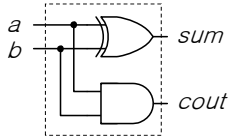
Chapter 2

Verilog를 사용한 논리 설계의 기초

간단한 조합회로의 Verilog 모델 – 구조적 모델

- Verilog example – structural model
  - half adder schematic



Verilog description

module이름 입출력 port list

```
module halfadder(sum, cout, a, b);
  input a, b;
  output cout, sum;

  xor (sum, a, b);
  and (cout, a, b);
endmodule
```

port list의 신호 순서가 의미 있음

port의 입출력 방향 선언
(port list와 선언순서와 무관)

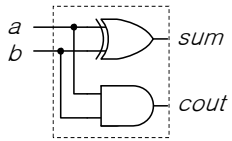
게이트 primitives
(출력, 입력, ...)

ANSI-C style module 선언 (Verilog-2001)

```
module halfadder(output sum, cout, input a, b);
  xor (sum, a, b);
  and (cout, a, b);
endmodule
```

```
module halfadder(
  output sum, cout,
  input a, b
);
  xor (sum, a, b);
  and (cout, a, b);
endmodule
```

port list에 입출력 방향 선언을
함께 할 때에는 선언순서가 의미 있음



기본적인 언어 규칙

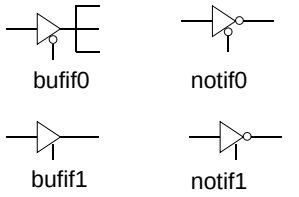
- 대소문자 구분(case-sensitive)
  - (예) Cout과 COUT은 같지 않음
- identifier는
  - alphabet, digit(0-9), underscore(_), $ 들로 구성됨
  - digit 또는 $로 시작하지 않음
  - 최대 1024 글자까지 가능
- 각 줄은 세미콜론(;)으로 끝남
  - 예외: endmodule ...
- comment
  - line comment: // *comment*
  - block comment; /* *comment* */

논리게이트와 Verilog primitives

- Verilog primitives

n-input 1-output	and or xor nand nor xnor
1-input n-ouput	buf not
	bufif0 notif0
	bufif1 notif1

3-states buffer/inverter



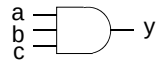
3-state 출력
- 0, 1, **Z**

high impedance (Z) 출력
- 출력이 끊어진 상태
- 여러 개의 출력이 함께 연결되는 bus 출력연결에 주로 사용

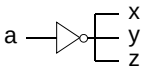
Instantiation of a gate

primitive는 output port가 앞에 위치함

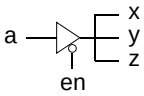
nand (y, a, b, c);
output input



not (x, y, z, a);
output input

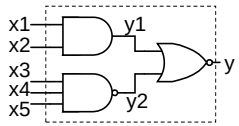


bufif0 (x, y, z, a, en);
output input enable



Connection by wires

- net 자료형
  - 두 gate 또는 module 간의 연결선
  - (종류) wire, wand, wor, tri, triand, trior, trireg
- wire
  - net로 주로 wire 형을 사용함
    wire y1, y2;
  - module의 port는 기본적으로 wire이다. (선언 불필요)

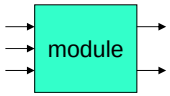


AOI(and-or-invert)

```
module AOI(x1, x2, x3, x4, x5, y);
   input  x1, x2, x3, x4, x5;
   output y;
   wire   y1, y2;

   nor (y, y1, y2);
   and (y1, x1, x2);
   nand (y2, x3, x4, x5);
endmodule
```


Module Ports

- Port
  - module 외부와의 인터페이스
- Port의 mode 선언
  - mode 종류: input, output, inout (양방향)
  - port의 mode를 선언하는 순서는 port 리스트의 순서와 같을 필요는 없다.



