Entity 선언과 Architecture Body 선언

1. Entity 선언부는 사용자가 설계하고자 하는 시스템의 외적 연결을 담당하는 부분이다. 외부와의 통신을 위한 입출력 선을 정의하는 것을 Entity 선언이라고 한다. 간단히 2 입력 and 게이트 회로를 가지고 예를 들어 보겠다.

```
예) entity and2 is
      port( a, b : in       std_logic;
            y    : out    std_logic );
      end and2;
```

여기서 entity 의 이름을 설정하는 것에 유의하도록 하자. VHDL 은 entity 명의 맨앞에 숫자를 먼저 표기할 수 없으며, 공백이 있어서는 안된다. 외부와의 통신을 위한 입력 선으로 a 와 b 가 출력 선으로 y 가 선언되었다.

2. Architecture Body 는 사용자가 설계하고자 하는 시스템 내부의 동작을 세부적으로 정의하는 부분이다.

```
예) architecture sample of and2 is
     begin
     y <= a and b;
     end sample;
```

여기서 sample 이란 architecture 의 이름이다. Architecture 가 이름을 가진다는 것은 하나의 entity 가 여러 개의 Architecture 를 가질수 있다는 것을 의미 하며 이름은 정해진 것이 아니라 임의대로 바꾸어도 된다. Architecture 의 시작과 끝은 begin 과 end 이다. 위 예제에서처럼 2 입력 and 게이트 회로에 대한 내부 동작은 표현이 간단하다.

객체(Object)와 자료형(Data Type) 및 연산자(Operator)

1. 객체(Object)란 VHDL 에서 값을 가질 수 있는 것을 말한다. 객체를 선언함에 있어 객체가 어떠한 형태, 즉 자료형을 사용할 것인가를 결정해야 한다.

① 객체의 종류

i) Signal

VHDL 합성시에 Wire 로 구현이 되며, 각 Component 의 연결에 사용되는 외적 변수이다.
Signal 형태의 객체에 값을 대입하기 위해서는 예에서와 같이 '<='를 사용한다.
외부 연결 신호의 선언에는 Signal 로 선언하는 방법과 Port 로 선언하는 방법이 있다. Port 와 Signal 선언에서의 또다른 차이는 입출력의 구분, 즉 Mode 에 대한 정의 여부이다.

◈ Port

예) port (a, b : in std_logic;
y : out std_logic);

◈ Signal 로 선언하는 방법

Signal 의 선언위치는 architecture 와 begin 사이이다.

예) architecture sample of logic is

```
signal a, b : std_logic;
signal temp : std_logic_vector(3 downto 0) := "1100";
  -- 초기값을 대입할 때는 기호 ':='를 사용한다.

  begin
    ......
end sample;
```

Signal 사용시 주의할 점은 대입기호 '<='는 그 값이 즉시 대입되는 것이 아니다. Process 문 내에서 값이 대입되는 시점은 Process 문이 끝나는 End process 를 만날 때 비로써 그 값이 대입됨을 유의해야 한다. 이것을 Δ delay 라고 한다.

ii) Variable

Variable 은 process 나 부프로그램, 즉 function 과 procedure 에서 사용되며 process 내부에서만 유효한 내적 변수이다. Variable 은 Signal 과 같이 Wire 로 구현되지 않으며 Sythesizer 에 의해서 기억소자로 구성되기도 하며 임시 레지스터 구현되는 것이 보통이다. Variable 에 사용되는 대입기호는 ':='이다. 이 기호는 문장을 만나는 즉시 입력된다.

◈ Variable 로 선언하는 방법

Variable 의 선언위치는 process 와 process 의 시작을 나타내는 begin 사이이다.

예)
```
process(a, b)
  variable temp1, temp2 : std_logic;
begin
  ......
end process;
```

또한 Variable 은 즉시 값이 대입되지만 process 문을 빠져 나오면 그 값을 상실해 버린다. Variable 의 값은 process 문 안에서만 유효하기 때문에 값을 보존하기 위해서는 process 를 빠져 나오기 전에 그 값을 Signal 에 대입해야 한다.

iii) Constant

Constant 는 초기에 선언한 상수의 값을 유지하는데 사용하며, 선언된 초기값을 바꿀 수 없다. 대입기호 역시 Variable 과 마찬가지로 ':='를 사용한다.

