1. 기본 논리 회로 설계

1.1 기본 논리 게이트 실습

1) AND 게이트

```
module AND2(A, B, Y);
    input A, B ;
    output Y;
    and (Y, A, B);
    // assign Y= A & B ;  // 비트
    // assign Y= A && B ; // 논리
endmodule
```

=> OR/NAND/NOR/XOR/XNOR 게이트

2) NOT 게이트

```
module NOT( A, Y);
   input A ;
   output Y;
   not (Y, A);
   //assign Y= !A;
   //assign Y= ~A;
endmodule
```

3) BUFFER 게이트

```
module BUF( A, Y);
   input A ;
   output Y;
   buf (Y, A);
   //assign Y=A;
endmodule
```

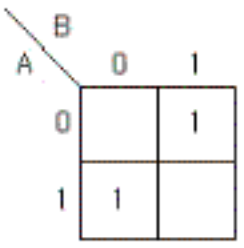
4) 3상태 버퍼

```
module S3_BUFF_1( A, E, Y);
   input A, E ;
   output Y;
   bufif1(Y, A, E);
   //bufif0(Y, A, E);
   //notif1(Y, A, E);
   //notif0(Y, A, E);
endmodule
```

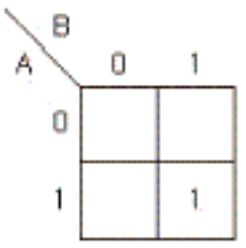
1.2 가산기 설계

1) 반가산기

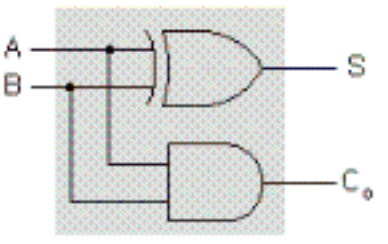
입력		출력	
A	B	S	C_o
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$S = A \oplus B$



$C_o = AB$



(a) 반가산기 변수 (b) 진리표 (c) 논리식 (d) 회로도

```
module H_ADDER(A, B, S, C_out); // 논리식 이용
   input A, B ;
   output S, C_out;
   xor(S, A, B);        //  assign S= A ^ B;
   and(C_out, A, B);    //  assign C_out= A & B;
endmodule
```

```
module H_ADDER_2(A, B, S, C_out); // 진리표이용
   input A, B ;
   output S, C_out;
   reg S, C_out;
   always @(A or B)
     begin
       if       (A==0 && B==0) begin S=0; C_out=0; end
       else if (A==0 && B==1) begin S=1; C_out=0; end
       else if (A==1 && B==0) begin S=1; C_out=0; end
       else if (A==1 && B==1) begin S=0; C_out=1; end
     end
endmodule
```