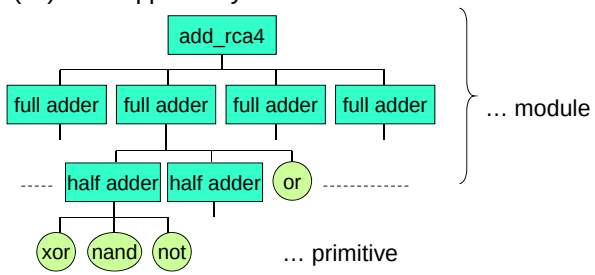

```
module AOI(x1, x2, x3, x4, x5, y);
   output y;
   input  x1, x2, x3, x4, x5;

   . . .
endmodule
```

```
module AOI(x1, x2, x3, x4, x5, y);
   input  x1, x2, x3, x4, x5;
   output y;

   . . .
endmodule
```

두 description은 같다.

Design hierarchy와 module instance

- Design hierarchy
  - (ex) 4-bit ripple carry adder

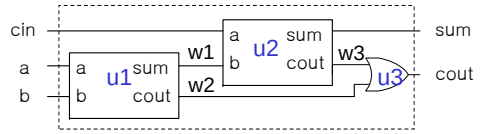


- Design 방법

  (1) top-down (2) bottom-up (3) 혼합방법
- nested module
  - 다른 module의 instance를 포함한 module

Nested modules

- 예: 2개의 half-adder를 사용한 full-adder 구현



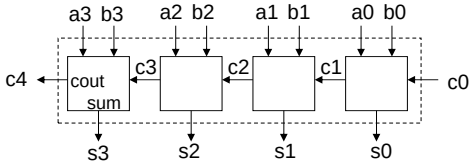

```
module fulladder(sum, cout, a, b, cin);
  output  sum, cout;
  input   a, b, cin;
  wire    w1, w2, w3;

  halfadder  u1 (w1, w2, a, b);
  halfadder  u2 (sum, w3, cin, w1);
  or  u3 (cout, w2, w3);
endmodule
```

module instance name (필수사항)

primitive instance name (선택사항)

Nested modules (cont')

- 예: 4 bit ripple carry adder(RCA)




```
module add_rca4 (c4, s3, s2, s1, s0, a3, a2, a1, a0, b3, b2, b1, b0, c0);
  output  c4, s3, s2, s1, s0;
  input   a3, a2, a1, a0, b3, b2, b1, b0, c0;
  wire    c3, c2, c1,

  fulladder  u1 (s0, c1, a0, b0, c0);
  fulladder  u2 (s1, c2, a1, b1, c1);
  fulladder  u3 (s2, c3, a2, b2, c2);
  fulladder  u4 (s3, c4, a3, b3, c3);
endmodule
```


Vector

- Vector

  wire [3:0] s; ... s는 4-bit vector (3: MSB index, 0: LSB index)
- Part select of a vector

  s[2] → 1

  s[2:1] → 2 (10_2)

  s[2:0] → 4 (100_2)

	3	2	1	0
s	1	1	0	0

```
module add_rca4 (c4, s3, s2, s1, s0, a3, a2, a1, a0, b3, b2, b1, b0, c0);
  output  c4, s3, s2, s1, s0;
  input   a3, a2, a1, a0, b3, b2, b1, b0, c0;
  wire    c3, c2, c1,
```

vector를 사용

```
module add_rca4 (c4, s, a, b, c0);
  output  c4;
  output [3:0]  s;
  input [3:0]  a, b;
  input  c0;
  wire [3:1] c;
```

4비트 ripple carry 가산기 – vector 사용

```
module add_rca4(c4, s, a, b, c0);
  input  [3:0] a, b;          // 2개의 4비트 입력
  input  c0                   // 캐리 입력
  output [3:0] s;             // 4비트 출력
  output c4;                  // 캐리 출력
  wire   [3:1] c;             // 중간 캐리

  fulladder u1 (s[0],c[1],a[0],b[0],c0);
  fulladder u2 (s[1],c[2],a[1],b[1],c[1]);
  fulladder u3 (s[2],c[3],a[2],b[2],c[2]);
  fulladder u4 (s[3],c4,a[3],b[3],c[3]);
endmodule
```


