SoC설계

Lab#9

Booth Multiplier

컴퓨터공학과

201402439

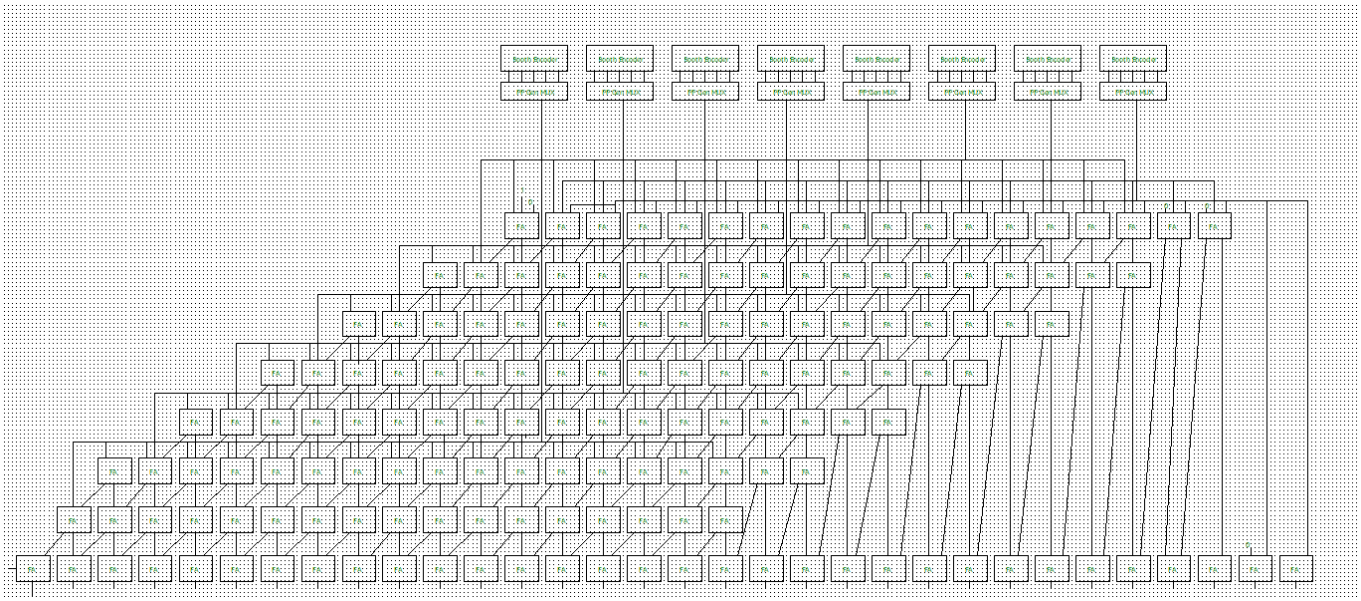
천원준

1. Purpose of the lab

이번 과제의 목표는 Booth Multiplier를 구현하고, 올바르게 작동되는지 확인해보는 것입니다.

2. Design Procedure

- Block diagram



3. Simulation

- Booth Index Encoder

```
`define p2A 3'd1
`define pA 3'd2
`define zero 3'd3
`define mA 3'd4
`define m2A 3'd5

//Radix-4 Booth encoder
module Booth_Index_Enc(B, Index, Sign);
   input [2:0] B;
   output [2:0] Index;
   output Sign;


   reg [2:0]Index;
   reg Sign;

   always @ (B) begin
      case(B)
         3'b000: begin Index = `zero; Sign = 1'b0; end
         3'b010: begin Index = `pA; Sign = 1'b0; end
         3'b100: begin Index = `m2A; Sign = 1'b1; end
         3'b110: begin Index = `mA; Sign = 1'b1; end
         3'b001: begin Index = `pA; Sign = 1'b0; end
         3'b011: begin Index = `p2A; Sign = 1'b0; end
         3'b101: begin Index = `mA; Sign = 1'b1; end
         3'b111: begin Index = `zero; Sign = 1'b0; end
         default: begin Index = 3'bx; Sign = 1'bx; end
      endcase
   end
endmodule
```

Radix-4 Booth Encoding의 방식에 맞추어 3비트의 입력 B로 출력 Index를 정합니다.

