



VHDL 이론 및 실습

주식회사 쿼텀베이스

 QuantumBase

Sales@quantumbase.com

www.quantumbase.com



VHDL개요

✚ Very High-speed Hardware Description Language

- 디지털 시스템 모델링
- 순차 또는 병행 처리
- 빠른 제품 개발에 대응
- 이식성이 우수한 하드웨어 개발 방법

✚ VHDL 표준

- VHDL IEEE std 1076-1987
- VHDL IEEE std 1076-1993

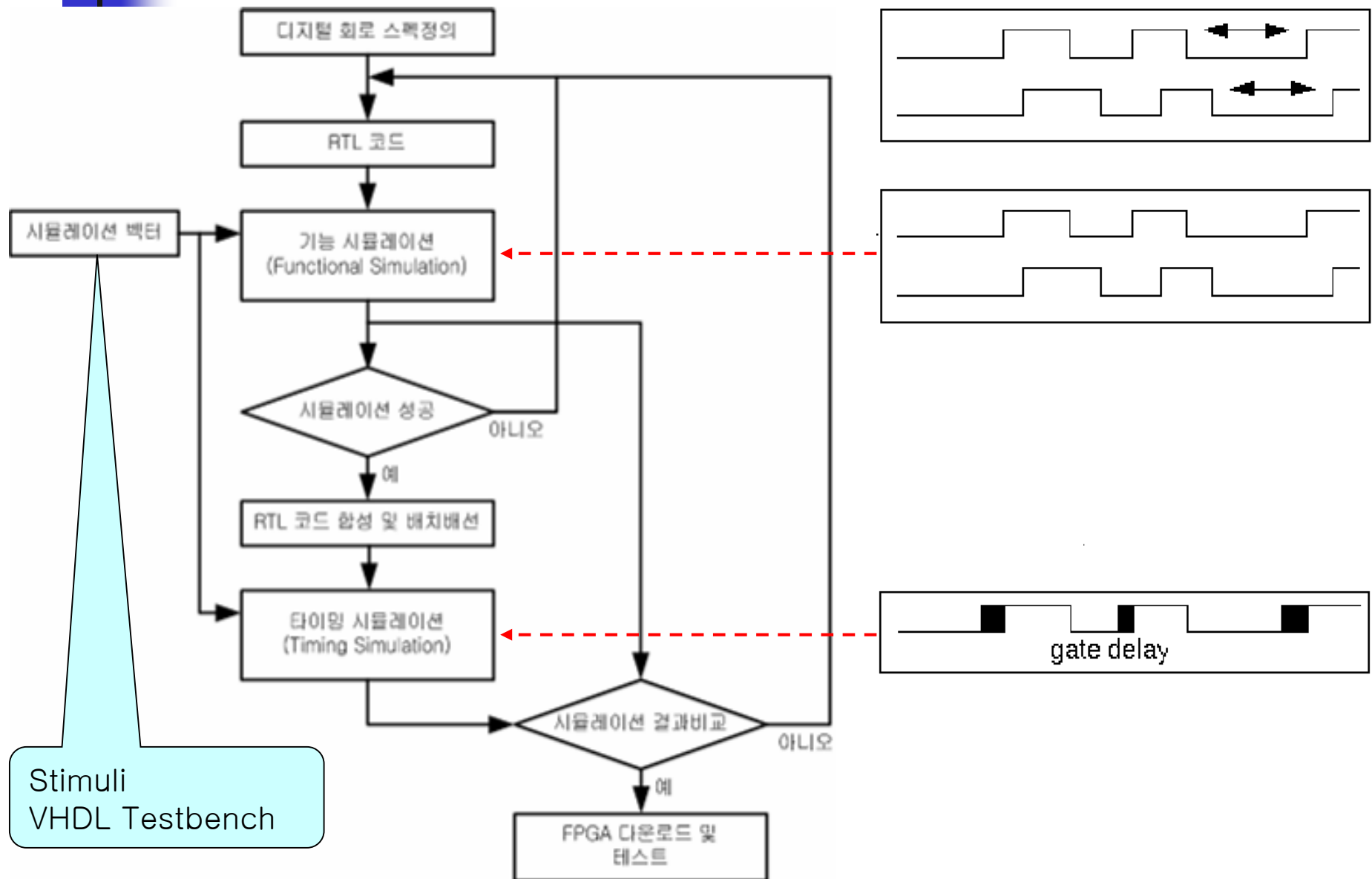
✚ 아날로그 디지털 혼합 확장 표준: VHDL-AMP



VHDL 역사

- ✚ 70년대 초: 초기 토론
- ✚ 70년대 후: 요구사항 정의
- ✚ 83년 : IBM, Intermetrics, TI 공동 개발 협약
- ✚ 84년 중반: Version 7.2
- ✚ 86년 중반: IEEE-Standard
- ✚ 87년: 미 국방성 표준으로 채택 -> IEEE 1076'87
- ✚ 93년: IEEE에서 공식적으로 업그레이드 -> IEEE 1076'93
- ✚ 1999년: VHDL-AMP 확장

FPGA를 이용한 디지털 회로설계 흐름





VHDL 코딩 템플릿

```
-----  
-- Entity: Coding example based on the code template  
--  
-- FILE NAME:      frametest.vhd  
-- VERSION:        1.0  
-- DATE:  
-- AUTHOR:  
--  
-- CODE TYPE: Behavioral Level  
--  
-- DESCRIPTION: This code shows how the template can be used  
-- in real VHDL coding  
--  
-----  
-- INCLUDE LIBRARY  
library IEEE;  
use IEEE.Std_Logic_1164.all;  
use IEEE. Std_Logic_arith.all;  
use work.cpackage.all;  
-- ENTITY DEFINITION  
entity frametest is  
    port( A1, A2, B1, B2, C1, C2: in std_logic;  
          Y1:                      out unsigned(7 downto 0);  
          Y2:                      out integer range 0 to 50;  
          PrimeColor: out PrimeColorType);  
end entity frametest;
```



VHDL 코딩 템플릿 *-continued*

-- ARCHITECTURE DEFINITION

architecture LOGIC of frametest is

-- SIGNAL, CONSTANT, FUNCTION DECLARATIONS

constant CST: unsigned(3 downto 0) := "1010";

constant TwentyFive: integer := 25;

begin

-- PROCESS DISCRIPTION

process (A1, A2, B1, B2, C1, C2)

begin

if (A1 = '1') then

Y1 <= CST & "0101";

elsif (A2 = '1') then

Y1 <= CST & "0111";

else

Y1 <= CST & "1111";

end if;

if (B1 = '1') then

Y2 <= 10;

elsif (B2 = '1') then

Y2 <= 15;

else

Y2 <= TwentyFive + 10 + 15;

end if;

if (A1 = '1') then

PrimeColor <= Red;

elsif (A2 = '1') then

PrimeColor <= Green;

VHDL 코딩 템플릿 -continued

```
        else
            PrimeColor <= Blue;
        end if;
    end process;
end architecture LOGIC;
-- CONFIGURATION DESCRIPTION
```

Code 템플릿 사용방법

- “--”로 시작되는 설명문으로 설계정보를 기입
- 설계 버전과 설계일자, 그리고 설계자 정보 기입
- 코드 형태 기술
- 현 엔티티의 기능 간략설명 포함
- 각 내부의 요소마다 그 시작 전에 설명문 삽입
- 코딩 시 가능한 한 많은 설명문을 포함

Code 템플릿 사용의의

- 코드를 읽기 쉽게 함(readability증대)
- 코드의 이식성 증대

HDL의 사용 목적과 일치

VHDL의 설계 구조

+ 설명문

- “--”를 사용하면 그 두개의 하이픈 이후의 동일 라인은 모두 설명문으로 취급됨
- 실제 하드웨어와는 전혀 상관없는 문
- 가독성과 이식성을 극대화 하기 위하여 가능한 많은 설명문을 사용함

+ 설명문 사용 예

```
-- INCLUDE LIBRARY
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_arith.all;
use work.mypackage.all;      -- New data type definition

-- ENTITY DEFINITION
entity test is
    ...
end entity test;
...
```


VHDL의 설계 구조 *-continued*

엔티티(Entity)

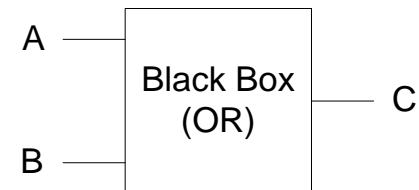
- 설계하고자 하는 디지털 회로를 블랙 박스로 놓았을때 최 외각에서 보이는 모양을 기술한 것
- 설계 회로 블록의 입력과 출력 입출력 포트를 명시
- 디지털 회로 블록의 가장 기본이 되는 정의를 내포

Entity 정의 방법

```
entity 엔티티이름 is
  port(
    포트이름 {, 포트이름 , ...} : [모드] 데이터형 {;
    포트이름 {, 포트이름 , ...} : [모드] 데이터형 } );
end entity 엔티티이름;
```

Entity 정의 예

```
entity LOGIC_OR is
  port ( A, B: in std_logic;
         C:   out std_logic);
end entity LOGIC_OR;
```





VHDL의 설계 구조 -continued

+ 식별자(Identifier) 작성규칙

- 첫자는 반드시 영문자
- 나머지는 영문자, 숫자, 밑줄 '_' 사용 가능
- '_'는 이름의 마지막에 사용할 수 없음
- 두개의 '_'가 연속되면 안됨
- 대문자와 소문자 구별이 없음

