제1절 프로세스(process)문

■ 프로세스(process)문
- 동작적 표현(behavioral description)방식, 하드웨어 시스템을 모듈로 기술하는데 용이.
  - 시스템 각 모듈의 병행처리를 담당
  - process문 내부는 순차처리
  - 복잡한 알고리즘을 용이하게 구현, 원활한 process사이의 통신을 담당

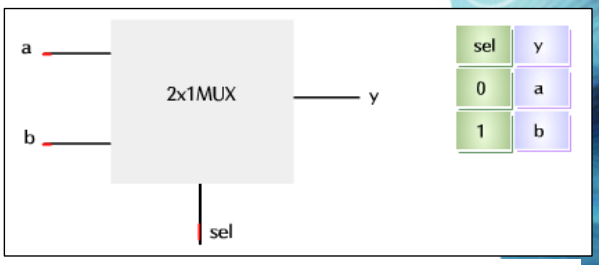
[형식 4.1] 프로세스문의 일반적인 구조

```
[프로세스-_레이블] process [(감지 신호)]
                    {선언문}
                   begin
                    {순차문}
                  end process [프로세스_레이블];
```

제1절 프로세스(process)문

[예제 4.1] 2x1 멀티플렉서

```
1   entity mux21 is
2       port(a, b, sel : in std_logic;
3                    y : out std_logic);
4   end mux21;
5   architecture sam of mux21 is
6     begin
7      mux_exam : process(a,b,sel)
8       begin
9         y <= a;
10        if (sel='1') then y <= b;
11        end if;
12     end process mux_exam;
13  end sam;
```



(a) 논리도 (b) 진리표

[그림 4.1] 2x1MUX.

제1절 프로세스(process)문

■ 대기문(wait statement)

▪ 대기문 : 프로세스문을 잠시 대기하기 위한 문장

- 프로세스문의 수행을 잠시 중단하고, 대기시켜 동작과 멈춤 사이의 변화를 제어
- 감지신호가 있을 경우, 감지신호의 변화가 있을 때까지 프로세스를 중지
- 감지신호가 없을 경우, 대기문을 사용하여 프로세스의 동작과 멈춤을 제어

[예제 4.2] 대기문

```
1  wait for 10ns;       -- 10ns동안 기다린 후, 다음 문장 수행
2  wait until clk='1';  -- clk의 상승이 있은 후, 다음 문장 수행
3  wait on a, b;        -- a, b신호에 변화가 있으면 다음 문장 수행
4  wait;                -- 무조건 대기
```

[예제 4.3] 복합적인 대기문

```
1  wait on a, b until clk = '1';              -- a, b의 변화가 있고, clk이 '1'이 될 때까지 대기
2  wait on a, b until clk = '1' for 10ns; -- a, b의 변화가 있고, clk이 '1' 이 될 때까지 대기하되 10ns만 대기
```

[예제 4.4] 대기문의 활용

```
1  architecture sam of and2 is
2    begin
3     process          -- 프로세스에 감지신호가 없는 대신
4       begin
5        wait on a, b;  -- 대기문을 사용
6          y <= a and b;
7     end process;
8  end sam;
```

제1절 프로세스(process)문

▪ 조건문(conditional statement)
if문, **case**문, **loop**문, **exit**문, **next**문, **null**문, **return**문
▪ **if**문(if statement)
[형식 4.2] if문의 종류
① 일반적 if문

```
if (조건) then          -- (조건)이 참이면
  {문장1};              -- {문장1}을 수행
 else                   -- 그렇지 않으면
  {문장2};              -- {문장2}를 수행
end if;
```

② 다중 if 문

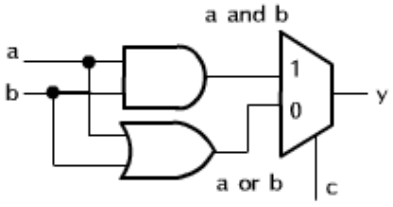
```
if (조건1) then         -- (조건1)이 참이면
   {문장1};             -- {문장1}을 수행
elsif (조건2) then      -- (조건1)이 거짓이고 (조건2)가 참이면
   {문장2};             -- {문장2}를 수행
     ..
else                    -- 모두가 참이 아니면
   {문장 n};            -- {문장 n}을 수행
end if;
```