◈ Constant 로 선언하는 방법

Constant 의 선언위치는 loop, generate 문, 부프로그램 또는 generic 에서 선언된다.

예) architecture sample of logic is

```
constant delay : integer := 5ns;
  Begin
    ……
End sample;
```

2. 자료형(Data Type)은 어떤 임의의 상태값이나 결과를 받아들이거나 입력값으로 처리할 수 있는 형태를 말하며, 기능과 표현상의 분류할 수 있다. VHDL 에서 객체는 거의 무한한 종류의 자료형을 사용할 수 있다. 또한 사용자가 직접 만들 수 있다.

① 자료형의 종류

자료형의 종류로는 크게 기능과 표현상의 분류로 구분되다. 간단히 요약하면 다음과 같다.

i) 표현적 분류

◈ Predefinde Data Type (이미 정의된 VHDL)

◈ User-Definde Type (사용자 정의)

ii) 기능적 분류

◈ Scalar Type(숫자형)

숫자형이라는 것은 0, 1, 2, 3 … 등의 숫자로 그 순서를 정의할 수 있는 자료형이다. 이러한 숫자형은 또다시 세부적으로 나뉜다.

열거형 (Enumeration Type)	BIT, BOOLEAN, CHARACTER, STD_LOGIC
정수형 (Integer Type)	INTEGER
실수형 (Floating Type)	REAL

물리형 (Physical Type)	UNIT : TIME, DISTANCE

◈ Composite Type(복합형)

혼합형은 이러한 숫자형을 재 조합하여 필요한 만큼만 사용할 수 있도록 선언할 경우와 여러 가지를 혼합하여 사용하고 싶을 때 사용한다. 혼합형은 배열형과 집합형으로 나뉜다.

배열형 (Array Type)	제한형 (Constraint Type)	TO, DOWNTO
	무제한형 (Unconstraint Type)	BIT_VECTOR, STRING, STD_LOGIC_VECTOR
집합형 (Record Type)		RECORD

◈ Access Type(연결형)

연결형은 일반 Computer 언어에서의 Linked List 를 사용할 수 있도록 만들어진 Point 형식과 유사하다. 연결형을 사용할 때 정확한 표현이 아니면 전혀 다른 결과를 출력하기 때문에 언어를 시작하는 초보자라면 되도록 사용을 피하는 것이 좋다. 또한 연결형은 시뮬레이션을 위한 용도 외에 하드웨어적으로 구현하는 것으로는 아직까지 그 지원이 미흡하다.

연결형 (Access Type)	ACCESS, NEW

◈ File Type

File Type 은 File 입출력에 관련된 Data Type 으로써 외부와의 입출력이나 특정 File 의 변수를 선택하여 사용한다. 또한 이 자료형은 VHDL 의 회로 표현에 대한 검증 단계, 즉 Simulation Debugging 에서 많이 사용되며 특별한 회로동작의 입출력을 제외하면 거의 사용하지 않는다. 그래서 Synthetic Code 가 아니란 것을 유의하자. 파일형은 연결형과 같이 초보자에게는 어려운 점이 있으니 유의하기 바란다.

파일형 (File Type)	FILE

② 선언 방법

자료형을 선언할 때는 Type ~ is 라는 예약어를 사용한다. 여기서는 Std_logic_1164 Package 에 선언된 내용을 예로 들었다.

ⅰ) 열거형 :: 문자열을 하나의 Data 로 묶어서 선언하는 것이다.

예) Type bit is ('0', '1');

Type boolean is (true, false);

ⅱ) 정수형 :: 정수형의 범위는 아래와 같이 구격화되어 있다.