- Partial Product Generator

```
//16bit Partial Product Generation MUX
module pp_Gen_MUX(A, Index, Dout);

   input [15:0] A;
   input [2:0] Index;
   output [16:0] Dout;
   reg [16:0] Dout;

   always @ (A or Index) begin
      casex(Index)
         `p2A: Dout = {A, 1'b0};
         `pA: Dout = {A[15], A};
         `zero: Dout = 17'b0;
         `mA: Dout = ~{A[15], A};
         `m2A: Dout = ~{A, 1'b0};
         default: Dout = 17'bx;
      endcase
   end
endmodule
```

피승수 A와 Booth Encoder에서 구해진 Index값을 받아서 부분곱 Dout을 결정합니다.

- Carry Save Adder

```
module CSA1(X, Y, Z, Carry, Sum); //18bit
   input [17:0] X, Y, Z;
   output [17:0] Carry, Sum;

   FA fulladd0(.a(X[0]), .b(Y[0]), .c(Z[0]), .cout(Carry[0]), .sum(Sum[0]));
   FA fulladd1(.a(X[1]), .b(Y[1]), .c(Z[1]), .cout(Carry[1]), .sum(Sum[1]));
   FA fulladd2(.a(X[2]), .b(Y[2]), .c(Z[2]), .cout(Carry[2]), .sum(Sum[2]));
   FA fulladd3(.a(X[3]), .b(Y[3]), .c(Z[3]), .cout(Carry[3]), .sum(Sum[3]));
   FA fulladd4(.a(X[4]), .b(Y[4]), .c(Z[4]), .cout(Carry[4]), .sum(Sum[4]));
   FA fulladd5(.a(X[5]), .b(Y[5]), .c(Z[5]), .cout(Carry[5]), .sum(Sum[5]));
   FA fulladd6(.a(X[6]), .b(Y[6]), .c(Z[6]), .cout(Carry[6]), .sum(Sum[6]));
   FA fulladd7(.a(X[7]), .b(Y[7]), .c(Z[7]), .cout(Carry[7]), .sum(Sum[7]));
   FA fulladd8(.a(X[8]), .b(Y[8]), .c(Z[8]), .cout(Carry[8]), .sum(Sum[8]));
   FA fulladd9(.a(X[9]), .b(Y[9]), .c(Z[9]), .cout(Carry[9]), .sum(Sum[9]));
   FA fulladd10(.a(X[10]), .b(Y[10]), .c(Z[10]), .cout(Carry[10]), .sum(Sum[10]));
   FA fulladd11(.a(X[11]), .b(Y[11]), .c(Z[11]), .cout(Carry[11]), .sum(Sum[11]));
   FA fulladd12(.a(X[12]), .b(Y[12]), .c(Z[12]), .cout(Carry[12]), .sum(Sum[12]));