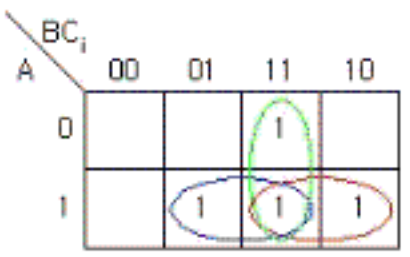
2) 전가산기

(a) 전가산기 변수

입력			출력	
A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

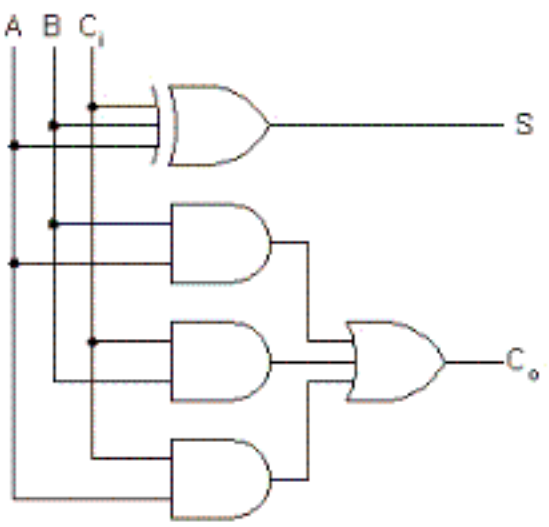
(b) 진리표

$S = A \oplus B \oplus C_i$

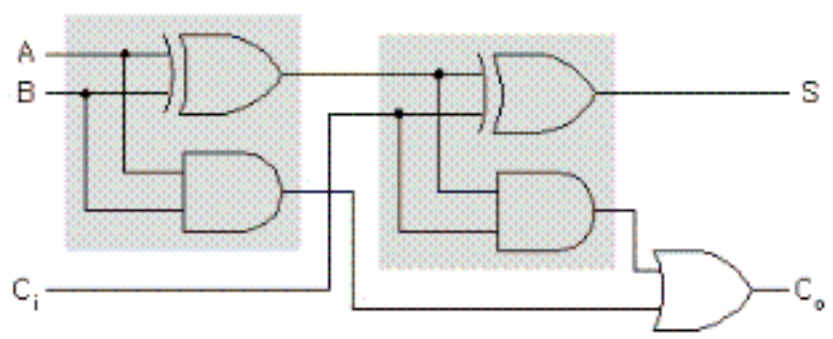


$C_o = AB + BC_i + AC_i$

(c) 논리식



(d) 회로도



$S = (A \oplus B) \oplus C_i$

$C_o = (A \oplus B)C_i + AB$

(e) 반가산기 2개와 OR 게이트를 사용한 회로도

```
module f_adder(An, Bn, Cn_1, Sn, Cn);  // (d) 회로도 이용1
   input An, Bn, Cn_1 ;
   output Sn, Cn;
   wire X0, X1, X2, X3;

   xor(X0, An, Bn);
   xor(Sn, X0, Cn_1);
   and(X1, An, Bn);
   and(X2, An, Cn_1);
   and(X3, Bn, Cn_1);
   or(Cn, X1, X2, X3);
 endmodule
```