예) Type integer is range -2147483647 to 2147483647;

Type byte is range -127 to 127;

ⅲ) 실수형 :: 실수형의 범위 역시 아래와 같이 -1.0E38 에서 1.0E38 까지 규격화되어 있다. 실수형의 최대값과 최소값은 High 와 Low 라는 Attribute 를 사용해서 나타낼 수도 있다.

예) Type real is range -1.0E38 to +1.0E38;

ⅳ) 물리형 :: 물리형은 시간, 거리 전류 등의 물리적인 단위를 나타낸다. 사용자가 범위를 정의하지 않으면 기본 단위의 고정 범위는 정수형의 범위과 같은 -2147483647 에서 2147483647 까지 이다.

```
예) Type time is range -1.0E38 to +1.0E38;
        unit ps;
            ns = 1000ps;
            us = 1000ns;
            ms = 1000us;
            sec = 1000ms;
            min = 60sec;
    end unit;
```

ⅴ) 집합형 :: 집합형은 다른 종류의 숫자형이나 배열형을 하나로 만들어 사용할 수 있는 것이다. 각각의 Field 로 나뉘어져서 별개의 Data Type 을 선언할 수 있다. Record Type 으로 선언된 객체의 요소 값을 참조하기 위해서 객체와 요소사이에 '.'을 사용한다.

```
예) Type information is
        record value : bit;
        context : integer;
    end record;
    variable kk : information;
    kk.value := '1';
    kk.context := 9;
```

vi) 배열형 :: 같은 종류의 숫자형을 한군데 묶어서 사용할 수 있다. 선언 방법에 따라 두 가지로 나뉜다.

◈ 제한형 :: 이미 선언된 Type 을 정해진 범위에서 열거하게 만든 것으로 일종의 BUS Type 으로 만든 것이다. 여기서 to 와 downto 는 내림차순과 오름차순으로 정의하는 것이다.

예) Type byte is array(7 downto 0) of bit;
variable a : byte;
a := "01110000";

◈ 무제한형 :: 제한형과는 다르게 선언된 Type 을 또 하나의 변형된 새로운 Type 으로 다시 만드는 열거형이다.

예) Type bit_vector(natural range <>) of bit;
Type string is array (positive range <>) of character;

vii) 연결형 :: 연결형은 동작적 모델링이나 하드웨어/소프트웨어 혼합형을 모델링 할 때 동적 메모리 할당(Dynamic Memory Allocation)을 필요로 하는 경우에 유용하고, 자료구조(Data Structure)의 크기를 미리 결정할 수 없는 경우에 사용한다. 예를 들어 큐(Que)나 스택(Stack) 구조에서 연결 리스트(Linked List)를 만들기 위해 사용한다. 연결형에는 3 가지로 구분할 수 있으나 2 가지의 형식만 알고 넘어가자.

◈ 불완전 선언 :: Type 선언시 아무런 표현도 하지 않고 선언자만을 표시

예) Type CELL;

◈ 일반 선언 :: Type 선언시 아무런 표현도 하지 않고 선언자만을 표시

예) Type LINK is access CELL;

viii) 파일형 :: 파일형은 VHDL 규격 패키지의 TEXTIO 부분에 선언되어 있기 때문에 활용만 하면 된다. 이것 또한 형식만 알고 넘어가자.

예) Type TEXT is file of STRING;
file INPUT : TEXT is in "std_input";
file OUTPUT : TEXT is out "std_output";

3. 연산자(Operator)는 아래 표에 간단히 요약했다. 연산자를 연속으로 사용할 경우 우선 순위에 따라 연산 순서가 결정된다. 괄호 안의 연산자는 우선한다. 특히 논리 연산자는 두 피연산자의 자료형이 같아야 하며 연산자의 우선 순위가 가장 낮으나 not 는 논리 연산자이지만 다른 연산자보다 우선 순위가 높다. and 나 or 연산자는 결합법칙이 성립되나,

여러 논리연산자를 함께 사용할 경우나 nand 혹은 nor 을 연속해서 사용할 경우 결합법칙이 성립되지 않으므로 괄호를 사용하고 결합순서에 주의해야 한다.

