**Lecture 02**

# FPGA 장치와 EDA 툴 소개

# FPGA 장치

- Configurable logic blocks + Programmable switch



switch

configurable logic block

I/O

I/O

I/O

I/O

I/O

in /6 → LUT → out

clk → DFF → out$_d$
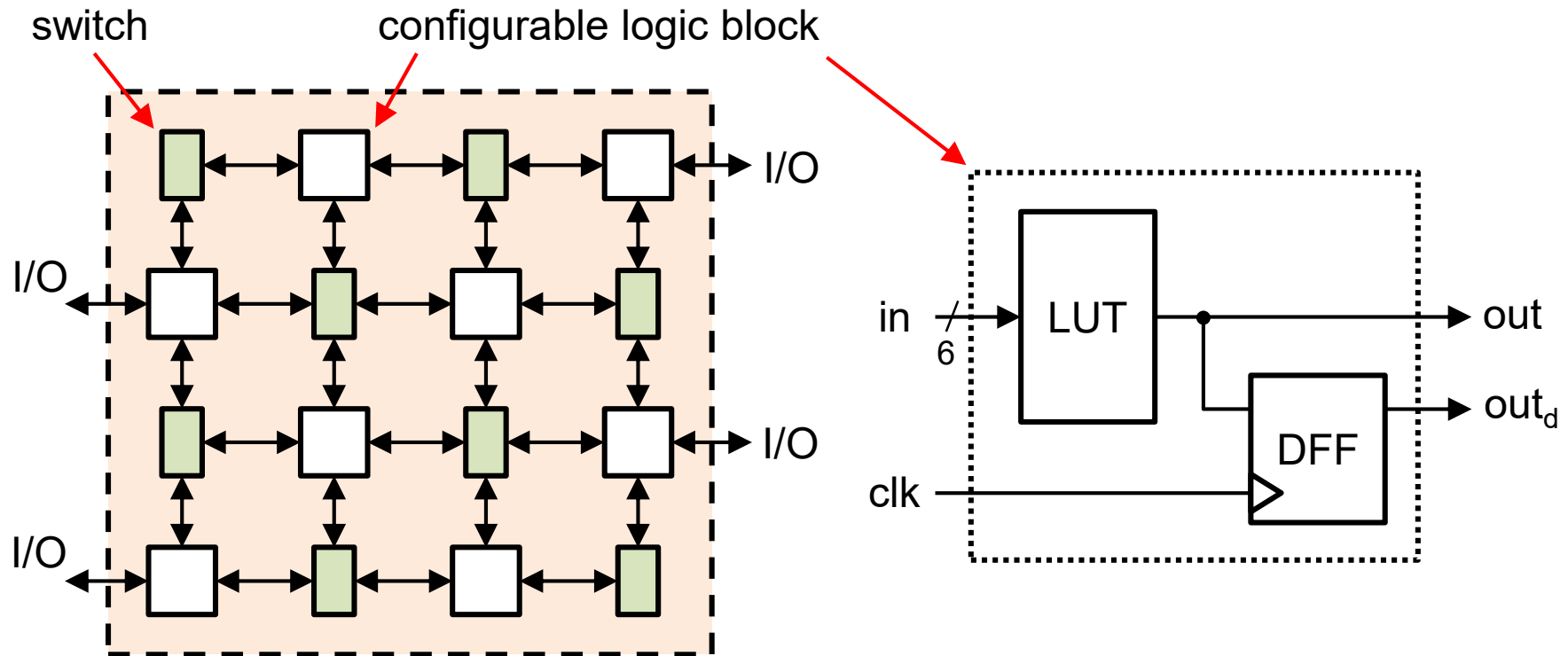
# FPGA 장치

- ## AMD Xilinx의 7 Series FPGAs

  https://docs.amd.com/v/u/en-US/ds180_7Series_Overview



**XILINX**

**7 Series FPGAs Data Sheet: Overview**

DS180 (v2.6.1) September 8, 2020      **Product Specification**

### General Description

Xilinx® 7 series FPGAs comprise four FPGA families that address the complete range of system requirements, ranging from low cost, small form factor, cost-sensitive, high-volume applications to ultra high-end connectivity bandwidth, logic capacity, and signal processing capability for the most demanding high-performance applications. The 7 series FPGAs include:

- Spartan®-7 Family: Optimized for low cost, lowest power, and high I/O performance. Available in low-cost, very small form-factor packaging for smallest PCB footprint.

- Artix®-7 Family: Optimized for low power applications requiring serial transceivers and high DSP and logic throughput. Provides the lowest total bill of materials cost for high-throughput, cost-sensitive applications.

- Kintex-7 Family: Optimized for best price-performance with a 2X improvement compared to previous generation, enabling a new class of FPGAs.

- Virtex®-7 Family: Optimized for highest system performance and capacity with a 2X improvement in system performance. Highest capability devices enabled by stacked silicon interconnect (SSI) technology.

Built on a state-of-the-art, high-performance, low-power (HPL), 28 nm, high-k metal gate (HKMG) process technology, 7 series FPGAs enable an unparalleled increase in system performance with 2.9 Tb/s of I/O bandwidth, 2 million logic cell capacity, and 5.3 TMAC/s DSP, while consuming 50% less power than previous generation devices to offer a fully programmable alternative to ASSPs and ASICs.

### Summary of 7 Series FPGA Features

- Advanced high-performance FPGA logic based on real 6-input look-up table (LUT) technology configurable as distributed memory.

- 36 Kb dual-port block RAM with built-in FIFO logic for on-chip data buffering.

- High-performance SelectIO™ technology with support for DDR3 interfaces up to 1,866 Mb/s.

