

# 합성을 위한 VHDL Coding Style 2

국일호

goodkook@csvlsi.kyunghee.ac.kr

http://www.csvlsi.kyunghee.ac.kr

지난 회에 이어 합성을 위한 VHDL Coding Style 의 두 번째 연재로서 간단한 기능을 갖는 중소형 집적회로 급(SSI/MSI:Small-Medium Scale IC)의 조합 회로 들인 Multiplexer, Encoder 와 Decoder, Priority Encoder, 그리고 RAM, PLA 등의 적절한 기술 방법과 합성의 예를 살펴 보도록 한다. 다음 연재에서는 State machine, 연산회로, 그리고 VHDL 로 설계할 때 최적의 결과를 얻기 위한 방법등에 대하여 살펴보기로 한다.

## 1. Multiplexer

MUX (Multiplexer)는 n-입력을 선택하여 1 개의 출력을 내는 디지털 논리 회로이다. 2 개 입력인 경우 간단하게 IF~THEN ELSE~END IF 문에 의하여 기술 될 수 있으나 3 개 이상의 입력인 경우 CASE~WHEN~END CASE 구문을 사용하여 선택의 병렬성을 유지하는 것이 좋다. 3 개 이상의 입력을 갖는 MUX 를 기술할 때 IF~THEN ELSE~END IF 로 기술한 경우 자칫 하면 Priority Encoder 가 될 수 있으므로 피하는 것이 좋다. [예제 1]은 순차구문에 의한 MUX 의 기술이다. MUX 에서 선택신호가 n 비트 인 경우  $2^{**n}$  개의 입력을 갖는다. 그러나 경우에 따라 2 이하의 입력 개수를 갖는 설계의 경우 OTHERS 조건을 반드시 지정해주어야 하며 그렇지 않을 경우 합성의 결과 " 불필요한 래치"가 생성되므로 주의 한다. OTHERS 조건에서 상수를 할당 하는 경우에 비하여 '-' (don't care)를 할당하는 것이 좀더 우수한 최적화 합성 결과를 얻을 수 있다.

[예제 1] 순차 구문에 의한 Multiplexer 의 기술

```
PROCESS(sel_n, sel_one)
BEGIN
    -- 2-to-1 select
    IF (sel_one='0') THEN
        out_one <= in_sig0;
    ELSE
        out_one <= in_sig1;
    END IF;

    -- N-to-1 select
    CASE sel_n IS
```

```

        WHEN "000" =>
            out_sig <= in_sig0;
        WHEN "001" =>
            out_sig <= in_sig1;
        WHEN "010" =>
            out_sig <= in_sig2;
        WHEN "110" =>
            out_sig <= in_sig3;
        WHEN "011" =>
            out_sig <= in_sig4;
        WHEN OTHERS =>
            out_sig <= (others => '-');
    END CASE;
END PROCESS;

```

[예제 2]는 병렬 구문에 의한 MUX의 표현이다. 병렬 구문에서는 WHEN~ELSE 혹은 WITH~SELECT 구문을 이용한다. 이 경우에도 OTHERS의 조건 사용에 주의하도록 한다.

[예제 2] 병렬구문에 의한 Multiplexer의 기술

```

-- 2-to-1 select
out_one <=
    in_sig0 WHEN sel_one='0' ELSE
    out_one <= in_sig1;

-- N-to-1 select
WITH sel_n SELECT
    Out_sig <=
        in_sig0 WHEN "000",
        in_sig1 WHEN "001",
        in_sig2 WHEN "010",
        in_sig3 WHEN "110",
        in_sig4 WHEN "011",
        (OTHERS=>'-') WHEN OTHERS;

```

## 2. Encoder 와 Decoder

Encoder 와 Decoder 의 기술 방법은 MUX 의 기술과 같다. 다만 상수할당을 하게 되며 모든 경우의 할당이 있게 되므로 OTHERS 조건이 필요 없다. [예제 3]은 3-to-8 decoder 인 TTL 138 의 순차구문에 의한 기술이며, 합성 결과는 그림 1 과 같다.

[예제 3] 순차구문에 의한 3-of-8 Decoder (TTL 138)

```
ENTITY ttl138 IS
PORT (
    a2, a1, a0 : IN std_logic;
    cs1, cs2, cs3 : IN std_logic;
    y0, y1, y2, y3, y4, y5, y6, y7 : out std_logic);
END ttl138;
```

```
ARCHITECTURE behave OF ttl138 IS
SIGNAL Y : std_logic_vector(7 DOWNTO 0);
BEGIN
PROCESS(a2,a1,a0,cs1,cs2,cs3)
VARIABLE a : std_logic_vector(2 DOWNTO 0);
VARIABLE sel : std_logic;
BEGIN
    a := a2 & a1 & a0;
    Sel <= cs1 and ((NOT cs2) AND (NOT cs3));

    IF (sel='1') THEN
        CASE a IS
            WHEN "000" =>
                Y <= "11111110";
            WHEN "001" =>
                Y <= "11111101";
            WHEN "010" =>
                Y <= "11111011";
            WHEN "011" =>
                Y <= "11110111";
            WHEN "100" =>
                Y <= "11101111";
            WHEN "101" =>
                Y <= "11011111";
            WHEN "110" =>
                Y <= "10111111";
            WHEN "111" =>
                Y <= "01111111";
        END CASE;
    ELSE
        Y <= (OTHERS=>'1');
    END IF;
END
```

END PROCESS;

```
y0 <= y(0);  
y1 <= y(1);  
y2 <= y(2);  
y3 <= y(3);  
y4 <= y(4);  
y5 <= y(5);  
y6 <= y(6);  
y7 <= y(7);
```

END behave;

