

VHDL의 이해

최기영

서울대학교
전자공학과

목차

1. 서론
2. 기본 구성
3. Lexical Element
4. Type 과 Object
5. Structural Description
6. Data Flow Description
7. Behavioral Description
8. Subprogram
9. 기타
10. VHDL-87과 VHDL-93의 다른 점

1. 서론

VHDL의 출현 배경

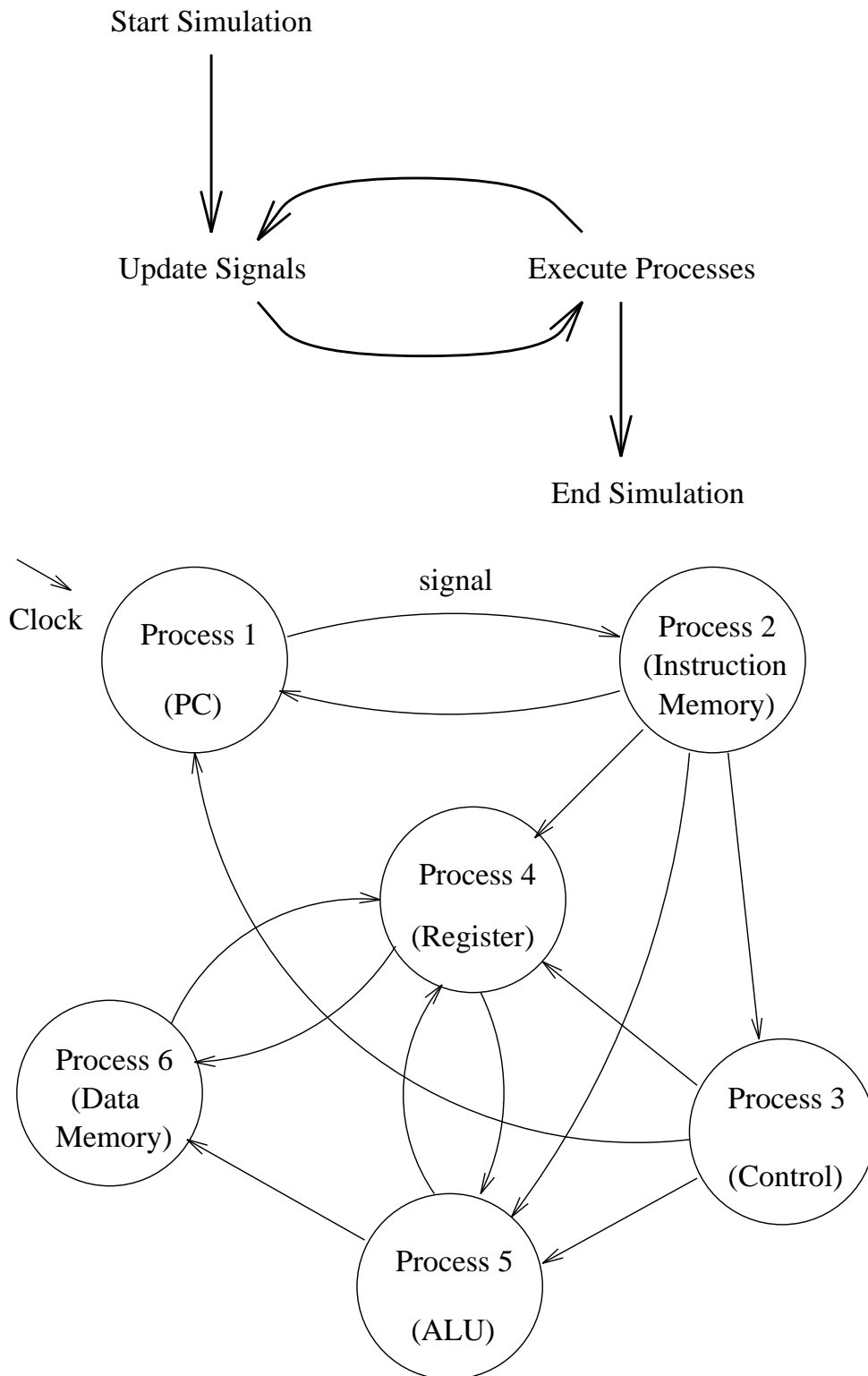
- IC 또는 시스템의 복잡도와 함께 그 설계의 난이도 급증.
- Top-down 설계. Algorithmic level, register transfer level 등 high-level에서 설계 시작.
트랜지스터 10^5 → cell 10^4 → block 10^2 → module 10^1 → VLSI → 시스템.
- 프로그램을 만들 듯 HDL(Hardware Description Language)을 이용하여 하드웨어를 표현.
- CAD tool을 이용하여 자동적으로 (필요에 따라 수동적인 방법도 사용하여) low-level의 설계를 생성.
- 설계 기간 단축, 설계 관리, 교환, 재사용에 효과적.
- HDL은 1970년대부터 많은 관심을 모음. CDL, DDL, ISP, PMS, AHPL 등.
- 표준 HDL의 필요성 인식.

- VHDL의 공식적 논의는 1981년 VHSIC program의 일환으로 시작.
- 제안 요청은 미국 공군에서 작성 1983년 초에 확정.
- Intermetrics, IBM, Texas Instruments 와 VHDL 및 지원 소프트웨어의 개발에 대한 계약.
- 1985년 VHDL version 7.2.
- 1987년 12월 IEEE Computer Society 산하 VHDL Analysis and Standardization Group이 IEEE Standard 1076-1987로 확정.
- 미국 국방성은 military standard 454로 지정.
- 많은 CAD 회사들이 VHDL 지원 tool을 다투어 개발.
- 1993년 새로운 version을 IEEE Standard 1076-1993으로 확정.

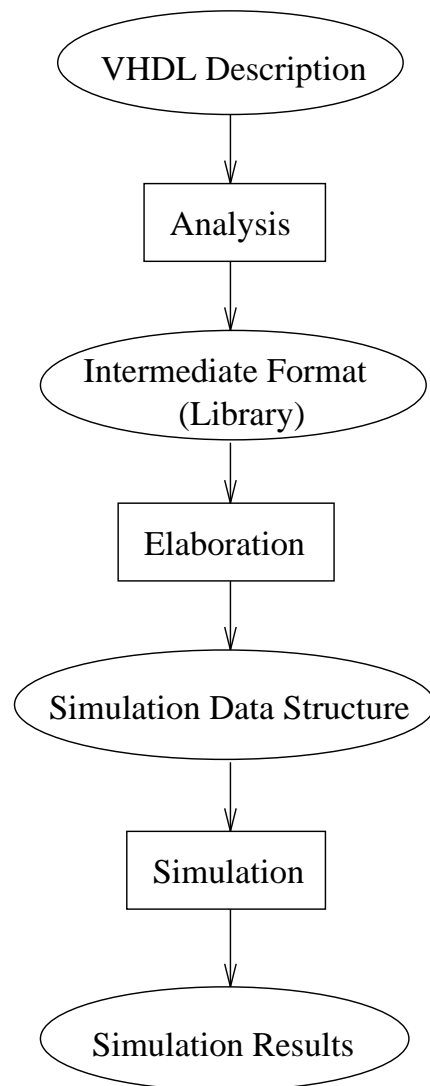
VHDL의 특징

- 표현 능력
 - 게이트 수준에서 시스템 수준까지.
 - *Structural, behavioral, data flow, mixed description.*
 - *Transport, inertial delay model.*
 - User-defined resolution function.
- 설계 관리에 대한 지원
 - Package.
 - Multiple architecture bodies.
 - Generic, generator 등을 이용한 parameterized design.
- 확장성
 - User-defined attribute.
 - User-defined type.
 - Overloading.
- CAD tool과의 연결 문제
 - 시뮬레이션 지향적 언어.
 - Synthesis, timing verification, critical path analysis, testing 등에 대한 지원 부족.

● 시뮬레이션 모델



● 시뮬레이션 과정



2. 기본 구성

- 최소한의 단위는 design entity.

design entity = entity declaration(interface)
+ architecture body(behavior)

- -- Entity declaration (comment)

```
entity mux is
```

```
    port (x, y, sel: in BIT;  
          mout: out BIT);
```

```
end mux;
```

```
-- Architecture body (comment)
```

```
-- Behavioral description (comment)
```

```
architecture behav of mux is
```

```
begin
```

```
    process (sel, x, y)
```

```
    begin
```

```
        if sel = '1' then
```

```
            mout <= x;
```

```
        else
```

```
            mout <= y;
```

```
        end if;
```

```
    end process;
```

```
end behav;
```


Entity Declaration

- Interface 를 정의
 - Port
 - Generic(Parameterized design 에 사용)
- Syntax

```
entity_declaration ::=  
    entity entity_name is  
        [generic (generic_list);]  
        [port (port_list);]  
        {entity_declarative_item}  
    [begin  
        entity_statement_part]  
    end [entity_name];
```

[...] : 0 번 또는 1 번 반복

{...}: 0 번 이상 반복

Architecture Body

- Behavior 를 정의
- 다음과 같은 description 이 가능
 - Structural description
 - Data flow description
 - Behavioral description
 - Mixed description
- architecture_body ::=
architecture architecture_name **of** entity_name **is**
 {architecture_declarative_item}
begin
 {concurrent_statement}
end [architecture_name];

Multiplexer의 structural description

-- Structural description

```
library prim;
```

```
use prim.gates.all;
```

```
architecture struct of mux is
```

```
    signal selb, t1, t2: BIT;
```

```
    -- configuration specification
```

```
begin
```