- High-speed serial connectivity with built-in multi-gigabit transceivers from 600 Mb/s to max. rates of 6.6 Gb/s up to 28.05 Gb/s, offering a special low-power mode, optimized for chip-to-chip interfaces.

- A user configurable analog interface (XADC), incorporating dual 12-bit 1MSPS analog-to-digital converters with on-chip thermal and supply sensors.

- DSP slices with 25 x 18 multiplier, 48-bit accumulator, and pre-adder for high-performance filtering, including optimized symmetric coefficient filtering.

- Powerful clock management tiles (CMT), combining phase-locked loop (PLL) and mixed-mode clock manager (MMCM) blocks for high precision and low jitter.

- Quickly deploy embedded processing with MicroBlaze™ processor.

- Integrated block for PCI Express® (PCIe), for up to x8 Gen3 Endpoint and Root Port designs.

- Wide variety of configuration options, including support for commodity memories, 256-bit AES encryption with HMAC/SHA-256 authentication, and built-in SEU detection and correction.

- Low-cost, wire-bond, bare-die flip-chip, and high signal integrity flip-chip packaging offering easy migration between family members in the same package. All packages available in Pb-free and selected packages in Pb option.

- Designed for high performance and lowest power with 28 nm, HKMG, HPL process, 1.0V core voltage process technology and 0.9V core voltage option for even lower power.

*Table 1:* **7 Series Families Comparison**

| Max. Capability | Spartan-7 | Artix-7 | Kintex-7 | Virtex-7 |
|---|---|---|---|---|
| Logic Cells | 102K | 215K | 478K | 1,955K |
| Block RAM[1] | 4.2 Mb | 13 Mb | 34 Mb | 68 Mb |
| DSP Slices | 160 | 740 | 1,920 | 3,600 |
| DSP Performance[2] | 176 GMAC/s | 929 GMAC/s | 2,845 GMAC/s | 5,335 GMAC/s |
| MicroBlaze CPU[3] | 260 DMIPs | 303 DMIPs | 438 DMIPs | 441 DMIPs |
| Transceivers | – | 16 | 32 | 96 |
| Transceiver Speed | – | 6.6 Gb/s | 12.5 Gb/s | 28.05 Gb/s |
| Serial Bandwidth | – | 211 Gb/s | 800 Gb/s | 2,784 Gb/s |
| PCIe Interface | – | x4 Gen2 | x8 Gen2 | x8 Gen3 |
| Memory Interface | 800 Mb/s | 1,066 Mb/s | 1,866 Mb/s | 1,866 Mb/s |
| I/O Pins | 400 | 500 | 500 | 1,200 |
| I/O Voltage | 1.2V–3.3V | 1.2V–3.3V | 1.2V–3.3V | 1.2V–3.3V |
| Package Options | Low-Cost, Wire-Bond | Low-Cost, Wire-Bond, Bare-Die Flip-Chip | Bare-Die Flip-Chip and High-Performance Flip-Chip | Highest Performance Flip-Chip |

**Notes:**
1. Additional memory available in the form of distributed RAM.
2. Peak DSP performance numbers are based on symmetrical filter implementation.
3. Peak MicroBlaze CPU performance numbers based on microcontroller preset.

# EDA 툴 (EDA: Electronic Design Automation)

- Synthesis(합성)
  ① High-level synthesis
  ② RT-level synthesis
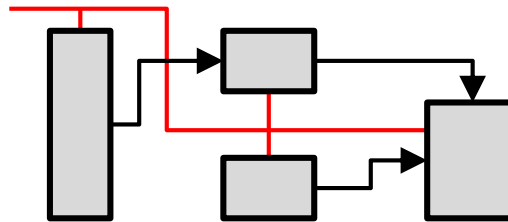  ③ Gate-level synthesis
  ④ Technology mapping

```
always @(a, b, sel) begin
  if (sel == 1) c = a;
  else          c = b;
end
```

Standard cell library
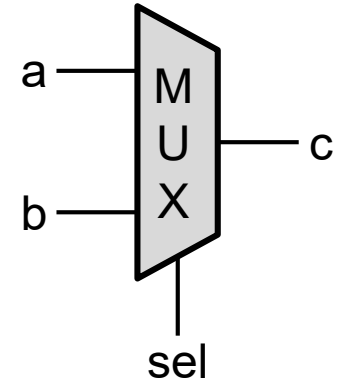
- Placement & Routing
  ① Placement
  ② Clock tree synthesis
  ③ Signal routing
  ④ Timing closure

- Device programming

# EDA 툴

- Synthesis(합성)
  ① High-level synthesis
  ② RT-level synthesis
  ③ Gate-level synthesis
  ④ Technology mapping
- Placement & Routing
  ① Placement
  ② Clock tree synthesis
  ③ Signal routing
  ④ Timing closure
- Device programming