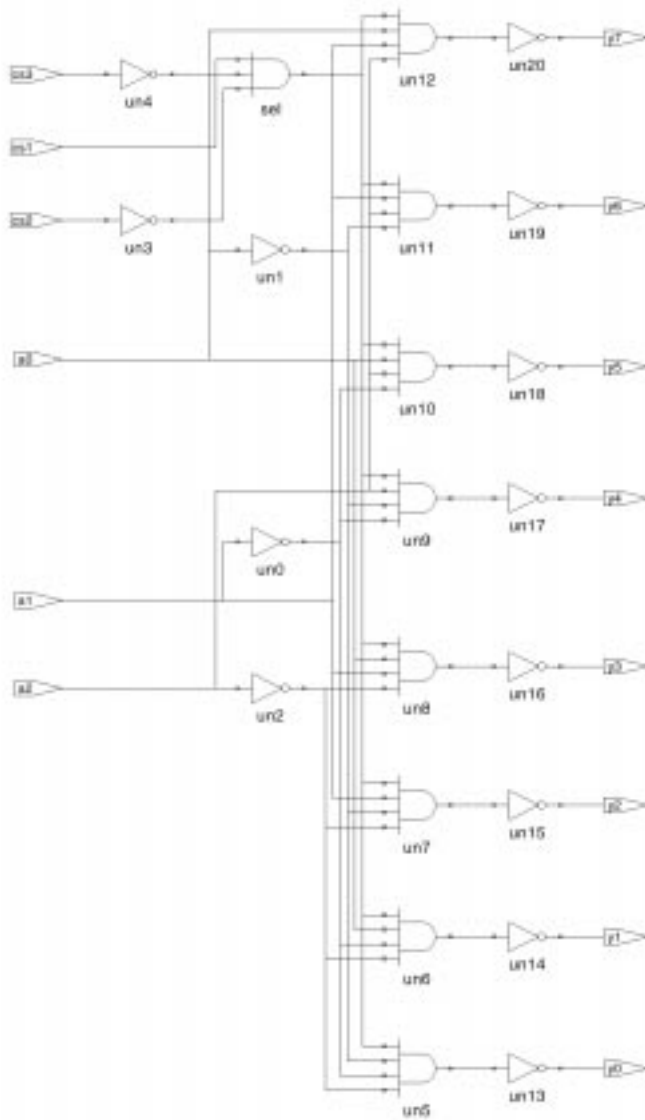


그림 1. TTL 138 (3-of-8 Decoder)의 합성결과

병렬구문으로 기술한 Encoder 의 예제로서 TTL 147 (Decimal-to-BCD Encoder) 은 [예제 4]와 같다. TTL Data Book 의 기능표와 비교해 보기 바란다. 그림 2 는 VHDL 로 기술한 TTL 147 의 합성결과 이다.

[예제 4] 병렬구문으로 기술한 TTL 147 (Decimal-to-BCD Encoder)

```
ENTITY ttl147 IS
PORT (
    d1, d2, d3, d4, d5, d6, d7, d8, d9 : IN std_logic;
    a0, a1, a2, a3 : OUT std_logic );
END ttl147;
```

```
ARCHITECTURE behave_147 OF ttl147 IS
SIGNAL d : std_logic_vector(8 DOWNTO 0);
SIGNAL a : std_logic_vector(3 DOWNTO 0);
BEGIN

    d <= d9 & d8 & d7 & d6 & d5 & d4 & d3 & d2 & d1;

    a0 <= a(0);
    a1 <= a(1);
    a2 <= a(2);
    a3 <= a(3);

    WITH d SELECT
        a <=
            "1111" WHEN "11111111",
            "1110" WHEN "11111110",
            "1101" WHEN "11111110-",
            "1100" WHEN "11111110--",
            "1011" WHEN "11111110---",
            "1010" WHEN "11111110----",
            "1001" WHEN "11111110-----",
            "1000" WHEN "11111110-----",
            "0111" WHEN "11111110-----",
            "0110" WHEN "11111110-----",
            "0101" WHEN "11111110-----",
            "0100" WHEN "11111110-----",
            "0011" WHEN "11111110-----",
            "0010" WHEN "11111110-----",
            "0001" WHEN "11111110-----",
            "0000" WHEN "11111110-----",
            "-----" WHEN OTHERS;

END behave_147;
```