Structural Connectivity

- port의 formal name과 actual name
  - formal name: module을 정의할 때에 사용한 port name
  - actual name: module instance의 port에 연결되는 signal name
- module instance의 port 이름 연관 방법
  1. 정의된 formal name 순서대로 actual name을 기술함

     (예) halfadder u1 (w1, w2, a, b); ... actual name (instance)

     module halfadder(sum, cout, a, b); ... formal name (선언문)

     . . .
  2. 두 이름의 연관 관계를 직접 기술함 → port가 많을 때 유용함

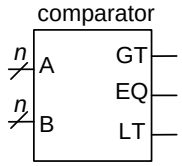
     .*formal name* (*actual name*)

     (예) halfadder u2 (.a(a), .b(b), .cout(w2), .sum(w1));

     ... 순서는 상관없음

Structural modeling 예 - 비교기

- n-비트 비교기



A, B	EQ	LT	GT
A = B	1	0	0
A < B	0	1	0
A > B	0	0	1

- 1-비트 비교기

A	B	EQ	LT	GT
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

EQ = A' B' + A B
LT = A' B
GT = A B'

1비트 비교기의 Verilog 모델

```
module comparator1(eq, lt, gt, a, b);
  input  a, b;                // 입력 포트 선언
  output eq, lt, gt;          // 출력 포트 선언
  wire   nota, notb, t1, t2;

  not u1 (nota, a);           // nota는 a'
  not u2 (notb, b);           // notb는 b'
  and u3 (lt, nota, b);       // lt = a'b
  and u4 (gt, a, notb);       // gt = ab'
  and u5 (t1, nota, notb);    // a'b'
  and u6 (t2, a, b);          // ab
  or u7 (eq, t1, t2);         // eq = a'b' + ab
endmodule
```

복잡함

EQ = A' B' + A B, LT = A' B, GT = A B'

작은 비교기를 사용한 큰 비교기 설계

- m-비트 비교기 + n-비트 비교기 → (m+n)비트 비교기

EQ = eq1 · eq0
LT = lt1 + eq1 · lt0
GT = gt1 + eq1 · gt0

작은 비교기 출력: eq0, lt0, gt0와 eq1, lt1, gt1
큰 비교기의 출력: EQ, LT, GT

```
module combine(eq, lt, gt, eq0, lt0, gt0, eq1, lt1, gt1);
   input  eq0, lt0, gt0, eq1, lt1, gt1;
   output eq, lt, gt;
   wire   t1, t2;

   and (eq, eq1, eq0);     // eq = eq1.eq0
   and (t1, eq1, lt0);
   or (lt, lt1, t1);       // lt = lt1 + eq1.lt0
   and (t2, eq1, gt0);
   or (gt, gt1, t2);       // gt = gt1 + eq1.gt0
endmodule
```


2-bit comparator와 4-bit comparator :

```
module comparator2(eq, lt, gt, A, B);
   output  gt, lt, eq;
   input   [1:0] A, B;
   wire    w1, w0, gt1, lt1, eq1, gt2, lt2, eq2;

   comparator1 u1 (eq1, lt1, gt1, A[1], B[1]);
   comparator1 u2 (eq0, lt0, gt0, A[0], B[0]);
   combine u3 (eq, lt, gt, eq0, lt0, gt0, eq1, lt1, gt1);
endmodule
```

```
module comparator4(eq, lt, gt, A, B);
   output  gt, lt, eq;
   input   [3:0] A, B;
   wire    w1, w0, gt1, lt1, eq1, gt2, lt2, eq2;

   comparator2 u1 (eq1, lt1, gt1, A[3:2], B[3:2]);
   comparator2 u2 (eq0, lt0, gt0, A[1:0], B[1:0]);
   combine u3 (eq, lt, gt, eq0, lt0, gt0, eq1, lt1, gt1);
endmodule
```