③ 기억소자가 내포된 if문

```
if (조건) then          -- (조건)을 만족하면 {문장}을
    {문장};             -- 수행하고 그렇지 않으면
end if;                 -- 과거상태를 유지 (기억), else가 없음
```


제1절 프로세스(process)문

[예제 4.5] 일반적 **if**문의 활용



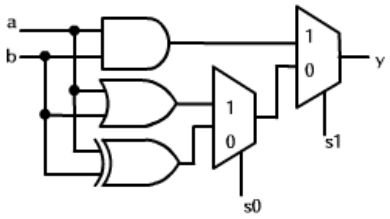
[그림 4.2] if문의 MUX등가회로

<VHDL 구문>

```
1   architecture sam1 of MUX_unit is
2      begin
3        process (a, b, c)          --감지신호가 a,b,c인 process문 선언
4          begin
5             if ( c = '1' ) then   --만일 MUX의 c가 참이면
6                   y <= a and b;  -- a and b를 y에 대입
7               else                -- 만일 c가 참이 아니면
8                   y <= a or b;   -- a or b를 y에 대입
9             end if ;
10         end process;
11     end sam1;
```

제1절 프로세스(process)문

[예제 4.6] 다중 **if**문의 활용



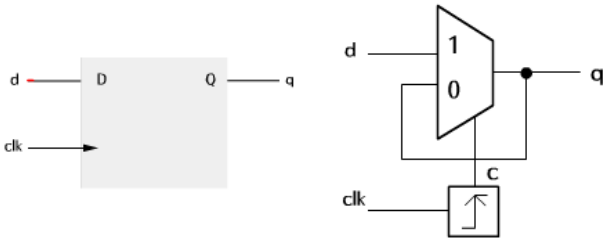
[그림 4.3] 다중 if문의 MUX등가회로

<VHDL 구문>

```
1  architecture sam2 of MUX_unit is
2        begin
3          process (a, b, s0, s1)      -- 감지신호가 a,b,s0,s1인 process선언
4            begin
5              if ( s1 = '1' ) then       -- MUX의 제어신호 s1이 참이면,
6                    y <= a and b;      -- a and b를 출력 y에 전달
7              elsif ( s0 = '1' ) then   -- s1이 참이 아니고, s0가 참이면,
8                    y <= a or b;       -- a or b를 출력 y에 전달
9              else                       -- 위의 두 조건이 모두 아니면,
10                   y <= a xor b ;     -- a xor b를 출력 y에 전달
11             end if;
12         end process ;
13  end sam2;
```


제1절 프로세스(process)문

[예제 4.7] **else** 없는 **if**문의 사용 예



[그림 4.4] D flip-flop과 MUX등가회로

<VHDL 구문>

```
1  architecture sam3 of MUX_unit is
2     begin
3        process (clk)              -- clk의 변화가 있을 때, process가 동작
4           begin
5              if clk'event and clk='1' then  -- event속성이 참이면,
6                 q <= d;                      -- d를 q에 대입
7              end if;                        -- else문이 없으므로 기억상태
8        end process;
9  end sam3;
```

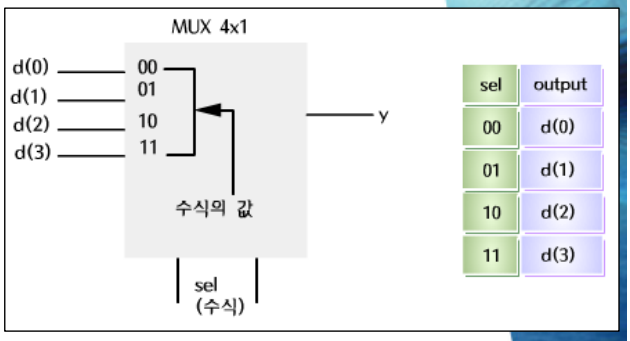
제1절 프로세스(process)문

▪ case문(case statement)
수식 값에 따라 문장을 선택하는 조건문, 수식 값은 정수형, 열거형 등의 자료형이 사용

[예제 4.8] case문의 활용

<VHDL 구문>

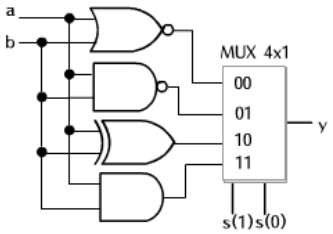
```
1 case sel is              -- sel값을 조사하여
2    when "00" =>          -- sel값이 "00"이면
3       y <= d(0);         -- y <= d(0) 문장을 수행하고
4    when "01" =>          -- sel값이 "01"이면
5       y <= d(1);         -- y <= d(1) 문장을 수행하고
6    when "10" =>          -- sel값이 "10"이면
7       y <= d(2);         -- y <= d(2) 문장을 수행하고
8    when others =>        -- sel값이 기타이면
9       y <= d(3);         -- y <= d(3) 문장을 수행
10 end case;
```



