VHDL을 이용한 효율적인 디지털 시스템 설계

인천대학교 공과대학 전자공학과

System ASIC Design Lab.

지도교수 : 조 중 휘

jcho@asiclab.inchon.ac.kr

목 차

Chap. 1 설계 방법론에 대한 이해

Chap. 2 기본 문법에 대한 이해

Chap. 3 Digital System의 VHDL Modeling 이해

Chap. 4 Test Bench Model의 이해

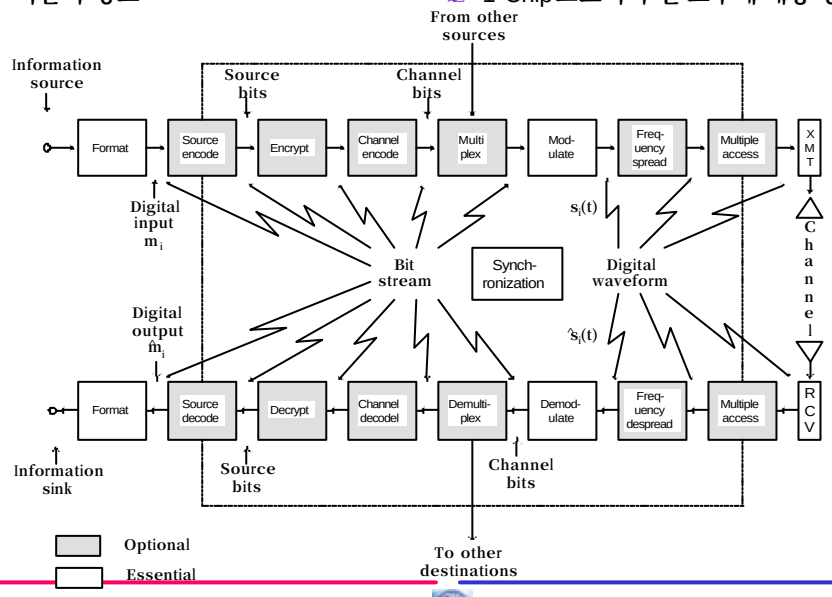
기초 실습

응용 실습

Chap. 1 설계 방법론에 대한 이해

ASIC 설계 방법론의 변화에 대한 필요성 (1)

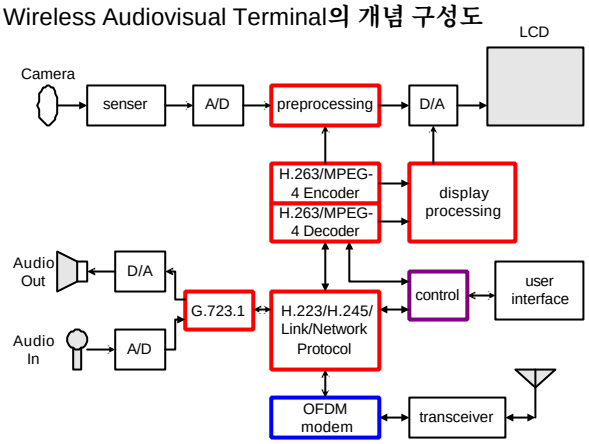
✍ 기본 구성도



✍ 1-Chip으로의 구현 요구에 대응 방안

ASIC 설계에 대한 방법론의 변화에 대한 필요성 (2)

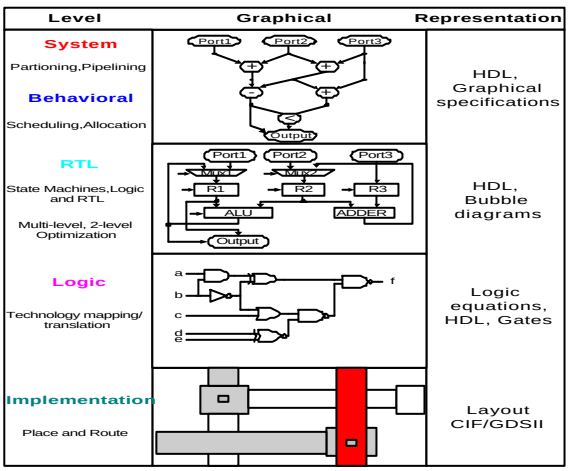
- Wireless Audiovisual Terminal의 개념 구성도



- 주요 구성부
  - H.324 CODEC : Video 및 Audio Data에 대한 압축과 복원의 기능
  - Digital Communication Module : Data 변조, 복조 및 Error 정정을 위한 기능
  - Micro-Controller 및 기타 : 16-Bit 이상의 Processor와 Camera, LCD Interface Chips
  - Analog Front End Module :고속 A/D, D/A 변환기 및 고주파 부품

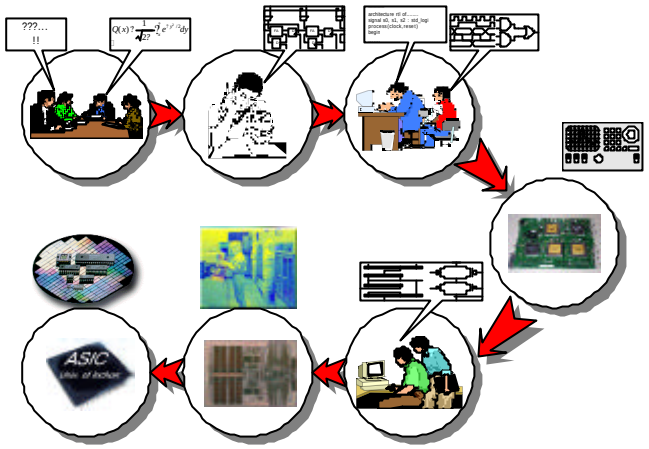
설계 대상체의 동작 및 구조 특성에 대한 표현의 다양성

- 설계 사양에 대한 표현 수준
  - System Level
  - Behavioral Level
  - RTL Level
  - Logic Level
  - Circuit Level
  - Layout Level



- 설계 사양에 대한 표현 방법
  - Graphic Diagram을 이용한 표현
  - 하드웨어 기술 언어 (HDL : Hardware Description Language)에 의한 표현
- 설계 단계에 대한 이해

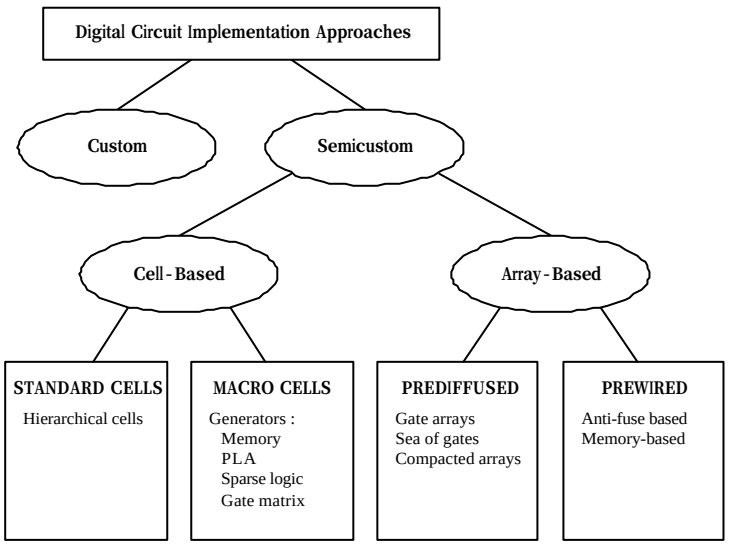
ASIC 구현의 여러 단계에 대한 개념 구성도



- 시스템 이해와 알고리즘 설계
- Behavioral 및 Register Transfer Level의 블록 구성도 설계
- VHDL을 이용한 표현, 검증 및 논리 회로도 합성
- FPGA를 이용한 설계와 검증을 통한 VHDL 표현에 대한 정확성 검증
- Physical 설계 및 ASIC으로의 제작
- System Board 구성을 통한 ASIC에 대한 동작과 성능의 검증

다양한 Digital ASIC 구현 방법

- Digital ASIC 구현의 종류



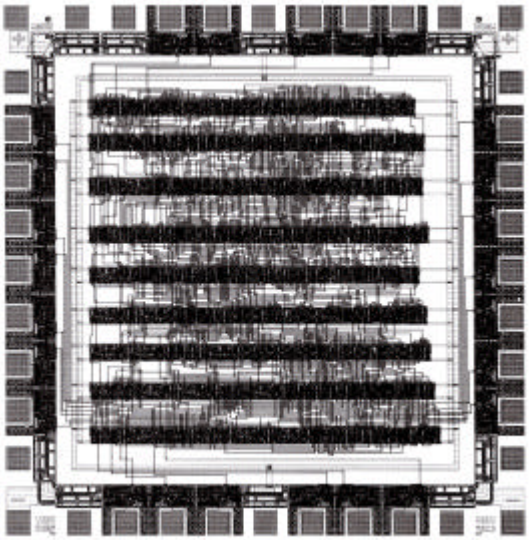
Digital ASIC 구현 방법에 대한 선택

✍ 응용과 시장성을 고려한 ASIC 제품의 특성 비교

	Custom	*Cell-based*	*Prediffused*	*Prewired*
Density	Very High	High	High	Medium - Low
Performance	Very High	High	High	Medium - Low
Flexibility	Very High	High	Medium	Low
Design time	Very Long	Short	Short	Very Short
Manufacturing time	Medium	Medium	Short	Very Short
Cost - low volume	Very High	High	High	Low
Cost - high volume	Low	Low	Low	High

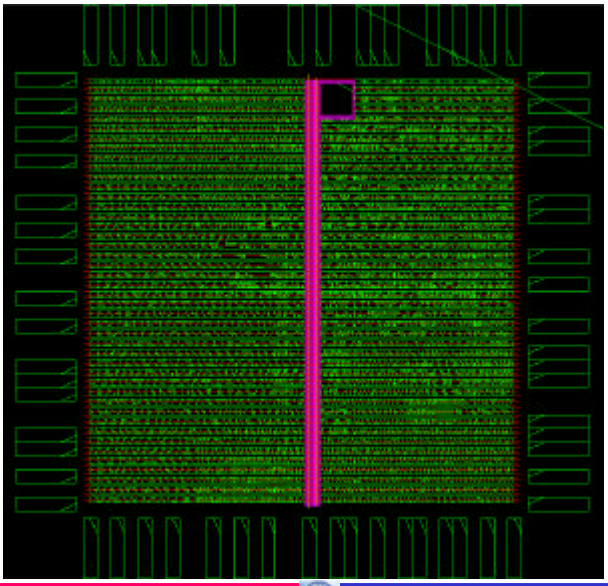
Standard-Cell Design에 대한 개념

✍ Boding Diagram 예제



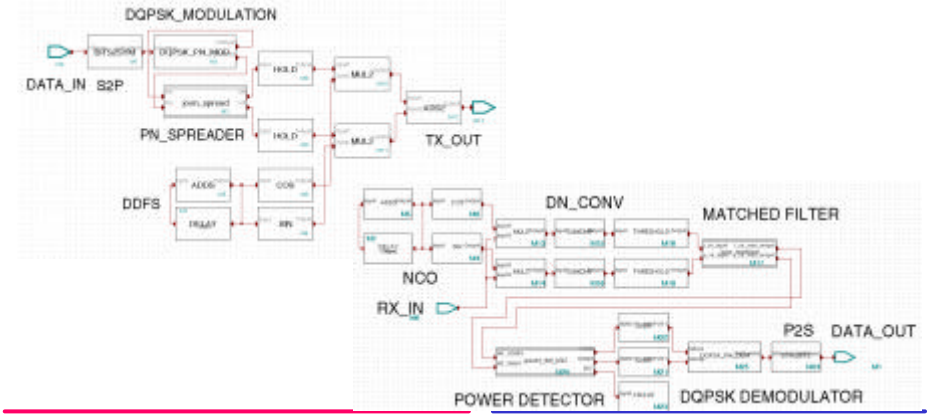
Sea-of-Gate Design에 대한 개념

- Boding Diagram 예제: FLEX Decoder Chip



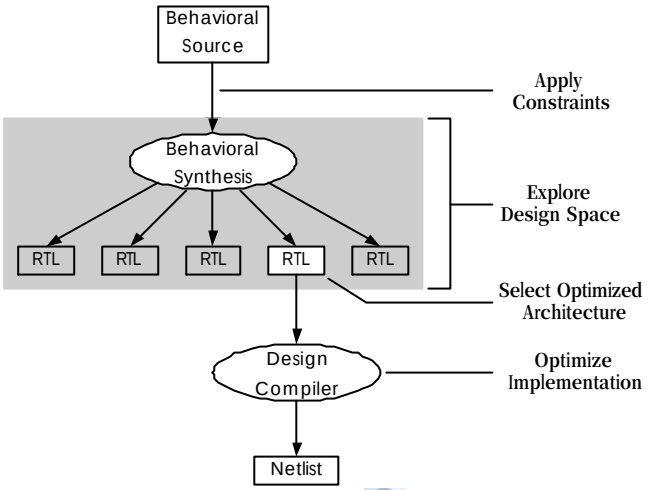
설계 단계-1 : System(Algorithm) Level에서의 설계

- C, MATLAB, SPW 및 COSSAP등을 이용한 System Level에서 설계와 검증
  - 제공되는 library를 이용하여 필요한 각 sub-block에 대한 선택
  - system의 성능에 대한 관찰 (예, BER에 대한 측정)
  - hardware의 complexity와 performance를 고려한 data-bus의 bit size 결정
- System 설계자와 Chip 설계자와의 충분한 교감이 요구됨
- COSSAP을 이용한 DS/SS Base Band MODEM에 대한 Modeling 예제



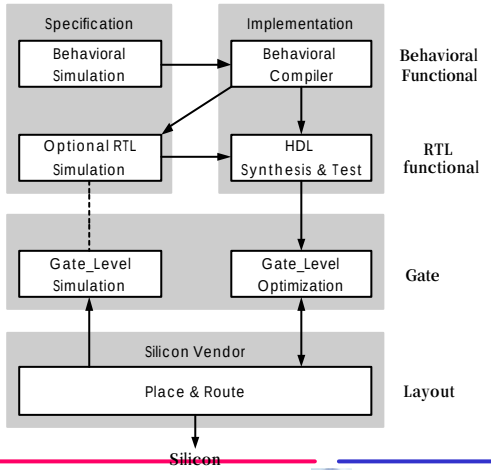
설계 단계-2 : Behavioral Compiler의 개념

- System Level 구성에서 Data Path Module에 대한 설계
  - Behavioral Compiler에 의하여 설계 Option에 따라 최적화된 RTL 회로도 생성
  - 여러 가지의 RTL 회로도 중에서 최적화된 것을 선택
    - 설계자 관점에서 우수하다고 판단한 RTL 회로도 생성



설계 단계-2 : Behavioral Compiler Design Flow

- Behavioral Compiler에 의한 Design Flow의 구성 이해
  - 동작의 구성 이해
    - Operator to State Binding
    - Variable to Register Binding
    - Function Unit Sharing 및 Control Logic에 대한 FSM 생성

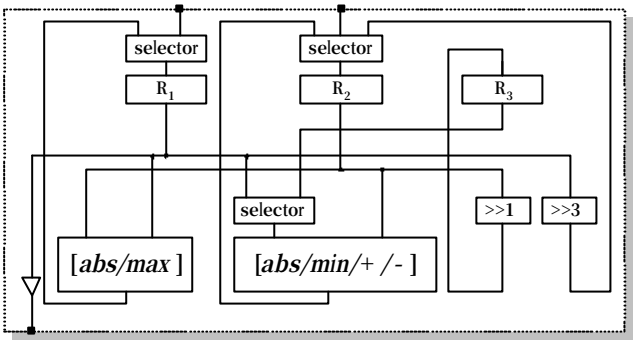


설계 단계-2 : Behavioral Compiler에 의한 RTL 회로도

- 다음의 예제에 대하여 생각하여 보자.
  - 조합논리 회로로의 구성
  - 동기 신호가 있는 시스템으로 구성

$$\sqrt{a^2 ? b^2} ? \max((0.875x ? 0.5y), x)$$
$$x ? \max((|a|,|b|)$$
$$y ? \min((|a|,|b|)$$

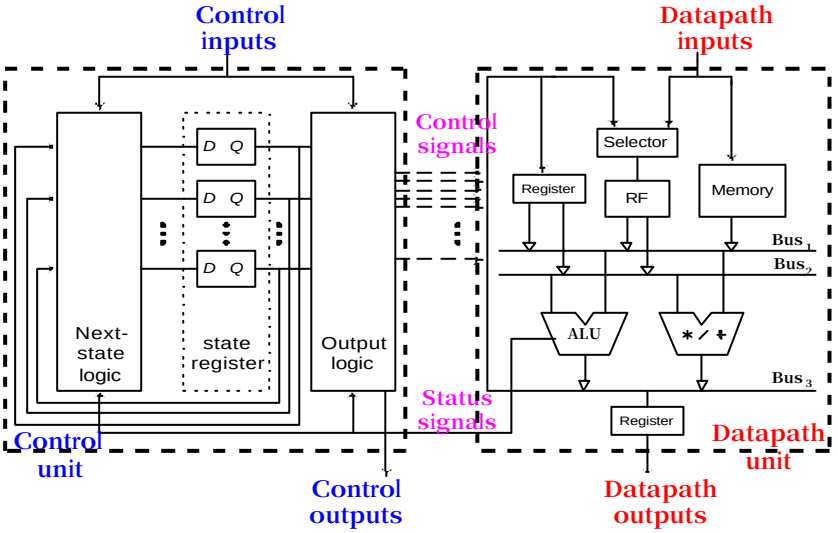
- Data Path Module에 대한 RTL 회로도



- Controller Module에 대한 RTL 회로도
  - FSM Table Model로 출력

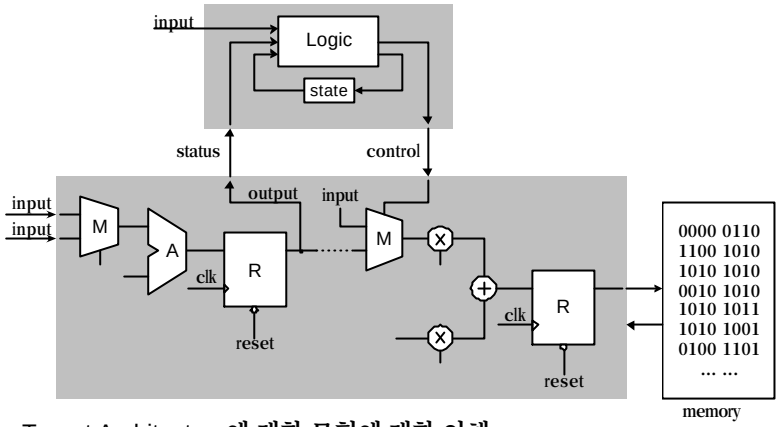
설계 단계-3 : Register-Transfer Level 회로도의 설계

- RTL 회로도 설계에 대한 Target Architecture



설계 단계-3 : Register-Transfer Level 회로도의 설계

- Synthesis 가능한 VHDL 표현을 위한 Hardware 개념도

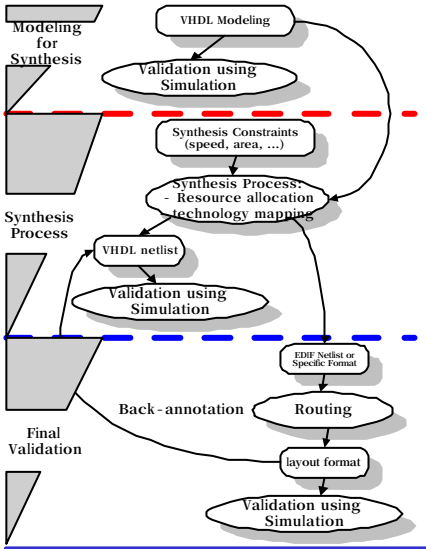


- Target Architecture에 대한 모형에 대한 이해
  - 조합 논리 회로 블록 : Basic Gates, Modular Units (mux, decoder, arithmetic units)
  - 동기 신호 요구 블록 : F/Fs, Registers, Counters, FSM(Finite State Machine)
  - 기억 소자 블록 : ROM, RAM
  - Bus 연결 요구 블록 : 3-Buffer

설계 단계-3 : Design Compiler에 의한 설계

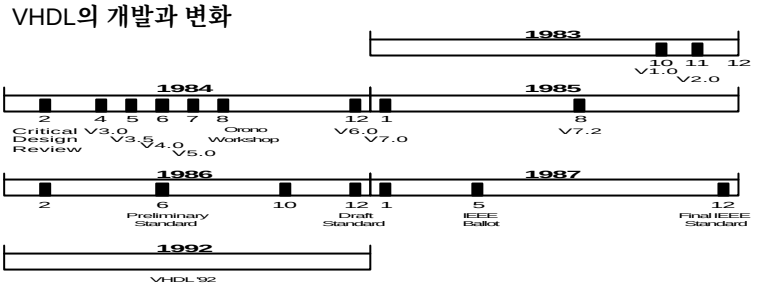
- Design Flow의 3단계 구성과 의미
  - RTL 회로 설계
    - 회로도 추출 및 효율적 VHDL 표현
    - 기능 Simulation에 의한 회로 검증
  - Gate Level 회로 설계
    - 면적,속도,Power를 고려한 조건 설정
    - Design Library의 설정 : FPGA, ASIC
    - Component Delay 를 고려한 기능 및 Timing Simulation에 의한 회로 검증
    - Physical Design을 위한 EDIF 생성
  - FPGA를 이용한 Physical Design
    - FPGA Tool에 의한 Compile 수행
    - 사용할 Device 및 Option의 설정
    - Layout Data를 이용한 Simulation
    - Target Board의 구성과 Emulation을 통한 검증
    - ASIC으로의 변환 여부에 따른 대응
  - ASIC 설계를 위한 Physical Design
    - Design House와의 연계 및 EDIF 전달
    - Wire Delay를 함께 고려한 Simulation
    - Sign-off 와 ASIC에 대한 기다림
    - Test Board 구성, 검증 및 평가

- Design Flow의 모형



HDL의 비교 및 VHDL의 변화 과정

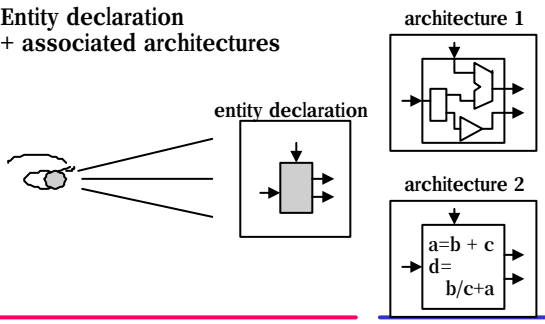
- 표준화를 목표로 하면서 사용되고 있는 HDL 의 비교
  - VHDL : 시스템에 대한 기술의 능력은 매우 높으나, synthesis를 위하여 sub-set VHDL과 modeling에 대한 guide가 요구.
  - Verilog : VHDL보다는 기술의 능력이 높지는 않으나 널리 사용되고 있으며, gate 레벨의 simulation 언어로는 매우 적합하여 design kit의 golden simulator에 사용
- VHDL의 개발과 변화



