VHDL이란?

" VHDL의 출현 배경 및 변화 과정"

미국 정부의 VHSIC Program의 주요 목적은 설계, 공정 및 제조 기술 분야에 있어서 미국의 기술 수준을 향상시키는 데 있었다. 또한 미국 정부는 이 Program의 일부로써 VHDL의 개발 노력을 지원하고 있었다. 그 지원 목적은 VHDL의 개발로 좀더 빠른 생산과 회사들간의 연락 기능 강화 및 개발 과정을 능률적으로 처리함으로써 비용 절감 효과를 제공하는 것이었다. 이러한 목적들을 효과적으로 달성할 수 있는 방법을 논의하기 위하여 메사추세츠의 Woods Hole에서 개최된 학술 대회를 시발점으로 1981년부터 VHDL이 개발되기 시작하였다.

Technology independent(기술 독립적)하며 표준 하드웨어 기술 언어의 개발을 목표로 한 Woods Hole 학술 대회에서 정부의 공식적인 제안 요구서를 위한 초안이 제출되었으며, 이것은 전자 공학 분야에 종사하는 전문가들에 의해서 검토되는 과정을 거쳐 1983년 초에 수정, 확정되었다.

1980년대 중반 이후부터 VHDL이 문서용이 아닌 Simulation용으로 검증되어져야 한다는 여론이 강해지면서 몇몇 VHDL Simulator가 등장하였으나 업체간에 표준화가 이루어지지 않은 관계로 호환성에 문제가 있었다. 이러한 문제를 해결하고 늘어가는 VHDL관련 CAD Tool 회사간의 표준 및 호환성을 위하여 IEEE에서 1987년에 IEEE-1076이라는 표준을 만들어 공포하였다.

이 시점에서 Synthesis(회로합성)는 아직 등장하지 않았으며 VHDL은 Simulation용으로 사용되었다. 1990년대에 들어서면서 VHDL 관련 Software 회사가 많이 등장하고 simulation뿐만 아니라 Synthesis의 기능을 갖춘 CAD Tool이 등장하면서 진정한 VHDL의 표준화가 요구되었고 1991년 IEEE-1164이 발표되면서 업체에서 공통으로 사용할 수 있는 VHDL이 탄생하였다.

1. VHDL 이란 ?

VHDL (VHSIC (Very High Speed Integrated Circuits) Hardware Description Language) 은 상위의 동작 레벨에서부터 하위의 게이트 레벨까지 하드웨어를 기술하고 설계하도록 하는 CAD업계 및 IEEE 표준언어이며 미국 정부가 지원을 공인한 하드웨어 설계 언어이다.

VHDL의 등장은 갈수록 복잡해지고 고집적화 되는 회로에 비례하여 어려워지는 하드웨어 설계환경에 새로운 장을 여는 계기가 되었다. 컴퓨터 기술의 발달과 함께 VHDL을 이용한 설계 기법의 발달은 비단 이 분야에 전공하는 사람뿐만 아니라 초보자에게도 보다 쉽게 회로를 설계할 수 있는 기회를 제공하고 있다.

현재 VHDL은 세계적으로 사용되고 있으며 산업체, 대학, 연구소 등에서 그 관심이 급격히 증가하고 있다.

2. ASIC 이란

많은 사람들이 알고 있는 말 중에 하나인 ASIC이란 Application Specific IC의 약자로 우리말로 옮기게 되면 특정용도 주문형 반도체이다. 이 ASIC이란 광범위하게는 특정용도 목적으로 사용되는 모든 반도체를 가리키기도 하지만 좁게는 특히 Gate Array를 가리키는 말이다. 우리가 흔히 ASIC이라 부르며 알고 있는 상식은 Gate Array, Embedded Array, Standard Cell (또는 Cell Based IC)세 종류를 가리키는 말로 알고 있다.

ASIC - Programmable IC Type : PLA, SPLD, CPLD, FPGA
- Memory IC Type: MICOM, ASIC Memory, FIFO
- Logic Based IC Type: Gate Array, Embedded Array,
Standard Cell, Full-Custom IC.