[그림 4.5] case문의 MUX 등가표현

제1절 프로세스(process)문

[예제 4.9] case문을 이용한 조합논리회로 설계



[그림 4.6] case문 활용의 MUX 등가회로

<VHDL 구문>

```
1  architecture sam4 of MUX_ca is
2   begin
3    process (s, a, b)              --감지신호가 s, a, b인 process선언
4     begin
5      case s is                    -- case문 선언
6       when "00" => y <= a nor b;   -- 조건이 "00"이면 "a nor b"를 y에 연결
7       when "01" => y <= a nand b; -- 조건이 "01"이면 "a nand b"를 y에 연결
8       when "10" => y <= a xor b;   -- 조건이 "10"이면 "a xor b"를 y에 연결
9       when others => y <= a and b; -- 기타 조건이면 "a and b"를 y에 연결
10      end case;
11   end process;
12  end sam4;
```

제1절 프로세스(process)문

▪ loop문(loop statement)

어떤 조건이 만족할 때까지 반복하는 문장의 순차문

♪ 종류 : **for ~ loop**문, **while ~ loop**문, **loop**문

[형식 4.3] loop문의 형식

① **for ~ loop**문

```
[레이블] : for (루프변수) in (변수범위) loop
            {순차처리문};        -- (변수범위: downto 혹은 to) 만큼 반복 수행
  end loop [레이블]
```

② **while ~ loop**문

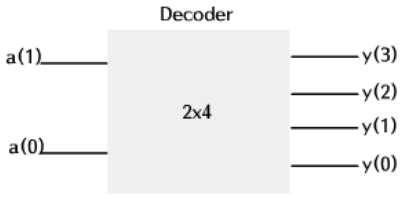
```
[레이블] : while (조건) loop
            {순차처리문};        -- (조건)이 참일 때까지 반복 수행
  end loop [레이블]
```

③ 단순 **loop**문

```
[레이블] :  loop
            {순차처리문};        -- 무한 반복 수행, loop을 나오기 위해
  end loop [레이블]              -- exit문이 필요
```


제1절 프로세스(process)문

[예제 4.10] **for ~ loop**문을 이용한 decoder의 설계



a(1)	a(0)	y(3)	y(2)	y(1)	y(0)
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

[그림 4.7] decoder 및 동작표

<VHDL구문>

```
1  architecture sam5 of decoder is
2  begin
3    process ( a )              -- 감지신호 a인 프로세스 선언
4    begin
5      loop1 : for k in 3 downto 0 loop  -- 변수가 k,변수범위
6              y(k) <= (a = k);          -- (3 downto 0)만큼 반복
7      end loop loop1;
8    end process;
9  end sam5;
```

제1절 프로세스(process)문

[예제 4.11] **while ~ loop**문의 활용 예

```
1 loop2 : while s <= 20 loop    -- "s <= 20" : s가 20보다 같거나 작다는 의미
2         s := s+1; t := t+s;  -- s값에 1을 더하여 s에 즉시 대입, s+ t값을t에 대입
3       end loop loop2;
```


제1절 프로세스(process)문

▪ 기타 제어문 : **exit**문, **next**문, **null**문

exit문/**next**문 : **loop**문의 반복을 제어하기 위한 문장

null문 : 아무런 수행 없이 다음 문장으로 넘겨주기 위한 문장

[형식 4.4] 기타 제어문의 형식

```
exit  [레이블] when (조건);    -- 조건이 참이면 해당 레이블의 loop를 빠져나감
next [레이블] when (조건);    -- 조건이 참이면 해당 loop의 처음을 수행
null                          -- 아무런 수행이 없는 문장
```