- Value 체계에 대한 변화
  - 1987년의 STD-1076 Version
    - STD library에 정의된 2-Value 체계 즉, {'0', '1'}를 기본으로 사용
  - 1990년을 전후로 하여 Value 체계에 대한 표준 안에 대한 요구
    - IEEE library에 정의된 9-value 체계 즉, {'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'}를 사용

Design 대상체의 표현을 위하여 요구되는 개념

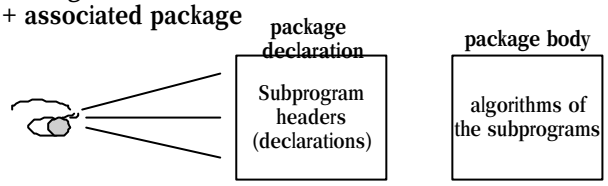
- Design Entity에 대한 정의와 예
  - 표현하고자 하는 Hardware 대상체를 의미함
  - 예시: 규모에 관계없이
    - NOT gate, ALU, 8051 Micro-Controller, DSP Board, Digital TV System등
- VHDL에서 Design Entity를 표현하기 위하여 요구되는 개념
  - 입력 및 출력 신호에 대한 표현: entity declaration design unit
  - 출력 신호에 대한 결정 과정에 대한 표현: architecture body design unit
  - entity unit과 architecture unit에 대한 결합의 표현: configuration design unit
  - 예시 모형



공유 정보의 표현을 위하여 요구되는 개념

- 공유 정보의 표현 개념과 예
  - Design Entity 사이에서 공통적으로 사용될 수 있는 내용을 표현하기 위하여 정의
  - 예시
    - Subprogram, 공통적으로 사용되는 상수 값, Component 등
- VHDL에서 공유 정보에 대한 표현을 위하여 요구되는 개념
  - 공유 정보에 대한 Interface 표현
    - Package Declaration Design Unit
  - 공유 정보 중에서 Sub-Program에 대한 Algorithm을 표현
    - Package Body Design Unit
  - 예시 모형

Package declaration + associated package

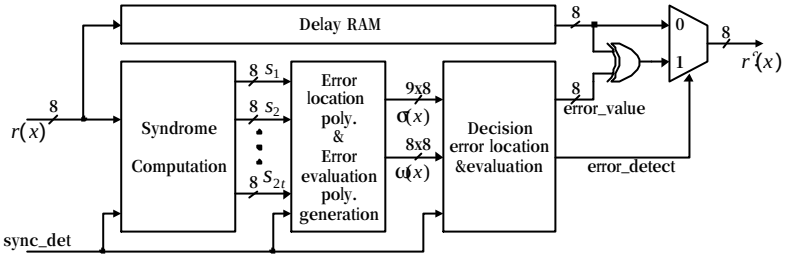


VHDL 표현에 요구되는 5가지 Design Unit의 개념

- Hardware 관점에서의 표현을 위한 3가지 Unit
  - entity declaration unit
    - 표현 대상체의 interface에 관한 정보를 기술
  - architecture body unit
    - 표현 대상체의 내부 동작 또는 구조를 기술
  - configuration declaration unit
    - entity unit과 architecture unit을 지정하는 정보 및 기타 결합 정보를 기술
    - simulation을 행할 때 요구됨
- Software 관점에서의 표현을 위한 2가지 Unit
  - package declaration unit
    - 서로 다른 design entity에서 공유할 정보를 선언하는 경우 에 사용
  - package body unit
    - 공유하는 정보 중에서 subprogram에 대한 동작을 기술하는 경우에 사용
- 최소로 요구되는 2가지 Unit과 의미
  - entity unit에 대한 표현
    - 9-Value 체계를 정의한 Package에 대한 Visibility의 부여
    - I/O 정보에 대한 표현
  - architecture unit에 대한 표현
    - 내부 알고리즘의 동작 특성을 표현하는 방식: Behavioral View Model
    - 구성 요소들의 연결 구조 특성을 표현하는 방식: Structural View Model

효율적 설계를 위한 System 분할

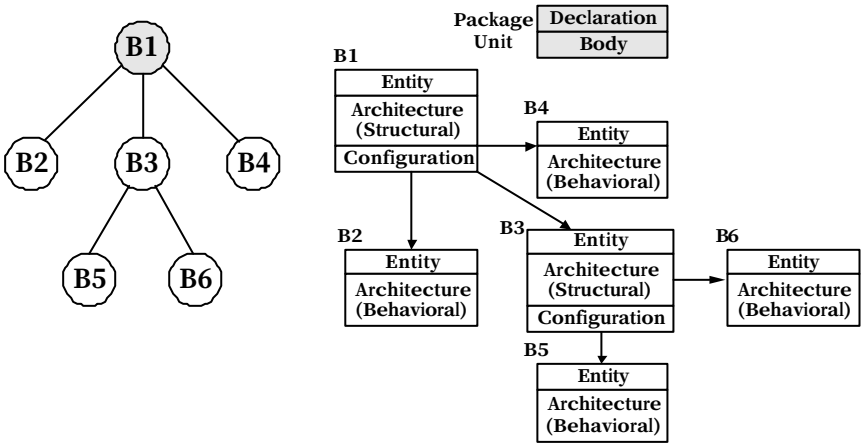
- 분할을 위한 기본 개념
  - System 개념 또는 설계자가 이해할 수 있는 범위에 따른 분할함
  - IP 또는 Macro Component로 구성되어 있는 경우의 분할함
  - Critical Path를 포함하는 Block과 이외의 Block은 분할함
    - 설계 제약 조건을 별도로 설정함
  - 논리 합성의 대상체는 4,000~5,000 Gate가 Optimization 관점에서 보편적으로 적합함
- 분할에 대한 예
  - (n, k, d) Reed-Solomon Decoder



- Design Tree
  - 분할 정보의 계층성을 표현하기 위한 Model

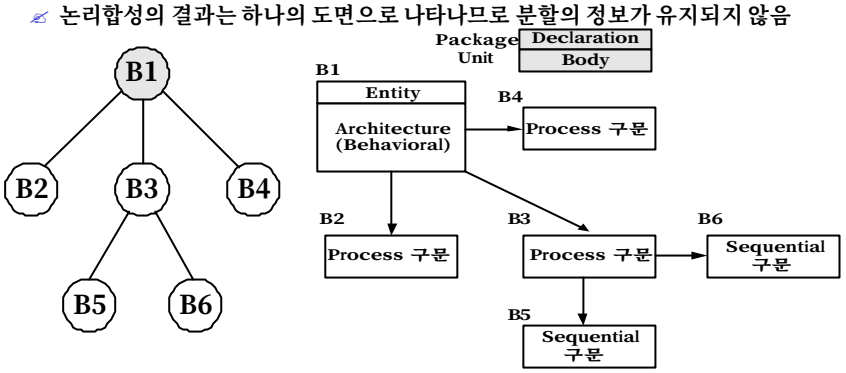
분할 정보와 VHDL 표현 개념 Model (1)

- Design Tree와 Fully 계층적 VHDL 표현
  - 모든 Design Node를 Entity Unit + Architecture Unit으로 표현
  - 논리 합성의 결과는 계층적으로 나타나므로 분할의 정보를 유지함



분할 정보와 VHDL 표현 개념 Model (2)

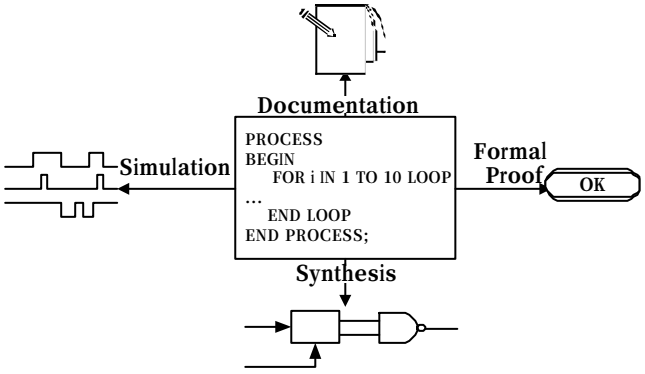
- Design Tree와 1-Level VHDL 표현
  - Root Design Node에 대하여 만 Entity Unit + Architecture Unit으로 표현
  - Root 이외의 Design Node는 Concurrent VHDL 구문을 사용하여 표현
  - 논리합성의 결과는 하나의 도면으로 나타나므로 분할의 정보가 유지되지 않음



- 여러 가지 다양한 표현이 가능함: 설계자의 선택 사항
  - 예시 (B3 Block이 Critical Path을 나타내는 것으로 가정)
    - B3에 대하여 Behavioral View표현 Model이용

VHDL의 다양한 사용 목적

- 사용 관점에 따른 목적의 다양성
  - System 설계자의 관점 : Simulation과 Documentation
  - Chip 설계자의 관점 : Simulation, Synthesis 및 Documentation
- Chip 설계자에의 유용성 및 제약성
  - IP(Intellectual Property) Model에 대한 개발과 Re-Use의 개념이 가능
  - Synthesis가 가능한 VHDL 표현과 효율적인 Modeling 방법의 이해가 요구



Full Adder에 대한 2가지 형태의 VHDL 표현

✍ 1-Level VHDL 표현

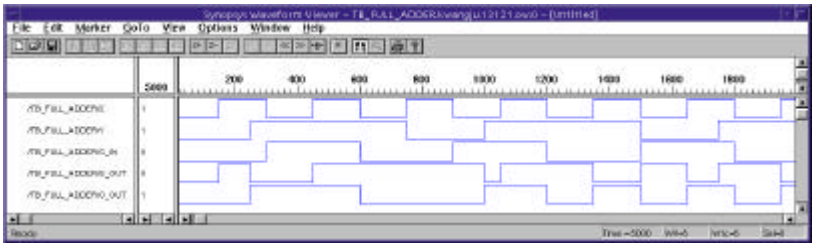
```
library IEEE;
use IEEE.std_logic_1164.all;
entity FULL_ADD is
  port (
    A          : in std_logic;
    B          : in std_logic;
    C_IN       : in std_logic;
    C_OUT      : out std_logic;
    SUM        : out std_logic);
end FULL_ADD;
architecture RTL1 of FULL_ADD is
  signal S0 : std_logic;
begin
  S0 <= A xor B;
  SUM <= S0 xor C_IN;
  C_OUT <= (A and B) or (S0 and C_IN);
end RTL1;
```

✍ 계층성을 고려한 VHDL 표현

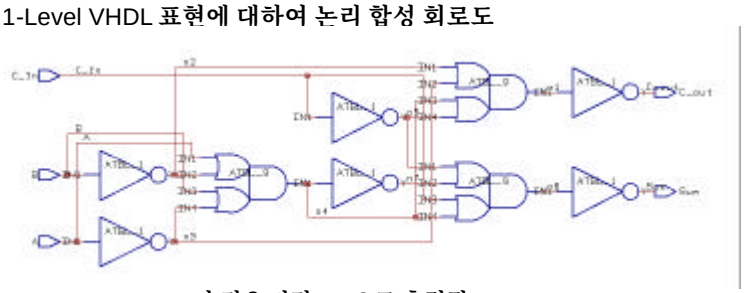
```
library IEEE; use IEEE.std_logic_1164.all;
entity FULL_ADD is
  port ( A, B, C_IN    : in std_logic;
         SUM, C_OUT : out std_logic);
end FULL_ADD;
architecture RTL2 of FULL_ADD is
  signal S0,S1,S2 : std_logic;
  for u0,u1 : HA use entity work.HA(RTL);
  for u2 : OR2 use entity work.OR2(RTL);
  component HA
  port ( A, B : in std_logic;
         SUM, CARRY : out std_logic);
  end component;
  component OR2
  port ( IN1, IN2 : in std_logic;
         OUT1 : out std_logic);
  end component;
begin
  u0 : HA port map (X, Y, S0, S1);
  u1 : HA port map (S0, C_IN, SUM, S2);
  u2 : OR2 port map (S1, S2, C_OUT);
end RTL2;
```


Simulation과 Synthesis 결과에 대한 비교 (1)

✍ Simulation 결과에 대한 Waveform



✍ 1-Level VHDL 표현에 대하여 논리 합성 회로도

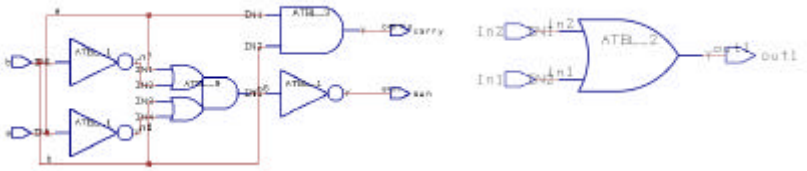


✍ LSI-10K Library의 경우 면적: 3.0으로 출력됨

Simulation과 Synthesis 결과에 대한 비교(2)

- 계층성을 갖는 VHDL 표현에 대한 논리 합성 회로도
  - Structural View의 상위 Node에 대한 논리 회로도

  - Behavioral View의 최하위 Node에 대한 논리 회로도 (HA: Half Adder, OR2: 2-input OR)



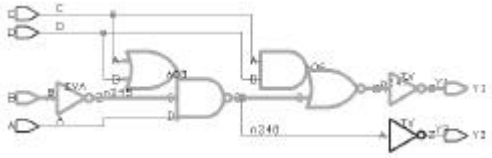
  - LSI-10K Library의 경우 면적: 1.5*2 + 0.5 = 3.5로 출력됨
- 합성 결과에 대한 차이점 분석과 이해
  - Optimization의 과정에 따른 차이 발생

설계 제약 조건에 따른 차이점 이해를 위한 VHDL 예제

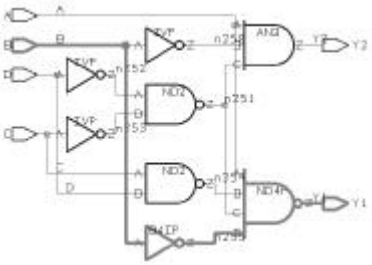
```
library IEEE;
use IEEE.std_logic_1164.all;
entity EX is
  port (A, B, C, D : in  std_logic;
        Y1, Y2  : out std_logic);
end EX;
architecture RTL of EX is
begin
  process (A, B, C, D)
    variable TMP:
       std_logic_vector(3 downto 0);
  begin
    TMP := A & B & C & D;
    case TMP is
      when "0000" =>  Y1<='1'; Y2<='0';
      when "0001" =>  Y1<='1'; Y2<='0';
      when "0010" =>  Y1<='1'; Y2<='0';
      when "0011" =>  Y1<='1'; Y2<='0';
      when "0100" =>  Y1<='1'; Y2<='0';
      when "0101" =>  Y1<='1'; Y2<='0';
      when "0110" =>  Y1<='1'; Y2<='0';
      when "0111" =>  Y1<='1'; Y2<='0';
      when "1000" =>  Y1<='1'; Y2<='0';
      when "1001" =>  Y1<='0'; Y2<='1';
      when "1010" =>  Y1<='0'; Y2<='1';
      when "1011" =>  Y1<='1'; Y2<='1';
      when "1100" =>  Y1<='1'; Y2<='0';
      when "1101" =>  Y1<='1'; Y2<='0';
      when "1110" =>  Y1<='1'; Y2<='0';
      when others =>  Y1<='1'; Y2<='0';
    end case;
  end process;
end RTL;
```

설계 제약 조건에 따른 결과 차이에 대한 이해

- Target Library : LSI_10K
  - Area를 최소로 하고자 하는 Area Design Constraint을 부과하고 합성한 경우
    - 5-cells(Area : 7.0), Critical Path(2.38ns)

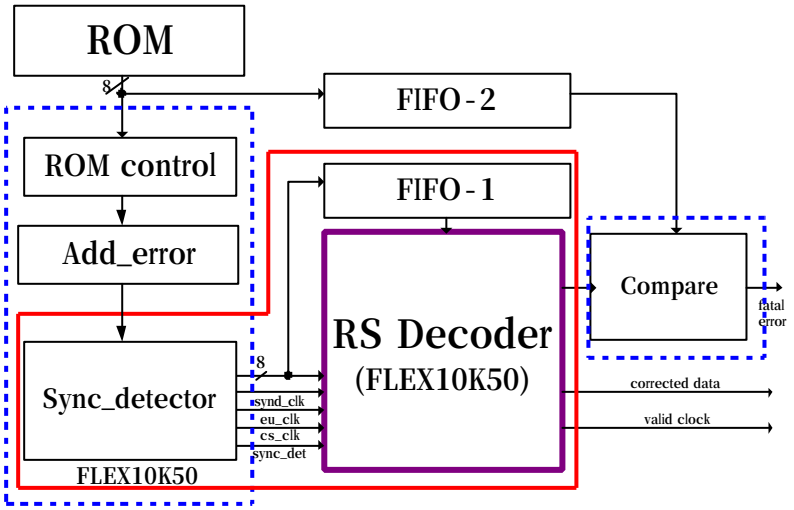


  - 동작 시간을 최소로 하고자 하는 Timing Design Constraint를 부과하고 합성한 경우
    - 8-cells(Area : 15.0), Critical Path(0.84ns)



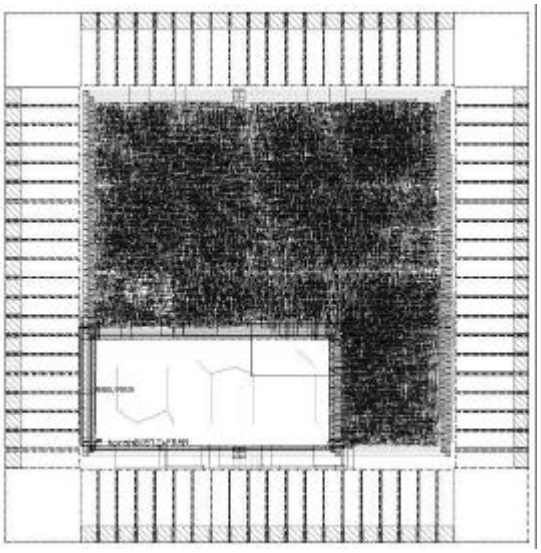
FPGA를 이용한 Fast Prototyping

- FPGA와 주변 회로를 이용한 검증용 Board 구성
  - Reed-Solomon Decoder



ASIC 구현과 검증

- Reed-Solomon Decoder에 대한 Bonding Diagram



Chap. 2 VHDL 기본 문법에 대한 이해

Character Set , Lexical Elements 및 Identifier

- Character Sets
  - Upper-Case Letters
  - Lower-Case Letters
  - Special Characters
  - Space Character
  - Format Effectors
- Lexical Elements
  - Delimiter
    - Single Character
    - Compound Character
  - Identifier
    - Reserved Identifier
    - 사용자가 정의하는 Identifier
  - Literal

- VHDL-87에 정의된 Reserved Word

abs	access	after	alias	all
and	architecture		array	
assert	attribute	begin	block	body
buffer	bus	case	component	
configuration	constant		disconnect	
downto	else	elsif	end	
entity	exit	file	for	
function	generate	generic	guarded	if
in	inout	is	label	
library	linkage	loop	map	mod
nand	new	next	nor	not
null	of	on	open	or
others	out	package	port	
procedure		process	range	
record	register	rem	report	
return	select	severity	signal	
subtype	then	to	transport	type
units	until	use	variable	wait
when	while	with	xor	

VHDL-93 추가 Reserved Word와 사용자 정의 Identifier

- VHDL-93에 추가된 Reserved Word

group	impure	inertial	literal	postponed	pure	reject	rol	ror
shared	sla	sll	sra	srl	unaffected		xnor	

- 사용자가 정의하는 Identifier의 규약과 예제
  - 규약
    - 첫 문자는 Alphabet이어야 한다.
    - 마지막 문자는 Alphabet 또는 숫자이다.
    - 중간은 Alphabet, 숫자 또는 _ 이어야 한다.
    - 단, _는 연속적으로 사용할 수 없다.
    - Alphabet에 대하여 소문자와 대문자에 대한 구분은 없다.
  - 올바른 예제

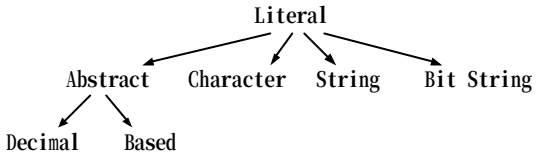
COUNT	cout	c_out	AB2_5C	VHSIC	X1	FFT
Decoder	A_B_C	xyZ	h333	STORE_NEXT_ITEM		

  - 틀린 예제: 의미를 생각하여 보자.