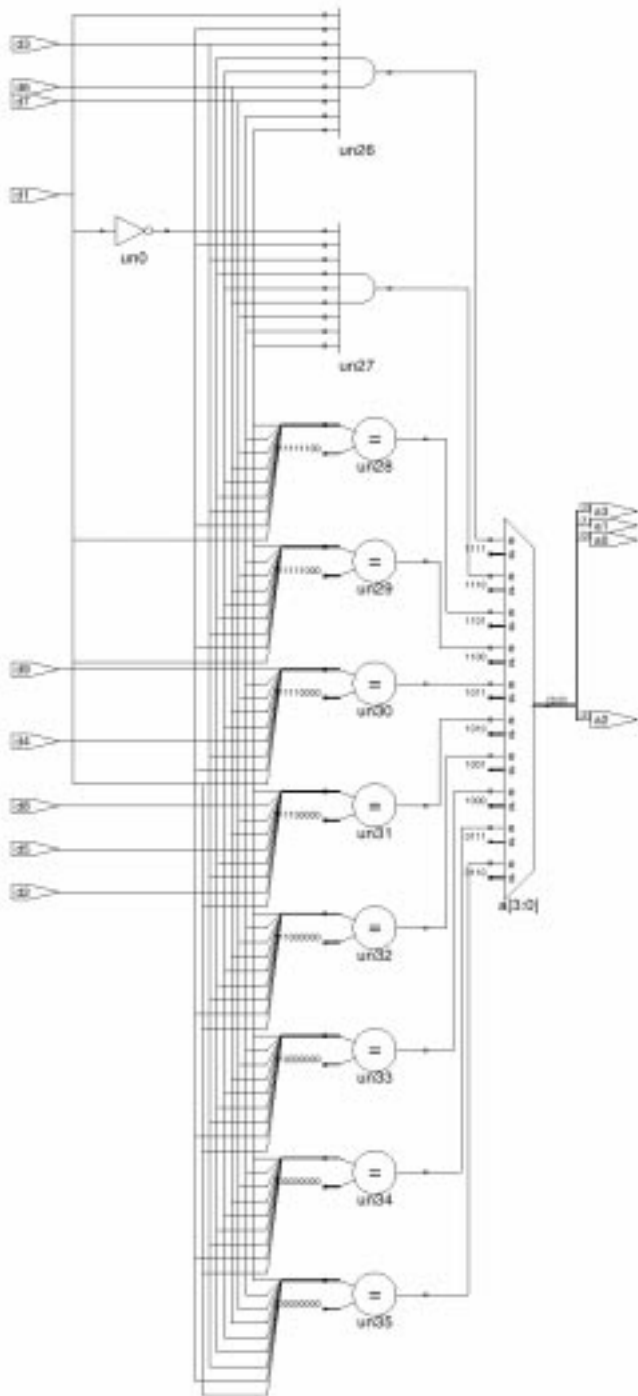


그림 2. TTL 147(decimal-to-BCD Encoder)의 합성 결과

### 3. Priority Encoder

Priority Encoder 는 MUX 와 같이  $n$ -개의 입력에서 1 개를 선택하는 Data Selector 를 설계할 때 발생한다.  $n$ -개 입력 MUX 를 IF~ELSE IF~ELSE~END IF 구문을 사용할 경우 Priority Encoder 의 경우가 발생한다. 기본적으로 IF 구문은 조건식의 1 비트의 조건, 즉 "TRUE" 혹은 "FALSE"에 의하여 두 입력 중

하나를 선택한다. [예제 5]는 MUX 설계에서 2 비트 이상의 조건신호에 IF 구문을 사용 함으로써 Priority Encoder 가 구성된 경우이다.

[예제 5] Priority Encoder

```
ENTITY pri_encoder IS
PORT (
    a, b, c : IN std_logic_vector(3 DOWNTO 0);
    sel : IN std_logic_vector(1 DOWNTO 0);
    o : OUT std_logic_vector(3 DOWNTO 0) );
END pri_encoder;
```

```
ARCHITECTURE behave OF pri_encoder IS
BEGIN
    PROCESS(sel)
    BEGIN
        IF sel="00" THEN
            o <= a;
        ELSIF sel="01" THEN
            o <= b;
        ELSE
            o <= c;
        END IF;
    END PROCESS;
END behave;
```

[예제 5]의 내용을 살펴보면 첫번째 IF 문은 sel(0)에 의하여 결정되며 두 번째의 IF 문은 sel(1)에 의하여 조건 판단이 이루어진다. 이를 다시 표현하면 다음과 같다.

```
PROCESS(sel)
BEGIN
    IF sel(0)="0" THEN
        o <= a;
    ELSE
        IF sel(1)="0" THEN
            o <= b;
        ELSE
            o <= c;
        END IF;
    END IF;
END PROCESS;
```

2 비트 선택신호 sel 이 병렬로 입력 되더라도 마치 sel(1)에 비하여 sel(0)의 선택이 우선순위가 높은 듯이 보인다는 점이다. 만일 Sel="10"의 경우  $o \leq c$  와  $o \leq a$  의 선택에 있어서 우선순위가 높은 sel(0)='0'의 조건에 따라 최종적으로  $o \leq a$  의 할당이 이루어진다.

#### 4. 메모리의 VHDL 기술

메모리 반도체는 2 개의 NOT 게이트 피드백에 의한 간단한 1 비트 기억소자의 규칙적인 배열과 어드레스 디코더로 구성되어 고집적화가 가능하다. 그러나 VHDL 로 메모리를 기술하는 방법은 특별히 존재하지 않으며 배열의 선언 및 플립-플롭 혹은 래치의 기술과 동일하게 표현 된다. [예제 6]은 VHDL 에 의한 Synchronous RAM 을 기술하는 방법이다. 데이터 입출력 포트가 분리 되어 있으며 읽기 및 쓰기 동작에 동기 클럭이 사용 되는 Synchronous RAM 이며 어드레스의 래치가 있다. Altera 의 Flex 10K 시리즈 디바이스에 내장된 EAB (Embedded Array Block)의 메모리를 기능적으로 모델링한 것이다. Altera 의 Flex 10K 의 EAB 구조는 그림 3과 같다.

[예제 6]의 메모리 모델 VHDL 은 합성 가능하지만 그 결과는 매우 비효율적이다. 모든 메모리 셀(Memory Cell)들이 플립 플롭으로 합성되어 실제로 FPGA 에 P&R 할 경우 2Kbits 메모리를 구현하는데 FLEX10K100 (10 만 게이트급)의 90%의 resource 를 사용하는 결과를 보여준다. 따라서 메모리 혹은 n-비트 연산기와 같은 Data Path 종류의 경우 비효율적인 게이트 소자들과 플립 플롭으로 합성하는 대신 합성기에서 제공하는 모델 제네레이터(Model Generator)를 이용하는 것이 적은 면적을 차지할 뿐만 아니라 고속의 동작을 보장 받을 수 있다. 이러한 방식으로 제공되는 라이브러리들을 LPM(Library of Parameterized Modules )이라 한다.

[예제 6] 읽기/쓰기 동기 램(Synchronous RAM)의 VHDL 기술에

```
entity RAM256x8 is
port (
    data : IN STD_LOGIC_VECTOR (7 downto 0);
    addr : IN STD_LOGIC_VECTOR (7 downto 0);
    we : IN STD_LOGIC;
    inclock : IN STD_LOGIC;
    outclock : IN STD_LOGIC;
    q : OUT STD_LOGIC_VECTOR (7 downto 0) );
end RAM256x8;
```

```
architecture sram_behaviour of RAM256x8 is
    constant low_address: natural := 0;
    constant high_address: natural := 255;
```



```

subtype byte is std_logic_vector( 7 downto 0 );

type memory_array is
    array (natural range low_address to high_address) of
byte;

signal mem : memory_array;
signal address : natural range low_address to high_address;
begin

address_latch : process ( addr, inclock)
begin
    if (inclock'event and inclock='1') then
        address <= to_integer( unsigned(addr) );
    end if;
end process;

mem_write : process ( address, we)
begin
    if (we = '1') then
        mem( address ) <= data(7 downto 0);
    end if;
end process;

mem_read : process ( address, outclock)
begin
    if (outclock'event and outclock='1') then
        q <= mem(address);
    end if;
end process;

end sram_behaviour ;