우선순위	논리 연산자(Local Operator)	or, and, nor, nand, xor, xnor
높다	관계 연산자(Relational Operator)	=, /=, >, <, >=, <=
↓	덧셈 연산자(Adding Operator)	+, -, &
↓	단항 연산자(Unary Operator)	+, -
↓	곱셈(Multiplying)	*, /, mod, rem
↓	기타 연산자	**, abs, not
낮다		

동작적 표현(Behavioral Description)과 구조적 표현(Structural Desciption)

VHDL 의 표현 방법은 설계의 의도와 용도에 따라 방법을 선택하거나 혼합해서 사용할 수 있다.

1. 동작적 표현은 자료흐름적 표현과 Process 문에 의한 표현으로 나뉘어 진다.

① 자료흐름적 표현(Data Flow Description)

부울 대수식, RTL 혹은 연산자를 사용하여 입력과 출력사이의 관계를 나타내는 것을 말한다. 자료흐름적 표현은 주로 병행 처리문에서 사용되며, 문장의 순서에 무관하게 동시에 수행된다.

예)
```
architecture data_flow of compare_logic is
  begin
  equal <= not( a(1) xor b(1) ) and not( a(0) xor b(0) );
end data_flow;
```

② Process 문에 의한 표현

자료흐름적 표현 방법보다 추상화된 개념이다. 회로의 기능적 표현을 기능적 혹은 알고리즘적으로 기술하는 것이다. Process 문은 하드웨어 시스템을 모듈별로 기술하는데 편리하다. 시스템은 하드웨어 모듈로 구성되어 있고, 각 모듈은 병행처리를 하면서 서로간의 통신을 통해 관계를 유지한다. Architecture 내에 여러 개의 Process 문이 있을 수 있으며 각 Process 문은 병행처리를 하지만, Process 문 내부는 순차처리를 한다. 또한 Process 문은 감지신호의 변화를 통해 동작하며 이를 통해 서로 통신을 한다.

예)
```
architecture sample of compare_logic is
  begin
process(a, b)                    -- ( )안의 a, b 를 감지신호라 한다.
  begin
    if a = b then
      equal <= '1';
    else
      equal <= '0';
    end if;
```

```
    end process;
  end sample;
```

2. 구조적 표현은 동작적 표현보다 하드웨어 구조에 가장 가까운 표현이라 할 수 있다. 하드웨어 회로상에서 IC 소자들을 서로 선으로 연결하여 시스템을 구성하듯이 이미 설계된 Component 를 이용하여, 그 Component 들을 서로 연결하여 System 을 기술하려는 방식이다. 구조적 표현에는 Generate 문과 Generic 문이 있다.

① Component 문

이미 설계한 Entity 를 부품으로 간주하여 구조적으로 설계하는 문이다. 이를 사용하기 위해서는 Component 를 선언해야 한다. 또한 Port map()이란 예약어를 사용하여 Component 를 사례화 시켜야 한다.

예) entity nand_component is

```
  port(
      in1, in2, in3, in4 : in std_logic;
      out1, out2 : out std_logic );
  end nand_component;
 architecture sample of compare_logic is
    component nand2   -- component nand2 를 선언
      port( a, b : in std_logic;
            y : out std_logic );
      end component;
   begin
    u1 : nand2 port map( in1, in2, out1);   -- nand2 를 사례화, 위치결합 방식
    u2 : nand2 port map( a=>in3, b=>in4, y=>out2 );    -- 이름결합 방식
  end sample;
```

예의 port map 에서 위치결합과 이름결합 방식이 있다. 위치결합은 component 와 결합할 때, port signal 이 나열된 위치순서대로 연결되며, 이름결합은 port signal 이 나열된 위치와 상관없이 각각의 '형식이름' => '실제이름'으로 연결된다. 이름결합은 직접 이름으로 연결되기 때문에 결합의 순서에 무관하다. 또한 component 를 사용하기 위해서는 사용할 component 가 작업 디렉토리에 등록이 되어 있어야 한다.

