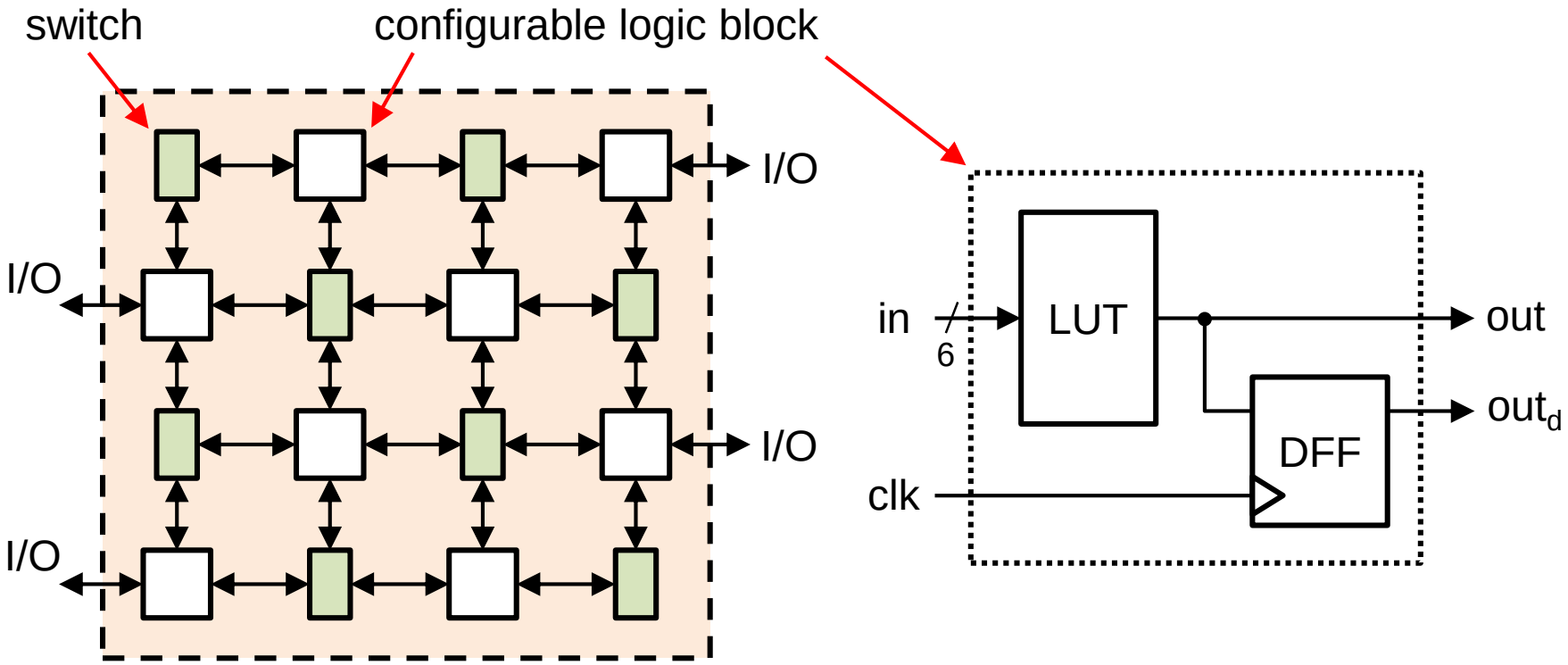
Lecture 02

FPGA 장치와 EDA 툴 소개

FPGA 장치

- Configurable logic blocks + Programmable switch



FPGA 장치

- 실제 FPGA 장치 – Zynq UltraScale

  https://docs.xilinx.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview

AMD XILINX — Zynq UltraScale+ MPSoC Data Sheet: Overview

Programmable Logic (PL)

Configurable Logic Blocks (CLB)
- Look-up tables (LUT)
- Flip-flops
- Cascadable adders