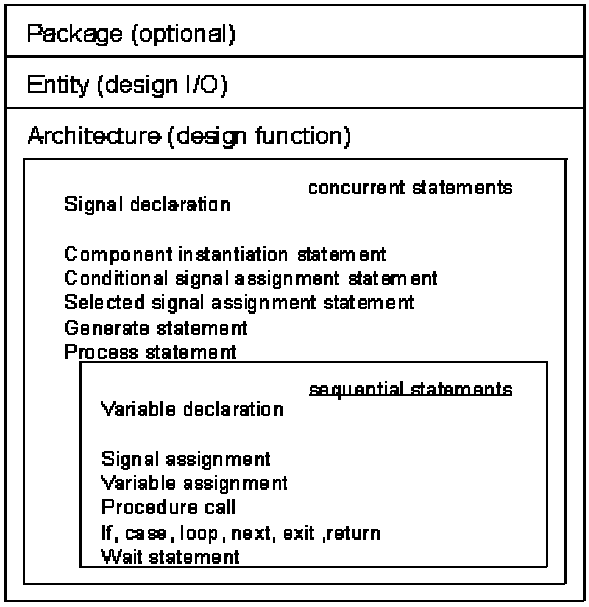
하지만 ASIC이란 반도체 회사에 제품의뢰를 하는 Gate Array 종류만을 가리키는 것이 아닌 특정 용도나 주문형 IC모두를 다 포함하고 있다. 위에서 보는 바와 같이 ASIC의 두 가지 동류로서의 Full Custom(완전 설계 방식)과 Semi Custom(반주문형 설계 방식)이라는 말은 일본에서 인위적으로 만들어 낸 것이다.
물론 ASIC에 대하여 정확하게 정의된 것이 없는 관계로 어쩌면 Gate Array 종류를 진짜 ASIC으로 부는 것도 옳다고 할 수도 있다. 어쨌거나 일반적으로 반도체 설계를 할 수 있는 방식은 일반인들이 ASIC이라 부르고 있는 Logic Based IC Type형태의 Gate Array와 Standard Cell, 그리고 Programmable IC Type형태의 EPLD, FPGA가 대표적이다.

3. FPGA
FPGA란 Field Programmable Gate Array의 준말로 Array Based와 Row Based 두 가지 방법이 있으며 구조는 Gate Array와 매우 흡사하지만 Program에 의해 내부 회로 배선이 연결되는 형식을 취하고 있다.
FPGA는 Logic Cell 위주의 설계 방식이기 때문에 SPLD Block 내부의 배선이 외부와 직접 연결될 수 있도록 고안되어 있어 일반 Gate Array와 매우 비슷하며 Timing Simulation이 반드시 필요하다.
다른 Programmable Device에 비해 속도가 월등히 뛰어나고 집적도가 좋으며 부품 단가도 훨씬 저렴하지만 이 종류는 단 한번밖에 구울 수 없기 때문이 주로 연구 개발용보다는 제품 생산용으로 많이 사용된다.
FPGA는 대개 2,000-20,000 Gates 급의 회로에 적당하며 JEDEC을 이용하여 굽도록 되어 있는 일반 PLD 종류에 비해 Programming 하는 File이 제품을 만든 회사의 고유 Programming Netlist인 ADL, QDF, XNF 등을 사용하도록 되어있다. Array Based 형식의 FPGA는 SPLD Block들을 2차원 배열 형식으로 늘어뜨린 다음 중간에 Interconnect Channel이 서로 교차하며 연결될 수 있도록 되어 있다. 이 Array Based형식의 FPGA는 그림에서 보는 바와 같이 두가지 종류가 있다. 대체로 MUX와 AND로 이루어진 Combinational Logic과 하나의 Flip-Flop으로 구성되어 있는 형식의 SPLD Block구조가 일반적이며 MUX들을 일렬로 배열한 다음 중간 중간에 Flip-Flip을 끼워 넣는 형식의 Logic Array형 제품도 있다. 일반적인 내부구조는 Xilinx의 CLB, Cypress와 QuickLogic의 Logic Cell 등이 대표적인 이 형식을 취하고 있으며 Logic Array형의 내부 구조는 Altera의 LAB가 이러한 형식을 취하고 있다.
반면 Row Based FPGA의 내부 구조는 MUX 구조형식의 Combinational Logic이 각 행마다 나열되어 있고 중간 중간에 Flip-Flop이 끼워져 있는 형식을 취하고 있다. Array Based FPGA의 SPLD Block은 Logic Array 구조 형식이 가로가 아닌 세로로 된것과 비슷하다. 다만 Array Based FPGA와의 다른 점이라면 Logic Cell끼리 연결할 수 있는 Interconnect가 나열된 Cell의 아래나 위에만 위치하여야 된다는 것이다. Actel이 이 Row Based FPGA의 대표적인 구조이다.

