1. Verilog HDL 문법

Verilog HDL의 역사

- Verilog HDL
  - 1983년 Gateway Design Automation사에서 하드웨어 기술언어인 HiLo와 C 언어의 특징을 기반으로 개발
  - 1991년 Cadence Design Systems가 Open Verilog International (OVI)라는 조직을 구성하고 Verilog HDL을 공개
  - 1993년 IEEE Working Group이 구성되어 표준화 작업을 진행
  - 1995년 12월 IEEE Std. 1364-1995로 표준화
  - 2001년에 IEEE Std. 1364-2001로 개정
  - Verilog HDL의 확장 형태인 SystemVerilog가 개발되어 IEEE 표준화를 추진

Verilog HDL과 C 언어

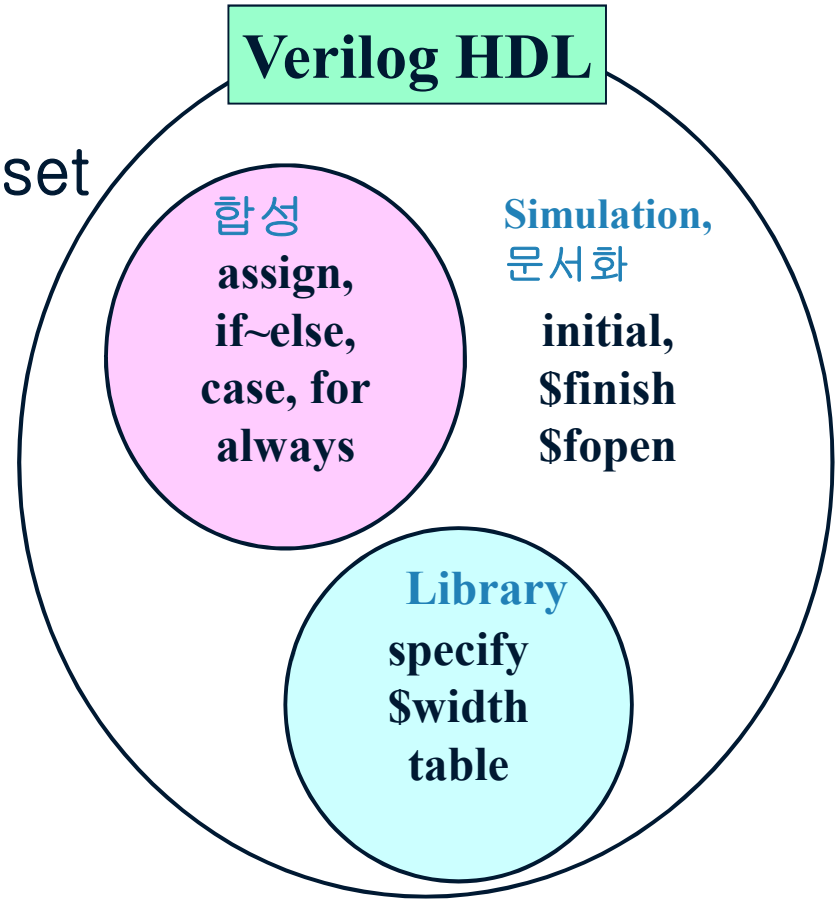
- C 언어와 유사점
  - 문법은 매우 유사
  - 절차형(procedural) 및 순차형 실행은 동일
  - if~else문 , case 문 , for loop 문 등 사용
  - 연산자 : 논리 (&, |, ^..), 산술(+, −, * /..), 비교 (<, >=, ==, ..)
- HDL 언어의 특성
  - 동시성(concurrent)
  - 병렬성 (parallel)
  - 추상화 (abstraction)

HDL의 3가지 Modeling

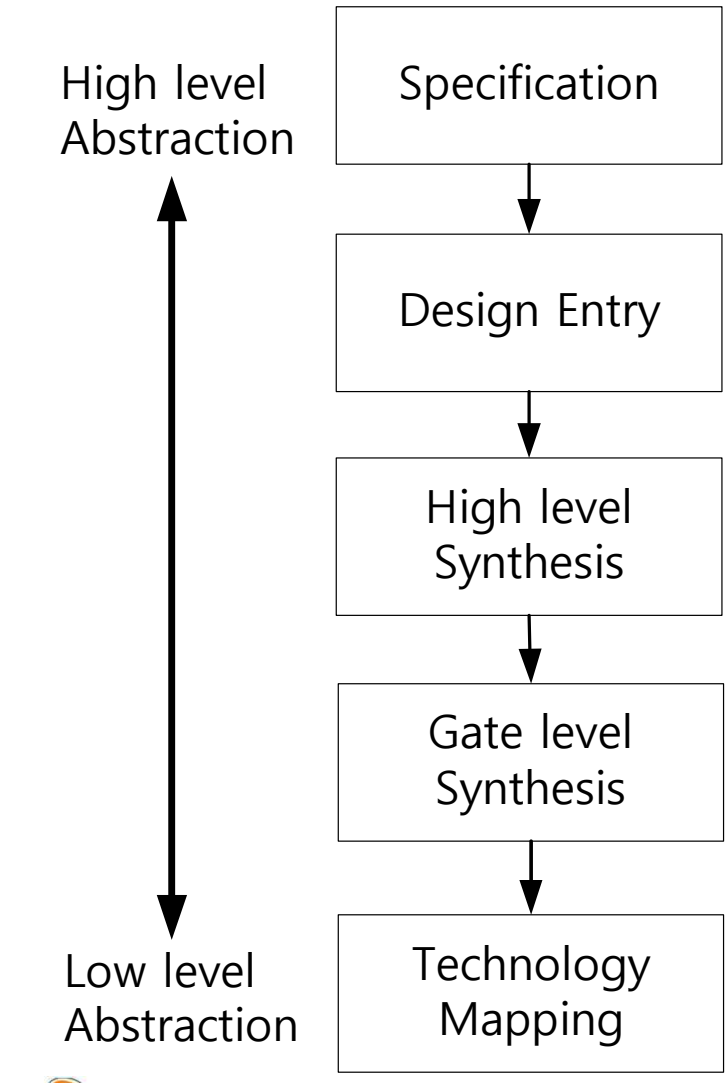
- 구조적 (Structural) 모델링
  - 논리 게이트, 플립플롭 등을 사용한 연결도 표현
  - 기존 설계한 회로를 포함한 네트리스트(netlist) 사용
- 데이터플로우 (dataflow) 모델링
  - 데이터 이동을 표현
  - 연산자를 사용한 연속할당문
- 동작적(behavioral) 모델링
  - if~else, case, while, for 등과 같은 구문 사용
  - 인간의 사고에 가장 근접한 표현

Verilog 언어의 기능

- 시뮬레이션 (simulation) : full set
- 합성(synthesis) : subset
- Library : system 함수
- 문서화 (documentation) : full set