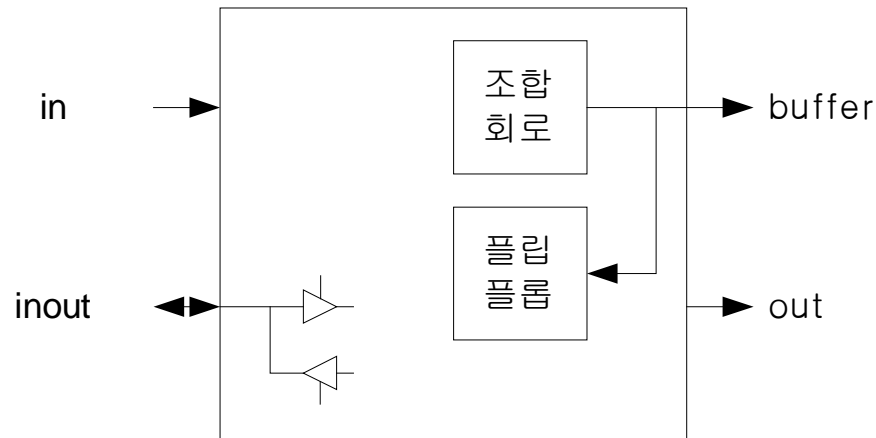
+ 엔티티이름: 식별자 작성규칙을 따라 작성

+ 포트이름: 식별자 작성규칙을 따름

+ 모드: in, out, inout, buffer, linkage이며 주로 사용되는 모드는 in, out, inout, buffer 임

VHDL의 설계 구조 -continued

✦ 포트의 주요 4가지 모드의 특징



✦ 아키텍처(Architecture)

- 앞서 정의한 엔티티의 실제 동작을 기술하는 부
- 하나의 엔티티에 여러 개의 아키텍처가 존재할 수 있음
- 하나의 엔티티에 다수의 아키텍처가 존재하는 경우, 그 중 실제 시뮬레이션 및 합성시에 사용될 아키텍처를 최종 연계시켜 사용

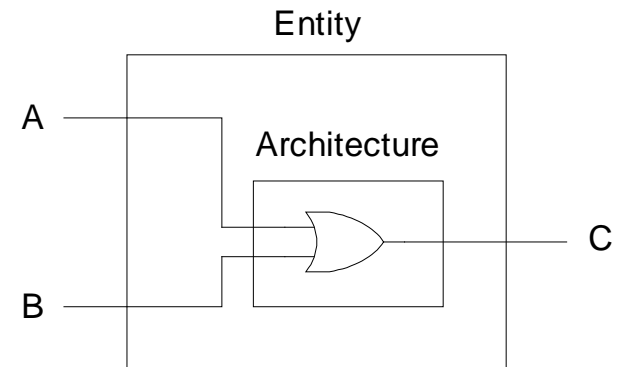
VHDL의 설계 구조 *-continued*

아키텍처 정의 방법

```
architecture 아키텍처이름 of 엔티티이름 is  
[선언문]  
begin  
    {동작 기술}  
end architecture 아키텍처이름;
```

아키텍처 정의 예

```
library IEEE;  
use IEEE.Std_Logic_1164.all;  
entity LOGIC_OR is  
    port ( A, B: in std_logic;  
          C:   out std_logic);  
    -- '87 표준의 경우에는  
    -- end LOGIC_OR  
end entity LOGIC_OR; -- '93 표준
```





VHDL의 설계 구조 *-continued*

```
architecture RTL of LOGIC_OR is
begin
    process ( A, B )
    begin
        C <= A or B;
    end process;
end architecture RTL; -- '93 표준
-- '87 표준의 경우에는
-- end RTL
```

- 아키텍처 정의에서의 [선언문]
 - ▶ 동작 기술시 사용되는 data type, function, variable, signal, component, constant 등이 선언되는 부

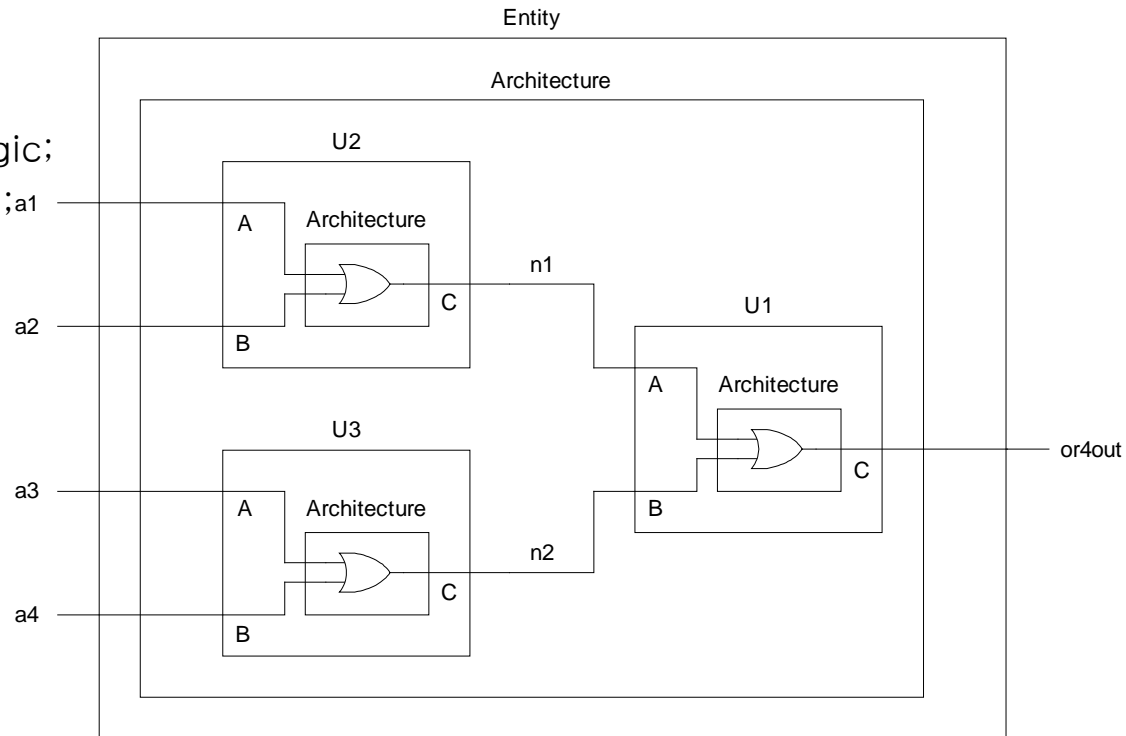
VHDL의 설계 구조 -continued

VHDL 계층적 설계 예

```
library IEEE;
use IEEE.Std_Logic_1164.all;
entity OR4 is
    port (a1, a2, a3, a4: in std_logic;
          or4out: out std_logic);
end entity OR4;
architecture MODEL of OR4 is

    component LOGIC_OR
    port ( A, B: in std_logic;
          C: out std_logic);
    end component;

    signal n1, n2: std_logic;
begin
    U1: LOGIC_OR port map(A => n1, B => n2, C => or4out);
    -- 위치연결을 사용한 경우
    -- U1: LOGIC_OR port map(n1, n2, or4out)
    U2: LOGIC_OR port map(A => a1, B => a2, C => n1);
    U3: LOGIC_OR port map(A => a3, B => a4, C => n2);
end architecture MODEL;
```





VHDL의 설계 구조 *-continued*

- ✦ Configuration(엔티티-아키텍처 조합구성)
 - 하나의 엔티티가 여러 아키텍처를 포함할 때, 특정 아키텍처를 지목하여 연결
 - 계층적 설계에서 특정 엔티티와 부품개체(component instance)를 연결

✦ Configuration 정의방법

```
configuration 조합구성이름 of 엔티티이름1 is
  for 아키텍처이름
    {for 개체라벨: 부품이름
      use entity 라이브러리이름.엔티티이름2(아키텍처이름);
    end for;}
end configuration 조합구성이름;
```



VHDL의 설계 구조 *-continued*

+ Configuration문 사용 예

```
library IEEE;
use IEEE.Std_Logic_1164.all;
entity LOGIC_OR is
    port ( A, B: in std_logic;
          C:  out std_logic);
end entity LOGIC_OR;

architecture RTL1 of LOGIC_OR is
begin
    process ( A, B )
    begin
        C <= A or B;
    end process;
end architecture RTL1;
```

```
architecture RTL2 of LOGIC_OR is
begin
    process (A, B)
    begin
        C <= '0' when (A='0' and B='0') else '1';
    end process;
end architecture RTL1;

configuration conf of LOGIC_OR is
    for RTL2
        -- 엔티티 LOGIC_OR와 RTL2를 연결시킴
    end for;
end configuration conf;
```


VHDL의 설계 구조 *-continued*

+ 함수(Function) 및 프로시저(Procedure)

- 함수와 프로시저는 주로 패키지에 포함되어 사용되는 subprogram임

+ 함수의 형식

```
-- 선언부(Declaration Part)
function 함수이름(매개변수리스트) return 데이터형;
-- 몸체부(Body Part)
function 함수이름(매개변수리스트) return 데이터형 is
    {선언문}
begin
    {순차문}
end function 함수이름
```

- + x와 y를 로직 or하여 그 결과를 돌려주는(return) 함수를 기술한 경우의 예



VHDL의 설계 구조 *-continued*

```
package FUNC_TEST is
    function LOG_OR ( x, y: bit ) return bit;
end package FUNC_TEST;
package body FUNC_TEST is
    function LOG_OR ( x, y: bit) return bit is
        variable RetOr: bit;
    begin
        RetOr := x or y;
        return RetOr;
    end function LOG_OR;
end package body FUNC_TEST;
```

- 매개변수 리스트의 매개변수 모드로서 in, out, inout을 사용할 수 있으며, 위와 같이 모드를 지칭하지 않는 경우는 in으로 인식 함
- package body에서 패키지 FUNC_TEST의 동작이 기술됨
- 예와 같이 함수에서는 입력변수는 입력 매개변수 리스트에, 출력변수는 별도 return문에 기술됨



VHDL의 설계 구조 *-continued*

함수 호출 예

```
-- include user defined package
use work.FUNC_TEST.all;
entity ORTEST is
    port(x, y: in      bit;
          z: out      bit);
end entity ORTEST;
architecture TEST of ORTEST is
begin
    z <= LOG_OR( x, y); -- function call
end architecture TEST;
```

- 현재 작업하고 있는 디렉토리가 work 라이브러리 임.
- use문에 work 라이브러리 밑의 FUNC_TEST 패키지를 사용할것을 정의 함.
- function call 부분에서 x,y를 매개변수로 받아 LOGIC_OR의 함수에서 처리되어 그 결과를 z로 돌려줌.

