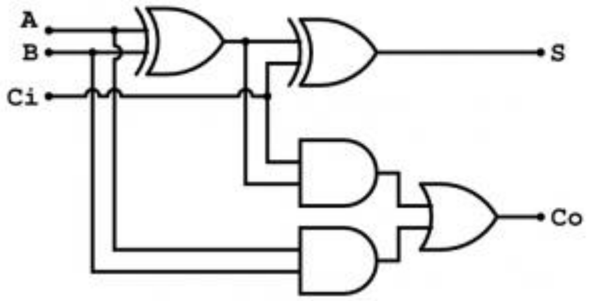
Verilog HDL

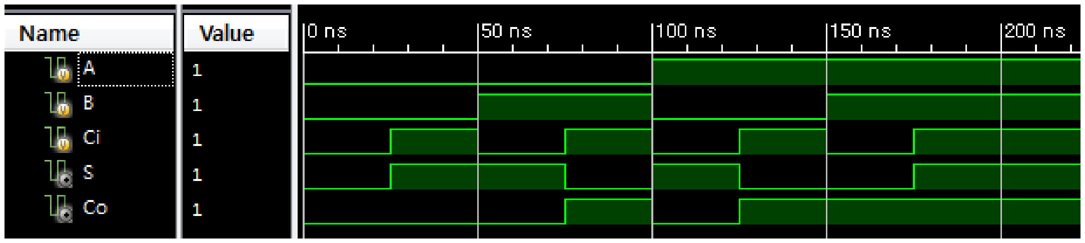
Lecture 01

HDL?

- HDL – Hardware Description Language

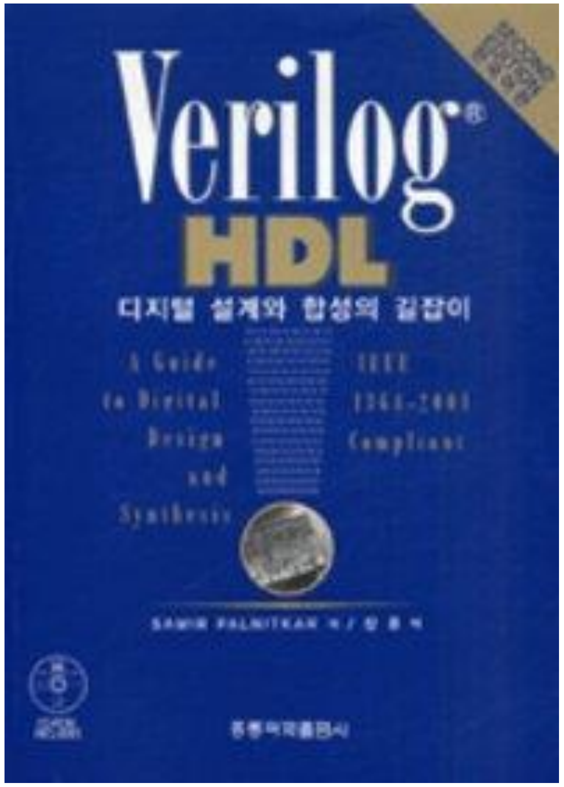
```
//Input
input A;
input B;
input Ci;

//Output
output S;
output Co;

//wires
wire w1, w2, w3;

//sum
xor( w1, A, B );
xor( S, w1, Ci );

//carry-out
and( w2, Ci, w1 );
and( w3, A, B );
or( Co, w2, w3 );
```





Name	Value
A	1
B	1
Ci	1
S	1
Co	1

Verilog HDL Reference

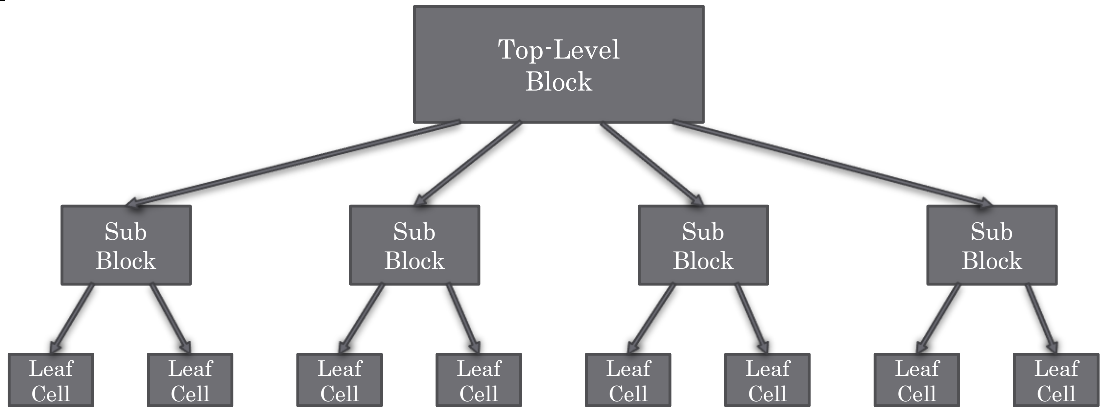

Verilog®
HDL
디지털 설계와 합성의 길잡이
A Guide to Digital Design and Synthesis
IEEE 1364-2001 Compliant
SAMIR PALNITKAR 저 / 장 훈 역