설계흐름도와 추상화레벨



사양: 동작모델,속도,전력,가격, 소형화..
C, System C, SystemVerilog

Behavioral Simulation

설계입력 : 블록도와 입출력 신호 연결도
그래픽 혹은 문서입력
Verilog, VHDL

Functional (RTL) Simulation

RTL : 동작적 표현, 클럭 타이밍
Adder, 디코더, ALU, 레지스터..
Verilog, VHDL

Gate level Simulation

게이트레벨,: 논리, 프리미티브 게이트 연결도
Verilog, VHDL

Timing Simulation

하드웨어 : SoC, ASIC, FPGA.
배치및배선, 타이밍검증
EDIF, JEDEC

Verilog 첫걸음

반가산기(Half Adder) 설계

진리표

x y	co s
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

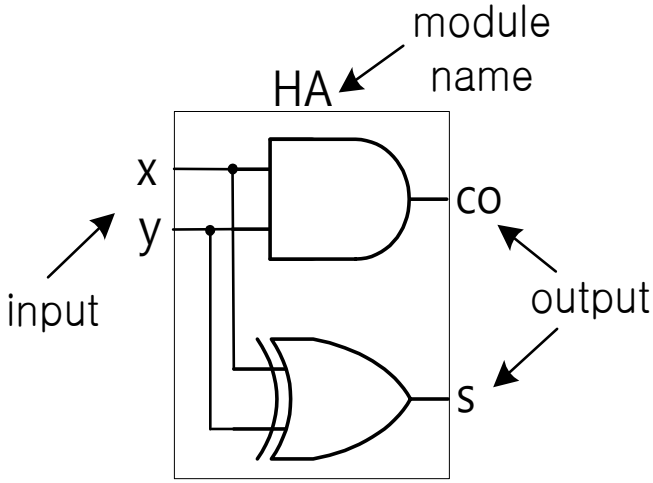
논리식

S = x xor y

co = xy

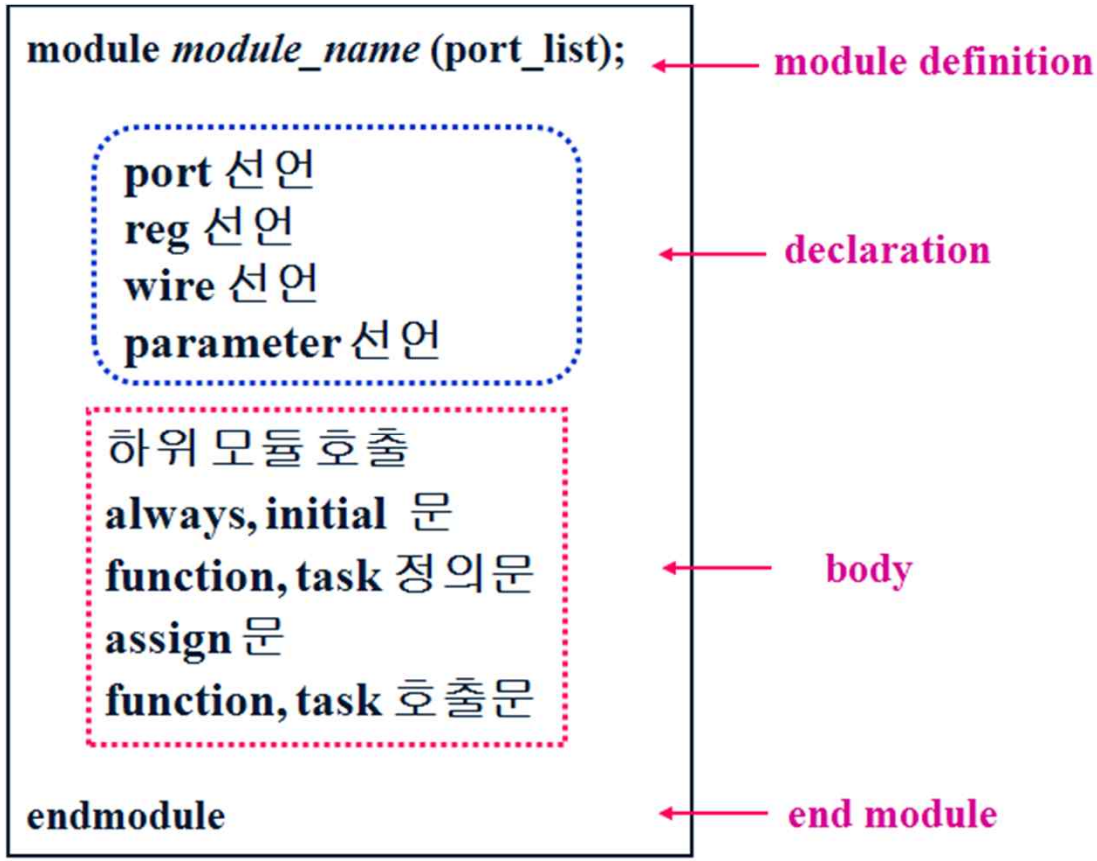
모듈정의

```
module HA (x, y, co, s);
input x;
input y;
output co, s;
```



반가산기(Half Adder) 설계

❑ 모듈구조



반가산기(Half Adder) 설계

❑모델링

```
// code1.2.1-1 : HA_s.v
module HA_s (x, y, co, s);
input x;
input y;
output co, s;
// 하위모듈 호출, 구조적 모델링
  and U1 (co, x, y); // 프리미티브 게이트 인스턴스
  xor U2 (s, x, y);
endmodule
```

```
// code 1.2.1-2 : HA_d.v
module HA_d (x, y, co, s);
input x;
input y;
output co, s;
// 연속할당문, 데이터플로우 모델링
   assign s = x ^ y ;  // bitwise XOR 논리연산자
   assign co = x & y; // bitwise AND 논리연산자
endmodule
```

반가산기(Half Adder) 설계

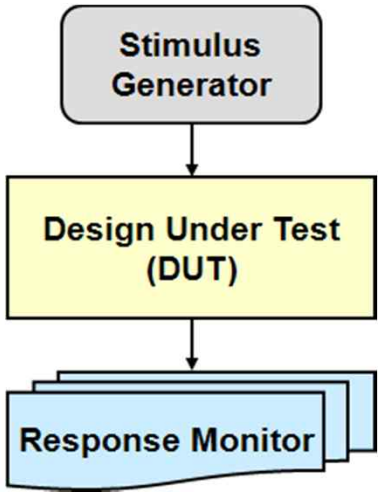
❑모델링

```
// code 1.2.1-3 : HA_b.v
module HA_b (x, y, co, s);
input x;
input y;
output co, s;
reg co, s;
// 절차형 블록문, 동작적 모델링
always @ (x or y)
begin  s = x ^ y ; // blocking 할당문
          co = x & y;
end
endmodule
```

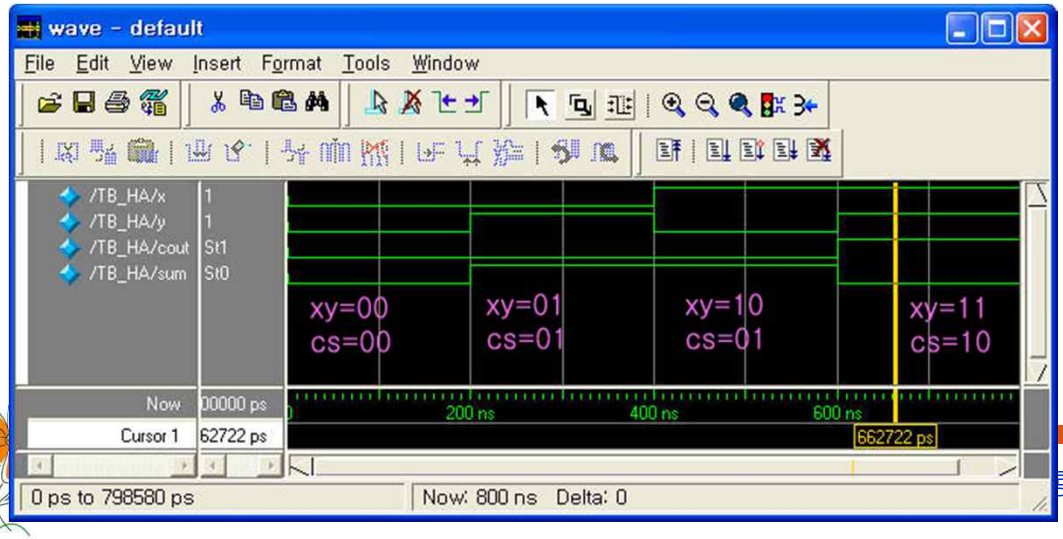
```
// code 1.2.1-4 : HA_t.v
module HA_t (x, y, co, s);
input x, y;
output co, s;
reg co, s;
// 절차형 블록문, 진리표의 동작적 모델링
   always @ (x or y)
       case ({x,y})
       2'b00 : {co,s} = 2'b00;
       2'b01 : {co,s} = 2'b01;
       2'b10 : {co,s} = 2'b01;
       2'b11 : {co,s} = 2'b10;
       default : {co,s} = 2'b00;
       endcase
endmodule
```

설계 검증

❑ 테스트벤치 – 시뮬레이션을 위한 입력 여기(stimulus)파형



시뮬레이션 파형




```
// code 1.2.2-1 : TB_HA.v
`timescale 1ns/1ps // 시뮬레이션 시간단위
module TB_HA;
reg x, y;
wire cout, sum;
// DUT instance
HA_b U1 (.x(x), .y(y), .co(cout), .s(sum) );
// input stimulus
initial begin
    x=0; y=0; #200; // 200 ns 지연
    x=0; y=1; #200; // 200 ns 지연
    x=1; y=0; #200;
    x=1; y=1; #200;
    $finish; // 시뮬레이션 종료
end  endmodule
```

1.3 Verilog 어휘 규칙

- 어휘 토큰 (lexical tokens)
  - 여백(white space)
  - 주석(comment)
  - 연산자(operator)
  - 수(number)
  - 문자열(string)
  - 식별자(identifier)
  - 키워드(keyword)

Verilog 어휘 규칙

- 여백(white space)
  - 빈칸(space), 탭(tab), 줄바꿈
  - 어휘 토큰들을 분리하기 위해 사용되는 경우를 제외하고는 무시
  - 공백(blank)과 탭은 문자열에서 의미 있게 취급
- 주석(comment)
  - HDL 소스코드의 설명을 위해 사용되며, 컴파일과정에서 무시됨
  - 단일 라인 주석문; // 로 시작되어 해당 라인의 끝까지
  - 블록 주석문; /* ~ */ 로 표시
    - 블록 주석문은 내포(nested)될 수 없음
- 연산자(operator)
  - 단항 연산자, 2항 연산자, 3항 연산자