```

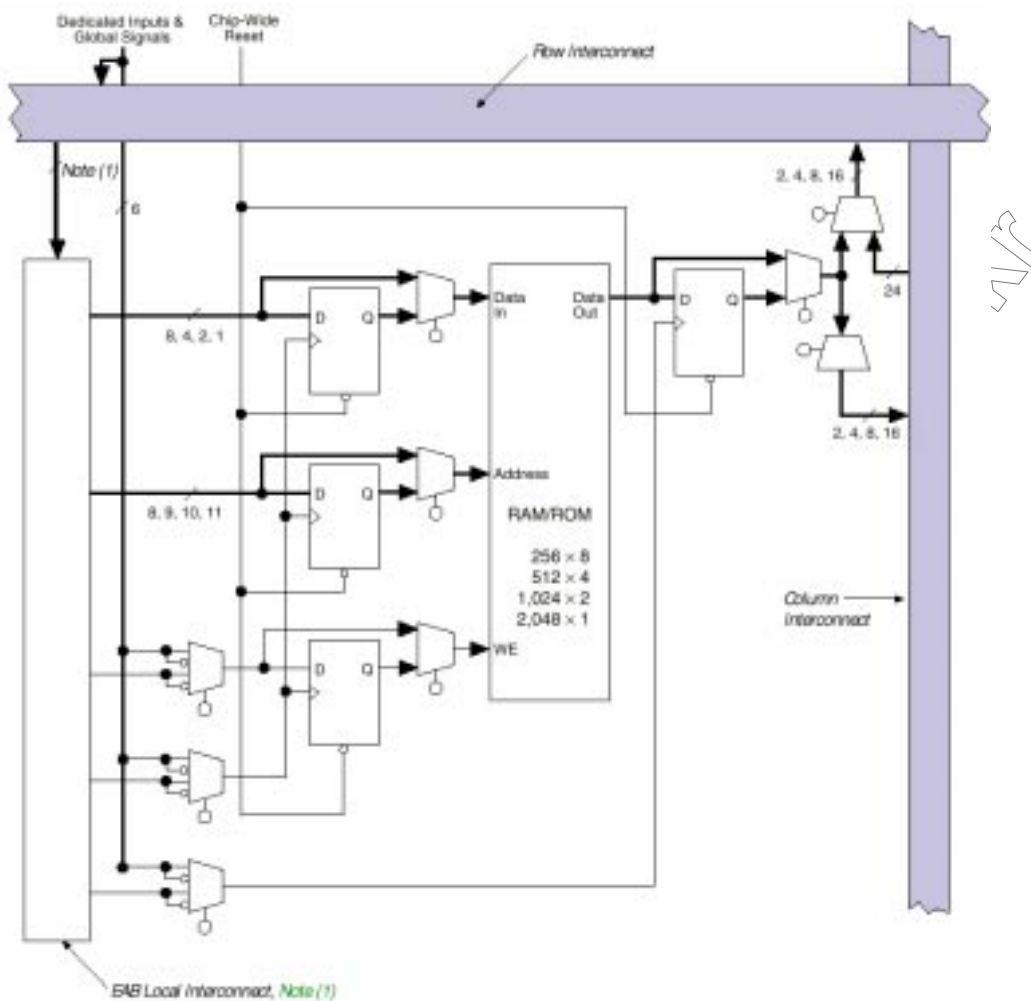


그림 3. Altera Flex 10K의 EAB 구조 (Altera DataBook)

그림 3의 EAB 구조에서 알수 있듯이 주소와 입출력 데이터 버스에 연결된 래치를 사용하지 않을 경우 비동기 램의 구성도 가능하다. [예제 6]은 비동기 램의 기술한 것이다.

[예제 7] 읽기/쓰기 비동기 램(Asynchronous RAM)의 VHDL 기술에

```

architecture sram_behaviour of RAM256x8 is
    constant low_address: natural := 0;
    constant high_address: natural := 255;

    subtype byte is std_logic_vector( 7 downto 0 );

    type memory_array is
        array (natural range low_address to high_address) of
            byte;
end architecture;

```

```

    signal mem : memory_array;
    signal address : natural range low_address to high_address;

begin

address <= to_integer( unsigned(addr) );

    mem_write : process ( address, we, data)
    begin
        if (we = '1') then
            mem( address ) <= data(7 downto 0);
        end if;
    end process;

end sram_behaviour;

```

메모리 합성을 위해서 디바이스 벤더에서 제공하는 LPM 을 이용하는 방법은 매우 간단하다. 다만 합성을 위해서 LPM 을 사용하므로 이와 동일하게 작동하는 [예제 6]과 같은 기능 모델을 따로 준비하여야 한다. [예제 8]은 Altera Flex10K 의 메모리 LPM 을 이용한 예제이다. entity 는 [예제 6]과 동일하도록 작성되어 있으므로 기능 시뮬레이션에서는 [예제 6]의 sram\_behaviour 을, 합성할 때는 [예제 7]의 synthesis 야키택쳐를 적용한다. 동일한 entity 에 서로 다른 architecture 를 연결하는 방법은 VHDL 의 configuration 구문을 이용할 수 있다.

LPM 의 component HDL 들은 디바이스 벤더의 P&R 툴과 함께 제공되므로 이를 참고 한다. [예제 8]의 LPM\_RAM\_DQ 는 Altera 의 메모리 LPM 중의 하나이며 이외에도 LPM\_RAM\_IO 가 있다.