Verilog HDL

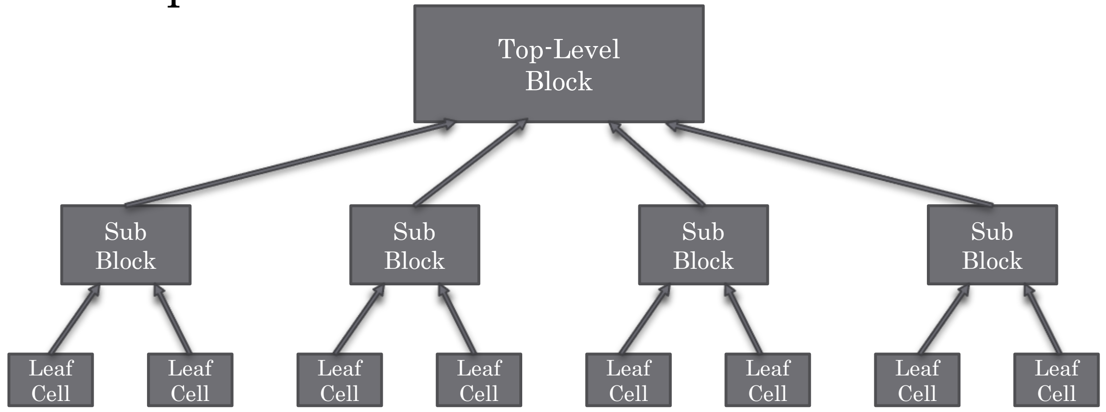
- RTL – Register Transfer Level
- Use
  - FPGA
  - ASIC
  - Simulation
- Design Methodologies
  - Top-Down
  - Bottom-Up
- Modeling
  - Gate-Level Modeling
  - Dataflow Modeling
  - Behavioral Modeling

Design Methodologies

- Top-Down

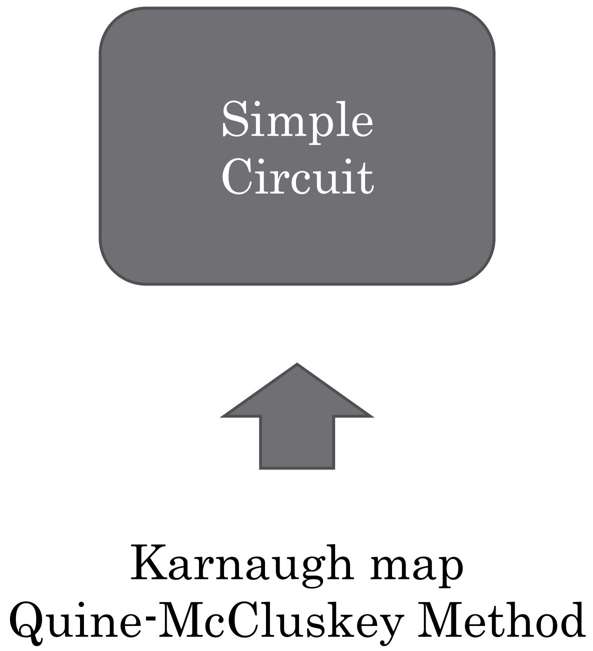


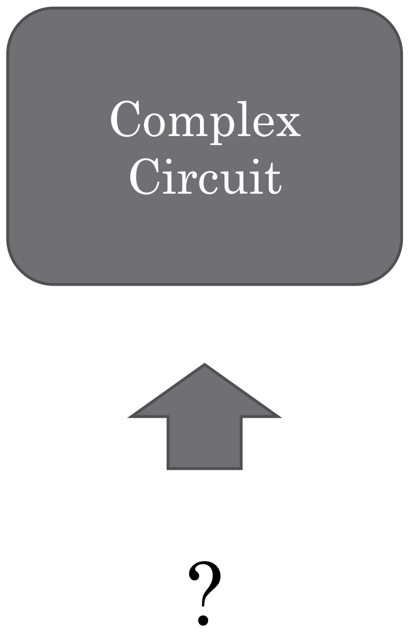
- Bottom-Up



Why?