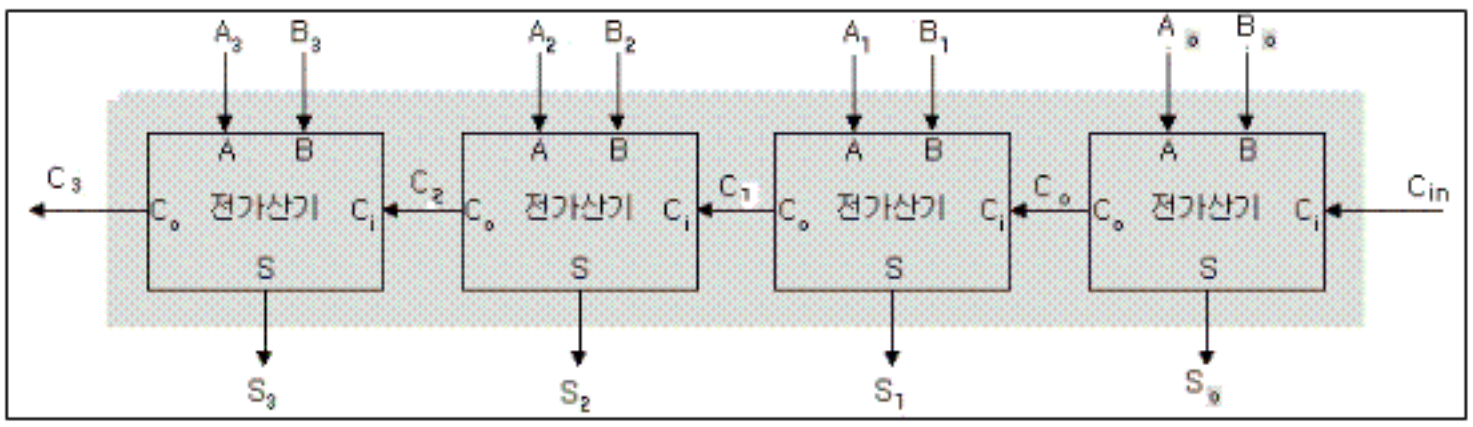
```
 module f_adder_1(An, Bn, Cn_1, Sn, Cn);  // (d) 회로도 이용2
   input An, Bn, Cn_1 ;
   output Sn, Cn;
//   wire X0, X1, X2, X3;

   xor(X0, An, Bn), (Sn, X0, Cn_1);
   and(X1, An, Bn), (X2, An, Cn_1), (X3, Bn, Cn_1);
   or(Cn, X1, X2, X3);
//   assign Sn = An ^ Bn ^ Cn_1,                  // (c) 논리식 이용
//          Cn = (An & Bn) | (An & Cn_1) | (Bn & Cn_1);
 endmodule
```

```
 module f_adder_4(An, Bn, Cn_1, Sn, Cn);  // (e) 회로도 이용
   input An, Bn, Cn_1 ;                   // 즉, 반가산기 이용
   output Sn, Cn;

   H_ADDER H_U0(An,Bn,X0,X1);       // 같은 파일 내에 H_ADDER가
   H_ADDER H_U1(X0,Cn_1,Sn,X2);     // 정의되어 있어야 함.
   or(Cn, X1, X2);
endmodule
```

```
module f_adder_5(An, Bn, Cn_1, Sn, Cn);  // 행위 수준 모델링
   input An, Bn, Cn_1 ;
   output Sn, Cn;
   reg Sn, Cn;

   always@(An or Bn or Cn_1)
    {Cn, Sn}=An + Bn + Cn_1;
endmodule
```

```
module f_adder_6(An, Bn, Cn_1, Sn, Cn);   // 진리표 이용
  input An, Bn, Cn_1 ;
  output Sn, Cn;
  reg Sn, Cn;

  always @(An or Bn or Cn_1)    // case 문으로 구현가능
   begin
    if (An==0 && Bn==0 && Cn_1==0) begin Sn=0; Cn=0; end
    else if (An==0 && Bn==0 && Cn_1==1) begin Sn=1; Cn=0; end
    else if (An==0 && Bn==1 && Cn_1==0) begin Sn=1; Cn=0; end
    else if (An==0 && Bn==1 && Cn_1==1) begin Sn=0; Cn=1; end
    else if (An==1 && Bn==0 && Cn_1==0) begin Sn=1; Cn=0; end
    else if (An==1 && Bn==0 && Cn_1==1) begin Sn=0; Cn=1; end
    else if (An==1 && Bn==1 && Cn_1==0) begin Sn=0; Cn=1; end
    else if (An==1 && Bn==1 && Cn_1==1) begin Sn=1; Cn=1; end
   end
endmodule
```

3) 4비트 가산기 설계




```
module adder_4(A0, A1, A2, A3, B0, B1, B2, B3, Cin, C3, S0, S1, S2, S3);
   input A0, A1, A2, A3, B0, B1, B2, B3, Cin ;
   output C3, S0, S1, S2, S3;
   wire C0, C1, C2;

   f_adder U0 (A0, B0, Cin, S0, C0);
   f_adder U1 (A1, B1, C0, S1, C1);
   f_adder U2 (A2, B2, C1, S2, C2);
   f_adder U3 (A3, B3, C2, S3, C3);
endmodule
```

```
module adder_4_2(A, B, Cin, S, C3);
  input [3:0] A, B ;
  input Cin;
  output C3;
  output [3:0] S;
  wire[2:0] C;

 f_adder U0 (A[0], B[0], Cin, S[0], C[0]);
 f_adder U1 (A[1], B[1], C[0], S[1], C[1]);
 f_adder U2 (A[2], B[2], C[1], S[2], C[2]);
 f_adder U3 (A[3], B[3], C[2], S[3], C3);
endmodule
```

```
module adder_4_3(A, B, Cin, S, Cout);  // 논리식을 이용한 표현, 행위 수준 모델링.
  parameter n=4;
  input [n-1:0] A, B ;
  input Cin;
  output Cout;
  output [n-1:0] S;
  reg [n-1:0] S;
  reg Cout;
  reg [n:0] C;
  integer i;

 always @(A or B or Cin)
   begin
     C[0] = Cin;
     for(i=0; i<n ; i=i+1)
       begin
         S[i] = A[i] ^ B[i] ^ C[i];
         C[i+1] = (A[i] & B[i] ) | (A[i] & C[i] ) |(B[i] & C[i]);
       end
     Cout=C[n];
   end
endmodule
```