Verilog 어휘 규칙

- 식별자(identifier)
  - 객체에 고유의 이름을 지정하기 위해 사용
  - 대소문자를 구별하여 인식
  - 가독성을 위해 밑줄 사용 가능
  - 단순 식별자; 일련의 문자, 숫자, 기호 $, 밑줄 등으로 구성
    - 첫번째 문자는 숫자나 기호 $ 사용 불가, 문자 또는 밑줄만 사용
  - 확장 식별자(escaped identifier);
    - ₩ (back slash)로 시작되며, 여백(빈칸, 탭, 줄바꿈) 등으로 끝남
    - 프린트 가능한 ASCII 문자들을 식별자에 포함시키는 수단을 제공
- 키워드(keyword)
  - Verilog 구성 요소를 정의하기 위해 미리 정의된 식별자
  - 확장문자가 포함된 키워드는 키워드로 인식되지 않음

Verilog 어휘 규칙

예 : 유효한 식별자의 예

```
shiftreg_a
busa_index
error_condition
merge_ab
_bus3
n$657
```

예 : 확장 식별자의 예

```
\busa+index
\-clock
\***error-condition***
\net1/\net2
\{a,b}
\a*(b+c)
```

Verilog 어휘 규칙

❑ 수 표현 (number representation)

- ❖ 정수형(integer) ; 10진수, 16진수, 8진수, 2진수
- ❖ 형식 : `[size_constant]'<base_format> <number_value>`

  - ➢ **[size_constant]:** 값의 비트 크기를 나타내는 상수
    - ✓ 0이 아닌 unsigned 10진수가 사용되며, 생략될 수 있음
    - ✓ unsized 수 (즉, 단순 10진수 또는 비트 크기가 지정되지 않은 수)는 32비트로 표현됨
    - ✓ 상위 비트가 x(unknown) 또는 z(high-impedance)인 unsized unsigned 상수는 그 상수가 사용되는 수식의 비트 크기만큼 확장됨
  - ➢ **'base_format** : 밑수(base)를 지정하는 문자(d, D, h, H, o, O, b, B)
    - ✓ signed를 나타내기 위해 문자 s 또는 S가 함께 사용될 수 있음
  - ➢ **number_value** : unsigned 숫자를 사용하여 값을 표현
    - ✓ **'base_format**에 적합한 숫자들로 구성
  - ➢ **base_format**과 **number_value** 사이에 + 또는 - 부호 사용 불가

Verilog 어휘 규칙

- 수 표현 (number representation)
  - 비트 크기와 밑수를 갖지 않는 단순 10진수는 signed 정수로 취급
  - 부호 지정자 없이 밑수만 지정되면 unsigned 정수로 취급
  - 밑수 지정자와 부호 지정자 s가 함께 사용되면 signed 정수로 취급
    - 부호 지정자 s는 비트 패턴에는 영향을 미치지 않으며, 비트 패턴의 해석에만 영향을 미침
  - 음수는 2의 보수(2's complementary) 형식으로 표현됨
  - 지정된 비트 크기보다 unsigned 수의 크기가 작은 경우
    - MSB 왼쪽에 0이 삽입
    - MSB가 x 또는 z이면, x 또는 z가 왼쪽에 삽입
  - 값에 물음표(?)가 사용되면 z로 취급
  - 첫번째 문자를 제외하고는 밑줄(underscore)이 사용될 수 있으며, 이는 수의 가독성(readability)을 좋게 함

Verilog 어휘 규칙

Number	# of Bits	Base	Dec. Equiv.	Stored
10	32	Decimal	10	00....01010
2'b10	2	Binary	2	10
3'd5	3	Decimal	5	101
8'o5	8	Octal	5	00000101
8'ha	8	Hex	10	00001010
3'b5	Invalid!			
3'b01x	3	Binary	-	01x
12'hx	12	Hex	-	xxxxxxxxxxxx
8'b0000_0001	8	Binary	1	00000001
8'bx01	8	Binary	-	xxxxxx01
'bz	Unsized	Binary	-	zz...zz(32bits)
8'HAD	8	Hex	173	10101101

Verilog 어휘 규칙

예 1.3.1 : unsized 상수

```
1250            // 부호없는 10진수
'o765           // 부호없는 8진수
'sHab74         // 부호있는 16진수
'b1100_111      // 부호없는 2진수
'bz             // zz....zz (32 bits)
 4af            // illegal (hexadecimal format requires 'h)
```

예 1.3.2 : sized 상수

```
3'b110          // 3비트 2진수, 110
4'b1zx0         // 4비트 2진수, 1zx0
9'O513          // 9비트 8진수, 1_0100_1011
8'hef           // 8비트 16진수, 1110_1111
8'bz            // 8비트 2진수, z로 채워짐, zzzzzzzz
6'b1            // 6비트 2진수, 00_0001
6'shA           // 6비트 부호있는 16진수, 00_1010
6'h97           // 6비트 16진수, 01_0111, 상위 2비트 잘려짐
12'd15          // 12비트 10진수, 0000_0000_1111
```

Verilog 어휘 규칙

예1.3.3 : signed

```
8'shE0          // 1110_0000, -32에 대한 2의 보수,
                //  8'hE0와 동일
-8'shE0         // -(8'1110_0000), 0010000, 즉 +32
-'d100          // 2^32 - 100 = 4294967196
-'sd100         // -100 = 4294967196
-5'd6           // -6에 대한 5비트 2의 보수, 11010
4'sb1010        // -6
12'sb0010       // 0000_0000_0010, +2
```

예 : 밑줄을 사용한 수의 표현

```
27_195_000
16'b0011_0101_0001_1111
32'h12ab_f001
```

Verilog 어휘 규칙

예 : MSB 자동 삽입

```
reg [11:0] a, b, c, d;
initial begin
   a = 'hx;      // yields xxx
   b = 'h3x;     // yields 03x
   c = 'hz3;     // yields zz3
   d = 'h0z3;    // yields 0z3
end
```

Verilog 어휘 규칙

- 실수형(real) ; IEEE Std. 754-1985(IEEE standard for double-precision floating-point number)

표현형식

value.value	decimal notation
*base*E*exponent*	scientific notation (the E is not case sensitive)

예1.3.4 : 실수표현

```
0.5
679.00123
3e4
0.5e-0
5.6E-2
87E-4
93.432_26e-5
```

예 : 문법적 오류

```
.12
9.
4.E3
.2e-7
```

Verilog 어휘 규칙

- 문자열(string)
  - 이중 인용 부호(“ ”) 사이에 있는 일련의 문자들
  - 단일 라인에 존재해야 하며, 여러 라인에 걸친 문자열은 사용 불가
  - 8비트 ASCII 값으로 표현되는 unsigned 정수형 상수로 취급
  - 문자열 변수는 reg형의 변수이며, 문자열 내의 문자 수에 8을 곱한 크기의 비트 폭을 가짐

예1.3.6 : string 저장

```
reg [8*18:1] str1;

initial begin
   str1 = "Hello Verilog HDL!";
end
```

Verilog 어휘 규칙

예1.3.7 : string 저장 및 인쇄

```
module str_test;
reg [8*10:1] str1;
initial begin
    str1 = “Hello”;
    $display(“%s is stored as %h", str1, str1);
    str1 = {str1, " !!!"};
    $display(“%s is stored as %h", str1, str1);
end
endmodule
```

실행 결과

```
Hello is stored as 0000000000048656c6c6f
Hello !!! is stored as 0048656c6c6f20212121
```

Verilog 어휘 규칙

- 특수 문자 앞에 확장 문자(escaped character)를 사용하면 일부 특수 문자를 문자열에 포함시킬 수 있음

확장문자를 이용한 특수문자의 표현

확장 문자열	확장 문자열에 의해 생성되는 특수 문자
\n	New line character
\t	Tab character
\\	\ character
\"	" character
\ddd	A character specified in 1 ~ 3 octal digits (0≤d≤7)

Verilog 논리값 (Logic Value)

❖ 4 logic value

논리값	설 명
0	zero, low, or false
1	one, high, or true
z or Z	high impedance (tri-stated or floating)
x or X	unknown or uninitialized

논리 강도 (Logic strength)

- 8개의 논리강도 – 4개의 driving, 3개의 capacitive, 2개의 high impedance (no strength)