36Kb Block RAM
- True dual-port
- Up to 72 bits wide
- Configurable as dual 18Kb

UltraRAM
- 288Kb dual-port
- 72 bits wide
- Error checking and correction

DSP Blocks
- 27 x 18 signed multiply
- 48-bit adder/accumulator
- 27-bit pre-adder

Programmable I/O Blocks
- Supports LVCMOS, LVDS, and SSTL
- 1.0V to 3.3V I/O
- Programmable I/O delay and SerDes

JTAG Boundary-Scan
- IEEE Std 1149.1 Compatible Test Interface

PCI Express
- Supports Root complex and End Point configurations
- Supports up to Gen3 speeds
- Up to five integrated blocks in select devices

100G Ethernet MAC/PCS
- IEEE Std 802.3 compliant
- CAUI-10 (10x 10.3125Gb/s) or CAUI-4 (4x 25.78125Gb/s)
- RSFEC (IEEE Std 802.3bj) in CAUI-4 configuration
- Up to four integrated blocks in select devices

Interlaken
- Interlaken spec 1.2 compliant
- 64/67 encoding
- 12 x 12.5Gb/s or 6 x 25Gb/s
- Up to four integrated blocks in select devices

Video Encoder/Decoder (VCU)
- Available in EV devices
- Accessible from either PS or PL
- Simultaneous encode and decode
- H.264 and H.265 support

System Monitor in PL
- On-chip voltage and temperature sensing
- 10-bit 200KSPS ADC with up to 17 external inputs

AMD XILINX — Zynq UltraScale+ MPSoC Data Sheet: Overview

Feature Summary

Table 1: **Zynq UltraScale+ MPSoC: CG Device Feature Summary**

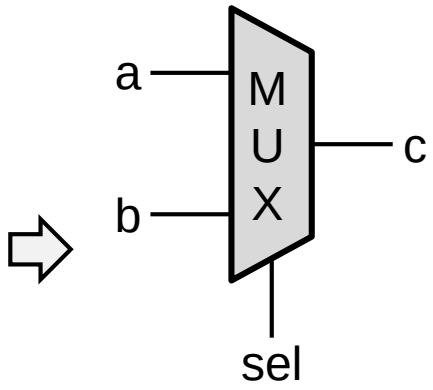
	ZU1CG	ZU2CG	ZU3CG	ZU3TCG	ZU4CG	ZU5CG	ZU6CG	ZU7CG	ZU9CG
Application Processing Unit	Dual-core Arm Cortex-A53 MPCore with CoreSight; NEON & Single/Double Precision Floating Point; 32KB/32KB L1 Cache, 1MB L2 Cache								
Real-Time Processing Unit	Dual-core Arm Cortex-R5F with CoreSight; Single/Double Precision Floating Point; 32KB/32KB L1 Cache, and TCM								
Embedded and External Memory	256KB On-Chip Memory w/ECC; External DDR4; DDR3; DDR3L; LPDDR4; LPDDR3; External Quad-SPI; NAND; eMMC								
General Connectivity	214 PS I/O; UART; CAN; USB 2.0; I2C; SPI; 32b GPIO; Real Time Clock; WatchDog Timers; Triple Timer Counters								
High-Speed Connectivity	4 PS-GTR; PCIe Gen1/2; Serial ATA 3.1; DisplayPort 1.2a; USB 3.0; SGMII								
System Logic Cells	81,900	103,320	154,350	157,500	192,150	256,200	469,446	504,000	599,550
CLB Flip-Flops	74,880	94,464	141,120	144,000	175,680	234,240	429,208	460,800	548,160
CLB LUTs	37,440	47,232	70,560	72,000	87,840	117,120	214,604	230,400	274,080
Distributed RAM (Mb)	1.0	1.2	1.8	2.1	2.6	3.5	6.9	6.2	8.8
Block RAM Blocks	108	150	216	144	128	144	714	312	912
Block RAM (Mb)	3.8	5.3	7.6	5.1	4.5	5.1	25.1	11.0	32.1
UltraRAM Blocks	0	0	0	48	48	64	0	96	0
UltraRAM (Mb)	0	0	0	14.0	13.5	18.0	0	27.0	0
DSP Slices	216	240	360	576	728	1,248	1,973	1,728	2,520
CMTs	3	3	3	1	4	4	4	8	4
Max. HP I/O[1]	156	156	156	52	156	156	208	416	208
Max. HD I/O[2]	24	96	96	72	96	96	120	48	120
System Monitor	2	2	2	2	2	2	2	2	2
GTH Transceiver[3]	0	0	0	8	16	16	24	24	24
GTY Transceivers	0	0	0	–	0	0	0	0	0
Transceiver Fractional PLLs	0	0	0	4	8	8	12	12	12
PCI Express	0	0	0	1	2	2	0	2	0
150G Interlaken	0	0	0	1	0	0	0	0	0
100G Ethernet w/ RS-FEC	0	0	0	–	0	0	0	0	0