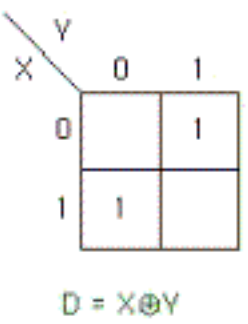
1.3 감산기 설계

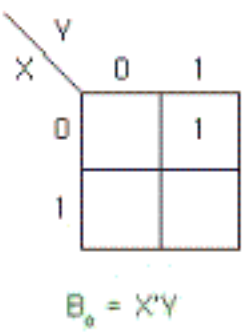
1) 반감산기

(a) 반감산기 변수

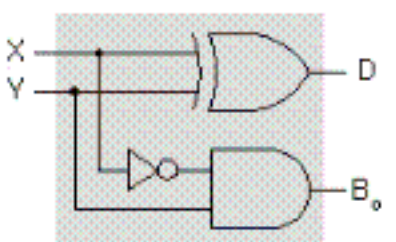
입력		출력	
X	Y	D	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

(b) 진리표





(c) 논리식



(d) 회로도

```
module H_SUBER(A, B, D, b_out);    // (d) 회로도 이용
  input A, B ;
  output D, b_out;

  not(X0, A);                 // X0 : not 게이트의 출력신호
  xor(D, A, B);
  and(b_out, X0, B);
endmodule
```

```
module H_SUBER_2(A, B, D, b_out);    // 논리식 이용
  input A, B ;
  output D, b_out;

  assign D= A ^ B;            // xor(D, A, B);
  assign b_out= !A & B;       // and(b_out, !A, B);
endmodule
```

```
module H_SUBER_3(A, B, D, b_out);  // 진리표 이용
 input A, B ;
 output D, b_out;
 reg D, b_out;

 always @(A or B)
  begin
   if      (A==0 && B==0) begin D=0; b_out=0; end
   else if (A==0 && B==1) begin D=1; b_out=1; end
   else if (A==1 && B==0) begin D=1; b_out=0; end
   else if (A==1 && B==1) begin D=0; b_out=0; end
  end
endmodule
```

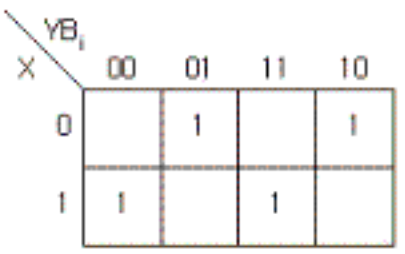
```
 // 왼쪽의 always 문 내부를
 // case 문으로 수정한 소스

case ({A,B})
 2'b00 : begin D=0;
                b_out=0; end
 2'b01 : begin D=1;
                b_out=1; end
 2'b10 : begin D=1;
                b_out=0; end
 2'b11 : begin D=0;
                b_out=0; end
endcase
```

2) 전감산기

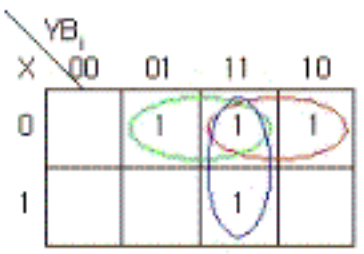
(a) 전감산기 변수

입력			출력	
X	Y	B_i	D	B_o
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(b) 진리표

