Lab #7 Booth Multiplier

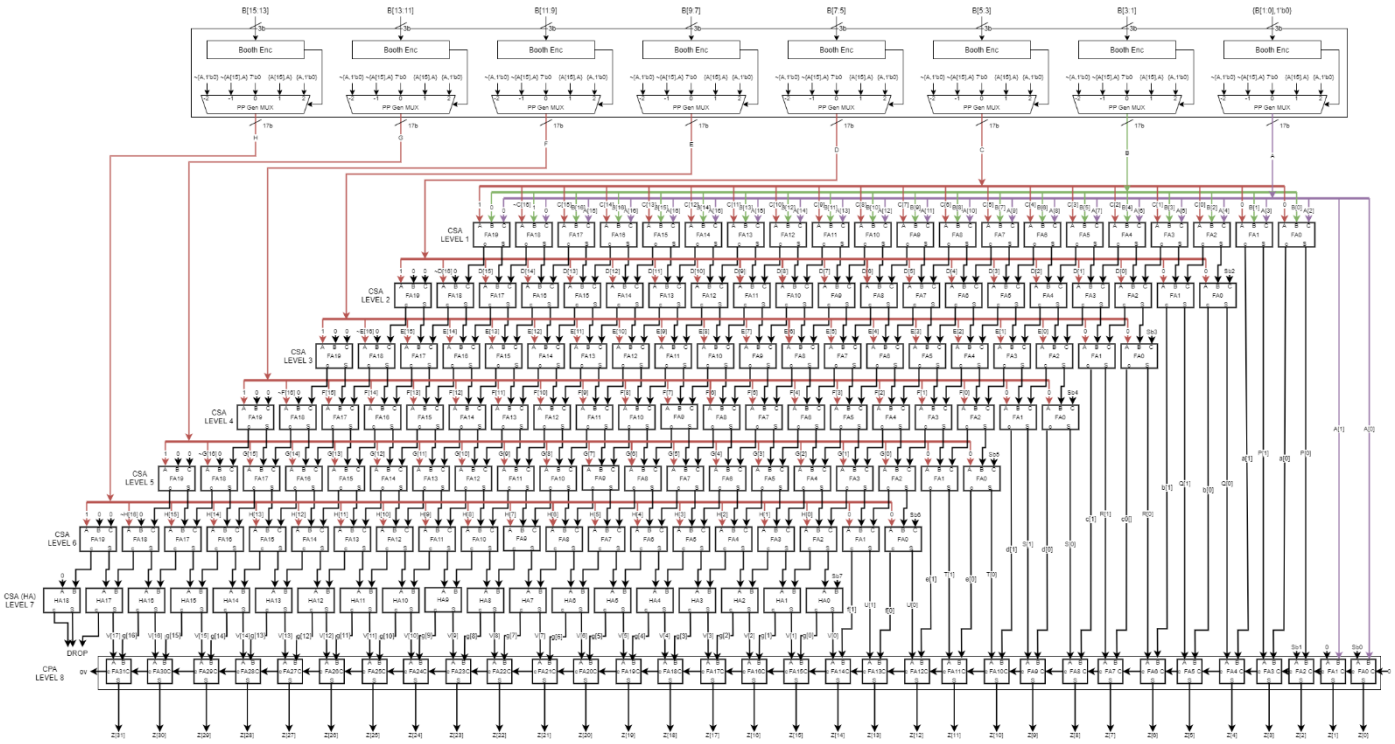
Class : 00

201602004 박태현

I. 실습 목적

Booth Multiplier의 구현

II. Design Procedure



III. Simulation

```
`define BOOTH_m2A 3'd1
`define BOOTH_mA 3'd2
`define BOOTH_0 3'd3
`define BOOTH_pA 3'd4
`define BOOTH_p2A 3'd5
```

3자리 곱하는 수에 대한 인코딩 값을 정의했습니다

```
module BOOTH(data, sel);
input [2:0] data;
output [2:0] sel;
reg [2:0] sel;

always @(data)
   case(data)
      3'b000 : begin sel = `BOOTH_0; end
      3'b010 : begin sel = `BOOTH_pA; end
      3'b100 : begin sel = `BOOTH_m2A; end
      3'b110 : begin sel = `BOOTH_mA; end
      3'b001 : begin sel = `BOOTH_pA; end
      3'b011 : begin sel = `BOOTH_p2A; end
      3'b101 : begin sel = `BOOTH_mA; end
      3'b111 : begin sel = `BOOTH_0; end
      default : begin sel = 3'bX; end
   endcase
endmodule
```

Booth 인코더입니다. 입력값에 대해 인코딩하여 sel로 넘겨줍니다

```
module PPGen (din, sel, dout, sign);
   input [15:0] din;
   input [2:0] sel; // -2A -A 0 A 2A
   output [16:0] dout;
   output sign;
   reg [16:0] dout;
   reg sign;

   always @(sel or din) begin
      casex(sel)
         `BOOTH_m2A: begin dout = ~{din, 1'b0}; sign = 1'b1; end
         `BOOTH_mA : begin dout = ~{din[15], din}; sign = 1'b1; end
         `BOOTH_0 : begin dout = 17'b0; sign = 1'b0; end
         `BOOTH_pA : begin dout = {din[15], din}; sign = 1'b0; end
         `BOOTH_p2A: begin dout = {din, 1'b0}; sign = 1'b0; end
        default : begin dout = 17'bX; sign = 1'bX; end
      endcase
   end
endmodule
```