VHDL의 설계 구조 *-continued*

프로시저의 형식

```
-- 선언부(Declaration Part)
procedure 프로시저이름(매개변수리스트);

-- 몸체부(Body Part)
procedure 프로시저이름(매개변수리스트) is
    {선언문}
begin
    {순차문}
end procedure 함수이름
```

x와 y를 로직 or하여 그 결과를 z로 할당하는 프로시저 예

```
package PROC_TEST is
    procedure LOG_OR (signal x, y:in bit; signal z:out bit );
end package PROC_TEST;
package body PROC_TEST is
    procedure LOG_OR ( signal x, y:in bit; signal z:out bit ) is
    begin
        z <= x or y;
    end procedure LOG_OR;
end package body PROC_TEST;
```

매개변수 리스트의 매개변수 모드로서 in, out, inout을 사용할 수 있으며, 위와 같이 모드를 지칭하지 않는 경우는 in으로 인식 함
package body에서 패키지 FUNC_TEST의



VHDL의 설계 구조 *-continued*

프로시저 호출 예

```
-- include user defined package  
use work.PROC_TEST.all;
```

```
entity ORTEST is
```

```
    port(x, y: in      bit;  
          z: out      bit);
```

```
end entity ORTEST;
```

```
architecture TEST of ORTEST is
```

```
begin
```

```
    LOG_OR( x, y, z); -- procedure call
```

```
end architecture TEST;
```

- use문에 work 라이브러리 밑의 PROC_TEST 패키지를 사용할 것을 정의 함.
- Procedure call 부에서 보면 입력신호와 출력 신호가 모두 매개 변수 리스트 내에 존재함
- 함수와 같이 결과를 돌려주는 형태가 아님

VHDL의 기본 표현요소

표현(Expression)

$$Y \leq \overbrace{A + B - C}^{\text{표현식}}$$

연산자(Operator) : +, - 피연산자(Operand) : A, B, C

피연산자(Operand)

- 리터럴 피연산자(literal operand)

- ▶ 문자열 리터럴(string literal)
예) "ABC"

- ▶ 비트열 리터럴

2진, 8진, 16진 이냐에 따라 B, O, X를 앞에 쓰고 다음에 큰 따옴표로 둘러싸인 값이 기술됨

예) B"1101", O"15", X"BF"

(note: 단일 리터럴은 '1', '0' 과 같이 작은 따옴표 (' ')로 둘러싼 하나의 숫자)

VHDL의 기본 표현요소

▶ 숫자 리터럴(numeric literals)

정수(integer), 실수(real), 물리적표현(physical expression)
(단, 합성을 위해서는 정수리터럴만을 사용함)

| | | |
|----|--------------|---|
| 예) | 1357 | -- 정수 1357 |
| | 2#1010_1111# | -- 2진수 10101111 |
| | 8#132# | -- 8진수 132 |
| | 16#FF# | -- 16진수 FF |
| | 16#F.01#E+2 | -- 16진수 $15.00390625 \times 16^2 = 3841.0010$ |
| | 12 ft | -- 거리 feet |
| | 3 kohm | -- 저항 kohm |

문 (Statement)

● 선언문(declaration statement)

설계중에 사용될 constant, data type, object, subprogram을 정의

● 병행처리문(concurrent statement)

하드웨어에의 특징인 병렬로 처리되는 하드웨어의 동작이나 구조를 기술하는데 사용



VHDL의 기본 표현요소

예) 하나의 architecture에 여러 개의 process문이 있다면 이 각각의 process문은 병렬로 동작하는 블록이 됨

- 순차 처리문(sequential statement)
기술된 순서대로 수행됨, (process문 내, 함수, 프로시저)

Note:

- 본 절에서 말하는 병행처리, 순차처리는 실제 이것이 조합회로인지 아니면 순차회로인지를 결정하는 것이 아님.
- 이것은 합성결과가 동일하다 할지라도 시뮬레이션은 순차연산을 기반으로 하기 때문에 시뮬레이션 결과가 다르게 나올 수 있음을 의미함



데이터 객체(data object)

- VHDL에서는 상수(constant), 변수(variable), 신호(signal)와 파일(file)과 같이 임의의 값을 가지고 있는 것을 지칭하며,
- 모든 객체는 데이터형을 가짐.(단 합성에 관계있는 것은 constant, variable, signal)

VHDL의 기본 표현요소

- 변수(variable)와 신호(signal)

- ▶ variable: process, function, procedure와 같은 subprogram에서 정의되며, 값의 할당시에는 할당과 동시에 즉시 값이 갱신됨
- ▶ signal: entity, architecture, package, block에 선언되며 포트와 같이 값의 할당시에는 일정 지연후에 값이 갱신

- variable 및 signal 정의

```
-- 변수선언
variable 변수이름{,변수이름,...}:데이터형 [:= 초기값]

-- 변수할당
변수이름 := 할당값

-- 신호선언
signal 신호이름{,신호이름, ...}:데이터형 [:= 초기값]

-- 신호할당
신호이름 <= 할당값
```



6. VHDL의 언어 문법

■ 데이터 형(Data type)

- numeric(정수 또는 실수), 부울형식(Boolean), 문자(character), time(fs에서 hr까지), string(문자열), bit(0 또는 1), bit_vector (비트의 배열)의 기본 데이터 형 존재
- IEEE에서는 std_logic_1164 패키지에 0과 1뿐만 아닌 다양한 상태를 표시할 수 있는 데이터 형인 std_logic, std_logic_vector, std_ulogic, 과 std_ulogic vector를 포함(∵ bit만으로는 고임피던스 상태나, Don't care상태 등 설계에 필요한 상태를 모두 표시할 수 없음)
- std_logic과 std_ulogic은 9개의 값을 표현할 수 있다는 것에는 두가지 데이터형이 동일하나, std_logic은 3상태 버스(tristate bus)나, 와이어로직(wired logic)과 같은 다중구동(multiple driver)모델을 구현할 수 있으므로 std_logic 형을 주로 사용.

6. VHDL의 언어 문법

열거형 데이터 형 (enumeration data type)

- 형식

Type 열거형이름 is (열거값 {,열거값 ...});

- 사용예

- ▶ 오른쪽 괄호 안의 각 element에 binary number가 오른쪽으로 갈수록 증가하는 방향으로 할당됨

type Rainbow is (Red, Orange, Yellow, Green, Blue, Indigo, Violet)

- 정의 결과

| | | | | | | | |
|-------|--------|--------|--------|--------|--------|--------|-------|
| Red | = 000, | Orange | = 001, | Yellow | = 002 | | |
| Green | = 003, | Blue | = 004, | Indigo | = 005, | Violet | = 006 |

- 열거형으로 지정된 열거 값은 숫자가 할당되어있는 것이므로 “Orange > Red” 와 같이 관계연산에 사용할 수 있다.

6. VHDL의 언어 문법

정수 데이터 형(integer data type)

- 형식

Type 정수형이름 is range 정수범위;

- 예)

- ▶ 정수형을 정의 할때는 반드시 그 범위를 정의 하도록 함
- ▶ 합성시 필요 레지스터의 크기는 범위의 최대값에 의존함
 - ✓ 아래 예에서 Thirties의 경우 최대값이 39이며 6bit 레지스터가 필요

Type CountValue is range 0 to 15;
Type Thirties is range 30 to 39;
Type Seventies is range 79 downto 0;

복합 데이터 형(composite data type)

- Composite array type

- ▶ RAM이나 ROM과 같은 구조를 모델링하는데 매우 편리함
- ▶ 배열의 원소는 어떤 데이터 형을 사용해도 됨 (단, 모든 원소가 동일한 데이터 형이어야 함)

6. VHDL의 언어 문법

- ▶ 이러한 배열형은 무제한적 배열(Unconstrained array)과 제한적 배열(Constrained array) 이 있음
- ▶ 형식

무제한적 배열:

type 배열이름 is array (데이터형 range<>) of std_logic;

제한적 배열:

type 배열이름 is array(데이터형 range 데이터범위) of std_logic;

● 예)

- ▶ 무제한적 배열

```
type Bit_vector is array (natural range<>) of bit;
```

● 예)

- ▶ 제한적 배열

```
subtype RAM_WORD is std_logic_vector(7 downto 0);  
type RAM_TABLE is array (0 to 255) of RAM_WORD;
```

6. VHDL의 언어 문법

- 레코드 형(Record type)
 - ▶ 레코드는 서로 같거나 다른 데이터형 객체의 집합
 - ▶ 레코드 형을 정의함으로써 서로 다른 데이터형을 하나의 레코드형으로 사용할수 있게 됨
 - ▶ 레코드를 구성하고 있는 데이터 객체를 필드(field)라고 부름
 - ▶ 형식

레코드참조:

레코드이름

레코드 필드 참조:

레코드이름.레코드필드

- 레코드 정의 예)

```
library IEEE;
use IEEE.STD_logic_1164.all;
use IEEE.STD_logic_arith.all;
package RecordTypes is
  type R1_Type is record
    I : integer range 7 downto 0;
    J : std_logic;
  end record;
```

레코드이름 → R1_Type

필드 → I, J

```
type R2_Type is record
  I : integer range 0 to 7;
  J : unsigned(1 downto 0);
end record;
end package RecordTypes;
```

레코드이름 → R2_Type

필드 → I, J

6. VHDL의 언어 문법

- 레코드 사용 예)

```
library IEEE;
use IEEE.STD_Logic_1164.all;
use IEEE.std_logic_arith.all;
use work.RecordTypes.all;

entity RECORDS is
  port ( A1, A2:  in std_logic;
         B1, B2:  in integer range 0 to 7;
         C:      in R1_type;
         Y:      out R2_type);
end entity RECORDS;

architecture RTL of RECORDS is
  signal M: R1_Type;
```