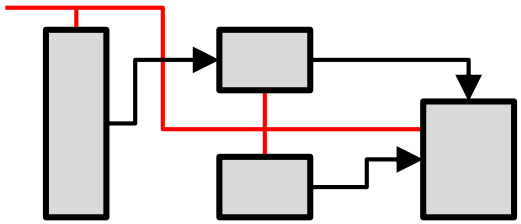
EDA 툴 (EDA: Electronic Design Automation)

- 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

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

Standard cell library





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

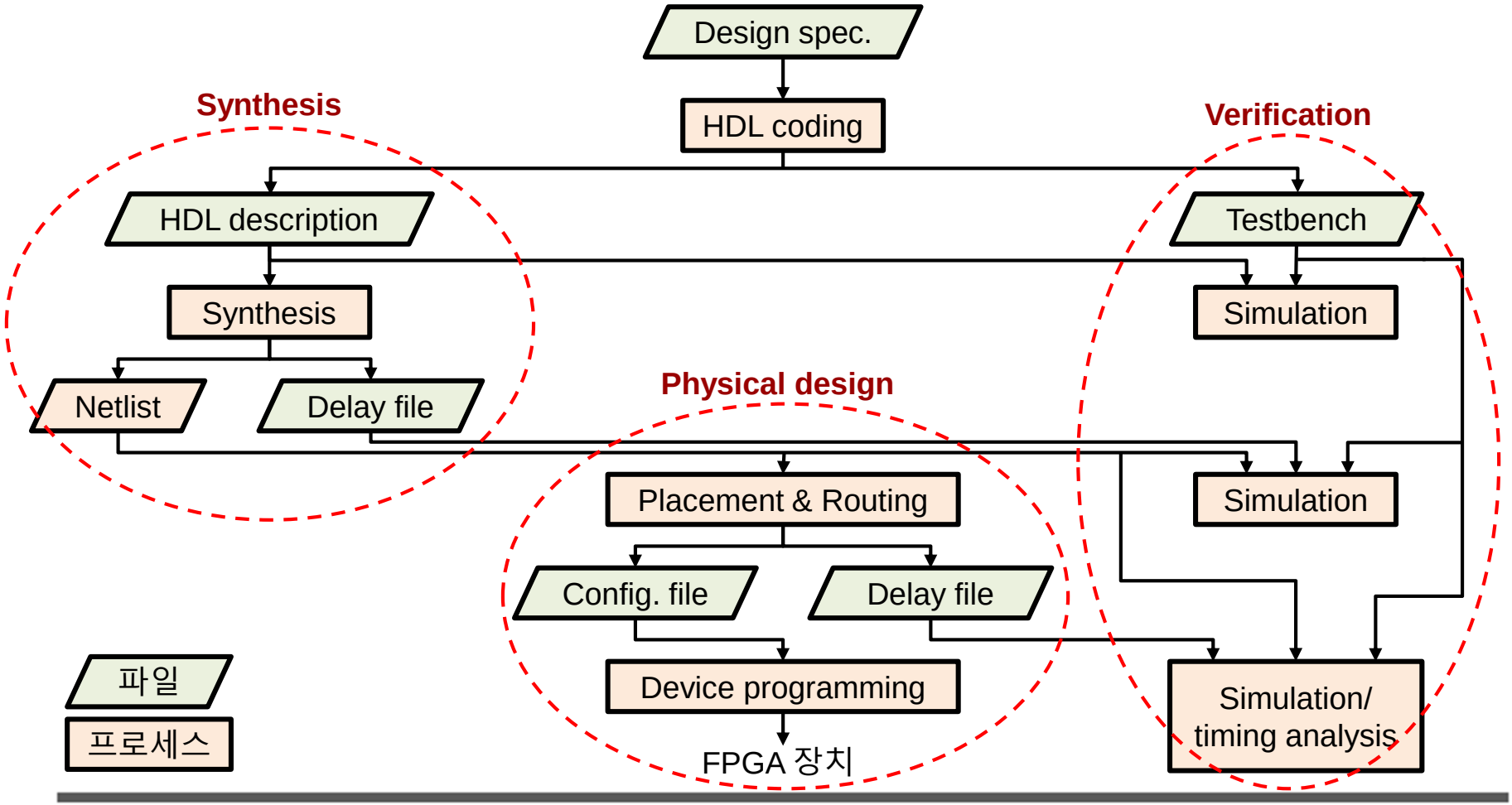
+ bare-metal programming
= **Vitis**

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html

+ Linux embedded programming
= **PetaLinux**

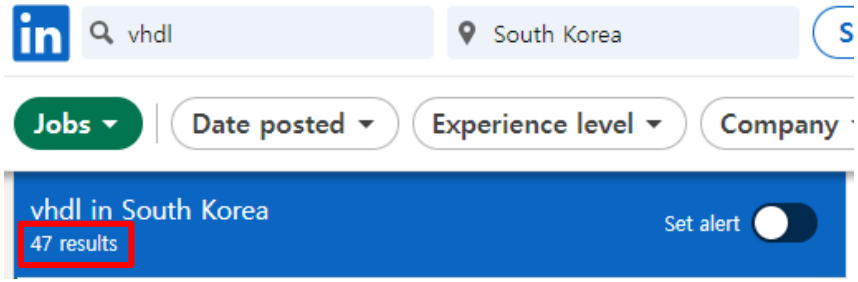
https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html