인코딩 값을 이용해 multiplicand를 1비트 확장합니다

```
module MUL16x16(x, y, z, ov);

   input [15:0] x;
   input [15:0] y;
   output [31:0] z;
   output ov;
   /* wire delarations */

   wire sign0, sign1, sign2, sign3, sign4, sign5, sign6, sign7;
   wire [2:0] sel0, sel1, sel2, sel3, sel4, sel5, sel6, sel7;
   wire [16:0] psum0, psum1, psum2, psum3, psum4, psum5, psum6, psum7;
   wire [20:0] sum0,sum1,sum2,sum3,sum4,sum5,sum6,
               car0,car1,car2,car3,car4,car5,car6;

   wire [17:0] csum,ccar;
```

곱셈에 사용할 wire를 정의합니다.

```
BOOTH booth0(.data({y[1:0],1'b0}), .sel(sel0));
BOOTH booth1(.data(y[3:1]), .sel(sel1));
BOOTH booth2(.data(y[5:3]), .sel(sel2));
BOOTH booth3(.data(y[7:5]), .sel(sel3));
BOOTH booth4(.data(y[9:7]), .sel(sel4));
BOOTH booth5(.data(y[11:9]), .sel(sel5));
BOOTH booth6(.data(y[13:11]), .sel(sel6));
BOOTH booth7(.data(y[15:13]), .sel(sel7));
```

위에서 정의한 대로 multiplier 3자리를 받아서 인코딩합니다

```
PPGen ppgen0(.din(x),.sel(sel0),.dout(psum0),.sign(sign0)); // 16:0
PPGen ppgen1(.din(x),.sel(sel1),.dout(psum1),.sign(sign1)); // 18:2
PPGen ppgen2(.din(x),.sel(sel2),.dout(psum2),.sign(sign2)); // 20:4
PPGen ppgen3(.din(x),.sel(sel3),.dout(psum3),.sign(sign3)); // 22:6
PPGen ppgen4(.din(x),.sel(sel4),.dout(psum4),.sign(sign4)); // 24:8
PPGen ppgen5(.din(x),.sel(sel5),.dout(psum5),.sign(sign5)); // 26:10
PPGen ppgen6(.din(x),.sel(sel6),.dout(psum6),.sign(sign6)); // 28:12
PPGen ppgen7(.din(x),.sel(sel7),.dout(psum7),.sign(sign7)); // 30:28
```

인코딩된 값을 받아서 확장을 하고, 해당 값의 부호를 받습니다

```
CSA CSA0(
.a({3'b0, ~psum0[16], {2{psum0[16]}}, psum0[16:2]}),
.b({2'b0, 1'b1, ~psum1[16], psum1[16:0]}),
.c({1'b1, ~psum2[16], psum2[16:0], 2'b0}),
.cout(car0),
.sum(sum0)
);
```

최초의 레벨에서는 최하위 부분곱 3개를 CSA로 넣어서 결과를 받습니다

```
CSA CSA1(
.a({2'b0, sum0[20:2]}),
.b({1'b0, car0[20:2], 1'b0}),
.c({1'b1, ~psum3[16], psum3[16:0], 1'b0, sign2}),
.cout(car1),
.sum(sum1)
);
```

두번째 레벨부터는 이전에 생성된 각 자리의 덧셈 결과와 캐리 결과 중 일부를 받아와서 그 다음 자리의 부분곱과 더합니다

```
CSA CSA2(
.a({2'b0, sum1[20:2]}),
.b({1'b0, car1[20:2], 1'b0}),
.c({1'b1, ~psum4[16], psum4[16:0], 1'b0, sign3}),
.cout(car2),
.sum(sum2)
);

CSA CSA3(
.a({2'b0, sum2[20:2]}),
.b({1'b0, car2[20:2], 1'b0}),
.c({1'b1, ~psum5[16], psum5[16:0], 1'b0, sign4}),
.cout(car3),
.sum(sum3)
);

CSA CSA4(
.a({2'b0, sum3[20:2]}),
.b({1'b0, car3[20:2], 1'b0}),
.c({1'b1, ~psum6[16], psum6[16:0], 1'b0, sign5}),
.cout(car4),
.sum(sum4)
);

CSA CSA5(
.a({2'b0, sum4[20:2]}),
.b({1'b0, car4[20:2], 1'b0}),
.c({1'b1, ~psum7[16], psum7[16:0], 1'b0, sign6}),
.cout(car5),
.sum(sum5)
);
```

각 부분곱을 모두 더할때까지 동일하게 더해줍니다

```
CSA_HA CSAHA(
.a(sum5[19:2]),
.b({car5[18:2],sign7}),
.cout(ccar),
.sum(csum)
);
```

모든 부분곱을 더한 후에 마지막으로 나온 덧셈 결과와 캐리 결과를 한번 더 반가산기로 더해줍니다.

```
Kogge32b ksa32(
.x({csum[17:0],sum5[1:0],sum4[1:0],sum3[1:0],sum2[1:0],sum1[1:0],sum0[1:0],psum0[1:0]}),
.y({ccar[16:0],car5[1:0],car4[1:0],car3[1:0],car2[1:0],car1[1:0],car0[1:0],sign1,1'b0,sign0}),
.cin(1'b0),
.cout(ov),
.sum(z)
);
```

