```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 08/28/2016 01:49:26 AM
// Design Name:
// Module Name: KOA_2
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module KOA_2    //#(parameter SW = 24)
//#(parameter SW = 54)
#(parameter SW = 12)
(
input wire clk,
input wire rst,
input wire load_b_i,
input wire [SW-1:0] Data_A_i,
input wire [SW-1:0] Data_B_i,
output wire [2*SW-1:0] sgf_result_o
);

   //wire [SW-1:0] Data_A_i;
    //wire [SW-1:0] Data_B_i;


    //wire [2*(SW/2)-1:0] result_left_mult;
    //wire [2*(SW/2+1)-1:0] result_right_mult;
    wire [SW/2+1:0] result_A_adder;
    //wire [SW/2+1:0] Q_result_A_adder;
    wire [SW/2+1:0] result_B_adder;
    //wire [SW/2+1:0] Q_result_B_adder;
    //wire [2*(SW/2+2)-1:0] result_middle_mult;

    wire [SW-1:0] Q_left;
    wire [SW+1:0] Q_right;
    wire [SW+3:0] Q_middle; ///Modificación J: Le he agregado dos bits al largo del puerto, para acomodar los 2 cero el resultado de una multiplicacion

    wire [2*(SW/2+2)-1:0] S_A;
    wire [2*(SW/2+2)-1:0] S_B;

    wire [4*(SW/2)+2:0] Result;
    ///////////////////////////////////////////////////////////
    wire [1:0] zero1;
    wire [3:0] zero2;
    assign zero1 =2'b00;
    assign zero2 =4'b0000;
    ///////////////////////////////////////////////////////////
    wire [SW/2-1:0] rightside1;
    wire [SW/2-3:0] leftside1;
    wire [SW/2:0] rightside2;

    wire [4*(SW/2)-1:0] sgf_r;

    assign rightside1 = (SW/2) *1'b0;
    assign rightside2 = (SW/2+1)*1'b0;
    assign leftside1 = (SW/2-2) *1'b0;

    localparam half = SW/2;
    localparam full_port = SW - 1;
    //localparam level1=4;
    //localparam level2=5;

    ////////////////////////////////////
//generate
//    case (SW%2)
//        0:begin
        /////////////////////////////////even////////////////////////////////
       //Multiplier for left side and right side

            KOA_1 #(.SW(SW/2)/*,.level(level1)*/) left(
                .Data_A_i(Data_A_i[full_port:half]/*P=SW/2*/),
                .Data_B_i(Data_B_i[full_port:half]/*P=SW/2*/),
                .sgf_result_o(Q_left) /*P=SW*//*result_left_mult*/
            );


            KOA_1 #(.SW(SW/2)) right(                     /*,.level(level1)*/
                .Data_A_i(Data_A_i[half-1:0]/*P=SW/2*/),
                .Data_B_i(Data_B_i[half-1:0]/*P=SW/2*/),
                .sgf_result_o(Q_right[full_port:0]/*P=SW*/)
                /*result_right_mult[2*(SW/2)-1:0]*/
            );

           //Adders for middle

            adder #(.W(SW/2)) A_operation (
                .Data_A_i(Data_A_i[SW-1:SW-SW/2]/*P=SW/2*/),
                .Data_B_i(Data_A_i[SW-SW/2-1:0]/*P=SW/2*/),
                .Data_S_o(result_A_adder[SW/2:0]/*P=SW/2+1*/)
            );

            adder #(.W(SW/2)) B_operation (
                .Data_A_i(Data_B_i[SW-1:SW/2]/*P=SW/2*/),
                .Data_B_i(Data_B_i[SW/2-1:0]/*P=SW/2*/),
                .Data_S_o(result_B_adder[SW/2:0]/*P=SW/2+1*/)
            );

           //multiplication for middle
        //Introducimos un par de ceros, ya que esta multiplicacion es siempre impar, gracias al sumador que viene detras.
        //Modificación: Le agregué otro bit al puerto de este multiplicador, para que fuera par.