   FA fulladd13(.a(X[13]), .b(Y[13]), .c(Z[13]), .cout(Carry[13]), .sum(Sum[13]));
   FA fulladd14(.a(X[14]), .b(Y[14]), .c(Z[14]), .cout(Carry[14]), .sum(Sum[14]));
   FA fulladd15(.a(X[15]), .b(Y[15]), .c(Z[15]), .cout(Carry[15]), .sum(Sum[15]));
   FA fulladd16(.a(X[16]), .b(Y[16]), .c(Z[16]), .cout(Carry[16]), .sum(Sum[16]));
   FA fulladd17(.a(X[17]), .b(Y[17]), .c(Z[17]), .cout(Carry[17]), .sum(Sum[17]));

endmodule

module CSA2(X, Y, Carry, Sum); //17bit
   input [16:0] X, Y;
   output [16:0] Carry, Sum;

   HA halfadd0(.a(X[0]), .b(Y[0]), .cout(Carry[0]), .sum(Sum[0]));
   HA halfadd1(.a(X[1]), .b(Y[1]), .cout(Carry[1]), .sum(Sum[1]));
   HA halfadd2(.a(X[2]), .b(Y[2]), .cout(Carry[2]), .sum(Sum[2]));
   HA halfadd3(.a(X[3]), .b(Y[3]), .cout(Carry[3]), .sum(Sum[3]));
   HA halfadd4(.a(X[4]), .b(Y[4]), .cout(Carry[4]), .sum(Sum[4]));
   HA halfadd5(.a(X[5]), .b(Y[5]), .cout(Carry[5]), .sum(Sum[5]));
   HA halfadd6(.a(X[6]), .b(Y[6]), .cout(Carry[6]), .sum(Sum[6]));
   HA halfadd7(.a(X[7]), .b(Y[7]), .cout(Carry[7]), .sum(Sum[7]));
   HA halfadd8(.a(X[8]), .b(Y[8]), .cout(Carry[8]), .sum(Sum[8]));
   HA halfadd9(.a(X[9]), .b(Y[9]), .cout(Carry[9]), .sum(Sum[9]));
   HA halfadd10(.a(X[10]), .b(Y[10]), .cout(Carry[10]), .sum(Sum[10]));
   HA halfadd11(.a(X[11]), .b(Y[11]), .cout(Carry[11]), .sum(Sum[11]));
   HA halfadd12(.a(X[12]), .b(Y[12]), .cout(Carry[12]), .sum(Sum[12]));
   HA halfadd13(.a(X[13]), .b(Y[13]), .cout(Carry[13]), .sum(Sum[13]));
   HA halfadd14(.a(X[14]), .b(Y[14]), .cout(Carry[14]), .sum(Sum[14]));
   HA halfadd15(.a(X[15]), .b(Y[15]), .cout(Carry[15]), .sum(Sum[15]));
   HA halfadd16(.a(X[16]), .b(Y[16]), .cout(Carry[16]), .sum(Sum[16]));

endmodule
```