VHDL로서 구현한 회로를 PLD나 FPGA로 프로그래밍 해서 사용하기도 하지만 요즘은 주로 Algorithm Test용도로 많이 사용됩니다. PLD나 FPGA Device 가격이 매우 비싸기 때문에 대량 생산 하기에는 무리가 많습니다. ASIC에서는 이론적인 알고리즘을 VHDL을 이용해서

FPGA로 Download해서 실제 칩상에서 동작하는 것을 보고 디버깅 하여 Chip을 설계합니다.

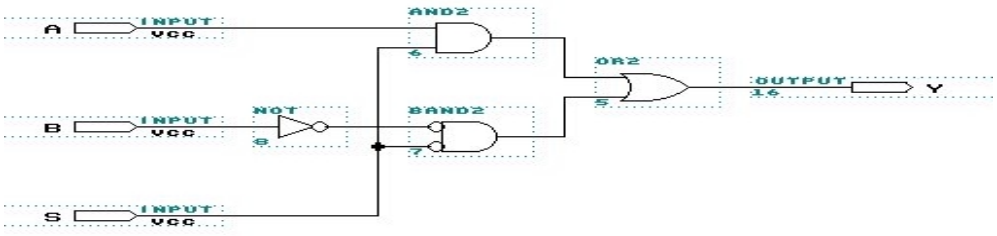
VHDL의 구조



Configuration
Package
Package Body
Entity
Architecture

간단하게 표현하면 위의 5가지로 구성되어 있습니다.
Mux2x1의 예제를 가지고 VHDL의 구성에 대해 설명하도록 하겠습니다.

Mux2x1의 Schematic 회로도



Mux2x1의 VHDL 표현

```
-- mux2x1.vhd

library ieee;
use ieee.std_logic_1164.all;

entity mux2x1 is
port ( a, b      : in std_logic_vector(3 downto 0);
       s         : in std_logic;
       y         : out std_logic_vector(3 downto 0));
end mux2x1;

architecture rtl of mux2x1 is
begin
    process(a, b, s)
    begin
        if ( s = '0') then
            y <= a;
        else
            y <= b;
        end if;
    end process;
end rtl;
```

여기서 진하게 마크된 것은 Reserved word(예약어)이다. 이는 VHDL 구문을 사용하면서 반듯이 지켜야 할 사항이다. (참고로 VHDL은 대소문자를 구분하지 않는다. 변수선언은 예외)

1. Library and Package

LIBRARY

```
library ieee;
```

여기가 LIBRARY 부분이다. VHDL에서는 사용할 라이브러리는 LIBRARY 키워드를 이용하여 지정해 주는데 다음과 같다.

형식

```
LIBRARY [library name];
```

실제로 이 라이브러리가 존재하는 디렉토리는 시뮬레이션 또는 합성툴의 환경 변수로 지정되어 있게 된다. Compass Tool의 경우 Manager 윈도우를 띄우면 라이브러리 패스 (library path)를 지정하는 것을 볼 수 있을 것이다. Max+Plus2 에도 User Library를 지정하는 메뉴가 있고 V-System/VHDL 에도 역시 라이브러리 패스를 지정하는 메뉴가 있습니다. 대개 라이브러리 이름과 디렉토리 패스를 연결시키도록 되어 있는데, 라이브러리 명(library name)과 패스(directory path)를 매핑(mapping) 시킨다고 한다.