[예제 8] LPM 을 이용한 메모리 기술

```

PACKAGE ram_constants IS
    constant ADDR_WIDTH: INTEGER := 8;
    constant DATA_WIDTH: INTEGER := 8;
END ram_constants;

ENTITY RAM256x8 IS
PORT(
    data : IN STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
    addr : IN STD_LOGIC_VECTOR (ADDR_WIDTH - 1 downto 0);
    we : IN STD_LOGIC;
    inclock : IN STD_LOGIC;
    outclock : IN STD_LOGIC;

```

```
    q : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0) );  
END RAM256x8;
```

```
ARCHITECTURE synthesis OF RAM256x8 IS
```

```
BEGIN
```

```
inst_1 : LPM_RAM_DQ  
  GENERIC MAP (  
    lpm_widthad => ADDR_WIDTH,  
    lpm_width => DATA_WIDTH)  
  PORT MAP (  
    data => data,  
    address => addr,  
    we => we,  
    inclock => inclock,  
    outclock => outclock,  
    q => q);
```

```
END synthesis;
```

메모리 모듈을 사용한 VHDL의 예는 [예제 8]과 같다. 앞의 예제에서 기술한 메모리 RAM VHDL을 컴퍼넌트로 이용한다.

[예제 9] RAM 모듈 인터페이스

```
ENTITY TOP_ram256x8 IS  
  port (  
    data: in std_logic_vector(7 downto 0);  
    address: in std_logic_vector(7 downto 0);  
    we: in std_logic;  
    inclock: in std_logic;  
    outclock: in std_logic;  
    q: out std_logic_vector(7 downto 0) );  
END TOP_ram256x8;
```

```
ARCHITECTURE TOP_a_ram256x8 OF TOP_ram256x8 IS
```

```
  component ram256x8  
  port (  
    data : IN STD_LOGIC_VECTOR (7 downto 0);  
    addr : IN STD_LOGIC_VECTOR (7 downto 0);
```

```

we : IN STD_LOGIC;
inclock : IN STD_LOGIC;
outclock : IN STD_LOGIC;
q : OUT STD_LOGIC_VECTOR (7 downto 0) );
end component;

```

BEGIN

```

umem : ram256x8
port map (
  data => data,
  addr => address,
  we => we,
  inclock => inclock,
  outclock => outclock,
  q => q );

```

END TOP\_a\_ram256x8;

고급의 합성기의 경우 같은 벤더라 하더라도 디바이스의 구조에 따라 적절하게 합성을 해내는 경우가 있는데 예를 들면 [예제 6]의 램 VHDL 을 메모리 블록을 따로 가지고 있지 않은 CPLD 를 타겟으로 합성하는 경우와 메모리 EAB 를 내장한 FPGA 로 합성하는 경우 합성결과가 서로 다르다. 이러한 Device Architecture Specific Synthesizer 들은 VHDL 소스로부터 자동으로 LPM 라이브러리를 추출해내는 기능을 가지고 있기 때문이다. 만일 LPM 을 자동으로 추출해내지 못하는 합성기를 이용하는 경우 상위 디자인만 합성하고 벤더의 P&R 틀에서 LPM 블록을 지정해 주면 된다. Altera 의 P&R 틀인 MaxPlus II 와 LPM 추출기능이 없는 합성기인 메타모어(Metamor)사의 합성기를 이용할 경우 메모리 LPM 을 합성해 내려면 [예제 9]의 메모리 인터페이스 VHDL 만을 합성하도록 한다. 이때 합성기는 RAM256x8 에 해당하는 Sub-Module 이 없다는 메시지("component : umem : No Entity bound to this instance.")를 출력하고 상위 모듈만을 합성하여 EDIF 파일을 출력한다.

이때 생성된 EDIF 파일을 이용해서 P&R 하려면 [예제 8]과 같이 LPM 을 기술한 VHDL 소스가 상위 모듈의 합성 결과인 EDIF 와 함께 존재하면 된다. 이 경우 반드시 메모리 Sub-Module 의 entity 명과 VHDL 파일명이 일치하여야 한다. Exemplar 사의 합성기들은 적절한 타겟 디바이스(target device)를 지정해 주면 RTL 수준의 VHDL 로부터 LPM 모듈을 추출해낼 수 있다. RTL 수준의 VHDL 을 합성하여 얻어진 EDIF 의 내용을 살펴보면 LPM 의 내용이 들어 있음을 알 수 있다. 다음은 LPM 을 RTL 수준의 VHDL 을 합성한 결과의 일부분이다.