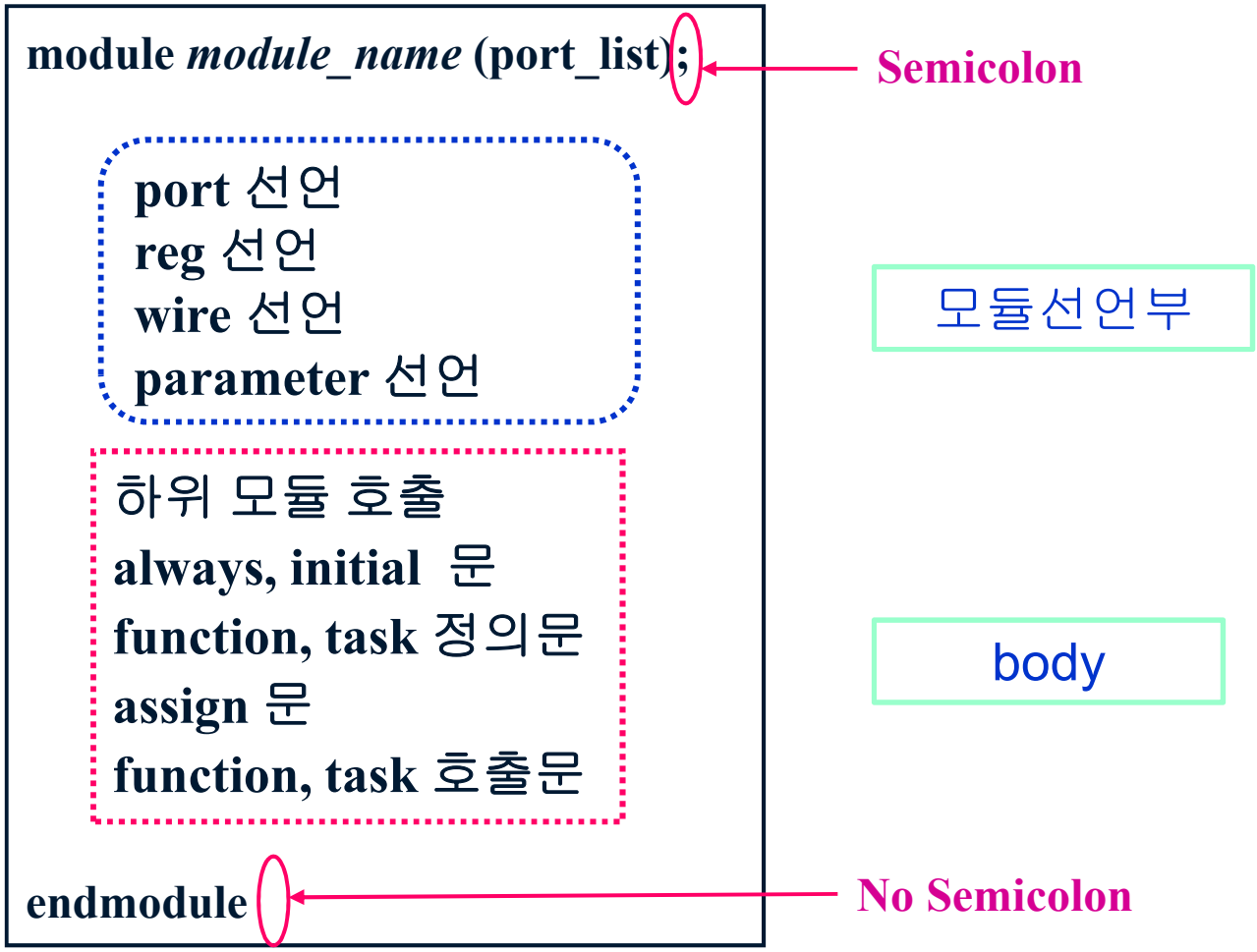
strength level	strength name	specification keyword		display mnemonic	
7	supply drive	supply0	supply1	Su0	Su1
6	strong drive	strong0	strong1	St0	St1
5	pull drive	pull0	pull1	Pu0	Pu1
4	large capacitive	large		La0	La1
3	weak drive	weak0	weak1	We0	We1
2	medium capacitive	medium		Me0	Me1
1	small capacitive	small		Sm0	Sm1
0	high impedance	highz0	highz1	HiZ0	HiZ1

Velilog Keyword

always	end	ifnone	notif0	rnmos	tranif0
and	endcase	initial	notif1	rpmos	tranif1
assign	endconfig†	instance†	or	rtran	tri
automatic†	endfunction	inout	output	rtranif0	tri0
begin	endgenerate†	input	parameter	rtranif1	tri1
buf	endmodule	integer	pmos	scalared	triand
bufif0	endprimitive	join	posedge	signed	trior
bufif1	endspecify	large	primitive	showcancelled†	trireg
case	endtable	liblist†	pull0	small	unsigned
casex	endtask	localparam†	pull1	specify	use†
casez	event	macromodule	pulldown	specparam	vectored
cell†	for	medium	pullup	strength	wait
cmos	force	module	pulsestyle_onevent†	strong0	wand
config†	forever	nand	pulsestyle_ondetect†	strong1	weak0
deassign	fork	negedge	rcmos	supply0	weak1
default	function	nmos	real	supply1	while
defparam	generate†	nor	realtime	table	wire
design†	genvar†	not	reg	task	wor
disable	highz0	noshowcancelled†	release	time	xnor
edge	highz1		**repeat**	tran	xor
else	if				

Verilog HDL의 모듈

- 모듈(module) – 설계의 기본단위, design entity



모듈 정의 (definition)

```
// 암시적 내부 연결( Implicit Internal Connection )
module module_name (port_name, port_name, ... );
module_items
endmodule
```

```
// 명시적 내부 연결 (Explicit Internal Connection)
module module_name (.port_name (signal_name ),
.port_name (signal_name ), ... );
module_items
endmodule
```

모듈 정의 (definition)

- module items
  - module_port_declarations
  - data_type_declarations
  - module_instances
  - primitive_instances
  - procedural_blocks
  - continuous_assignments
  - task_definitions
  - function_definitions

모듈 정의 (definition) – 포트 연결 예

```
// code 1.4.1-1:ex1.v 모듈 정의의 암시적 연결, 등가 비교기
module ex1 ( a1, b1, out1);
input [3:0] a1, b1;
output out1;
   assign out1 = ( a1 >= b1 ); // continuous assignment
endmodule
```

```
// code 1.4.1-2: ex2.v 모듈 정의에 포트선언 목록을 포함한 암시적 연결
// 2 input MUX with 2 bit widths
module ex2 (input wire [1:0] i0, i1, input wire sel, output wire [1:0] out2);
wire t0, t1;
   assign out2 = {t1, t0}; // concatenation
   assign t1 = sel ? i1[1] : i0[1];
   assign t0 = sel ? i1[0] : i0[0];
endmodule
```

모듈 정의 (definition) – 포트 연결 예

```
// 예 1.4.1 모듈 정의의 명시적 연결 예
module exp_port1 (.a(n1), .b(n2) );
   // n1, n2는 모듈 내부에서 선언
   // a, b는 포트연결로 정의
module exp_port2 (.a({b,c}), d, .f(g, h[2]));
   // b, c, d, g, h[2]는 모듈 내부에서 선언
   // a, d, f는 포트연결로 정의
module exp_port3 ({a, b}, .c(d))
   // a, b, d는 모듈 내부에서 선언
   // {a, b}, c는 포트연결로 정의
```

포트 선언 (port declaration)

❑ 포트선언 형식

```
port_direction data_type signed [port_size] port_name, port_name, .. ;
```