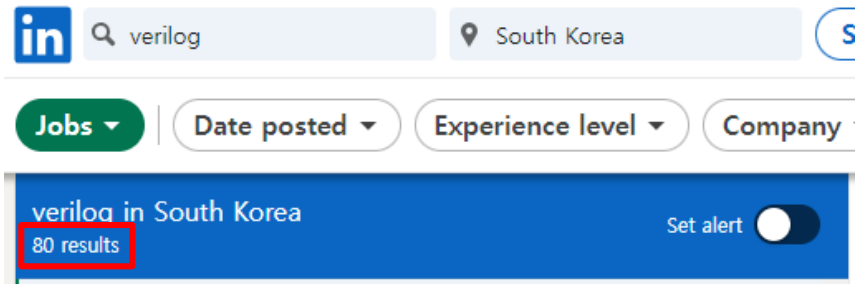
FPGA 설계 과정

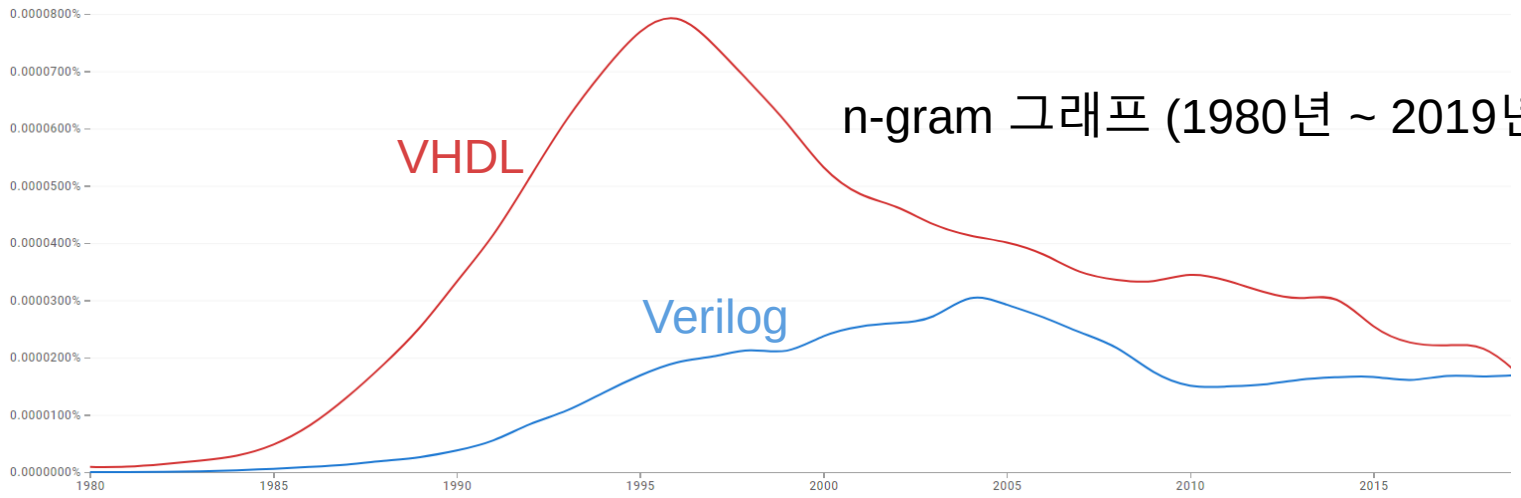

Design spec.
HDL coding
Synthesis
HDL description
Synthesis
Netlist
Delay file
Verification
Testbench
Simulation
Physical design
Placement & Routing
Config. file
Delay file
Device programming
FPGA 장치
Simulation
Simulation/
timing analysis
파일
프로세스

Verilog HDL

- VHDL 있는데, 왜 Verilog?







n-gram 그래프 (1980년 ~ 2019년)

Verilog HDL

- 반도체 시장 현황

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

```
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

```
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

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

wire odd;