[예제 4.12] **for ~ loop**문 내 **exit**문의 활용

<VHDL 구문>

```
1  process (rst)
2  begin
3     if rst = '1' then                    -- reset = 1이면 for ~ loop문 수행
4  loop3:for x in add downto 0 loop -- 변수x, 변수범위"add downto 0"인 for ~ loop문
5        if x>40 then exit loop3;          -- x가 40 이상이면 if문 수행, loop3를 빠져나감
6        else
7           mem(x) <= (others => '0');  -- x가 40보다 작으면 수행
8        end if;
9     end loop loop3;
10 end process;
```

제1절 프로세스(process)문

[예제 4.13] **for ~ loop**문 내 **exit ~ when**문의 활용

<VHDL 구문>

```
1   process (rst)
2    begin
3     if rst = '1' then                    -- reset = 1이면 for ~ loop문 수행
4   loop3:for x in add downto 0 loop --변수x, 변수범위"add downto 0"인 for ~ loop문
5       exit loop3 when x>40;              -- x가 40 이상이면 loop3를 빠져나감
6           mem(x) <= (others => '0');  -- x가 40 이하이면 수행
7     end loop loop3;
8   end process;
```

[예제 4.14] **for ~ loop**문 내 **next**문의 활용

<VHDL 구문>

```
1   process (rst)
2    begin
3     if rst = '1' then                    -- reset = 1이면 for ~ loop문 수행
4  loop4 : for x in 8 downto 0 loop --변수x, 변수범위"8 downto 0"인 for ~ loop문
5       if x=5 then                        -- x=5이면,
6            next;                         -- loop4를 다시 시작, 5행, 6행을 건너 뜀
7           else
8            mem(x) <= (others => '0');  -- x=5가 이니면 수행
9       end if;
10       end loop loop4;
11   end process;
```


제1절 프로세스(process)문

[예제 4.15] **while ~ loop**문 내 **next**문의 활용

<VHDL 구문>

```
1   process (rst)
2     variable x : integer
3   begin
4       x := '0';
5     if rst = '1' then                    -- reset = 1이면 for ~ loop문 수행
6  loop5 : while x < 9 loop             -- while 조건 loop문
7       if x=5 then                        -- x=5이면,
8            next;                         -- loop5를 다시 시작, 5행, 6행을 건너 뜀
9           else
10           mem(x) <= (others => '0'); -- x=5가 이니면 수행
11            x := x + 1;
12       end if;
13      end loop loop5;
14  end process;
```

[예제 4.16] **null** 문

<VHDL 구문>

```
1  case sel is
2     when "00" =>              -- sel이 "00"이면,
3        y <= d(0) after 10ns; -- 10ns 후, y에 d(0)대입
4     when "01"|"10" =>         -- sel이 "01" 혹은 "10"이면, ("|"은 "혹은"의 의미)
5        y <= d(1) after 10ns; -- 10ns 후, y에 d(1) 대입
6     when others => null;    -- sel이 기타의 조건인 경우 아무런 동작이 없음
7  end case;
```

제1절 프로세스(process)문

▪ **return** 문

함수(function) 혹은 프로시저(procdeure) 등 부프로그램을 수행한 후, 호출한 문장으로 되돌아 갈 경우에 사용

[예제 4.17] **return**문의 활용(boolean을 bit로 되돌려 주는 변환함수)

<VHDL 구문>

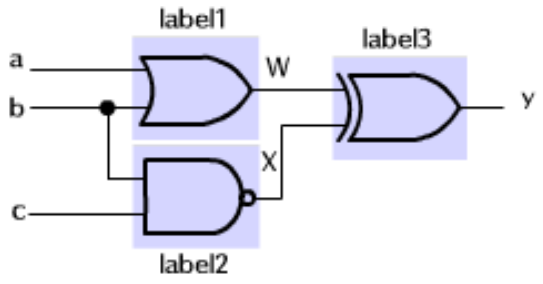
```
1  function bool_bit (a : boolean) return bit is -- 부울형 값 x를 bit로
2    begin                                       -- 되돌리라는 함수 선언
3      if a then        -- a값을 판단하여 참이면
4          return '1';  -- 값 '1'을 출력으로 되돌려 주고
5      else             -- 참이 아니면
6          return '0';  -- '0'을 출력으로 되돌려 줌
7      end if;
8  end bool_bit;
```