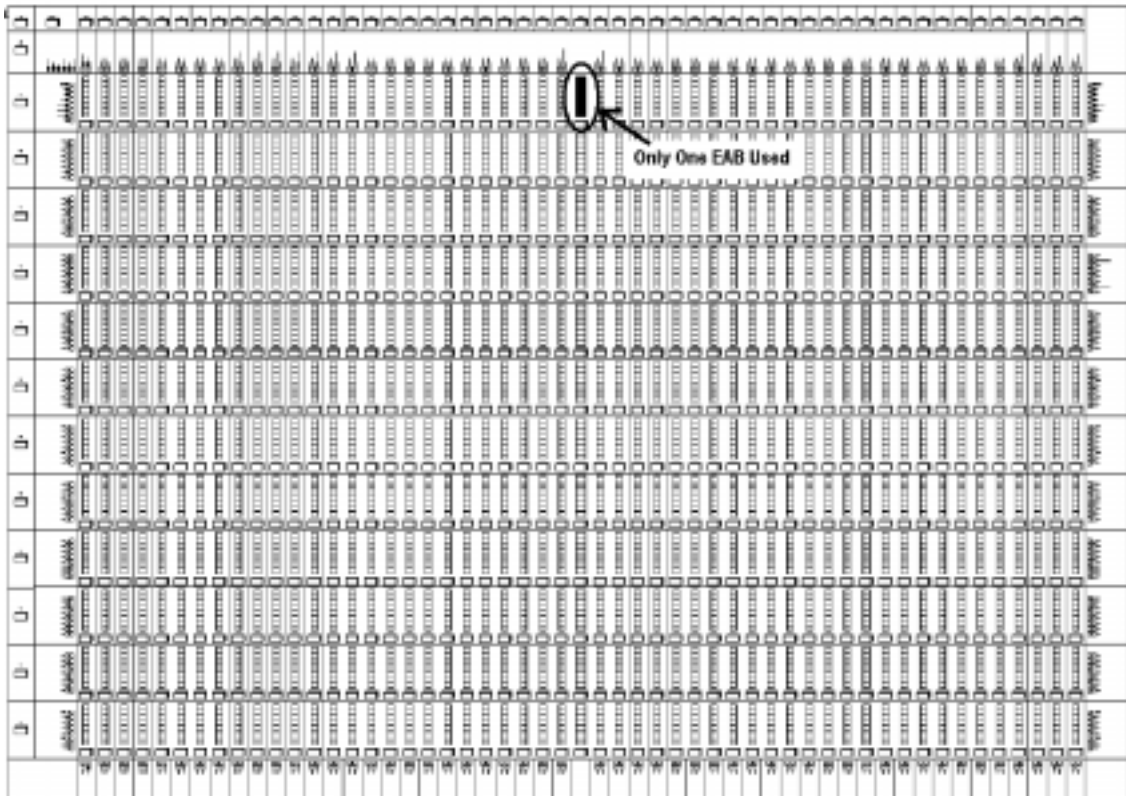
(cell lpm\_ram\_dq\_4\_8\_8 (cellType GENERIC)

```

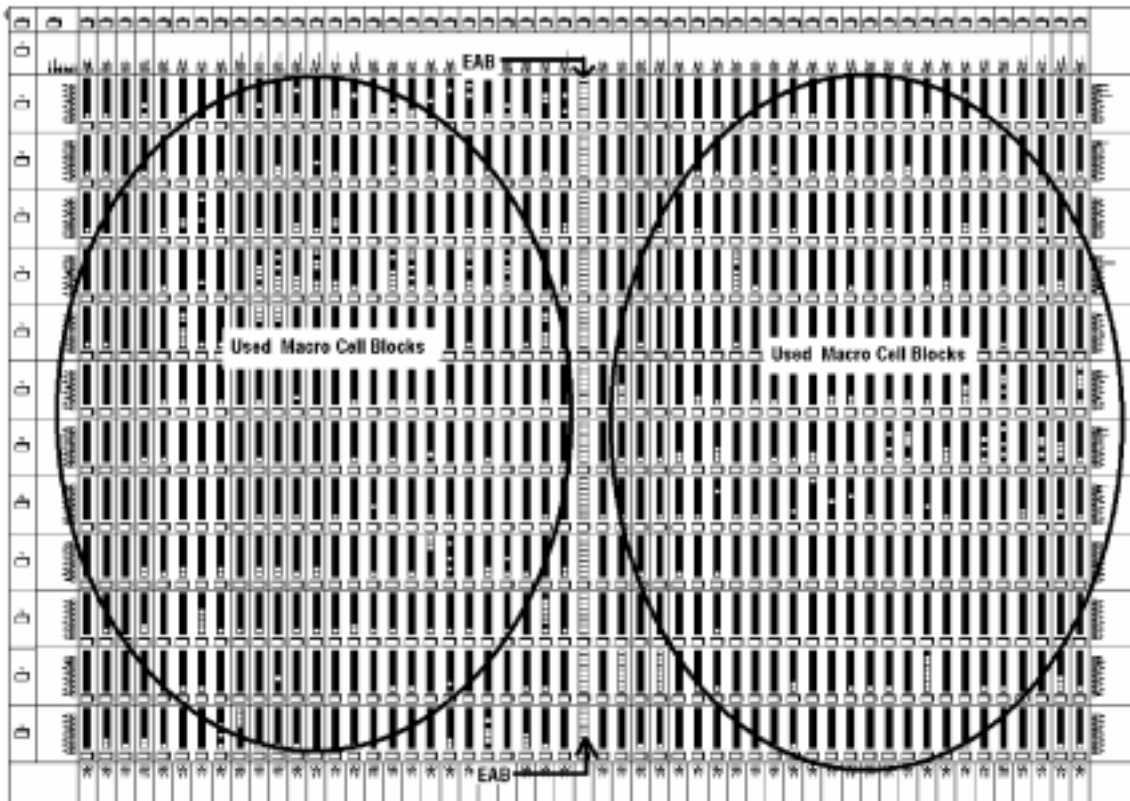
(view INTERFACE (viewType NETLIST)
(interface
(port (array (rename data "data(7:0)") 8 )(direction INPUT))
(port (array (rename address "address(7:0)") 8 )(direction INPUT))
(port we (direction INPUT))
(port (array (rename q "q(7:0)") 8 )(direction OUTPUT)))
(property lpm_width (string "8"))
(property lpm_widthhad (string "8"))
(property lpm_numwords (string "256"))
(property lpm_type (string "LPM_RAM_DQ"))
(property lpm_indata (string "UNREGISTERED"))
(property lpm_address_control (string "UNREGISTERED"))
(property lpm_outdata (string "UNREGISTERED"))))
(external PRIMITIVES
(edifLevel 0)

```

그림 4는 2Kbit 메모리 모듈을 합성한 후 10만 게이트급의 Flex10K100 디바이스에 P&R 한 플로어 플랜(Floor Plan)이다. LPM으로 합성한 결과와 플립-플롭으로 합성한 결과 커다란 차이가 있음을 알수 있다. 타겟 디바이스의 구조에 적합한 LPM으로 합성한 경우 그림 4의 (a)와 같이 단 1개의 EAB만을 차지하는 반면 플립 플롭으로 합성한 경우 (b) 거의 모든 디바이스 리소스를 사용하고 있다.