이제 각 자리로 나온 덧셈과 캐리를 통해서 캐리 전파 가산기를 거쳐서 2의 보수 형태의 결과로 변환해 결과를 출력합니다. CPA는 Kogge Stone Adder를 32비트로 확장한 것을 사용했습니다

```
module CSA (a, b, c, cout, sum);
   input [20:0] a;
   input [20:0] b;
   input [20:0] c;
   output [20:0] cout, sum;

   /* Adder Connection */
   FA fadd20(a[20],b[20],c[20],cout[20],sum[20]);
   FA fadd19(a[19],b[19],c[19],cout[19],sum[19]);
   FA fadd18(a[18],b[18],c[18],cout[18],sum[18]);
   FA fadd17(a[17],b[17],c[17],cout[17],sum[17]);
   FA fadd16(a[16],b[16],c[16],cout[16],sum[16]);
   FA fadd15(a[15],b[15],c[15],cout[15],sum[15]);
   FA fadd14(a[14],b[14],c[14],cout[14],sum[14]);
   FA fadd13(a[13],b[13],c[13],cout[13],sum[13]);
   FA fadd12(a[12],b[12],c[12],cout[12],sum[12]);
   FA fadd11(a[11],b[11],c[11],cout[11],sum[11]);
   FA fadd10(a[10],b[10],c[10],cout[10],sum[10]);
   FA fadd9(a[9],b[9],c[9],cout[9],sum[9]);
   FA fadd8(a[8],b[8],c[8],cout[8],sum[8]);
   FA fadd7(a[7],b[7],c[7],cout[7],sum[7]);
   FA fadd6(a[6],b[6],c[6],cout[6],sum[6]);
   FA fadd5(a[5],b[5],c[5],cout[5],sum[5]);
   FA fadd4(a[4],b[4],c[4],cout[4],sum[4]);
   FA fadd3(a[3],b[3],c[3],cout[3],sum[3]);
   FA fadd2(a[2],b[2],c[2],cout[2],sum[2]);
   FA fadd1(a[1],b[1],c[1],cout[1],sum[1]);
   FA fadd0(a[0],b[0],c[0],cout[0],sum[0]);
endmodule
```

각 자리의 CSA는 20비트를 계산합니다

```
module CSA_HA (a, b, cout, sum);
   input [17:0] a;
   input [17:0] b;
   output [17:0] cout, sum;

   /* Adder Connection */
   HA hadd17(a[17],b[17],cout[17],sum[17]);
   HA hadd16(a[16],b[16],cout[16],sum[16]);
   HA hadd15(a[15],b[15],cout[15],sum[15]);
   HA hadd14(a[14],b[14],cout[14],sum[14]);
   HA hadd13(a[13],b[13],cout[13],sum[13]);
   HA hadd12(a[12],b[12],cout[12],sum[12]);
   HA hadd11(a[11],b[11],cout[11],sum[11]);
   HA hadd10(a[10],b[10],cout[10],sum[10]);
   HA hadd9(a[9],b[9],cout[9],sum[9]);
   HA hadd8(a[8],b[8],cout[8],sum[8]);
   HA hadd7(a[7],b[7],cout[7],sum[7]);
   HA hadd6(a[6],b[6],cout[6],sum[6]);
   HA hadd5(a[5],b[5],cout[5],sum[5]);
   HA hadd4(a[4],b[4],cout[4],sum[4]);
   HA hadd3(a[3],b[3],cout[3],sum[3]);
   HA hadd2(a[2],b[2],cout[2],sum[2]);
   HA hadd1(a[1],b[1],cout[1],sum[1]);
   HA hadd0(a[0],b[0],cout[0],sum[0]);
endmodule
```

HA는 18비트를 계산합니다

```
module FA(a, b, c, cout, sum);
   input a, b, c;
   output cout, sum;
   assign {cout,sum} = a + b + c;
endmodule

module HA(a, b, cout, sum);
   input a, b;
   output cout, sum;
   assign {cout,sum} = a + b;
endmodule
```