제1절 프로세스(process)문

▪ 다수의 프로세스(multiple process)

하나의 아키텍처 내에 다수의 process문이 존재

[예제 4.18] process문의 병행처리



[그림 4.8] process의 병행처리를 위한 조합논리

제1절 프로세스(process)문

<VHDL 구문>

```
1   architecture sample of logic is      -- process와 process간은 병행처리
2     signal w, x : std_logic;                  -- w, x를 선연결 정보인 신호로 선언
3     begin
4    label1 : process(a, b)                -- OR게이트 표현
5      begin
6      if(a='1')or(b='1') then w<='1';    -- process 내부는 순차문, w <= a or b 문과 동일
7      else    w <= '0';
8      end if;
9    end process;
10  label2 : process(b, c)                 -- NAND게이트 표현
11     begin
12      if (b='0')or(c='0') then x <='1';  -- process 내부는 순차문, x <= b nand c 와 동일
13      else   x <= '0';
14      end if;
15     end process;
16  label3 : process(w, x)                 -- XOR게이트 표현
17      begin
18       if (w=x) then y <='0';            -- process 내부는 순차문, y <= w xor x와 동일
19       else    y <= '1';
20       end if;
21   end process;
22  end  sample;
```


제2절 동작적 표현(behavioral representation)

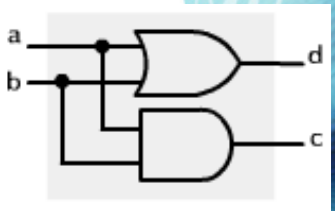
▪ 동작적 표현방식(behavioral description)
디지털 시스템의 내용을 설명하듯이 기술하는 것으로 순차처리문과 병행처리문으로 분류

▪ 병행처리문(concurrent statement)
ㅎ 디지털 시스템의 하드웨어 구조를 기술하는 VHDL 구문은 병행처리에 기반

[예제 4.19] 병행처리의 조합회로

<VHDL 구문>

```
1   architecture sam1 of comb_logic is    -- sam2와 등가 아키텍처 구문
2     begin
3       c <= a and b;       -- 병행처리 대입문에서는 순서와 무관
4       d <= a or b;
5   end sam1;
6  -- 병행처리 비교 아키텍처 구문
7   architecture sam2 of comb_logic is    -- sam1과 등가 아키텍처 구문
8     begin
9       d <= a or b;        -- 병행처리 대입문에서는 순서와 무관
10       c <= a and b;
11  end sam2;
```



[그림 4.9] 병행처리의 조합회로

제2절 동작적 표현(behavioral representation)

▪ 신호의 대입(signal assignment)

신호의 대입은 '<=' 기호를 사용, '<='의 오른쪽에서 왼쪽으로 신호가 대입

순차처리문의 대입은 process문 내에서 signal의 대입, 병행처리문의 대입은 아키텍처 내 signal의 대입

▪ 신호(signal)의 대입문

[예제 4.20] 신호의 대입문

```
w  <=  a;          -- w에 a의 값이 하드웨적으로 연결
x  <=  '0';        -- x가 상수 '0'으로 연결
y  <=  a and b;    -- y에 "a and b"의 값을 연결
```

signal이름 값

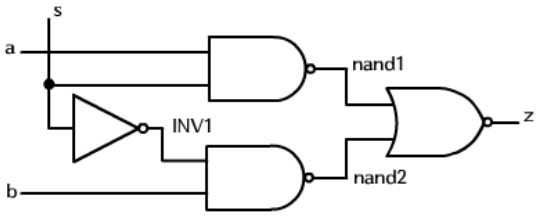
▪ 병행신호 대입문

[예제 4.21] 병행신호 대입문

```
1  architecture con of assign is
2     signal in1, in2 : std_logic;  -- in1, in2가 signal이라고 선언
3   begin
4     in1 <= a or b;          -- in1에 "a or b"의 값이 연결
5     in2 <= b or c;          -- in2에 "b or c"의 값이 연결
6     out <= in1 and in2;     -- out에 "(a or b) and (b or c)"의 값이 연결
7  end con;
```


제2절 동작적 표현(behavioral representation)