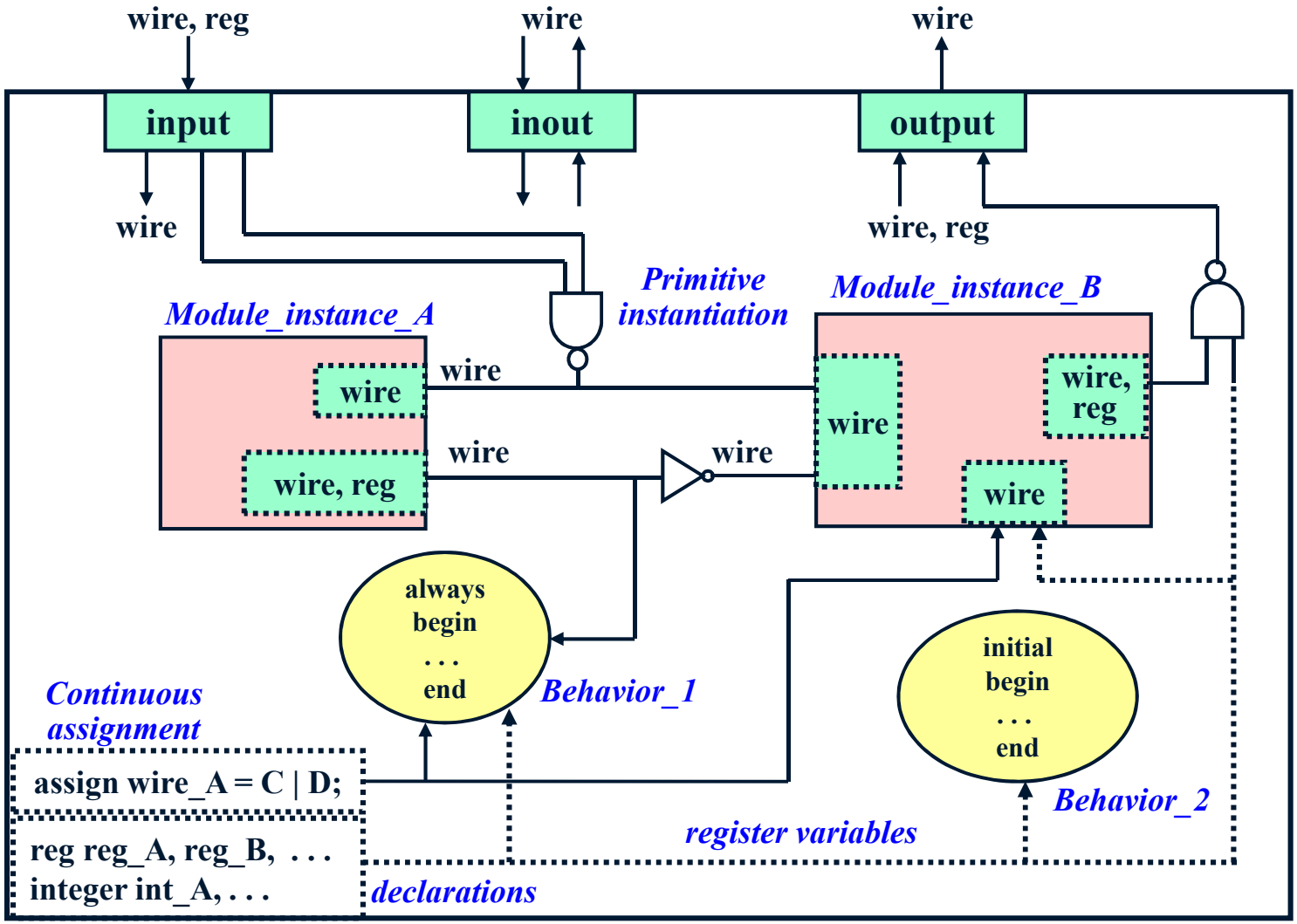
❑ 포트 방향

- ❖ input : 스칼라(scalar)나 벡터(vector)의 입력 포트 선언
- ❖ output : 스칼라(scalar)나 벡터(vector)의 출력 포트 선언
- ❖ inout : 스칼라(scalar)나 벡터(vector)의 양방향 포트 선언

포트 사용 예

```
// 예 1.4.2 포트 선언 예
input a1, a2, en;                    // 3개의 스칼라 1 비트 포트
input signed [7:0] a, b;             // 2개의 8비트 signed 값을 갖는 포트
output reg signed [16:0] res; // 데이터형과 signed 속성을 갖는 포트
output reg [11:0] cnt1;              // little endian 표기방식
inout [0:15] data_bus;               // big endian 표기방식
input [15:12] addr;                  // msb:lsb는 정수 값
parameter BW = 32;
input [BW-1:0] addr;                 // 상수 표현식 사용 가능
parameter SIZE = 4096;
input [log2(SIZE)-1:0] addr; // 상수 함수를 선언에서 호출가능
```

Verilog HDL의 모듈



Verilog 모델링 예

❑ 게이트 프리미티브를 이용한 모델링 예 (반가산기 회로)

Module Name　　Module Ports

```
module Half_Adder (a, b, sum, cout);
  input   a, b;
  output sum, cout;
  wire     cout_bar;

  xor (sum, a, b);
  nand (cout_bar, a, b);
  not (cout, cout_bar);
endmodule
```

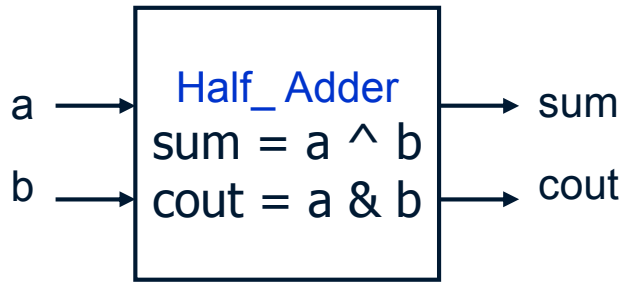
Declaration of port modes

Declaration of internal signals

Verilog keyword

Instantiation of Primitive gates

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



1.5 Data Type

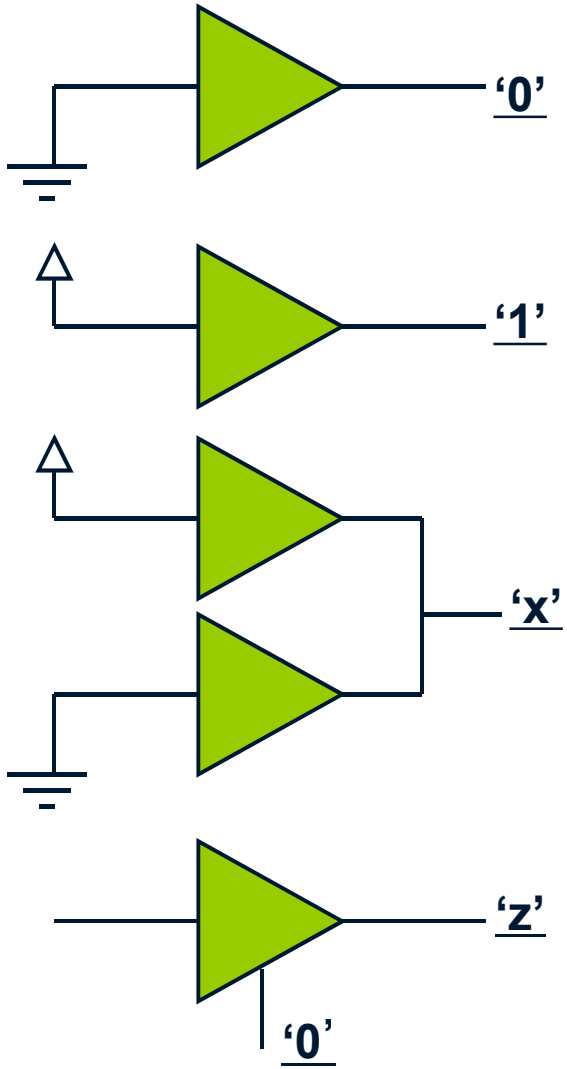
Verilog의 논리값

Verilog의 논리값 집합

논리값	의 미
0	logic zero, or false condition
1	logic one, or true condition
x	unknown logic value
z	high – impedance state

Verilog의 논리값

- Zero, low, false, logic low, ground, VSS
- One, high, true, logic high, power, VDD, VCC
- X, unknown : occurs at logical conflict which cannot be resolved
- HiZ, high impedence, tri-stated, disabled or disconnected driver



Verilog HDL의 자료형

- Net 자료형 : 소자간의 물리적인 연결을 추상화
  - wire, tri, wand, wor, triand, trior, supply0, supply1, tri0, tri1, trireg
  - Default 자료형 ; 1비트의 wire
- Variable 자료형 (Regiser 자료형): 절차형 할당문 사이의 값의 임시 저장
  - 프로그래밍 언어의 variable과 유사한 개념
  - reg, integer, real, time, realtime

net 자료형과 variable 자료형의 할당 모드

자료형	할당 모드				
	프리미티브 출력	연속 할당문	절차형 할당문	assign… deassign PCA	force… release PCA
Net	Yes	Yes	No	No	Yes
Variable	Comb (No) Seq (Yes)	No	Yes	Yes	Yes

*** PCA : Procedural Continuous Assignment**

variable 자료형

- variable 자료형 ; reg, integer, real, time, realtime

키워드	기능
reg	임의 비트 크기의 unsigned variable
integer	32 비트 signed variable
real	배정도 부동소수점 variable
time	64 비트 unsigned variable
realtime	배정도 부동소수점 variable

- 절차적 할당문(procedural assignment)의 실행에 의해 그 값이 바뀌며, 할당에서부터 다음 할당까지 값을 유지
- default 초기값
  - reg, time, integer 자료형 : x (unknown)
  - real, realtime 자료형 : 0.0
- variable이 음의 값을 할당 받는 경우,
  - signed reg, integer, real, realtime 자료형 : 부호를 유지
  - unsigned reg, time 자료형 : unsigned 값으로 취급

variable 자료형

- reg
  - 절차적 할당문에 의해 값을 받는 객체의 자료형
  - 할당 사이의 값을 유지
  - 하드웨어 레지스터를 모델링하기 위해 사용될 수 있음
    - edge-sensitive (플립플롭 등)와 level-sensitive (래치 등)의 저장소자들을 모델링할 수 있음
    - reg는 조합논리회로의 모델링에도 사용되므로, reg가 항상 하드웨어적인 저장소자를 의미하지는 않음

예 : reg 자료형 선언 예

```
reg a;              // a scalar reg
reg[3:0] v;         // a 4-bit vector reg made up of(from most to
                    // least significant) v[3], v[2], v[1], and v[0]
reg signed [3:0] signed_reg; // a 4-bit vector in range -8 to 7
reg [-1:4] b;               // a 6-bit vector reg
reg [4:0] x, y, z;          // declares three 5-bit regs
```

variable 자료형

```
module dff (clk, d, q);
  input  d ,clk;
  output q;
  reg    q;

  always @(posedge clk)
     q <= d;

endmodule
```