**Vivado**

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html
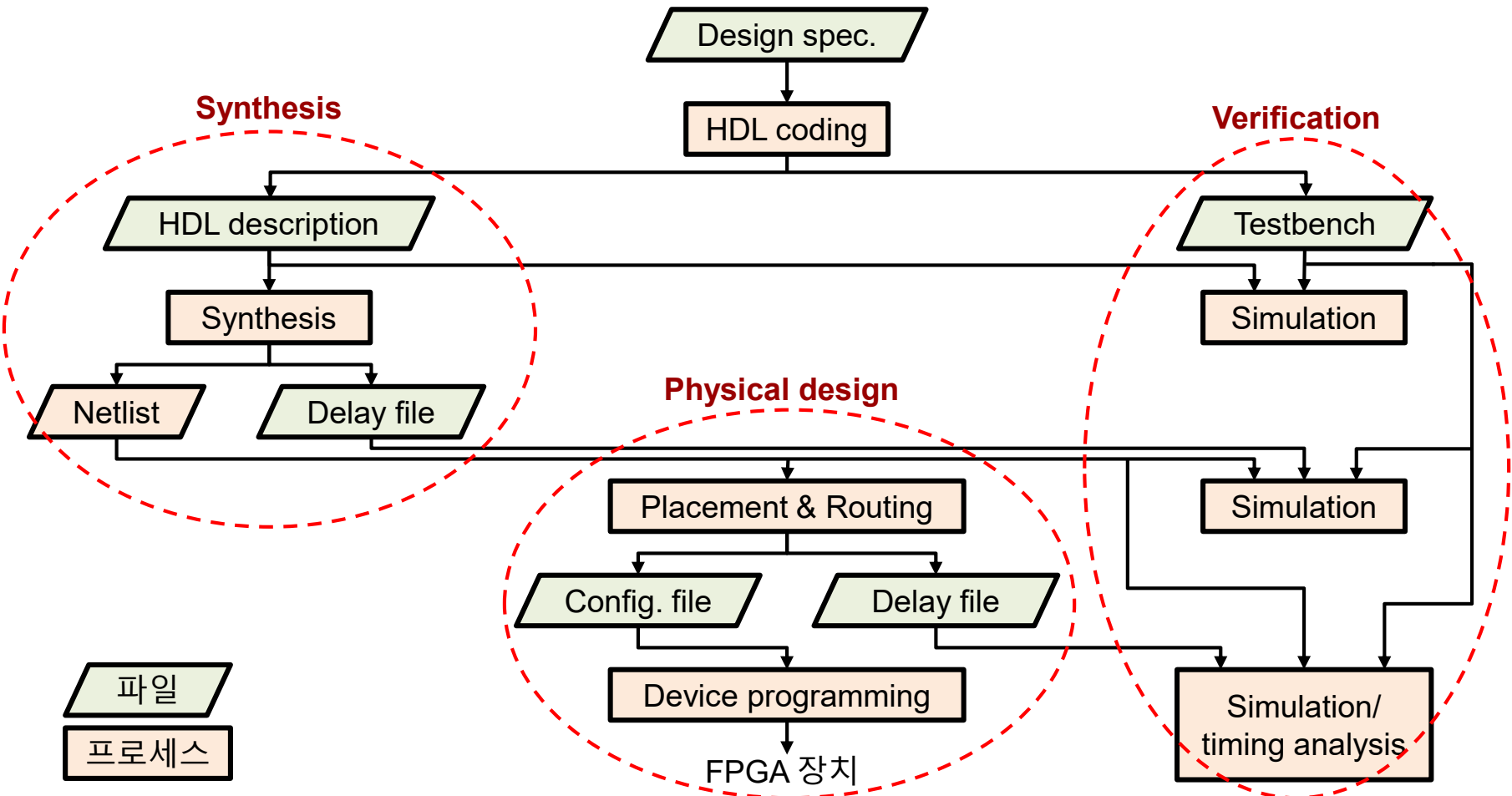
+ bare-metal programming
= **Vitis**

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html
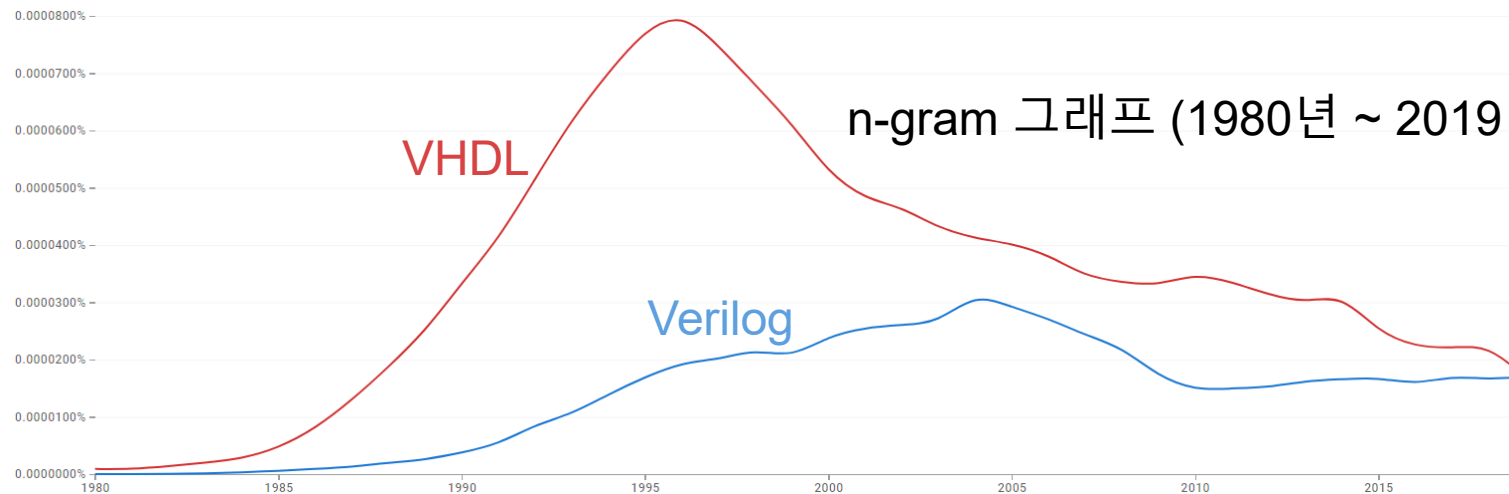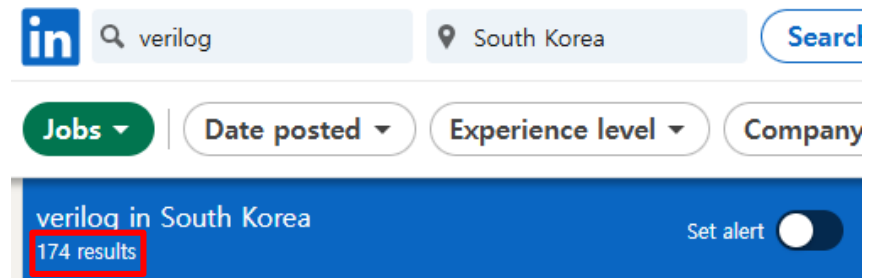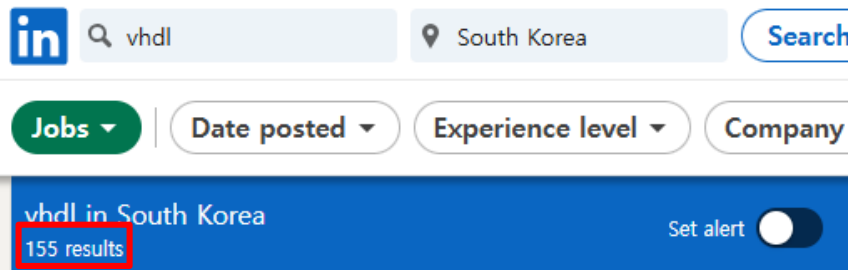
+ Linux embedded programming
= **PetaLinux**