Four-value Logic

- Four value logic system
  - 0 : de-assertion, False
  - 1 : assertion, True
  - x (don't care) : 값을 알 수 없음(ambiguous)
  - z (high impedance) : 값이 공급(drive)되지 않음, 즉 끊어진 상태
- Four value에 대한 논리 연산

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

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	0	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

Four-value Logic (계속)

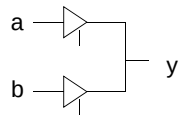
- Four value 논리 연산의 성질
  - z 입력은 x 입력과 같이 취급

AND x • 0 = 0 x • 1 = x
OR: x + 0 = x x + 1 = 1
XOR: x ⊕ 0 = x x ⊕ 1 = x
not: x' = x
buffer: (x) = x

	buf	not
0	0	1
1	1	0
x	x	x
z	x	x

- Resolution of Multiple drivers

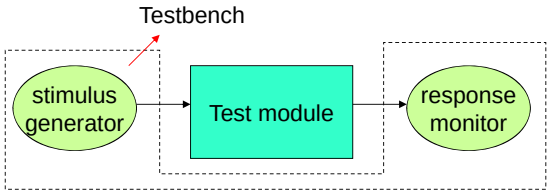
	0	1	x	z
0	0	x	x	0
1	x	1	x	1
x	x	x	x	x
z	0	1	x	z



설계 검증

- Verification method
  - logic simulation: 회로에 stimulus pattern을 공급하고 동작을 확인
  - formal verification: 수학적인 증명

- Testbench
  - test circuit에 stimulus pattern을 공급하고 결과를 출력하는 Verilog module



- Test 시 주의 사항
  - 큰 회로의 경우에는 logic simulation을 사용하여 완전한 검증을 하는 것이 어려움 ← 너무 많은 경우의 수
  - test되지 않은 경우 때문에 회로의 잘못을 찾지 못할 수 있음.

→ 체계적이며 정교한 test plan을 세울 필요가 있음

- Testbench의 작성
  - Verilog 문장으로 기술
    (합성도구에서 지원하지 않는 경우가 있음)
  - Waveform editor를 이용하여 stimulus pattern 편집
    Simulation결과도 waveform으로 확인

Verilog로 작성한 Testbench

- 예: halfadder module에 대한 testbench

```
module Test_Add_half();
  wire  sum, cout;
  reg   a, b;

  halfadder u1 (sum, cout, a, b);

  initial begin
    #100
    $finish;
  end

  initial begin
    #10  a = 0;  b = 0;    // a b = 0 0
    #10  b = 1;            // a b = 0 1
    #10  a = 1;            // a b = 1 1
    #10  b = 0;            // a b = 1 0
  end
endmodule
```

wire는 physical wire를 **표현**
reg는 값을 저장하는 자료형(변수)

initial begin … end
single pass behavior를 기술함

#100
100 time unit의 delay를 나타냄

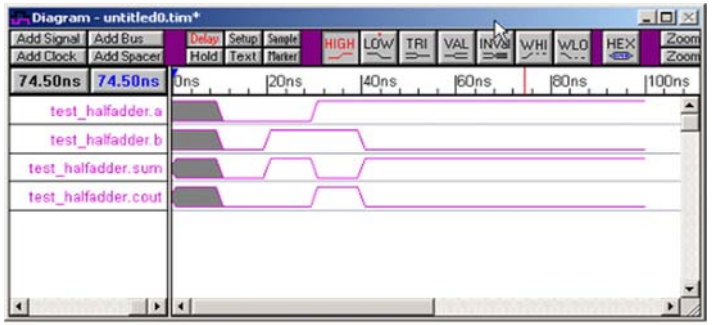
$finish;
simulation 종료

stimulus pattern

두 initial 문장은 time 0에서
동시에 simulation을 시작함

시뮬레이션 Waveform

- waveform
  - 대부분의 graphic 기반의 simulation 도구는 testbench의 signal들에 대한 waveform을 보여줌