D 플립플롭

```
module mux21_if(a, b, sel, out);
   input  [1:0] a, b;
   input        sel;
   output [1:0] out;
   reg    [1:0] out;

   always @(a or b or sel)
      if(sel == 1'b0)
         out = a;
      else
         out = b;
endmodule
```

2 : 1 MUX

variable 자료형

- integer 자료형
  - 정수형 값을 취급하며, 절차적 할당문에 의해 값이 변경됨
  - signed reg로 취급되며, 연산 결과는 2의 보수가 됨
- time 자료형
  - 시뮬레이션 시간을 처리하거나 저장하기 위해 사용됨
  - 64비트의 reg와 동일하게 작용
  - unsigned 값이고 unsigned 연산이 이루어짐
- real, reatime 자료형
  - 실수형 값을 취급

예 : Variable 자료형 선언 예

```
integer a;          // integer value
time last_chng;     // time value
real float ;        // a variable to store real value
realtime rtime ;    // a variable to store time as a real value
```

net 자료형

❑ net 자료형

- ❖ 논리 게이트나 모듈 등의 하드웨어 요소들 사이의 물리적 연결을 나타내기 위해 사용
- ❖ 연속 할당문(continuous assignment), 게이트 프리미티브 등과 같은 구동자(driver)의 값에 의해 net의 값이 연속적으로 유지됨
  - ➢ 값을 저장하지 않음 (단, trireg net는 예외)
- ❖ 구동자가 연결되지 않으면, default 값인 high-impedance (z)가 됨
  - ➢ 단, trireg net는 이전에 구동된 값을 유지
- ❖ default 자료형은 1비트의 wire
- ❖ default 초기값은 z
  - ➢ trireg net는 default 초기값으로 x를 가짐

net 자료형

Verilog net 자료형

자료형 이름	의 미
wire	함축된 논리적 동작이나 기능을 갖지 않는 단순한 연결을 위한 **net**
tri	함축된 논리적 동작이나 기능을 갖지 않는 단순한 연결을 위한 **net**이며, 하드웨어에서 **3**상태**(tri－state)**가 되는 점이 **wire**와 다름
wand	다중 구동자를 갖는 **net**이며, **'wired-and'(**즉, **open collector logic)**의 하드웨어 구현을 모델링하기 위해 사용
wor	다중 구동자를 갖는 **net**이며, **'wired-or'(**즉, **emitter coupled logic)**의 하드웨어 구현을 모델링하기 위해 사용
triand	**wand**와 동일하게 다중 구동자를 갖는 **net**이며, 하드웨어에서 **3**상태**(tri-state)**를 갖는 점이 다름
trior	**wor**와 동일하게 다중 구동자를 갖는 **net**이며, 하드웨어에서 **3**상태**(tri-state)**를 갖는 점이 다름
supply0	회로접지**(circuit ground)**에 연결되는 **net**
supply1	전원**(power supply)**에 연결되는 **net**
tri0	저항성 **pulldown (resistive pulldown)**에 의해 접지로 연결되는 **net**
tri1	저항성 **pullup (resistive pullup)**에 의해 전원으로 연결되는 **net**
trireg	물리적인 **net**에 저장되는 전하를 모델링하는 **net**

net 자료형

- wire와 tri
  - 회로 구성요소들 사이의 연결에 사용
  - wire : 단일 게이트 또는 단일 연속 할당문에 의해 구동되는 net에 사용
  - tri : 3상태 net에 사용

wire, tri net의 진리표

wire/tri	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z

예 : net 자료형 선언 예

```
wire w1, w2;          // declares two wires, 생략 가능
wire [7:0] bus;       // a 8-bit bus
wire enable=1'b0;     // wire with initial value of 0
wand w3;              // a scalar net of type wand
tri [15:0] busa;      // a three-state 16-bit bus
```

net 자료형

- wired net
  - 다중 구동자를 갖는 설계를 지원하기 위해 사용

wand, triand net의 진리표

wand/ triand	0	1	x	z
0	0	0	0	0
1	0	1	x	1
x	0	x	x	x
z	0	1	x	z

wor, trior net의 진리표

wor/ trior	0	1	x	z
0	0	1	x	0
1	1	1	1	1
x	x	1	x	x
z	0	1	x	z

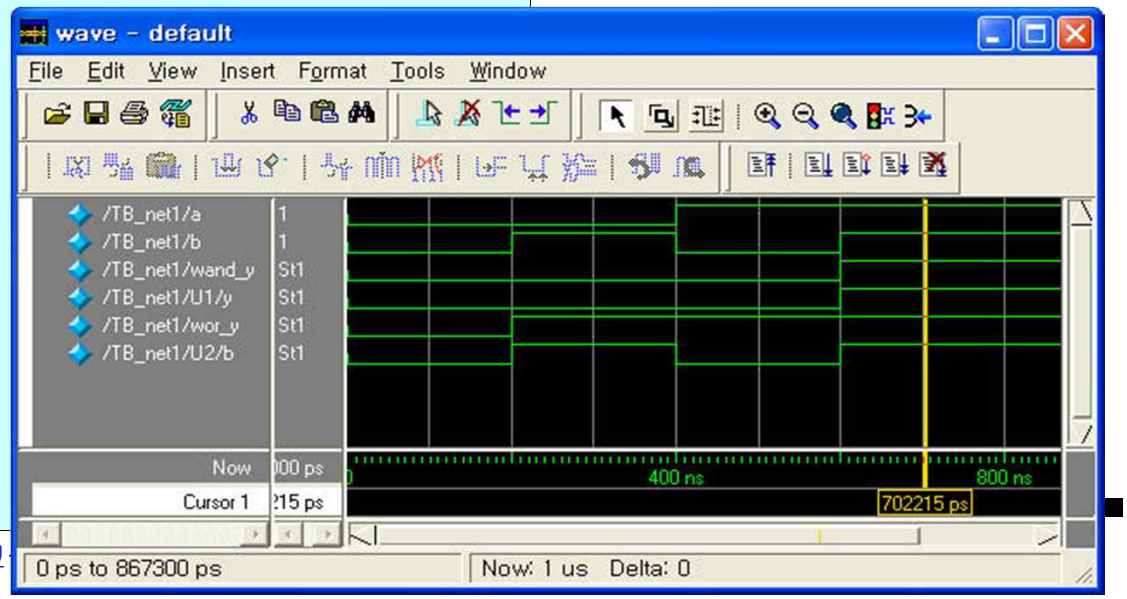
net 자료형

```
// code 1.5.2-1 : wand_test.v
module wand_test(a, b, y);
input a, b;
output y;
wand y;
   assign y = a;
   assign y = b;
endmodule
```

```
//code 1.5.2-2 : wor_test.v
module wor_test(a, b, y);
input a, b;
output y;
wor y;
   assign y = a;
   assign y = b;
endmodule
```

net 자료형

```
// code 1.5.2-3 : TB_net1.v
`timescale 1ns/1ps
module TB_net1;
wire wand_y, wor_y;
reg a, b;
   wand_test U1 (.a(a), .b(b), .y(wand_y) );
   wor_test    U2 (.a(a), .b(b), .y(wor_y) );
initial begin
   a=0; b=0; #200;
   a=0; b=1; #200;
   a=1; b=0; #200;
   a=1; b=1; #200;  end
endmodule
```



벡터

- 범위지정 **[msb:lsb]** 을 갖는 다중 비트의 net 또는 reg 자료형
- signed로 선언되거나 signed로 선언된 포트에 연결되는 경우를 제외하고는 unsigned로 취급
- 단일 할당문으로 값을 받을 수 있음

```
data_type [msb:lsb] identifier;
```

```
reg  [7:0]  rega;   // 8-bit reg
wire [15:0] d_out;  // 16-bit wire
```

배열

- 별도의 자료형이 없으며, reg 또는 wire 선언을 이용하여 선언
- 배열 전체 또는 일부분은 단일 할당문에 의해 값을 할당 받을 수 없으며, 또한 수식에 사용될 수 없음
  - 배열을 구성하는 element만 단일 할당문으로 값을 할당 받을 수 있음
- RAM, ROM, Register File 등의 메모리 모델링에 사용

2차원 배열

```
data_type identifier [Uaddr:Laddr][Uaddr2:Laddr2];
```

벡터의 2차원배열

```
data_type [msb:lsb] identifier [Uaddr:Laddr][Uaddr2:Laddr2];
```

배열

예 : 배열선언

```
reg [7:0] mema[0:255];   // a memory mema of 256 8-bit registers
reg arrayb[7:0][0:255]; // a 2-D array of 1-bit registers
wire w_array[7:0][5:0]; // an array of wires
integer inta[1:64];      // an array of 64 integer values
time chng_hist[1:1000]; // an array of 1000 time values
```