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html
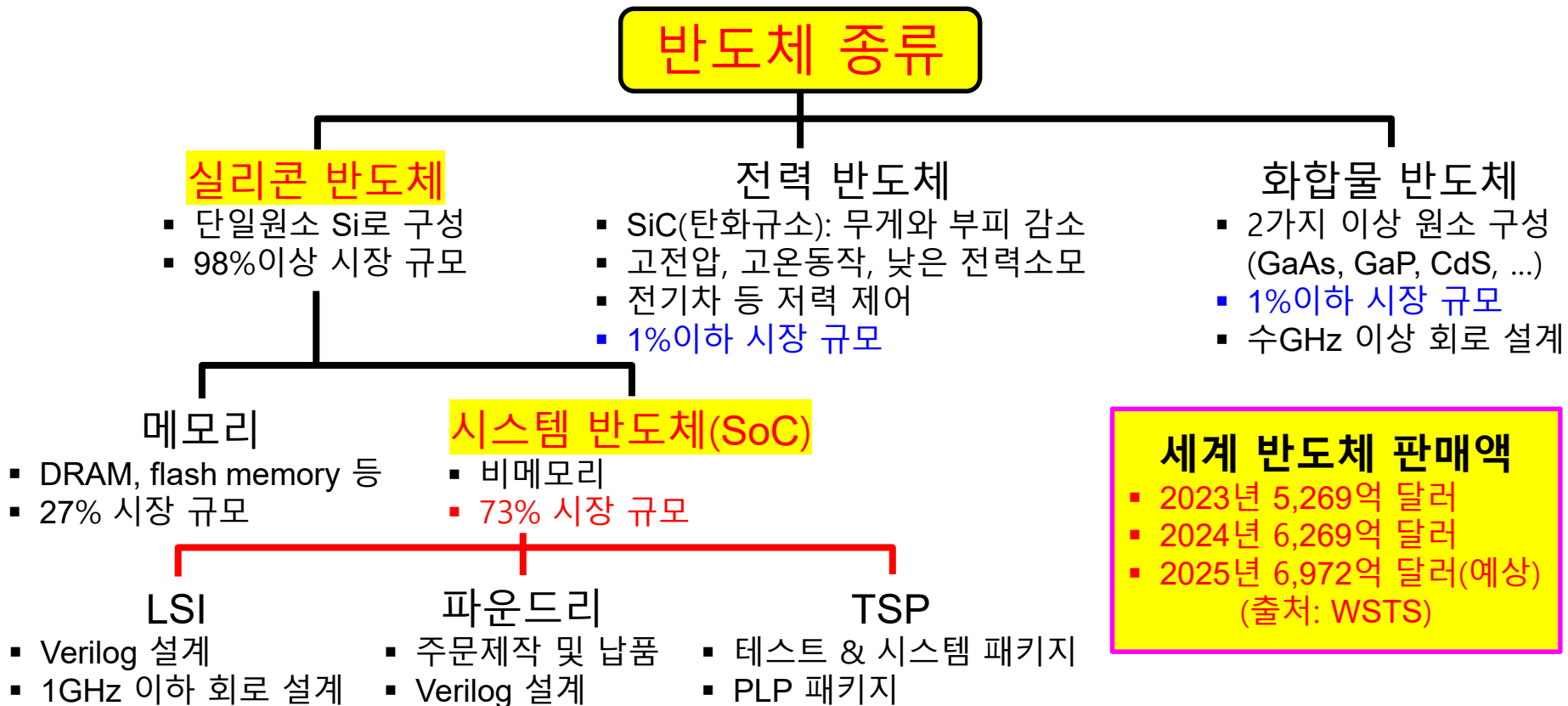
# FPGA 설계 과정

# Verilog HDL

- VHDL 있는데, 왜 Verilog?



n-gram 그래프 (1980년 ~ 2019년)