Package

use ieee.std_logic_1164.all;
이는 ieee라는 라이브러리에서 std_logic_1164 라는 이름의 패키지(package)를 가져다가 그 안에 있는 함수(function)들을 모두 사용한다는 뜻으로 ALL 이라고 한 것이다. ALL은 VHDL 의 키워드이다. 물론 ALL 대신 패키지 내에서 사용할 함수를 지정할 수도 있다.

형식
USE [library name.package name. function name];

사용툴의 Library 디렉토리에 가면 ieee.vhd 라는 파일이 있다. 이것을 직접 확인해 보는 것도 좋은 방법이다. 각각의 툴마다 조금씩 차이가 나는 것을 알 수 있을 것이다.

2. entity

```
entity mux2x1 is
port ( a, b      : in std_logic_vector(3 downto 0);
           s      : in std_logic;
           y      : out std_logic_vector(3 downto 0));
end mux2x1;
```

시스템을 구성하는 부분품으로서 이들 사이의 상호 연결을 위한 통로 역할(interface)을 하는 것이 ENTITY이다. 말하자면 내부 설계에 대한 입출력 등을 기술하는 “포장”인 셈이다. 시스템의 입장에서 보면 내부 설계에 대한 것은 가려져 있고 다만 ENTITY에 기술된 입출력 포트만을 보게되는데 이런 의미에서 ENTITY를 “암흑상자”(Black Box)라고 하기도 한다.

위의 예제에서 보면 Entity name은 mux2x1이고 각종 포트에 대한 설명을 한다. 몇몇의 툴에서는 Entity name과 파일의 이름이 일치해야 하는 경우도 있으니 참조하기 바란다(Max Plus II의 경우). 입력과 출력에 대한 포트의 설명이다. **포트 name은 Reserved word와 일치해서는 안 된다.(중요)** std_logic 이외에도 객체형(Object Types)이 있지만 뒤에 설명하는 것이 좋겠다.

형식

```
entity 이름 is
        generic(매개변수 리스트);
        port(매개변수 리스트);
        {declaration}
begin
        {statement}
end entity_name;
```

is

entity의 이름 뒤에 붙는다.

generic

설계 매개변수를 회로에 전달함으로써 표현을 특정 크기나 개수 등에 국한시키지 않고 보다 일반화시키기 위해서 사용. 매개변수 리스트를 통해서 타이밍, 부속 컴포넌트의 수, 비트 수, 온도 등과 같은 것을 사례화된 컴포넌트에 전달할 수 있도록 한다.

port

괄호안의 매개변수 리스트를 통해서 외부와의 연결을 나타낸다. 즉 포트이름, 신호 흐름, 자료형을 나타낸다.신호의 흐름은 **in, out, inout, buffer, linkage** 의 다섯가지가 있으며, **in**은 신호가 entity로 들어오는 경우만 가능하고, **out**은 entity로부터 나가는 경우에만 가능하다.
그리고 입출력 모두 가능한 경우에는 inout을 사용한다. 또한 out에서는 나가는 신호를 entitry 내에서 다시 읽을 수 없다. 이를 가능하도록 기능을 추가한 것이 buffer이다. buffer는 외부로부터의 입력은 허용되지 않지만, 출력값을 내부회로에서 다시 읽을 수 있다.

inout은 입출력 포트로 사용되므로 소스(source) 또는 목적지(destination)에 모두 쓸 수 있다. 그러나 한 개의 문장에 모두 쓸 수 없다. 입출력 모드가 정해진 경우 할당문(assign statement)에서의 엄격한 규정은 하드웨어를 다루기 때문이다. 하드웨어를 기술할 때 assign 이란 연결관계를 표현한 것(netlist)으로서 프로그래밍 언어에서와 같은 오퍼레이션(move operation)이 아니다.

buffer는 inout과 같은 입출력 포트로서 assign의 소스(source)측 또는 목적지(destination)측에 쓸 수 있다. inout 모드와 다른 점은 단일 할당문 내에서 소스와 목적지측 모두 동시에 사용할 수 있다는 것이다. 이는 buffer 모드에 이미 F/F을 내포하고 있다는 의미를 갖는다.

linkage는 동작에 영향을 주지 않으며 단지 포트로서 연결된 경우를 말한다.