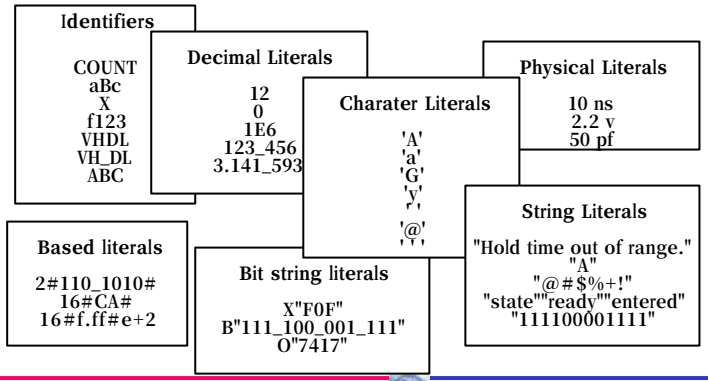
2CA	My-name	H$B	LOOP	Decode_	N#3

VHDL Literal에 대한 정의

- Literal에 대한 정의 Tree



- 예제



기억 장소의 가변성 속성을 정의하는 Class에 대한 정의

- 기억 장소의 선언을 위한 Syntax 정의
  - Data_Class Object_Identifier :
    Data_Type [:= initial value] ;
  - 예제

```
signal A: std_logic;
constant B: std_logic_vector(3 downto 0) :=
    "0001";
```

- 기억 장소의 속성을 정의하는 개념
  - Data Class
- Constant
  - Static Data를 기억하기 위한 기억 장소에 대한 object 임
  - 기억 장소의 값은 선언과 함께 정의됨
  - H/W적인 의미와 이해
- Signal
  - 선언과 함께 초기값을 지정할 수 있음
  - Simulation 도중에 변경 가능한 Dynamic Data를 기억하는 기억 장소임
    - 기억 장소 값에 대한 변화 요인이 발생하고 일정 시간 후 에 변경됨
  - H/W적인 의미와 이해
- Variable
  - 선언과 함께 초기값을 가질 수 있음
  - Simulation 도중에 변경 가능한 Dynamic Data를 기억하는 기억 장소임
    - 기억 장소 값에 대한 변화 요인이 발생하면서 동시에 변경됨
  - H/W적인 의미와 이해
- File
  - Simulation Vector 또는 결과를 File로 처리하기 위하여 정의된 기억 장소임
  - H/W로 구현되지 않는 기억 장소임
- 선언 영역의 차이
  - Constant에 대한 선언부
    - entity,architecture,process
    - package,function,procedure
    - block
  - Signal에 대한 선언부
    - entity, architecture, package,
    - block
  - Variable에 대한 선언부
    - process, function, procedure

기억 장소에 저장되는 Data 값의 종류에 대한 정의

- Data 값 종류의 분류를 위하여 정의된 개념 : Data Type
  - 표준화 Group에서 정의하고, Package 구문에 기술되어 있는 Data Type
  - VHDL 사용자가 편의에 따라 정의하고 사용하는 Data Type
- 의미에 따른 분류 : ++의 Type은 Synthesis Tool에 의하여 지원되지 않음
  - Scalar Data Types
    - Enumeration Type, Numeric type, Physical Type++
  - Composite Data Types
    - Array Type, Record Type
  - Access Data Type++
  - File Data Type++
- Type 정의를 위한 package 구문
  - standard package에 정의된 사항
    - 1987년에 정의된 사항
  - std_logic_1164에 정의된 사항
    - 1993년에 정의된 사항

- Standard Package에 정의된 사항

```
package STANDARD is
  type BOOLEAN is (FALSE, TRUE);
  type BIT is ('0', '1');
  type CHARACTER is ( 생략 );
  type SEVERITY_LEVEL is (NOTE, WARNING,
     ERROR, FAILURE);
  type INTEGER is implementation_defined ;
  type REAL is implementation_defined ;
  type TIME is range implementation_defined ;
    units
      fs;
      ps       = 1000 fs;
      중간 생략
    end units;
  subtype DELAY_LENGTH is TIME;
  function NOW return time;
  subtype NATURAL is integer range 0 to
     integer'high;
  subtype POSITIVE is integer range 1 to
     integer'high;
  type STRING is array(positive range <>) of
     character;
  type BIT_VECTOR is array(natural range <>) of bit;
end STANDARD;
```

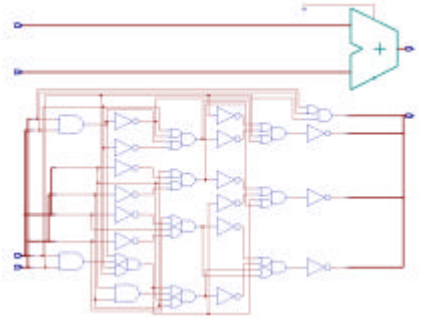

1993년에 정의된 Package와 Type 정의 예제

- Std_logic_1164 Package

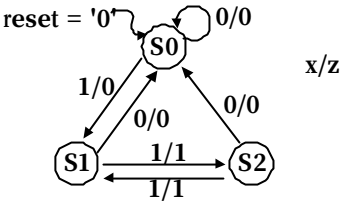
```
package STD_LOGIC_1164 is
  type STD_ULOGIC is (
     'U',    --Uninitialized
     'X',    --Forcing Unknown
     '0',    --Forcing 0
     '1',    --Forcing 1
     'Z',    --High Impedance
     'W',    --Weak Unknown
     'L',    --Weak 0
     'H',    --Weak 1
     '-'     --Don't care
    );
  type STD_ULOGIC_VECTOR is
     array(natural range <>) of std_ulogic;
  function resolved(s: std_ulogic_vector)
     return std_ulogic;
  subtype STD_ULOGIC is resolved
     std_ulogic;
  type STD_LOGIC_VECTOR is array(natural
     range <>) of std_logic;
  subtype X01 is resolved std_ulogic range 'X'
     to '1';
  subtype X01Z is resolved std_ulogic range
     'X' to 'Z';
  subtype UX01 is resolved std_ulogic range
     'U to '1';
  subtype UX01Z is resolved std_ulogic range
     'U' to 'Z';
  function "and" (l: std_logic; r: std_logic)
     return std_logic;
  중간생략;
  function "and" (l: std_logic_vector;
    r: std_logic_vector) return std_logic_vector;
  중간생략;


  중간생략
  function Is_X (s: std_ulogic) return boolean;
end STD_LOGIC_1164;
```

사용자의 정의가 요구되는 Data Type 예 (1)

- Integer Data Type
  - type COUNTVALUE is range 0 to 15;
  - type TWENTIES is range 20 to 29;
  - Synthesis tool에 의하여 요구되는 bit 수는 0부터 시작하여 숫자의 최대값까지임
  - 예제와 합성 차이에 대한 이해
    - Integer로 선언된 자료의 덧셈
    - 0 ~ 15의 범위로 선언된 자료의 덧셈



- Enumeration Data Type
  - 제어부 설계 사양 표현에 따른 FSM 표현에 이용됨
  - type STATE_NAME is (A, B, C, D);
  - 2진수 값이 가장 왼쪽(0이 할당)부터 오른쪽 element 순서로 1씩 증가되어 할당됨
  - 예제



  - type STATE is (S0, S1, S2);
  - signal C_STATE, N_STATE : STATE;

사용자의 정의가 요구되는 Data Type 예 (2)

- 8*5 Bit의 2차원 Memory 선언: 경우-1
  - Word Addressing Mode가 요구되는 Memory 표현에 사용
  - 예제-1

```
subtype WORD1 is
   std_logic_vector(4 downto 0);
type MEMORY1 is array(0 to 7) of
    WORD;
```

  - 예제-2

```
type WORD2 is array(4 downto 0) of
   std_logic;
type MEMORY2 is array(0 to 7) of
   WORD;
```

  - 예제-3

```
type MEMORY3 is array(0 to 7) of
   std_logic_vector(4 downto 0);
```

- 8*5 Bit의 2차원 Memory 선언: 경우-2
  - Bit Addressing Mode에 사용

```
type MEMORY4 is
  array(0 to 7, 4 downto 0) of std_logic;
```

  - 구조 이해
  - Bit-addressing Mode가 요구되는 2차원 Memory 표현에 사용

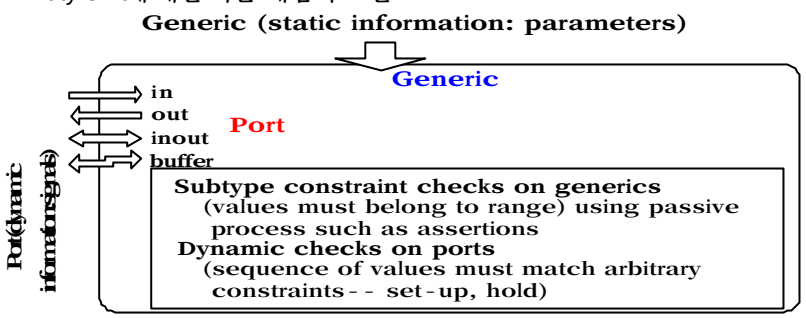
- Composite Record Data Type 예제

```
type FloatPointType is
  record
     SIGN : std_logic;
     EXPONENT : unsigned(0 to 6);
     FRACTION : unsigned(24 downto 1);
end record;
```

- Composite Array Data Type
  - type unsigned is array(natural range <>) of std_logic;
  - type unsigned_word is array(natural range 7 downto 0) of std_logic;

Entity Declaration Unit의 정의

- Entity Unit에 대한 기본 개념의 표현

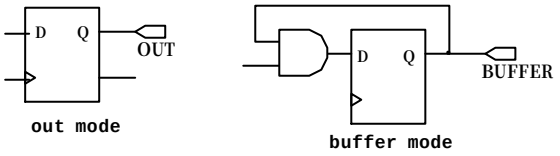


- Syntax 정의

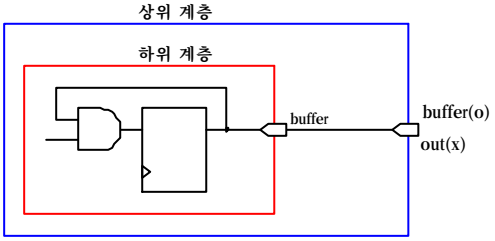
```
entity ENTITY_NAME is
    [ generic ( LIST_OF_GENERICS_AND_THEIR_TYPES ) ; ]
    [ port ( LIST_OF_PORTS_AND_THEIR_MODE ) ; ]
    [ DECLARATIONS ]
  [ begin
    { ENTITY_STATEMENT } ]
end [ ENTITY_NAME ] ;
```


Out Mode와 Buffer Mode에 대한 이해

- 회로에 따른 Out Mode와 Buffer Mode의 정의와 이해



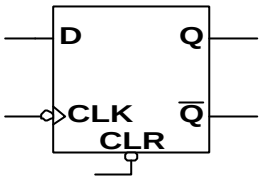
- Buffer Mode의 사용에 따른 주의 사항



  - 하위 계층에서 Buffer Mode로 출력되는 경우에는,
  - 상위 계층에서도 Buffer Mode로 접속 되어야 한다.
- Buffer Mode에 대한 사용 제약
  - Buffer Mode를 Out Mode로 표현하기 위한 방법의 이해

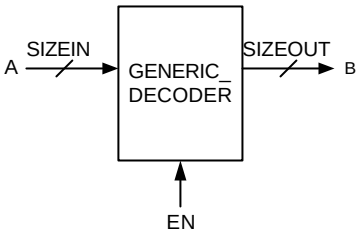
Entity Declaration Unit의 예제

- D Flip-Flop과 Entity Unit




```
-- 9-value Package 공유를 위한 Visibility
library IEEE; use IEEE.std_logic_1164.ALL;
-- Pin Name, Mode, Data Type의 표현
entity DFF is
  port (D : in std_logic;
        CLK : in std_logic;
        CLR : in std_logic;
        -- D, CLK, CLR : in std_logic;
        Q, QBAR : out std_logic);
end DFF;
```

- Generic Decoder와 Entity Unit



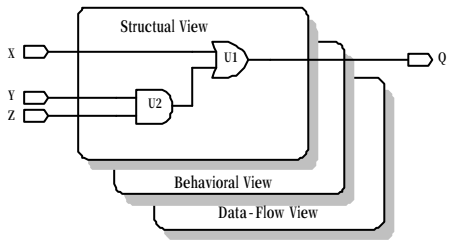

```
library IEEE; use IEEE.std_logic_1164.ALL;
entity GENERIC_DECODER is
 -- Static Information에 대한 표현
  generic (SIZEIN, SIZEOUT : integer);
 -- Dynamic Pin Data에 대한 표현
  port (EN: in std_logic;
     A: in std_logic_vector(SIZEIN-1
        downto 0);
     B : out std_logic_vector(SIZEOUT-1
        downto 0));
end GENERIC_DECODER;
```


Architecture Body Unit에 대한 개념과 표현 형태

- Syntax 정의

```
architecture A_NAME of  E_NAME is
  [ DECLARATIONS ]
begin
  {CONCURRENT_STATEMENT}
end [A_NAME] ;
```

- 예제 회로



- Behavioral view
  - Q='1'이기 위한 조건
  - x='1' 이거나, Y와 Z가 동시에 '1'인 경우라 분석
  - 동작적 특성의 구문을 이용하여 기술
- Data-flow view
  - Boolean 또는 state equation 의 형태로 분석
  - 함수적 특성의 구문을 이용하여 기술
- Structural view
  - and gate와 or gate의 연결에 의한 구성으로 분석
  - 구조적 특성의 구문을 이용하여 기술
  - and 및 or gate는 Behavioral View 또는 Data-flow View 로 표현됨
- Mixed view
  - 일부 특성은 Behavioral View 또는 Data-flow View 로 표현됨
  - 일부 특성은 Structural View 로 표현됨
  - 가능하면 사용하지 않는다.

Architecture Body Unit에 대한 다양한 표현 형태

- Entity design unit

```
library IEEE; use IEEE.std_logic_1164.all;
entity EXAMPLE is
  port (X, Y, Z: in std_logic;
        Q: out std_logic);
end EXAMPLE;
```

- Behavioral View에 의한 표현

```
architecture RTL1 of EXAMPLE is
begin
  process (X, Y, Z)
  begin
    if X='1' then
      Q <= '1';
    elsif (Y='1' and Z='1') then
      Q <= '1';
    else
      Q <= '0';
    end if;
  end process;
end RTL1;
```

- Data-flow View에 의한 표현

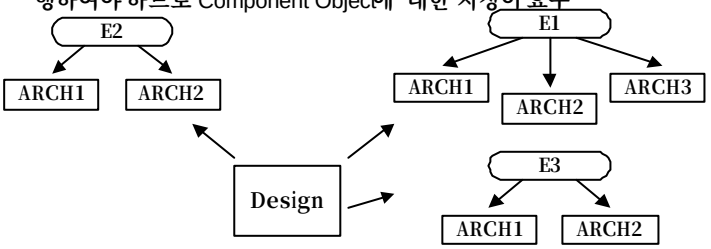
```
architecture RTL2 of EXAMPLE is
begin
  Q <= X or (Y AND Z);
end RTL2;
```

- Structural View에 의한 표현

```
architecture RTL3 of EXAMPLE is
  component OR2
    port (I1, I2: in std_logic;
          O1: out std_logic);
  end component;
  component AND2
    port (I1, I2: in std_logic;
          O1: out std_logic);
  end component;
  for U0: OR2 use entity work.OR2(RTL);
  for U1: AND2 use entity work.AND2(RTL);
  signal S1: std_logic;
begin
  U0: OR2 port map (X, S1, Q);
  U1: AND2 port map (Y, Z, S1);
end RTL3;
```


Configuration에 대한 개념

- 개념의 필요성
  - Entity Unit에 대한 Architecture Unit이 2개 이상으로 표현한 경우 Simulation을 행할 때 특정 Architecture Unit에 대한 지정이 요구
  - 계층적 설계를 표현할 때 하위 레벨의 하드웨어 Component에 대한 Object Link를 행하여야 하므로 Component Object에 대한 지정이 요구



- 앞의 예제에 대한 고찰
  - Entity Unit EXAMPLE에 대한 3가지 형태의 Architecture Unit이 존재
  - Simulation을 행할 때는 Entity Unit과 대응하는 Architecture Unit에 대한 지정이 요구
  - Architecture Unit의 선언부에 표현하는 방식
    - Configuration Specification에 표현
  - 별도의 Configuration Unit에 표현하는 방식
    - Configuration Declaration Unit에 표현

Configuration Specification : 분산형

- 개념 정의
  - Architecture Unit을 Structural View 의 관점에서 표현하는 경우
  - 사용할 component에 대하여 compile 된 library의 이름과 해당 entity unit 및 architecture unit에 대한 이름을 지정함으로써 component binding을 행함
  - 1-Level의 단위로 표현함
  - architecture unit의 선언부에 기술
- Syntax 정의

```
for INSTANTIATION_LIST : COMPONENT_NAME
    [ use entity ENTITY_NAME { ( ARCHITECTURE_NAME ) } ]
    [ generic map ( GENERIC_ASSOCIATION_LIST ) ]
    [ port map ( PORT_ASSOCIATION_LIST ) ] ;
```

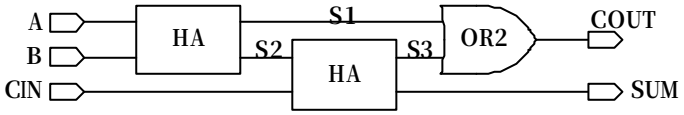
- 표현 예제
  - for U0 : NAND1, NAND2 use entity work.NAND_GATE(RTL);
    - 가장 기본 형태임
  - for U1 : INV1, INV2, INV3 use entity work.INVERTER;
  - for ALL : NOR use configuration LIB1.NOR_C;
    - NOR component가 compile된 library 이름이 LIB1인 경우
  - for others : NOT use entity work.NOT(RTL);

Configuration Declaration Unit : 분산형

- Behavioral View의 표현인 경우: 반드시 표현할 필요 없음

```
configuration TOP of XDOWN is
      for BEHAVIORAL  end for;
end TOP;
```

- Structural View의 표현인 경우: 분산형으로 표현되지 않는 경우에 사용 권장
  - 계층적 구조에 대하여 최상위 design entity에서의 configuration 표현 예

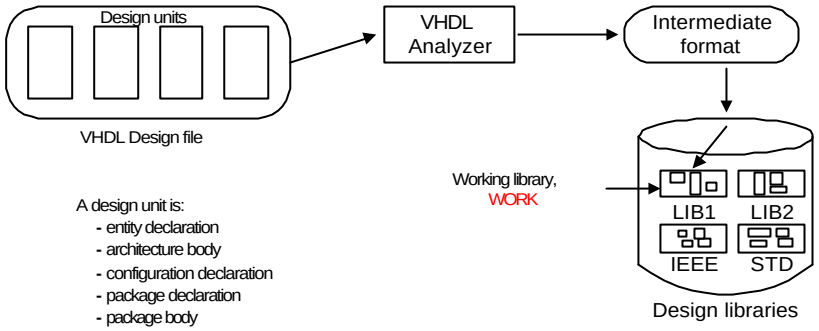



```
configuration FA_C of FA_E is
  for FA_A
    for u1,u2: HA use entity WORK.HA_E(HA_A) generic map (3 ns, 2 ns);
      for HA_A
        for u1: XOR2 use entity WORK.XOR2_E(XOR2_A) generic map (3 ns); end for;
        for u2: AND2 use entity WORK.AND2_E(AND2_A) generic map (2 ns);  end for;
      end for;
    end for;
    for u3: OR2 use entity  WORK.OR2_E(OR2_A) generic map (2 ns);  end for;
  end for;
end;
```

  - Generate 구문과 configuration declaration unit의 표현 예: generate 구문에서 설명

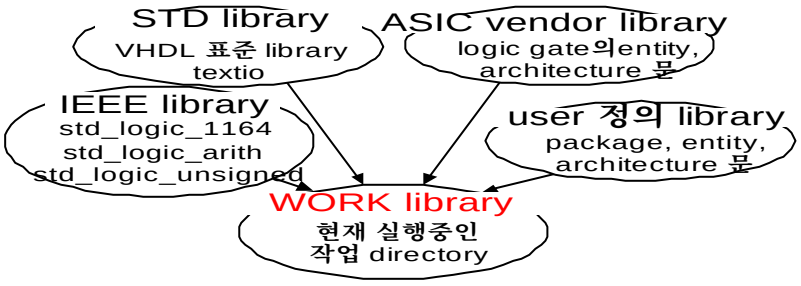
Source file과 Compiled file의 관계

- VDHL source file을 compile하는 경우
  - library의 지정이 요구되며, compiled file (intermediate format)은 지정된 library에 저장
- WORK library에 대한 개념 정의
  - VHDL 사용자가 사용하고 있는 현재의 library를 편의상 WORK로 부르기로 약속함
- IEEE VHDL 표준화 위원회(VASG)에서 지정한 library
  - 2-value를 위한 STD 와 9-value를 위한 IEEE
- VHDL Analyzer에 의한 처리 과정



Library의 종류와 Visibility의 범위

- Library의 종류



- VHDL Code에서 Package Unit을 사용하기 위한 Visibility의 부여 방법
  - 공유하는 package가 WORK library에 있는 경우 design unit 앞에 다음을 기술한다.
    - use WORK.package_name.item_name;
  - 공유하는 package가 LIB2에 Compile되어 있고, 공유를 희망하는 Code는 LIB1에 Compile되는 경우 기술하는 design unit 앞에 다음을 기술한다.
    - library LIB2;
    - use LIB2.package_name.item_name;

Library Visibility의 표현과 영향의 범위

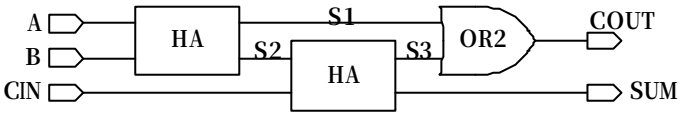
- Entity Unit의 표현 앞에 기술하는 경우
  - 해당되는 architecture와 configuration unit까지만 영향을 받음

```
--library visibility의 부여
library IEEE; use IEEE.std_logic_1164.ALL; use IEEE.std_logic_unsigned.all;
--entity unit에 대한 표현 시작
entity ADD_SUB is
  port ( A, B      : in std_logic_vector(3 downto 0);
         SEL       : in std_logic;
         S         : out std_logic_vector(3 downto 0);
         CF, ZF    : out std_logic);
end ADD_SUB;
```

- Package Declaration Unit의 표현 앞에 기술하는 경우
  - 해당되는 package body unit까지만 영향을 받음

```
library IEEE;use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all; use IEEE.std_logic_signed.all;
package PACK_MATCH is
   function COMP2C ( A: std_logic_vector) return std_logic_vector;
   function TWO_MUL ( A: std_logic_vector; B: std_logic) return std_logic_vector;
end PACK_MATCH;
```


Structural View의 표현을 위한 기본 개념의 정의



- 회로와 Structural View VHDL 표현
  - 사용 Component에 대한 선택과 지정
    - Architecture Unit에 선언
    - Component에 대한 D.B.를 지정
  - 회로 연결에 요구되는 Wire의 지정
    - Architecture Unit에 선언
  - 회로 구성 요소의 구성
    - Port Map 구문을 이용한 연결
- Component의 선언
  - Component 이름에 대한 선언
  - I/O Port에 대한 선언
    - Formal Name: Pin Name
  - 예제

```
component HA
  port (A, B: in std_logic;
        C, D: out std_logic);
end component;
```

- Component에 대한 지정
  - Configuration 구문에 의한 지정
  - 예제

```
for U0: HA use entity work.HA(RTL);
```

- Component를 연결하는 구문
  - 3가지 형태로 정의
  - Component Instantiation 구문
    - Actual Name: Wire Name
    - Pin과 Wire의 연결 표현 방법
  - 반복적인 특성의 H/W의 경우
    - if 또는 for generate 구문
    - Component Instantiation 구문
  - Pin과 Wire를 결합하는 2가지 방법
    - Positional Association Mapping
    - Named Association Mappin
- Wire의 Data Type
  - 반드시 Signal 이여야 한다.