# Verilog HDL

- ## 반도체 시장 현황

**반도체 종류**

**실리콘 반도체**
- 단일원소 Si로 구성
- 98%이상 시장 규모

전력 반도체
- SiC(탄화규소): 무게와 부피 감소
- 고전압, 고온동작, 낮은 전력소모
- 전기차 등 저력 제어
- 1%이하 시장 규모

화합물 반도체
- 2가지 이상 원소 구성 (GaAs, GaP, CdS, ...)
- 1%이하 시장 규모
- 수GHz 이상 회로 설계

메모리
- DRAM, flash memory 등
- 27% 시장 규모

**시스템 반도체(SoC)**
- 비메모리
- 73% 시장 규모

LSI
- Verilog 설계
- 1GHz 이하 회로 설계

파운드리
- 주문제작 및 납품
- Verilog 설계

TSP
- 테스트 & 시스템 패키지
- PLP 패키지

**세계 반도체 판매액**
- 2023년 5,269억 달러
- 2024년 6,269억 달러
- 2025년 6,972억 달러(예상) (출처: WSTS)

# Verilog HDL

- 예시: even-parity detector

| 입력 | | | 출력 |
|---|---|---|---|
| a[2] | a[1] | a[0] | even |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Verilog HDL

- 예시: even-parity detector

```verilog
module even_parity_detector (
    input  [2:0] a,
    output       even
);

wire odd;

assign odd  = a[2]^a[1]^a[0];
assign even = ~odd;

endmodule
```

- 짧음
- 이해하기 쉬움
- 프로그래밍 언어와 비슷함

```vhdl
library ieee;
use ieee.std_logic_1164.all;

-- entity declaration
entity even_parity_detector is
    port (
        a   : in std_logic_vector(2 downto 0);
        even: out std_logic
    );
end even_parity_detector;

-- architecture body
architecture xor_arch of even_parity_detector is
    signal odd: std_logic;
begin
    even <= not odd;
    odd  <= a(2) xor a(1) xor a(0);
end xor_arch;
```

# Verilog HDL

- 예시: even-parity detector

```verilog
module even_parity_detector
(
    input  [2:0] a,
    output       even
);

wire odd;

assign odd  = a[2]^a[1]^a[0];
assign even = ~odd;

endmodule
```
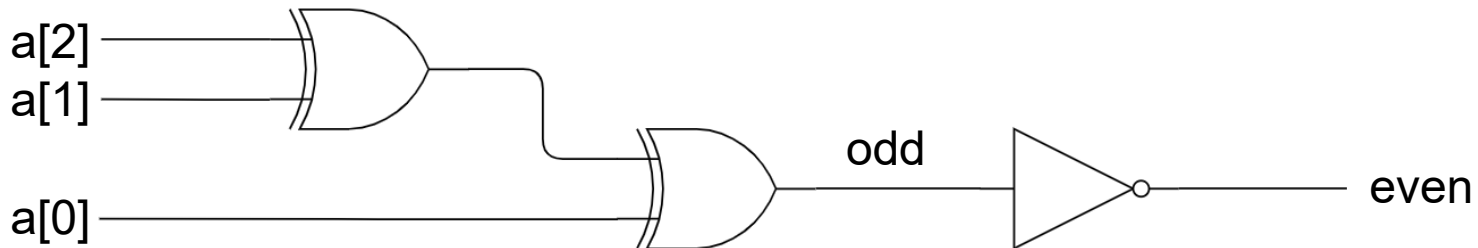
| |
|---|
| 설계 기술 시작 |
| 포트 기술<br>• 입력: 3-bit 신호<br>• 출력: 1-bit 신호 |
| Net 기술 |
| 조합 회로 합성됨 |
| 설계 기술 끝 |

# Verilog 연산자

- 산술연산자

| 기호 | 기능 | 피연산자 수 | 예 |
|:---:|:---:|:---:|:---:|
| + | 더하기 | 2 | 4 + 3 = 7 |
| − | 빼기 | 2 | 5 − 4 = 1 |
| * | 곱하기 | 2 | 3 * 7 = 21 |
| / | 나누기 | 2 | 18 / 2 = 9 |
| % | 나머지(modulo) | 2 | 25 % 3 = 1 |
| ** | 거듭제곱(power) | 2 | 3 ** 3 = 27 |

# Verilog 연산자

- 시프트(shift)연산자

| 기호 | 기능 | 피연산자 수 | 예 |
|---|---|---|---|
| >> | 논리 오른쪽 시프트 | 2 | 0111 >> 2 = 0001<br>1000 >> 2 = 0010 |
| << | 논리 왼쪽 시프트 | 2 | 0111 << 2 = 1100<br>1000 << 2 = 0000 |
| >>> | 산술 오른쪽 시프트 | 2 | 0111 >>> 2 = 0001<br>1000 >>> 2 = 1110 |
| <<< | 산술 왼쪽 시프트 | 2 | 0111 <<< 2 = 1100<br>1000 <<< 2 = 0000 |

# Verilog 연산자

- 관계연산자

| 기호 | 기능 | 피연산자 수 | 예 |
|---|---|---|---|
| > | ~보다 크다 | 2 | 4 > 1 = true (1'b1) |
| < | ~보다 작다 | 2 | 9 < 7 = false (1'b0) |
| >= | ~보다 크거나 같다 | 2 | 5 >= 5 = true |
| <= | ~보다 작거나 같다 | 2 | 6 <= 9 = true |
| == | ~보다 같다 (논리 등가) | 2 | 9 == 8 = false |
| != | ~보다 다르다 (논리 부등) | 2 | 1 != 45 = true |
| === | case 등가 | 2 | 4'b1z0x === 4'b1z0x = true |
| !== | case 부등 | 2 | 4'b1z0x !== 4'b1z0x = false |