- Carry Propagation Adder

```
module CPA(X, Y, Cin, Cout, Sum, ov);
   input [31:0] X, Y;
   input Cin;

   output [30:0] Sum; //MSB is overflow bit
   output Cout;
   output ov;

   wire[30:0] Carry;


   FA fulladd0(.a(X[0]), .b(Y[0]), .c(Cin), .cout(Carry[0]), .sum(Sum[0]));
   FA fulladd1(.a(X[1]), .b(Y[1]), .c(Carry[0]), .cout(Carry[1]), .sum(Sum[1]));
   FA fulladd2(.a(X[2]), .b(Y[2]), .c(Carry[1]), .cout(Carry[2]), .sum(Sum[2]));
   FA fulladd3(.a(X[3]), .b(Y[3]), .c(Carry[2]), .cout(Carry[3]), .sum(Sum[3]));
   FA fulladd4(.a(X[4]), .b(Y[4]), .c(Carry[3]), .cout(Carry[4]), .sum(Sum[4]));
   FA fulladd5(.a(X[5]), .b(Y[5]), .c(Carry[4]), .cout(Carry[5]), .sum(Sum[5]));
   FA fulladd6(.a(X[6]), .b(Y[6]), .c(Carry[5]), .cout(Carry[6]), .sum(Sum[6]));
   FA fulladd7(.a(X[7]), .b(Y[7]), .c(Carry[6]), .cout(Carry[7]), .sum(Sum[7]));
   FA fulladd8(.a(X[8]), .b(Y[8]), .c(Carry[7]), .cout(Carry[8]), .sum(Sum[8]));
   FA fulladd9(.a(X[9]), .b(Y[9]), .c(Carry[8]), .cout(Carry[9]), .sum(Sum[9]));
   FA fulladd10(.a(X[10]), .b(Y[10]), .c(Carry[9]), .cout(Carry[10]), .sum(Sum[10]));
   FA fulladd11(.a(X[11]), .b(Y[11]), .c(Carry[10]), .cout(Carry[11]), .sum(Sum[11]));
   FA fulladd12(.a(X[12]), .b(Y[12]), .c(Carry[11]), .cout(Carry[12]), .sum(Sum[12]));
   FA fulladd13(.a(X[13]), .b(Y[13]), .c(Carry[12]), .cout(Carry[13]), .sum(Sum[13]));
   FA fulladd14(.a(X[14]), .b(Y[14]), .c(Carry[13]), .cout(Carry[14]), .sum(Sum[14]));
   FA fulladd15(.a(X[15]), .b(Y[15]), .c(Carry[14]), .cout(Carry[15]), .sum(Sum[15]));
   FA fulladd16(.a(X[16]), .b(Y[16]), .c(Carry[15]), .cout(Carry[16]), .sum(Sum[16]));
   FA fulladd17(.a(X[17]), .b(Y[17]), .c(Carry[16]), .cout(Carry[17]), .sum(Sum[17]));
   FA fulladd18(.a(X[18]), .b(Y[18]), .c(Carry[17]), .cout(Carry[18]), .sum(Sum[18]));
   FA fulladd19(.a(X[19]), .b(Y[19]), .c(Carry[18]), .cout(Carry[19]), .sum(Sum[19]));
   FA fulladd20(.a(X[20]), .b(Y[20]), .c(Carry[19]), .cout(Carry[20]), .sum(Sum[20]));
   FA fulladd21(.a(X[21]), .b(Y[21]), .c(Carry[20]), .cout(Carry[21]), .sum(Sum[21]));
   FA fulladd22(.a(X[22]), .b(Y[22]), .c(Carry[21]), .cout(Carry[22]), .sum(Sum[22]));
   FA fulladd23(.a(X[23]), .b(Y[23]), .c(Carry[22]), .cout(Carry[23]), .sum(Sum[23]));
   FA fulladd24(.a(X[24]), .b(Y[24]), .c(Carry[23]), .cout(Carry[24]), .sum(Sum[24]));
   FA fulladd25(.a(X[25]), .b(Y[25]), .c(Carry[24]), .cout(Carry[25]), .sum(Sum[25]));
   FA fulladd26(.a(X[26]), .b(Y[26]), .c(Carry[25]), .cout(Carry[26]), .sum(Sum[26]));
   FA fulladd27(.a(X[27]), .b(Y[27]), .c(Carry[26]), .cout(Carry[27]), .sum(Sum[27]));
   FA fulladd28(.a(X[28]), .b(Y[28]), .c(Carry[27]), .cout(Carry[28]), .sum(Sum[28]));
   FA fulladd29(.a(X[29]), .b(Y[29]), .c(Carry[28]), .cout(Carry[29]), .sum(Sum[29]));
   FA fulladd30(.a(X[30]), .b(Y[30]), .c(Carry[29]), .cout(Carry[30]), .sum(Sum[30]));
   FA fulladd31(.a(X[31]), .b(Y[31]), .c(Carry[30]), .cout(Cout), .sum(ov));

endmodule
```