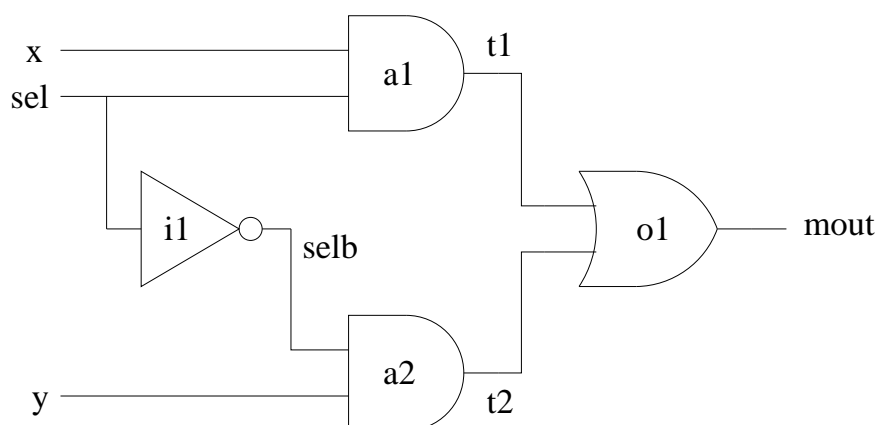
```
    i1: inv port map (selb, sel);
```

```
    a1: and2 port map (t1, x, sel);
```

```
    a2: and2 port map (t2, y, selb);
```

```
    o1: or2 port map (mout, t1, t2);
```

```
end struct;
```



Multiplexer의 data flow description

-- Data flow description

architecture dflow **of** mux **is**

signal t1, t2: BIT;

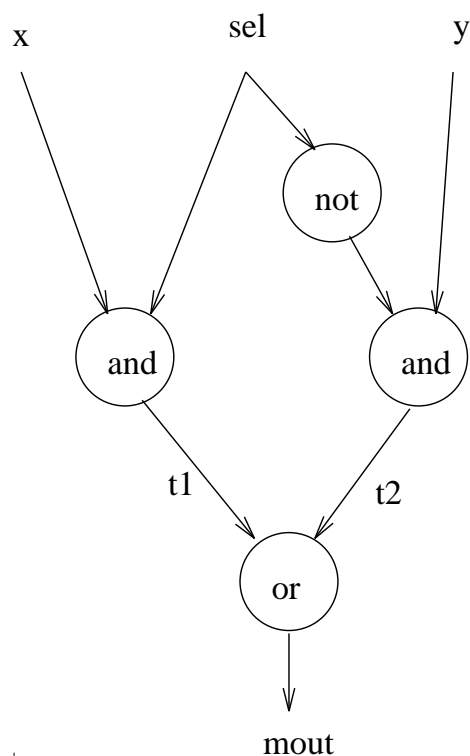
begin

t1 <= x **and** sel;

t2 <= y **and** (**not** sel);

mout <= t1 **or** t2;

end dflow;



Test Bench

```
-- Test bench for 2-way multiplexer
-- Entity declaration
entity test_mux is
end test_mux;

-- Architecture body
-- Needs TEXTIO package
use STD.TEXTIO.all;
architecture mixed of test_mux is
    -- signal declarations
    signal x1, x2, y, sel: BIT;
    -- mux component declaration
    component mux
        port (x, y, sel: in BIT;
             mout: out BIT);
    end component;
    -- configuration specification
    for all: mux use entity work.mux(struct);
```

begin

```
-- component instantiation statement
-- for mux component instantiation
m1: mux port map (x1, x2, sel, y);
-- concurrent signal assignment statements
-- for test vector
sel <= '0', '1' after 50 ns, '0' after 100 ns,
    '1' after 125 ns;
x1 <= '1', '0' after 25 ns, '1' after 50 ns,
    '0' after 75 ns, '1' after 100 ns;
x2 <= '0', '1' after 25 ns, '0' after 50 ns,
    '1' after 75 ns, '0' after 100 ns;
-- process statements
-- for displaying simulation results
-- header
```

process

```
    variable strbuf: LINE;
```

begin

```
    WRITE (strbuf, " time sel x1 x2 y");
    WRITELINE (output, strbuf); wait;
```

```
end process;
```

```
-- simulation results
process (sel, x1, x2, y)
    variable strbuf: LINE;
begin
    WRITE ( strbuf, NOW, RIGHT, 6);
    WRITE (strbuf, sel, RIGHT, 5);
    WRITE (strbuf, x1, RIGHT, 5);
    WRITE (strbuf, x2, RIGHT, 5);
    WRITE (strbuf, y, RIGHT, 5);
    WRITELINE (output, strbuf);
end process;
end mixed;
```

time	sel	x1	x2	y
0 ns	0	0	0	0
0 ns	0	1	0	0
25 ns	0	0	1	0
25 ns	0	0	1	1
50 ns	1	1	0	1
75 ns	1	0	1	1
75 ns	1	0	1	0
100 ns	0	1	0	0
125 ns	1	1	0	0
125 ns	1	1	0	1

Package

- 흔히 쓰이는 type이나 subprogram 등을 한 장소에 모아 선언해 됨으로써 다른 여러 설계가 공유하여 사용할 수 있도록 해 줌.
- C와 같은 프로그래밍 언어에서 include(header) file과 library(archive) file을 이용하여 type이나 subprogram을 여러 프로그램에서 공유하여 사용할 수 있게 하는 것과 흡사.
- Design entity와 마찬가지로 두 부분으로 나뉨.

package = package declaration(interface)
 + package body(subprogram body)

- package_declaration ::=
 package *package_simple_name* **is**
 package_declarative_item
 end [*package_simple_name*];

package_body ::=
 package body *package_simple_name* **is**
 package_body_declarative_item
 end [*package_simple_name*];

```

-- package declaration
package four_valued_logic is
  type MVL4 is ('0', '1', 'X', 'Z');
  type MVL4_vector is
    array(integer range <>) of MVL4;
  function "and" (l, r: MVL4) return MVL4;
  -- other function declarations such as "or" and "xor"
  component and2
    port (output: out MVL4;
          input1: in MVL4;
          input2: in MVL4);
  end component;
  -- other component_declarations such as or2 and xor2
end four_valued_logic;

```

```

-- package body
package body four_valued_logic is
  function "and" (l, r: MVL4) return MVL4 is
  begin
    if l = '0' or r = '0' then
      return '0';
    elsif l = '1' and r = '1' then
      return '1';
    else
      return 'X';
    end if;
  end;
  -- other function definitions
end four_valued_logic;

```

Design Unit 과 분석

- Design unit: 독립적으로 분석될 수 있는 단위. entity declaration, architecture body, package declaration, package body 등.
- Design unit 은 분석되면 중간 형태(intermediate form) 로 되어 독립적으로 library 에 저장됨(library unit).
- Design file 은 여러 개의 design unit 으로 이루어지며, 여러 개의 design unit 을 포함하는 physical file 로 구현.
- $\text{design_file} ::= \text{design_unit design_unit}$

$\text{design_unit} ::= \text{context_clause library_unit}$

$\text{context_clause} ::= \text{context_item}$

$\text{library_unit} ::=$
 primary_unit
 | secondary_unit

$\text{context_item} ::=$
 library_clause
 | use_clause

```

primary_unit ::=
    entity_declaration
    | configuration_declaration
    | package_declaration

```

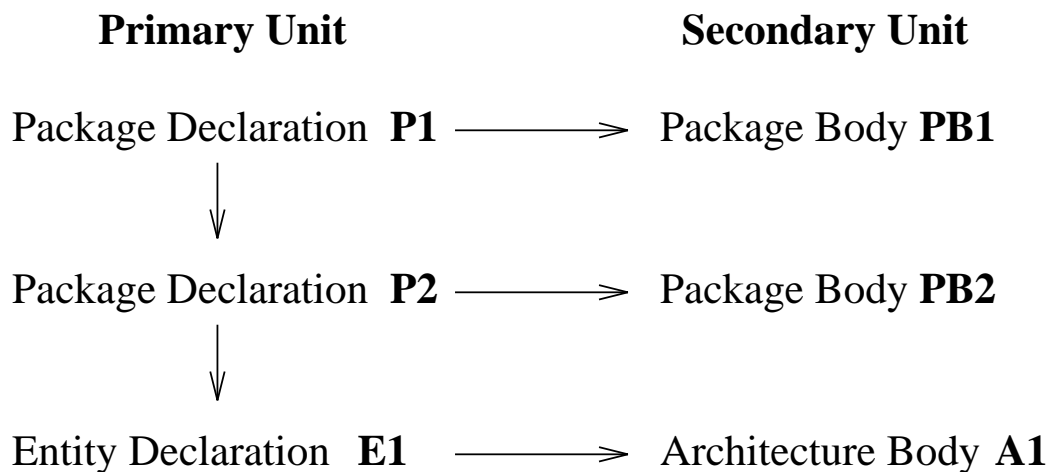
```

secondary_unit ::=
    architecture_body
    | package_body

```

• 분석 순서

- 주어진 design unit 을 분석하기 전에 그 *design unit*이 참조(reference)하는 모든 다른 primary unit 들을 분석해야 한다.
- Secondary unit 을 분석하기 전에 대응되는 primary unit 을 먼저 분석해야 한다.



Design Library

- Design unit 이 분석된 결과인 중간 형태의 자료(library unit)가 저장되는 장소.
- File 이나 directory 로 구현.
- Resource sharing 을 가능하게 해 줌.
- 각각의 design library 는 하나의 “logical name” 을 가짐.
- VHDL 은 logical name 이 STD 인 design library 를 지원. 이는 STANDARD 라는 package 와 TEXTIO 라는 package 를 포함.
- WORK 라는 logical name 을 갖는 design library 는 분석할 때에 사용되는 working library 임.
- Library 와 이에 포함된 package 를 사용하려면 library clause 와 use clause 를 이용.

```
library tech_lib;  
use tech_lib.four_valued_logic.all;  
-- all declarations in four_valued_logic is directly visible
```

- **Implicit context item:**

```
library STD, WORK;  
use STD.STANDARD.all;
```

3. Lexical Element

- Character set: ISO 646-1983 7-bit coded character set.
 - 대문자: A – Z
 - 숫자: 0 – 9
 - 특수 문자: " # & ' () * + , - . / : ; < = > _ |
 - space character: space character
 - format effector: horizontal tab, vertical tab, carriage return, line feed, form feed
 - 소문자: a – z
 - 기타 특수 문자: ! \$ % @ ? [\] ^ ` { } ~
- Character literal, string literal, comment 외에는 case-insensitive.
- Identifier: letter{[_]letter_or_digit}.
CLOCK, Clock, clock, D1, D_1A
- Literal: decimal, based, character, string, bit string.