# Verilog 연산자

- 비트연산자

| 기호 | 기능 | 피연산자 수 | 예 |
|---|---|---|---|
| ~ | 비트 부정 | 1 | ~4'b0011 = 4'b1100 |
| & | 비트 and | 2 | 2'b01 & 2'b11 = 2'b01 |
| \| | 비트 or | 2 | 2'b10 \| 2'b01 = 2'b11 |
| ^ | 비트 xor | 2 | 2'b00 ^ 2'b11 = 2'b11 |
| & | 축약 and | 1 | &4'b0111 = 1'b0 |
| \| | 축약 or | 1 | \|4'b1100 = 1'b1 |
| ^ | 축약 xor | 1 | ^4'b1010 = 1'b0 |

# Verilog 연산자

- 논리연산자

| 기호 | 기능 | 피연산자 수 | 예 |
|------|------|-------------|-----|
| ! | 논리 부정 | 1 | !4'b0010 = false (1'b0) |
| && | 논리 and | 2 | 2'b01 && 2'b10 = true (1'b1) |
| \|\| | 논리 or | 2 | 2'b00 \|\| 3'b000 = false (1'b0) |

- 기타연산자

| 기호 | 기능 | 피연산자 수 | 예 |
|------|------|-------------|-----|
| { } | 결합 | 아무 개수 | {3'b001, 2'b11, 1'b0} = 6'b001110 |
| {{ }} | 반복 | 아무 개수 | {3{3'b100}} = 9'b100100100 |
| ? : | 조건 | 3 | (3'b110 > 3'b011) ? 1'b1 : 1'b0 = 1'b1 |

# Verilog 연산자

■ 회로 합성

| 연산자 | | 회로 합성 |
|---|---|---|
| 산술 | + | **가능** |
| | − | |
| | * | 일반적으로 **가능**,<br>고성능 컴퓨팅을 위해 직접 설계 필요 |
| | / | 일반적으로 **불가능**, 직접 설계 필요 |
| | % | |
| | ** | |

# Verilog 연산자

- 회로 합성

| 연산자 | 회로 합성 |
|:---:|:---:|
| 시프트 | **가능**, 단순한 "선 연결" |
| 결합 | |
| 반복 | |
| 관계 | **가능**, 비교기(comparator) |
| 비트 | **가능**, 논리 게이트 |
| 논리, 조건 | **가능**, MUX(multiplexer) |

# Verilog 연산자

- 회로 합성 (논리, 조건 연산자)



```
assign out = (a == b) ? (in + 1) :
             (a > b ) ? (in + 2) :
                        (in + 3);
```
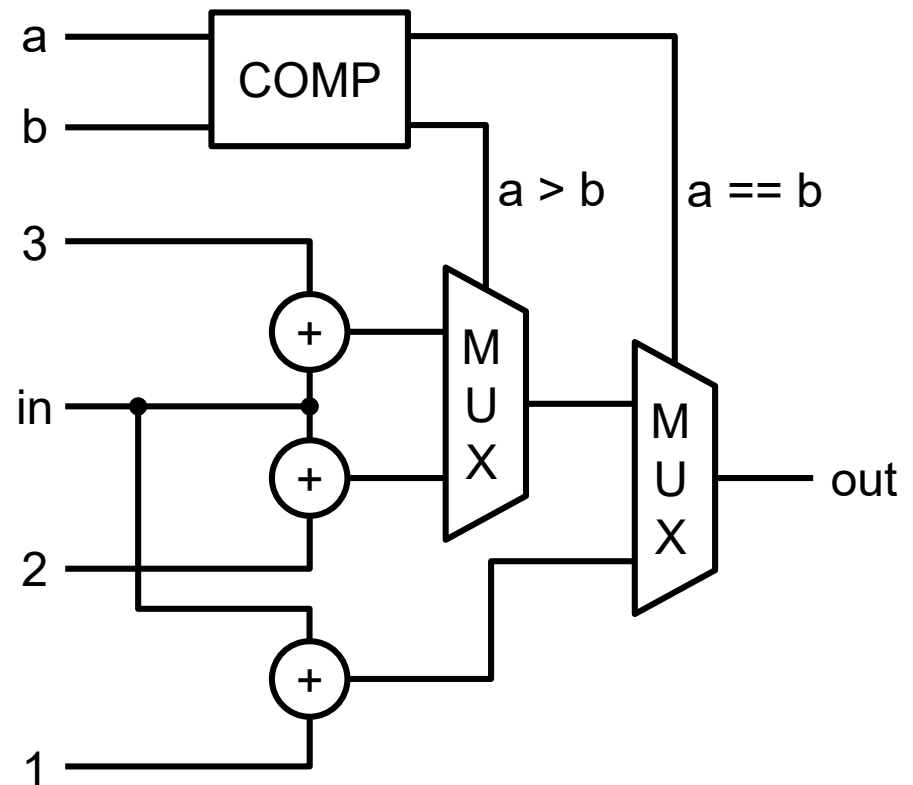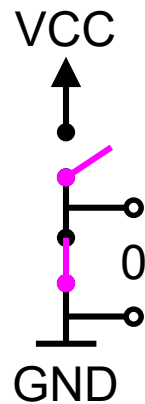
# Verilog 연산자

- 연산자의 우선순위

| | |
|:---:|:---:|
| !, ~, +, − (단항) | **제일 높음** |
| ** | |
| *, /, % | │ |
| +, − (이항) | │ |
| >>, <<, >>>, <<< | │ |
| >, >=, <, <= | │ |
| ==, !=, ===, !== | │ |
| & | │ |
| ^ | │ |
| \| | │ |
| && | ↓ |
| \|\| | |
| ? : | **제일 낮음** |

# z 및 x

- z (high impedance)