- Booth Multiplier

```
module BoothMUL(A, B, m, ov); //m : output, ov : overflow bit
   input [15:0] A, B;
   output [30:0] m;
   output ov;

   //wire
   wire [7:0] Sb;
   wire [2:0] Index0, Index1, Index2, Index3, Index4, Index5, Index6, Index7;
   wire [16:0] pout0, pout1, pout2, pout3, pout4, pout5, pout6, pout7;
   wire [17:0] carry0, carry1, carry2, carry3, carry4, carry5, carry6, carry7;
   wire [17:0] sum0, sum1, sum2, sum3, sum4, sum5, sum6, sum7;
   wire cout;

   Booth_Index_Enc booth0(.B({B[1:0], 1'b0}), .Index(Index0), .Sign(Sb[0]));
   Booth_Index_Enc booth1(.B(B[3:1]), .Index(Index1), .Sign(Sb[1]));
   Booth_Index_Enc booth2(.B(B[5:3]), .Index(Index2), .Sign(Sb[2]));
   Booth_Index_Enc booth3(.B(B[7:5]), .Index(Index3), .Sign(Sb[3]));
   Booth_Index_Enc booth4(.B(B[9:7]), .Index(Index4), .Sign(Sb[4]));
   Booth_Index_Enc booth5(.B(B[11:9]), .Index(Index5), .Sign(Sb[5]));
   Booth_Index_Enc booth6(.B(B[13:11]), .Index(Index6), .Sign(Sb[6]));
   Booth_Index_Enc booth7(.B(B[15:13]), .Index(Index7), .Sign(Sb[7]));



   pp_Gen_MUX ppgen0(.A(A), .Index(Index0), .Dout(pout0));
   pp_Gen_MUX ppgen1(.A(A), .Index(Index1), .Dout(pout1));
   pp_Gen_MUX ppgen2(.A(A), .Index(Index2), .Dout(pout2));
   pp_Gen_MUX ppgen3(.A(A), .Index(Index3), .Dout(pout3));
   pp_Gen_MUX ppgen4(.A(A), .Index(Index4), .Dout(pout4));
   pp_Gen_MUX ppgen5(.A(A), .Index(Index5), .Dout(pout5));
   pp_Gen_MUX ppgen6(.A(A), .Index(Index6), .Dout(pout6));
   pp_Gen_MUX ppgen7(.A(A), .Index(Index7), .Dout(pout7));
```

```
    CSA1 csa0(
        .X({pout2[15:0], 2'b0}),
        .Y({1'b1, ~pout1[16], pout1[15:0]}),
        .Z({1'b0, ~pout0[16], pout0[16],pout0[16:2]}),
        .Carry(carry0),
        .Sum(sum0));

    CSA1 csa1(
        .X({pout3[15:0], 2'b0}),
        .Y({1'b1, ~pout2[16], sum0[17:2]}),
        .Z({1'b0, carry0[17:2], Sb[2]}),
        .Carry(carry1),
        .Sum(sum1));

    CSA1 csa2(
        .X({pout4[15:0], 2'b0}),
        .Y({1'b1, ~pout3[16], sum1[17:2]}),
        .Z({1'b0, carry1[17:2], Sb[3]}),
        .Carry(carry2),
        .Sum(sum2));

    CSA1 csa3(
        .X({pout5[15:0], 2'b0}),
        .Y({1'b1, ~pout4[16], sum2[17:2]}),
        .Z({1'b0, carry2[17:2], Sb[4]}),
        .Carry(carry3),
        .Sum(sum3));

    CSA1 csa4(
        .X({pout6[15:0], 2'b0}),
        .Y({1'b1, ~pout5[16], sum3[17:2]}),
        .Z({1'b0, carry3[17:2], Sb[5]}),
        .Carry(carry4),
        .Sum(sum4));

    CSA1 csa5(
        .X({pout7[15:0], 2'b0}),
        .Y({1'b1, ~pout6[16], sum4[17:2]}),
        .Z({1'b0, carry4[17:2], Sb[6]}),
        .Carry(carry5),
        .Sum(sum5));


    //CSA half adder
    CSA2 csa6(
        .X({~pout7[16], sum5[17:2]}),
        .Y({carry5[17:2], Sb[7]}),
        .Carry(carry6),
        .Sum(sum6));

    //CPA(32bit)
    CPA cpa0(
        .X({1'b0,sum6[16:0],sum5[1:0],sum4[1:0],sum3[1:0],sum2[1:0] ,sum1[1:0],sum0[1:0],pout0[1:0]}),
        .Y({carry6[16:0],carry5[1:0],carry4[1:0],carry3[1:0],carry2[1:0] ,carry1[1:0],carry0[1:0],Sb[1],1'b0,Sb[0]}),
        .Cin(1'b0),
        .Cout(cout),
        .Sum(m),
        .ov(ov));

endmodule
```