(a) LPM을 사용한 합성 결과

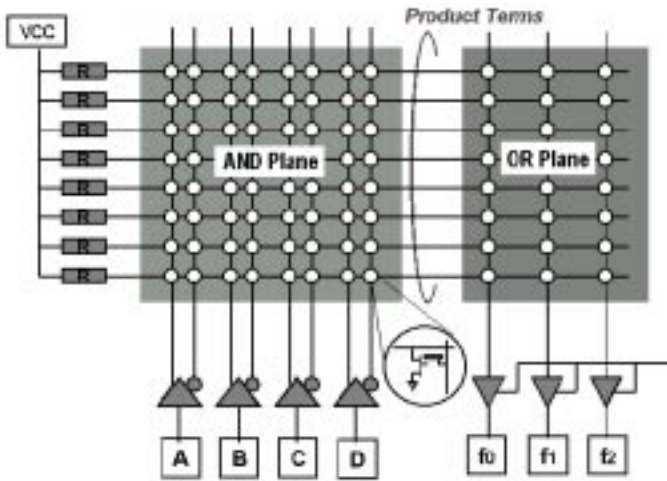


(b) 플립 플롭을 이용한 합성

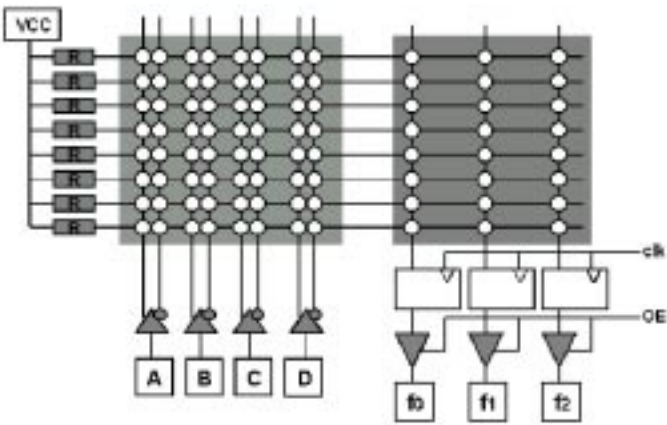
그림 4. 메모리 모듈의 합성 및 P&R 결과 Floor Plan

## 5. PLA

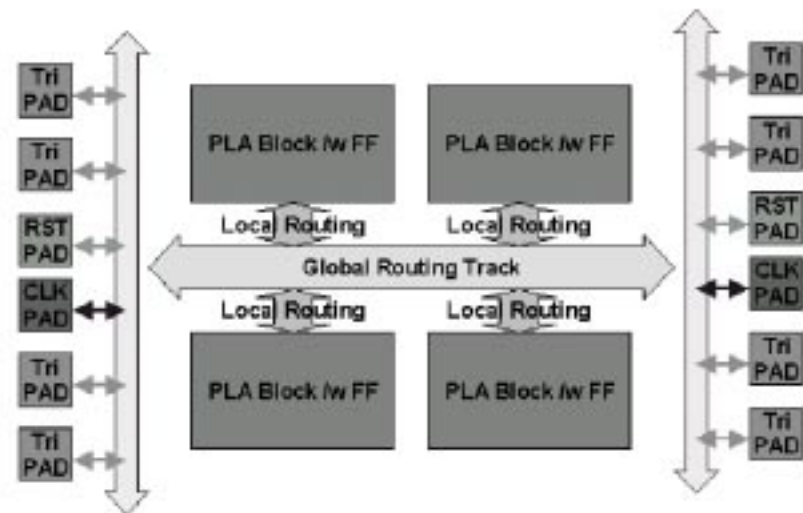
논리회로의 기능을 논리식으로 표현하면 논리곱과 합 (Sum of Product term)으로 나타낼 수 있다. PLA(Programmable Logic Array)는 프로그래머블 논리 회로 소자의 가장 기본적인 형태로서 논리식의 곱부분 (AND plane)과 합의 부분 (OR plane)을 각각 AND 스위치 와 OR 스위치의 배열로 구성한 것이다. AND 와 OR 의 스위치 열(Array)들은 게이트 부분에 Charge 와 Dis-charge 가 가능한 트랜지스터들로 만들어져 있다. ROM 은 AND Plane 부분이 조합 가능한 모든 논리곱을 표현하는 어드레스 디코더로 구성하고 OR 부분을 프로그램이 가능하도록 만들어진 PLD(Programmable Logic Device) 라 할 수 있다. 그림 5 는 PLA 의 구조를 간단하게 나타내었다. 수직-수평의 와이어가 교차하는 부분에 트랜지스터 스위치가 존재하며 이들 스위치의 작동가능 상태를 설정하는 것을 PLA 프로그래밍 이라 한다.



(a) 기본적인 PLA 구조



(b) 출력에 F/F 가 있는 PLA



(c) CPLD 의 기본구조

그림 5. PLA/CPLD 의 내부구조



[예제 10]은 PLA 를 VHDL 로서 표현한 것이다. PLA 의 구조는 ROM 메모리와 같이 트랜지스터의 배열로 구성하며 고 집적화가 가능하긴 하지만 VHDL 로 표현하여 합성하면 결국 논리 소자들의 등가회로가 된다. 다만 여러 개의 PLA 구조를 매크로 블록으로 집적화한 CPLD 의 경우 효과적인 표현일수 있다. CPLD 혹은 FPGA 들은 제조사 마다 내부구조가 서로 다르며 이에 따라 효과적인 설계의 지침이 제시되어 있는 것이 보통이므로 이를 잘 활용하면 향상된 설계결과를 얻을 수 있을 것이다. VHDL 로 설계한 후 합성을 거치면 결국 타겟 디바이스에 적절한 형태의 등가 논리 회로를 얻게 되는 것이다. [예제 10]은 PLA 를 VHDL 로 기술한 예제로서 AND plane 에서 입력 벡터에서 '1'의 위치를 검출한 후 위치를 OR plane 을 통하여 출력한다. PLA 의 Junction Map 을 VHDL 의 배열로 표현하였으며 AND 와 OR 의 기능은 Procedure 에 기술되었다. 그림 6 은 예제의 PLA map 을 나타내었다. [예제 10]의 VHDL 로 표현한 PLA VHDL 은 일부 합성기에서는 정확한 합성결과 보여 주지 못하므로 주의 하도록 한다. 실제로 [예제 10]을 메타모어 합성기 3.0.x 와 밴티스(Vantis) Mach5 디바이스를 타겟으로 합성한 결과 합성가능 하였으나 올바른 동작 결과를 얻지 못한다. Exemplar 합성기는 정확한 합성결과를 얻을 수 있었다. 그림 7 은 [예제 10]의 합성 결과이다.

