  무 부호(unsigned)형은 부호가 없는 2진형의 계산, 부호(signed)형은 부호가 있으며, 2의 보수로 계산

# 제3절 연산자(operator)와 속성(attribute)

- 연산자(operators)
  - 논리 연산자(logic operators)
    논리조합을 만드는데 사용하며, bit, boolean, bit_vector를 지원.

[표 3.1] 논리 연산자(AND 게이트)

| A | B | A and B |
|---|---|---|
| True '1' | True '1' | True '1' |
| True '1' | False '0' | False '0' |
| False '0' | True '1' | False '0' |
| False '0' | False '0' | False '0' |




[그림 3.8] 논리 연산의 논리도

[예제 3.25] 논리연산자의 활용 (bit_vector, bit, boolean)

```
1  architecture example of combi_logic is
2      signal   w, x, y   : std_logic_vector (3 downto 0);
3      signal   a, b, c, d : std_logic ;
4      signal   e, f, g, h : boolean ;
5     begin
6          w <= x and y;              -- w, x, y는 같은 수의 bit_vector 형임
7          d <= (a and b) and c ;     -- a, b, c, d는 같은 bit 형 임
8          e <= (f nand g) xor h ;    -- e, f, g는 같은 boolean형 임
9   end example;
```





# 제3절 연산자(operator)와 속성(attribute)

- 관계 연산자(relational operators)
  비교함수를 만드는데 사용되며, 대부분의 자료형을 지원.

[표 3.2] 관계 연산자의 종류

| 연 산 자 | 의 미 |
|---|---|
| = | 같음(equal to) |
| /= | 같지 않음(not equal to) |
| < | 왼쪽의 값이 작음(less than) |
| <= | 왼쪽의 값이 작거나 같음(less than or equal to) |
| > | 왼쪽의 값이 큼(greater than) |
| >= | 왼쪽의 값이 크거나 같음(greater than or equal to) |

[예제 3.26] 관계 연산자의 사용

```
1  variable num1 : real := 100.0;
2  variable num2 : std_logic_vector(7 downto 0) :=('0','0','0','0','0','0','0')
3  variable num3, num4 : bit_vector(2 downto 0)
4     num1 /= 350.54                -- num1과 350.54는 같지 않음
5     num1 = 100.0                  -- num1과 100.0은 같음
6     num2 /= ('1','0','0','0','0','0','0')   -- num2가 오른쪽과 같지 않음
7     num1 > 45.54                  -- num1은 45.54보다 큼
8     num2 < ('1','0','0','0','0','0','0')    -- num2가 오른쪽보다 작음
```

# 제3절 연산자(operator)와 속성(attribute)

▪ 산술 연산자(arithmetic operators)
정수와 실수 등의 숫자형만 지원 가능하며, 다른 자료형의 경우 IEEE 1076.3의 numeric_std 패키지 이용.

[표 3.3] 산술 연산자의 종류

| 연 산 자 | 의 미 |
|---|---|
| + | 덧셈(addition) |
| - | 뺄셈(subtraction) |
| * | 곱셈(multiplication) |
| / | 나눗셈(division) |
| mod | 모듈(modulus) |
| rem | 나머지(remainder) |
| abs | 절대값(absolute value) |
| ** | 제곱(exponentiation) |

[예제 3.27] 산술 연산자의 사용

```
 1  architecture arith of or2 is
 2   begin
 3       process(a,b)
 4        begin
 5           y1 <= a+b;          -- a 더하기 b
 6           y2 <= a-b;          -- a 빼기 b
 7           y3 <= a*b;          -- a 곱하기 b
 8           y4 <= a mod b;      -- a를 b로 나눈 모듈
 9           y5 <= a rem b;      -- a를 b로 나눈 나머지
10           y6 <= abs a;        -- a의 절대값
11        end process;
12    end arith;
```



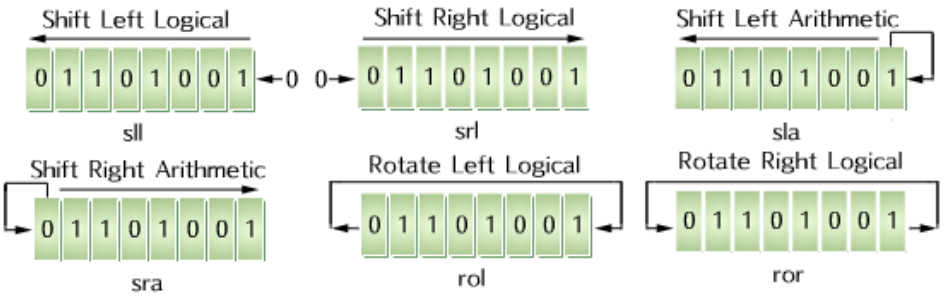


# 제3절 연산자(operator)와 속성(attribute)

▪ 순환 연산자(shift operators)
순환 연산자는 1차원적인 bit의 배열이나 부울형을 지원.