 R1_Type 레코드의 instance

Instance M의 필드 J

Instance M의 필드 I

```
begin
  process(A1, A2, B1, B2, C)
  begin
    M.I <= B1 + B2;
    M.J <= A1 and A2;
    if( C = M ) then
      Y.I <= M.I - C.I;
      Y.J <= M.J & C.J;
    else
      Y.I <= 0;
      Y.J <="00";
    end if;
  end process;
end architecture RTL;
```

6. VHDL의 언어 문법

실수 형(floating point type)

- 실수형은 합성을 위한 회로에서는 사용해서는 안되고 다만 상위 레벨의 동작 모델링에서만 사용함
- 정의

```
type Real is range -1.0E38 to 1.0E38;
```

부울대수 형(Boolean type)

- 참과 거짓만을 표현하기 위한 형
- 정의

```
type Boolean is (false, true);
```

- 예)

```
entity Boolean_test is
  port( True_false:  in boolean;
        out_result:  out bit);
end entity Boolean_test;
```

```
architecture BEH of Boolean_test is
begin
  out_result <= '1' when (True_false) else '0';
end architecture BEH;
```


6. VHDL의 언어 문법

✦ Bit 및 bit_vector형(Bit & bit_vector type)

- '0' 과 '1'의 2진 상태만을 나타내주며, 산술지원이 안되고 tristate 또는 wired logic지원이 안됨
- Bit_vector는 다수비트로 구성된 데이터형

✦ Std_ulogic 및 std_ulogic_vector 형

- '0', '1', 'U', 'Z' 상태등 여러 로직상태를 표현.
- 분해형이 아니므로(unresolved) 다구동 형태의 하드웨어 또는 wired logic의 지원이 안됨

✦ Std_logic 및 std_logic_vector 형

- 회로 합성을 목적으로 VHDL 설계시 가장 보편적이고 권장되는 데이터형
- '0', '1', 'U', 'Z' 등 다양한 형태의 논리상태를 표현할 수 있으며 분해형(resolved)이므로 다구동 형태나, wired logic등 다양한 하드웨어 구현이 가능
- Std_logic_vector는 다수 비트로 구성된 std_logic 데이터 형임

6. VHDL의 언어 문법

IEEE 1076.3 unsigned 와 signed형

- Unsigned 형은 부호가 없는 수를 나타낼때 쓰임
- Signed형은 전체 비트의 최상위 비트가 sign으로 사용되며, 연산시 2's complement 형태로 연산
- 이 모든것은 std_logic형을 기반으로 정의 되었으나 std_logic형에 직접 할당할 수 없고 data conversion함수를 통하여 데이터형을 동일하게 하여 할당하여야 함.
- IEEE라이브러리의 std_logic_arith에 정의되어 있으며 다음과 같은 함수가 제공

conv_integer:

- Integer, unsigned, signed, std_ulogic을 integer값으로 변환시킴.
- 변환하는 수의 범위는 -2147483647~+2147483647

Conv_unsigned:

Integer, unsigned, signed, std_ulogic을 파라미터에 명시한 크기의 unsigned값으로 변환

Conv_signed:

Integer, unsigned, signed std_ulogic을 파라미터에 명시한 크기의 signed값으로 변환

6. VHDL의 언어 문법

연산자 (Operators)

연산
순위
높음



연산
순위
낮음

| 연 산 자 | 기 능 |
|-----------|---|
| 기타 연산자 | ** (지수), abs (절대치) |
| 산술(곱셈)연산자 | * (곱셈), / (나눗셈), mod (모듈로), rem (나머지) |
| 산술부호연산자 | + (양의부호), - (음의부호) |
| 가산연산자 | + (덧셈), - (뺄셈), & (연쇄) |
| 쉬프트연산자 | sll (논리 왼쪽 쉬프트), srl (논리 오른쪽 쉬프트), sla (산술 왼쪽 쉬프트), sra (산술 오른쪽 쉬프트), rol (논리 왼쪽 회전), ror (논리 오른쪽 회전) |
| 관계연산자 | = (같음), /= (다름), < (적음), <= (적거나 같음), > (큼), >= (크거나 같음) |
| 논리연산자 | not (부정), and (논리 AND), or (논리 OR), nand (논리 NAND), nor (논리 NOR), xor (논리 XOR) xnor (논리 XNOR) |

6. VHDL의 언어 문법

기타연산자

- **: 지수연산
 - ▶ 왼쪽 피연산자는 integer, floating point type가능, 오른쪽 피연산자는 integer type만 가능
 - ▶ 연산결과는 왼쪽 피연산자와 같음
- Abs: 절대값 계산을 위한 연산자.
 - ▶ 모든 numeric 피연산자에 적용가능

산술곱셈연산자

- *, / : 곱셈과 나눗셈
 - ▶ 왼쪽, 오른쪽 피연산자의 데이터 형이 동일해야 함
 - ▶ Integer또는 floating point type이 피연산자로 사용될 수 있음.
- Mod, rem: 모듈로 연산 나머지 연산
 - ▶ 왼쪽, 오른쪽 피연산자의 데이터 형이 모두 integer type이어야 함
- 이러한 연산자는 behavioral level의 코딩에 관련한 것이며 실제 하드웨어 구현시에는, 사용하지 않음.



6. VHDL의 언어 문법

산술부호 연산자

- 부호로서의 +, -
 - ▶ 오른쪽에만 피연산자를 갖는 단일요소(unary)연산자
 - ▶ 결과 같은 피연산자의 자료형과 동일

가산연산자

- +(덧셈), -(뺄셈)
 - ▶ 피연산자는 모두 numeric형이어야 하며, 그 결과 역시 numeric형임
- & : 연쇄연산자
 - ▶ 단일비트 또는 비트열을 서로 연쇄하여 새로운 비트열을 생성하는데 사용하는 연산자
 - 예) “000” & “111” 의 결과: “000111”

쉬프트 연산자

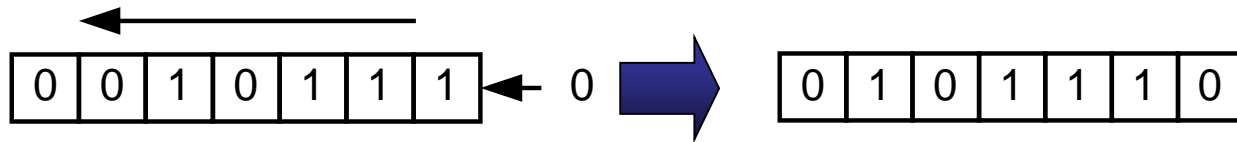
- sll, srl, sla, sra, rol, ror: 논리 왼쪽 쉬프트, 논리오른쪽 쉬프트, 산술 왼쪽 쉬프트 산술 오른쪽 쉬프트, 왼쪽 회전, 오른쪽회전

6. VHDL의 언어 문법

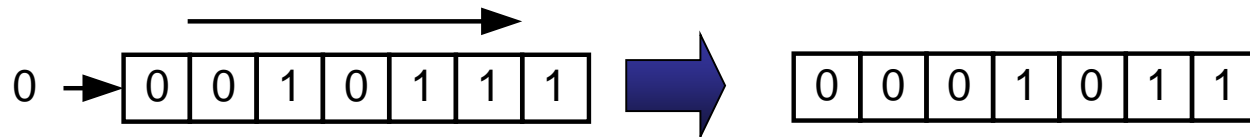
- 예)

- ▶ variable SHIFT_TEST: std_logic_vector(7 downto 0):="0010111"

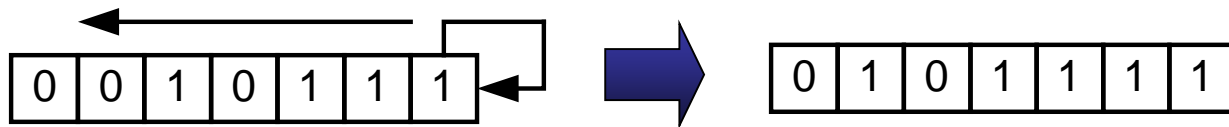
- ▶ SHIFT_TEST sll 1(1비트 논리 왼쪽 쉬프트)



- ▶ SHIFT_TEST srl 1(1비트 논리 오른쪽 쉬프트)



- ▶ SHIFT_TEST sla 1(1비트 산술 왼쪽 쉬프트)

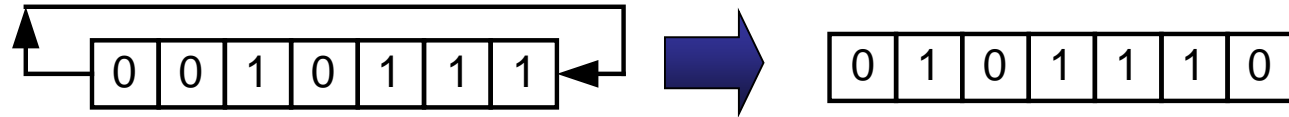


6. VHDL의 언어 문법

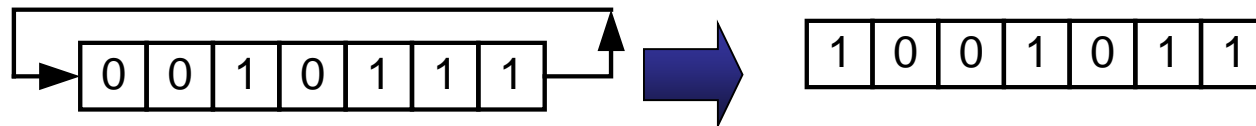
- ▶ SHIFT_TEST sra 1 (1비트 산술 오른쪽 쉬프트)



- ▶ SHIFT_TEST rol 1 (1비트 왼쪽 회전)



- ▶ SHIFT_TEST ror 1 (1비트 오른쪽 회전)



관계 연산자

- 데이터를 비교할때 사용하며 대부분의 자료형을 지원함
- 관계연산의 결과는 항상 부울형

6. VHDL의 언어 문법

- 비트열인 vector 연산시 왼쪽 과 오른쪽의 피연산자 길이가 다를 수 있지만 짧은쪽의 오른쪽에 '0'을 추가 하여 같은 길이로 만들어 비교 함=> ∴ 원하는 결과가 나오지 않을 수 있음
- 예) “1001” 과 “110”의 비교
“1001”과 “1100”과의 비교가 수행되므로 “110”이 “1001”보다 큰것으로 틀린 결과를 생성할 수 있음