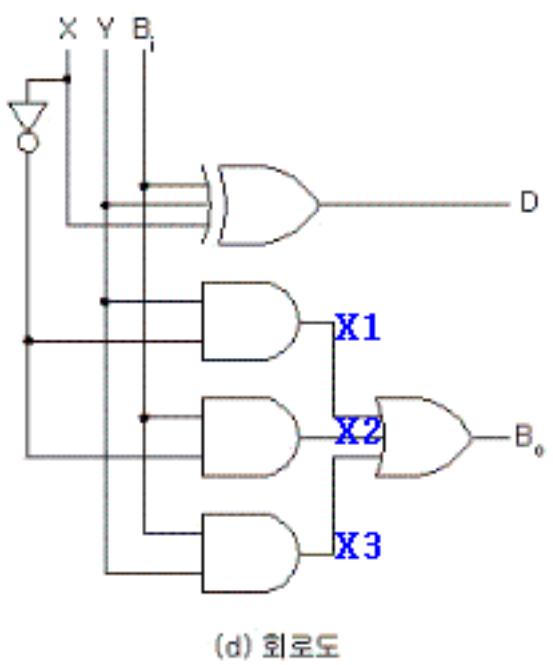


$D = X \oplus Y \oplus B_i$

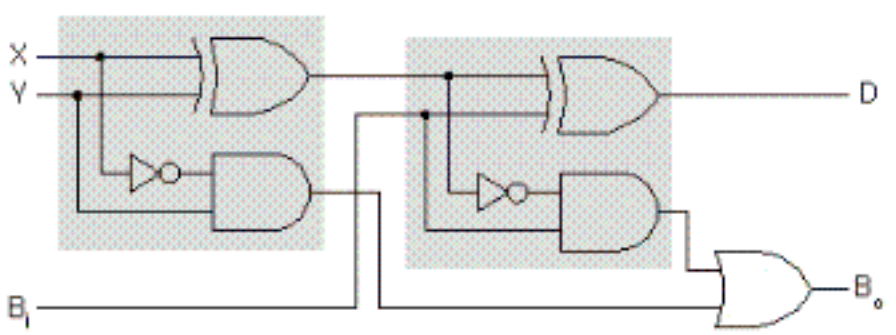


$B_o = X'Y + X'B_i + YB_i$

(c) 논리식



(d) 회로도



$D = (X \oplus Y) \oplus B_i$

$B_o = (X \oplus Y)'B_i + X'Y$

(e) 반감산기 2개와 OR 게이트를 사용한 회로도

```
module f_suber(An, Bn, bon_1, Dn, bon);
   input An, Bn, bon_1 ;            // (d) 회로도 이용
   output Dn, bon;
   wire X0, X1, X2, X3;  // AND gate의 출력

   xor(X0, An, Bn);
   xor(Dn, X0, bon_1);
   and(X1, !An, Bn);
   and(X2, !An, bon_1);
   and(X3, Bn, bon_1);
   or(bon, X1, X2, X3);
endmodule
```

```
// (e) 회로도 이용.
// => 즉, 반감산기를 이용.

 xor(X1, An, Bn);
 xor(Dn, X1, bon_1);
 and(X2, !An, Bn),
     (X5, !X1, bon_1);
 or(bon, X2, X5);
```

```
   assign Dn = An ^ Bn ^ bon_1;    // 논리식 이용.
   assign bon = (!An & Bn) | (!An & bon_1) | (Bn & bon_1);
```

```
  // 위의 반감산기 H_SUBER_2를 파생하여 구현. (e) 회로도 이용.
   H_SUBER_2 H_U0(An,Bn,X1,X0);
   H_SUBER_2 H_U1(X1,bon_1,Dn,X2);
   or(bon, X0, X2);
```

```
module f_suber_4(An, Bn, bon_1, Dn, bon);
  input An, Bn, bon_1 ;
  output Dn, bon;
  reg Dn, bon;
  always @(An or Bn or bon_1)
   begin
    if (An==0 && Bn==0 && bon_1==0) begin Dn=0; bon=0; end
    else if (An==0 && Bn==0 && bon_1==1) begin Dn=1; bon=1; end
    else if (An==0 && Bn==1 && bon_1==0) begin Dn=1; bon=1; end
    else if (An==0 && Bn==1 && bon_1==1) begin Dn=0; bon=1; end
    else if (An==1 && Bn==0 && bon_1==0) begin Dn=1; bon=0; end
    else if (An==1 && Bn==0 && bon_1==1) begin Dn=0; bon=0; end
    else if (An==1 && Bn==1 && bon_1==0) begin Dn=0; bon=0; end
    else if (An==1 && Bn==1 && bon_1==1) begin Dn=1; bon=1; end
   end
endmodule
```

```
module f_suber_5(An, Bn, bon_1, Dn, bon);
  input An, Bn, bon_1 ;
  output Dn, bon;
  reg Dn, bon;

  always@(An or Bn or bon_1)
   {bon, Dn}=An - Bn - bon_1;
endmodule
```