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

endmodule
```

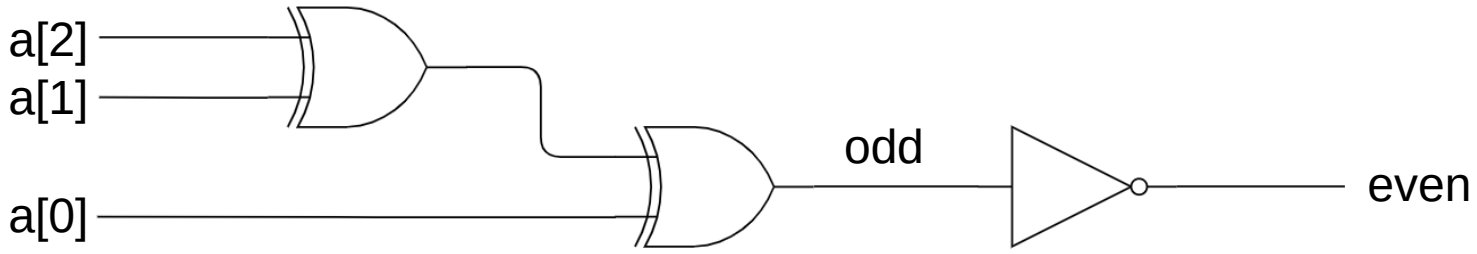
설계 기술 시작

포트 기술
- 입력: 3-bit 신호
- 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 1000 >> 2 = 0010
<<	논리 왼쪽 시프트	2	0111 << 2 = 1100 1000 << 2 = 0000
>>>	산술 오른쪽 시프트	2	0111 >>> 2 = 0001 1000 >>> 2 = 1110
<<<	산술 왼쪽 시프트	2	0111 <<< 2 = 1100 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**, 설계 시 더하기(+) 및 빼기(–)를 바로 사용
	–	
	*	일반적으로 **가능**, 고성능을 위해 직접 설계 필요