예 : 배열요소에 의한 할당

```
mema = 0;              // Illegal syntax- Attempt to write to entire array
arrayb[1] = 0;         // Illegal Syntax - Attempt to write to elements
                       // [1][0]..[1][255]
arrayb[1][12:31] = 0;     // Illegal Syntax - Attempt to write to
                          // elements [1][12]..[1][31]
mema[1] = 0;              // Assigns 0 to the second element of mema
arrayb[1][0] = 0;     // Assigns 0 to the bit referenced by indices [1][0]
inta[4] = 33559;      // Assign decimal number to integer in array
chng_hist[t_index] = $time; // Assign current simulation time to
                            // element addressed by integer index
```

배열

❑ 메모리

- ❖ reg형 요소를 갖는 1차원 배열
- ❖ 메모리 전체가 단일 할당문으로 값을 할당 받을 수 없음
  - ➢ 인덱스로 지정되는 워드 단위로만 값을 할당하거나 수식에 사용될 수 있음

예 : 메모리

```
reg [1:n] rega;    // An n-bit register
reg mema [1:n];    // A memory of n 1-bit registers
```

parameter

- variable 또는 net 범주에 속하지 않는 상수값
- 회로의 비트 크기 또는 지연값을 지정하기 위해 사용
- **defparam** 문 또는 모듈 인스턴스 문의 parameter overriding에 의해 값을 변경시킬 수 있음
- 자료형과 범위지정을 가질 수 있음
  - 범위가 지정되지 않은 경우, 상수 값에 적합한 크기의 비트 폭을 default로 가짐

예 : parameter 선언

```
parameter msb = 7;                  // defines msb as a constant value 7
parameter e = 25, f = 9;            // defines two constant numbers
parameter r = 5.7;                  // declares r as a real parameter
parameter byte_size = 8, byte_mask = byte_size - 1;
parameter average_delay =(r + f) / 2;
parameter signed [3:0] mux_selector = 0;
parameter real r1 = 3.5e17;
parameter p1 = 13'h7e;
parameter [31:0] dec_const = 1'b1;  // value converted to 32 bits
parameter newconst = 3'h4;          // implied range of [2:0]
```

parameter

모듈 인스턴스의 parameter overriding

```
module modXnor (y_out, a, b);
  parameter  size=8, delay=15;
  output     [size-1:0] y_out;
  input      [size-1:0] a, b;
  wire       [size-1:0] #delay  y_out= a ~^ b; // bit-wise XNOR with delay
endmodule

module Param;
  wire  [7:0] y1_out;
  wire  [3:0] y2_out;
  reg   [7:0] b1, c1;
  reg   [3:0] b2, c2;
  modXnor            G1 (y1_out, b1, c1);   // use default parameters
  modXnor  #(4, 5)   G2 (y2_out, b2, c2);   // overrides default parameters
endmodule
```

primitive gate의 delay

```
// Primitive instantiation with 3 units of delay
nand #3 G1 (out_nd2, in0, in1);
```

1.6 연산자 (Operator)

Verilog의 연산자

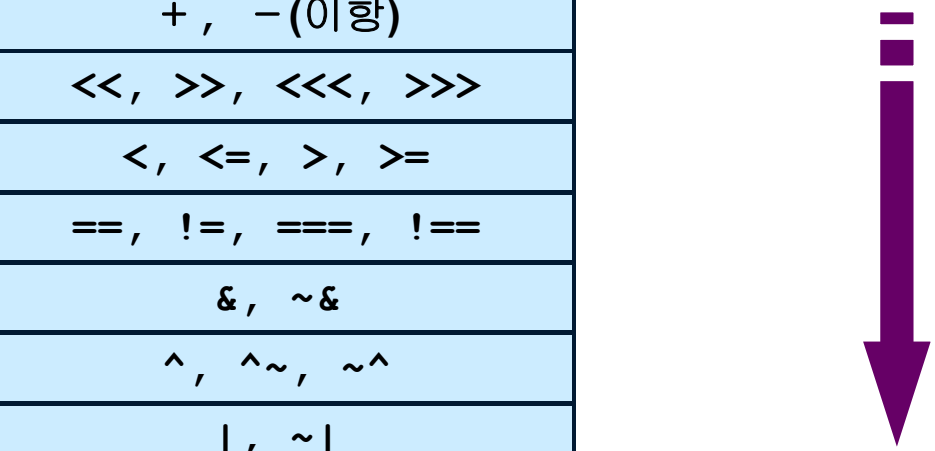
Verilog HDL의 연산자

연산자	기 능	연산자	기 능
{}, {{}}	결합, 반복	^	비트 단위 **exclusive or**
+, −, *, /, **	산술	^~ 또는 ~^	비트 단위 **xnor**
%	나머지	&	축약(**reduction**) **and**
>, >=, <, <=	관계	~&	축약 **nand**
!	논리 부정	|	축약 **or**
&&	논리 **and**	~|	축약 **nor**
||	논리 **or**	^	축약 **xor**
==	논리 등가	^~ 또는 ~^	축약 **xnor**
!=	논리 부등	<<	논리 왼쪽 시프트
===	**case** 등가	>>	논리 오른쪽 시프트
!==	**case** 부등	<<<	산술 왼쪽 시프트
~	비트 단위 부정	>>>	산술 오른쪽 시프트
&	비트 단위 **and**	? :	조건
|	비트 단위 **or**	or	**Event or**

Verilog의 연산자

Verilog 연산자의 우선순위

+, −, !, ~ (단항)	**Highest precedence**
**	
*, /, %	
+, −(이항)	
<<, >>, <<<, >>>	
<, <=, >, >=	
==, !=, ===, !==	
&, ~&	
^, ^~, ~^	
|, ~|	
&&	
||	
? : (conditional operator)	**Lowest precedence**



연산자

실수형 수식에 사용될 수 있는 연산자

연산자	기 능	연산자	기 능
+, −, *, /, **	산술	||	논리 **or**
+, −	부호	==	논리 등가
>, >=, <, <=	관계	!=	논리 부등
!	논리 부정	? :	조건
&&	논리 **and**	or	**Event or**

산술 연산자

- 피연산자의 비트에 x 나 z 가 포함된 경우, 전체 결과 값은 x
- 나누기와 나머지 연산자에서 두 번째 피연산자가 0인 경우, 결과값은 x
- 나머지 연산자의 결과 값은 첫번째 피연산자의 부호를 따름
- 거듭제곱 연산자에서 다음의 경우에는 결과 값이 정의되지 않음
  - 첫번째 피연산자가 0이고 두 번째 피연산자가 양수가 아닌 경우
  - 첫번째 피연산자가 음수이고 두 번째 피연산자가 정수 값이 아닌 경우

기 호	기 능
+	더하기
–	빼기
*	곱하기
/	나누기(몫)
%	나머지(modulo)
**	거듭제곱(power)

오퍼랜드 자료형	해 석
unsigned net	Unsigned
signed net	Signed, 2's complement
unsigned reg	Unsigned
signed reg	Signed, 2's complement
integer	Signed, 2's complement
time	Unsigned
real, realtime	Signed, floating – point

산술연산자 사용 예

```
-d10 / 5                  // (10의 2의 보수)/5 = (2^32-10)/5
5 / 0                     // 5/0 = x
(-7) % (+4) = -3          // 나머지, 왼쪽 오퍼랜드의 부호를 따른다.
(+7) % (-2) = +1
   // 정수, 레지스터 연산 예
integer intA;
reg [15:0] regA;
reg signed [15:0] regS;
intA = -4'd12;
regA = intA / 3;           // -4, intA는 integer, regA는 65532
regA = -4'd12;             // regA는 65524
intA = regA / 3;           // 21841
intA = -4'd12 / 3;         // 1431655761, -4d'12= 2^32-12
regA = -12 / 3;            // -4, -12는 integer 자료형, regA는 65532
regS = -12 / 3;            // -4, regS는 signed reg
regS = -4'sd12 / 3;        // 1, -4'sd12는 40이므로 4/3=1
```

관계 연산자

- 산술 연산자보다 낮은 우선 순위를 가짐
- 피연산자의 비트에 x 나 z가 포함된 경우, 결과 값은 1비트의 x
- 두 피연산자의 비트 수가 다른 경우에는, 비트 수가 작은 피연산자의 MSB 쪽에 0이 채워져 비트 수가 큰 피연산자에 맞추어진 후, 관계를 판단함
- 피연산자 중 하나가 실수형이면 다른 피연산자가 실수형으로 변환된 후, 비교됨

관계 연산자

관계 연산자 식	의 미
a < b	a가 b보다 작다
a > b	a가 b보다 크다
a <= b	a가 b보다 작거나 같다
a >= b	a가 b보다 크거나 같다

관계 연산자

예 : 관계 연산자 수식

```
// A = 9, B = 4
// D = 4'b1001, E = 4'b1100, F = 4'b1xxx

A <= B    // 결과 값은 거짓(0)
A > B     // 결과 값은 참(1)
E >= D    // 결과 값은 참(1)
E < F     // 결과 값은 x
```

```
① a < b - 1     // ①과 ②는 결과가 동일
② a <(b - 1)
③ b - (1 < a)   // ③과 ④는 결과가 다를 수 있음
④ b - 1 < a
```