반가산기와 전가산기를 정의했습니다

Kogge Stone Adder 32bit

```
module Kogge32b(x,y,cin,cout,sum);
input [31:0] x,y;
input cin;
output cout;
output [31:0] sum;
wire [31:0]
   G_Z,P_Z, // input_cell
   G_A,P_A, // merge level 1
   G_B,P_B, // merge level 2
   G_C,P_C, // merge level 3
   G_D,P_D, // merge level 4
   G_E,P_E; // merge level 5
wire [32:1] c;

input_cell level_Z0(x[0],y[0],P_Z[0],G_Z[0]);
input_cell level_Z1(x[1],y[1],P_Z[1],G_Z[1]);
input_cell level_Z2(x[2],y[2],P_Z[2],G_Z[2]);
input_cell level_Z3(x[3],y[3],P_Z[3],G_Z[3]);
input_cell level_Z4(x[4],y[4],P_Z[4],G_Z[4]);
input_cell level_Z5(x[5],y[5],P_Z[5],G_Z[5]);
input_cell level_Z6(x[6],y[6],P_Z[6],G_Z[6]);
input_cell level_Z7(x[7],y[7],P_Z[7],G_Z[7]);
input_cell level_Z8(x[8],y[8],P_Z[8],G_Z[8]);
input_cell level_Z9(x[9],y[9],P_Z[9],G_Z[9]);
input_cell level_Z10(x[10],y[10],P_Z[10],G_Z[10]);
input_cell level_Z11(x[11],y[11],P_Z[11],G_Z[11]);
input_cell level_Z12(x[12],y[12],P_Z[12],G_Z[12]);
input_cell level_Z13(x[13],y[13],P_Z[13],G_Z[13]);
input_cell level_Z14(x[14],y[14],P_Z[14],G_Z[14]);
input_cell level_Z15(x[15],y[15],P_Z[15],G_Z[15]);

input_cell level_Z16(x[16],y[16],P_Z[16],G_Z[16]);
input_cell level_Z17(x[17],y[17],P_Z[17],G_Z[17]);
input_cell level_Z18(x[18],y[18],P_Z[18],G_Z[18]);
input_cell level_Z19(x[19],y[19],P_Z[19],G_Z[19]);
input_cell level_Z20(x[20],y[20],P_Z[20],G_Z[20]);
input_cell level_Z21(x[21],y[21],P_Z[21],G_Z[21]);
input_cell level_Z22(x[22],y[22],P_Z[22],G_Z[22]);
input_cell level_Z23(x[23],y[23],P_Z[23],G_Z[23]);
input_cell level_Z24(x[24],y[24],P_Z[24],G_Z[24]);
input_cell level_Z25(x[25],y[25],P_Z[25],G_Z[25]);
input_cell level_Z26(x[26],y[26],P_Z[26],G_Z[26]);
input_cell level_Z27(x[27],y[27],P_Z[27],G_Z[27]);
input_cell level_Z28(x[28],y[28],P_Z[28],G_Z[28]);
input_cell level_Z29(x[29],y[29],P_Z[29],G_Z[29]);
input_cell level_Z30(x[30],y[30],P_Z[30],G_Z[30]);
input_cell level_Z31(x[31],y[31],P_Z[31],G_Z[31]);


black_cell level_0A(P_Z[0],G_Z[0],1'b0,cin       ,P_A[0],G_A[0]);
black_cell level_1A(P_Z[1],G_Z[1],P_Z[0],G_Z[0],P_A[1],G_A[1]);
black_cell level_2A(P_Z[2],G_Z[2],P_Z[1],G_Z[1],P_A[2],G_A[2]);
black_cell level_3A(P_Z[3],G_Z[3],P_Z[2],G_Z[2],P_A[3],G_A[3]);
black_cell level_4A(P_Z[4],G_Z[4],P_Z[3],G_Z[3],P_A[4],G_A[4]);
black_cell level_5A(P_Z[5],G_Z[5],P_Z[4],G_Z[4],P_A[5],G_A[5]);
black_cell level_6A(P_Z[6],G_Z[6],P_Z[5],G_Z[5],P_A[6],G_A[6]);
black_cell level_7A(P_Z[7],G_Z[7],P_Z[6],G_Z[6],P_A[7],G_A[7]);
black_cell level_8A(P_Z[8],G_Z[8],P_Z[7],G_Z[7],P_A[8],G_A[8]);
black_cell level_9A(P_Z[9],G_Z[9],P_Z[8],G_Z[8],P_A[9],G_A[9]);
black_cell level_10A(P_Z[10],G_Z[10],P_Z[9],G_Z[9],P_A[10],G_A[10]);
black_cell level_11A(P_Z[11],G_Z[11],P_Z[10],G_Z[10],P_A[11],G_A[11]);
black_cell level_12A(P_Z[12],G_Z[12],P_Z[11],G_Z[11],P_A[12],G_A[12]);
black_cell level_13A(P_Z[13],G_Z[13],P_Z[12],G_Z[12],P_A[13],G_A[13]);
black_cell level_14A(P_Z[14],G_Z[14],P_Z[13],G_Z[13],P_A[14],G_A[14]);
black_cell level_15A(P_Z[15],G_Z[15],P_Z[14],G_Z[14],P_A[15],G_A[15]);

black_cell level_16A(P_Z[16],G_Z[16],P_Z[15],G_Z[15],P_A[16],G_A[16]);
black_cell level_17A(P_Z[17],G_Z[17],P_Z[16],G_Z[16],P_A[17],G_A[17]);
black_cell level_18A(P_Z[18],G_Z[18],P_Z[17],G_Z[17],P_A[18],G_A[18]);
black_cell level_19A(P_Z[19],G_Z[19],P_Z[18],G_Z[18],P_A[19],G_A[19]);
black_cell level_20A(P_Z[20],G_Z[20],P_Z[19],G_Z[19],P_A[20],G_A[20]);
black_cell level_21A(P_Z[21],G_Z[21],P_Z[20],G_Z[20],P_A[21],G_A[21]);
black_cell level_22A(P_Z[22],G_Z[22],P_Z[21],G_Z[21],P_A[22],G_A[22]);
black_cell level_23A(P_Z[23],G_Z[23],P_Z[22],G_Z[22],P_A[23],G_A[23]);