VCC
0

GND

VCC
1

GND

VCC
z

GND

dir

in

out

tri-state buffer
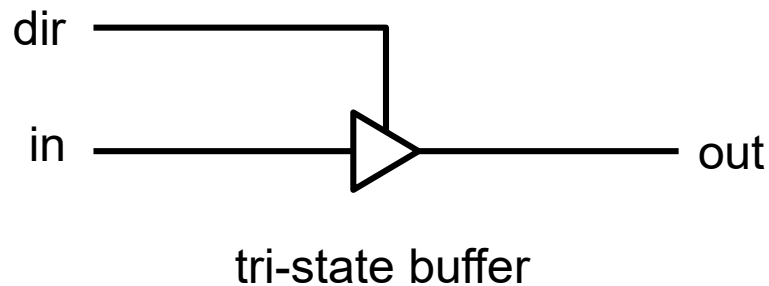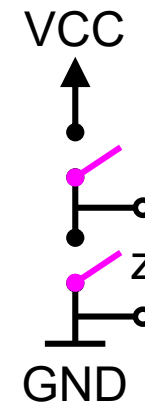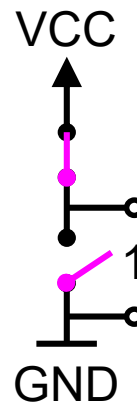
```
assign out = (dir) ? in : 1'bz;
```

| dir | out |
|-----|-----|
| 0   | z   |
| 1   | in  |

# z 및 x

- Bidirectional I/O port
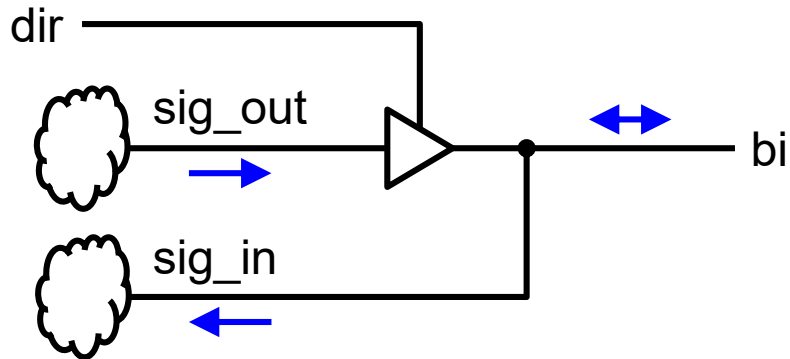


```verilog
module bi_io_port (
    input in,
    input dir,
    inout bi
);

wire sig_in;

assign bi = (dir) ? in : 1'bz;
assign sig_in = bi;

endmodule
```

# z 및 x

- x (don't care)
  - Verilog 코드 작성 시 실제로 나타나지 않은 패턴들을 x로 표시할 수 있음
  - x에 대해 시뮬레이션과 합성의 차이가 있음
    - 합성 시 회로 최적화를 위해 x를 0이나 1로 설정됨
    - 시뮬레이션 시 x는 0이나 1이 아닌 특정 값으로 할당됨
      - "unknown" 혹은 초기화 되지 않다는 의미
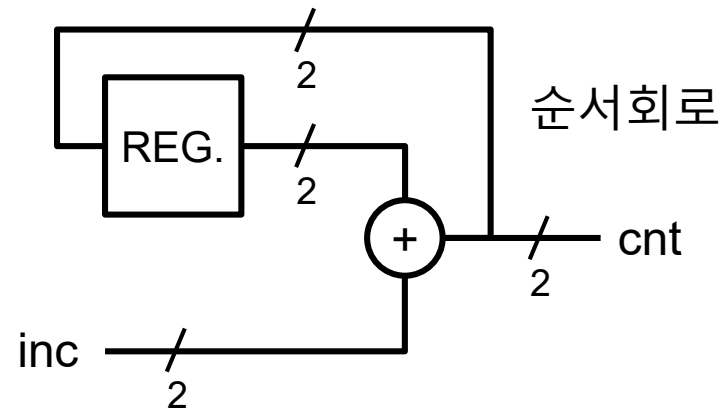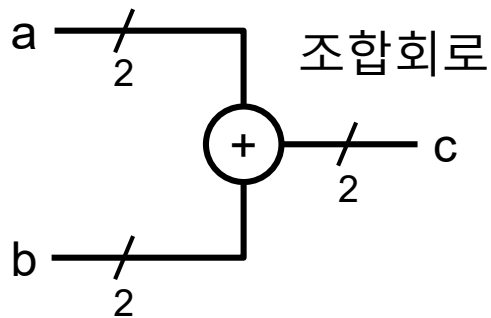
| 입력 | | 출력 |
|---|---|---|
| a | b | c |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | x |

```verilog
assign c = (a==1'b0 && b==1'b0) ? 1'b0 :
           (a==1'b0 && b==1'b1) ? 1'b1 :
           (a==1'b1 && b==1'b0) ? 1'b1 :
                                  1'bx;
```

회로 최적화를 위해 합성 시
a = 1'b1, b = 1'b1 → c = 1'b1을 사용

# 조합 및 순서 회로

- 조합회로(combinational circuit)
  - 메모리(또는 상태)를 포함하지 않음
  - 같은 입력 → 같은 출력   } 출력 = $f$(입력)