  - $monitor 와 같은 함수를 사용하여 signal 값을 text로 출력할 수 있음

Propagation Delay

- propagation delay
  - 실제 회로에서의 timing verification은 propagation delay를 고려해야 함
  - simulation을 할 때에는 빠른 검증을 위해서 0 delay 또는 unit delay를 가정하고 한다.
- propagation delay의 Verilog modeling

```
module halfadder (sum, cout, a, b);
  input  a, b;
  output c_out, sum;

  xor  #2 u1 (sum, a, b);
  and  #1 u2 (cout, a, b);
endmodule
```

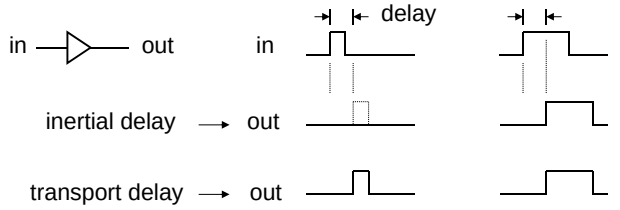
#2 … 2 time unit delay

XOR gate의 propagation delay는 2 time unit임

  - 실제의 timing 특성은 synthesis 도구 내에 포함되어 있음
  - 합성도구에서 사용자를 위해서 지원되지 않을 수 있음

Delay model

- Inertial Delay (관성 지연)
  - gate의 delay보다 작은 pulse는 무시됨
  - gate의 지연에 적용되는 모델
    - 출력은 입력의 변화에 즉시 변화되지 않는 특성을 모델링
- Transport Delay (전달 지연)
  - 입력의 변화가 그대로 출력의 변화로 전달됨
  - wire를 통한 signal의 전달을 모델링



Event-Driven Simulation

- Event
  - signal 값의 변화를 event라고 함
- Event-driven simulation
  - simulation은 event가 발생할 때 마다 수행함
  - 모든 gate와 behavior는 concurrent 동작한다
- Event scheduling
  - 입력 event 발생하면 회로의 동작을 simulation하여 출력 값 산출
    → 출력 값이 변하면 출력과 연결된 다른 회로의 입력에 대해서 회로의 지연시간 후에 event scheduling
    (지연시간이 0인 경우에는 Δ 시간 후)

Truth Table model

- 진리표 모델
  - 조합회로:진리표로 동작을 기술
    입력 : 출력
  - 순차회로:상태전이표로 동작을 기술
    입력 : 현재상태 : 다음상태
- User-defined primitive

```
primitive udp_or (y, x1, x2);
  output y;
  input  x1, x2;
  table  // x1  x2 : y
     0 0 : 0;
     0 1 : 1;
     1 0 : 1;
     1 1 : 1;
  endtable
endprimitive
```

(주의) Synthesis tool 에서는 사용할 수 없음

UDP는 single 1-bit output port만을 갖는다.

■ x 입력을 고려한 UDP 정의

```
primitive udp_or (y, x1, x2);
   output y;
   input  x1, x2;
   table  // x1  x2 : y
      0 0 : 0;
      0 1 : 1;
      1 0 : 1;
      1 1 : 1;
      1 x : 1;
      x 1 : 1;
   endtable
endprimitive
```

정의되지 않은 입력에 대해서
출력은 x가 됨

입력값 z는 입력값 x로 취급됨

■ ? 표기법을 사용한 UDP 정의

```
primitive udp_or (y, x1, x2);
   output y;
   input  x1, x2;
   table  // x1  x2 : y
      0 0 : 0;
      ? 1 : 1;
      1 ? : 1;
   endtable
endprimitive
```

? 는 0, 1, x입력을 모두 포함함
즉, don't care 조건

(cf) x는 unknown으로 0, 1과 다름