1비트의 경우는 std_logic으로 표현하고, n비트의 경우는 std_logic_vector(n downto 0)라고 표현한다.

```
std_logic_vector(0 to 3)
std_logic_vector(3 downto 0)
```

위의 두 가지 모두 MSB가 왼쪽에 있는 4비트를 표현한 것이다.

3. architecture

다음은 architecture 부분이다.

```
architecture rtl of mux2x1 is
begin
    process(a, b, s)
    begin
        if ( s = '0') then
            y <= a;
        else
            y <= b;
        end if;
    end process;
end rtl;
```

Architecture name과 Entity name을 다르게 잡아주는 것이 좋다고 생각된다. Architecture name과 Entity name을 정할 때 그 파일이 어떤 파일인지 자기가 구별할 수 있도록 하는 습관은 좋은 습관이다. 모든 Architecture내부에서는 모든 문장들이 Concurrent(동시에 수행)하게 동작된다. 모든 하드웨어 설계의 기본의 동작이 Concurrent하게 발생된다. Sequential(순차적으로)하게 표현하기 위해서 Process 문을 사용한다. Process 문안에서의 모든 동작은 순차적으로 발생된다. 즉, Process내부는 Sequential하게 동작하지만 각각의 Process문은 Concurrent하게 동작한다는 것이다.

하나의 Architecture 안에 다수의 Process 문을 사용할 수 있다. 그리고 하나의 Entity에는 다수의 Architecture 문을 구성할 수 있다.

형식

```
architecture 아키텍쳐_이름 of 엔티티_이름 is
        {declarations}
begin
        internal Behavior Description
end 아키텍쳐_이름;
```

declarations

아키텍쳐의 선언문은 begin과 end 사이에서 사용할 signal, type, variable, constant 등을 선언한다.

internal Behavior Description

하드웨어의 내부적 동작은 여러 가지로 표현할 수 있다.

- 동작적 기술, 자료흐름 기술, 구조적 기술, 혼합적 기술

동작적 기술이란 하드웨어적인 구조와 상관없이 고급 프로그래밍언어를 사용하여 알고리즘을 표현한 것이며, 자료흐름 구조는 연산자, 함수 등을 사용하여 RTL 레벨에서 신호의 흐름을 나타내는 것이다. 또한 구조적 기술은 하드웨어 구조에 가장 가까운 표현으로서 연결상

태 등을 나타낸다.

4. component

- instantiation

컴포넌트 선언에서 선언된 컴포넌트를 사용하여 실제 설계상에 필요로 하는 예를 지정해 준다. 즉 설계에서 실제로 사용되는 부속 컴포넌트로 컴포넌트 선언에서 정의된 요소를 사용하여 포트와 신호를 연결하여 표현한다.

레이블 : 컴포넌트_이름
[generic map(결합_리스트)]
[port map(결합_리스트)]
레이블이 가장 먼저 오며 컴포넌트_이름은 반드시 컴포넌트 선언에서 선언된 이름이어야 한다. generic map과 port map은 설계하고자 하는 회로의 실제 generic과 port값을 컴포넌트에 있는 generic과 port를 결합_리스트를 통해서 연결 또는 mapping 시켜 준다.

```
__instance_name: __component_name
        PORT MAP (      __formal_parameter => __actual_parameter,
                        __formal_parameter => __actual_parameter);
```

- declaration

한 설계 내에서 필요로 하는 컴포넌트들을 정의 해주는 구문이 컴포넌트 선언이며, 같은 컴포넌트를 여러 곳에서 반복적으로 사용할 수도 있다.

```
COMPONENT __component_name
        PORT(
                __input_name, __input_name              : IN      STD_LOGIC;
                __bidir_name, __bidir_name              : INOUT STD_LOGIC;
                __output_name, __output_name    : OUT   STD_LOGIC);
END COMPONENT;
```

```
std_logic
                'U',   -- Uninitialized
                'X',   -- Forcing   Unknown
                '0',   -- Forcing   0
                '1',   -- Forcing   1
                'Z',   -- High Impedance
                'W',   -- Weak      Unknown
                'L',   -- Weak      0
                'H',   -- Weak      1
                '-'    -- Don't care
```