            KOA_1 #(.SW(SW/2+2)/* Port length = SW/2 + 2*/) middle (

                .Data_A_i({1'b0 /*P=1*/,result_A_adder[SW/2:0]/*P=SW/2+1*/}), /*Q_result_A_adder[SW/2+1:0]*/
                .Data_B_i({1'b0 /*P=1*/,result_B_adder[SW/2:0]/*P=SW/2+1*/}), /*Q_result_B_adder[SW/2+1:0]*/
                .sgf_result_o(Q_middle[SW+3:0]) /*result_middle_mult[2*(SW/2)+2:0]*/

                //Vamos a truncar este resultado en la siguiente etapa del puerto
            );



//            wire [SW-1:0] Q_left;
//            wire [SW+1:0] Q_right;
//            wire [SW+3:0] Q_middle;

           ///Subtractors for middle

            substractor #(.W(SW+2)) Subtr_1 (
                .Data_A_i(Q_middle[SW+1:0]  /*P=SW+2*/), /*result_middle_mult//*/
                .Data_B_i({zero1/*P=2*/, Q_left /*P=SW*/}), /*result_left_mult//*/
                .Data_S_o(S_A[SW+1:0]/*P=SW+2*/)
            );

            substractor #(.W(SW+2)) Subtr_2 (
                .Data_A_i(S_A[2*(SW/2)+1:0] /*P=SW+2*/),
                .Data_B_i({zero1 /*P=2*/, Q_right[SW-1:0] /*P=SW*/}), /*result_right_mult//*/
                .Data_S_o(S_B[2*(SW/2)+1:0])  //Port width is SW+1
            );


        //wire [SW-1:0] Q_left; /*P=SW*/
        //wire [SW+1:0] Q_right; /*P=SW+2*/

           //Final adder
            adder #(.W(2*SW)) Final(
                .Data_A_i({Q_left /*P=SW*/ ,Q_right[SW-1:0]/*P=SW*/}),   //Port width is 2*SW   /*result_left_mult,result_right_mult*/
```

```
                  .Data_B_i({leftside1,S_B[2*(SW/2)+1:0],rightside1}),
                  .Data_S_o(Result[4*(SW/2):0]/*P=2*SW+1*/)
              );

             //Final Register

              RegisterAdd #(.W(2*SW)) finalreg ( //Data X input register
                  .clk(clk),
                  .rst(rst),
                  .load(load_b_i),
                  .D(Result[4*(SW/2)-1:0]),
                  .Q({sgf_result_o})
              );

//          end
//      1:begin
//          ////////////////////////////////odd//////////////////////////////////
//          //Multiplier for left side and right side

//             KOA_1 #(.SW(SW/2)/*,.level(level2)*/) left_high(

//                          .Data_A_i(Data_A_i[SW-1:SW/2]),
//                          .Data_B_i(Data_B_i[SW-1:SW/2]),
//                          .sgf_result_o(/*result_left_mult*/Q_left)
//              );

//              /*RegisterAdd #(.W(2*(SW/2))) leftreg( //Data X input register
//                  .clk(clk),
//                  .rst(rst),
//                  .load(1'b1),
//                  .D(result_left_mult),
//                  .Q(Q_left)
//              );//*/

//              KOA_1 #(.SW((SW/2)+1)/*,.level(level2)*/) right_lower(

//                  .Data_A_i(Data_A_i[SW/2-1:0]),
//                  .Data_B_i(Data_B_i[SW/2-1:0]),
//                  .sgf_result_o(/*result_right_mult*/Q_right)
//              );

//              /*RegisterAdd #(.W(2*((SW/2)+1))) rightreg( //Data X input register
//                  .clk(clk),
//                  .rst(rst),
//                  .load(1'b1),
//                  .D(result_right_mult),
//                  .Q(Q_right)
//              );//*/

//             //Adders for middle

//              adder #(.W(SW/2+1)) A_operation (
//                  .Data_A_i({1'b0,Data_A_i[SW-1:SW-SW/2]}),
//                .Data_B_i(Data_A_i[SW-SW/2-1:0]),
//                .Data_S_o(result_A_adder)
//            );

//            adder #(.W(SW/2+1)) B_operation (
//                .Data_A_i({1'b0,Data_B_i[SW-1:SW-SW/2]}),
//                .Data_B_i(Data_B_i[SW-SW/2-1:0]),
//                .Data_S_o(result_B_adder)
//            );

//            //segmentation registers for 64 bits

//            /*RegisterAdd #(.W(SW/2+2)) preAreg ( //Data X input register
//                            .clk(clk),
//                            .rst(rst),
//                            .load(1'b1),
//                            .D(result_A_adder),
//                            .Q(Q_result_A_adder)
//                        );//

//            RegisterAdd #(.W(SW/2+2)) preBreg ( //Data X input register
//                            .clk(clk),
//                            .rst(rst),
//                            .load(1'b1),
//                            .D(result_B_adder),
//                            .Q(Q_result_B_adder)
//                        );//*/
//           //multiplication for middle

//            KOA_1 #(.SW(SW/2+2)/*,.level(level2)*/) middle (

//                .Data_A_i(/*Q_result_A_adder*/result_A_adder),
//                .Data_B_i(/*Q_result_B_adder*/result_B_adder),
//                .sgf_result_o(/*result_middle_mult*/Q_middle)
//            );

//           //segmentation registers array


//            /*RegisterAdd #(.W(2*((SW/2)+2))) midreg ( //Data X input register
//                .clk(clk),
//                .rst(rst),
//                .load(1'b1),
//                .D(result_middle_mult),
//                .Q(Q_middle)
//            );//*/

//           ///Subtractors for middle

//            substractor #(.W(2*(SW/2+2))) Subtr_1 (
//                .Data_A_i(/*result_middle_mult//*/Q_middle),
//                .Data_B_i({zero2, /*result_left_mult//*/Q_left}),
//                .Data_S_o(S_A)
//            );

//            substractor #(.W(2*(SW/2+2))) Subtr_2 (
//                .Data_A_i(S_A),
//                .Data_B_i({zero1, /*result_right_mult//*/Q_right}),
//                .Data_S_o(S_B)
//            );

//           //Final adder

//            adder #(.W(4*(SW/2)+2)) Final(
//                .Data_A_i({/*result_left_mult,result_right_mult*/Q_left,Q_right}),
//                .Data_B_i({S_B,rightside2}),
//                .Data_S_o(Result[4*(SW/2)+2:0])
//            );

//           //Final Register



//            RegisterAdd #(.W(4*(SW/2)+2)) finalreg ( //Data X input register
//                .clk(clk),
//                .rst(rst),
//                .load(load_b_i),
//                .D(Result[2*SW-1:0]),
//                .Q({sgf_result_o})
//            );
//        end
//    endcase
//endgenerate




endmodule

```