② Generate 문

Generate 문은 Component 를 반복적으로 사용하기 위해서 사용한다. Generate 문은 단순 반복생성을 위한 for-generate 문과 주어진 조건에 따라 반복처리하는 if-generate 문이 있다. 예를 들기 위해 긴 entity 이름을 사용했다. 사용자는 굳이 그럴 필요는 없다.

◈ for-generate 문

예) entity nand_component_for_generate is

```
      port ( a, b : in std_logic_vector( 3 downto 0 );
               y : out std_logic_vector( 3 downto 0 ) );
      end nand_component_generate;

   architecture sample of nand_component_generate is
      component nand2
```

```
    port ( a, b : in std_logic;
             y : out std_logic );
    end component;
begin
 g1 : for i in 3 downto 0 generate
 ux : nand2 port map ( a(i), b(i), y(i) );
 end generate g1;
end sample;
```

◈ if-generate 문

예)
```
entity xor_component_if_generate is
 port ( a : in std_logic_vector( 4 downto 0 );
        prity_check : out std_logic );
end nand_component_generate;

architecture sample of nand_component_generate is
    signal y : std_logic_vector( 3 downto 0 );
    component xor2
    port (
          a, b : in std_logic;
           c : out std_logic );
    end component;
 begin
    g1 : for i in 3 downto 0 generate   -- g1, g2, g3, u4, ux 는 레이블이다.
      g2 : if i =3 generate
           u4 : xor2 port map ( a(i+1), a(i), y(i) );
           end generate g2;
      g3 : if i < 3 generate
           ux : xor2 port map ( y(i+1), a(i), y(i) );
           end generate g3;
    end generate g1;
    parity_check <= y(0);
 end sample;
```

③ Generic 문

Generic(일반화)이란 Entity 내에 기술하며 Generic 의 매개변수를 Entity 에 전달함으로써, 회로의 개수나 입출력의 크기가 매개변수에 의해 결정되게 하는 것을 말한다. Generic 의 매개변수로 사용되는 객체는 상수이며, 모드는 in 으로 되어 있기 때문에 반드시 명시할 필요가 없다. 또한 이 매개변수는 사용용도에 따라 두 가지로 나뉜다. 반복생성의 개수를 위한 매개상수와 입출력의 크기를 위한 매개상수이다.

Generic 의 사용은 사용할 Component 의 Entity 에 먼저 선언을 하고 다음 Component 를 사용하는 Logic System 에서 이를 이용한다.

◈ 반복생성의 개수를 위한 매개상수

예)
```
--[ 사용될 Component ] --
  entity nand_generic is
```

```
        generic ( size : integer := 8 );
          port ( x, y : in std_logic_vector( size-1 downto 0 );
                    z : out std_logic_vector ( size-1 downto 0 ) );
        end nand_generic;

      architecture sample of nand_generic is
        begin
         z <= x nand y;
        end sample;
```

-- [Generic 으로 선언된 Component 를 사용하는 Logic System] --

```
 entity nand_sys is
     port( a, b : in std_logic_vector( 3 downto 0 );
           c : out std_logic_vector( 3 downto 0 ) );
      end nand_sys;

 architecture sample of nand_sys is
  component nand_generic
    generic( size : integer );
        port ( x, y : in std_logic_vector( size-1 downto 0 );
                  z : out std_logic_vector( size-1 downto 0 ) );
    end component;
   begin
      ux : nandg generic map(4) port map( a, b, c );
  end sample;
```

◈ 입출력의 크기를 위한 매개상수

예) --[사용될 Component] --

```
      entity nandx_generic is
       generic ( size : integer );
         port ( x : in std_logic_vector( size-1 downto 0 );
                 z : out std_logic );
       end nandx_generic;

      architecture sample of nandx_generic is
       begin
        process( x )
            variable temp : std_logic;
          begin
            temp := x(0);
            for I in 1 to size-1 loop
              temp := temp and x(i);
            end loop;
            z <= not (temp);
        end process;
      end sample;
```