Literal

- **Decimal literal:**

12	0	1E6	123_456	-- 정수 literal
12.0	0.0	0.456	3.14159_26	-- 실수 literal
1.34E-12	1.0E+6	6.023E24	03.14E0	-- 지수가 있는 실수 literal

- **Based literal:**

-- integer literals of value 255

2#1111_1111#16#FF# 016#0ff#

-- integer literals of value 224

16#E#E1 2#1110_0000#

-- real literals of value 4095.0

16#F.FF#E+2 2#1.1111_1111_111#E11

- **Character literal:**

'A' '*' ''' ''''

- **String literal (array of characters):**

"Setup time is too short" -- an error message

"" -- an empty string literal

" " "A" "" "" "" -- three string literals of length 1

"Characters such as \$, %, and } are allowed in string literals"

- **Bit string iiteral (array of Bits):**

B"1111_1111"

X"FF" -- equivalent to B"1111_1111"

O"377" -- equivalent to B"0_1111_1111"

4. Type 와 Object

Type

- Scalar type

- Enumeration type

```
type BOOLEAN is (FALSE, TRUE); -- predefined
```

```
type BIT is ('0', '1'); -- predefined
```

```
type MVL4 is ('0', '1', 'X', 'Z');
```

```
-- overloads '0' and '1'
```

- Integer type

```
type BYTE_LENGTH_INTEGER is range 0 to 255;
```

```
subtype HIGH_BIT_LOW is
```

```
    BYTE_LENGTH_INTEGER range 0 to 127;
```

```
type INTEGER is range implementation_defined;
```

```
subtype NATURAL is
```

```
    INTEGER range 0 to INTEGER'HIGH;
```

```
subtype POSITIVE is
```

```
    INTEGER range 1 to INTEGER'HIGH;
```

– **Physical type**

type TIME **is range** implementation_defined

units

fs; -- femtosecond

ps = 1000 fs; -- picosecond

ns = 1000 ps; -- nanosecond

us = 1000 ns; -- microsecond

ms = 1000 us; -- millisecond

sec = 1000 ms; -- second

min = 60 sec; -- minute

hr = 60 min; -- hour

end units;

type LENGTH is range 0 to 1E10

units

-- primary unit:

A; -- angstrom

-- metric lengths:

nm = 10 A; -- nanometer

um = 1000 nm; -- micrometer (or micron)

mm = 1000 um; -- millimeter

cm = 10 mm; -- centimeter

m = 1000 mm; -- meter

-- English lengths:

mil = 254000 A; -- mil

inch = 1000 mil; -- inch

end units;

- Floating point type

type REAL is range implementaion_defined

type PDF is range 0.0 to 1.0

- **Composite type**

- **Array**

type MY_WORD is array (0 to 31) of BIT;

-- a memory word type with an ascending range

type DATA_IN is array (7 downto 0) of MVL4;

-- an input data type with a descending range

**type STRING is array (POSITIVE range <>) of
CHARACTER;**

-- an array type predefined in package STANDARD

**type BIT_VECTOR is array (NATURAL range <>) of
BIT;**

-- an array type predefined in package STANDARD

**type MEMORY is array (INTEGER range <>) of
MY_WORD;**

-- a memory array type

subtype BYTE is BIT_VECTOR (7 downto 0);

type MEMORY1 is array (0 to 31, 7 downto 0) of BIT;

type MEMORY2 is array (0 to 31) of BYTE;

– **Record**

type DATE **is**

record

DAY : INTEGER **range** 1 **to** 31;

MONTH : MONTH_NAME;

YEAR : INTEGER **range** 0 **to** 4000;

end record;


```
-- recursive access type
type CELL; -- incomplete type declaration
type LINK is access CELL;
type CELL is
  record
    VALUE : INTEGER;
    SUCC : LINK;
    PRED : LINK;
  end record;
variable HEAD : LINK := new CELL' (0,null);
variable NXT : LINK := HEAD.SUCC;
```



```

-- mutually dependent access types
type PART; -- incomplete type declaration
type WIRE; -- incomplete type declaration
type PART_PTR is access PART;
type WIRE_PTR is access WIRE;
type PART_LIST is array (POSITIVE range <>)
    of PART_PTR;
type WIRE_LIST is array (POSITIVE range <>)
    of WIRE_PTR;
type PART_LIST_PTR is access PART_LIST;
type WIRE_LIST_PTR is access WIRE_LIST;
type PART is
    record
        PART_NAME : STRING (0 to 20);
        CONNECTIONS : WIRE_LIST_PTR;
    end record;
type WIRE is
    record
        WIRE_NAME : STRING (0 to 20);
        CONNECTS : PART_LIST_PTR;
    end record;

```

- **File**

- **type SFT is file of STRING** -- Defines a file type that
 - can contain an indefinite
 - number of strings

- type NFT is file of NATURAL** -- Defines a file type that
 - can contain only non-
 - negative integer values

Object

- 주어진 type의 값을 가지며 constant, variable, signal로 분류(VHDL-93에서 file 추가됨).
- Constant
 - Declaration 후에는 그 값이 변경 안됨.
 - `constant PI : REAL := 3.141592;`
`constant CYCLE_TIME : TIME := 100 ns;`
`constant Propagation_Delay;`
- Signal
 - 시간에 따라 값이 변해 온 과정을 가지는 object.
 - 하나 또는 여러 개의 driver를 가지며 driver는 현재의 값과 예상되는 미래의 값을 가짐.
 - Assign된 값이 미래의 값을 결정.
 - `signal S : BIT_VECTOR (1 to 8);`
`signal CLK1, CLK2 : TIME := 0 ns;`
`signal OUTPUT : WIRED_OR MVL9;`

- Variable

- Assignment statement 를 이용하여 그 값을 바꿀 수 있으며, signal 과 달리 그 값이 즉시 바뀐다.
- **variable** INDEX : INTEGER range 0 to 99 := 0;
 - initial value is determined by the initial value
 - expression

variable COUNT : POSITIVE;

- initial value is POSITIVE'LEFT, or 1

variable MEMORY : BIT_MATRIX (0 to 7, 0 to 1023);

- initial value is the aggregate of the initial value
- of each element

- File

```
architecture table of rom is
```

```
    type MVL4_file is file of MVL4_vector (7 downto 0);
```

```
    file rom_data: MVL4_file is in "rom.dat";
```

```
begin
```

```
    process -- (sensitivity list)
```

```
        variable first_time : BOOLEAN := TRUE;
```

```
        variable data : MVL4_vector (7 downto 0);
```

```
        variable mem is array (0 to 31) of  
            MVL4_vector (7 downto 0);
```

```
        variable address : NATURAL := 0;
```

```
begin
```

```
    if first_time = TRUE then
```

```
        -- initialization
```

```
        while not endfile(rom_data) loop
```

```
            read(rom_data, data);
```

```
            mem(address) := data;
```

```
            address := address + 1;
```

```
        end loop;
```

```
        first_time := FALSE;
```

```
    else
```

```
        -- behavior of rom
```

```
    end if;
```

```
    end process;
```

```
end table;
```

5. Structural Description

- 하드웨어를 subcomponent의 연결로 표현.
- 관련 구문 및 요소:
 - Component declaration.
 - Component instantiation.
 - Configuration specification.
 - Generate statement.

Component Declaration

- Subcomponent 를 instantiation 하여 사용하기 전에 가상적으로 선언해 줌.
- Subcomponent 의 interface 를 정의.

- component dff

```
port (d, clk, rst: in MVL4;  
      q, qb: out MVL4);  
end component;
```

```
component and_n
```

```
generic (tp1h, tphl: time := 0 ns;  
        n: positive := 2);  
port (x: in MVL4_vector(1 to n);  
      y: out MVL4);  
end component;
```

Component Instantiation Statement

- Design entity 내의 subcomponent instance를 나타내며 그 instance의 port와 signal의 연결을 표현하는 데 사용.

- `component and2`

```
port (y: out MVL4;  
      in1, in2: in MVL4);
```

```
end component;
```

```
...
```

```
a1: and2 port map (load, clk, ena);
```

```
-- positional association
```

```
a2: and2 port map (in1 => clk, in2 => ena,  
                  y => load);
```

```
-- named association
```



```

component and_n
    generic (tp1h, tphl: time := 0 ns;
              n: positive := 2);
    port (y: out MVL4;
          x: in MVL4_vector(1 to n));
end component;
...
a3: and_n port map (x(1) => clk, x(2) => ena, y=> load);
    -- use default (n = 2)
a4: and_n generic map (n => 3)
    port map (load, x(1)=>clk, x(2)=>ena, x(3)=>rst_b);
    -- positional association followed by named association

component dff
    port (d, clk, rst: in MVL4;
          q, qb: out MVL4);
end component;
...
d1: dff port map (d(1), load, rst, r(1));
d2: dff port map (d(2), load, rst, open, rb(2));
    -- unconnected port

```

Configuration Specification

- Component declaration은 가상적인 component를 선언.
- Component instantiation statement는 이 가상적 component를 instantiation해서 사용.
- Library 내의 design entity와 component instance와의 binding 필요 — configuration specification과 configuration declaration 내의 component configuration.