	/	일반적으로 **불가능**, 직접 설계 꼭 필요
	%	
	**	

Verilog 연산자

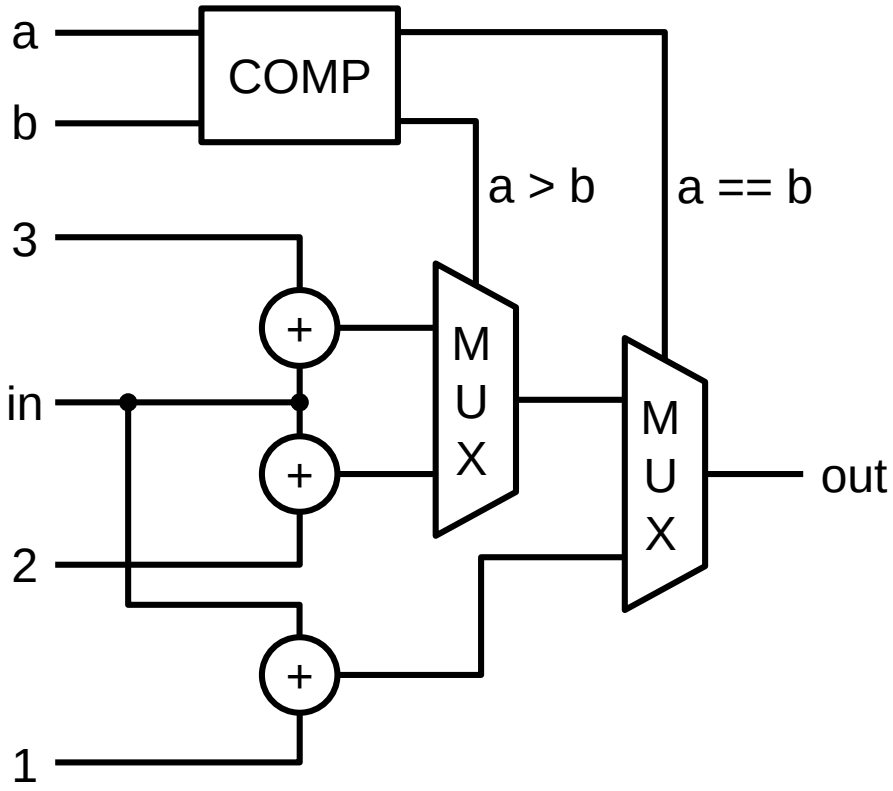
- 회로 합성

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);
```



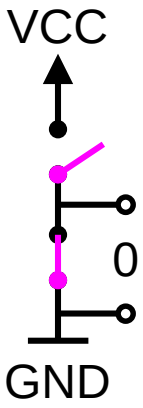
Verilog 연산자

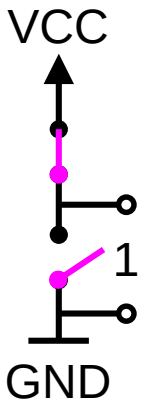
- 연산자의 우선순위

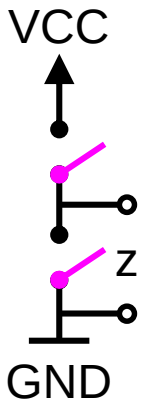
!, ~, +, – (단항)	**제일 높음**
**	
*, /, %	
+, – (이항)	
>>, <<, >>>, <<<	
>, >=, <, <=	
==, !=, ===, !==	
&	
^	
|	
&&	
||	
? :	**제일 낮음**