Booth Multiplier는 16비트 입력 A, B를 받아 곱셈 결과값 m과 오버플로우 비트 ov를 출력합니다.

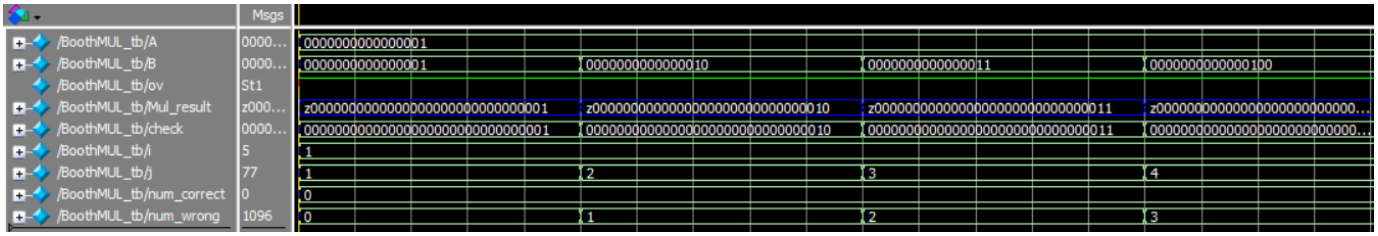
우선 Booth Encoder 에서 만든 Index와 A를 PP Generator에 넣어 줍니다. PP Generator는 pout을 결과로 출력합니다.

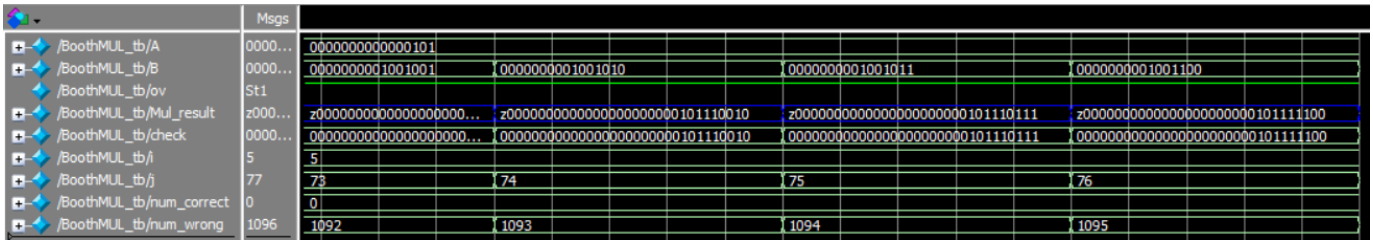
Pout과 Sign bit를 적절히 합해주면서 결과를 얻어냅니다.

- Booth Multiplier testbench

```
`timescale 1ns/100ps

module BoothMUL_tb;
   reg [15:0] A, B;

   wire ov;
   wire [31:0] Mul_result;

   reg [31:0] check;
   integer i, j;
   integer num_correct, num_wrong;

   BoothMUL booth0(.A(A), .B(B), .m(Mul_result), .ov(ov));

   initial begin
      num_correct = 0;
      num_wrong = 0;

      $display("Done");
      for(i=1; i<256; i=i+1) begin
         A=i;
         for(j=1; j<256; j=j+1) begin
            B=j;
            check = $signed(A)*$signed(B);

            #10
            if({ov, Mul_result} == check)
               num_correct = num_correct+1;
            else
               num_wrong = num_wrong+1;
         end
      end

      $display("num_correct=%d, num_wrong=%d", num_correct, num_wrong);
   end
endmodule
```

직접 구한 결과와 곱셈기가 계산한 결과가 같다면 num_correct를, 다르다면 num_wrong을 1씩 증가하도록 했습니다. 이때, check값은 signed integer를 곱하는 것이기 때문에, $signed(A)*$signed(B)를 해줍니다.





곱셈기로 계산한 값({ov, Mul_result}) 의 ov값 부분이 비정상적으로(High-Z) 나오는 것을 발견하였습니다. 이로 인해, 곱셈 연산 자체는 정상적으로 수행되나, num_wrong이 올라가는 것을 확인하였습니다.

4. Evaluation

설계한 곱셈기의 곱셈 결과값은 제대로 나오지만, overflow bit 부분에 High-Z 값이 나오는 현상이 있었습니다.

5. Discussion

이번 실습의 key는 Booth Encoding 과정을 통해 연속적인 1 값의 개수(곱셈 연산의 횟수)를 줄여주는 것입니다.

흥미로웠던 점은 연속적인 1 값들을 Booth Encoding을 통해 획기적으로 줄일 수 있다는 점이었습니다.

이번 과제를 하면서 실수했던 점은, overflow bit 값을 제대로 출력해내지 못했다는 점입니다.

이 design을 향상시키려면, overflow bit 값이 제대로 출력되도록 고쳐야 합니다.