Component Instantiation 구문에 의한 표현

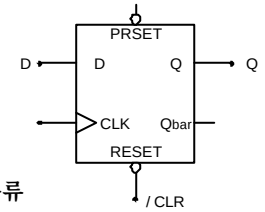
- 예제 회로에 대한 VHDL 표현

```
library IEEE; use IEEE.std_logic_1164.all;
entity FA is
  port (A, B, CIN : in std_logic;
        SUM, COUT : out std_logic);
end FA;
--
architecture RTL of FA is
  component HA
    port ( A, B : in std_logic;
           C, S : out std_logic);
  end component;
  component OR2
    port ( A, B: in std_logic;
           Y : out std_logic);
  end component;
  for U0, U1: HA use entity work.HA(RTL);
  for U2: OR2 use entity work.OR2(RTL);
  signal S1, S2, S3 : std_logic;
begin
  --Positional Association Mapping 예제
  U0 : HA port map (A, B, S1, S2);
  U1 : HA port map (CIN, S2, S3, SUM);
  U2 : OR2 port map (S1, S3, COUT);
  --Named Association Mapping 예제
  --U0 : HA port map (A=>A, B=>B,
  --          C=>S1, S=>S2);
  --U1 : HA port map (A=>CIN, B=>S1,
  --          C=>S3, S=>SUM);
  --U2 : OR2 port map (A=>S1, B=>S3,
  --          Y=>COUT);
end RTL;
```

- Compile 순서와 주의 사항
  - HA와 OR2 회로의 VHDL 표현
  - 하위 Component부터 Compile 함
    - HA와 OR2를 FA보다 먼저 수행
  - 하위 Component에 대한 변화가 있는 경우는 상위 Component에 대한 Compile은 다시 하여야 함

Unconnected Port에 대한 처리 방법

- Unconnected Port에 대한 의미
  - Pin에 대응하는 Wire가 없는 Port
- Unconnected Port에 대한 회로 예제



- 종류
  - Unconnected Output Port
  - Unconnected Input Port
- VHDL 표현에서의 처리 방향
  - Unconnected Input Port의 경우
    - VCC를 '1'로 정의하고 연결함
    - GND를 '0'으로 정의하고 연결함
  - Unconnected Output Port의 경우
    - Reserved Identifier인 OPEN을 연결함

- Unconnected Port에 대한 처리 예제

```
architecture RTL of EX1 is
  component EX2
    port (CLK,PRESET,RESET,D:
          in std_logic;
          Q, QBAR : out std_logic);
  end component;
  for U1: EX2 use entity work.DSR(RTL);
  signal VCC : std_logic;
begin
  --VHDL-87에서의 처리 방법
  VCC <= '1';
  U1 : EX2 port map (D => D,
       CLK =>CLK, PRESET =>VCC,
       RESET=>CLR, Q=>Q,
       QBAR=>open);
  --VHDL-93에서의 처리 방법
  --U1 : EX2 port map (D => D,
  --     CLK=>CLK, PRESET =>'1',
  --     RESET=>CLR, Q=>Q,
  --     QBAR=>open);
end RTL;
```

Generic Map Command의 사용

- Generic 구문의 Behavioral 표현

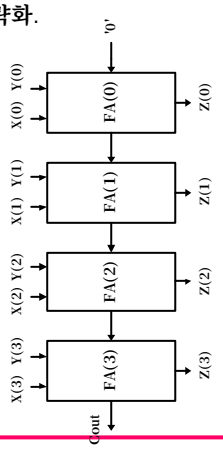
```
entity AND_GATE is
  generic (TDELAY : time := 10 ns;
           N : positive := 2);
  port (A : in std_logic_vector(N-1 downto 0);
        C : out std_logic);
end AND_GATE;
architecture RTL of AND_GATE is
begin
  P0 : process (A)
    variable INT : std_logic;
  begin
    INT := '1;
    for I in (A'length-1) downto 0 loop
      if (A(I)='0') then
        INT := '0';
      end if;
    end loop;
    C <= INT after Tdelay;
  end process;
end RTL;
```

- Generic Map을 갖는 Structural 표현

```
entity EX is
   port (D1, D2, D3, D4, D5 : in std_logic;
         Q1, Q2 : out std_logic);
end EX;
architecture RTL of EX is
  component AND_COMP
    generic (TDELAY : time; N : positive);
    port (A : in std_logic_vector(N-1 downto 0);
          C : out std_logic);
  end component;
  for all : AND_COMP use entity
            work.AND_GATE(RTL);
begin
  U1: AND_COMP
      generic map (n=>2, TDELAY=>8 ns)
      port map (A(0)=>D1, A(1)=>D2, C=>Q1);
  U2: AND_COMP
      generic map (n=>3, TDELAY=>12 ns)
      port map (A(0)=>D3, A(1)=>D4,
                A(2)=>D5, C=>Q2);
end RTL;
```


For_Generate 구문에 의한 Structure View 표현

- 4-Bit Ripple Carry Adder 예제 회로
  - 반복되는 Component의 Symbol Name과 Formal Name이 서로 같은 경우.
  - Component Instantiation 구문의 사용 횟수를 줄일 수 있어 구문을 간략화.



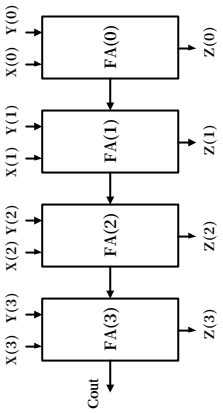
- Architecture와 Configuration Unit

```
architecture RTL1 of FA4 is
  -- FA Component에 대한 선언을 기술한다.
  signal TMP : std_logic_vector(4 downto 0);
begin
  TMP(0) <= '0';
  G: for I in 0 to 3 generate
    L : FA port map ( X(I), Y(I), TMP(I),
                      Z(I), TMP(I+1));
  end generate;
  COUT <= TMP(4);
end RTL1;
--
configuration CONF_FA4 of FA4 is
  for RTL
    for G
      for L: FA use entity work.FA(RTL); end for;
    end for;
  end for;
end CONF_FA4;
```

- Port Map 구문 횟수 감소
- Configuration에 대한 복잡성 증대

If 및 For..Generate 구문을 이용한 Structure View 표현

- 4-Bit Ripple Carry Adder 예제 회로
  - 일부의 Component들의 Symbol Name이 다른 경우.
  - 조건적으로 Component Instantiation 구문을 사용하여 구문을 간략화.



- Architecture Unit의 설계

```
architecture RTL1 of FA4 is
  -- FA, HA Component에 대한 선언에 대한
  -- 표현을 행하여야 함
  signal T : std_logic_vector(4 downto 1);
begin
  G0: for I in 0 to 3 generate
    G1 : if I=0 generate
      L0 : HA port map (X(I),Y(I),Z(I),T(I+1));
    end generate;
    G2 : if I>=1 and I<=3 generate
      L1 : FA port map
            (X(I),Y(I),T(I),Z(I),T(I+1));
    end generate;
  end generate;
  COUT <= TMP(4);
end RTL1;
```


Generate 구문과 Configuration Declaration Unit의 예제

- Configuration Unit에 대한 표현

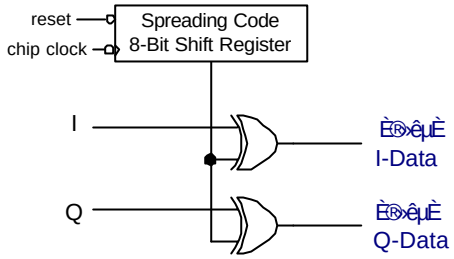
```
configuration CONF_FA4 of FA4 is
  for RTL1
    for G0
      for G1
        for L0: HA use entity work.HA(RTL);
        end for;
      end for;
      for G2
        for L1: FA use entity work.FA(RTL);
        end for;
      end for;
    end for;
  end for;
end CONF_FA4;
```

- 앞의 Component를 사용하는 구문
  - component configuration에 주의

```
architecture RTL of TB_FA4 is
  signal X, Y: std_logic_vector(N-1 downto 0);
  signal Z: std_logic_vector(N-1 downto 0);
  signal COUT: std_logic;
  component FA4
    generic(N: integer := 4);
    port (X: in std_logic_vector(N-1 downto 0);
          Y: in std_logic_vector(N-1 downto 0);
          Z: out std_logic_vector(N-1 downto 0);
          COUT: out std_logic);
  end component;
  for U0: FA4 use configuration
                  work.CONF_FA4;
begin
  U0: FA4 port map (X, Y, Z, COUT);
  X <= "0011", "1010" after 40 ns;
  Y <= "1000", "1110" after 20 ns;
end RTL;
```

H/W의 병렬 동작과 Sensitivity Signal의 개념 정의

- 임의의 하드웨어 구성



- Sensitivity Signal에 대한 정의
  - 임의의 하드웨어 Block에 대한 모든 입력 신호를 의미함
  - 예제
    - Shift Register Block
      - reset 신호, chip clock 신호
    - XOR Gates Block
      - I, Q 및 Shift Register의 출력 신호
- 병렬 동작에 대한 개념
  - Simulation 동작의 이해
    - Active State: 동작 상태
    - Suspend State: Sensitivity Signal의 변화를 검사하는 상태
  - 8-Bit Shift Register
    - Reset='0' 또는 Chip Clock에의 Falling Edge 발생 조건을 검사함
    - 위의 조건을 만족하는 경우 Shift 동작을 수행하고, 위의 조건 검사를 수행하기 위해 천이한다.
  - 2개의 XOR Gate Group
    - Register의 출력, I 또는 Q 중의 어느 한 값이라도 변화를 검사함
    - 위의 조건을 만족하는 경우 XOR Gate의 동작을 수행하고, 위의 조건 검사를 계속 수행하기 위해 천이한다.
- VHDL 표현의 다양성 고찰
  - 3개의 Concurrent 구문으로 표현
  - 2개의 Concurrent 구문으로 표현
  - 1개의 Concurrent 구문으로 표현

VHDL 구문에 대한 분류

- Behavioral View Concurrent 구문
  - signal assignment statement
    - basic signal assignment
    - conditional signal assignment
    - selected signal assignment
  - process statement with many sequential statements
  - concurrent subprogram call statement
  - block statement
    - basic block
    - guarded block signal assignment
  - concurrent assert statement
- Structural View Concurrent 구문
  - component instantiation statement
  - Generate statement
- 기술 순서에 따른 수행을 행하는 구문
  - Behavioral View 표현을 위하여 정의
  - 사용에의 제한
    - process 구문 내부에 사용
    - function 또는 procedure와 같은 subprogram 내부에 사용
  - wait statement
    - Vector 생성 표현에 주로 사용
  - basic signal assignment statement
  - variable assignment statement
  - if statement
  - case statement
  - loop statements
    - for..loop statement
    - while loop statement
    - infinite loop statement
  - next statement, exit statement
  - null statement
  - sequential subprogram call statement
  - sequential assert statement

Data에 대한 연산을 행하기 위한 Operator에 대한 정의

- Logical Operators
  - not and or nand nor xor xnor(VHDL-93)
  - Operand Type : std_logic, boolean
  - Result Type : std_logic, boolean
- Relational Operators
  - = /= < <= > >=
  - Operand Type : any type
  - Result Type: boolean
- Arithmetic Operators
  - + - * / ** MOD REM ABS
  - Operand Type : integer, real, physical
  - Result Type: integer, real, physical
- Concatenation Operators : &
  - Operand Type : array of any type
  - Result Type : array of any type

- MOD와 REM의 차이점 이해
  - A = B*N + (A mod B) (N: 임의 정수)
  - A = (A/B)*B + (A rem B)
  - A=10, B=3: A mod B = 1, A rem B = 1
  - A=-10, B=3: A mod B = 2, A rem B = -1
  - A=11, B=- 4: A mod B = -1, A rem B = 3

- 표현에서의 주위 사항
  - 동일 수준의 operator 연산 순서 표현
    - 좌측부터 연산 되지 않음
    - 괄호를 사용하여야 한다
  - 예제

```
reset='0' or (clk='0' and clk'event)
A <= ((B nand C) nand D) nand E;
A <= (B and C) or (D and E)
```


Predefined Type and Array Attributes에 대한 정의와 예

- Attribute에 대한 정의
  - type attribute, array attribute, signal attribute, entity attribute
- Array Attribute에 대한 예제
  - 예제 설명을 위하여 다음과 같은 Type 선언문을 가정한다.

```
type COLOR is (RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET);
type STATE is (S0, S1, S2, S3, S4);
type INDEXD is range 7 downto 0;
```

  - 보편적으로 사용되는 Signal Type Attribute의 예에 따른 Return Value 결정

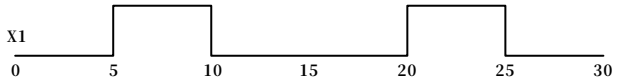
```
COLOR'pos(GREEN) = 3                COLOR'val(3) = GREEN
STATE'pos(S2) = 2                   STATE'val(0) = S0
COLOR'left = RED        STATE'left = S0        STATE'right = S4
COLOR'low  = RED        COLOR'high = VIORET    STATE'high = S4
COLOR'pred(GREEN) = YELLOW                  STATE'succ(S3) = S4
COLOR'leftof(GREEN) = YELLOW                STATE'rightof(S2) = S3
INDEXD'left = 7                             INDEXD'right = 0
INDEXD'low  = 0                             INDEXD'high = 7
INDEXD'pred(6) = 5                          INDEXD'leftof(6) = 7
INDEXD'succ(6) = 7                          INDEXD'rightof(6) = 5
INDEXD'range = 7 downto 0                   INDEXD'reverse_range = 0 to 7
INDEXD'length = 8
```

Predefined Signal Attributes에 대한 정의

- 정의와 기능에 대한 설명
  - X2'active
    - 현재의 Simulation Cycle 동안 X2에 Transaction이 발생하면 True
  - X3'event
    - 현재의 Simulation Cycle 동안 X3에 Event가 발생하면 True
  - X2'stable(5 ns)
    - 현재의 Simulation Cycle부터 5 ns 이전 동안 X2에 Event가 발생하지 않았으면 True
  - X2'quiet(5 ns)
    - 현재의 Simulation Cycle부터 5 ns 이전 동안 X2에 Transaction이 발생하지 않았으면 True
  - X2'last_event
    - 현재의 Simulation Cycle의 관점에서 가장 최근에 발생한 Event로부터 경과한 시간
  - X2'last_active
    - 현재의 Simulation Cycle의 관점에서 가장 최근에 발생한 Active로부터 경과한 시간
  - X2'last_value
    - 현재의 Simulation Cycle의 관점에서 가장 최근에 발생한 Event 에 대하여 직전의 값

Predefined Signal Attributes에 대한 예제

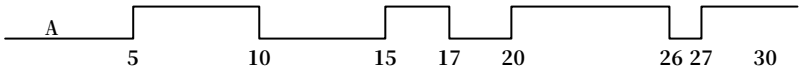
- 예제를 위한 Signal X1의 변화를 표현하는 파형
  - Simulation이 수행되고 있는 시간 = 15 ns 인 경우 Attribute에 대한 값의 판정



    - X1'stable(8 ns) : FALSE
    - X1'stable(2 ns) : TRUE
    - X1'event : FALSE
    - X1'last_event : 5 ns
    - X1'last_value : '1'
  - Simulation이 수행되고 있는 시간 = 20 ns 인 경우 Attribute에 대한 값의 판정
    - X1'event and X1='1' : TRUE
    - X1='1' and (not X1'stable) : TRUE
  - Simulation이 수행되고 있는 시간 = 25 ns 인 경우 Attribute에 대한 값의 판정
    - X1'event and X1='0' : TRUE
    - X1='0' and (not X1'stable) : TRUE
- X1'delayed(2 ns)에 따라 생성되는 파형을 생각하여 보자
  - 각 Value가 2 ns delay된 파형을 생성함

Delay Model에 대한 정의와 예제 (VHDL-87)

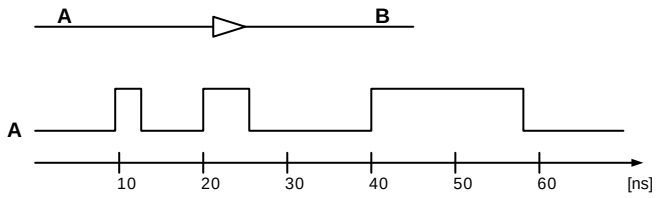
- Delay 정의
  - Delay Selection
    - Inertial Delay : VHDL default delay
      - 제시된 delay 보다 짧은 변화가 입력에 나타나는 경우는 noise로 취급하여 출력에 영향을 주지 않음.
      - 이외의 경우는 제시된 delay 후에 입력의 변화가 출력에 나타남.
      - Component delay를 위하여 사용함이 보편적임.
    - Transport Delay
      - 입력에의 변화가 제시된 delay 후에 출력에 반드시 나타남.
      - Interconnect delay modeling에 사용함이 보편적임.
  - Internal Delay
    - Delta Delay : concurrent 하게 수행되는 구문들의 event driven simulation을 행하기 위하여 정의된 delay이다.
- Simulation 예제 : 다음 신호에 대한 각 구문의 수행에 따른 결과를 Plot 하여 보자.



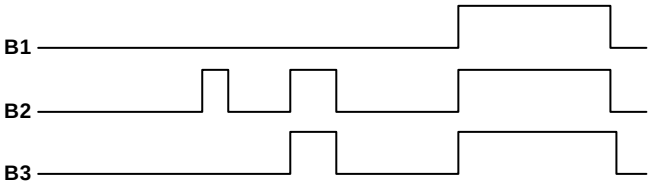
  - B <= transport A after 3 ns;
  - C <= A after 3 ns;
  - D <= A;

Delay Model에 대한 정의와 예제 (VHDL-93)

- VHDL-87에서 Inertial Delay Model에 대한 수정을 다음과 같이 행함
  - Reject Delay Model의 제시

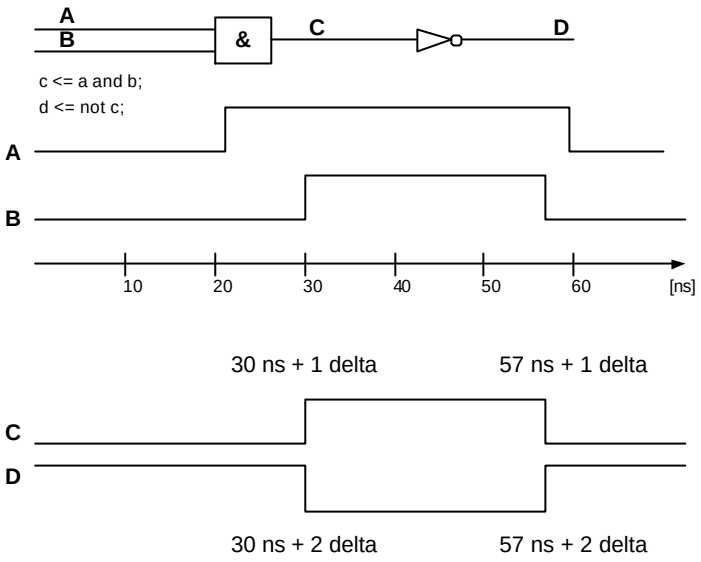


b1 <= inertial a after 10 ns;
b2 <= transport a after 10 ns;
b3 <= reject 4 ns inertial a after 10 ns;



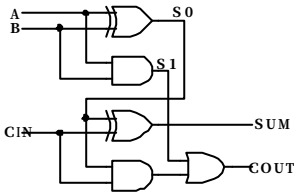
Delta Delay Model에 대한 이해

- VHDL Simulation Engine 구동을 위하여 정의된 Delay Model



Basic Signal Assignment 구문 예제

- Full-Adder 회로와 각 Gate에 대한 Sensitivity Signal에 대한 개념 정의



- Basic Signal Assignment 구문에 의한 Architecture Unit의 설계

```
architecture RTL of ADDER is
  signal S0, S1 : std_logic;
begin
  --각 gate를 signal assignment 구문으로 표현
  SUM <= S0 xor CIN;
  COUT <= S1 or (S0 and CIN);
  S0 <= A xor B;
  S1 <= A and B;
end RTL;
```

- 다양한 형태의 Signal Assignment 구문
  - Aggregate : 하나의 Target Signal에 여러 개로부터의 Source 값을 Assign하는 경우

```
architecture RTL of EX is
  signal A : std_logic_vector( 4 downto 0);
  signal B : std_logic_vector( 4 downto 0);
begin
  A <= (others => '0');
  -- A <= "00000";
  -- A <= ('0', '0', '0', '0', '0');

  B <= (1=>C(2), 3=>C(1), others => D(0));
  -- B <= D(0)&C(1)&D(0)&C(2)&D(0);
end RTL;
```

  - 하나의 Source 값을 여러 개의 Target Signal에 나누는 경우의 표현 방법에 대하여도 생각하여 보자.

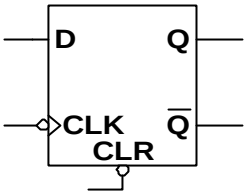
Conditional Signal Assignment 구문의 정의와 예제

- Syntax 정의와 동작 개념

```
signal_identifier <= waveform_1 when condition_1 else
                     waveform_2 when condition_2 else
                        ……..
                     waveform_(N-1) when condition_(N-1) else
                     waveform_N ;
```

- D Type Flip-Flop의 VHDL 표현과 Symbol

```
architecture A2 of DFF is
begin
  Q <= '0' when CLR='0' else
       D when (CLK='0' and CLK'EVENT) else
       Q;
       -- D when (CLK='0' and CLK'EVENT);  Valid in VHDL-93
  QBAR <= not Q;
end A2;
```



- Data Selector에 대하여 Conditional Signal Assignment 구문에 의한 표현

```
MUXOUT <= I0 when (SEL='0') else
          I1;  -- when (SEL='1') else 'X';
```


Selected Signal Assignment 구문의 정의와 예제

- Syntax 정의와 동작 개념

```
with expression select
   signal_identifier <=
      waveform1 when expression_1,
         ……..
      waveform(n-1) when expression(n-1),
      waveform(n) when others ;
```

- 2*1 MUX에 대한 표현

```
library IEEE;
use IEEE.std_logic_1164.ALL;
entity MUX21 is
  port (SEL, A, B : in std_logic;
        C : out std_logic);
end MUX21;
architecture RTL of MUX21 is
begin
  with SEL select
    C <= A after 10 ns when '0',
         B after 10 ns when others;
end RTL;
```

- Half-Adder에 대한 표현

```
library IEEE;
use IEEE.std_logic_1164.ALL;
entity HA is
  port (A, B : in std_logic;
        SUM, COUT : out std_logic);
end HA;
architecture RTL of HA is
   signal TMP1, TMP2 :
         std_logic_vector(1 downto 0);
begin
   TMP1 <= A & B;
   with TMP1 select
     TMP2 <= "00" when "00",
             "01" when "01",
             "01" when "10",
             "10" when others;
   (COUT, SUM) <= TMP2;
end RTL;
```

Process 구문에 대한 정의와 예제

- Syntax 정의

```
[ Label : ] process [ ( Sensitivity_Signals ) ]
 -- declaration statements
begin
 -- sequential activity statements
end process [ Label ] ;
```

- Sensitivity Signal이 있는 경우 동작
  - Wait 구문을 포함할 수 없다.
  - Simulation에 대한 수행 명령과 함께 각 Signal의 초기값을 갖고 내부 구문에 대한 1회의 수행을 수행한다.
  - 수행 완료와 함께 Process 문에 표현된 Sensitivity Signal List에 포함된 Signal 중 어느 하나라도 Event가 생기는 가를 검사하여 Event가 발생하면 Process 문에 대한 Simulation을 다시 수행한다.
  - 합성 H/W 표현의 이용에 권장
- Sensitivity Signal이 없는 경우 동작
  - 1개 이상의 Wait 구문을 포함한다.
  - Simulation의 시작과 함께 Wait 구문을 만날 때까지의 구문을 수행한다.
  - Wait 구문을 만나면 해당 조건을 만족할 때까지는 Simulation을 중지한다.
  - Stimulus Vector 표현의 이용에 권장
- Sensitivity Signal의 중요성 이해
  - Half-Adder에 대한 표현 예제

```
process (A, B)              process (A)
begin                       begin
  SUM <= A xor B;             SUM<=A xor B;
  COUT <= A and B;            COUT<=A and B;
end process;                end process;
```

  - 각 경우에 따른 이해와 문제점 파악
    - Simulation의 경우 (다른 결과)
    - Synthesis의 경우 (동일 결과)
    - 우측의 표현이 정확한 표현임

Signal 및 Variable Assignment 구문의 정의와 예제(1)

- Assignment 구문의 종류 : Target Identifier의 Object에 따라 2가지로 분류
  - Signal Assignment 구문에 대한 정의
    - [label:] signal_identifier <= [transport] expression [after expression];
  - Variable Assignment 구문에 대한 정의
    - [label:] variable_identifier := value_expression;
- 2가지 Assignment 구문에 대한 차이점을 이해하기 위한 Code와 결과 예제-1

```
process
  variable num,sum:integer := 0;
begin
   wait for 10 ns;
   num := num + 1;
   sum := sum + num;
end process;
--
process  begin
   wait for 10 ns;
   num <= num + 1;
   sum <= sum + num;
end process;
```

	variable process			***signal process***		
	time	*num*	*sum*	*time*	*num*	*sum*
	0	0	0	0	0	0
	10	1	1	10	0	0
	10+	1	1	10+	1	0
	20	2	3	20	1	0
	20+	2	3	20+	2	1
	30	3	6	30	2	1
	30+	3	6	30+	3	3