- 순서회로(sequential circuit)
  - 메모리(또는 상태)를 포함함
  - 같은 입력 → 회로 상태에 따라 출력 달라질 수 있음   } 출력 = $f$(입력, 상태)

a —/2—
         (+)—/2— c
b —/2—
             조합회로

REG. 순서회로
inc —/2—
             (+)—/2— cnt

# initial 및 always 블록

- initial 블록
  - 한 번만 실행
  - 시뮬레이션 때만 사용

- always 블록
  - 반복 실행
  - 설계 및 시뮬레이션 때 사용 가능

```
initial
begin
    statements;
end


always @(sensitivity_list)
begin
    statements;
end
```

procedural assignments

```
assign y1 = x1 & x2
assign y2 = x1 | x2
```

continuous assignments

# continuous 및 procedural

- 예: y = a & b & c

```verilog
module cont_assign (
    input   a, b, c,
    output y
);

assign y = a & b & c;

endmodule
```

```verilog
module proc_assign (
    input   a, b, c,
    output reg y
);

always @(a, b, c)
begin
    y = a & b & c;
end

endmodule
```
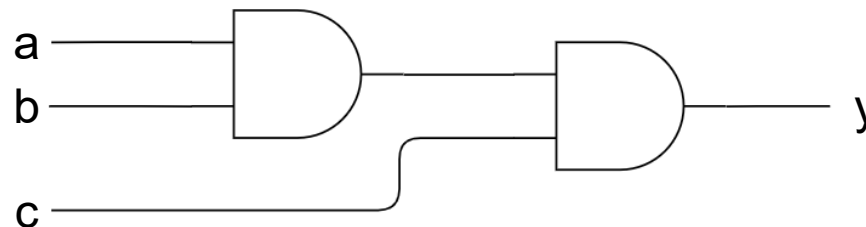
```verilog
module proc_assign (
    input   a, b, c,
    output reg y
);

always @*
begin
    y = a & b & c;
end

endmodule
```
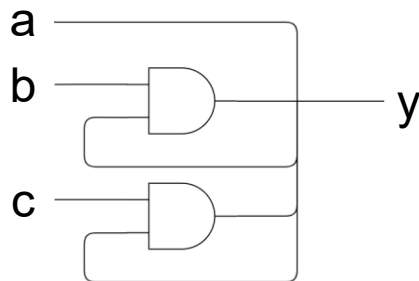
# continuous 및 procedural

- 예: y = a & b & c

```verilog
module cont_assign (
    input  a, b, c,
    output y
);

assign y = a;
assign y = y & b;
assign y = y & c;

endmodule
```
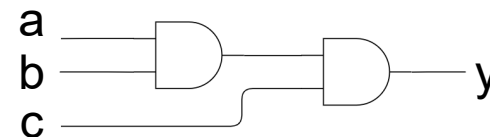
```verilog
module proc_assign (
    input  a, b, c,
    output reg y
);

always @*
begin
    y = a;
    y = y & b;
    y = y & c;
end

endmodule
```

# if … else …

```
if condition
   begin
      statements;
   end
else
   begin
      statements;
   end


assign out = (a == b) ? (in + 1) :
             (a > b ) ? (in + 2) :
                        (in + 3);


if (a == b)
   out = in + 1;
else if (a > b)
   out = in + 2;
else
   out = in + 3;
```
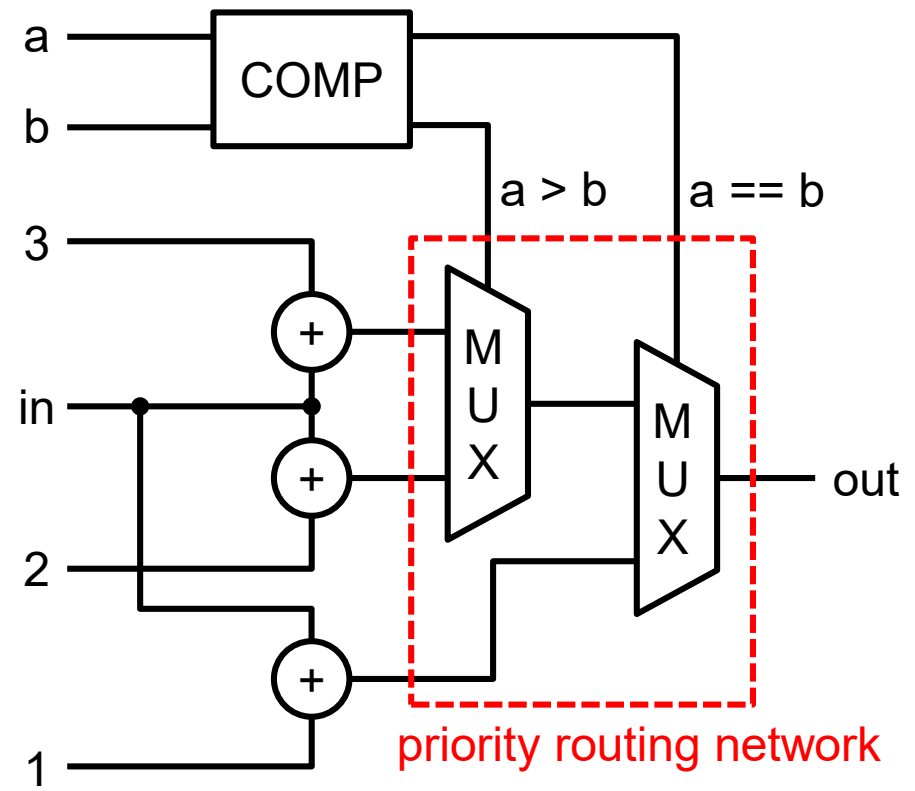


priority routing network

# case

```
case expression
    item:
        begin
            statements;
        end
    item:
        begin
            statements;
        end
    item:
        begin
            statements;
        end
    default:
        begin
            statements;
        end
endcase
```

```
case (sel)
    2'b00: out = a;
    2'b01: out = b;
    2'b10: out = c;
    2'b11: out = d; // default 사용 가능
endcase
```

multiplexing network