등가 연산자

- 관계 연산자 보다 낮은 우선순위를 가짐
- 피연산자의 비트끼리 비교
- 두 피연산자의 비트 수가 다른 경우에는, 비트 수가 작은 피연산자의 MSB 쪽에 0이 채워져 비트 수가 큰 피연산자에 맞추어진 후, 등가를 판단함
- case equality와 case inequality 연산자(===, !==)는 대부분의 EDA 툴에서 논리합성이 지원되지 않으므로, 합성을 위한 RTL 수준의 모델링에는 사용하지 않는 것이 좋음

등가 연산자

관계 연산자 식	의 미
a === b	a와 b는 같다. (x와 z가 포함된 일치를 판단)
a !== b	a와 b는 같지 않다. (x와 z가 포함된 불일치를 판단)
a == b	a와 b는 같다. (결과가 x가 될 수 있음)
a != b	a와 b는 같지 않다. (결과가 x가 될 수 있음)

등가 연산자

예 : 등가 연산자

```
// A = 9, B = 4
// D = 4'b1001, E = 4'b1100
// F = 4'b1xxz, G = 4'b1xxz, H = 4'b1xxx

A === B     // 결과 값은 거짓(0)
D != E      // 결과 값은 참(1)
D == F      // 결과 값은 x
F === G     // 결과 값은 참(1)
F === H     // 결과 값은 거짓(0)
G !== H     // 결과 값은 참(1)
```

논리 연산자

❖ 참 또는 거짓의 판단이 모호한 경우에는 결과값은 x

논리 연산자 식	의 미
a && b	a와 b의 논리 AND
a || b	a와 b의 논리 OR
!a	a의 부정 (NOT a)

```
// A = 3, B = 0, C = 2'b0x,
// D = 2'b10인 경우에,
A && B     // 결과 값은 0
A || B     // 결과 값은 1
!A         // 결과 값은 0
!B         // 결과 값은 1
C && D     // 결과 값은 x
```

예 : 논리 연산자

```
// alpha = 237, beta=0인 경우에,
regA = alpha && beta;       // regA에는 0이 할당된다.
regB = alpha || beta;       // regB에는 1이 할당된다.

a < size-1 && b != c && index != last_one
a < (size-1) && (b != c) && (index != last_one)    // recommended

if(!reset)
```

비트 연산자

- 피연산자의 해당 비트들에 대한 연산을 수행
- 피연산자의 비트 수가 같지 않으면, 비트 수가 작은 피연산자의 MSB 위치에 0이 채워진 후, 연산됨

비트 and 연산자

&	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

비트 or 연산자

|	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

비트 부정 연산자

~	0
0	1
1	0
x	x
z	x

비트 xor 연산자

^	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

```
//D = 4'b1001, E = 4'b1101, F = 4'b10x1
~D          // 결과 값은 4'b0110
D & E       // 결과 값은 4'b1001
D | E       // 결과 값은 4'b1101
D ^ E       // 결과 값은 4'b0100
D ~^ E      // 결과 값은 4'b1011
D & F       // 결과 값은 4'b10x1
```

축약(reduction) 연산자

❖ 단항 연산자, 피연산자의 단위 비트들에 적용되어 단일 비트의 결과값 생성

축약 and

&	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

축약 nand

~&	0	1	x	z
0	1	1	1	1
1	1	0	x	x
x	1	x	x	x
z	1	x	x	x

축약 xnor

~^	0	1	x	z
0	1	0	x	x
1	0	1	x	x
x	x	x	x	x
z	x	x	x	x

연산자 / 피연산자	연산 결과						설 명
	&	~&	|	~|	^	~^	
4'b0000	0	1	0	1	0	1	모든 비트가 0인 경우
4'b1111	1	0	1	0	0	1	모든 비트가 1인 경우
4'b0110	0	1	1	0	0	1	1의 개수가 짝수인 경우
4'b1000	0	1	1	0	1	0	1의 개수가 홀수인 경우

```
reg[7:0] cnt;
assign parity = ^cnt;
assign parity = cnt[7]^cnt[6]^cnt[5]^cnt[4]^cnt[3]^cnt[2]^cnt[1]^cnt[0];
```

시프트 연산자

- 논리 시프트 연산자 (<<, >>)
  - << : 우측 피연산자 값만큼 좌측으로 시프트 후, 비어 있는 비트에 0을 채움
  - >> : 우측 피연산자 값만큼 우측으로 시프트 후, 비어 있는 비트에 0을 채움
- 산술 시프트 연산자 (>>>, <<<)
  - <<< : 우측 피연산자 값만큼 좌측으로 시프트 후, 비어 있는 비트에 0을 채움
  - >>> : 우측 피연산자 값만큼 우측으로 시프트 후, 비어 있는 비트에 좌측 피연산자의 MSB를 채움
- 우측 피연산자
  - x 또는 z가 포함된 경우, 시프트 연산의 결과 값은 x
  - 항상 unsigned 수

시프트 연산자

```
// A = 4'b1100
B = A >> 1   // 오른쪽으로 1비트 시프트, 결과 값은 B=4'b0110
C = A << 1   // 왼쪽으로 1비트 시프트, 결과 값은 B=4'b1000
D = A << 2   // 왼쪽으로 2비트 시프트, 결과 값은 B=4'b0000
```

```
module shift;
  reg [3:0] start, result;
  initial begin
     start = 1;
     result =(start << 2);     // 결과 값은 0100
  end
endmodule
```

```
module ashift;
  reg signed [3:0] start, result;
  initial begin
        start = 4'b1000;
        result =(start >>> 2);    // 결과 값은 1110
  end
endmodule
```

조건 연산자

conditional_expr ::= expr1 ? expr2 : expr3

- expr1이 참(1, 즉 0, x 또는 z가 아닌 값)으로 평가되면 expr2의 값이 좌변 변수에 할당
- expr1이 x 또는 z이면, expr2와 expr3을 함께 평가하여 비트 단위로 비교된 값이 좌변의 변수에 할당
  - expr3이 real 형 값이 아니면 결과 값은 비트 단위로 비교되어 결정되며, real 형 값인 경우에는 결과 값은 0이 됨

조건에 애매성이 존재하는 경우의 조건 연산자의 결과 값 결정

? :	0	1	x	z
0	0	x	x	x
1	x	1	x	x
x	x	x	x	x
z	x	x	x	x

조건 연산자를 이용한 3상태 버퍼

```
wire [15:0] busa = drive_busa ? data : 16'bz;
```

결합 연산자

- 중괄호 { }에 의해 묶인 두 개 이상의 표현이 갖는 비트들을 결합
  - 결합되는 피연산자들은 각각의 크기를 결정할 수 있어야 결합이 가능
  - unsized 상수는 결합 연산자로 결합시킬 수 없음
- 대입문의 좌측 또는 우측에 사용 가능
- 비트 폭이 일치하지 않는 변수의 연산이나 대입이 허용됨
  - 우변의 비트 폭이 작은 경우, 우변의 MSB에 0을 붙여 연산 됨
  - 좌변의 비트 폭이 우변 보다 작을 경우, MSB는 누락되어 저장

```
{a, b[3:0], w, 3'b101}
  //결합 연산자의 결과
{a, b[3], b[2], b[1], b[0], w, 1'b1, 1'b0, 1'b1}
```

```
wire [15:0] addr_bus;
assign addr_bus = {addr_hi, addr_lo};
// addr_hi, addr_lo는 8 비트 신호
```

```
wire [3:0] a, b, sum;
wire carry;
assign {carry, sum} = a + b;   // 4비트 데이터의 덧셈은 5비트 결과
// 좌변이 5 비트이므로, 우변의 a+b는 MSB에 0을 붙인 5비트로 연산 됨
```

반복 연산자

❖ {a{b}}의 형태로 표현하여 b를 a회 반복

➢ 반복 횟수 a는 0, x, z가 아닌 상수이어야 함

```
{4{w}}                                  // {w, w, w, w}와 동일한 표현.
a[31:0] = {1'b1, {0{1'b0}} };           // 우변이 {1'b1}가 되므로 잘못된 표현임.
a[31:0] = {1'b1, {1'bz{1'b0}} };        // 우변이 {1'bz}가 되므로 잘못된 표현임.
a[31:0] = {1'b1, {1'bx{1'b0}} };        // 우변이 {1'bx}가 되므로 잘못된 표현임.
```

```
result = {func(w), func(w), func(w), func(w)}; //반복함수호출
// 위 문자은 다음과 같다.
y=func(w);
result = {y,y,y,y};
// 혹은, 다음과도 같다.
result = {4{func(w)}};

{b, {3{a, b}}}                          // {b, a, b, a, b, a, b}와 동일함.
```