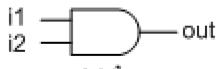
- How to design “Digital Circuit”





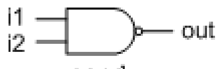
Basic Circuit

- Logic Gates



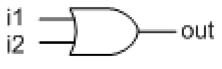
and

		i1			
and		0	1	x	z
	0	0	0	0	0
i2	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x



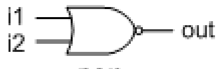
nand

		i1			
nand		0	1	x	z
	0	1	1	1	1
i2	1	1	0	x	x
	x	1	x	x	x
	z	1	x	x	x



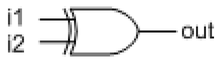
or

		i1			
or		0	1	x	z
	0	0	1	x	x
i2	1	1	1	1	1
	x	x	x	x	x
	z	x	x	x	x



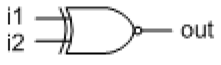
nor

		i1			
nor		0	1	x	z
	0	1	0	x	x
i2	1	0	0	0	0
	x	x	0	x	x
	z	x	0	x	x



xor

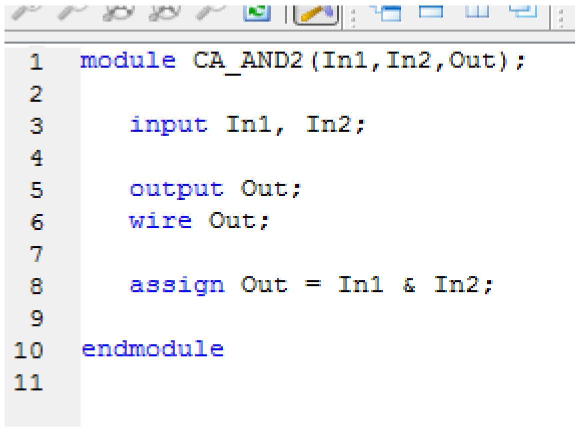
		i1			
xor		0	1	x	z
	0	0	1	x	x
i2	1	1	0	x	x
	x	x	x	x	x
	z	x	x	x	x



xnor

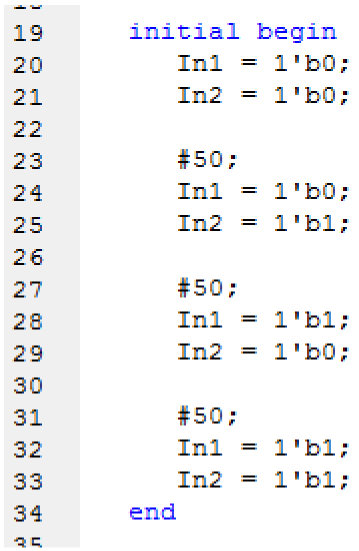
		i1			
xnor		0	1	x	z
	0	1	0	x	x
i2	1	0	1	x	x
	x	x	x	x	x
	z	x	x	x	x

AND Gate



```
module CA_AND2(In1,In2,Out);

   input In1, In2;

   output Out;
   wire Out;

   assign Out = In1 & In2;

endmodule
```

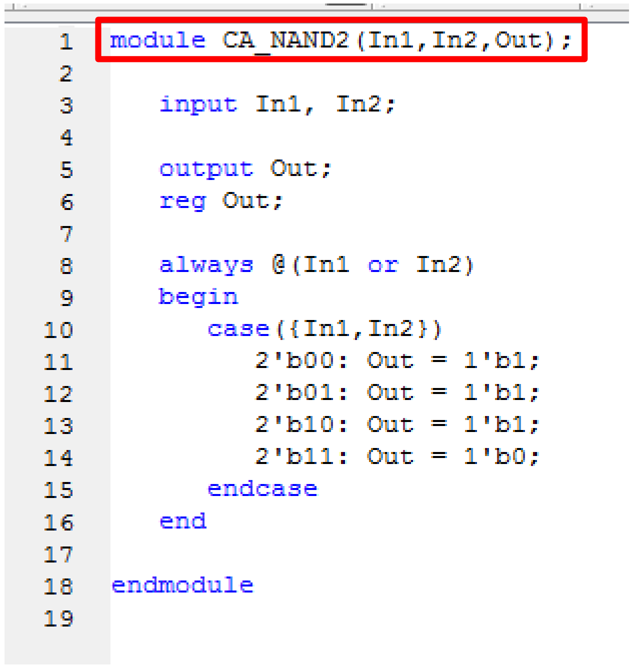
Module
(CA_AND2.v)



```
   initial begin
      In1 = 1'b0;
      In2 = 1'b0;

      #50;
      In1 = 1'b0;
      In2 = 1'b1;

      #50;
      In1 = 1'b1;
      In2 = 1'b0;

      #50;
      In1 = 1'b1;
      In2 = 1'b1;
   end
```