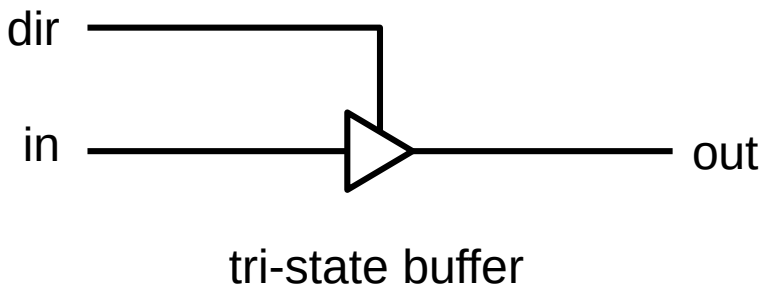
z 및 x

- z (high impedance)







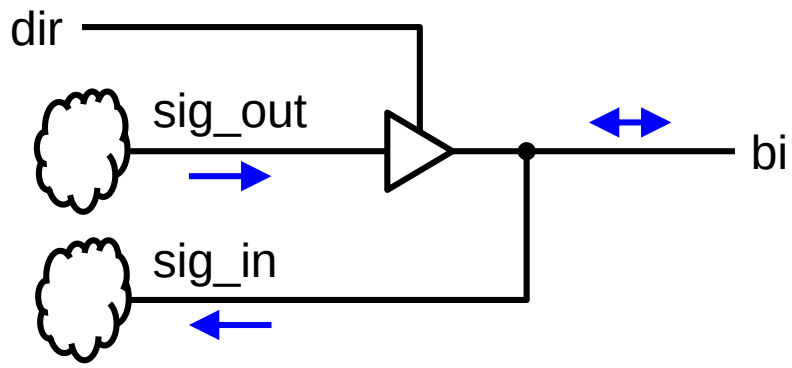



```
assign out = (dir) ? in : 1'bz;
```

dir	out
0	z
1	in

z 및 x

- Bidirectional I/O port




```
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 작상 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 0과 1 대신 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

```
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;
```

합성 시 a = 1'b1, b = 1'b1 → c = 1'b1

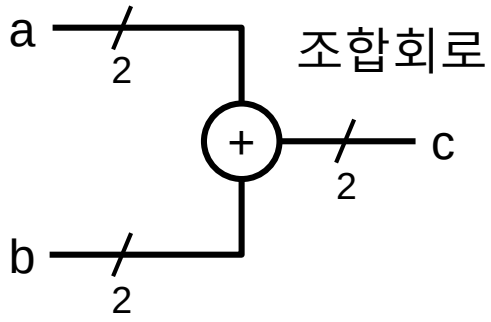
조합 및 순서 회로

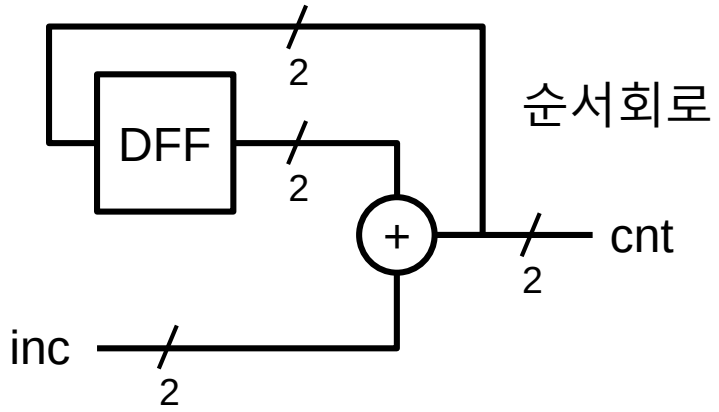
- 조합회로(combinational circuit)
  - 메모리(또는 상태)를 포함하지 않음
  - 같은 입력 → 같은 출력

  } 출력 = *f*(입력)

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

  } 출력 = *f*(입력, 상태)





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

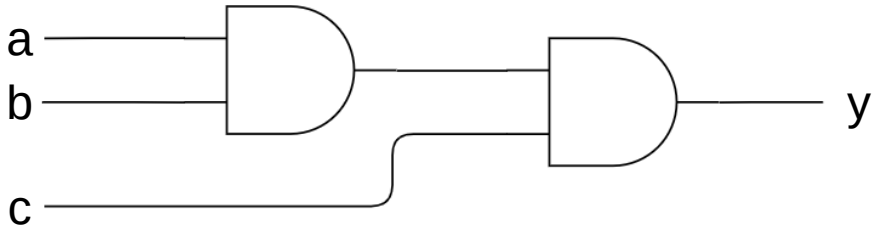
continuous 및 procedural

- 예: y = a & b & c

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

assign y = a & b & c;

endmodule
```