- -- example 1

```
entity andn is
```

```
    generic (tp1h, tphl: time := 0 ns;
```

```
              n: positive := 2);
```

```
    port (output: out MVL4;
```

```
          inputs: in MVL4_vector(1 to n));
```

```
end andn;
```

```
architecture behav of andn is
```

```
    ...
```

```
begin
```

```
    ...
```

```
end behav;
```

```

component and_n
  generic (tp1h, tphl: time := 0 ns;
           n: positive := 2);
  port (y: out MVL4;
        x: in MVL4_vector(1 to n));
end component;

-- configuration specification
for a1: and_n use entity prim.andn(behav)
  port map (inputs => x, output => y);
...
a1: and_n generic map (n => 3)
  port map (load, x(1)=>clk, x(2)=>ena, x(3)=>rst_b);

```

```
-- example 2
entity dff is
    port (d, clk, rst: in MVL4;
          q, qb: out MVL4);
end dff;

architecture struct of dff is
    ...
begin
    ...
end struct;

architecture behav of dff is
    ...
begin
    ...
end behav;
```

```
component dff
  port (d, clk, rst: in MVL4;
        q, qb: out MVL4);
end component;

-- configuration specification
for d1,d2: dff use entity WORK.dff(struct);
for others: dff use entity WORK.dff(behav);
...

d1: dff port map (d(1), load, rst, r(1));
d2: dff port map (d(2), load, rst, open, rb(2));
```

Generate Statement

- 하드웨어의 일부 표현을 여러 번 반복하여 사용할 수 있도록 해 줌.

- **entity** reg4 **is**

```
port (clk, ena, rst: in MVL4;  
      d: in MVL4_vector (4 downto 1);  
      r: out MVL4_vector (4 downto 1));
```

```
end reg4;
```

architecture struct **of** reg4 **is**

```
signal load: MVL4;
```

```
component dff
```

```
port (d, clk, rst: in MVL4;  
      q, qb: out MVL4);
```

```
end component;
```

```
component and_n
```

```
generic (tplh, tphl: time := 0 ns;  
        n: integer := 2);
```

```
port (x: in MVL4_vector (1 to n);  
      y: out MVL4);
```

```
end component;
```

```
for all: dff use entity WORK.dff(struct);
```

```
for ag1: and_n use entity prim.andn(behav)
```

```
port map (inputs => x, outputs => y);
```

begin

ag1: and_n **generic map** (n => 2)

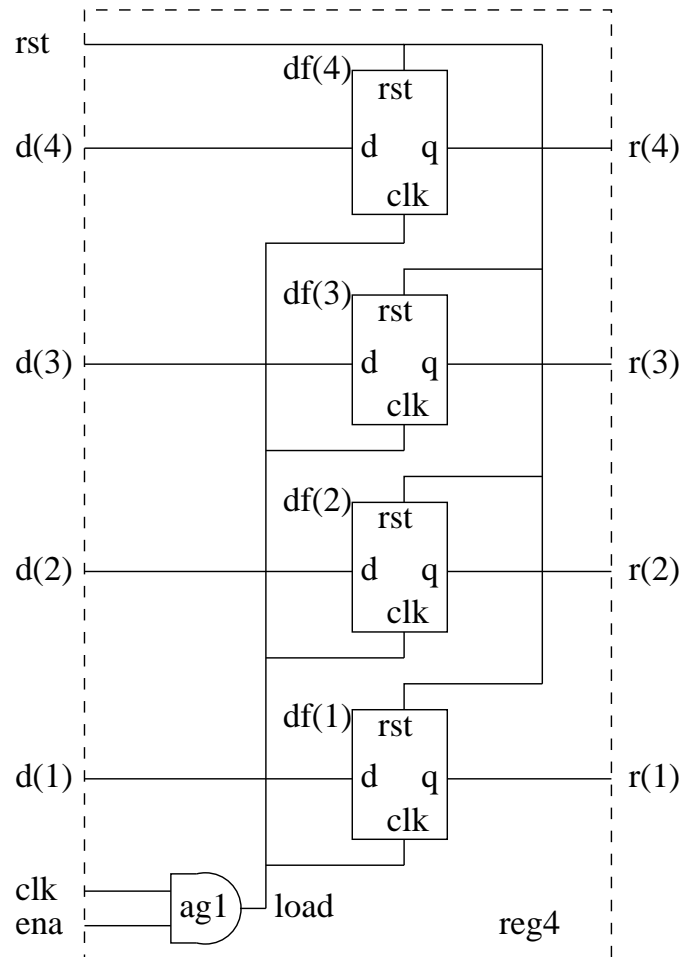
port map (x(1) => clk, x(2) => ena,
y => load);

G4: **for i in 1 to 4 generate**

df: dff **port map** (d(i), load, rst, r(i));

end generate;

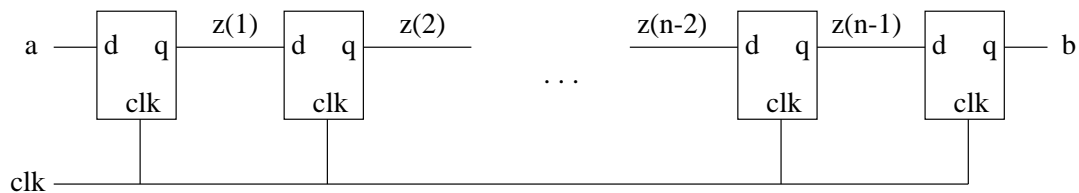
end struct;



```

component dff
  port (d, clk: in MVL4;
         q: out MVL4);
end component;
...
L1: for i in 0 to (n-1) generate
  L2: if i = 0 generate
    dffx: dff port map (a,clk,z(1));
  end generate L2;
  L3: if i = (n-1) generate
    dffx: dff port map (z(n-1),clk,b);
  end generate L3;
  L4: if (i>0) and (i<(n-1)) generate
    dffx: dff port map (z(i),clk,z(i+1));
  end generate L4;
end generate L1;

```



6. Data Flow Description

- Structural description과 behavioral description의 중간 성격.
- 주로 피연산자에 연산을 하여 signal에 assign하는 형태.
- 관련 구문 및 요소:
 - 연산자.
 - 피연산자.
 - Concurrent signal assignment statement.

연산자

- 연산자의 종류와 우선 순위

logical_operator ::= **and** | **or** | **nand** | **nor** | **xor**

relational_operator ::= = | /= | < | <= | > | >=

adding_operator ::= + | - | &

sign ::= + | -

multiplying_operator ::= * | / | **mod** | **rem**

miscellaneous_operator ::= ** | **abs** | **not**

• Logical operator

- and, nand, or, nor, xor, not
- 피연산자의 type은 BIT 또는 BOOLEAN.
- -- signal assignment statements using logical operations
 - q <= a **and** b;
 - en <= a **or** b **or** c;
 - c <= **not** (a **and** b);
 - c <= **not** a **and** b; -- equivalent to (**not** a) **and** b
 - clk <= **not** clk **after** 20 ns;
- boolean expressions
- if** a **and** c **then**
 - statements for true_expression
- else**
 - statements for false_expression
- end if;**

• Relational Operator

Operator	Operation	Operand Type	Result Type
=	equality	any type	BOOLEAN
/=	inequality	any type	BOOLEAN
< <= > >=	ordering	any scalar type or discrete array type	BOOLEAN

- 연산자 = 과 /= 은 file type 을 제외한 다른 모든 type 의 피연산자에 적용.
- <, <=, >, >= 은 모든 scalar type 또는 discrete array type 에 적용.

(null array) < (non-null array) -> TRUE

$\boxed{1, 2, 3} < \boxed{2, 3} \rightarrow \text{TRUE}$

$\boxed{1, 2, 3} < \boxed{1, 3} \rightarrow \text{TRUE}$

• Adding operator 와 sign

- +, -, &(concatenation)
- +와 -의 두 피연산자는 같은 numeric type(integer, floating point, physical type)이어야 함.
- &의 피연산자는 1차원 array 또는 array element 이어야 함.

(1-dimensional array type) & (same array type)

(1-dimensional array type) & (the element type)

(the element type) & (1-dimensional array type)

(the element type) & (the element type)

a(3 to 7) & b(2 downto 0)

- Sign +와 -는 unary operator 로 피연산자는 numeric type 이어야 함.
- Sign 은 multiplying operator 나 **, abs, not 보다 낮은 우선 순위를 가짐.

A/+B -- illegal expression

A**~B -- illegal expression

A/(+B) -- legal expression

A**(-B) -- legal expression

• Multiplying operator

- *와 /는 integer type과 floating point type의 피연산자에 대하여 정의됨.
- (physical type) * (INTEGER 또는 REAL) -- physical type
- (INTEGER 또는 REAL) * (physical type) -- physical type
- (physical type) / (INTEGER 또는 REAL) -- physical type
- (physical type) / (physical type) -- universal integer type
- rem과 mod의 피연산자는 integer type이어야 함.
- a := 7 **rem** 3; -- a becomes 1
- b := 7 **mod** 3; -- b becomes 1
- c := (-7) **rem** 3; -- c becomes -1
- d := (-7) **mod** 3; -- d becomes 2
- e := 7 **rem** (-3); -- e becomes 1
- f := 7 **mod** (-3); -- f becomes -2
- g := (-7) **rem** (-3); -- g becomes -1
- h := (-7) **mod** (-3); -- h becomes -1

- Miscellaneous operator

- abs의 피연산자는 numeric type이어야 함.
- **는 모든 integer type과 floating point type에 대하여 정의됨.
- **의 왼쪽 피연산자는 integer type과 floating point type이 모두 허용되지만 오른쪽 피연산자는 INTEGER이어야 함.
- **의 exponent가 음의 값을 가지는 경우 왼쪽 피연산자는 반드시 floating point type이어야 함.