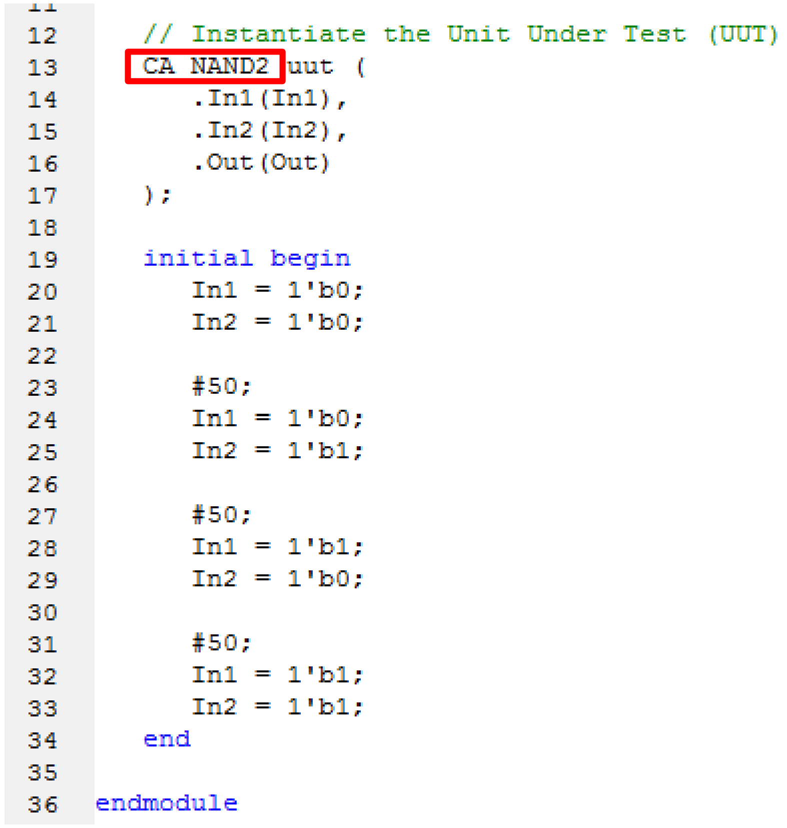
Test Bench
(tb_CA_AND2.v)

NAND Gate



```
module CA_NAND2(In1,In2,Out);

    input In1, In2;

    output Out;
    reg Out;

    always @(In1 or In2)
    begin
        case({In1,In2})
            2'b00: Out = 1'b1;
            2'b01: Out = 1'b1;
            2'b10: Out = 1'b1;
            2'b11: Out = 1'b0;
        endcase
    end

endmodule
```

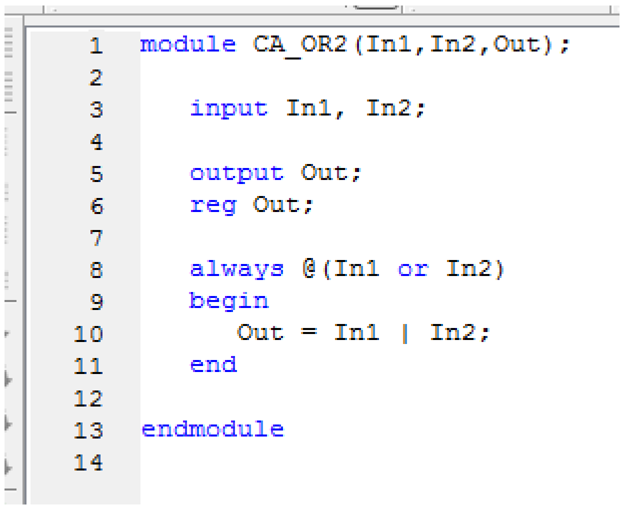
Module
(CA_NAND2.v)



```
    // Instantiate the Unit Under Test (UUT)
    CA_NAND2 uut (
        .In1(In1),
        .In2(In2),
        .Out(Out)
    );

    initial begin
        In1 = 1'b0;
        In2 = 1'b0;

        #50;
        In1 = 1'b0;
        In2 = 1'b1;

        #50;
        In1 = 1'b1;
        In2 = 1'b0;

        #50;
        In1 = 1'b1;
        In2 = 1'b1;
    end

endmodule
```