Signal 및 Variable Assignment 비교를 위한 예제

✍ Signal Assignment 구문 사용

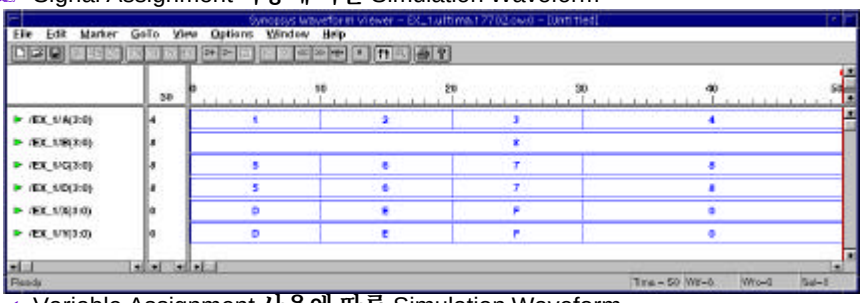
```
library IEEE; use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity EX_1 is end EX_1;
architecture RTL of EX_1 is
  signal A, B, C, D, X, Y:
     std_logic_vector(3 downto 0);
begin
  process (A, B, C, D)
  begin
    D <= A;
    X <= B + D;
    D <= C;
    Y <= B + D;
  end process;
  A <= "0001", "0010" after 10 ns,
        "0011" after 20 ns, "0100" after 30 ns;
  B <= "1000";
  C <= "0101", "0110" after 10 ns,
        "0111" after 20 ns, "1000" after 30 ns;
end RTL;
```

✍ Variable Assignment 구문 사용

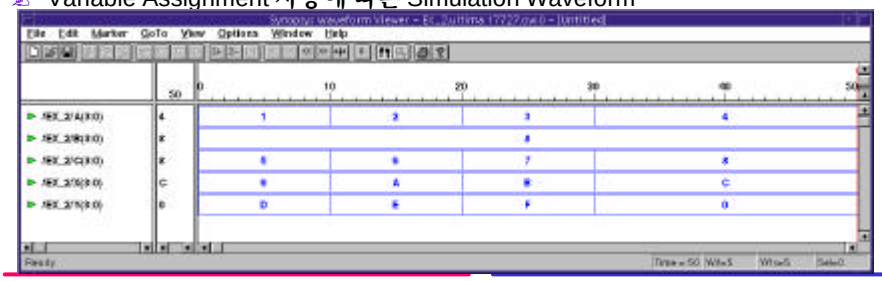
```
library IEEE; use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
entity EX_2 is end EX_2;
architecture RTL of EX_2 is
  signal A, B, C, X, Y:
     std_logic_vector(3 downto 0);
begin
  process (A, B, C)
   variable D: std_logic_vector(3 downto 0);
  begin
    D := A;
    X <= B + D;
    D := C;
    Y <= B + D;
  end process;
  A <= "0001", "0010" after 10 ns,
        "0011" after 20 ns, "0100" after 30 ns;
  B <= "1000";
  C <= "0101", "0110" after 10 ns,
        "0111" after 20 ns, "1000" after 30 ns;
end RTL;
```


Simulation에 따른 Waveform 비교

✍ Signal Assignment 사용에 따른 Simulation Waveform



✍ Variable Assignment 사용에 따른 Simulation Waveform



Wait 구문 정의와 예제

- Syntax 정의

  wait [on list] [until condition] [for time] ;

- 사용 예제
  - wait;
    - 이후의 구문은 Simulation의 Restart 동작 때까지는 영원히 수행되지 않는다.
  - wait on A;
    - Signal A의 값이 변화할 때까지 Simulation은 중단된다.
  - wait until (CLK='0' and CLK'event);
    - CLK Signal의 Falling Edge를 검출할 때까지는 Simulation은 중단된다.
  - wait for 100 ns;
    - 100 ns의 Simulation 시간을 기다리고, 다음의 구문을 수행한다.
  - wait until A='1' for 50 ns;
    - 최대 50 ns 만을 기다리면서 그 동안에 A에 변화가 있고 A='1'을 만족하면 다음의 구문을 수행한다.
  - wait on A until (C='1') for 50 ns;
    - 최대 50 ns 만을 기다리면서 그 동안에 A에 변화가 있고 C='1'을 만족하면 다음의 구문을 수행한다.
- Wait 구문에 대한 사용 제한 (주의 깊은 이해가 요구됨)
  - Synthesis Code에 사용: Clock 신호의 검출을 wait 구문으로 표현하는 합성 Tool
    - wait until 구문에 의한 초기화 신호와 CLK Edge 검출
  - 나머지 형태는 Test Bench Modeling에만 사용 가능함

다양한 Process 구문의 동작에 대한 이해

- Process 구문과 Simulation 차이 이해

```
process (A)
begin
 C1<=NOT A;
end process;
```

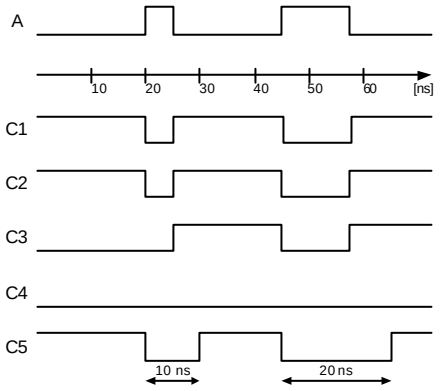
```
process
begin
 C2<=NOT A;
 wait on A;
end process;
```

```
process
begin
  wait on A;
  C3<=NOT A;
end process;
```

```
process
begin
 wait until A='1';
 C4<=not A;
end process;
```

```
process
begin
 C5<=NOT A;
 wait until A='1' for 10 ns;
end process;
```

- Simulation에 따른 파형 이해



Clock 신호 생성기 표현과 Wait 구문의 사용

- Test Bench Modeling에서 clock 신호에 대한 다양한 VHDL 표현의 예
  - 가정: CLK1, CLK2, CLK3, CLK4, CLK5, CLK6의 초기값은 모두 '0'이다.

```
process begin
  CLK1 <= not CLK1;
  wait for 25 ns;
end process;
--
process (CLK3) begin
  CLK3 <= not CLK3 after 25 ns;
end process;
--
process
begin
  CLK5 <= '0'; wait for 20 ns;
  CLK5 <= '1'; wait for 30 ns;
end process;
```

```
process begin
  CLK2 <= not CLK2 after 25 ns;
  wait on CLK2;
end process;
--

CLK4 <= not CLK4 after 25 ns;

--
process begin
  if (now = 0 ns) then
    CLK6 <= '0'; wait for 100 ns;
  else
    CLK6 <= NOT CLK6; wait for 25 ns;
  end if
end process;
```

- Quiz: 위의 구문에 대한 Simulation 파형을 예측하여 보자.

If 구문에 대한 정의와 사용 예제

- Syntax 정의와 동작 개념
  - if...then…end if;
  - if... then...else…end if;
  - if…then…elsif …elsif …else…end if;
  - if…then….if…then……end if….end if;
  - 주의 사항
    - Latch 생성 요인이 있음

- IF문을 이용한 다양한 예제
  - 예제-1: 2*1 MUX에 대한 표현

```
process (A, B, SEL)
begin
  if SEL='0' then
    Y <= A;
  else
    Y <= B;
  end if;
end process;
```

  - 예제-2: 4*1 MUX에 대한 표현

```
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;
  else
    Y <= D;
  end if;
end process;
```

  - 예제-3: Latch가 생성됨에 주의할 것

```
process (EN, D)
begin
  if EN='1' THEN
    Q <= D;
  end if
end process;
```

Case 구문에 대한 정의와 사용 예제

- Syntax 정의와 동작 개념

```
case expression is
  when condition_1 =>
       (sequential_statements;)
   …..
  when condition_(n-1) =>
       (sequential_statements;)
  when others =>
       (sequential_statements;)
end case ;
```

  - when condition 구문 표현

```
when Value =>
when Value | Value | Value… =>
when Value to Value =>
when others =>
```

- Truth Table에 대한 표현에 적합함
  - 1 - Bit 단위의 입력 신호를 & operator를 사용하여 Multi-Bits의 Variable로 설정
  - 설정 변수를 *expression* 으로 사용하여 각 경우를 분류

- 1-Bit Half-Adder에 대한 표현

```
architecture RTL1 of EX is
 --signal TMP1 :
 --        std_logic_vector(1 downto 0);
begin
 --TMP1 <= A & B;
 --process (TMP1)
 process (A, B)
    variable TMP1, TMP2 :
             std_logic_vector(1 downto 0);
 begin
    TMP1 := A & B;
    case TMP1 is
       when “00” => TMP2 := “00”;
       when “01” | “10” => TMP2 := “01”;
       when others => TMP2 := “10”;
    end case;
    (COUT, SUM) <= TMP2;
 end process;
end RTL;
```

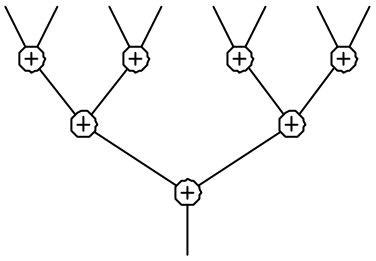

For..Loop 구문에 대한 정의와 사용 예제

- Syntax 정의

```
[Label:]
for loop_parameter in discrete_range loop
   --sequential statements
end loop [Label] ;
```

- Range 표현 방법
  - integer_expression to (또는 downto)
  - integer_expression array_attribute'range
  - array_attribute'reverse_range
- loop_parameter의 주의 사항
  - Signal 또는 Variable로 선언하지 않고 사용한다.
- 예제

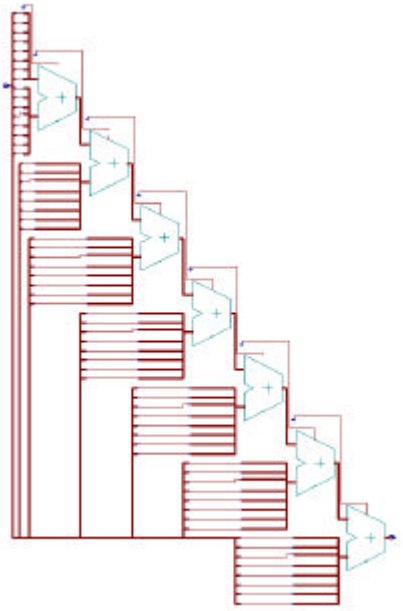
- For..Loop의 합성 이해를 위한 예제
  - 다음의 Data-flow Graph에 대한 VHDL 구문을 생각하여 보자.



  - 표현에 대한 고찰
    - Signal Assignment에 의한 Data-Flow 표현
    - For..Loop 구문을 이용하여 간결하게 표현

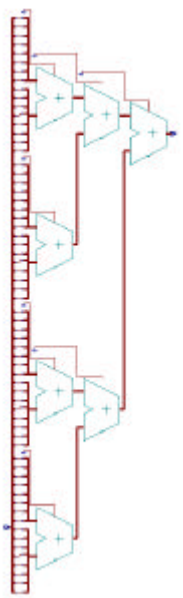
For..Loop 구문 사용에 따른 논리 합성 예(1)

```
library IEEE; use IEEE.std_logic_1164.all;
package PACK is
  type RAM_TYPE is array(0 to 7) of
     std_logic_vector(7 downto 0);
end PACK;
library IEEE; use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.PACK.all;
entity EX_3 is
  port (A: in RAM_TYPE;
        Y: out std_logic_vector(7 downto 0));
end EX_3;
architecture RTL of EX_3 is
begin
  process (A)
    variable SUM: std_logic_vector( 7 downto 0);
  begin
    SUM := (others => '0');
    for I in 0 to 7 loop
      SUM := SUM + A(I);
    end loop;
    Y <= SUM;
  end process;
end RTL;
```



For..Loop 구문 사용에 따른 논리 합성 예(2)

```
library IEEE; use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.PACK.all;
entity EX_4 is
  port (A: in RAM_TYPE;
        Y: out std_logic_vector(7 downto 0));
end EX_4;
architecture RTL of EX_4 is
begin
  process (A)
    variable COUNT: integer;
    variable B: RAM_TYPE;
  begin
    COUNT := 8;
    for I in 0 to 7 loop
      B(I) := A(I);
    end loop;
    for LEVEL  in 0 to 2 loop
      COUNT := COUNT / 2;
      for I in 0 to (COUNT-1) loop
        B(I) := B(I*2) + B(I*2+1);
      end loop;
    end loop;
    Y <= B(0);
  end process;
end RTL;
```



While..Loop 구문에 대한 정의와 사용 예제

- While..Loop 구문의 Syntax와 개념

```
[Label:] while condition loop
  --sequential statements
end loop [Label] ;
```

- 1 ~ N 까지 정수 값의 합 계산 예제

```
entity ADD_LOOP is
  port ( N : in integer;
         SUM : out integer);
end ADD_LOOP;
architecture RTL of ADD_LOOP is
begin
  process (N)
    variable TMP, I : integer;
  begin
    TMP := 0; I := 1;
    while (I <= N) loop
      TMP := TMP + I;    I := I + 1;
    end loop;
    SUM <= TMP;
  end process;
end RTL;
```

- Infinite Loop 구문 예제

```
process (N)
  variable TMP, I : integer;
begin
  TMP := 0;  I := 0;
  FIRST: loop
    I := I + 1;
    TMP := TMP + 1;
    exit when  (I = N) ;
  end loop FIRST;
end process;
```

- 합성에의 고려 사항
  - Advanced Synthesis Tool에서만 합성이 가능한 구문임에 주의
  - Behavioral Compiler에서 지원함

Next와 Exit 구문에 대한 정의와 사용 예제

- Next 구문의 Syntax와 개념
  - next;
  - next loop_label ;
  - next when boolean_expression ;
  - next loop_label
    when boolean_expression ;
- 예제

```
LBL1 : process (S)
  variable TMP, J : integer := 0;
begin
  TMP := 0;  J := 0;
  A_LOOP: for I in 0 to 7 loop
    J := J + 1;
    if J > S then
      next A_LOOP;
    end if;
    TMP := TMP + 1;
  end loop;
end process LBL1;
```

- Exit 구문의 Syntax와 개념
  - exit;
  - exit loop_label ;
  - exit when boolean_expression ;
  - exit loop_label
    when boolean_expression ;
- 예제

```
LBL2 : process (S)
  variable SUM, CNT : integer := 0;
begin
  SUM:= 0;   CNT:= 0;
  FIRST: loop
    CNT:=CNT+1;   SUM:=SUM+CNT;
    exit when SUM > 100;
  end loop FIRST;
  --SECOND: loop
  --   CNT:=CNT+1;   SUM:=SUM+CNT;
  --   if SUM > 100 then exit;
  --end loop SECOND;
end process LBL2;
```

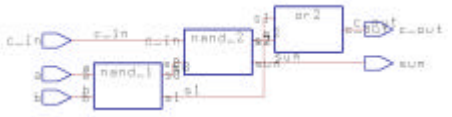
Block 구문에 대한 정의와 사용 예제

- Basic Block Statement의 사용 예와 합성에의 사용에 따른 차이점

```
library IEEE; use IEEE.std_logic_1164.all;
entity BLK_FA is
  port ( A, B, C_IN: in std_logic;
         C_OUT, SUM: out std_logic);
end BLK_FA;
architecture RTL of BLK_FA is
  signal S0, S1, S2 : std_logic;
begin
  NAND_1: block begin
    S0 <= A xor B;
    S1 <= A and B;
  end block;
  NAND_2: block begin
    SUM <= S0 xor C_IN;
    S2 <= S0 and C_IN;
  end block;
  OR2: block begin
    C_OUT <= S1 or S2;
  end block;
end RTL;
```

- Script File과 논리 합성 예

```
read -f vhdl blk_fa.vhd
group -hdl_block nand_1
group -hdl_block nand_2
group -hdl_block or2
/* create_clock clk */
max_delay 0 -to all_outputs()
compile -map_effort high
create_schematic
write -f db
write -f edif -hierarchy -o blk_fa.edf
```

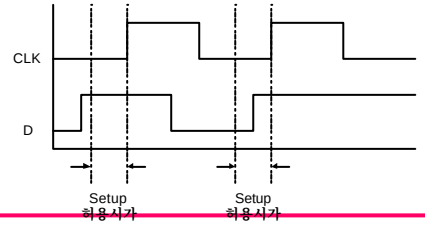


- Guarded Block Statement의 예

```
D_FF: block(clk='0' and clk'event)
begin
  Q <= guarded D;
end block;
```


Assert 구문에 대한 정의와 사용 예제(1)

- Syntax 정의와 개념
  - assert condition ;
  - assert condition report "message" ;
  - assert condition severity level ;
  - assert condition report "message" severity level ;
- 특성
  - Passive 구문: H/W로 구성될 수 없음
  - Concurrent, Sequential Assert 구문
- D Flip-Flop과 Assert 구문의 사용 예
  - 비동기 입력 신호에 대한 조건 검사
  - Setup Time Violation 여부 검사
- Setup Time에 대한 의미 파악



- Assert 구문에 대한 사용 예

```
library IEEE; use IEEE.std_logic_1164.all;
entity DFF_A_SR is
  generic (D_SETUP : time := 2 NS);
  port (SD_N, RD_N, CLK, DIN : in std_logic;
        Q, Q_N : out std_logic);
begin
  assert (SD_N='1' or RD_N='1')
    report "PRESET and CLEAR are both '0'" ;
  process (CLK, DIN)
    variable D_CHANGE : time := 0 ns;
    variable D_VALUE, CLK_VALUE : std_logic;
  begin
    if (D_VALUE /= DIN) then
      D_CHANGE := now;    D_VALUE := DIN;
    end if;
    if (CLK_VALUE /= CLK) then
      CLK_VALUE := CLK;
      if (CLK='1') then
        assert (now - D_CHANGE >= D_SETUP)
          report "SETUP VIOLATION" ;
      end if;
    end if;
  end process;
end DFF_A_SR;
```

Assert 구문에 대한 정의와 사용 예제(2)

- Assert 구문을 이용한 검증 예제

Package 정의와 예제

- Package에 대한 정의
  - 공유하는 정보를 표현하기 정의
- Syntax 구성과 정의

```
package package_name is
  --exported_subprogram_declarations
  --exported_constant_declarations
  --exported_components
  --exported_type_declarations
  --attribute_declarations
  --attribute_specifications
end [package_name] ;
package body  package_name is
  --exported_subprogram_bodies
  --internal_subprogram_declarations
  --internal_subprogram_bodies
  --internal_constant_declarations
  --internal_components
  --internal_type_declarations
end [package_name] ;
```

- Package 선언부

```
package EX_PKG is
  subtype INT8 is INTEGER range 0 to 255;
  constant  ZERO   : INT8 := 0;
  constant  MAX    : INT8 := 100;
  procedure Increment (variable Count:
                inout INT8);
end EX_PKG;
```

- Package 몸체부

```
package body EX_PKG is
  procedure Increment (variable Count:
                inout INT8) is
  begin
    if (Count >= MAX) then
          Count := Zero;
    else
          Count := Count + 1;
    end if;
  end Increment;
end EX_PKG;
```

Predefined Package에 대한 이해

- Predefined Packages
  - 표준화 Group에서 정의한 Package.
    - Library Name : STD
    - Package Name
      - standard
      - textio
      - textio package에 대하여는 visibility를 부여하여야 한다.
    - Library Name : IEEE
    - Package Name
      - std_logic_1164
      - std_logic_textio
      - 사용을 위하여 Visibility를 부여하여야 한다.
  - Synopsys에서 정의한 Package.
    - Library Name : IEEE
    - Package Name
      - std_logic_unsigned
      - std_logic_signed
      - std_logic_arith
    - 차이점 이해를 위한 예제

- Package에 대한 위치와 이해
  - 위치

    $/synopsys/packages/IEEE/src/*.vhd
  - 내용

    std_logic_1164.vhd

    std_logic_arith.vhd

    std_logic_signed.vhd

    std_logic_unsigned.vhd

    std_logic_textio.vhd

    기타

Unsigned Data 연산을 위한 표현 예제

- 선언부 표현
  - Data Type: std_logic_vector
  - Package: std_logic_unsigned
- VHDL 예제

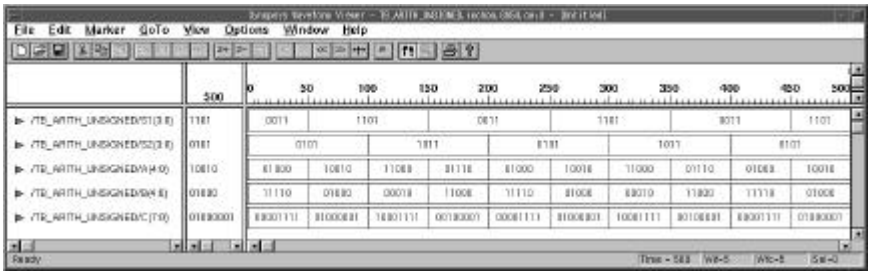
```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity TB_UNSIGNED is end TB_UNSIGNED;
architecture RTL of TB_UNSIGNED is
  signal s1, s2 : std_logic_vector(3 downto 0);
  signal a, b : std_logic_vector(4 downto 0);
  signal c, d : std_logic_vector(7 downto 0);
begin
  process
  begin
    s1 <= "0011";  s2 <= "0101";  wait for 50 ns;
    s1 <= "1101";  s2 <= "0101";  wait for 50 ns;
    s1 <= "1101";  s2 <= "1011";  wait for 50 ns;
    s1 <= "0011";  s2 <= "1011";  wait for 50 ns;
  end process;
  a <= ('0'&s1) + s2;  b <= ('0'&s1) - s2;
  c <= s1 * s2;  d <= s1 * "1000";
end RTL;
```

- 선언부 표현
  - Data Type: unsigned
  - Package: std_logic_arith
- VHDL 예제

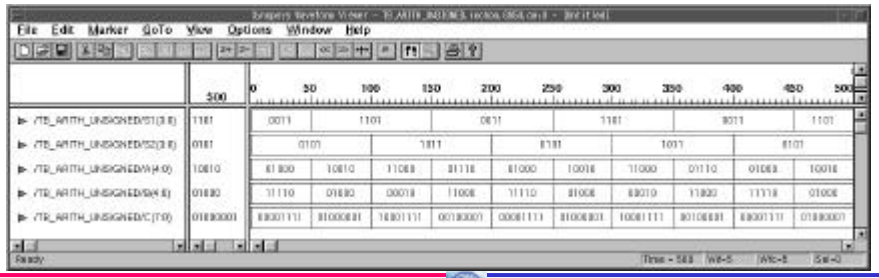
```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity TB_ARITH_UNSIGNED is
end TB_ARITH_UNSIGNED;
architecture RTL of TB_ARITH_UNSIGNED is
  signal s1, s2 : unsigned(3 downto 0);
  signal a, b : unsigned(4 downto 0);
  signal c : unsigned(7 downto 0);
begin
  process
  begin
    s1 <= "0011"; s2 <= "0101"; wait for 50 ns;
    s1 <= "1101"; s2 <= "0101"; wait for 50 ns;
    s1 <= "1101"; s2 <= "1011"; wait for 50 ns;
    s1 <= "0011"; s2 <= "1011"; wait for 50 ns;
  end process;
  a <= ('0'&s1) + s2;
  b <= ('0'&s1) - s2;
  c <= s1 * s2;
end RTL;
```