5. configuration Specification

컴포넌트 선언은 설계에 사용될 컴포넌트를 선언해 주며, 여기서 선언된 컴포넌트를 사용하여 컴포넌트 사례화문을 실제적으로 연결함으로써 설계를 한다.
configuration specification 은 instantiation에서 사용되는 컴포넌트 각각에 어느 라이브

러리 내에 있는 어떠한 엔티티 선언 그리고 그 엔티티의 어느 아키텍쳐를 instantiation 할 것인지를 지정해 준다.
for 컴포넌트_명세 use 바인딩_표시

6. concurrent statement

하드웨어 시스템의 모듈들은 서로간에 병행적으로 수행되므로 시스템의 동작으로 표현하기 위해서는 일반적으로 concurrent statement를 사용하는 것이 바람직하며 그 내부의 동작은 순차적으로 이루어진다고 볼 수 있으므로 sequential statement를 사용하는 것이 좋다, 프로세스문 자체는 병행문이므로 여러개의 프로세스문이 있으면 이들은 병행적으로 수행되며 프로세스문 내부는 하나씩 차례로 수행되는 sequential statement로 표현된다.

Concurrent Procedure Call
__label: __procedure_name(__actual_parameter, __actual_parameter);
Concurrent Signal Assignment Statement
__signal <= __expression;

7. delay modeling

설계를 한다는 뜻으로 어떻게 설계할 것인지는 설계자에 달려있다. 그러나 어떤 방식의 설계가 있는지는 알고 있어야 한다. 모든 것의 기초가 되는 것이다. 어떻게 코딩할 것인지를 선택해야 한다.
Modeling abstraction의 종류에는 세 가지가 있습니다. 먼저 **Behavioral Level**(동작적인 모델링 - 에 대해서 먼저 알아봅시다. 이 레벨에서는 구체적인 하드웨어 시스템이 어떻게 동작해야 하는가를 기술한 최상위 모델링 기법을 제공한다. 각종 시스템 설계용 알고리즘들이 어떻게 동작하는가를 비교 평가한다든지 이들 알고리즘들이 예측 가능한 입력 조건하에서 제대로 작동하는지 알아보고 싶을 때, 그리고 구체적인 회로 설계 과정에서 하드웨어 구현 방식과 설계 사양 등을 함께 고려하고자 할 때 매우 유용하기 때문입니다. 이 레벨에서는 구체적인 정보보다는 시스템 레벨의 추상적인 기술형태이다.
Architectural Level의 설계가 끝나면 다음으로 **Logic Level**(다시 말하면 RTL Level)의 설계를 합니다. 이 레벨은 논리 회로 설계시 상세한 블록 다이어그램과 같습니다. 블록 다이어그램에서 하드웨어 함수 블록은 그 블록에 연결된 입출력 신호 및 데이터 버스 등을 통해 알아낼 수 있다. 이 레벨에서는 Boolean 식에서 RTL에 이르기까지 다양한 방식으로 기술할 수 있습니다. 흔히 RTL 모델링을 **Data flow description**이라고 합니다.
다음 단계로는 **Structural Level**입니다. 이 레벨에서의 코딩 구문은 netlist 표현 방식과 유사하다. 게이트 또는 플립플롭이거나 동작적 기술 레벨 내지는 RTL 코드로 기술된 Component들이 선으로 연결된 회로로 표현됩니다.
VHDL에서는 설계 과정에서 이들 세 가지 기술 형태를 임의로 혼합된 형태도 허용합니다. 이 세 가지 레벨이 기술되면 이 각각의 과정을 Synthesis 과정을 수행합니다.
Behavioral level synthesis, Dataflow level synthesis, Structural level synthesis를 수행합니다. 소스를 코딩했다면 이것을 다시 게이트 레벨이나 RTL 레벨로 바꾸어 주어야 합니다. Synthesis란 번역만의 작업이 아니다. 소스 코드의 **Translation**(번역)뿐만 아니라 회로의 **Optimization**(최적화) 기능을 동시에 수행합니다. 합성을 하는 이유는 설계 생산성 향상을 도모할 수 있고(Schematic에 비해 10배정도), Time-to-market(시장 출하 시간)을 단축시킬 수 있고, 시스템 수준의 설계를 가능하게 해줍니다.
Asic에서의 합성 단계를 보면 첫 번째로 상위 합성(High-level(Architectural) synthesis)을 거치고 다음으로 논리 합성(Logic synthesis)을 거칩니다. 다음 단계로는 테스트 합성(Test synthesis) 과정을 거치고, 그 다음으로 레이아웃 합성(Layout synthesis)을 거칩니