black_cell level_24A(P_Z[24],G_Z[24],P_Z[23],G_Z[23],P_A[24],G_A[24]);
black_cell level_25A(P_Z[25],G_Z[25],P_Z[24],G_Z[24],P_A[25],G_A[25]);
black_cell level_26A(P_Z[26],G_Z[26],P_Z[25],G_Z[25],P_A[26],G_A[26]);
black_cell level_27A(P_Z[27],G_Z[27],P_Z[26],G_Z[26],P_A[27],G_A[27]);
black_cell level_28A(P_Z[28],G_Z[28],P_Z[27],G_Z[27],P_A[28],G_A[28]);
black_cell level_29A(P_Z[29],G_Z[29],P_Z[28],G_Z[28],P_A[29],G_A[29]);
black_cell level_30A(P_Z[30],G_Z[30],P_Z[29],G_Z[29],P_A[30],G_A[30]);
black_cell level_31A(P_Z[31],G_Z[31],P_Z[30],G_Z[30],P_A[31],G_A[31]);
```

```
black_cell level_1B(P_A[1],G_A[1],1'b0,cin        ,P_B[1],G_B[1]);
black_cell level_2B(P_A[2],G_A[2],P_A[0],G_A[0],P_B[2],G_B[2]);
black_cell level_3B(P_A[3],G_A[3],P_A[1],G_A[1],P_B[3],G_B[3]);
black_cell level_4B(P_A[4],G_A[4],P_A[2],G_A[2],P_B[4],G_B[4]);
black_cell level_5B(P_A[5],G_A[5],P_A[3],G_A[3],P_B[5],G_B[5]);
black_cell level_6B(P_A[6],G_A[6],P_A[4],G_A[4],P_B[6],G_B[6]);
black_cell level_7B(P_A[7],G_A[7],P_A[5],G_A[5],P_B[7],G_B[7]);
black_cell level_8B(P_A[8],G_A[8],P_A[6],G_A[6],P_B[8],G_B[8]);
black_cell level_9B(P_A[9],G_A[9],P_A[7],G_A[7],P_B[9],G_B[9]);
black_cell level_10B(P_A[10],G_A[10],P_A[8],G_A[8],P_B[10],G_B[10]);
black_cell level_11B(P_A[11],G_A[11],P_A[9],G_A[9],P_B[11],G_B[11]);
black_cell level_12B(P_A[12],G_A[12],P_A[10],G_A[10],P_B[12],G_B[12]);
black_cell level_13B(P_A[13],G_A[13],P_A[11],G_A[11],P_B[13],G_B[13]);
black_cell level_14B(P_A[14],G_A[14],P_A[12],G_A[12],P_B[14],G_B[14]);
black_cell level_15B(P_A[15],G_A[15],P_A[13],G_A[13],P_B[15],G_B[15]);

black_cell level_16B(P_A[16],G_A[16],P_A[14],G_A[14],P_B[16],G_B[16]);
black_cell level_17B(P_A[17],G_A[17],P_A[15],G_A[15],P_B[17],G_B[17]);
black_cell level_18B(P_A[18],G_A[18],P_A[16],G_A[16],P_B[18],G_B[18]);
black_cell level_19B(P_A[19],G_A[19],P_A[17],G_A[17],P_B[19],G_B[19]);
black_cell level_20B(P_A[20],G_A[20],P_A[18],G_A[18],P_B[20],G_B[20]);
black_cell level_21B(P_A[21],G_A[21],P_A[19],G_A[19],P_B[21],G_B[21]);
black_cell level_22B(P_A[22],G_A[22],P_A[20],G_A[20],P_B[22],G_B[22]);
black_cell level_23B(P_A[23],G_A[23],P_A[21],G_A[21],P_B[23],G_B[23]);
black_cell level_24B(P_A[24],G_A[24],P_A[22],G_A[22],P_B[24],G_B[24]);
black_cell level_25B(P_A[25],G_A[25],P_A[23],G_A[23],P_B[25],G_B[25]);
black_cell level_26B(P_A[26],G_A[26],P_A[24],G_A[24],P_B[26],G_B[26]);
black_cell level_27B(P_A[27],G_A[27],P_A[25],G_A[25],P_B[27],G_B[27]);
black_cell level_28B(P_A[28],G_A[28],P_A[26],G_A[26],P_B[28],G_B[28]);
black_cell level_29B(P_A[29],G_A[29],P_A[27],G_A[27],P_B[29],G_B[29]);
black_cell level_30B(P_A[30],G_A[30],P_A[28],G_A[28],P_B[30],G_B[30]);
black_cell level_31B(P_A[31],G_A[31],P_A[29],G_A[29],P_B[31],G_B[31]);

black_cell level_3C(P_B[3],G_B[3],1'b0,cin        ,P_C[3],G_C[3]);
black_cell level_4C(P_B[4],G_B[4],P_A[0],G_A[0],P_C[4],G_C[4]);
black_cell level_5C(P_B[5],G_B[5],P_B[1],G_B[1],P_C[5],G_C[5]);
black_cell level_6C(P_B[6],G_B[6],P_B[2],G_B[2],P_C[6],G_C[6]);
black_cell level_7C(P_B[7],G_B[7],P_B[3],G_B[3],P_C[7],G_C[7]);
black_cell level_8C(P_B[8],G_B[8],P_B[4],G_B[4],P_C[8],G_C[8]);
black_cell level_9C(P_B[9],G_B[9],P_B[5],G_B[5],P_C[9],G_C[9]);
black_cell level_10C(P_B[10],G_B[10],P_B[6],G_B[6],P_C[10],G_C[10]);
black_cell level_11C(P_B[11],G_B[11],P_B[7],G_B[7],P_C[11],G_C[11]);
black_cell level_12C(P_B[12],G_B[12],P_B[8],G_B[8],P_C[12],G_C[12]);
black_cell level_13C(P_B[13],G_B[13],P_B[9],G_B[9],P_C[13],G_C[13]);
black_cell level_14C(P_B[14],G_B[14],P_B[10],G_B[10],P_C[14],G_C[14]);