-- [Generic 으로 선언된 Component 를 사용하는 Logic System] --

```
      entity nand_sys is
          port( a : in std_logic_vector( 3 downto 0 );
```

```
        b : out std_logic );
end nand_sys;
architecture sample of nand_sys is
  component nandx_generic
  generic ( size : integer );
  port ( x : in std_logic_vector( size-1 downto 0 );
         z : out std_logic );
  end component;
  begin
      ux : nandx generic map(4) port map( a, b );
  end sample;
```

순차 처리문과 병행 처리문

1. 순차 처리문은 Process 문 내에서 기술되며 wait 문, if 문, case 문 및 for-loop 문이 있다. 그러나 각각의 Process 문은 병행처리한다고 말한 바있다. 이제 순차 처리문의 종류를 알아보자.

① Process 문

Process 문은 다음과 같은 형식으로 사용한다. 레이블은 생략해도 되며, 감지신호 리스트는 대개 Process 내의 입력신호들로 구성되며, 이 감지신호의 변화가 생길 때 process 문이 수행된다. 또한 하나의 Entity 내에 여러 개의 Architecture 가 있을 수 있는 것처럼 Process 문 또한 하나의 Architecture 내에 여러 개의 Process 문을 사용할 수 있다.

```
[레이블 :] process [(감지신호 리스트)]
            begin
                순차처리문;
            end process [레이블];
```

② If 문

고급언어와 같이 조건이 참이면 수행되는 if (조건) then 형식이 있다. if 문은 또한 사용방법에 따라 다음과 같이 3 가지의 동작을 할 수 있다.

◈ 일반적 if 문

if 문은 멀티 플랙서(Multiplexer, 이하 MUX 로 표기)를 이용하여 등가적으로 표현할 수 있으며, if 문의 하드웨어 구현에 대한 이해에 도움을 준다.

예)
```
if ( sel = '1' ) then
    y <= a;
else
    y <= b;
end if;
```

◈ 다중 if 문

다중 if 문의 경우에도 같은 방식의 MUX 논리를 적용할 수 있다. 다중 if 문은 여러 개의 조건을 갖는 경우이다.

```
예) if ( sel0 = '1' ) then
        y <= a;
    elsif ( sel1 = '1' ) then
        y <= b;
    else
        y <= c;
    end if;
```

◈ 기억소자가 내포된 if 문

이것은 else 문이 없는 if 문이다. 여기서는 if 문에서는 조건이 거짓일 경우 수행되어야 하는 else 문이 없으므로, else 에 해당되는 출력은 과거의 출력 값을 그대로 유지하게 된다. 따라서 출력은 조건이 거짓일 때는 과거의 출력 값을 지녀야 하므로 VHDL 합성시에 기억소자가 내포된다. 이러한 기억소자가 내포된 if 문의 예가 Latch 와 Flip/Flop 이다. Latch 와 Flip/Flop 의 차이는 if 의 조건이 Level Trigger 에 의해 출력이 변화되면 Latch 이고, if 의 조건이 신호의 상승 혹은 하강에 의해,즉 Rising Edge 나 Falling Edge Trigger 에 의해 출력이 변화되면 Flip/Flop 이다. VHDL 에서 이들을 표현하기 위해 event 라는 예약어를 사용한다.

```
예) if clk'event and clk = '1' then      -- Rising Edge 에서 동작하는 Flip/Flop
        q <= d;
    end if;

    if clk'event and clk = '0' then      -- Falling Edge 에서 동작하는 Flip/Flop
        q <= d;
    end if;

    if en = '1' then                     -- Level Trigger 에 의해 동작하는 Latch
        q <= d;
    end if;
```

③ Case 문

Case 문은 수식 값에 따라 문장을 선택한다. Case 문의 수식 값은 integer 형, 열거형 등과 같은 자료형이 주로 사용되며, 진리표와 같은 기능표에 대한 설계에 적합하다. Case 문은 When 의 값과 비교하여 일치하면 그 문장을 수행한다. When 의 수식을 여러 값으로 표현할 때는 '|'(또는)이라는 기호를 사용한다.

```
예) case sel is
        when "00" => y <= d(0);
        when "01" => y <= d(1);
        when "02" => y <= d(2);
        when others => y <= d(3);
    end case;
```