[표 3.4] 순환 연산자의 종류

| 연 산 자 | 의 미 |
|---|---|
| sll(shift left logical) | 논리에 대한 왼쪽방향으로의 순환 |
| srl(shift right logical) | 논리에 대한 오른쪽 방향으로의 순환 |
| sla(shift left arithmetic) | 산술에 대한 왼쪽 방향으로의 순환 |
| sra(shift right arithmetic) | 산술에 대한 오른쪽 방향으로의 순환 |
| rol(rotate left) | 논리에 대한 왼쪽 방향으로의 순환이동 |
| ror(rotate right) | 논리에 대한 오른쪽 방향으로의 순환이동 |




[그림 3.9] 순환 연산

# 제3절 연산자(operator)와 속성(attribute)

▪ 연결 연산자(concatenation operators)
연결 연산자는 원하는 새로운 형태의 1차원 배열을 만드는데 편리한 기능 제공

[표 3.5] 연결 연산자의 종류

| 연산자 | 의 미 |
|---|---|
| & | 연결 연산자 |

[예제 3.29] 연결 연산자의 사용

```
 1  entity con is
 2    port(A, B : in std_logic_vector (2 downto 0);   -- 입력 A :"100", 입력 B : "010"이라 가정
 3         Y : out std_logic_vector (14 downto 0));  -- 출력 Y는 15bit
 4  end con;
 5  architecture cdma of con is
 6     constant C : std_logic_vector (2 downto 0) :="001"; -- C를 상수로 선언하고, 3bit "001"
 7  begin
 8      process(A, B)
 9      begin
10        Y <= A & B & C & C & "110";  -- 최상위 비트부터 “100”, “010”, “001”,
11      end process;                   -- “001”, "110"이 연결되어
12  end cdma;                          -- "100010001001110"이 출력
```

[제3장 객체와 자료형 및 연산자]

# 제3절 연산자(operator)와 속성(attribute)

▪ 연산자의 우선순위(operators priority)

[표 3.7] 연산자의 우선순위

| 구 분 | 종 류 | 우선 순위 |
|---|---|---|
| 논리 연산자 | and, or, nand, nor, xor, xnor | 7 |
| 관계 연산자 | =, /=, <=, >, >= | 6 |
| 쉬프트 연산자 | sll, srl, sla, sra, rol, ror | 5 |
| 덧셈 연산자 | +, -, & | 4 |
| 부호 | +, - | 3 |
| 곱셈 연산자 | *, /, mod, rem | 2 |
| 기타 연산자 | **, abs, not | 1 |

# 제3절 연산자(operator)와 속성(attribute)

■ 속성(attribute)
자료형에 대한 동작이나 상태 표현을 위한 특성으로 엔티티, 아키텍처, 자료형 및 신호 등에 대한 정보의 제공

[예제 3.33] 많이 사용되는 속성

```
1   type index is integer range 1 to 30;
2   type state is (one, two, three, four);
3    subtype short_state is states range two to four;
4   signal byte : std_logic_vector(7 downto 0);
5   signal clk : std_logic;
```

<속성의 예>

① 속성'left : index'left = 1 -- index의 가장 왼쪽의 값 '1'을 나타냄
② 속성'low : index'low = 1 -- index의 가장 작은 값 '1'을 나타냄
③ 속성'high : index'high = 30 -- index의 가장 큰 값 '30'을 나타냄
④ 속성'right : state'right = four -- state의 가장 오른쪽의 값 'four'를 나타냄
속성'length : index'length = 10 -- index의 길이가 '10'을 나타냄
⑤ 속성'range : byte'range = (7 **downto** 0) -- byte의 범위가(7 downto 0)를 나타냄
⑥ 속성'event : clk'event -- clk의 event가 있을 때 'true'
속성의 사용 : **signal** att : state;
att <= states'right -- att = four

(※ 인용부호 ' : tick로 발음)