black_cell level_15C(P_B[15],G_B[15],P_B[11],G_B[11],P_C[15],G_C[15]);

black_cell level_16C(P_B[16],G_B[16],P_B[12],G_B[12],P_C[16],G_C[16]);
black_cell level_17C(P_B[17],G_B[17],P_B[13],G_B[13],P_C[17],G_C[17]);
black_cell level_18C(P_B[18],G_B[18],P_B[14],G_B[14],P_C[18],G_C[18]);
black_cell level_19C(P_B[19],G_B[19],P_B[15],G_B[15],P_C[19],G_C[19]);
black_cell level_20C(P_B[20],G_B[20],P_B[16],G_B[16],P_C[20],G_C[20]);
black_cell level_21C(P_B[21],G_B[21],P_B[17],G_B[17],P_C[21],G_C[21]);
black_cell level_22C(P_B[22],G_B[22],P_B[18],G_B[18],P_C[22],G_C[22]);
black_cell level_23C(P_B[23],G_B[23],P_B[19],G_B[19],P_C[23],G_C[23]);
black_cell level_24C(P_B[24],G_B[24],P_B[20],G_B[20],P_C[24],G_C[24]);
black_cell level_25C(P_B[25],G_B[25],P_B[21],G_B[21],P_C[25],G_C[25]);
black_cell level_26C(P_B[26],G_B[26],P_B[22],G_B[22],P_C[26],G_C[26]);
black_cell level_27C(P_B[27],G_B[27],P_B[23],G_B[23],P_C[27],G_C[27]);
black_cell level_28C(P_B[28],G_B[28],P_B[24],G_B[24],P_C[28],G_C[28]);
black_cell level_29C(P_B[29],G_B[29],P_B[25],G_B[25],P_C[29],G_C[29]);
black_cell level_30C(P_B[30],G_B[30],P_B[26],G_B[26],P_C[30],G_C[30]);
black_cell level_31C(P_B[31],G_B[31],P_B[27],G_B[27],P_C[31],G_C[31]);

black_cell level_7D(P_C[7],G_C[7],1'b0,cin        ,P_D[7],G_D[7]);
black_cell level_8D(P_C[8],G_C[8],P_A[0],G_A[0],P_D[8],G_D[8]);
black_cell level_9D(P_C[9],G_C[9],P_B[1],G_B[1],P_D[9],G_D[9]);
black_cell level_10D(P_C[10],G_C[10],P_B[2],G_B[2],P_D[10],G_D[10]);
black_cell level_11D(P_C[11],G_C[11],P_C[3],G_C[3],P_D[11],G_D[11]);
black_cell level_12D(P_C[12],G_C[12],P_C[4],G_C[4],P_D[12],G_D[12]);
black_cell level_13D(P_C[13],G_C[13],P_C[5],G_C[5],P_D[13],G_D[13]);
black_cell level_14D(P_C[14],G_C[14],P_C[6],G_C[6],P_D[14],G_D[14]);
black_cell level_15D(P_C[15],G_C[15],P_C[7],G_C[7],P_D[15],G_D[15]);

black_cell level_16D(P_C[16],G_C[16],P_C[8],G_C[8],P_D[16],G_D[16]);
black_cell level_17D(P_C[17],G_C[17],P_C[9],G_C[9],P_D[17],G_D[17]);
black_cell level_18D(P_C[18],G_C[18],P_C[10],G_C[10],P_D[18],G_D[18]);
black_cell level_19D(P_C[19],G_C[19],P_C[11],G_C[11],P_D[19],G_D[19]);
black_cell level_20D(P_C[20],G_C[20],P_C[12],G_C[12],P_D[20],G_D[20]);
black_cell level_21D(P_C[21],G_C[21],P_C[13],G_C[13],P_D[21],G_D[21]);
black_cell level_22D(P_C[22],G_C[22],P_C[14],G_C[14],P_D[22],G_D[22]);
black_cell level_23D(P_C[23],G_C[23],P_C[15],G_C[15],P_D[23],G_D[23]);
black_cell level_24D(P_C[24],G_C[24],P_C[16],G_C[16],P_D[24],G_D[24]);
black_cell level_25D(P_C[25],G_C[25],P_C[17],G_C[17],P_D[25],G_D[25]);
black_cell level_26D(P_C[26],G_C[26],P_C[18],G_C[18],P_D[26],G_D[26]);
black_cell level_27D(P_C[27],G_C[27],P_C[19],G_C[19],P_D[27],G_D[27]);
black_cell level_28D(P_C[28],G_C[28],P_C[20],G_C[20],P_D[28],G_D[28]);
black_cell level_29D(P_C[29],G_C[29],P_C[21],G_C[21],P_D[29],G_D[29]);
black_cell level_30D(P_C[30],G_C[30],P_C[22],G_C[22],P_D[30],G_D[30]);
black_cell level_31D(P_C[31],G_C[31],P_C[23],G_C[23],P_D[31],G_D[31]);
```

```
black_cell level_15E(P_D[15],G_D[15],1'b0, cin       ,P_E[15],G_E[15]);

black_cell level_16E(P_D[16],G_D[16],P_A[0], G_A[0],P_E[16],G_E[16]);

black_cell level_17E(P_D[17],G_D[17],P_B[1], G_B[1],P_E[17],G_E[17]);
black_cell level_18E(P_D[18],G_D[18],P_B[2], G_B[2],P_E[18],G_E[18]);

black_cell level_19E(P_D[19],G_D[19],P_C[3], G_C[3],P_E[19],G_E[19]);
black_cell level_20E(P_D[20],G_D[20],P_C[4], G_C[4],P_E[20],G_E[20]);
black_cell level_21E(P_D[21],G_D[21],P_C[5], G_C[5],P_E[21],G_E[21]);
black_cell level_22E(P_D[22],G_D[22],P_C[6], G_C[6],P_E[22],G_E[22]);

black_cell level_23E(P_D[23],G_D[23],P_D[7], G_D[7],P_E[23],G_E[23]);