논리 연산자

- 조합논리를 만드는데 사용됨
- 피연산자는 부울형과 bit 및 bit_vector형 지원 그 연산결과는 피연산자의 데이터 형과 동일
- 지원되는 논리연산자는 not, and, or, nand, nor, xor, xnor

| A | not A |
|---|-------|
| 0 | 1 |
| 1 | 0 |



6. VHDL의 언어 문법

| A | B | A and B |
|---|---|---------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| A | B | A nand B |
|---|---|----------|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| A | B | A xor B |
|---|---|---------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| A | B | A or B |
|---|---|--------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| A | B | A nor B |
|---|---|---------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

| A | B | A xnor B |
|---|---|----------|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

6. VHDL의 언어 문법

속성 (attributes)

- 객체에 대한 특성을 참조할 때 사용
- 형식

객체'속성

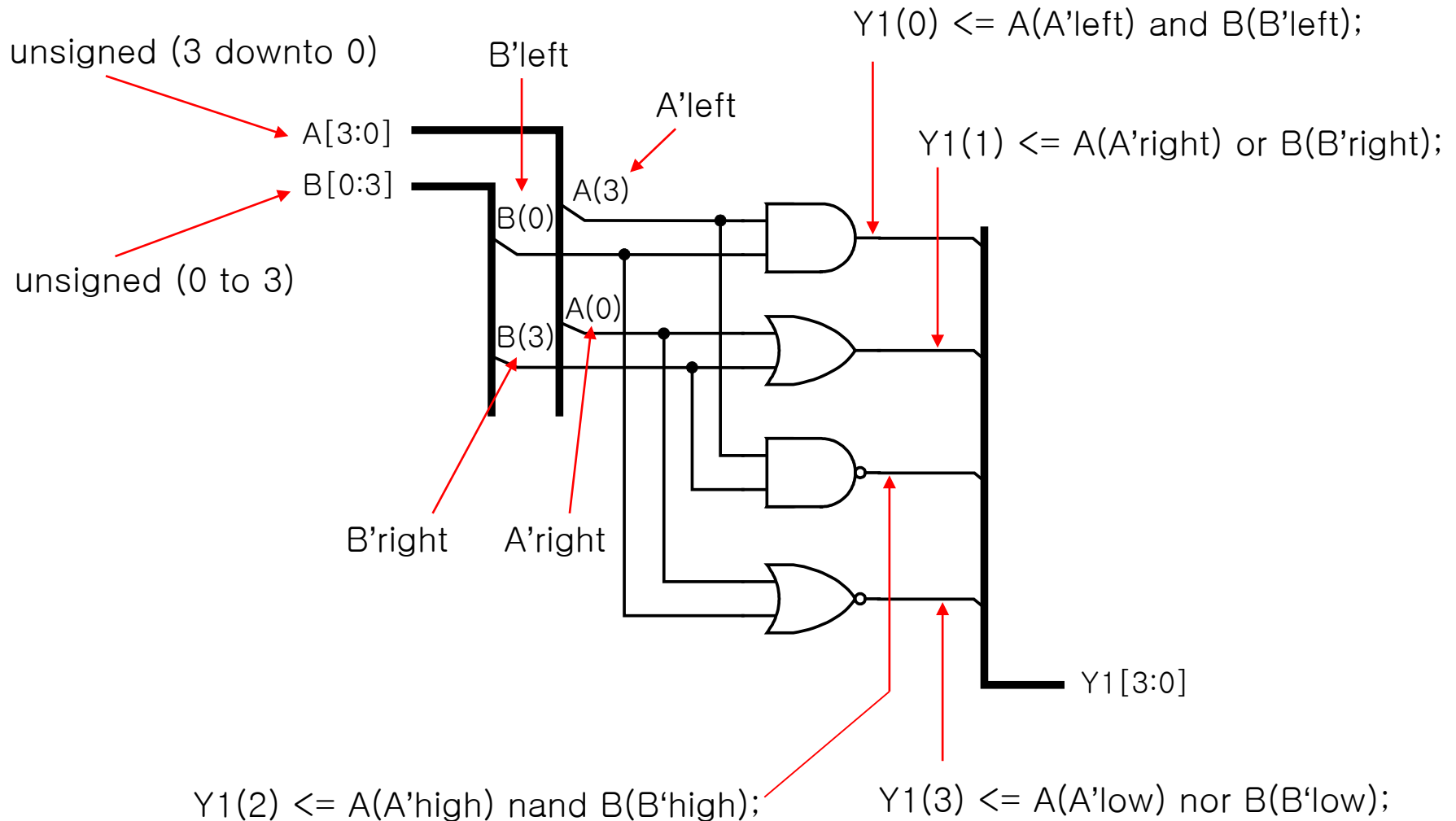
- 예)

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Std_logic_arith.all;
entity ATTRIBUTES_TYPES is
    port( A:    in unsigned (3 downto 0);
          B:    in unsigned (0 to 3);
          Y1:   out unsigned( 3 downto 0));
end entity ATTRIBUTES_TYPES;
architecture LOGIC of ATTRIBUTES_TYPES is
```

```
begin
    process(A, B)
    begin
        Y1(0) <= A(A'left) and B(B'left);
        Y1(1) <= A(A'right) or B(B'right);
        Y1(2) <= A(A'high) nand B(B'high);
        Y1(3) <= A(A'low) nor B(B'low);
    end process;
end architecture LOGIC;
```

6. VHDL의 언어 문법

- 합성 회로도



6. VHDL의 언어 문법

- 예)

- ▶ 합성과 모의 실험에 모두 많이 쓰이고 유용한 속성인 'event'의 예
- ▶ 'event'는 현재 시뮬레이션 시점에서 객체의 이벤트가 발생하였는지의 여부를 부울형으로 돌려줌
- ▶ 신호 clock의 신호레벨 변화를 감지 하고, 변화의 결과가 1인 경우를 다음과 같이 표기 하므로써 clock신호의 rising edge를 감지함

```
if ( clock'event and clock='1' ) then  
    S:= A and B;  
end if;
```



7. 조합회로의 설계

✦ 논리/산술연산모델링

- 모델링시 순차적 연산부와 동시연산부를 각각각 정확히 기술하여야 하는데 유의함
- 다음의 예를 통하여 순차문과 병렬처리 부의 형성을 확인함
- 예)

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Std_logic_arith.all;
entity DATA_FLOW is
    port (A1: in unsigned(7 downto 0);
          A2, B2, C2, D2, E2: in unsigned(1 downto 0);
          Y1: out std_logic;
          Y2: out unsigned(3 downto 0) );
end entity DATA_FLOW;
architecture RTL of DATA_FLOW is
    signal S1, S2: std_logic;
```

7. 조합회로의 설계

begin
I { process(A1(3 downto 0))
 variable V: unsigned(1 downto 0);
 begin
 V:=(A1(0) nor A1(1)) & (A1(2) nor A1(3));
 S1 <= V(0) nand V(1);
 end process;

II { process(A1(7 downto 4))
 variable V:unsigned(1 downto 0);
 begin
 V:=(A1(4) nor A1(5)) & (A1(6) nor A1(7));
 S2 <= V(0) nand V(1);
 end process;

III { Y <= S1 nor S2;

IV { process(A2, B2, C2, D2, E2)
 begin
 Y2 <= A2 + (B2 - C2) + (D2 * E2);
 end process;
end architecture RTL;

sensitivity list

- 프로세스 I, II, IV 각각은 병렬 처리문
- 프로세스 내부의 문은 순차처리문
- III 할당문은 process 외부에 존재하므로 병렬처리되는 wired logic
- 프로세스 IV의 D2*E2문은 실제 하드웨어 구현을 위해서는 세부적인 하드웨어 코드를 사용하여 구현



7. 조합회로의 설계

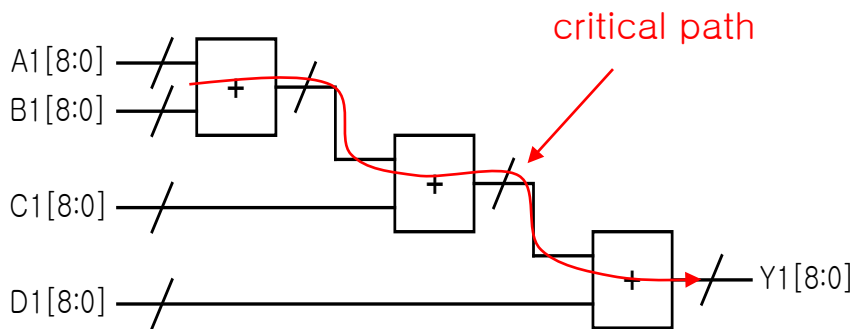
논리구조 제어

- 괄호를 사용하여 로직의 논리구조를 결정할 수 있음
- 코딩시에 괄호를 사용하면 최적화가 간편하게 되고, 최적화시 CPU의 사용량이 감소함.
- 그러나 잘못된 괄호의 사용은 최적화에 있어서 문제를 야기시킬 수 있음.
- 예)

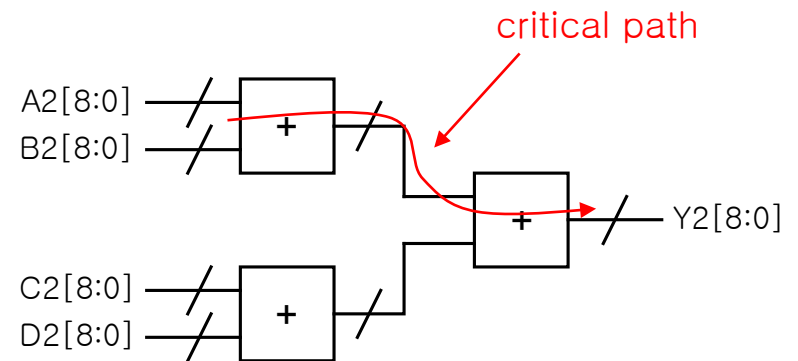
```
library IEEE;  
use IEEE.STD_logic_1164.all;  
use IEEE.STD_logic_arith.all;  
entity LOGIC_STRUCT is  
    port(A1, B1, C1, D1, A2, B2, C2, D2:in unsigned(8 downto 0);  
          Y1, Y2: out unsigned(10 downto 0));  
end entity LOGIC_STRUCT;
```