④ Loop 문

Loop 문은 반복처리 하기 위한 것이다. Loop 문은 For-Loop 형식과 While-Loop 형식 및 단순 Loop 형식이 있다. 형식은 다음과 같다.

```
◆ [ 레이블 ] : for 루프변수 in 변수범위 loop
                  순차 처리문;              -- 변수 범위만큼 반복
               end loop [ 레이블 ];
◆ [ 레이블 ] : while 조건 loop
                  순차 처리문;              -- 조건이 참일 때까지 반복
               end loop [ 레이블 ];
◆ [ 레이블 ] : loop
                  순차 처리문;                -- 무한 반복
               end loop [ 레이블 ];
```

◈ For-Loop 형식

For-Loop 문은 루프변수가 1 씩 증가 또는 감소하면서, 최종값에 도달할 때까지 Loop 문에 둘러싸인 순차 처리문을 반복처리한다. 루프변수는 어떠한 객체로도 선언되지 않아야 되며, 오직 For-Loop 의 루프변수로만 사용되어야 한다.

예)
```
entity and_logic is
    port ( a, b : in std_logic_vector( 3 downto 0 );
              y : out std_logic_vector( 3 downto 0 ) );
end and_logic;
architecture sample of and_logic is
 begin
   process( a, b )
   begin
      for I in 3 downto 0 loop
           y(i) <= a(i) and b(i);       -- 변수 i 에 대해서 4 번 반복
        end loop;
      end process;
end sample;
```

◈ While-Loop 형식

While (조건) Loop 문은 조건이 참이면 loop 에 둘러싸인 순차 처리문을 반복수행한다. While-Loop 문의 조건은 반복횟수가 명확히 결정되지 않으면, 논리합성을 할 수 없으므로 주의해야 한다.단순 Loop 문은 무한히 반복하므로 Loop 를 빠져 나오기 위해서 exit 문이 필요하며, While 문의 경우와 같이 반복횟수가 정해지지 않을 경우에는 VHDL 논리합성이 될 수 없다. 대부분의 VHDL 합성기는 While-Loop 문과 단순 Loop 문을 지원하지 않는다.

2. 병행 처리문

하드웨어 회로에서는 입력선로에서 출력선로로 신호가 전달되어 처리될 때 순차처리되는 것이 아니라 병행처리된다. 따라서 하드웨어 구조를 기술하는 VHDL 의 문장은 병행처리에 기반을 두고 있다고 볼 수 있다. Architecture 문 내부에 표현되는 모든 VHDL 문장은 Process 문의 내부를 제외하고는 모두가 순서에 무관하는 병행 처리문이다. 이러한 병행 처리문에는 조건적 병행처리문(When~Else)과 선택적 병행 처리문(With~Select)이 있다.

① 조건적 병행 처리문 :: 순차처리문인 if 문과 유사하다.

```
signal_이름 <= 파형 1      when (조건 1) else
               파형 2      when (조건 2) else
               파형 n-1    when (조건 n-1) else
               파형 n;
```

예)
```
entity logic is
    port ( a, b, c, d : in std_logic;
                y : out std_logic );
    end logic;
architecture sample of logic is
 begin
     y <=   '0' when d='0' else
            '0' when c='1' else
            '0' when ( a='1' ) and ( b='1' ) else
            '1';
    end sample;
```

② 선택적 병행 처리문 :: with 이하의 수식값에 의해 판단하며, Case 문과 유사하다.

```
with (수식) select
      signal_이름 <=  파형 1      when (선택값 1),
                      파형 2      when (선택값 2),
                      파형 n-1    when (선택값 n-1) else
                      파형 n      when others;
```

예)
```
entity logic is
    port ( x : in std_logic_vector( 1 downto 0 );
           y : out std_logic_vector( 3 downto 0 ) );
    end logic;
architecture sample of logic is
  begin
   with x select
        y <=   "0001" when "00",
               "0010" when "01",
               "0100" when "10",
```

```
                    "1000" when others;
end sample;
```