black_cell level_24E(P_D[24],G_D[24],P_D[8], G_D[8],P_E[24],G_E[24]);
black_cell level_25E(P_D[25],G_D[25],P_D[9], G_D[9],P_E[25],G_E[25]);
black_cell level_26E(P_D[26],G_D[26],P_D[10], G_D[10],P_E[26],G_E[26]);
black_cell level_27E(P_D[27],G_D[27],P_D[11], G_D[11],P_E[27],G_E[27]);
black_cell level_28E(P_D[28],G_D[28],P_D[12], G_D[12],P_E[28],G_E[28]);
black_cell level_29E(P_D[29],G_D[29],P_D[13], G_D[13],P_E[29],G_E[29]);
black_cell level_30E(P_D[30],G_D[30],P_D[14], G_D[14],P_E[30],G_E[30]);
black_cell level_31E(P_D[31],G_D[31],P_D[15], G_D[15],P_E[31],G_E[31]);

carry_eval_cell c1(P_A[0],G_A[0],cin,c[1]);

carry_eval_cell c2(P_B[1],G_B[1],cin,c[2]);
carry_eval_cell c3(P_B[2],G_B[2],cin,c[3]);

carry_eval_cell c4(P_C[3],G_C[3],cin,c[4]);
carry_eval_cell c5(P_C[4],G_C[4],cin,c[5]);
carry_eval_cell c6(P_C[5],G_C[5],cin,c[6]);
carry_eval_cell c7(P_C[6],G_C[6],cin,c[7]);

carry_eval_cell c8(P_D[7],G_D[7],cin,c[8]);
carry_eval_cell c9(P_D[8],G_D[8],cin,c[9]);
carry_eval_cell c10(P_D[9],G_D[9],cin,c[10]);
carry_eval_cell c11(P_D[10],G_D[10],cin,c[11]);
carry_eval_cell c12(P_D[11],G_D[11],cin,c[12]);
carry_eval_cell c13(P_D[12],G_D[12],cin,c[13]);
carry_eval_cell c14(P_D[13],G_D[13],cin,c[14]);
carry_eval_cell c15(P_D[14],G_D[14],cin,c[15]);

carry_eval_cell c16(P_E[15],G_E[15],cin,c[16]);
carry_eval_cell c17(P_E[16],G_E[16],cin,c[17]);
carry_eval_cell c18(P_E[17],G_E[17],cin,c[18]);
carry_eval_cell c19(P_E[18],G_E[18],cin,c[19]);
carry_eval_cell c20(P_E[19],G_E[19],cin,c[20]);
carry_eval_cell c21(P_E[20],G_E[20],cin,c[21]);
carry_eval_cell c22(P_E[21],G_E[21],cin,c[22]);
carry_eval_cell c23(P_E[22],G_E[22],cin,c[23]);
carry_eval_cell c24(P_E[23],G_E[23],cin,c[24]);
carry_eval_cell c25(P_E[24],G_E[24],cin,c[25]);
carry_eval_cell c26(P_E[25],G_E[25],cin,c[26]);
carry_eval_cell c27(P_E[26],G_E[26],cin,c[27]);
carry_eval_cell c28(P_E[27],G_E[27],cin,c[28]);
carry_eval_cell c29(P_E[28],G_E[28],cin,c[29]);
carry_eval_cell c30(P_E[29],G_E[29],cin,c[30]);
carry_eval_cell c31(P_E[30],G_E[30],cin,c[31]);
carry_eval_cell c32(P_E[31],G_E[31],cin,c[32]);

assign sum[0] = cin ^ P_Z[0];
assign sum[1] = c[1] ^ P_Z[1];
assign sum[2] = c[2] ^ P_Z[2];
assign sum[3] = c[3] ^ P_Z[3];
assign sum[4] = c[4] ^ P_Z[4];
assign sum[5] = c[5] ^ P_Z[5];
assign sum[6] = c[6] ^ P_Z[6];
assign sum[7] = c[7] ^ P_Z[7];
assign sum[8] = c[8] ^ P_Z[8];
assign sum[9] = c[9] ^ P_Z[9];
assign sum[10] = c[10] ^ P_Z[10];
assign sum[11] = c[11] ^ P_Z[11];
assign sum[12] = c[12] ^ P_Z[12];
assign sum[13] = c[13] ^ P_Z[13];
assign sum[14] = c[14] ^ P_Z[14];
assign sum[15] = c[15] ^ P_Z[15];

assign sum[16] = c[16] ^ P_Z[16];
assign sum[17] = c[17] ^ P_Z[17];
assign sum[18] = c[18] ^ P_Z[18];
assign sum[19] = c[19] ^ P_Z[19];
assign sum[20] = c[20] ^ P_Z[20];
assign sum[21] = c[21] ^ P_Z[21];
assign sum[22] = c[22] ^ P_Z[22];
assign sum[23] = c[23] ^ P_Z[23];
assign sum[24] = c[24] ^ P_Z[24];
assign sum[25] = c[25] ^ P_Z[25];
assign sum[26] = c[26] ^ P_Z[26];
assign sum[27] = c[27] ^ P_Z[27];
assign sum[28] = c[28] ^ P_Z[28];
assign sum[29] = c[29] ^ P_Z[29];
assign sum[30] = c[30] ^ P_Z[30];
assign sum[31] = c[31] ^ P_Z[31];

assign cout = c[32];
```

```
module input_cell(a,b,p,g);
input a,b;
output p,g;

assign p = a ^ b;
assign g = a & b;
endmodule

module black_cell(Pim,Gim,Pmj,Gmj,P,G);
input Pim,Gim,Pmj,Gmj;
output P,G;

assign P = Pim & Pmj;
assign G = Gim | (Pim & Gmj);

endmodule

module carry_eval_cell(Pi0,Gi0,c0,ci);
input Gi0,Pi0,c0;
output ci;

assign ci = Gi0 | (Pi0 & c0);

endmodule
```