Concurrent Signal Assignment Statement

- 주어진 값, 입력 signal, 또는 이에 대한 연산의 결과 등을 delay 에 대한 정보와 함께 출력 signal에 전달.
- 각 concurrent signal assignment statement는 하나의 독립된 concurrent process로서 동작. 입력 signal에 변화가 있을 때마다 반복 수행됨.
- Conditional signal assignment

```
qb <= not q;
```

```
qb <= not q after 0 ns; -- equivalent
```

```
q <= not q; -- oscillation
```

```
architecture data_flow of dff is
```

```
begin
```

```
    q <= '0' when rst = '0' else
```

```
        d when clk = '1' and (not clk' stable) else
```

```
        -- rising edge
```

```
        q;
```

```
    qb <= not q;
```

```
end data_flow;
```

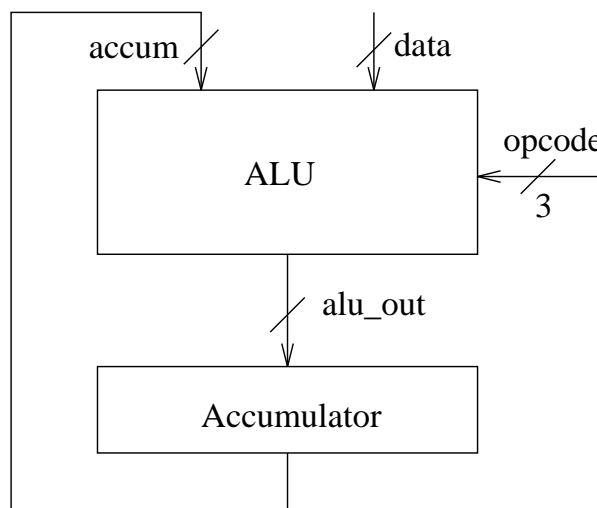


```
architecture data_flow of register is  
begin  
    reg <= data after 2 ns when enable = '0' and  
        clk = '1' and (not clk'stable) else  
        reg;  
end data_flow;
```

- Selected signal assignment

S1: **with** opcode **select**

```
alu_out <= data + accum after 3 ns when "000",  
          data - accum after 3 ns when "001",  
          data and accum after 3 ns when "010",  
          data xor accum after 3 ns when "011",  
          data when "100"|"101",  
          accum when others; -- last choice
```



- **Concurrent assertion statement**

```
ASS1: assert Clk /= 'X'
```

```
    report "Unknown value on Clk"    -- report if Clk = 'X'
```

```
    severity Error;
```

```
    -- one of NOTE, WARNING, ERROR, FAILURE
```

7. Behavioral Description

Process Statement

- Behavioral description은 process statement로 이루어짐.
- Process statement는 sequential statement로 구성.
- 내부의 sequential statement들을 순서대로 wait statement를 만날 때까지 반복 수행함.
- Sensitivity list가 있는 경우에는 그 list에 포함되어 있는 signal의 값이 변해야만 sequential statement들이 반복 수행됨.
- Process statement의 sensitivity list는 wait statement가 그 process 내의 마지막 sequential statement로 삽입되어 있는 것과 동일함.
- Process statement에 sensitivity list가 있는 경우에는 그 내부에 다른 wait statement를 사용할 수 없음.
- Sensitivity list도 없고 wait statement도 없는 process statement는 무한히 반복 수행되어 시뮬레이션이 끝날 수 없게 되므로 주의해야 함.

-- Three equivalent architecture bodies

```
architecture EX1 of and2 is  
begin  
    y <= a and b after 7 ns; -- concurrent  
end EX1;
```

```
architecture EX2 of and2 is  
begin  
    process (a, b) -- sensitivity list  
    begin  
        y <= a and b after 7 ns; -- sequential  
    end process;  
end EX2;
```

```
architecture EX3 of and2 is  
begin  
    p1: process  
    begin  
        y <= a and b after 7 ns; -- sequential  
        wait on a, b; -- sequential, explicit wait statement  
    end process p1;  
end EX3;
```

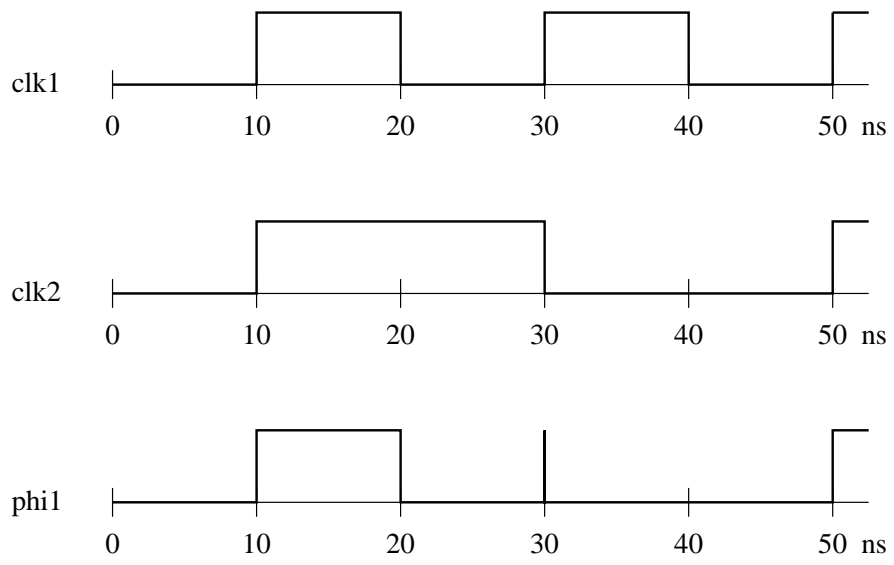
```

architecture behav of clock_gen is
    signal clk1, clk2, phi1: MVL4 := '0';
begin
    c1: process (clk1)
    begin
        clk1 <= not clk1 after 10 ns;
    end process c1;

    c2: process
    begin
        wait until clk1 = '1'; -- wait for a rising edge
        clk2 <= not clk2;
    end process c2;

    p1: process(clk1, clk2)
    begin
        phi1 <= clk1 and clk2;
    end process p1;
end behav;

```



Sequential Statement

- **Process** 나 **subprogram** 내에서 순서대로 수행됨.
- `sequential_statement ::=`
 - `wait_statement`
 - | `assertion_statement`
 - | `signal_assignment_statement`
 - | `variable_assignment_statement`
 - | `procedure_call_statement`
 - | `if_statement`
 - | `case_statement`
 - | `loop_statement`
 - | `next_statement`
 - | `exit_statement`
 - | `return_statement`
 - | `null_statement`

- **Wait statement**

- **Process statement** 나 **procedure** 의 수행을 일시 정지 시킴.

- `wait_statement ::=`

- `wait [sensitivity_clause] [condition_clause]`
`[timeout_clause];`

- `sensitivity_clause ::= on sensitivity_list`

- `condition_clause ::= until condition`

- `timeout_clause ::= for time_expression`

- **Sensitivity clause** 가 명기되어 있지 않으면 **condition clause** 의 조건을 구성하는 모든 **signal** 이 **sensitivity list** 를 구성하고 있는 것처럼 동작.

– **wait**; -- indefinite wait

wait on oe, wr, rd; -- sensitivity clause

wait until clk = '1'; -- condition clause, rising edge

wait for 50 ns; -- timeout clause

wait on oe, wr, rd -- reactivate when one of the
until ce = '0'; -- signals in the sensitivity
-- list changes AND the
-- condition is true

wait on oe, wr, rd -- reactivate when one of the
until ce = '0' -- of the signals in the sensitivity
for 200 ns; -- list changes AND the
-- condition is true OR when
-- the allotted time has elapsed

- Assertion statement

- 설계한 것이 원하는 대로 동작하는지 검증하는 데 사용.
- 기능면에서 concurrent assertion statement 와 같음.

- Signal assignment statement

- Process statement 나 procedure 내에서 signal에 어떤 값을 전달.
- Driver가 갖고 있는 projected output waveform을 변경.
- Target의 값이 즉시 변하지는 않으며, 적어도 delta delay는 걸림.
- process (a, b)
begin
 y <= 10.0 after 20 ns, 20.0 after 40 ns;
 x <= a xor b; -- implicit "after 0 ns" clause
 (a, b) <= "10"; -- aggregate target
end process;

- Variable assignment statement

- Target variable의 값을 즉시 변경.

- process

- variable today, tomorrow: date;

- begin

- ...

- today.month := October;

- today.day := 12;

- today.year := 1993;

- tomorrow := (13, October, 1993);

- right-hand side expression is an aggregate

- ...

- end process;

- process (...)

- variable first_time: boolean := true; -- boolean variable

- begin

- if first_time = true then

- ...-- initialization

- first_time := false;

- end if;

- ...

- end process;

```
data(3 downto 0) := "1100"; -- bit_vector variable
```

```
addr(7 downto 4) := data(0 to 3);
```

```
-- implicit subtype conversion
```

- Process 내의 variable 은 process 가 정지 상태에 있어도 그 값이 계속 유지되나 procedure 내의 variable 은 return 할 때까지만 그 값을 유지하며, 일단 return 하고 나면 그 값을 잃음.