Unsigned Data 연산을 위한 Simulation 예제

- std_logic_vector 와 std_logic_unsigned package를 이용한 Simulation



- unsigned 와 std_logic_arith package를 이용한 simulation



Signed Data 연산을 위한 표현 예제

- 선언부 표현
  - Data Type: std_logic_vector
  - Package: std_logic_signed
- VHDL 예제

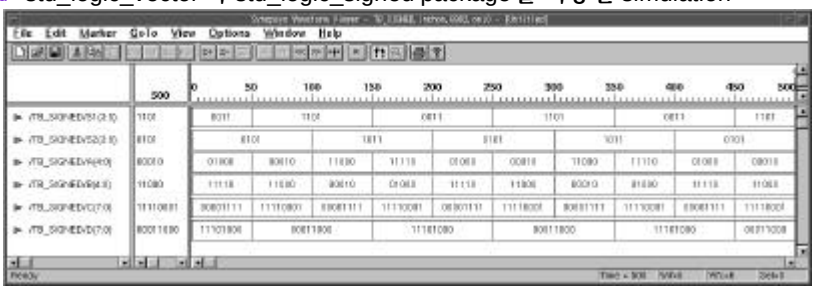
```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
entity TB_SIGNED is end TB_SIGNED;
architecture RTL of TB_SIGNED is
  signal s1, s2 : std_logic_vector(3 downto 0);
  signal a,b : std_logic_vector(4 downto 0);
  signal c, d : std_logic_vector(7 downto 0);
begin
  process
  begin
    s1 <= "0011"; s2 <= "0101"; wait for 50 ns;
    s1 <= "1101"; s2 <= "0101"; wait for 50 ns;
    s1 <= "1101"; s2 <= "1011"; wait for 50 ns;
    s1 <= "0011"; s2 <= "1011"; wait for 50 ns;
  end process;
  a <= (s1(3)&s1) + (s2(3)&s2);
  b <= (s1(3)&s1) - (s2(3)&s2);
  c <= s1 * s2;  d <= s1 * "1000";
end RTL;
```

- 선언부 표현
  - Data Type: signed
  - Package: std_logic_arith
- VHDL 예제

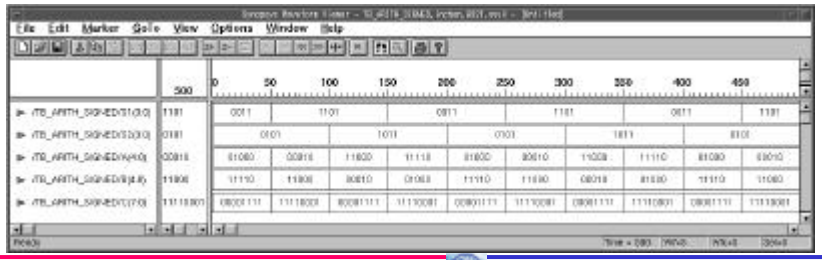
```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity TB_ARITH_SIGNED is
end TB_ARITH_SIGNED;
architecture RTL of TB_ARITH_SIGNED is
  signal s1, s2 : signed(3 downto 0);
  signal a, b : signed(4 downto 0);
  signal c : signed(7 downto 0);
begin
  process
  begin
    s1 <= "0011";  s2 <= "0101";  wait for 50 ns;
    s1 <= "1101";  s2 <= "0101";  wait for 50 ns;
    s1 <= "1101";  s2 <= "1011";  wait for 50 ns;
    s1 <= "0011";  s2 <= "1011";  wait for 50 ns;
  end process;
  a <= (s1(3)&s1) + (s2(3)&s2);
  b <= (s1(3)&s1) - (s2(3)& s2);
  c <= s1 * s2;
end RTL;
```

Signed Data 연산을 위한 Simulation 예제

- std_logic_vector 와 std_logic_signed package를 이용한 simulation



- signed 와 std_logic_arith package를 이용한 simulation



Subprogram 종류와 Function Subprogram의 정의

- Subprogram 에 대한 분류
  - Function
  - Procedure
- Subprogram 의 표현 영역
  - Package 선언부
  - Package 몸체부
  - Architecture 선언부
  - Process 선언부
- Function에 대한 Syntax 정의
  - Package 선언부에서의 표현

```
function name (formal_parameter_list)
  return data_type is
```

  - Package 선언부 이외에서의 표현

```
function name (formal_parameter_list)
  return data_type is
  -- declarative statements
begin
  -- sequential statements
  -- including return expression
end name ;
```

- Parameter 사용의 유의 사항
  - Data Mode
    - 모두 in 이다.
  - Data Object
    - 표기 경우 : constant 또는 signal
    - 표기 않은 경우 : constant
  - Formal Name과 Actual Name의 관계
    - signal의 경우 : signal만 가능
    - constant의 경우: signal,constant,variable 가능
- Function의 표현 영역과 Visibility
  - Package 선언부에 표현된 경우
    - Package 외부에서 이용 가능
  - Package Body에 표현된 경우
    - 해당 Package Body에서만 이용 가능
  - Architecture 선언부에 표현된 경우
    - 해당 Architecture에서만 이용 가능
  - Process 선언부에 표현된 경우
    - 해당 Process에서만 이용 가능

Function Subprogram에 대한 예제와 이용

- Function Subprogram 예제

```
package MYPACK is
  function MAX (A, B: in std_logic_vector)
        return std_logic_vector;
  function EQUAL (A, B : in std_logic_vector)
        return  boolean;
end;
package body MYPACK is
   function MAX (A, B: in std_logic_vector)
        return std_logic_vector is
   begin
    if A > B then
      return (A);
     else
        return (B);
      end if;
   end MAX ;
   function EQAUL(A, B : in std_logic_vector)
      return boolean is
   begin
      return (A=B);
   end EQUAL;
end MYPACK;
```

- Function Subprogram 의 이용 예제

```
library IEEE; use IEEE.std_logic_1164.all;
use work.MYPACK.all;
entity EX is
   port (D1, D2: in
            std_logic_vector(3 downto 0);
     DATA1, DATA2: in
            std_logic_vector(7 downto 0);
     Q: out std_logic_vector(3 downto 0);
     DATA_OUT: out boolean);
end EX;
architecture RTL of EX is
begin
  Q <= MAX(d1, d2);
        --Concurrent function call
  process (DATA1, DATA2)
  begin
    DATA_OUT <= EQUAL(DATA1, DATA2);
        --Sequential function call
  end process;
end RTL;
```


Procedure Subprogram의 정의

- Procedure에 대한 Syntax 정의
  - Package 선언부에서의 표현

```
procedure name (formal_parameter_list) ;
```

  - Package 선언부 이외에서의 표현

```
procedure name (formal_parameter_list) is
  -- declarative statements
begin
  -- sequential statements
end name ;
```

- Parameter 사용의 유의 사항
  - Data Mode
    - in, out, inout 모두 가능하다.
    - 표기 않는 경우
      - in mode로 약속됨
  - Data Class
    - Mode가 in이고, 표기하는 경우
      - constant, signal, variable 모두 가능하다.
    - Mode가 in이고, 표기 않는 경우
      - constant로 약속됨
    - Mode가 inout 또는 out이고, 표기하는 경우
      - variable, signal이 가능하다.
    - Mode가 inout 또는 out 이고, 표기 않는 경우
      - variable로 약속됨
  - Formal Name과 Actual Name의 관계
    - signal의 경우: signal만 가능
    - variable의 경우: variable만 가능
    - constant의 경우: signal,constant,variable 가능
- Procedure의 표현 영역과 Visibility
  - Package 선언부에 표현된 경우
    - Package 외부에서 이용 가능
  - Package Body에 표현된 경우
    - 해당 Package Body에서만 이용 가능
  - Architecture 선언부에 표현된 경우
    - 해당 Architecture에서만 이용 가능
  - Process 선언부에 표현된 경우
    - 해당 Process에서만 이용 가능

Procedure Subprogram에 대한 예제와 이용(1)

✍ Procedure 선언 예제

```
package MYPACK is
  procedure COMPUTE (A, B: in integer;
     MEAN, MAX: out integer);
end MYPACK;
package body MYPACK is
  procedure COMPUTE (A, B: in integer;
     MEAN, MAX: out integer) is
  begin
    MEAN := (A+B)/2;
    if A> B then
      MAX := A;
    else
      MAX := B;
    end if;
  end COMPUTE;
end MYPACK;
```

✍ Procedure에 대한 잘못된 사용 예제

```
use work.MYPACK.all;
entity EX is
  port (D1, D2, D3, D4: in integer;
        Q1, Q2, Q3, Q4: out integer);
architecture RTL1 of EX is
begin
  COMPUTE(D1, D2, Q1, Q2);
  --오류 발생, Q1, Q2 not variables
  --Concurrent procedure call
  process(D3, D4)
    variable A, B: integer;
  begin
    COMPUTE(D3, D4, A, B);
    --정상적인 동작, A and B variables
    --Sequential procedure call
    Q3 <= A;
    Q4 <= B;
  end process;
end RTL1;
```


Procedure Subprogram에 대한 예제와 이용(2)

✍ Procedure 선언 예제

```
package MYPACK is
  procedure COMPUTE (A, B: in integer;
     signal MEAN, MAX: out integer);
end MYPACK;
package body MYPACK is
  procedure COMPUTE (A, B: in integer;
     signal MEAN, MAX: out integer) is
  begin
    MEAN <= (A+B)/2;
    if A> B then
      MAX <= A;
    else
      MAX <= B;
    end if;
  end COMPUTE;
end MYPACK;
```

✍ Procedure에 대한 올바른 사용 예제

```
use work.MYPACK.all;
entity EX is
  port (D1, D2, D3, D4: in integer;
        Q1, Q2, Q3, Q4: out integer);
architecture RTL2 of EX is
begin
  COMPUTE(D1, D2, Q1, Q2);
  --정상적인 동작, Q1 and Q2 signals
  --Concurrent procedure call
  process(D3, D4)
  begin
    COMPUTE(D3, D4, Q3, Q4);
    --정상적인 동작, Q3 and Q4 signals
    --Sequential procedure call
  end process;
end RTL2;
```

Procedure Subprogram에 대한 예제와 이용(3)

```
library IEEE; use IEEE.std_logic_1164.ALL;
package EXAMPLE is
  procedure RCA (A, B : in std_logic_vector;
    CIN : in std_logic; SUM : out std_logic_vector;
    COUT : out std_logic);
end EXAMPLE;
package body EXAMPLE is
 function XOR3 (A, B, C : in std_logic) return
    std_logic is
 begin
    return (A xor B xor C);
 end XOR3;
 procedure RCA (A, B : in std_logic_vector;
    CIN : in std_logic; SUM : out std_logic_vector;
    COUT : out std_logic) is variable C:
    std_logic_vector((A'high-A'low+1) downto 0);
 begin
    C(0) := CIN;
    for I in 0 to (A'high-A'low) loop
      SUM(I+SUM'low) :=
        XOR3(A(I+A'low), B(I+B'low), C(I));
      C(I+1) := (A(I+A'low) and (I+B'low))
        or (C(I) and (A(I+A'low) or B(I+B'low)));
    end loop;
   COUT := C(C'high);
 end RCA;
end EXAMPLE;
```

```
library IEEE; use IEEE.std_logic_1164.ALL;
use work.EXAMPLE.all;
entity ADDER_TEST is
 port ( A, B: in
          std_logic_vector(15 downto 0);
        CIN : in std_logic;
        SUM : out
          std_logic_vector(15 downto 0);
        COUT : out std_logic);
end ADDER_TEST;
architecture RTL of ADDER_TEST is
begin
 process (A,B,CIN)
    variable TMP_SUM:
      std_logic_vector(SUM'range);
    variable TMP_COUT : std_logic;
 begin
    RCA(A, B, CIN, TMP_SUM, TMP_COUT);
    SUM <= TMP_SUM;
    COUT <= TMP_COUT;
  end process;
end RTL;
```


Operator Overload에 대한 개념 정의와 효용성

- 사용의 필요성과 정의 형식
  - 사용자가 정의하는 임의의 data type에 대하여 표준화된 연산자를 사용하고자 하는 경우
  - 연산자에 대한 overload를 위한 function을 작성하여야 함
  - Function의 이름에 대응하는 연산자를 " " 사이에 표기 하여야 한다.
    - " 연산자 "
- 정의 예제
  - Synopsys와 Vendor가 제공하는 Package에 대부분의 경우가 정의되어 있으며, 본 예제는 일부임

```
function "+" (L: std_logic_vector,
              R: std_logic_vector)
    return std_logic_vector;
function "+" (L: std_logic_vector,
              R: integer)
    return std_logic_vector;
```

- VHDL Compiler의 동작 개념
  - Compile을 행하는 VHDL Code에 + Operator의 사용이 요구되는 경우에 Compiler는 Operand의 Data Type이 모두 일치하는 Function을 찾는다.
  - Type이 모두 일치하는 Function이 있으면 해당 Function Algorithm에 따라 동작을 수행하며, 일치하는 Function이 없는 경우는 Analysis에 따른 Error Message를 출력한다.
  - 일치하는 Function이 복수 개 있는 경우는 마지막으로 일치한 Function Algorithm에 따라 동작한다.
- C <= A + B 예제에 대한 이해
  - A, B가 std_logic_vector인 경우
  - A는 std_logic_vector이고, B는 integer인 경우
- 효용성에 대한 고찰
  - 동일한 의미의 연산자에 대한 Function Name을 하나로 정의함으로써 사용자의 편리성 부여

Subprogram Overload에 대한 개념 정의와 효용성

- 사용의 필요성
  - Subprogram의 기본적인 기능은 동일하나, Formal Parameter 또는 Return Value에 대한 Data Type이 다른 경우
  - 위의 Subprogram을 동일한 이름으로 정의하여 사용
- 정의 예제

```
library IEEE;
use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
package EXAMPLE is
   function CONV_2COMP
    (A : std_logic_vector; K : integer)
   return std_logic_vector;
   function CONV_2COMP
    (A : integer;    K : integer)
   return std_logic_vector;
end EXAMPLE;
```

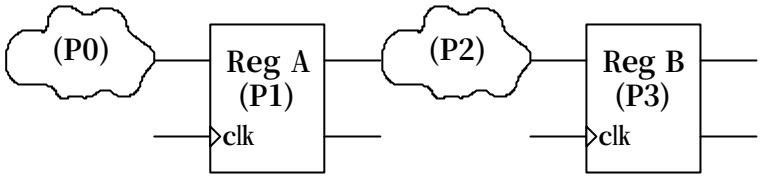
```
package body EXAMPLE is
   function CONV_2COMP
    (A : std_logic_vector;  K : integer)
      return std_logic_vector is
   begin
    return (not (A(K-1) downto 0)+1);
   end;
 --
   function CONV_2COMP
    (A : integer;  K : integer)
      return std_logic_vector is
   begin
    return (not
      conv_std_logic_vector(A,K)+1);
   end;
end EXAMPLE;
```

- VHDL Compiler의 동작 이해
  - B <= CONV_2COMP(A, 8); 로 표현되는 구문을 이해하자.
    - A가 std_logic_vector 인 경우
    - A가 integer 인 경우

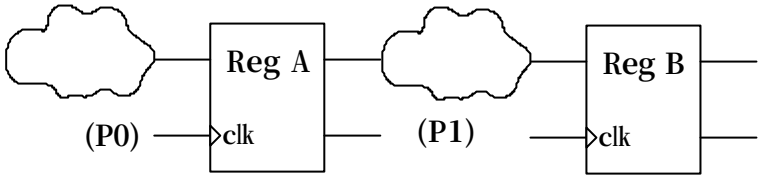
Chap 3. Digital System의 VHDL Modeling 이해

Digital System과 RTL 회로도 분석 모형

- 분석 모형-1
  - 4개의 Process 구문을 사용하여 표현



- 분석 모형-2
  - 2개의 Process 구문을 사용하여 표현



Digital System의 VHDL Modeling Template

- H/W Model에 대한 VHDL Modeling
  - Entity + Architecture 로 표현
- Entity Unit에 대한 표현
  - Visibility에 대한 부여
    - 9-value Visibility 부여
    - Arithmetic Operator Visibility 부여
    - 사용자 정의 Package에 대한 Visibility 부여
  - 입,출력 신호에 대한 표현
    - Block Diagram 구성
    - Generic, Port Data의 구분
    - Data의 Identifier, Mode, Type의 결정
- Architecture Unit에 대한 표현
  - 동작 알고리즘에 대한 분석
  - 조합논리 회로 블록의 추출
  - 동기 신호 종류에 따른 순서 논리 회로 블록의 추출과 분류
    - 동기 신호 및 초기화 신호가 같으면 하나의 블록으로 분류

- Basic Modeling Template

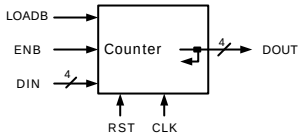
```
library IEEE; use IEEE.std_logic_1164.ALL;
entity name is
   port ( 입력 및 출력 신호에 대한 정의 구문 );
end name;
architecture architecture_name of name is
    요구되는 다양한 선언 (예: signal);
begin
 -- 동기 신호가 있는 블록의 표현
 sync : process (clock, reset등 동기 요구 신호)
 begin
     VHDL statements for state elements
 end process sync;
-- (중간 생략)
-- 조합 논리 회로 블록의 표현
 comb : process (블록의 모든 입력신호)
    요구되는 다양한 선언 (예: variable)
 begin
    조합논리회로 알고리즘을 표현하는 구문
 end process comb;
end architecture_name;
```

4-Bit Up-Counter에 대한 VHDL Modeling

- Counter 동작 표현 Table

RST	CLK	ENB	LOADB	DOUT
0	X	X	X	"0000"
1	↓	1	0	DIN
1	↓	1	1	DOUT+1

- Block Diagram에 대한 추출

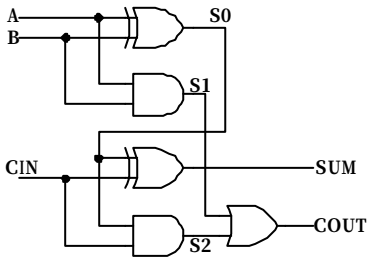


- VHDL 표현

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity COUNTER is
  port ( RST,CLK,LOADB,ENB: in std_logic;
      DIN : in std_logic_vector(3 downto 0);
      DOUT : out std_logic_vector(3 downto 0));
end COUNTER;
architecture RTL of COUNTER is
  signal T1,T2: std_logic_vector(3 downto 0);
begin
  A: process (RST, CLK) begin
     if RST = '0' then
       T1 <= (others => '0');
     elsif CLK'event and CLK = '1' then
       if ENB = '1' then
         T1 <= T2;
       end if;
     end if;
  end process;
  --
  B: process (LOADB, DIN, T1) begin
      if LOADB='0' then
        T2 <= DIN;
      else
        T2 <= T1 + 1;
      end if;
  end process;
  --
  DOUT <= T1;
end RTL;
```


조합 논리 회로 블록에 대한 알고리즘의 분석 형태

- 알고리즘 분석 형태
  - Gate-Level의 논리 회로도 표현
  - Boolean Equation 표현
  - 진리치 표에 의한 표현
  - 기능표에 의한 표현
  - Shift 연산 기능에 의한 Data 변환 표현
  - Arithmetic 및 Logic 연산 기능의 표현
  - 3-state Buffer 회로의 요구
- 전가산기의 회로도 표현



- 표현에 대한 고찰
  - 각 Gate의 동작을 Logic Operator를 사용하여 표현
  - Wire에 대응하는 Signal의 선언 요구
- Signal Assign 구문에 의한 표현

```
library IEEE; use IEEE.std_logic_1164.ALL;
entity ADDER is
  port (A, B, CIN : in std_logic;
        SUM, COUT : out std_logic);
end ADDER;
architecture RTL1 of ADDER is
  -- Wire에 대응하는 Signal의 선언
  signal S0, S1, S2 : std_logic;
begin
  --각 Gate를 Logical Operator와 함께 표현
  S0 <= A xor B;
  S1 <= A and B;
  SUM <= S0 xor CIN;
  S2 <= S0 and CIN;
  COUT <= S1 or S2;
end RTL1;
```

Boolean Equation으로 표현된 경우의 VHDL 표현

- 전가산기에 대한 Equation 표현
  - $sum = \bar{a}\bar{b}c_{in} + \bar{a}b\bar{c}_{in} + a\bar{b}\bar{c}_{in} + abc_{in}$
  - $cout = \bar{a}bc_{in} + a\bar{b}c_{in} + ab\bar{c}_{in} + abc_{in}$
- 표현에 대한 고찰
  - 각 Boolean Equation을 Data Flow 형태로 표현
- Signal Assign 구문에 의한 표현

```
architecture RTL2 of ADDER is
begin
  SUM <= (not A and not B and Cin) or
         (not A and B and not Cin) or
         (A and B and not Cin) or
         (A and B and Cin);
  COUT <= (not A and B and Cin) or
          (A and not B and Cin) or
          (A and B and not Cin) or
          (A and B and C);
end RTL2;
```

- Multi-Bit 연산이 요구되는 Entity Unit

```
entity MULTIBIT is
  port (A,B,C : IN
        std_logic_vector(3 DOWNTO 0);
    Z: OUT std_logic_vector(3 DOWNTO 0));
end MULTIBIT;
```

- 2가지 Architecture Unit에 대한 표현

```
architecture RTL2 of MULTIBIT is
begin
   --Multi-Bit 연산으로 표현하면 간결하다.
   Z <= A or (B nand C);
end RTL2;
--
architecture RTL1 of MULTBIT is
begin
   --각 Bit에 대응하는 Equation으로 표현
   Z(3) <= A(3) nor (B(3) nand C(3));
   Z(2) <= A(2) nor (B(2) nand C(2));
   Z(1) <= A(1) nor (B(1) nand C(1));
   Z(0) <= A(0) nor (B(0) nand C(0));
end RTL1;
```


Truth Table로 표현된 경우의 VHDL 표현

- 1-Bit 전가산기에 대한 Truth Table

a	b	cin	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- 표현에 대한 고찰
  - 1-Bit 크기의 입력 변수들을 & 연산자로 묶어 Variable 변수로 설정
  - 설정한 변수가 갖는 값을 Case 구문을 사용한 분류
  - 분류한 각 경우에 따라 출력 값 결정
    - Signal Assign 구문을 사용
    - Variable Assign 구문의 사용과 최종적인 출력 값 결정 방법
- Architecture Unit에 대한 표현