이전에 구현한 것을 확장했습니다

- Testbench

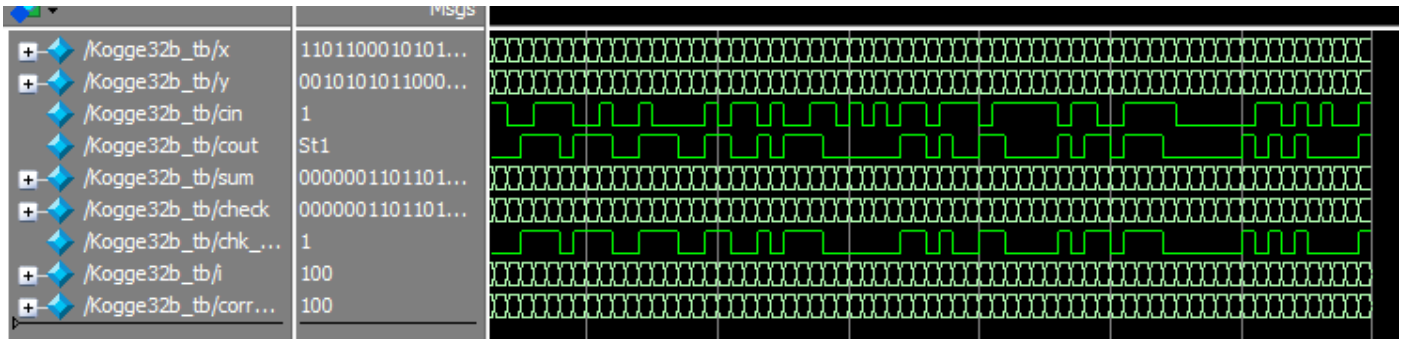
```
module Kogge32b_tb;

reg [31:0] x,y;
reg cin;
wire cout;
wire [31:0] sum;
reg [31:0] check;
reg chk_out;

parameter iter = 100;
integer i;
integer correct_cnt;


Kogge32b iks(x,y,cin,cout,sum);

initial begin
   correct_cnt = 0;
   for(i = 0; i < iter; i=i+1) begin
      x = $random;
      y = $random;
      cin = $random;
      {chk_out, check} = x + y + cin;
      #10
      if({cout, sum} == {chk_out, check})
         correct_cnt = correct_cnt + 1;
      else
         $display($time, " : %d + %d + %d = %d, but was (%d)\n",x,y,cin,{chk_out, check},{cout,sum});
   end

   $display("correct count = %d\n",correct_cnt);
end

endmodule
```

이전에 테스트한 것을 확장하여 테스트를 했습니다

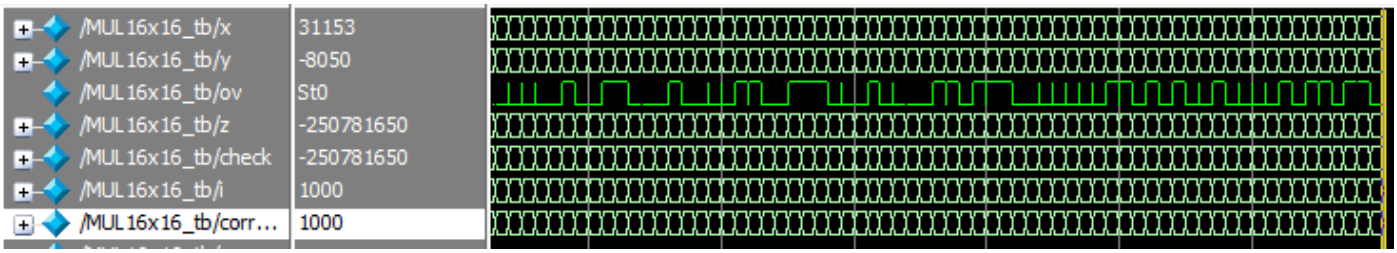
```
module MUL16x16_tb;

    reg signed [15:0] x,y;
    wire ov;
    wire signed [31:0] z;
    reg signed [31:0] check;

    parameter iter = 1000;
    integer i;
    integer correct_cnt;
    integer seed = 3;

    MUL16x16 mul(x,y,z,ov);

    initial begin
      correct_cnt = 0;
      for(i = 0; i < iter; i=i+1) begin
         x = $random(seed);
         y = $random(seed);
         check = x * y;
         #10
         if(check == z)
            correct_cnt = correct_cnt + 1;
         else
            $display($time, " : %d * %d = %d, but was (%d)\n",x,y,check,z);
      end
      $display("correct count = %d\n",correct_cnt);
    end

endmodule
```

비교할 정답을 부호 있는 reg 묶음으로 정의하고 단순히 곱셈을 한것과 곱셈기를 거친 것을 비교했습니다

IV. Evaluation



KSA를 먼저 테스트하여 검증을 했습니다



1000번의 테스트에서 모두 일반 곱셈 결과와 동일하다는 결과를 얻었습니다

V. Discussion

테스트 벤치에 문제가 있는데 테스트에 문제가 있을 것이라 생각하지 못해서 고생을 했습니다.

테스트 벤치도 잘 작성해야된다는 것을 알았습니다.

곱셈에 들어가는 계산 과정을 CSA 파이프라인을 이용해 효과적으로 줄일 수 있다는 것을 알았습니다

복잡하더라도 블록 다이어그램을 그리는 것이 구현과 이해에 더 도움이 되는 것 같습니다

0을 계속 1'b0으로 쓰지 않고 0으로 쓰는 실수를 했습니다