- **If statement**

```
- if (rst_b = '0') then  
    q <= 0;  
elsif (clk = '1') and clk'event then  
    if (load_b = '0') then  
        q <= d;  
    else  
        q <= q + 1;  
    end if  
end if;
```

- **Case statement**

- **case** state **is**

- when** 0 =>

- if input = '1' then

- state := 1;

- else

- null;

- end if;

- when** 1 =>

- if input = '0' then

- state := 3;

- else

- state := 2;

- end if;

- when** 2 | 4 =>

- state := 0;

- when** 3 =>

- if input = '0' then

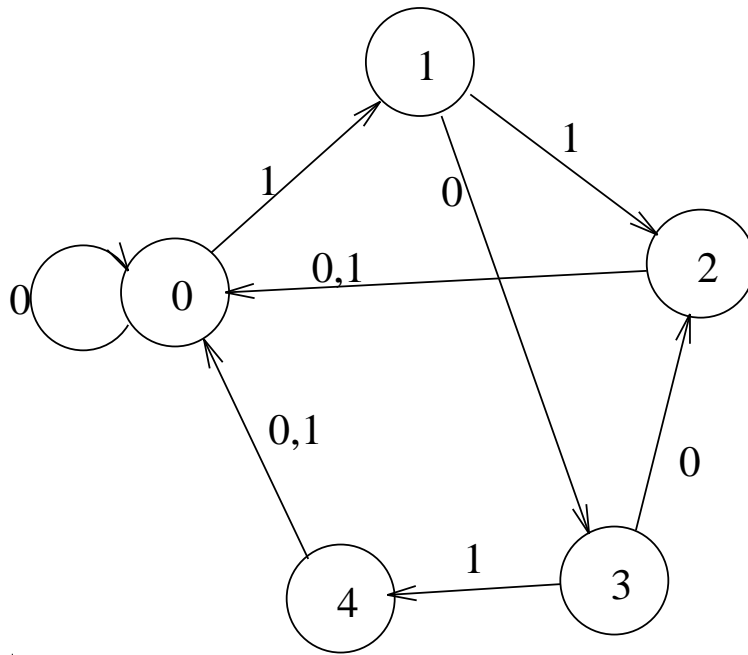
- state := 2;

- else

- state := 4;

- end if;

- end case;**



```

- case opcode is
  when "000" => alu_out <= data + accum after 3 ns;
  when "001" => alu_out <= data - accum after 3 ns;
  when "010" => alu_out <= data and accum after 3 ns;
  when "011" => alu_out <= data xor accum after 3 ns;
  when "100" | "101" => alu_out <= data;
  when others => alu_out <= accum;
end case;

```

-- equivalent statement

-- concurrent conditional signal assignment

S1: **with** opcode **select**

```

alu_out <= data + accum after 3 ns when "000",
      data - accum after 3 ns when "001",
      data and accum after 3 ns when "010",
      data xor accum after 3 ns when "011",
      data when "100"|"101",
      accum when others; -- last choice

```

• Loop statement

– – infinite loop

l1: loop

 a := a + 1;

end loop l1;

– – **for** iteration scheme

– – shift left characters in a string

f1: for i in 0 to 3 loop

 str(i) <= str(i + 1);

end loop f1;

– – **while** iteration scheme

– – get the index of an array element whose value is '0'

while val(index) /= '0' loop

 index := index + 1;

end loop;

- Next statement

- Loop statement 에서 현재의 반복 수행을 중단.
- Loop statement 가 완료되지 않은 경우에는 다음 반복 수행을 계속.
- next_statement ::=
 next [*loop_label*] [**when** condition];
- Loop label 이 있는 next statement 는 해당 loop 에 적용.
- Loop label 이 없는 next statement 는 그 next statement 를 포함하는 loop 중 가장 내부의 loop 에 적용.
- outer_loop: **for** j **in** 1 **to** 10 **loop**
 inner_loop: **for** i **in** 1 **to** 10 **loop**
 if j = 4 **and** i = 5 **then**
 next outer_loop;
 end if;
 x(j,i) := y(i,j);
 end loop inner_loop;
end loop outer_loop;

- Exit statement

- 이를 포함하는 loop statement 의 수행을 완료하고 빠져 나오는 데 사용.
- exit_statement ::=
 exit [*loop_label*] [**when** condition];
- Loop label 이 있는 exit statement 는 해당 loop 에 적용.
- Loop label 이 없는 exit statement 는 그 exit statement 를 포함하는 loop 중 가장 내부의 loop 에 적용.
- l1: loop
 a := a + 1;
 exit l1 **when** a = 10;
end loop l1;

- Null statement

- 다음 statement 로 수행 순서를 넘겨 주는 것 외에는 아무 일도 하지 않음.
 - case opcode is
 - when "000" => accum <= data + accum after 3 ns;
 - when "001" => accum <= data - accum after 3 ns;
 - when "010" => accum <= data and accum after 3 ns;
 - when "011" => accum <= data xor accum after 3 ns;
 - when "100" | "101" => accum <= data;
 - when others => null;
- end case;

8. Subprogram

Function

- Delay 없이 계산 결과를 return.
- Type conversion, overloading operator, signal resolution 등에 이용.
- 내부에서 signal assignment statement를 사용할 수 없음.
- Function declaration function body의 두 부분으로 정의되며, 그 사용은 function call로 이루어짐.

- **Type conversion**

- -- type definition in package declaration

```
subtype BYTE is BIT_VECTOR (7 downto 0);
```

- -- function declaration in package declaration

```
function byte_to_int (x:BYTE) return integer;
```

```
function int_to_byte (x:integer) return BYTE;
```

- -- function body in package body

```
function byte_to_int (x:BYTE) return integer is
```

```
    variable sum: integer := 0;
```

```
    variable weight: integer := 1;
```

```
begin
```

```
    for i in 0 to 7 loop
```

```
        if x(i) = '1' then
```

```
            sum := sum + weight;
```

```
        end if;
```

```
        weight := weight * 2;
```

```
    end loop;
```

```
    return sum;
```

```
end;
```



```

function int_to_byte (x:integer) return BYTE is
    variable result: BYTE;
    variable temp: integer;
begin
    temp := x;
    for i in 0 to 7 loop
        if temp mod 2 = 1 then
            result(i) := '1';
        elsif
            result(i) := '0';
        end if;
        temp := temp / 2;
    end loop;
    return result;
end;

```

-- signal declaration in architecture body

```

signal count: BYTE;

```

-- function call in architecture body

```

count <= int_to_byte (byte_to_int (count) + 1);

```

• Operator overloading

- -- function body
- "not" is overloaded to handle multiple valued logic

```
function "not" (op1: in MVL4) return MVL4 is  
begin
```

```
    case op1 is
```

```
        when '0' => return '1';
```

```
        when '1' => return '0';
```

```
        when others => return 'X';
```

```
    end case;
```

```
end "not";
```

```
architecture df of example is
```

```
    signal x, y, a, b: MVL4
```

```
    -- function declaration
```

```
    function "not" (op1: in MVL4) return MVL4;
```

```
begin
```

```
    ...
```

```
    -- following two styles are possible
```

```
    a <= "not"(b);      -- used as a function
```

```
    y <= not x;        -- used as an operator
```

```
    ...
```

```
end;
```

- Signal resolution

- 다수의 driver 를 가진 signal 의 값을 driver 의 값들로부터 계산하기 위하여 resolution function 사용.

```

- architecture behav of res_ex is
    function wired_or (inputs: bit_vector) return bit;
    signal res_sig: wired_or bit;
    signal dr1, dr2: bit;
begin
    source_1: process(dr1)
    begin
        res_sig <= dr1;
    end process;
    source_2: process(dr2)
    begin
        res_sig <= dr2;
    end process;
end;

function wired_or (inputs: bit_vector) return bit is
begin
    for i in inputs'range loop
        if inputs(i) = '1' then
            return '1';
        end if;
    end loop;
    return '0';
end;

```

Procedure

- Procedure declaration과 procedure body의 두 부분으로 정의되며, 그 사용은 procedure call에 의하여 이루어짐.

- **procedure** finish is

begin

assert false

report "Simulation complete";

severity failure;

end;

procedure monitor (name: **in** string; value: **in** bit;) **is**
variable ...;

begin

WRITE (strbuf, NOW, RIGHT, 6);

WRITE (strbuf, " " & name & "=");

WRITE (strbuf, value);

WRITELINE (output, strbuf);

end;

-- sequential procedure call

process

wait for 175 ns;

 finish;

end process;

-- concurrent procedure call

architecture behav **of** proc_ex **is**

begin

 ...

 monitor("clock", clk);

 monitor("read", rd);

 monitor("write", wt);

end behav;

Subprogram Interface

- Formal parameter는 constant, variable, signal 등의 object class로 분류.

- in, inout, out 중 한 가지 mode를 취함.

- interface_constant_declaration ::=
 [**constant**] identifier_list: [**in**] subtype_indication
 [:= *static_expression*]

interface_signal_declaration ::=
 [**signal**] identifier_list: [mode] subtype_indication
 [:= *static_expression*]

interface_variable_declaration ::=
 [**variable**] identifier_list: [mode] subtype_indication
 [:= *static_expression*]

- Mode가 명시되어 있지 않으면 in으로 간주.
- Function의 경우 formal parameter의 mode는 in만 허용되고 object class는 constant와 signal만 허용.
- 만일 mode가 in이고 object class가 명시되어 있지 않으면 constant로 간주.
- 만일 mode가 inout이나 out이고 object class가 명시되어 있지 않으면 variable로 간주.

- Formal parameter가 signal이면 actual parameter도 signal.
- Formal parameter가 variable이면 actual parameter도 variable.
- Formal parameter가 constant이면 actual parameter는 expression.

9. 기타

Attribute

- 이름이 붙여진 것(named entity)은 어느 것이나 그것이 갖는 특징을 정의하기 위하여 attribute를 사용할 수 있음.
- Predefined attribute
 - 표준 VHDL에서는 미리 정의해 놓고 있는 attribute.
 - HIGH
 - * INTEGER 'HIGH는 value로 INTEGER type이 가질 수 있는 가장 큰 값.
 - * type MEMORY1 is array (0 to 31, 7 downto 0) of BIT;
의 경우 MEMORY1 'HIGH(2)는 function이며 7을 return.
 - RANGE
 - * 위에서 정의한 MEMORY1에 대하여 MEMORY1 'RANGE(2)는 7 downto 0.
 - LENGTH
 - * 위에서 정의한 MEMORY1에 대하여 MEMORY1 'LENGTH(2)는 $8(\text{MEMORY1 'HIGH}(2) - \text{MEMORY1 'LOW}(2) + 1)$.