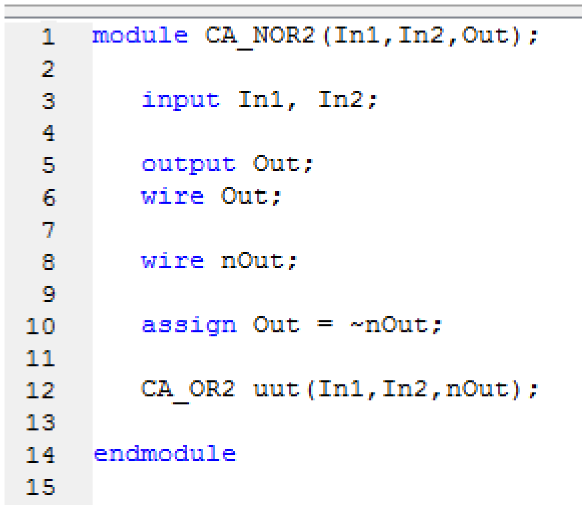
Test Bench
(tb_CA_NAND2.v)

OR, NOR Gate



```
module CA_OR2(In1,In2,Out);

    input In1, In2;

    output Out;
    reg Out;

    always @(In1 or In2)
    begin
       Out = In1 | In2;
    end

endmodule

```

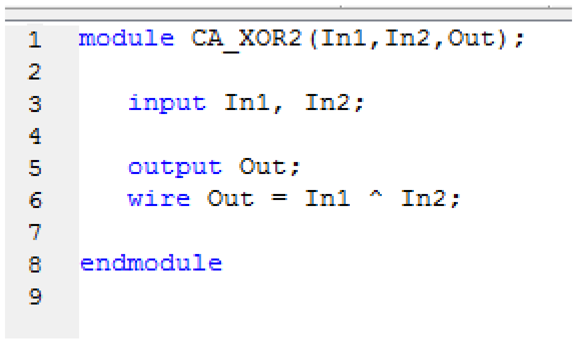
Module
(CA_OR2.v)



```
module CA_NOR2(In1,In2,Out);

    input In1, In2;

    output Out;
    wire Out;

    wire nOut;

    assign Out = ~nOut;

    CA_OR2 uut(In1,In2,nOut);

endmodule

```

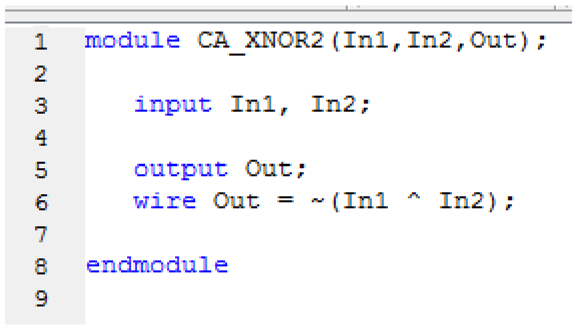
Module
(CA_NOR2.v)

XOR, XNOR Gate



```
module CA_XOR2(In1,In2,Out);

   input In1, In2;

   output Out;
   wire Out = In1 ^ In2;

endmodule

```

Module
(CA_XOR2.v)

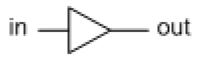


```
module CA_XNOR2(In1,In2,Out);

   input In1, In2;

   output Out;
   wire Out = ~(In1 ^ In2);

endmodule

```

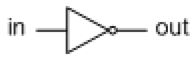
Module
(CA_XNOR2.v)

Basic Circuit

- 3-State Buffer



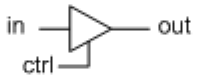
buf



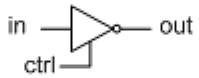
not

buf	in	out
	0	0
	1	1
	x	x
	z	x

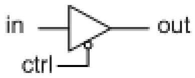
buf	in	out
	0	1
	1	0
	x	x
	z	x



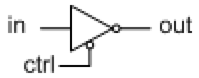
bufif1



notif1



bufif0



notif0

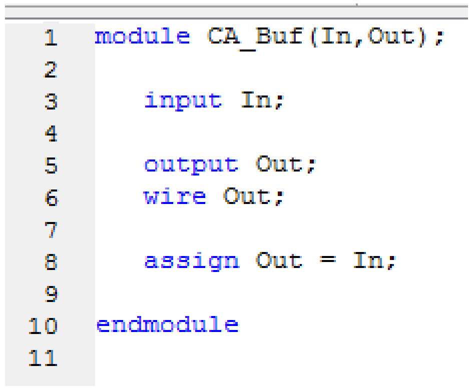
bufif1		ctrl			
		0	1	x	z
in	0	z	0	L	L
	1	z	1	H	H
	x	z	x	x	x
	z	z	x	x	x