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

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

endmodule
```

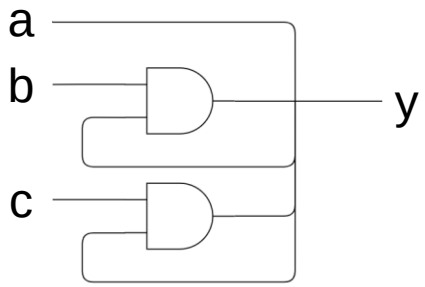
```
module proc_assign (
   input  a, b, c,
   output y
);

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

endmodule
```



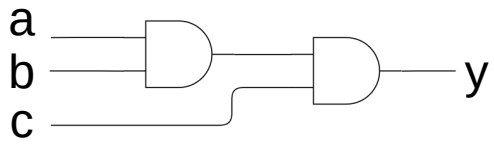
continuous 및 procedural

- 예: y = a & b & c

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

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

endmodule
```




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

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

endmodule
```

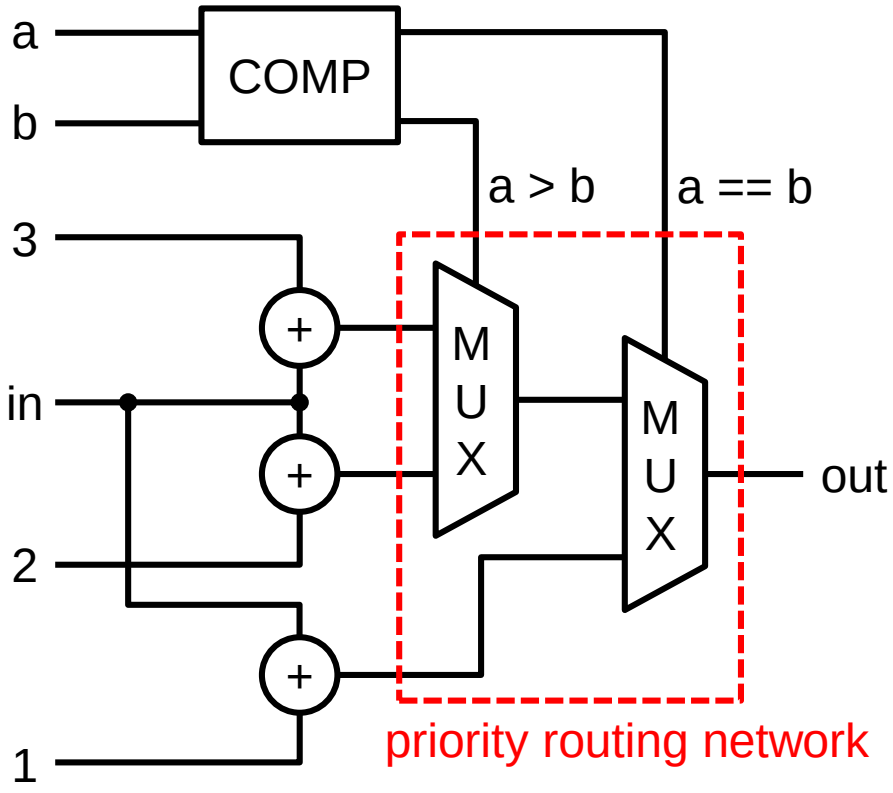


if … else …

```
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;
```



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