- DELAYED

* S'DELAYED(T)는 S를 T 시간만큼 지연시킨 것과 같은 signal.

P: process(S)

begin

R <= transport S after T;

end process;

- STABLE

* S'STABLE(T)는 S에 T 시간동안 event가 발생하지 않았으면 TRUE, 발생했으면 FALSE인 signal.

- EVENT

* S'EVENT는 S에 현재의 시뮬레이션 사이클 동안 event가 발생했으면 TRUE, 아니면 FALSE를 return하는 function.

● User-defined attribute

- 사용자가 임의로 정의하여 사용할 수 있는 attribute.

- 호환성을 유지하는 데에는 장애가 되므로 주의.

```

- package physical_attributes is
    type physical_size is record
        width, height: Integer;
    end record;
    type location is record
        x, y: Integer;
    end record;
    attribute layout_size: physical_size;
    attribute placement: location;
    attribute pin_number: Integer;
    attribute width: Integer;
end physical_attributes;

architecture nand_impl of sr_flip_flop is
    component nand_gate
        port (a, b: in bit; y: out bit);
    end component;
    attribute placement of nand1: label is (200, 100);
    attribute placement of nand2: label is (200, 90);
begin
    nand1: nand_gate port map (s, qbar, q);
    nand2: nand_gate port map (s, q, qbar);
end nand_impl;

```

Delay Modeling

- Transport delay model

- 다음과 같은 과정을 수행.

1. Projected output waveform에 추가하고자 하는 새로운 transaction의 시점 이후의 시점에 이미 추가되어 있는 transaction이 있으면 모두 제거한다.
2. Projected output waveform에 새로운 transaction을 모두 추가한다.

- `signal s: integer := 0;`

...

`process`

`begin`

`-- transport delay model`

`s <= transport 1 after 1 ns,`

`2 after 3 ns,`

`3 after 5 ns;`

`wait;`

`end process;`

`(0, 0 ns) (1, 1 ns) (2, 3 ns) (3, 5 ns)`

`s <= transport 4 after 4 ns;`

`(0, 0 ns) (1, 1 ns) (2, 3 ns) (4, 4 ns)`

• Inertial delay model

- 다음과 같은 과정을 추가로 수행.
 1. 새로운 transaction에 모두 표를 한다.
 2. 가장 이른 시점의 새로운 transaction과 그 직전에 있는 기존의 transaction이 같은 값을 가지면 그 기존의 transaction에 표를 한다.
 3. Driver의 현재 값에 해당하는 transaction에 표를 한다.
 4. 표가 되어 있지 않은 모든 transaction을 제거한다.
- `signal s: integer := 0;`

...

`process`

`begin`

`-- inertial delay model`

`s <= 1 after 1 ns,`

`3 after 3 ns,`

`5 after 5 ns;`

`wait;`

`end process;`

`(0, 0 ns) (1, 1 ns) (3, 3 ns) (5, 5 ns)`

`s <= 3 after 4 ns,`

`4 after 5 ns;`

`(0, 0 ns) (3, 3 ns) (3, 4 ns) (4, 5 ns)`

Entity Statement

- Entity declaration 내에 포함되어 주로 입력 신호의 setup/hold time이나 pulse width 등 인터페이스를 검사하기 위한 목적으로 사용.

- **entity** Latch **is**

port (Din: **in** Word;

 Dout: **out** Word;

 Load: **in** Bit;

 Clk: **in** Bit);

constant Setup: Time := 12 ns;

constant PulseWidth: Time := 50 ns;

use WORK.TimingMonitors.**all**;

begin

 -- concurrent assertion statement

 assert Clk='1' **or** Clk'Delayed'Stable(PulseWidth);

 -- concurrent procedure call

 CheckTiming(Setup, Din, Load, Clk);

end;

Block Statement

- Subcomponent 를 component instantiation statement 를 사용하지 않고 직접 기술.
- Nesting 이 가능하여 계층 구조를 표현할 수 있음.
- ... -- full adder description

```
FAdder: block
  signal P, G : bit;
begin
  -- generate P and G signal
  PGgen: block
  begin
    P <= A xor B;
    G <= A and B;
  end block PGgen;
  -- generate carry signal
  Cout <= G or (P and Cin);
  -- generate sum signal
  Sum <= P xor Cin;
end block FAdder;
...
```

```

...
-- full adder description
FAdder: block
    signal P, G : bit;
begin
    -- generate P and G signal
    PGgen: block
        port(in1, in2 in : bit;
            out1, out2 out : bit);
        port map(in1 => A, in2 => B, out1 => P, out2 => G);
    begin
        out1 <= in1 xor in2;
        out2 <= in1 and in2;
    end block PGgen;
    -- generate carry signal
    Cout <= G or (P and Cin);
    -- generate sum signal
    Sum <= P xor Cin;
end block FAdder;
...

```



```

...
-- component declaration
component FAddComp
    port(LA, LB, LCin in : bit;
          LCout, LSum out : bit);
end component
-- configuration specification
for FAdder use entity FullAdder(dflow)
    port map (in1 => LA, in2 => LB, in3 => LCin,
              out1 => LCout, out2 => LSum);
...
-- full adder description
FAdder: FullAdder
    port map (LA => A, LB => B, LCin => Cin,
              LCout => Cout, LSum => Sum);
...

```

```

entity FullAdder is
    port(in1, in2, in3 in : bit;
          out1, out2 out : bit);
end FullAdder;

architecture dflow of FullAdder is
    signal P, G : bit;
begin
    -- generate P and G signal
    PGgen: block
    begin
        P <= in1 xor in2;
        G <= in1 and in2;
    end block PGgen;
    -- generate carry signal
    out1 <= G or (P and in3);
    -- generate sum signal
    out2 <= P xor in3;
end dflow;

```

- Guard expression

- GUARD 라는 boolean type 의 signal 값을 결정.
- Guarded assignment 의 수행 여부를 결정.
- Guarded target 의 경우 disconnect 여부를 결정.

```
function wired_or (inputs: bit_vector) return bit is
    constant float_value: bit := '0';
begin
    if inputs'length = 0 then
        -- this is a bus whose drivers are all off
        return float_value;
    else
        for i in inputs'range loop
            if inputs(i) = '1' then
                return '1';
            end if;
        end loop;
        return '0';
    end if;
end;
```

Alias Declaration

- 또 다른 이름을 붙여 주는 데 사용.
- `variable REAL_NUMBER : BIT_VECTOR(0 to 31);`
`alias SIGN : BIT is REAL_NUMBER(0);`
`alias MANTISSA : BIT_VECTOR(23 downto 0) is`
`REAL_NUMBER(8 to 31);`
`alias EXPONENT : BIT_VECTOR(1 to 7) is`
`REAL_NUMBER(1 to 7);`

Configuration Declaration

- 별도의 library unit 에서 component instance 와 design entity 의 binding 을 해 줌.
- Architecture body 는 그대로 두고 component 의 binding 에 대한 정보를 모아 놓은 configuration declaration 만 수정.

- -- an architecture of a microprocessor

architecture Structure_View **of** Processor **is**

component ALU **port** (...) **end component**;

component MUX **port** (...) **end component**;

component Latch **port** (...) **end component**;

begin

A1: ALU **port map** (...);

M1: MUX **port map** (...);

M2: MUX **port map** (...);

M3: MUX **port map** (...);

L1: Latch **port map** (...);

L2: Latch **port map** (...);

end Structure_View;

```

library TTL, WORK;
configuration V4_27_87 of Processor is
    use WORK.all;
    for Structure_View -- (external) block configuration
        for A1:ALU -- component configuration
            use configuration TTL.SN74LS181;
        end for;
        for M1,M2,M3: MUX
            use entity Multiplex4(Behavior);
        end for;
        for all: Latch
            -- use defaults
        end for;
    end for;
end V4_27_87;

```

```
for D1: DSP
  for DSP_STRUCTURE
    -- Binding specified in design entity or else defaults
    for Filterer
      -- Configuration items for filtering components
    end for;
    for Processor
      -- Configuration items for processing components
    end for;
  end for;
end for;
```

10. VHDL-87과 VHDL-93의 다른 점

Group

- 여러 개의 `named entity` 를 하나의 그룹으로 다룰 수 있도록 함.

- `group PIN2PIN is (signal, signal);`

-- Groups of this type consist of two signals

`group RESOURCE is (label<>);`

-- Groups of this type consist of any number of labels

`group DIFF_CYCLES is (group <>);`

-- A group of groups

`group G1: RESOURCE(L1, L2);`

-- A group of two labels

`group G2: RESOURCE(L3, L4, L5);`

-- A group of three labels

`group C2Q: PIN2PIN(PROJECT.GLOBALS.CK,Q);`

-- Groups may associate named

-- entities in different declarative

-- parts (and regions)

`group CONSTRAINT1: DIFF_CYCLES(G1, G2);`

-- A group of groups

`attribute PROPAGATION_DELAY: TIME;`

`attribute PROPAGATION_DELAY of C2Q: group`

File

- VHDL-87에서는 file이 variable object로 분류.
- VHDL-93에서는 네 번째 object로 file object가 추가됨.
- File을 마음대로 열고 닫을 수 있음(procedure FILE_OPEN, FILE_CLOSE 추가).
- Mode 지정 가능(READ_MODE, WRITE_MODE, APPEND_MODE).
- FILE_OPEN_STATUS 지원.

Shared Variable

- 일반 프로그래밍 언어에서의 global variable 역할을 함.
- Signal로 처리할 수 없는 경우(같은 시점에서 값이 계속 변할 때 앞의 값들이 무시되면 곤란한 경우)에 편리함.