notif1		ctrl			
		0	1	x	z
in	0	z	1	H	H
	1	z	0	L	L
	x	z	x	x	x
	z	z	x	x	x

bufif0		ctrl			
		0	1	x	z
in	0	0	z	L	L
	1	1	z	H	H
	x	x	z	x	x
	z	x	z	x	x

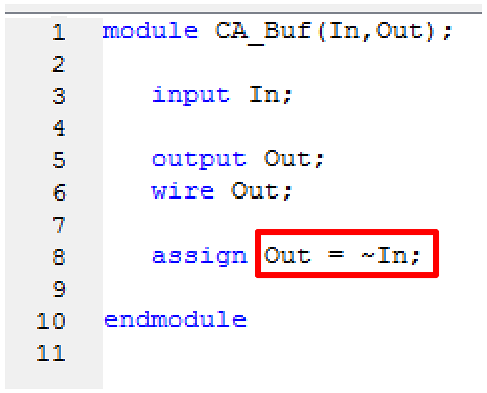
notif0		ctrl			
		0	1	x	z
in	0	1	z	H	H
	1	0	z	L	L
	x	x	z	x	x
	z	x	z	z	z

Buffer, Inverter



```
module CA_Buf(In,Out);

   input In;

   output Out;
   wire Out;

   assign Out = In;

endmodule
```

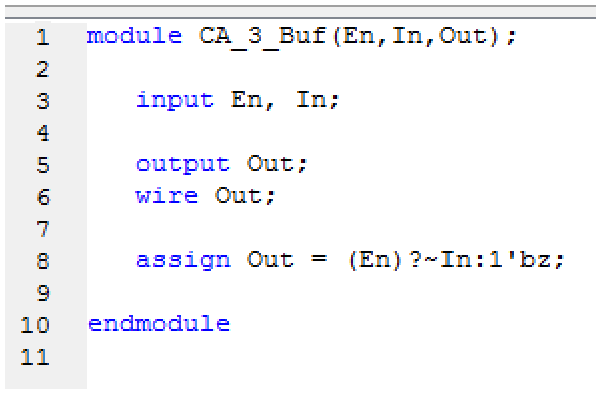
Module
(CA_Buf.v)



```
module CA_Buf(In,Out);

   input In;

   output Out;
   wire Out;

   assign Out = ~In;

endmodule
```

Module
(CA_Inv.v)

3-State Buffer

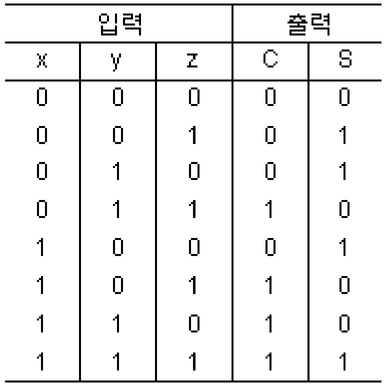


```
module CA_3_Buf(En,In,Out);

   input En, In;

   output Out;
   wire Out;

   assign Out = (En)?~In:1'bz;

endmodule

```

Module
(CA_3_Buf.v)