다. 여기서 테스트 합성이란 회로 테스트를 위해서 넣어주는 회로에 대한 합성을 말합니다.

8. operator

구분	종류	우선 순위
논리 연산자	And, or, nand, xor	6
관계 연산자	=,/=,<,<=,>,>=	5
가감산 연산자	+,-,&	4
부호	+,-	3
승제산 연산자	*,/, mod, rem	2
기타 연산자	**,abs, not	1

기타연산자입니다. **, abs, not 가 있습니다. **는 지수 계산을 위한 연산자가 왼쪽 피연산자는 integer type이나 floating point type을 사용할 수 있지만 오른쪽 피연산자는 integer type만 가능합니다. 그 결과는 왼쪽 피연산자와 같은 type이 됩니다. **abs**는 절대값의 계산을 위하여 사용되며 모든 numeric 피연산자에 대해서 적용할 수 있습니다. **not**는 단항 연산자로서 피연산자의 논리값을 참이면 거짓으로, 거짓이면 참으로 바꾸는 역할을 합니다.

다음은 **곱셈연산자**입니다. *, /, mod, rem이 있습니다. 곱셈 연산자(*)와 나눗셈연산자(/)는 왼쪽 및 오른쪽 피연산자는 같은 data type이어야 하며 integer type 또는 floating point type이 가능하고 그 결과도 피연산자와 같은 data type이 되어야 합니다.
나머지 계산(rem)과 모듈 계산(mod)의 경우에는 왼쪽 오른쪽 연산자 모두 integer type이야 하고 그 결과도 integer type이고 그 결과도 integer type이 된다. A rem B는 A의 부호를 따르고 그 결과의 절대값은 B의 절대값을 따른다.

다음으로 우선 순위가 높은 것은 **단항연산자**가 있다. +/-의 보호를 나타내는 연산자이다. 그렇게 설명은 필요하지 않다고 본다. 다음은 **덧셈연산자**가 있다. 실제 연산을 수행하는 +/-와 &(접속 연산자)가 있다. 다음의 우선 순위가 있는 것은 **관계연산자**가 있다. =, /=. >, <, <= 등이 있다. 우선 순위가 가장 낮은 것은 **논리연산자**가 있다. or, and, nor, nand, xor, xnor 등이 있다.

9. block / scope

block statement는 설계상의 어떤 부분을 돋보이게 하기 위해서 레이블을 붙이고 구문적 괄호로소 block 과 end block으로 둘러싼다. 이 구문적 괄호는 의미나 해석에 영향을 미치지 않는다.
scope는 다중 block문 내에서 안쪽 block문과 바깥쪽 block문에 동일한 변수명으로 선언했을 경우 변수명 그대로 사용할 경우 안쪽 block문의 변수로 취급하며, 만일 바깥쪽의 block문의 변수를 사용해야할 경우 ‘ . ‘ 으로 block의 lable과 함께 표시를 해주어야 한다.

10. attribute

자료형에 대한 동작이나 상태를 표현하도록 하는 지정된 특성이다.
객체는 어떠한 순간에 단 하나의 값만 가지지만 여러가지의 속성을 가질수 있다.

Powered by VLSI/CAD LAB of Daejin University

If you have any question
Mail to somehero@dreamwiz.com