3) 4비트 전감산기

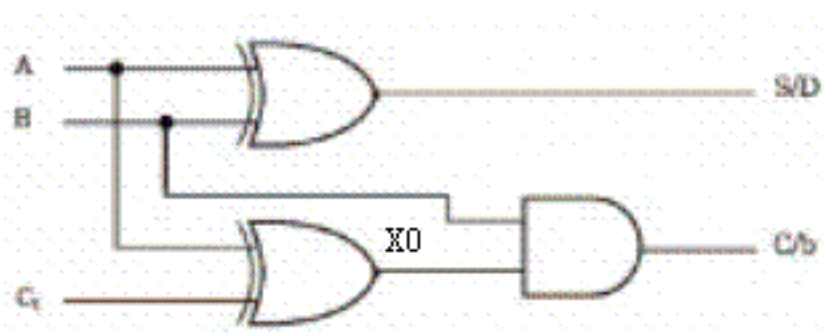
연결은 4비트 전가산기와 유사.

```
module suber_4(A0, A1, A2, A3, B0, B1, B2, B3, bin, bo3, D0, D1, D2, D3);
  input A0, A1, A2, A3, B0, B1, B2, B3, bin ;
  output bo3, D0, D1, D2, D3;
  wire bo0, bo1, bo2;

 f_suber U0 (A0, B0, bin, D0, bo0);
 f_suber U1 (A1, B1, bo0, D1, bo1);
 f_suber U2 (A2, B2, bo1, D2, bo2);
 f_suber U3 (A3, B3, bo2, D3, bo3);
endmodule
```

```
module suber_4_1(A, B, bin, D, bo3);
   input [3:0] A, B ;
   input bin;
   output [3:0] D;
   output bo3;
   wire[2:0] bo;

  f_suber_2 U0 (A[0], B[0], bin, D[0], bo[0]);
  f_suber_2 U1 (A[1], B[1], bo[0], D[1], bo[1]);
  f_suber_2 U2 (A[2], B[2], bo[1], D[2], bo[2]);
  f_suber_2 U3 (A[3], B[3], bo[2], D[3], bo3);
endmodule
```