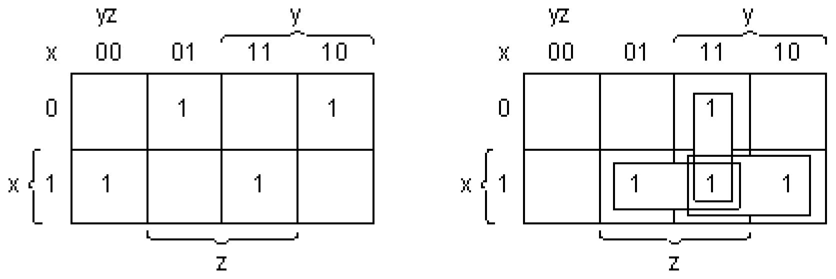
Basic Circuit

- Full-Adder



입력			출력	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

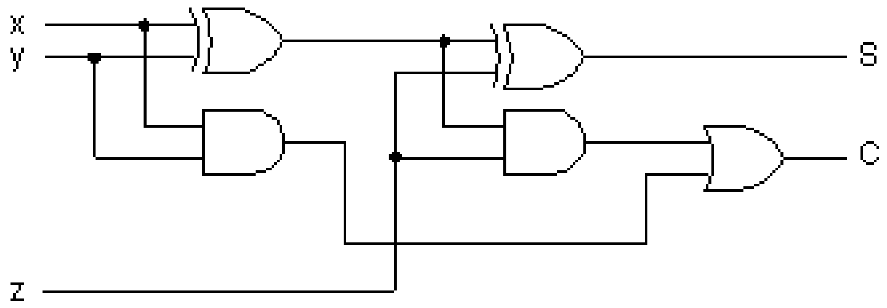
True Table



$S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$

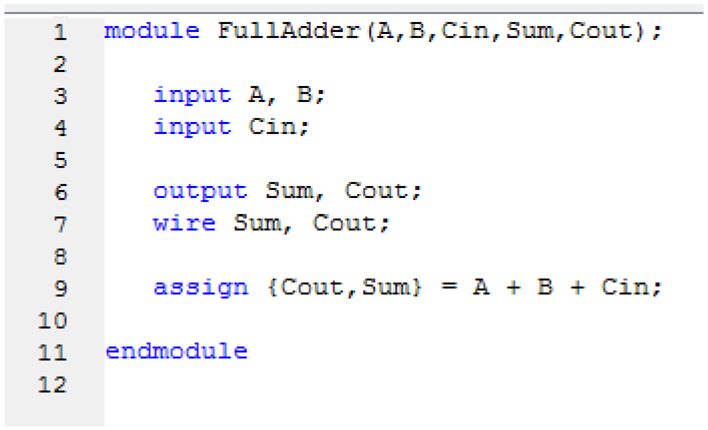
$C = xy + xz + yz$

Karnaugh map



Logic Circuit

Full-Adder

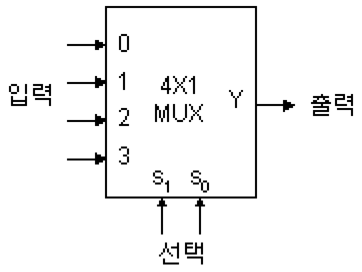


```verilog
module FullAdder(A,B,Cin,Sum,Cout);

   input A, B;
   input Cin;

   output Sum, Cout;
   wire Sum, Cout;

   assign {Cout,Sum} = A + B + Cin;

endmodule

```

Module
(FullAdder.v)

Basic Circuit

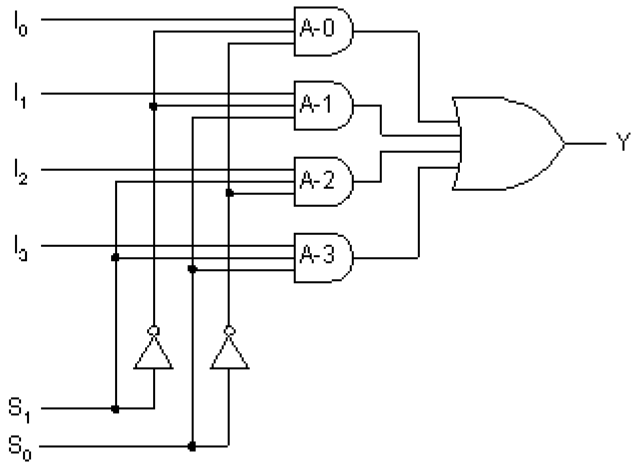
- 4x1 Multiplexer



Block Diagram

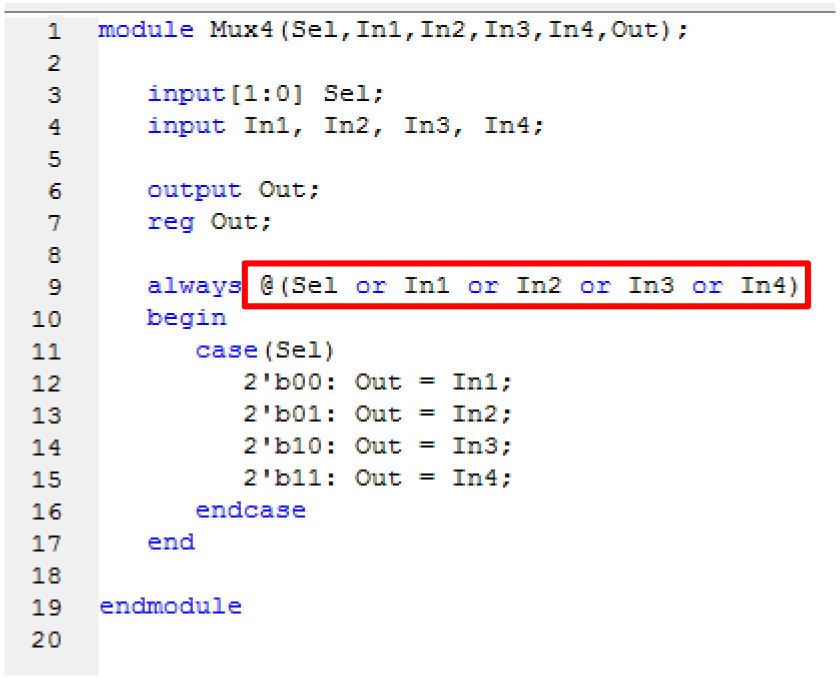
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