7. 조합회로의 설계

```
architecture LOGIC of LOGIC_STRUCT is
begin
  process(A1, B1, C1, D1, A2, B2, C2, D2)
  begin
    -- No parentheses used
    Y1 <= A1 + B1 + C1 + D1;
    -- Structured by parenthesis
    Y2 <= (A2 + B2) + (C2 + D2);
  end process;
end architecture LOGIC;
```



Y1 <= A1 + B1 + C1 + D1의 합성된 구조

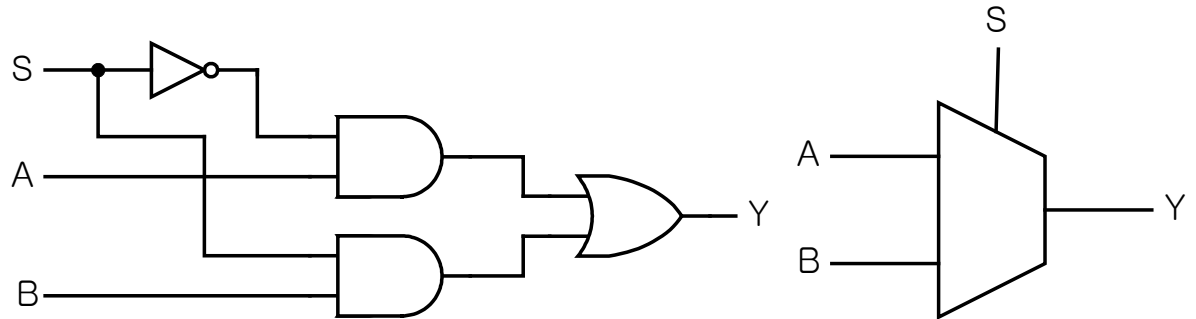


Y2 <= (A2 + B2) + (C2 + D2)의 합성된 구조

7. 조합회로의 설계

다중화기 (Multiplexer)

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | X | 0 |
| 0 | 1 | X | 1 |
| 1 | X | 0 | 0 |
| 1 | X | 1 | 1 |



2X1 MUX의 설계예

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
entity MUX_2_1 is
```

```
    port(Sel1, A1, B1, Sel2, A2, B2, Sel3, A3, B3:in std_logic;
```

```
          Y1, Y2, Y3: out std_logic);
```

```
end entity MUX_2_1;
```

```
architecture MUX_MODEL of MUX_2_1 is
```

7. 조합회로의 설계

begin

I { -- 2X1 mux model 1
Y1 <= A1 when Sel1 = '1' else B1;

조건부 할당문 사용

II { process(Sel2, A2, B2, Sel3, A3, B3)
begin
-- 2X1 mux model 2
Y2 <= B2;
if (Sel2 = '1') then
Y2 <= A2;
end if;

default를 if문 앞에 기술

III { -- 2X1 mux model 3
if (Sel3 = '1') then
Y3 <= A3;
else
Y3 <= B3;
end if;
end process;

•가장 보편적으로 사용되는 if문
•조건이 만족하지 않는 경우를
else이후에 기술 (default)

end architecture MUX_MODEL;

7. 조합회로의 설계

- 조건부 할당문

- ▶ 병행처리문이므로 process문 밖에 있어야 함

신호이름 <= 참인경우의 값 when 조건 else 거짓인경우의 값

- if 문의 구성

if 문의 첫번째 형식:

```
if 조건 then
    {순차문}
end if;
```

if 문의 두번째 형식:

```
if 조건 then
    {순차문1}
else
    {순차문2}
end if;
```

if 문의 세번째 형식:

```
if 조건1 then
    {순차문1}
elsif 조건2 then
    {순차문2}
elsif .....

else
    {순차문n}
end if;
```

7. 조합회로의 설계

- 4X1 MUX의 설계예

- ▶ 2×1 다중화기 보다 선택 조건이 많으므로 if문 이외 다른 조건문을 이용하여 설계
- ▶ 설계방법
 - ✓ 하나의 if문을 사용하고 다수의 elsif 문을 사용하는 방법
 - ✓ 조건부 할당문을 사용하는 방법
 - ✓ 중첩된 if문을 사용하는 방법
 - ✓ case문을 사용하는 방법
 - ✓ 선택적 할당문을 사용하는 방법

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Std_logic_arith.all;
entity MUX4_1 is
    port (Sel:  in unsigned(1 downto 0);
          A, B, C, D: in std_logic;
          Y:      out std_logic);
end entity MUX4_1;
```

```
-- 하나의 if문을 사용하고 다수의 elsif 문을 사용
architecture BEH1 of MUX4_1 is
```

```
begin
    process(Sel, A, B, C, D)
    begin
        if (Sel ="00") then
            Y <= A;
        elsif (Sel = "01") then
            Y <= B;
        elsif (Sel = "10") then
            Y <= C;
        elsif (Sel = "11") then
            Y <= D;
        end if;
    end process;
end architecture BEH1;
```

7. 조합회로의 설계

-- 조건부 할당문을 사용하는 방법

architecture BEH2 of MUX4_1 is

begin

-- 조건부 할당문은 병행처리문이므로

-- process문 밖에 있다.

Y <= A when Sel = "00" else

B when Sel = "01" else

C when Sel = "10" else

D; -- when Sel = "11"

end architecture BEH2;

-- 중첩된 if문을 사용하는 방법

architecture BEH3 of MUX4_1 is

begin

process(Sel, A, B, C, D)

begin

if (Sel(1) = '0') then

if(Sel(0) = '0') then

Y <= A;

else

Y <= B;

end if;

else

if(Sel(0) = '0') then

Y <= C;

else

Y <= D;

end if;

end if;

end process;

end architecture BEH3;

-- case문을 사용하는 방법

architecture BEH4 of MUX4_1 is

begin

process(Sel, A, B, C, D)

begin

case Sel is

when "00" => Y <= A;

when "01" => Y <= B;

when "10" => Y <= C;

when "11" => Y <= D;

when others => Y <= A;

end case;

end process;

end architecture BEH4;

7. 조합회로의 설계

```
-- 선택적 할당문을 사용하는 방법
architecture BEH5 of MUX4_1 is
begin
    -- 선택적 할당문은 병행처리문이므로
    -- process문 밖에 있다.
    with Sel select
        Y <= A when "00",
            B when "01",
            C when "10",
            D when "11",
            A when others;
end architecture BEH5;

-- 엔티티-아키텍처 조합구성
configuration conf of MUX4_1 is
    for BEH4
    -- 엔티티 MUX4_1 와 BEH4를 연결시킴
    end for;
end configuration conf;
```

• case 문 형식

```
case 수식 is
    when 값1 => 순차문1;
    when 값2 => 순차문2;
    .
    .
    .
    when 값n => 순차문n;
    [when others => 순차문;]
end case;
```

- case문은 수식의 값에 따라서 문장을 선택
- when 다음에오는 값의 데이터 형은 모두 같아야 함
- (정수형, 열거형 bit string이 모두 가능)
- when others문장은 when 값에 모든 경우가 다 포함되지 않은 경우에는 반드시 포함시켜야 함



7. 조합회로의 설계

- 선택적 할당문

- ▶ 병행처리문이므로 process문 밖에 있음(앞의 예에서 BEH5)

```
with 수식 select
```

```
    신호이름 <= 할당될신호1 when 값1,
```

```
        <= 할당될신호2 when 값2,
```

```
        .
```

```
        .
```

```
    <= 할당될신호n when 값n,
```

```
    <= 디폴트로할당될신호 when others;
```

인코더 (Encoder)


- | Inputs | | | | | | | | Outputs | | |
|--------|----|----|----|----|----|----|----|---------|----|----|
| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | Y2 | Y1 | Y0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

7. 조합회로의 설계

- 예 1) if ... elsif 문 사용예

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity ENCODE_8_3_IF_ELSE is
    port(A: in unsigned(7 downto 0);
         B: out unsigned(2 downto 0));
end entity ENCODE_8_3_IF_ELSE;
architecture BEH of ENCODE_8_3_IF_ELSE is
begin
    process(A)
    begin
        if ( A = "00000001" ) then
            Y <= "000";
        elsif (A = "00000010") then
            Y <= "001";
        elsif (A = "00000100") then
            Y <= "010";
```

```
        elsif (A = "00001000") then
            Y <= "011";
        elsif (A = "00010000") then
            Y <= "100";
        elsif (A = "00100000") then
            Y <= "101";
        elsif (A = "01000000") then
            Y <= "110";
        elsif (A = "10000000") then
            Y <= "111";
        else
            Y <= "XXX";
        end if;
    end process;
end architecture BEH;
```



7. 조합회로의 설계

- 예 2) 조건부 할당문을 사용

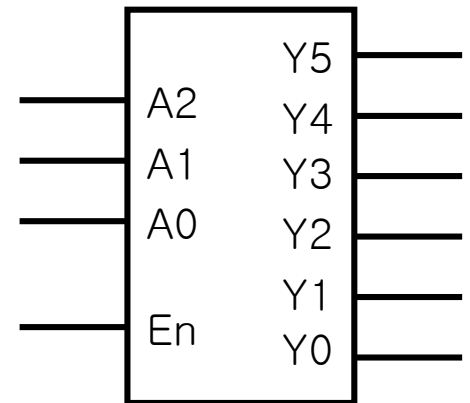
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity ENCODE_8_3_CSA is
    port(A:  in unsigned(7 downto 0);
          Y:  in unsigned(2 downto 0));
end entity ENCODE_8_3_CSA;
architecture LOGIC of ENCODE_8_3_CAS is
begin
    Y <= "000" when "00000001" else
        "001" when "00000010" else
        "010" when "00000100" else
        "011" when "00001000" else
        "100" when "00010000" else
        "101" when "00100000" else
        "110" when "01000000" else
        "111" when "10000000" else
        "XXX";
end architecture LOGIC;
```

default

7. 조합회로의 설계

- 인에이블 신호가 있는 3-6 2진 디코더
 - 디코더 진리표

| Inputs | | | Outputs | | | | | | |
|--------|----|----|---------|----|----|----|----|----|--|
| A2 | A1 | A0 | Y5 | Y4 | Y3 | Y2 | Y1 | Y0 | |
| X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |



7. 조합회로의 설계

- if 문을 이용한 En 구현예

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity DECODER is
    port( En: in std_logic;
          A: in integer range 0 to 7;
          Y: out unsigned(5 downto 0));
end entity DECODER;
architecture BEH of DECODER is
begin
    process (En, A)
    begin
        if (En = '0') then
            Y <= "000000";
        else
```

En=0 이면 출력은
"000000"

En≠0 인
경우의
출력

```
        case A is
            when 0 => Y <= "000001";
            when 1 => Y <= "000010";
            when 2 => Y <= "000100";
            when 3 => Y <= "001000";
            when 4 => Y <= "010000";
            when 5 => Y <= "100000";
            when others => Y <= "000000";
        end case;
    end if;
end process;
end architecture BEH;
```

특정조건 이외의 경우 default 할당

7. 조합회로의 설계

- 연쇄연산자 &와 case문을 이용한 예

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity DECODER is
    port( En: in std_logic;
          A: in integer range 0 to 7;
          Y: out unsigned(5 downto 0));
end entity DECODER;
architecture BEH of DECODER is
begin
    process (En, A)
        variable En_temp: std_logic_vector(3 downto 0);
    begin
        En_temp := En & conv_std_logic_vector(A,3);
    end process;
end architecture BEH;
```

En 과 입력의 concatenation

```
case En_temp is
    when "1000" => Y <= "000001";
    when "1001" => Y <= "000010";
    when "1010" => Y <= "000100";
    when "1011" => Y <= "001000";
    when "1100" => Y <= "010000";
    when "1101" => Y <= "100000";
    when others => Y <= "000000";
end case;
```

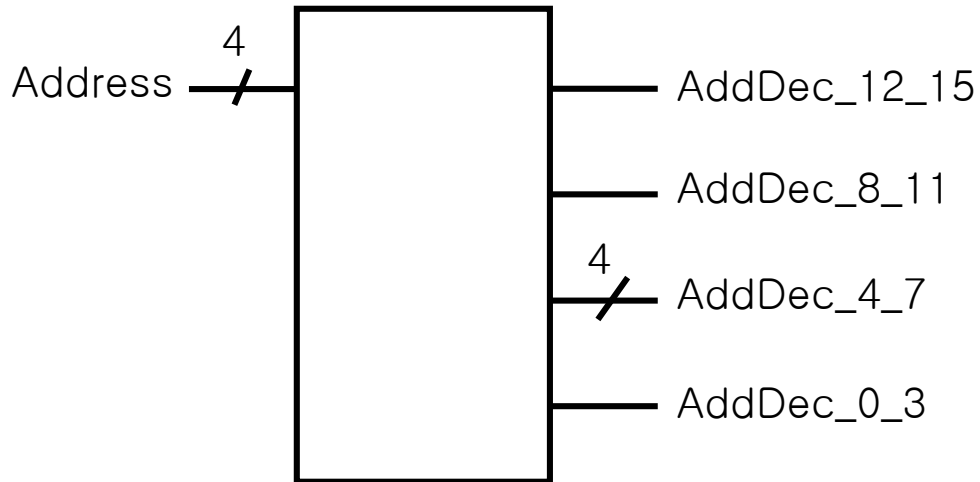
```
end process;
end architecture BEH;
```

Concatenation을 위한 변수(variable) 정의

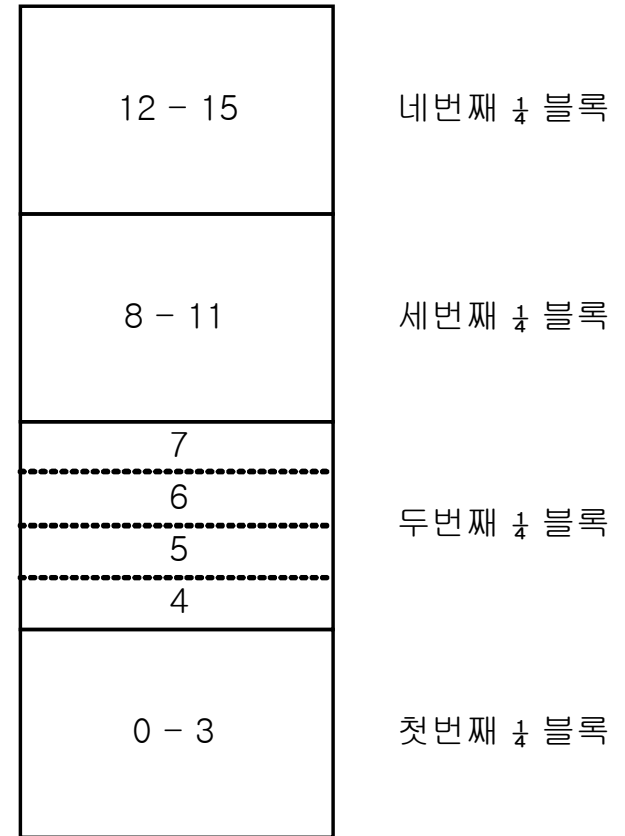
포트 A는 integer형이므로 std_logic_vector로 정의된 변수 En_temp에 연쇄하여 적용하기 위해서 integer형을 std_logic_vector형으로 변환

7. 조합회로의 설계

4bit address decoder



```
Address:           in integer range 0 to 15;  
AddDec_0_3:        out std_logic;  
AddDec_8_11:       out std_logic;  
AddDec_12_15:      out std_logic;  
AddDec_4_7:        out unsigned(3 downto 0));
```



7. 조합회로의 설계

구현 코드

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity ADD_DEC is
    port (Address:    in integer range 0 to 15;
          AddDec_0_3, AddDec_8_11,
          AddDec_12_15: out std_logic;
          AddDec_4_7: out unsigned(3 downto 0));
end entity ADD_DEC;
architecture BEH of ADD_DEC is
begin
    process(Address)
    begin
        -- 첫번째 1/4 블록
        if(Address >= 0 and Address <=3 ) then
            AddDec_0_3 <= '1';
        else
            AddDec_0_3 <= '0';
        end if;
```

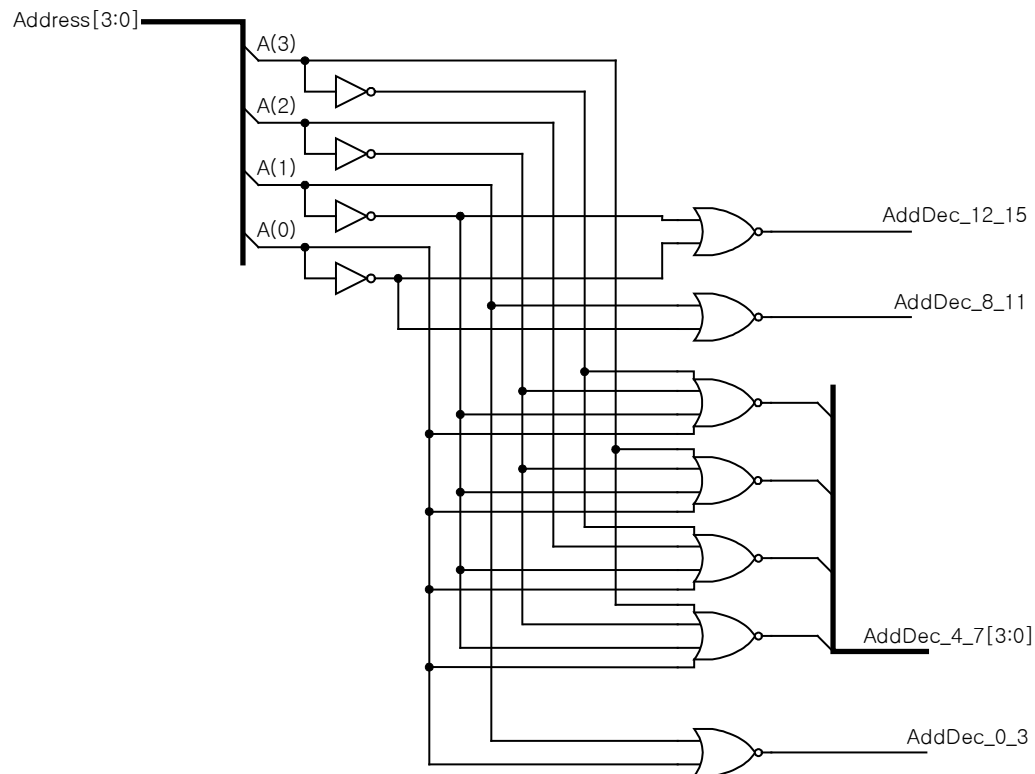
```
-- 두번째 1/4 블록
        For N in AddDec_4_7'range loop
            if(Address = N + 4 ) then
                AddDec_4_7(N) <= '1';
            else
                AddDec_4_7(N) <= '0';
            end if;
        end loop;
-- 세번째 1/4 블록
        if(Address >= 8 and Address <=11 ) then
            AddDec_8_11 <= '1';
        else
            AddDec_8_11 <= '0';
        end if;
-- 네번째 1/4 블록
        if(Address >= 12 and Address <=15 ) then
            AddDec_12_15 <= '1';
        else
            AddDec_12_15 <= '0';
        end if;
    end process;
end architecture BEH;
```

7. 조합회로의 설계

- for loop 문의 형식