1.4 가/감산기 회로

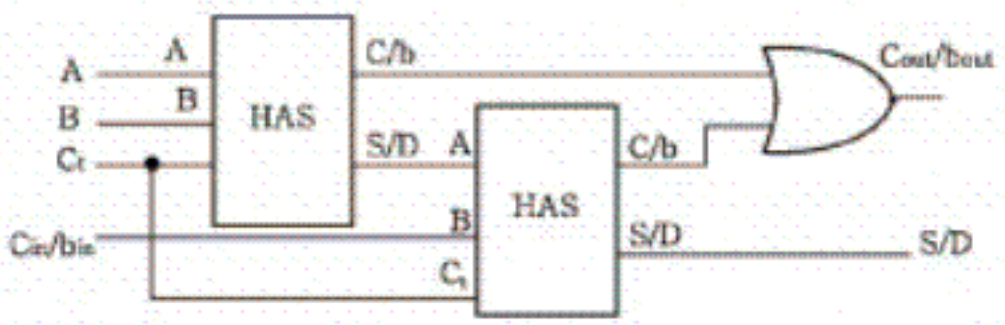
1) 반 가/감산기




```
module H_S_D(A, B, Ct, S_D, C_b);
   input A, B, Ct ;
   output S_D, C_b;

   xor(S_D, A, B);
   xor(XO, A, Ct);
   and(C_b, XO, B);
 endmodule
```

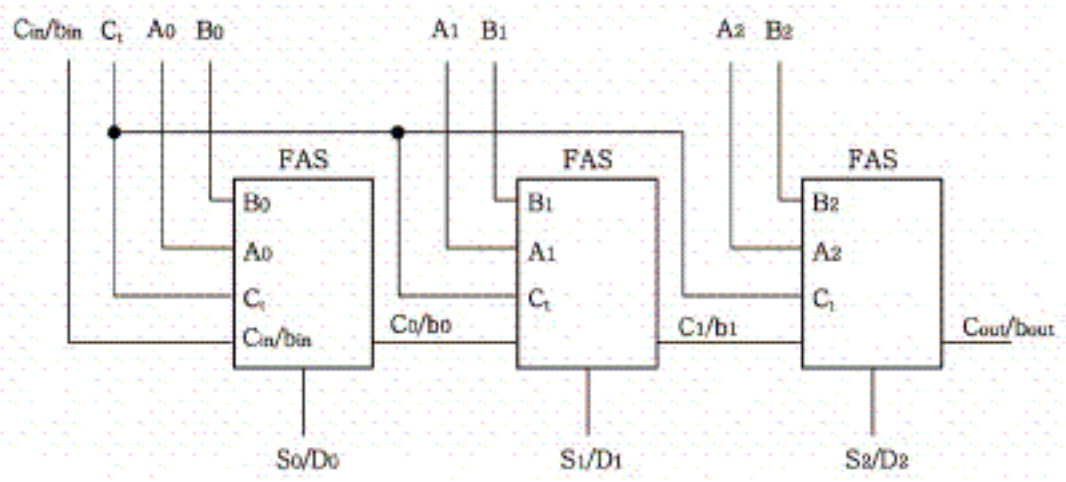
```
// 논리식 이용.
   assign S_D = A ^ B;
   assign C_b = (A^Ct) & B;
```

2) 전 가/감산기



```
module F_S_D(A, B, Ct, C_bin, S_D, C_bout);
    input A, B, Ct, C_bin ;
    output S_D, C_bout;

    H_S_D U1(A, B, Ct, X1, X0);
    H_S_D U2(X1, C_bin, Ct, S_D, X2);
    or(C_bout, X0, X2);
 endmodule
```