```
architecture RTL2 of ADDER is
begin
   process (A, B, CIN)
     variable TMP :
        std_logic_vector(2 downto 0);
   begin
     --입력 신호를 하나의 variable로 묶는다.
     TMP := A & B & CIN;
     case TMP is
        when "000" =>
           SUM <='0'; COUT <='0';
        when "001" | "010" | "100" =>
           SUM <='1';   COUT <='0';
        when "011" | "101" | "110" =>
           SUM <='0'; COUT <='1';
        when others =>
           SUM <='1'; COUT <='1';
     end case;
   end process;
end RTL2;
```

기능표로 표현된 경우의 VHDL 표현

- 4*2 Priority Encoder에 대한 기능표

입력변수				출력변수
a	b	c	d	enc_out
-	-	-	0	00
-	-	0	1	01
-	0	1	1	10
0	1	1	1	11

- 표현에 대한 고찰
  - Conditional Signal Assign 구문의 이용
  - Process 문의 IF 문을 이용한 표현
- Conditional Signal Assign 구문 이용
  - 출력이 하나인 경우에 표현이 간결

```
architecture RTL1 of PRIO_ENC is
begin
 ENC_OUT <= "00" when D = '0' else
            "01" when C = '0' else
            "10" when B = '0' else
            "11";
end RTL1;
```

- Process와 IF 구문을 이용
  - 출력 신호가 여러 개 인 경우에 조건을 1회씩 만 판단하므로 표현이 간결

```
architecture RTL2 of PRIO_ENC is
begin
  process (A,B,C,D)
  begin
    if D='0' then
      ENC_OUT <= "00";
    elsif C='0' then
      ENC_OUT <= "01";
    elsif B='0' then
      ENC_OUT <= "10";
    else
      ENC_OUT <= "11";
    end if;
  end process;
end RTL2;
```


기능표로 표현된 경우의 응용 예제 (1)

- Enable 신호를 갖는 4*1 MUX의 동작 특성에 대한 기능표

입력변수		출력변수
ENABLE	SEL	Y
0	00	I0
0	01	I1
0	10	I2
0	11	I3
1	--	"0000"

- Process와 Case 구문을 이용한 표현

```
process(SEL,ENABLE,I0,I1,I2,I3) begin
  if ENABLE='0' then
    case SEL is
      when "00" => Y <= I0;
      when "01" => Y <= I1;
      when "10" => Y <= I2;
      when others => Y <= I3;
    end case;
  else
    Y <= "0000";
  end if;
end process;
```

- Selected Signal Assign 구문 이용

```
architecture RTL2 of MUX41 is
 signal TMP : std_logic_vector(1 downto 0);
begin
  with SEL select
    TMP <= I0 when "00",
           I1 when "01",
           I2 when "10",
           I3 when others;
  with ENABLE select
    Y <= TMP when '0',
         "0000" when others;
end RTL2;
```

- 입력이 많은 MUX의 표현 고찰
  - 입력이 적은 MUX를 Component로 표현하고, Structural Modeling을 하는 방법
  - 입력 개수에 관계없이 process 구문과 case 구문을 이용하여 표현하고, 합성 Tool에 의한 분해를 행하는 방법

기능표로 표현된 경우의 응용 예제 (2)

- Enable 신호를 갖는 2*4 DEC의 동작 특성에 대한 기능표

Enable 신호 G	디코더의 입력 DATA	디코더의 출력 Y
0	00	"1110"
0	01	"1101"
0	10	"1011"
0	11	"0111"
1	--	"1111"

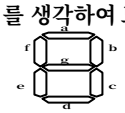
- Process 구문을 이용한 표현

```
process(G,DATA) begin
  if G='0' then
    case DATA is
      when "00" => Y <= "1110";
      when "01" => Y <= "1101";
      when "10" => Y <= "1011";
      when others => Y <= "0111";
    end case;
  else
    Y <= "1111";
  end if;
end process;
```

- Selected Signal Assign 구문 이용

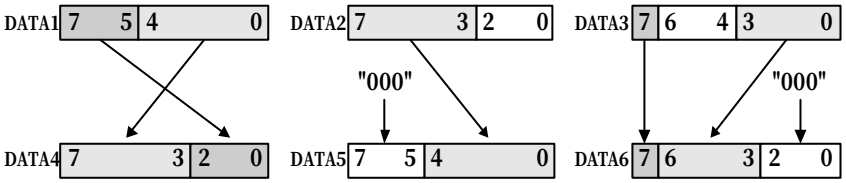
```
architecture RTL2 of EDECODER is
 signal TMP : std_logic_vector(2 downto 0);
begin
  TMP <= G & DATA;
  with TMP select
    Y <= "1110" when "000",
         "1101" when "001",
         "1011" when "010",
         "0111" when "011",
         "1111" when others;
end RTL2;
```

- BCD-to-7 Segment Decoder의 표현
  - 입력 조건에 Don't Care 존재하므로 VHDL 표현에 이용하면 면적을 최적화함에 유리하다.
  - 다양한 표현에 따른 합성 결과의 차이를 생각하여 보자.



Shift 기능에 의한 Data의 자리 변환 요구의 VHDL 표현

- Shift 기능에 의한 Data의 자리 변환 요구 예제와 분석



- Basic Signal Assign 구문과 & Operator를 이용하면 표현이 간결함
  - & Operator의 장점 : Shift 또는 Rotate 의미에 대한 표현이 매우 용이하다.

```
-- DATA4(7 downto 3) <= DATA1(4 downto 0);
-- DATA4(2 downto 0) <= DATA1(7 downto 5);
DATA4 <= DATA1(4 downto 0) & DATA1(7 downto 5);
-- DATA5(7 downto 5) <= "000";
-- DATA5(4 downto 0) <= DATA2(7 downto 3);
DATA5 <= "000" & DATA2(7 downto 3);
-- DATA6(7) <= DATA3(7);
-- DATA6(6 downto 3) <= DATA3(3 downto 0);
-- DATA6(2 downto 0) <= "000";
DATA6 <= DATA3(7) & DATA3(3 downto 0)& "000";
```

Arithmetic 연산의 기능이 요구되는 VHDL 표현의 예제

- 8가지 기능을 갖는 ALU 기능표

선택 신호	기능에 대한 표현
000	op1 + op2
001	op1 - 1
010	op1 - op2
011	op1 + 1
100	op1 and op2
101	op1 or op2
110	not op1
111	op1

- Entity Unit에 대한 표현

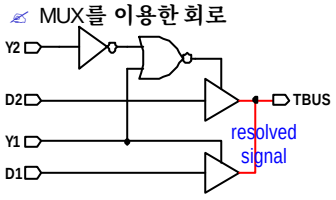
```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity SIM_ALU is
  port (OP1,OP2:in
          std_logic_vector(7 downto 0);
    CIN : in std_logic;
    SEL : in std_logic_vector(2 downto 0);
    ALU_OUT : out
          std_logic_vector(7 downto 0);
    ZERO_FLAG : out std_logic );
end SIM_ALU;
```

- Case 문을 이용한 기능 분류와 표현

```
architecture RTL of SIM_ALU is
begin
  process (OP1, OP2, CIN, SEL)
    variable TMP:std_logic_vector(7 downto 0);
  begin
    case SEL is
        when "000"=>TMP:= OP1+OP2+CIN;
        when "001"=>TMP:= OP1-1;
        when "010"=>TMP:= OP1-OP2+CIN;
        when "011"=>TMP:= OP1+1;
        when "100"=>TMP:= OP1 and OP2;
        when "101"=>TMP:= OP1 or OP2;
        when "110" =>TMP:= not OP1;
        when others =>TMP:= OP1;
    end case;
    ALU_OUT <= TMP;
    if TMP=0 then  ZERO_FLAG <= '1';
             else  ZERO_FLAG <= '0';
    end if;
  end process;
end RTL;
```


3-State Buffer 회로 요구에 대한 VHDL 표현

- Resolved Signal 정의
  - Data Source를 2개 이상 갖는 Wire
- Resolved Signal이 요구되는 회로의 구성 방법
  - 3-State Buffer를 이용한 회로
  - MUX를 이용한 회로



- Selected Signal Assign 구문의 이용

```
architecture RTL1 of TRISTATE is
begin
  TBUS<= D1 when (Y1='1')
          else 'Z';
  TBUS<= D2 when (Y1='0' and Y2='1')
          else 'Z';
end RTL1;
```

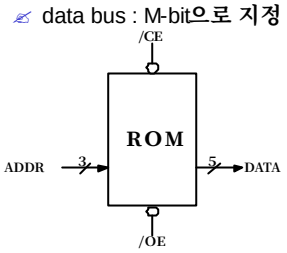
- Process 구문과 IF 문을 이용한 표현
  - 3-Buffer가 OFF인 경우는 'Z'를 할당

```
architecture RTL2 of TRISTATE is
begin
  process (Y1, D1) begin
    if (Y1='1') then
      TBUS<=D1;
    else
      TBUS<='Z';
    end if;
  end process;
  process (Y1, Y2, D2) begin
    if (Y1='0' and Y2='1') then
      TBUS<=D2;
    else
      TBUS<='Z';
    end if;
  end process;
end RTL2;
```

- Resolved Function 동작의 이해
  - '0' 또는 '1'이 한 경우만 존재하는 경우
  - 여러 가지 값이 존재하는 경우

기억 소자에 대한 VHDL 표현 : ROM 회로

- 요구되는 ROM 형태
  - address bus : N-bit으로 지정
  - data bus : M-bit으로 지정



- Entity Unit에 대한 표현

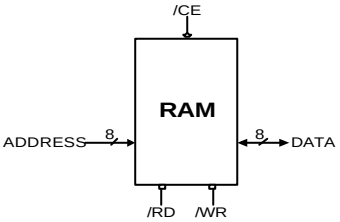
```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.ALL;
entity ROM is
  generic (N: integer := 3; M: integer := 5);
  port (
   ADDR: in std_logic_vector(N-1 downto 0);
   CE_N, OE_N : in std_logic;
   DATA : out std_logic_vector(M-1 downto 0));
end ROM;
```

- Architecture Unit의 고찰과 표현
  - 2차원 기억 공간의 선언이 요구
  - ROM의 초기값에 대한 표현이 요구
  - ROM address의 Integer로의 변환이 요구

```
architecture RTL of ROM is
  subtype WORD is
     std_logic_vector(M-1 downto 0);
  type TABLE is array(0 to 2**N-1) of WORD;
  -- type TABLE is array(0 to 2**N-1) of
  -- std_logic_vector(M-1 downto 0);
  constant ROM_DATA : TABLE :=
   TABLE'( "10101", "10000", "11111", "11011",
      "10001", "01100", "00101", "10011");
begin
   process (CE_N,OE_N,ADDR) begin
      if CE_N='0' and OE_N='0' then
         DATA <=
            ROM_DATA(conv_integer(ADDR));
      else
         DATA <= (others=>'Z');
      end if;
   end process;
end RTL;
```


기억 소자에 대한 VHDL 표현 : RAM 회로

- 요구되는 RAM 형태
  - address bus : 8-bit으로 지정
  - data bus : 양방향 8-bit으로 지정



- Entity Unit에 대한 표현

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.ALL;
entity RAM is
   port (
     ADDR : in std_logic_vector(7 downto 0);
     CE_N, RD_N, WR_N : in std_logic;
     DATA : inout
        std_logic_vector(7 downto 0));
end RAM;
```

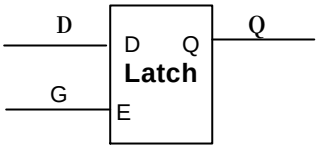
- Architecture Unit의 고찰과 표현
  - 2차원 기억 공간의 선언이 요구
  - RAM address의 Integer로의 변환이 요구
  - Read 이외의 경우는 Data Bus 모든 Bit에 'Z'를 출력하도록 요구 : Resolved Signal

```
architecture RTL of RAM is
    subtype RAM_WORD is
      std_logic_vector(7 downto 0);
    type RAM_TABLE is array(0 to 255) of
      RAM_WORD;
    signal RAM_DATA : RAM_TABLE;
begin
    process (CE_N,RD_N,WR_N,ADDR,DATA)
      variable TMP : integer;
    begin
      DATA <= "ZZZZZZZZ";
      TMP := conv_integer(ADDR);
      if CE_N='0' and RD_N='0' then
          DATA <= RAM_DATA(TMP);
      elsif CE_N='0' and WR_N='0' then
          RAM_DATA(TMP) <= DATA;
      end if;
    end process;
end RTL;
```

Latch 회로에 대한 VHDL 표현

- D-Latch의 기능 정의와 회로도

G	Q(t+1)
o	Q(t)
1	D



- Process 구문과 IF 문을 이용

```
architecture RTL1 of DLATCH1 is
begin
   process (G, D) begin
      if G='1' then
         Q <= D;
         QBAR <= not D;
      end if;
   end process;
end RTL1;
```

- Signal Assign 구문을 이용

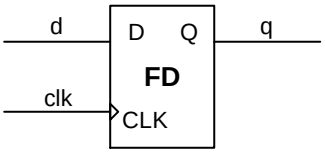
```
architecture RTL2 of DLATCH1 is
   signal TMP : std_logic;
begin
   TMP<=D when G='1' else TMP;
   Q <= TMP;
   QBAR <= not TMP;
end RTL2;
```

- Latch 생성에 대한 분석

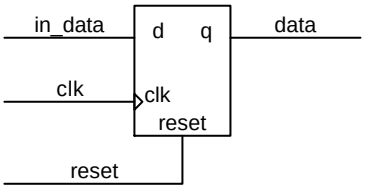
```
architecture RTL of EX is
 signal A, B, C, D, E:
        std_logic_vector(1 downto 0);
begin
   process (C, D, E, EN) begin
     if EN= '1' then
        A <= C;  B <= D;
     else
        A <= E;
     end if;
   end process;
 end RTL;
```


Flip-Flop 회로에 대한 VHDL 표현

- Synchronous D Flip-Flop 회로도



- Asynchronous D Flip-Flop 회로도



- 여러 가지 형태의 VHDL 표현

```
A1: process (CLK)  begin
   if (CLK='1' and CLK'event) then
      Q <= D;
   end if;
end process;
A2: process begin
   wait until CLK='1';
      Q <= D;
end process;
```

- VHDL 표현(Synopsys Model)

```
A1: process (RESET, CLK)  begin
   if RESET='1' then
      DATA <= '0';
   elsif (CLK='1' and CLK'event) then
      DATA <= IN_DATA;
   end if;
end process;
```

Flip-Flop 회로 생성 VHDL 표현에 대한 이해 (1)

✍ 분석 예제-1

```
architecture RTL1 of EX is
  signal A, B: std_logic_vector(3 downto 0);
begin
  process (CLK)
  begin
    if (CLK='1' AND CLK'EVENT) then
      if Q(3)/='1' then
        Q <= A + B;
      end if;
    end if;
  end process;
end RTL1;
```

✍ 생성되는 회로를 예측하여 보자.

✍ 분석 예제-2

```
architecture RTL2 of EX is
  signal A, B: std_logic_vector(3 downto 0);
begin
  process (CLK)
    variable INT:
      std_logic_vector(3 downto 0);
  begin
    if (CLK='1' AND CLK'EVENT) then
      if INT(3)/='1' then
        INT := A + B;
        Q <= INT;
      end if;
    end if;
  end process;
end RTL2;
```

✍ 생성되는 회로를 예측하여 보자.

Flip-Flop 회로 생성 VHDL 표현에 대한 이해 (2)

✍ 분석 예제-3

```
architecture RTL3 of EX is
begin
  process (RESET_N, CLK)
  begin
    if (RESET_N='0') then
      Q1 <= '0';  Q2 <= '0';
    elsif (CLK='1' AND CLK'EVENT) then
      Q1 <= A and B;
      Q2 <= C;
    end if;
  end process;
end RTL3;
```

✍ 생성되는 회로를 예측하여 보자.

✍ 분석 예제-4

```
architecture RTL4 of EX is
   signal Q1, I2: std_logic;
begin
  process (CLK)
  begin
    if (CLK='1' AND CLK'EVENT) then
      Q1 <= D;
      I2 <= Q1 and A;
    end if;
  end process;
  Q2 <= I2 and B;
end RTL4;
```

✍ 생성되는 회로를 예측하여 보자.

Variable 및 Signal Assignment에 따른 Flip-Flop 생성 이해

- Signal과 Shift-Register 회로 구성

--4개의 Flip-Flop으로 구성되는 회로

```
library IEEE; use IEEE.std_logic_1164.all;
entity SHREG4 is
 port ( D_IN, CLK : in std_logic;
        D_OUT: out std_logic);
end SHREG4;
architecture RTL of SHREG4 is
 signal S0, S1, S2: std_logic;
begin
 process (CLK) begin
   if (CLK='1' and CLK'event) then
     S0 <= D_IN;      S1 <= S0;
     S2 <= S1;        D_OUT <= S2;
   end if;
  end process;
end RTL;
```

- 논리 합성 회로를 구성하여 보자.

- Variable과 Shift-Register 회로 구성

--1개의 Flip-Flop으로 구성되는 회로

```
process (CLK)
  variable S0, S1, S2: std_logic;
begin
  if (CLK='1' and CLK'event) then
    S0 := D_IN;      S1 := S0;
    S2 := S1;        D_OUT <= S2;
  end if;
end process;
```

--4개의 Flip-Flop으로 구성되는 회로

```
process (CLK)
  variable S0, S1, S2: std_logic;
begin
  if (CLK='1' and CLK'event) then
    D_OUT <= S2;          S2 := S1;
    S1 := S0;             S0 := D_IN;
  end if;
end process;
```

- 논리 합성 회로를 각각 구성하여 보고, 차이점을 이해하고, 토의하자.

Register 회로에 대한 VHDL 표현

- Multi-Bits 기억 장치에 대한 표현
  - Register
  - Shift-Register
  - FIFO
  - Cycle Buffer

- 8-Bit Register의 동작 특성 정의

RESET	LD	ENABLE	CLK	REG
0	-	-	-	"00000000"
1	1	-	-	LDDATA
1	0	0	-	REG
1	0	1	?(x)	REG
1	0	1	?(o)	REGIN

```
library IEEE; use IEEE.std_logic_1164.ALL;
entity REGISTER1 is
  port (RESET, LD, ENABLE, CLK : in
          std_logic;
    REGIN : in std_logic_vector(7 downto 0);
    REG : out std_logic_vector(7 downto 0));
end REGISTER1;
architecture RTL2 of REGISTER1 is
begin
  process (RESET, LD, CLK)  begin
    if RESET='0' then
      REG <= (others =>'0');
    elsif LD='1' then
      REG <= LDDATA;
    elsif ENABLE='1' then
      if (CLK='0' and CLK'event) then
        REG <= REGIN;
      end if;
    end if;
  end process;
end RTL2;
```

Counter 회로에 대한 VHDL 표현

- 출력을 Out Mode로 설정한 경우

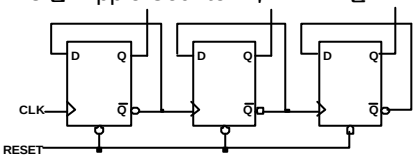
```
library IEEE;  use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity CNT8 is
  port (RESET,CLK: in std_logic;
        CNT: out std_logic_vector(2 downto 0));
end CNT8;
architecture RTL1 of CNT8 is
  signal TMP1,TMP2:
         std_logic_vector(2 downto 0);
begin
   CNT <= TMP2;
   TMP1 <= TMP2 + '1';
   process (RESET, CLK)
   begin
      if (RESET='0') then
         TMP2 <= "000";
      elsif (CLK='1' and CLK'event) then
         TMP2 <= TMP1;
      end if;
   end process;
end RTL1;
```

- 출력을 Buffer Mode로 설정한 경우
  - 일시적인 기억 장소 Signal에 대한 사용이 요구되지 않는다.
  - Buffer Mode 사용에 따른 주의점
  - 권장 표현 방식 : 전자의 표현 방식

```
library IEEE;  use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;
entity CNT8 is
   port (RESET,CLK: in std_logic;
   CNT: buffer std_logic_vector(2 downto 0));
end CNT8;
architecture RTL2 of CNT8 is
begin
   process (RESET, CLK)
   begin
      if (RESET='0') then
         CNT <= "000";
      elsif (CLK='1' and CLK'event) then
         CNT <= CNT + '1';
      end if;
   end process;
end RTL2;
```


비동기식 Counter 회로에 대한 VHDL 표현

- 8진 Ripple Counter 회로도 표현



- 표현에 대한 고찰
  - Generate 구문을 이용한 표현
- D Flip-Flop의 표현

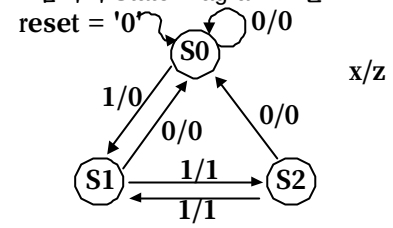
```
architecture RTL1 of DFFR is
    signal TMP_Q : std_logic;
begin
   Q <= TMP_Q;  QBAR <= not TMP_Q;
   process (RESET, CLK) begin
    if (RESET='0') then
      TMP_Q <= '0';
    elsif (CLK='1' and CLK'event) then
      TMP_Q <= D;
    end if;
   end process;
end RTL1;
```

- Generate 구문에 의한 표현

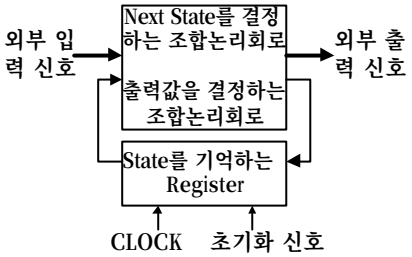
```
architecture RTL2 of RPLCNT8 is
  -- Clock을 위한 중간 Wire에 대한 선언
  signal CNTB: std_logic_vector(3 downto 0);
  -- D Flip-Flop에 대한 Component 선언
  component DFFR
    port (RESET, CLK, D : in std_logic;
          Q, QBAR : out std_logic);
  end component;
  -- Component Specification 표현
  for U: use entity DFFR work.DFFR(RTL1);
begin
   CNTB(0) <= CLK;
   GEN1: for I in 0 to 2 generate
      U: DFFR port map
                (CLK => CNTB(I),
                 RESET => RESET,
                 D => CNTB(I+1),
                 Q => CNT(I),
                 QBAR => CNTB(I+1));
   end generate GEN1;
end RTL2;
```

Finite State Machine 회로에 대한 표현 (1)

- 임의의 State Diagram 표현



- FSM의 Hardware 개념도
  - Moore Type Machine
  - Mealy Type Machine



- 표현에 대한 고찰
  - 조합논리회로 블록에 대한 표현
  - State Register 블록에 대한 표현
  - Symbolic State Name 에 대한 Enumeration 형의 Data Type 선언
  - 현재 및 다음 상태 변수에 대한 선언

- Symbolic State 인 경우의 Modeling

```
library IEEE; use IEEE.std_logic_1164.ALL;
entity MEALYFSM is
    port (RESET, CLK, X : in std_logic;
          Z : out std_logic);
end MEALYFSM;
architecture RTL1 of MEALYFSM is
 --Symbolic State Name을 위한 Type 선언
  type STATE is (S0, S1, S2);
 --State Signal에 대한 선언
  signal C_STATE, N_STATE : STATE;
begin
```


Finite State Machine 회로에 대한 표현 (2)

```
SYNC: process (RESET, CLK) begin
   if RESET='0' then
      C_STATE <= S0;
   elsif CLK='0' and CLK'event then
      C_STATE <= N_STATE;
   end if;
end process;
COMB : process (C_STATE, X) begin
   case C_STATE is
      when S0 =>
         Z <= '0';
         if X='0' then
            N_STATE <= S0;
         else
            N_STATE <= S1;
         end if;
      when S1 =>
         if X='0' then
            N_STATE <= S0; Z <= '0';
         else
            N_STATE <= S2; Z <= '1';
         end if;
      when others =>
         if X='0' then
            N_STATE <= S0; Z<= '0';
         else
            N_STATE <= S1; Z <= '1';
         end if;
   end case;
end process;
end RTL1;
```