[예제 4.22] 2x1MUX를 통한 하드웨어의 병행처리



[그림 4.10] 2x1 MUX의 병행처리

<VHDL구문>

```
1  architecture comb of mux21 is
2     signal nand1, nand2, inv1 : std_logic;  -- nand1, nand2, inv1를 signal로 선언
3   begin
4     and1 <= a nand s;       -- 병행처리 대입문
5     and2 <= b nand inv1;    -- 병행처리 대입문
6     inv1 <= not s;          -- 병행처리 대입문
7       z <= and1 nor and2;   -- 병행처리 대입문
8  end comb;
```

제2절 동작적 표현(behavioral representation)

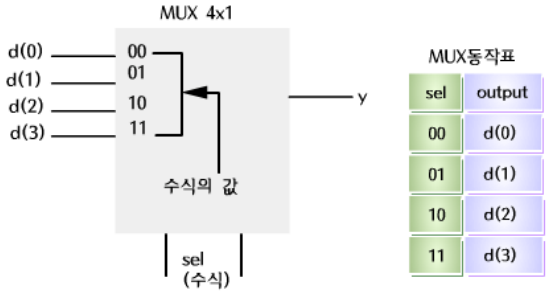
■ 조건적 병행 신호처리문(condional concurrent signal assignment)

▪ **when ~ else**문

[형식 4.5] **when ~ else**문의 형식

신호_이름 <= 값1 **when** (조건1) **else**
값2 **when** (조건2) **else**
. . . .
값n-1 **when** (조건n-1) **else**
값n;

[예제 4.26] **when ~ else**문 활용



[그림 4.14] 4x1MUX의 설계

제2절 동작적 표현(behavioral representation)

<VHDL구문>

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity mux4_1 is
4     port ( d : in std_logic_vector(3 downto 0);  -- 4bit data입력
5          sel : in std_logic_vector(1 downto 0);  -- 2bit 제어입력
6          y : out std_logic_vector(3 downto 0));  -- 4bit data출력
7   end mux4_1;
8   architecture sample of mux4_1 is
9     begin
10      y <= d(0) when sel = "00" else
11           d(1) when sel = "01" else
12           d(2) when sel = "10" else
13           d(3);
14   end sample;
```

제2절 동작적 표현(behavioral representation)

▪ 선택적 병행 신호처리문(selected concurrent signal assignment)

▪ **with ~ select**문

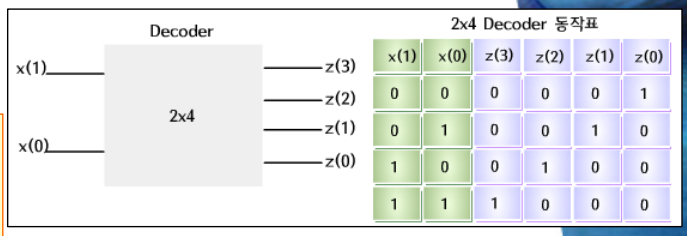
[형식 4.6] **with ~ select**문의 형식

```
신호_이름 <= 값1 when 선택 신호값1
             값2 when 선택 신호값2
          . . . .
             값n-1 when 선택 신호값n-1
             값n  when others;
```

[예제 4.27] **with ~ select**문의 활용

<VHDL 구문>

```
1  with x select
2      z <= "0001" when "00"
3           "0010" when "01"
4           "0100" when "10"
5           "1000" when others;
```

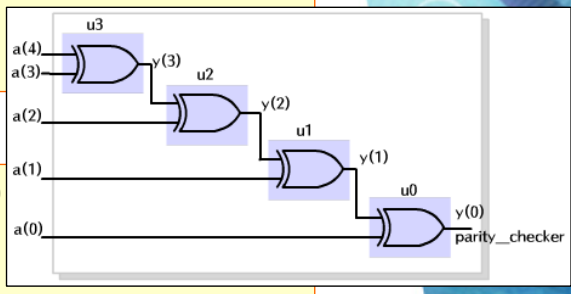


[그림 4.15] 2x4 decoder설계

제3절 자료흐름적 표현(dataflow description)

▪ 자료흐름적 표현(data flow description)
입력에서 출력으로의 자료의 흐름, 동작적 표현과 구조적 표현의 중간 단계로 자료가 입력에서 출력으로 흘러가는 시스템의 기능을 표현, 부울대수, 논리연산자 및 함수 등으로 표현하며, 병행신호 처리문으로 기술

▪ 자료흐름표현의 활용



[그림 4.16] 패리티 생성기

[예제 4.28] 자료흐름 표현의 활용(패리티 생성기)

<VHDL구문>