```
[라벨:] for 루프변수 in 변수범위 loop  
    {순차문}  
end loop [라벨];
```

- 합성된 address decoder



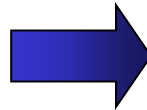
7. 조합회로의 설계

- while loop문
 - ▶ for loop와 유사한 반복제어문
 - ▶ 형식

```
[라벨:] while 조건 loop
    {순차문}
end loop [라벨];
```

for문 사용의 경우

```
-- 두번째 1/4 블록
For N in AddDec_4_7'range loop
    if(Address = N + 4 ) then
        AddDec_4_7(N) <= '1';
    else
        AddDec_4_7(N) <= '0';
    end if;
end loop;
```



while문 사용의 경우

```
-- 두번째 1/4 블록
While N <= AddDec_4_7'high loop
    if(Address = N + 4 ) then
        AddDec_4_7(N) <= '1';
    else
        AddDec_4_7(N) <= '0';
    end if;
    N := N+1;
end loop;
```

7. 조합회로의 설계

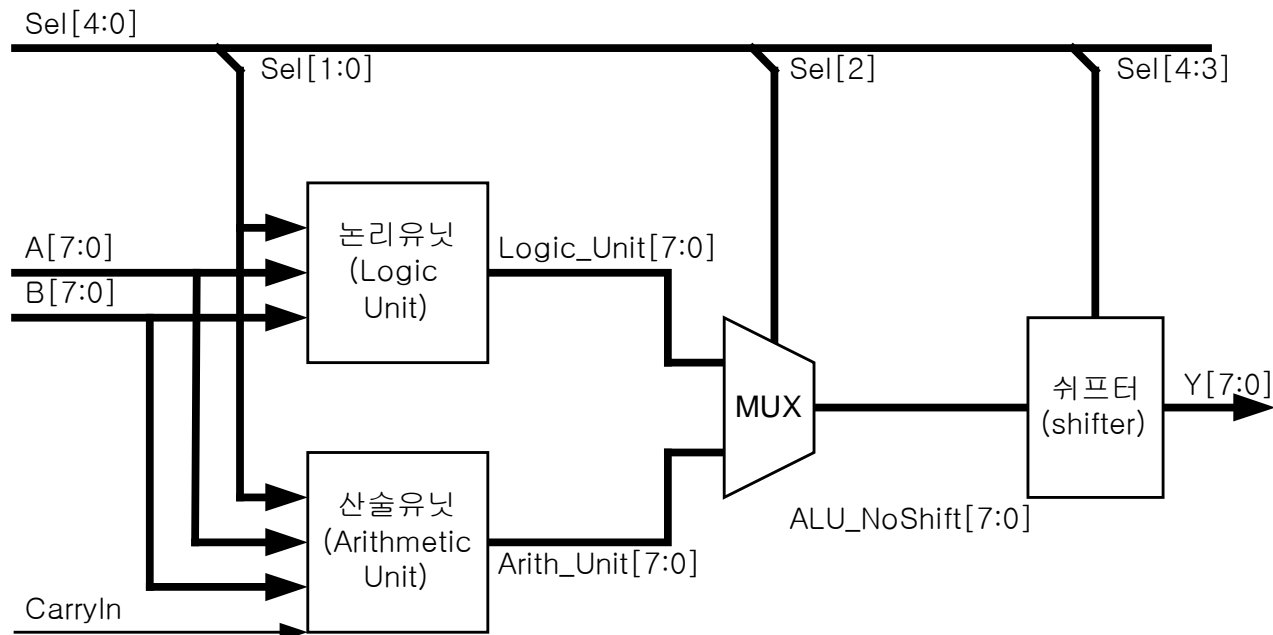
ALU (Arithmetic Logic Unit)

- CPU의 핵심 요소이며, 산술/논리연산 담당
- 8비트 입력 버스에 대한 산술적 논리적 연산수행
- 아래와 같은 ALU 명령어세트

| S4 | S3 | S2 | S1 | S0 | Cin | 동작식 | 기능설명 |
|----|----|----|----|----|-----|---------------------------------|------------------|
| 0 | 0 | 0 | 0 | 0 | 0 | $Y \leq A$ | A의 전달 |
| 0 | 0 | 0 | 0 | 0 | 1 | $Y \leq A + 1$ | A의 증가 |
| 0 | 0 | 0 | 0 | 1 | 0 | $Y \leq A + B$ | 가산 |
| 0 | 0 | 0 | 0 | 1 | 1 | $Y \leq A + B + 1$ | 캐리포함한 가산 |
| 0 | 0 | 0 | 1 | 0 | 0 | $Y \leq A + B_{\text{bar}}$ | A와 B의 1의보수와와의 가산 |
| 0 | 0 | 0 | 1 | 0 | 1 | $Y \leq A + B_{\text{bar}} + 1$ | 감산 |
| 0 | 0 | 0 | 1 | 1 | 0 | $Y \leq A - 1$ | A의 감소 |
| 0 | 0 | 0 | 1 | 1 | 1 | $Y \leq A$ | A의 전달 |
| 0 | 0 | 1 | 0 | 0 | 0 | $Y \leq A \text{ and } B$ | AND |
| 0 | 0 | 1 | 0 | 1 | 0 | $Y \leq A \text{ or } B$ | OR |
| 0 | 0 | 1 | 1 | 0 | 0 | $Y \leq A \text{ xor } B$ | XOR |
| 0 | 0 | 1 | 1 | 1 | 0 | $Y \leq A_{\text{bar}}$ | A의 1의보수 |
| 0 | 0 | 0 | 0 | 0 | 0 | $Y \leq A$ | A전달 |
| 0 | 1 | 0 | 0 | 0 | 0 | $Y \leq \text{shl } A$ | A의 왼쪽 쉬프트 |
| 1 | 0 | 0 | 0 | 0 | 0 | $Y \leq \text{shr } A$ | A의 오른쪽 쉬프트 |
| 1 | 1 | 0 | 0 | 0 | 0 | $Y \leq 0$ | 0를 전달 |

7. 조합회로의 설계

- 상기 모든 기능을 하나의 case문으로 모델링 가능
 - ▶ 이러한 경우에는 합성의 효율의 감소 초래
 - ▶ 합성된 결과가 원하는 만큼 최적화 되지 않을 수 도 있음
- 설계시 기능별로 잘 구분하여 코딩하여야 최적의 합성결과를 얻을 수 있음.
- 기능분리 된 ALU 구조도



7. 조합회로의 설계

- 구현 코드

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_std.all; -- IEEE 1076.3 synthesis package
```

```
entity ALU is
```

```
port (Sel:    in unsigned(4 downto 0);
```

```
      CarryIn: in std_logic;
```

```
      A, B:    in unsigned(7 downto 0);
```

```
      Y:       out unsigned(7 downto 0));
```

```
end entity ALU;
```

```
architecture MODEL_ALU of ALU is
```

```
begin
```

```
  ALU_AND_SHIFT:
```

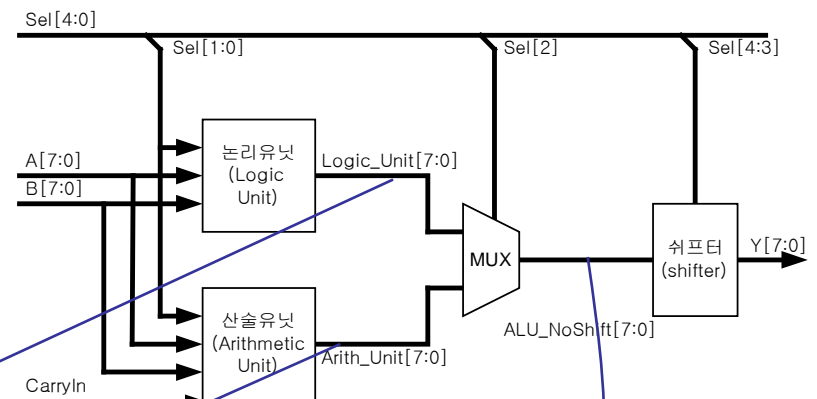
```
  process (Sel, A, B, CarryIn)
```

```
    variable Sel0_1_CarryIn: unsigned(2 downto 0);
```

```
    variable LogicUnit, ArithUnit,
```

```
      ALU_NoShift: unsigned(7 downto 0);
```

```
  begin
```



7. 조합회로의 설계

-- 논리유닛

```
LOGIC_UNIT: case Sel(1 downto 0) is
  when "00" => LogicUnit := A and B;
  when "01" => LogicUnit := A or B;
  when "10" => LogicUnit := A xor B;
  when "11" => LogicUnit := not A;
  when others => LogicUnit := (others => 'X');
end case LOGIC_UNIT;
```

-- 산술연산유닛

```
Sel0_1_CarryIn := Sel(1 downto 0) & carryIn;
ARITH_UNIT: case Sel0_1_CarryIn is
  when "000" => ArithUnit := A;
  when "001" => ArithUnit := A + 1;
  when "010" => ArithUnit := A + B;
  when "011" => ArithUnit := A + B + 1;
when "100" => ArithUnit := A + not B;
  when "101" => ArithUnit := A - B;
  when "110" => ArithUnit := A - 1;
  when "111" => ArithUnit := A;
  when others => ArithUnit := (others => 'X');
end case ARITH_UNIT;
```

-- 논리유닛과 산술연산유닛의 결과선택

```
LA_MUX: if (Sel(2) = '1') then
```

```
  ALU_NoShift := LogicUnit;
```

```
else
```

```
  ALU_NoShift := ArithUnit;
```

```
end if LA_MUX;
```

-- 쉬프트동작

```
SHIFT: case Sel(4 downto 3) is
```

```
  when "00" => Y <= ALU_NoShift;
```

```
  when "01" => Y <= ALU_NoShift sll 1;
```

```
  when "10" => Y <= ALU_NoShift srl 1;
```

```
  when "11" => Y <= (others => '0');
```

```
  when others => Y <= (others => 'X');
```

```
end case SHIFT;
```

```
end process ALU_AND_SHIFT;
```

```
end architecture MODEL_ALU;
```