3) 3비트 전 가/감산기



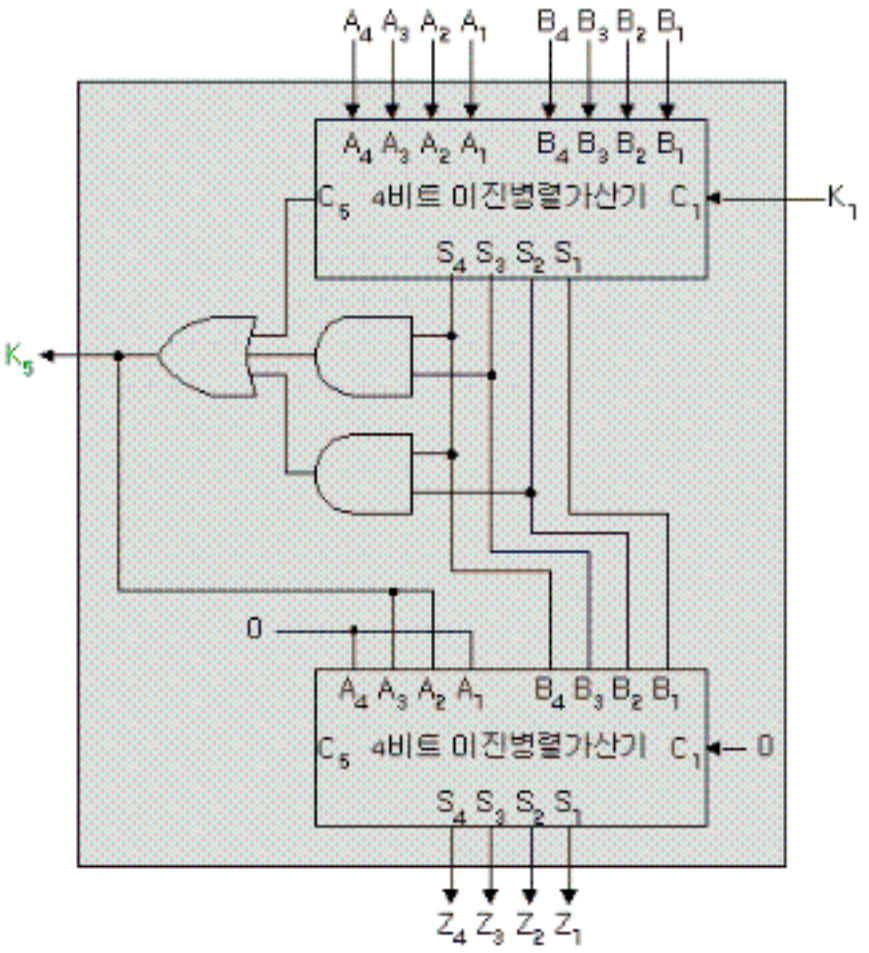

```
module S_D_3(A, B, Ct, C_bin, S_D, C_bout);
    input [2:0] A, B;
    input Ct, C_bin ;
    output [2:0] S_D ;
    output  C_bout;

    F_S_D U0(A[0], B[0], Ct, C_bin, S_D[0], C_b0);
    F_S_D U1(A[1], B[1], Ct, C_b0, S_D[1], C_b1);
    F_S_D U2(A[2], B[2], Ct, C_b1, S_D[2], C_bout);
 endmodule
```

1.5 BCD 가산기

이진병렬가산기출력 (Binary Sum)					BCD가산기출력 (BCD Sum)					십진수
C_5	S_4	S_3	S_2	S_1	K_5	Z_4	Z_3	Z_2	Z_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

$K_5 = C_5 + S_4S_3 + S_4S_2$

if K_5=1 then +6



(e)이진병렬가산기 출력을 BCD 코드로 변환

```
module BCD_ADDER_4(A, B, Cin, S, Cout);
   input  [3:0] A, B;
   input  Cin ;
   output  [3:0] S ;
   output   Cout;
   reg  [3:0] SUM, TMP;

   ADDER_4_2 U0(A, B, Cin, SUM, X1);
   and(X2, SUM[1], SUM[3]);
   and(X3, SUM[2], SUM[3]);
   or(Cout, X1, X2, X3);
   ADDER_4_2 U1({1'b0, Cout, Cout, 1'b0}, SUM, 1'b0, S, X5);
endmodule
```

```
module BCD_ADDER_4_1(A, B, Cin, S, Cout);
    input  [3:0] A, B;
    input  Cin ;
    output  [3:0] S ;
    output   Cout;
    reg  [3:0] S;
    reg  Cout;
    reg  [4:0] X;

    always@(A or B or Cin)
      begin
        X=A+B+Cin;
        if((X<0) {Cout,S}=X;
        else {Cout,S}=X+6;
      end
endmodule
```