[예제 10] PLA 의 VHDL 기술

```
entity pla is
    port ( test_vector    : in  std_logic_vector (3 downto 0);
          result_vector  : out std_logic_vector (2 downto 0) );
end pla;

architecture behave of pla is

    type std_logic_pla is array (natural range <>, natural range <>) of
        std_logic;

    procedure pla_table ( constant invec  : in  std_logic_vector;
                        signal  outvec   : out std_logic_vector;
                        constant table   : in  std_logic_pla      ) is
        variable x : std_logic_vector (table'range(1)) ; -- product lines
        variable y : std_logic_vector (outvec'range) ;  -- outputs
        variable b : std_logic ;
    begin
        assert (invec'length + outvec'length = table'length(2))
            report "Size of Inputs and Outputs do not match table size"
            severity ERROR ;

        -- Calculate the AND plane
```

```

x := (others=>'1');
for i in table'range(1) loop -- PLA Table ROWs
  for j in invec'range loop
    b := table (i,table'left(2)-invec'left+j) ;
    if (b='1') then
      x(i) := x(i) AND invec (j) ;
    elsif (b='0') then
      x(i) := x(i) AND NOT invec(j) ;
    end if ;
    -- If b is not '0' or '1' (e.g. '-') product line is
insensitive to invec(j)
  end loop ;
end loop ;

-- Calculate the OR plane
y := (others=>'0') ;
for i in table'range(1) loop
  for j in outvec'range loop
    b := table(i,table'right(2) - outvec'right+j) ;
    if (b='1') then
      y(j) := y(j) OR x(i);
    end if ;
  end loop ;
end loop ;
outvec <= y ;

end pla_table ;

constant pos_of_fist_one : std_logic_pla (4 downto 0, 6 downto 0) :=
( "1---000", -- first '1' is at position 0
  "01--001", -- first '1' is at position 1
  "001-010", -- first '1' is at position 2
  "0001011", -- first '1' is at position 3
  "0000111" ) ;-- There is no ' 1' in the input

begin

  pla_table ( test_vector, result_vector, pos_of_fist_one);

end behave;

```

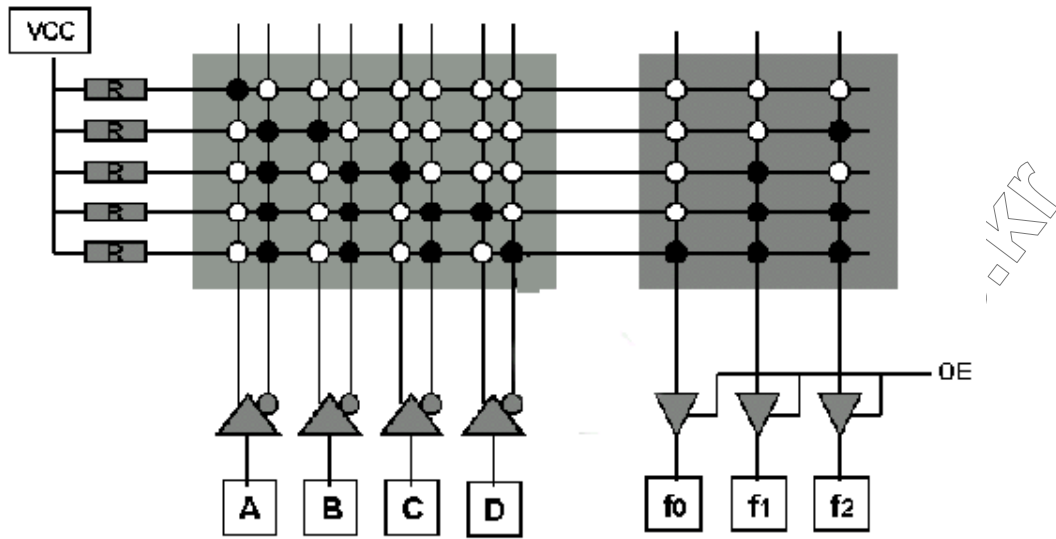


그림 6. [예제 10] PLA 예제의 junction map

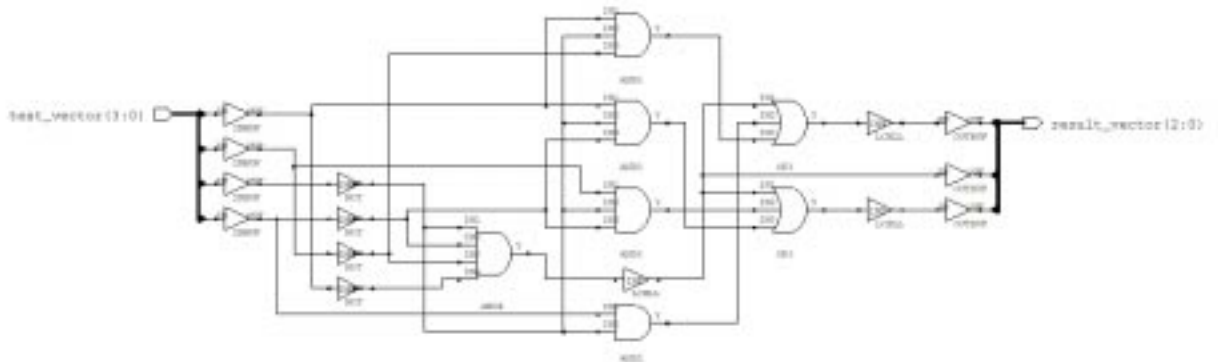


그림 7. PLA 합성 결과

<http://www.kkri.ac.kr>