- variable_declaration ::=

```
[shared] variable identifier_list : subtype_indication  
    [ := expression];
```

Impure Function

- Stack operation 또는 난수 발생기: 같은 actual parameter 의 값으로 같은 function 을 불리도 다른 값을 return.
- Impure function 에서는 file 이나 shared variable 과 같은 외부 자료를 읽고 쓸 수 있음(side effect).

```
• file F: TEXT is "filename";  
  impure function GET_INTEGER return INTEGER is  
    variable L: LINE;  
    variable DATA: INTEGER;  
  begin  
    READLINE(F, L);  
    READ(L, DATA);  
    return DATA;  
  end GET_INTEGER;
```

Component Instantiation

- Design entity를 instantiation 할 때 component declaration과 configuration specification이 없어도 됨.
- VHDL-93 syntax:

component_instantiation_statement ::=

instantiation_label:

instantiated_unit

[generic_map_aspect]

[port_map_aspect];

instantiated_unit ::=

[**component**] *component_name*

| **entity** *entity_name* [(*architecture_identifier*)]

| **configuration** *configuration_name*

- -- instantiation of a component

component

COMP **port**(A,B:inout BIT);

end component;

for C: COMP **use entity** X(Y)

port map(P1 => A, P2 =>B);

...

C: COMP **port map**(A => S1, B =>S2);

-- instantiation of a design entity

C: **entity** Work.X(Y) **port map**(P1 => S1, P2 =>S2);

-- instantiation of a design entity through

-- configuration declaration

configuration Alpha **of** X **is**

for Y

...

end for;

end configuration Alpha;

C: **configuration** Work.Alpha

port map(P1 => S1, P2 =>S2);

Incremental Binding

- **Primary binding:** configuration specification에서의 binding
- **Incremental binding:** component configuration에서의 binding. Configuration declaration에서 generic의 값이나 port의 연결 변경 가능. Backannotation에 이용 가능.

- **entity XOR_GATE is**
 generic(I1toO,I2toO: DELAY_LENGTH := 4 ns);
 port(I1,I2: **in** BIT; O: **out** BIT);
end entity XOR_GATE;

archetecture Structure of Half_Adder is

```

...
component XOR_GATE is
    generic(I1toO,I2toO: DELAY_LENGTH);
    port(I1,I2: in BIT; O: out BIT);
end component XOR_GATE;
for L1: XOR_GATE use
    entity WORK.XOR_GATE(Behavior)
        -- primary binding indication
        generic map(3 ns, 3 ns)
        port map(I1=>I1, I2=>I2, O=>O);
    ...
begin
    ...
    L1: XOR_GATE port map(X,Y,Sum);
    ...
end architecture Structure;

```

configuration Different of Half_Adder is

```

for Structure
    for L1: XOR_GATE
        generic map(2.9 ns, 3.6 ns);
        -- incremental binding indication
    end for
    ...
end for
end configuration Different;

```

Postponed Process

- **Postponed process:** 마지막 시뮬레이션 사이클에서만 수행되는 process.
- `Qbar <= not Q;`
`postponed assert Qbar = not Q`
`report "Error in Qbar";`

Operator

- Logical operator: and, or, nand, nor, xor, not에 xnor 추가.
- Shift operator: sll, srl, sla, sra, rol, ror 추가.

Inertial Delay Model

- **Transport delay:** 폭이 작은 펄스도 모두 통과.
- **Inertial delay:** 주어진 시간 이내의 폭을 갖는 펄스는 제거됨. 제거를 위한 기준 시간(pulse rejection limit)을 출력력이 지연되는 시간보다 작게 할 수 있음.
- -- The following three assignments are equivalent to each other
Output_pin <= Input_pin **after** 10 ns;
Output_pin <= **inertial** Input_pin **after** 10 ns;
Output_pin <= **reject** 10 ns **inertial** Input_pin **after** 10 ns;

-- The following two assignments are equivalent to each other
Output_pin <= **transport** Input_pin **after** 10 ns;
Output_pin <= **reject** 0 ns **inertial** Input_pin **after** 10 ns;

Output_pin <= **reject** 7 ns **inertial** Input_pin **after** 10 ns;
-- results in the same output as the following two assignments
-- involving an extra signal Temp
Temp <= **inertial** Input_pin **after** 7 ns;
Output_pin <= **transport** Temp **after** 3 ns;

Predefined Attribute

- BEHAVIOR 와 STRUCTURE 삭제.
- ASCENDING, IMAGE, VALUE, DRIVING, DRIVING_VALUE, SIMPLE_NAME, INSTANCE_NAME, PATH_NAME 추가.

Foreign Architecture와 Foreign Subprogram

- Non-VHDL 부분과 VHDL 부분의 인터페이스를 제공.
- **function F return INTEGER;**
attribute FOREIGN of F: function is
 "implementation-dependent information";

Unaffected

- 기존의 문제:

```
architecture data_flow of latch is
begin
  with clock select
    output <= input after 5 ns when '1',
              output when others;
end data_flow;
```

Signal output 의 driver 가 기존에 갖고 있는 transaction 이 새로운 transaction 에 의하여 모두 제거됨.

- 수정 후:

```
architecture data_flow of latch is
begin
  with clock select
    output <= input after 5 ns when '1',
              unaffected when others;
end data_flow;
```

Generate Statement 내의 Declarative Part

- 기존의 문제:

```
entity reg4 is
  port (clk, ena, rst: in MVL4;
        d: in MVL4_vector (4 downto 1);
        r: out MVL4_vector (4 downto 1));
end reg4;
```

```
architecture struct of reg4 is
  signal load: MVL4;
  component dff
    port (d, clk, rst: in MVL4;
          q, qb: out MVL4);
  end component;
  component and_n
    generic (tplh, tphl: time := 0 ns;
             n: integer := 2);
    port (x: in MVL4_vector (1 to n);
          y: out MVL4);
  end component;
  for all: dff use entity WORK.dff(struct);
  -- configuration specification of df in G4?
  for ag1: and_n use entity prim.andn(behav)
    port map (inputs => x, outputs => y);
begin
  ag1: and_n generic map (n => 2)
    port map (x(1) => clk, x(2) => ena,
              y => load);
  G4: for i in 1 to 4 generate
    df: dff port map (d(i), load, rst, r(i));
  end generate;
end struct;
```

- 수정 후:

```
architecture struct of reg4 is
  signal load: MVL4;
  component dff
    port (d, clk, rst: in MVL4;
          q, qb: out MVL4);
  end component;
  component and_n
    generic (tplh, tphl: time := 0 ns;
             n: integer := 2);
    port (x: in MVL4_vector (1 to n);
          y: out MVL4);
  end component;
  for ag1: and_n use entity prim.andn(behav)
    port map (inputs => x, outputs => y);
begin
  ag1: and_n generic map (n => 2)
    port map (x(1) => clk, x(2) => ena,
              y => load);
  G4: for i in 1 to 4 generate
    for df: dff use entity WORK.dff(struct);
      -- configuration specification of df
    begin
      df: dff port map (d(i), load, rst, r(i));
    end generate;
end struct;
```

Signature

- Attribute name에서의 prefix나 entity designator 또는 alias declaration에서의 name이 overload된 sub-program이나 overload된 enumeration literal인 경우 그 구분이 필요.
- **function** "or" (Left, Right: MVL) **return** MVL;
attribute BuiltIn **of**
 "or" [MVL, MVL **return** MVL]: **function is** TRUE;
 -- Because of the presence of the signature, this attribute
 -- specification decorates only the "or" function
 --declared above.

type OpCode **is** (NOP,ADD,SUB,AND,OR,JMP);

attribute Mapping **of** JMP [**return** OpCode]: **literal**
 is "001";

Alias

- Object 이외에도 alias 를 붙여서 사용할 수 있도록 확장.
- Q 에 import 된 X 에 대한 alias 를 “alias Y is X;” 와 같이 선언하면 Y 는 R 로 export 될 수 있음.

- `alias STD_BIT is STD.STANDARD.BIT;`
 - explicit alias declaration
 - implicit alias declarations
 - `alias '0' is STD.STANDARD.'0'`
 - `[return STD.STANDARD.BIT];`
 - `alias '1' is STD.STANDARD.'1'`
 - `[return STD.STANDARD.BIT];`
 - `alias "and" is STD.STANDARD."and"`
`[STD.STANDARD.BIT, STD.STANDARD.BIT`
`return STD.STANDARD.BIT];`
 - `alias "or" is STD.STANDARD."or"`
`[STD.STANDARD.BIT, STD.STANDARD.BIT`
`return STD.STANDARD.BIT];`
 - ...

기타

- Formal port 에 대응되는 actual로 expression도 허용.
- ISO 8-bit coded character set(ISO 8859-1:1987(E)) 으
로 확장.
- Basic identifier 에 extended identifier 를 추가 확장.

`\BUS\`, `\bus\` -- Two different identifiers, neither of
 -- which is the reserved word **bus**

`\a\|b\` -- An identifier containing three characters

`\74LS00\` -- An identifier

- Bit string literal 을 BIT type 의 array로부터 '0' 과
'1' 로만 이루어진 string literal로 변경.

```
type MVL is ('X', '0', '1', 'Z');
```

```
type MVL_VECTOR is array (NATURAL range <>)  
  of MVL;
```

```
constant c3: MVL_VECTOR := O"777";
```

```
  -- c3="11111111";
```

- Report statement 추가.
- Concatenation 결과의 왼쪽 bound는 결과의 base type 이 갖는 index subtype의 왼쪽 bound.
- Formal과 actual의 association에서 type의 변환을 위해 서 conversion function 외에 type conversion 사용 가능.
- Sequential statement 에도 label 표시 가능.
- STANDARD package 에 subtype DELAY_LENGTH 를 선언.

```

subtype DELAY_LENGTH is TIME
  range 0 fs to TIME' HIGH;
impure function NOW return DELAY_LENGTH;

```