True Table



Logic Circuit

4x1 Multiplexer

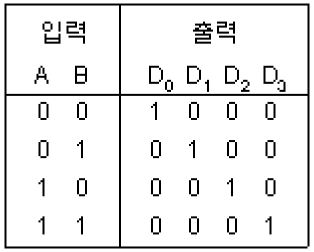


```
module Mux4(Sel,In1,In2,In3,In4,Out);

   input[1:0] Sel;
   input In1, In2, In3, In4;

   output Out;
   reg Out;

   always @(Sel or In1 or In2 or In3 or In4)
   begin
      case(Sel)
         2'b00: Out = In1;
         2'b01: Out = In2;
         2'b10: Out = In3;
         2'b11: Out = In4;
      endcase
   end

endmodule

```

Module
(Mux4.v)

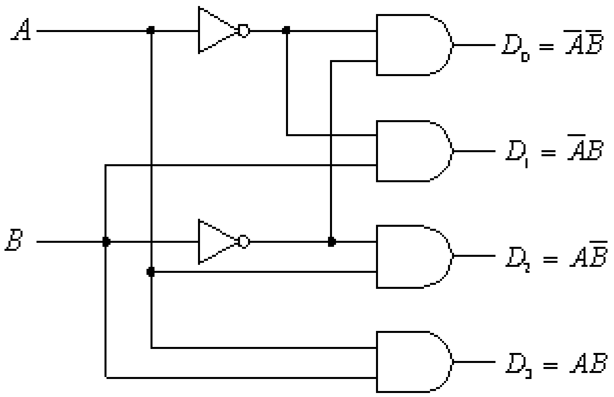
Basic Circuit

- 2x4 Decoder



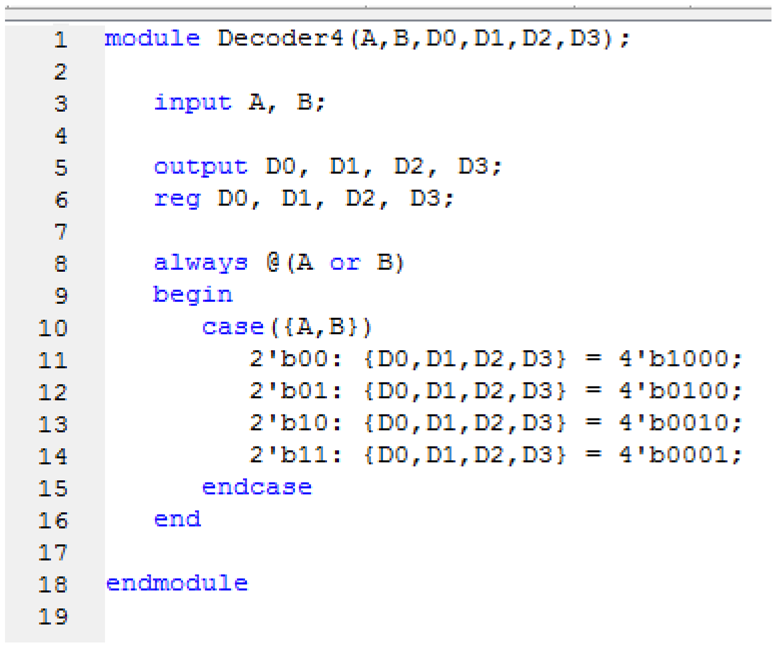
입력		출력			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

True Table



Logic Circuit

2x4 Decoder


```
module Decoder4(A,B,D0,D1,D2,D3);

   input A, B;

   output D0, D1, D2, D3;
   reg D0, D1, D2, D3;

   always @(A or B)
   begin
      case({A,B})
         2'b00: {D0,D1,D2,D3} = 4'b1000;
         2'b01: {D0,D1,D2,D3} = 4'b0100;
         2'b10: {D0,D1,D2,D3} = 4'b0010;
         2'b11: {D0,D1,D2,D3} = 4'b0001;
      endcase
   end

endmodule

```

Module
(Decoder4.v)