```
1   entity pari_check is
2     port ( a : in std_logic_vector(4 downto 0);
3          parity_checker : out std_logic);
4   end pari_check;
5   architecture data_flow of pari_check is
6     begin
7     parity_checker <= a(4) xor a(3) xor a(2) xor a(1) xor a(0);
8   end data_flow;
```

제4절 구조적 표현(structural description)

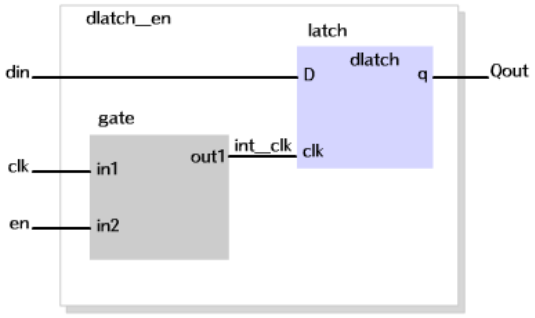
■ 구조적 표현(structural description)
구조적 표현은 하드웨어의 내부적 설계에 가장 가까운 표현, 이미 설계된 부품을 이용하여 이들 상호간의 연결을 통한 시스템의 설계

▪ 직접 개체화(direct instantiation)

[형식 4.7] 엔티티 개체의 표현

개체_이름 : **entity** work. 엔티티_이름(아키텍처_이름)

[예제 4.29] 엔티티 개체화



[그림 4.17] 엔티티 개체를 위한 논리도

제4절 구조적 표현(structural description)

<VHDL 구문>

```
1   entity dlatch_en is          -- 외부단자 인터페이스 선언
2     port(din, clk, en : in bit;
3                qout : out bit);
4   end dlatch_en;
5   architecture structure of dlatch_en is
6     signal int_clk : bit;      -- 내부연결정보 int_clk를 선언
7    begin
8    gate : entity work.and2(beh)        -- 부품 AND게이트 and2 개체화
9      port map (in1 => clk, in2 => en,  -- 부품 and2 단자 연결
10              out1 => int_clk);        -- 이름을 지정하여 결합
11    latch : entity work.dlatch(beh)    -- 부품 dlatch의 개체화
12      port map (d => din, clk => int_clk, -- 부품 dlatch의 단자 연결
13               q => qout);                -- 이름 결합
14  end structure;
```

제4절 구조적 표현(structural description)

■ 부품 개체(사례)화(component instantiation)

▪ **component**문

[형식 4.8] **component**문의 선언 형식

```
component 부품_이름
  [generic(범용어_리스트);]   -- 일반화(generic)
  [port(포트_리스트);]
end component;
```

[예제 4.30] **component**문의 활용

```
1  architecture structure of count is
2     component dff          -- 부품 dff를 사례화
3        port (rst, clk, d : in std_logic;
4                    q : out std_logic);
5     end component diff;
6   begin ・・・
```

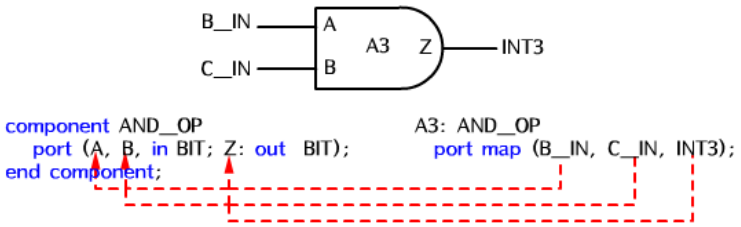

제4절 구조적 표현(structural description)

[형식 4.9] 부품 개체화(component instantiation) 형식

레이블 : 부품_이름 [**port map**(결합_리스트);
generic map(결합_리스트);]

ᔕ결합_리스트 : 위치결합, 이름결합

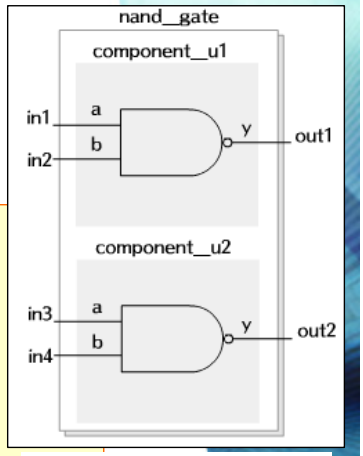
① 위치결합: component의 형식적 이름과 실제이름의 결합을 port신호가 나열된 순서대로 연결



② 이름결합 : component의 형식적 이름과 실제이름의 결합을 직접 지정하여 연결

제4절 구조적 표현(structural description)

[예제 4.31] **component**문 선언과 부품 사례화



[그림 4.18] NAND_gate

<VHDL 구문>