- State Code Assignment 방법
  - Tool에 의하여 정의되는 경우
  - 사용자에 의하여 정의되는 경우
- Tool에 의하여 정의되는 경우
  - Symbolic State Name의 사용
  - Simulation의 경우
    - Code Assign을 하지 않음
    - 파형에 Symbolic Name이 출력
  - Synthesis의 경우
    - Binary Code로 Assign 함
    - 선언된 State Name의 순서에 따라 Binary Code로 Encoding 된다.
    - S0=(00), S1=(01), S2=(10)

Finite State Machine 회로에 대한 표현 (3)

- State Code가 사용자에 의하여 정의
  - State Code Assignment 방법-1

```
library IEEE; use IEEE.std_logic_1164.ALL;
entity MEALYFSM is
    port (RESET, CLK, X : in std_logic;
          Z : out std_logic);
end MEALYFSM;
architecture RTL2 of MEALYFSM is
   constant S0:  std_logic_vector(1 downto 0)
              := "00";
   constant S1:  std_logic_vector(1 downto 0)
              := "01";
   constant S2:  std_logic_vector(1 downto 0)
              := "11";
   signal C_STATE, N_STATE :
             std_logic_vector(1 downto 0);
begin
   --이하의 VHDL Code는 앞의 표현과 동일함
end RTL2;
```

  - State Code Assignment 방법-2

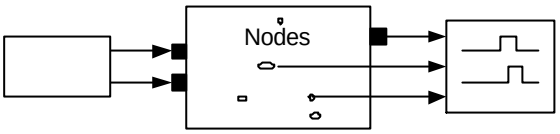
```
library IEEE; use IEEE.std_logic_1164.ALL;
entity MEALYFSM is
    port (RESET, CLK, X : in std_logic;
          Z : out std_logic);
end MEALYFSM;
architecture RTL3 of MEALYFSM is
  --Symbolic State Name을 위한 Type 선언문
  type STATE is (S0, S1, S2);
  attribute ENUM_ENCODING: string;
  attribute ENUM_ENCODING of STATE:
      type is "00 01 11";
  -- State Signal에 대한 선언문
  signal C_STATE, N_STATE : STATE;
begin
  --이하의 VHDL Code는 앞의 표현과 동일함
end RTL3;
```


Chap. 4 VHDL 검증을 위한 Test Bench Model의 이해

H/W 검증 개념과 S/W 검증 개념의 비교

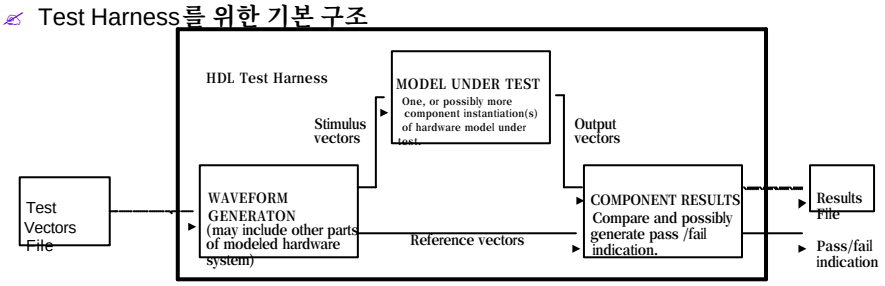
- H/W 검증 개념을 표현하는 Model

- S/W 검증 개념을 표현하는 Model



VHDL의 Test Bench Modeling의 개념

- Test Harness를 위한 기본 구조

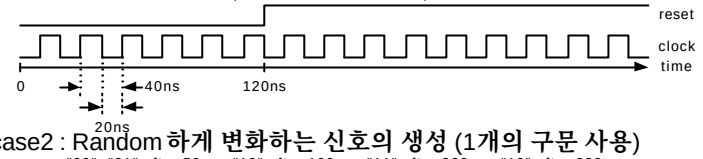


- Synthesis 를 목적으로 표현된 Hardware VHDL Model
- Simulation을 위한 입력 Data를 생성하는 Vector Generation VHDL Model
  - Test Harness 내부에서 생성
  - Array에 상수 형태로 저장된 Vector를 읽으면서 수행
  - 별도의 System File에 저장된 Vector를 읽으면서 수행
- Simulation Tool에 의한 계산 값과 설계자의 기대 값을 비교하는 VHDL Model
  - 출력된 파형에 대하여 설계자가 추적하여 Simulation 결과를 판단하는 방식
  - Simulation 결과를 File로 출력하는 방식
  - 기대 값과 계산 값을 assert 구문을 비교하여 Pass/Fail 만을 출력하는 방식

Test Vector 생성에 대한 3가지 모형과 VHDL 표현

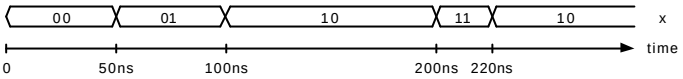
- Case1 : 초기화 신호 및 반복적 특성의 Clock 신호 생성

```
signal clock, reset : std_logic:='0';
begin
 clock <= not clock after 20 ns; reset <= '1' after 120 ns;
```



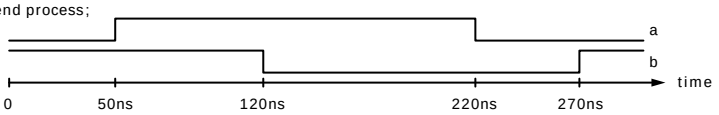
- case2 : Random 하게 변화하는 신호의 생성 (1개의 구문 사용)

```
x <= "00", "01" after 50 ns, "10" after 100 ns, "11" after 200 ns, "10" after 220 ns;
```



- case3 : Random 하게 변화하는 신호의 생성 (여러 개의 구문 사용)

```
process
begin
  a <= '0'; b <= '1'; wait for 50 ns;
  a <= '1'; wait for 70 ns;
  b <= '0'; wait for 100 ns;
  a <= '0'; wait for 50 ns;
end process;
```

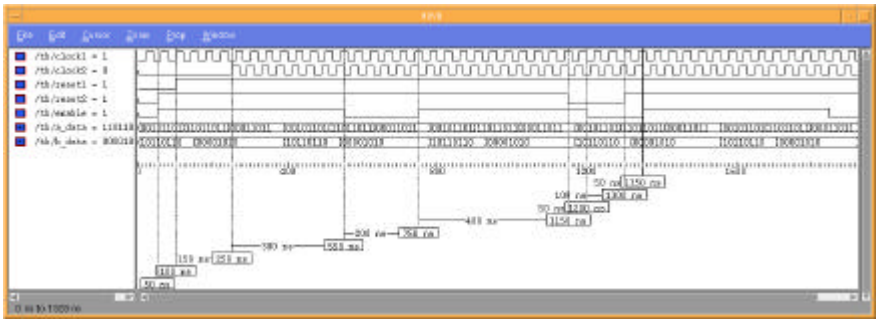


Test Vector 생성을 위한 Modeling 예제

```
--9-value visibility 부여
library IEEE; use IEEE.std_logic_1164.ALL;
--entity unit에 대한 표현
entity TB is
  port (CLOCK1 : buffer std_logic := '0';
        CLOCK2, RESET1, RESET2 : out std_logic ;
        ENABLE : out std_logic;
        A_DATA, B_DATA : out
            std_logic_vector(7 downto 0));
end TB;
--architecture unit에 대한 표현
architecture TB_A of TB is
begin
  CLOCK1 <= not CLOCK1 after 20 ns;
  RESET1 <= '0', '1' after 100 ns;
  process  begin
    if now = 0 ns then
      CLOCK2 <= '0';   wait for 250 ns;
    else
      CLOCK2 <= '1';   wait for 20 ns;
      CLOCK2 <= '0';   wait for 20 ns;
    end if;
  end process;
process  begin
    A_DATA <= "00101101";
    B_DATA <= "10110110";
    wait for 120 ns;
    A_DATA <= "11011011";  wait for 30 ns;
    B_DATA <= "00001010";  wait for 90 ns;
    A_DATA <= "00011011";  wait for 150 ns;
  end process;
  process  begin
    if now = 0 ns then
      RESET2 <= '0';  ENABLE <= '0';
        wait for 50 ns;
    else
      RESET2 <= '1';  ENABLE <= '1';
        wait for 300 ns;
      ENABLE <= '0' after 200 ns, '1' after 400 ns;
        wait for 800 ns;
    RESET2 <= '0', '1' after 150 ns; ENABLE <= '0'
      after 50 ns;
        wait for 200 ns;
    end if;
  end process;
end TB_A;
```

Test Vector 생성 Waveform과 의미 분석

- Test Vector 생성을 표현하는 Waveform

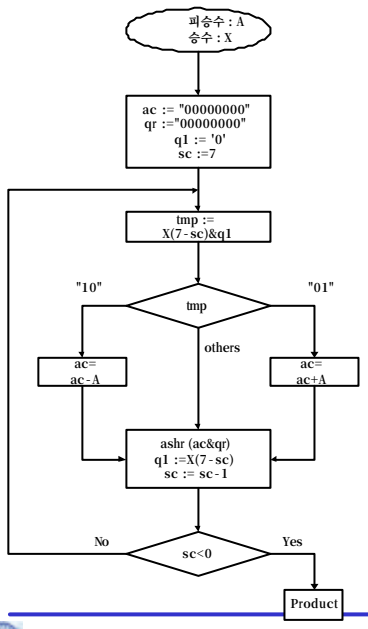


- 생성 신호에 대한 의미 분석
  - CLOCK1, CLOCK2와 같이 주기성을 갖는 신호
  - RESET1, RESET2와 같이 변화의 빈도가 매우 적은 특징을 갖는 신호
  - A_DATA, B_DATA와 같이 변화의 빈도가 많은 특징을 갖는 신호
    - 보편적인 제어 신호와 Data 신호가 해당
    - Process 구문을 사용한 표현이 적합함

설계-1 : Booth Multiplier에 대한 개념 설계

- Booth Multiplier를 위한 Algorithm

A		1	0	1	1			-5	
X	x	1	1	0	1			-3	
Y		0	/1	1	/1			recorded multiplier	
Add -A		0	1	0	1				
Shift		0	0	1	0	1			
Add A	+	1	0	1	1				
		1	1	0	1	1			
Shift		1	1	1	0	1	1		
Add -A	+	0	1	0	1				
		0	0	1	1	1	1		
Shift		0	0	0	1	1	1	1	
Shift		0	0	0	0	1	1	1	1



설계-1 : Booth Multiplier에 대한 VHDL 설계

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.numeric_std.ALL;
--
entity BOOTH is
  generic (N : integer := 8);
  port (RESET, CLOCK, LOAD : in  std_logic;
        MULTIPLICAND, MULTIPLIER : in
            std_logic_vector(N-1 downto 0);
        PRODUCT : out
            std_logic_vector(2*N-1 downto 0));
end BOOTH;
--
architecture RTL of BOOTH is
  signal Q1       : std_logic;
  signal AC, BR  : std_logic_vector(N-1 downto 0);
  signal QR       : std_logic_vector(N-1 downto 0);
  signal SC       : integer;begin
  process (RESET, CLOCK)
    variable TMP_AC  : std_logic_vector(N-1 downto 0);
  begin
    if RESET='0' then
      BR <= (OTHERS => '0');
      QR <= (OTHERS => '0');
      AC <= (OTHERS => '0');
      Q1 <= '0';
      SC <= N;
      PRODUCT <= (OTHERS => '0');
    elsif (CLOCK='1' AND CLOCK'event) then
      if LOAD='1' then
        BR <= MULTIPLICAND;
        QR <= MULTIPLIER;
        AC <= (OTHERS => '0');
        Q1 <= '0';
        SC <= N;
        PRODUCT <= (OTHERS => '0');
      else
        if (SC=0) then
          PRODUCT <= AC & QR;
        else
          if QR(0)='0' and Q1='1' then
            TMP_AC := AC + BR;
          elsif QR(0)='1' and Q1='0' then
            TMP_AC := AC + not BR + '1';
          else
            TMP_AC := AC;
          end if;
        end if;
        Q1 <= QR(0);
        QR <= TMP_AC(0) & QR(N-1 downto 1);
        AC <= TMP_AC(N-1) & TMP_AC(N-1 downto 1);
        SC <= SC - 1;
      end if;
    end if;
  end process;
end RTL;
--
```


설계-1 : 결과값을 파형으로 분석하는 Test Bench 예

Vector 파형 생성기의 표현

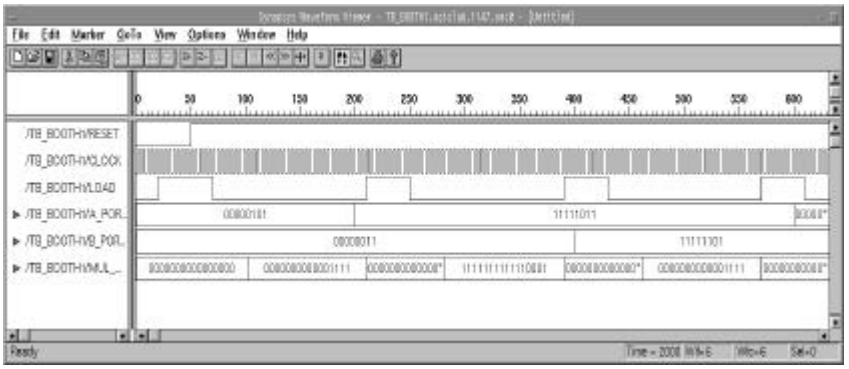
```
library IEEE; use IEEE.std_logic_1164.ALL;
library IEEE.numeric_std.ALL;
entity VEC_GEN is
  port ( CLOCK, RESET, LOAD : out std_logic;
    A_PORT, B_PORT : out std_logic_vector(7 downto 0));
end VEC_GEN;
architecture VEC_GEN_A of VEC_GEN is
  signal CLOCK_S : std_logic := '0';
begin
    RESET <= '1' after 50 ns;
    CLOCK_S <= not CLOCK_S after 2 ns;
    CLOCK <= CLOCK_S;
    LOAD <= '0', '1' after 20 ns, '0' after 70 ns,
              '1' after 210 ns, '0' after 250 ns,
              '1' after 390 ns, '0' after 430 ns,
              '1' after 570 ns, '0' after 610 ns;
    A_PORT_DRV : process  begin
       A_PORT <= "00000101";    wait for 200 NS;
       A_PORT <= "11111011";    wait for 400 NS;
    end process;
    B_PORT_DRV : process  begin
      B_PORT <= "00000011";      wait for 400 NS;
      B_PORT <= "11111101";      wait for 400 NS;
    end process;
end VEC_GEN_A;
```

파형 생성기를 갖는 Test Bench

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.numeric_std.ALL;
entity TB_BOOTH1 is end TB_BOOTH1;
architecture TB_BOOTH1_A of TB_BOOTH1 is
  component BOOTH
    generic (n : integer := 8);
    port (RESET, CLOCK, LOAD : in std_logic;
          MULTIPLICAND, MULTIPLIER : in
             std_logic_vector(n-1 downto 0);
          PRODUCT : out
             std_logic_vector(2*n-1 downto 0));
  end component;
  -- Vector Generation Block에 대한 선언
  signal A_PORT, B_PORT: std_logic_vector(7 downto 0);
  signal MUL_OUT : std_logic_vector(15 downto 0);
  signal RESET, CLOCK, LOAD : std_logic;
  for u0 : BOOTH use entity work.BOOTH(RTL);
  for u1 : VEC_GEN use entity
      work.VEC_GEN(VEC_GEN_A);
begin
  u0 : BOOTH port map(RESET, CLOCK, LOAD,
           A_PORT, B_PORT, MUL_OUT);
  u1 : VEC_GEN port map ( CLOCK, RESET, LOAD,
           A_PORT, B_PORT );
end TB_BOOTH1_A;
```

설계-1 : Test Bench VHDL 표현에 대한 Simulation

- VHDL에 대한 Functional Simulation 파형



- Simulation 예제-1 : (+5) = $(00000101)_2$, (+3)=$(00000011)_2$
  결과 = $(00000000\_00001111)_2$
- Simulation 예제-2 : (-5) = $(11111011)_2$, (+3)=$(00000011)_2$
  결과 = $(11111111\_11110001)_2$
- Simulation 예제-3 : (-5) = $(11111011)_2$, (-3)=$(11111101)_2$
- Simulation 예제-4 : (+5) = $(00000101)_2$, (-3)=$(11111101)_2$

설계-1 : ASSERT문을 이용한 Test Bench 예

- 파형 비교를 위한 Compare 블록

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.numeric_std.ALL;
entity COMP is
  port ( RESET, LOAD : in std_logic;
         A_PORT, B_PORT : in signed(7 downto 0);
         MUL_OUT : in signed(15 downto 0));
end COMP;
architecture COMP_A of COMP is
  constant CLOCK_PERIOD : time := 5 ns;
begin
 process  begin
   wait on RESET, LOAD;
   if RESET='0' then
     wait for (CLOCK_PERIOD);
     assert (MUL_OUT=0) report "assert violation-1";
   elsif LOAD='1' then
     wait for (CLOCK_PERIOD);
     assert (MUL_OUT=0) report "assert violation-2";
   else
     wait until (LOAD='0' and load'event);
     wait on MUL_OUT;
     assert (MUL_OUT=(A_PORT*B_PORT))
          report "assert violation-3";
   end if;
 end process;
end COMP_A;
```

- 파형 비교 블록을 갖는 Test Bench

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.numeric_std.ALL;
entity TB_BOOTH2 is end TB_BOOTH2;
architecture TB_BOOTH2_A of TB_BOOTH2 is
 -- Compare에 대한 Component 선언이외는 보충 할 것
 component COMP
   port ( RESET, LOAD : in std_logic;
          A_PORT, B_PORT : in signed(7 downto 0);
          MUL_OUT : in signed(15 downto 0));
 end component;
 signal A_PORT, B_PORT: signed(7downto 0);
 signal MUL_OUT : signed(15 downto 0);
 signal RESET, CLOCK, LOAD: std_logic;
 for u0 : BOOTH use entity work.BOOTH(RTL);
 for u1 : VEC_GEN use entity
          work.VEC_GEN(VEC_GEN_A);
 for u2 : COMP use entity work.COMP(COMP_A);
begin
 u0 : BOOTH port map(RESET, CLOCK, LOAD,
          A_PORT, B_PORT, MUL_OUT);
 u1 : VEC_GEN port map ( CLOCK, RESET, LOAD,
          A_PORT, B_PORT );
 u2 : COMP port map ( RESET, LOAD, A_PORT,
          B_PORT, MUL_OUT );
end TB_BOOTH2_A;
```

설계-1 : File 처리 구문을 갖는 Test Bench 예

✍ Test Bench 표현

```
library IEEE; use IEEE.std_logic_1164.ALL;
use IEEE.std_logic_signed.ALL;
use IEEE.std_logic_textio.ALL; use std.textio.ALL;
entity TB_BOOTH3 is end TB_BOOTH3;
architecture TB_BOOTH3_A of TB_BOOTH3 is
  component BOOTH
    generic (n : integer := 8);
    port ( RESET, CLOCK, LOAD : in std_logic;
           MULTIPLICAND, MULTIPLIER : in
            std_logic_vector(n-1 downto 0);
           PRODUCT: out std_logic_vector(2*n-1 downto
    0));
  end component;
  signal A_PORT, B_PORT: std_logic_vector(7 downto
    0);
  signal MUL_OUT: std_logic_vector(15 downto 0);
  signal RESET, CLOCK, LOAD: std_logic :='0';
 for u0 : BOOTH use entity work.BOOTH(RTL);
begin
 RESET <= '1' after 50 ns;
 CLOCK <= not CLOCK after 2 ns;
 u0 : BOOTH port map(RESET, CLOCK, LOAD,
 A_PORT, B_PORT, MUL_OUT);
```

```
process
   file VECTOR_FILE : text is in "vectors";
   variable LOAD_V : std_logic;
   variable A_PORT_V, B_PORT_V :
      std_logic_vector(7 downto 0);
   variable DELAY : time;
   variable L  : line;
   variable GOOD_NUMBER : boolean;
 begin
    while not endfile(VECTOR_FILE) loop
     readline ( VECTOR_FILE, L );
     read (L, DELAY, GOOD => GOOD_NUMBER);
      next when not GOOD_NUMBER;
      READ(l, LOAD_V);
      READ(l, A_PORT_V);
      READ(l, B_PORT_V);
      WAIT FOR delay - now;
      LOAD <= LOAD_V;
      A_PORT <= A_PORT_V;
      B_PORT <= B_PORT_V;
    end loop;
    wait;
 end process;
```


설계-1 : File 처리 구문을 갖는 Test Bench 예

```
  process
    file TABLE_FILE : text is out "results";
    variable TABLE_LINE : line;
  begin
    wait on MUL_OUT;
    write ( TABLE_LINE, now, RIGHT, 10, ns );
    write ( TABLE_LINE, ' ' );
    write ( TABLE_LINE, MUL_OUT(15) );
    write ( TABLE_LINE, MUL_OUT(14) );
    write ( TABLE_LINE, MUL_OUT(13) );
    write ( TABLE_LINE, MUL_OUT(12) );
    write ( TABLE_LINE, MUL_OUT(11) );
    write ( TABLE_LINE, MUL_OUT(10) );
    write ( TABLE_LINE, MUL_OUT(9) );
    write ( TABLE_LINE, MUL_OUT(8) );
    write ( TABLE_LINE, MUL_OUT(7) );
    write ( TABLE_LINE, MUL_OUT(6) );
    write ( TABLE_LINE, MUL_OUT(5) );
    write ( TABLE_LINE, MUL_OUT(4) );
    write ( TABLE_LINE, MUL_OUT(3) );
    write ( TABLE_LINE, MUL_OUT(2) );
    write ( TABLE_LINE, MUL_OUT(1) );
    write ( TABLE_LINE, MUL_OUT(0) );
    writeline ( TABLE_FILE, TABLE_LINE );
  end process;
end TB_BOOTH3_A;
```

✍ Stimulus Vector File : “vectors”

```
 0 NS      0 00000101 00000011
 20 NS     1 00000101 00000011
 70 NS     0 00000101 00000011
200 NS     0 11111011 00000011
210 NS     1 11111011 00000011
250 NS     0 11111011 00000011
390 NS     1 11111011 00000011
400 NS     1 11111011 11111101
430 NS     0 11111011 11111101
570 NS     1 11111011 11111101
600 NS     1 00000101 11111101
610 NS     0 00000101 11111101
800 NS     0 11111011 00000011
```

✍ 출력 File : “results”

```
   0 NS    0000000000000000
 106 NS    0000000000001111
 214 NS    0000000000000000
 286 NS    1111111111110001
 394 NS    0000000000000000
 466 NS    0000000000001111
 574 NS    0000000000000000
 646 NS    1111111111110001
```

설계-1 : FPGA를 이용한 Prototyping과 검증

- Booth Multiplier에 대하여 FPGA를 이용한 검증 Board의 예