```
1   entity nand_gate is
2     port(in1, in2, in3, in4 : in std_logic;
3                out1, out2 : out std_logic);
4   end nand_gate;
5   architecture example of nand_gate is
6     component nand2  -- 아키텍처의 begin앞에 부품 nand2 선언
7       port (a, b : in std_logic; y : out std_logic);
8     end component;
9    begin
10    u1 : nand2 port map (in1, in2, out1);                -- 단자의 위치결합
11    u2 : nand2 port map (a=>in3, b=>in4, y=>out2); -- 단자의 이름결합
12   end example;
```


제4절 구조적 표현(structural description)

■ 생성(generate)문

component를 반복적으로 생성하여 사용하기 위한 문장

[형식 4.10] 생성(generate)문의 형식

① 단순 반복 생성

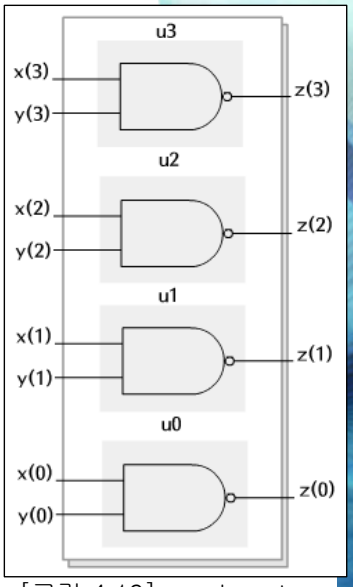
```
레이블: for (변수) in (변수범위) generate
          {병행처리문} ;        -- 병행처리문을 변수범위 만큼 반복
end generate [레이블];
```

② 조건 반복 생성

```
레이블: if (조건) generate   -- 조건을 만족할 때 반복
          {병행처리문} ;
end generate [레이블];
```

제4절 구조적 표현(structural description)

[예제 4.32] 단순반복 생성문의 활용



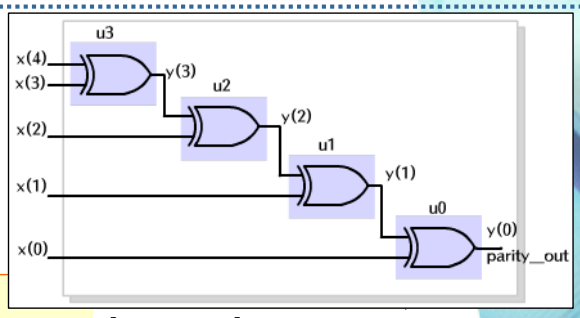
[그림 4.19] nand_system

<VHDL구문>

```
1   architecture example of nand_system is
2      component nand2
3         port (a, b : in std_logic; y : out std_logic);
4      end component;
5    begin
6      g1 : for i 3 downto 0 generate          -- 4개의 게이트를
7      ux : nand2 port map (a(i), b(i), y(i)); -- 반복 생성
8   end example;
```


제4절 구조적 표현(structural description)

[예제 4.33] 조건 반복 생성문의 활용



[그림 4.20] parity_checker

<VHDL 구문>

```
1   architecture example of xor_system is
2      signal y : bit_vector(3 downto 0);  -- 내부의 선연결 정보를 신호로 선언
3    begin
4      g1: for i in 3 downto 0 generate    -- 4번의 단순반복을 하되,
5           g2: if i = 3 generate          -- i=3의 조건을 만족하면 u3을 수행
6           u3 : xor2 port map (x(i+1), x(i), y(i));
7           end generate g2;
8           g3: if i < 3 generate          -- i<3의 조건을 만족하면 ux를 수행
9            ux : xor2 port map (y(i+1), x(i), y(i));
10           end generate g3;
11     end generate g1;
12              parity_out <= y(0);
13   end example;
```