# Brodric Young
# ECEN 340
# Lab #3: 8-bit Adder

**Purpose:**

1. To reinforce structural Verilog coding style (figures 3.18 and 3.19 of the textbook).
2. To gain exposure to the Verilog Data Flow programming style (continuous assignment statements found in figures 3.20 and 3.21 of the textbook).
3. To learn how to instantiate modules into a higher-level module.

**Procedure:**

**Part 1 (first lab day).** Create a Vivado project called "adder8" (or something similar) as you did in the Lab 0 tutorial. Implement a full adder module called "fulladd" as shown in one of the figures, 3.18 – 3.21 of the textbook. Figures 3.18 and 3.19 of the textbook implement the single-bit full adder using the familiar structural form. Figures 3.20 and 3.21 implement the exact same circuit using a continuous assignment statement. Depending on your previous experience and understanding, pick one of these styles to implement your single bit adder.

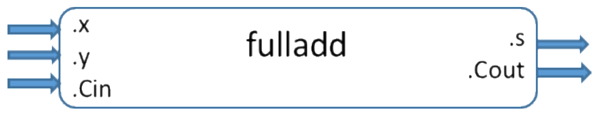


Figure 1 – Diagram representing a single full-adder with input ports on the left and output port on the right.

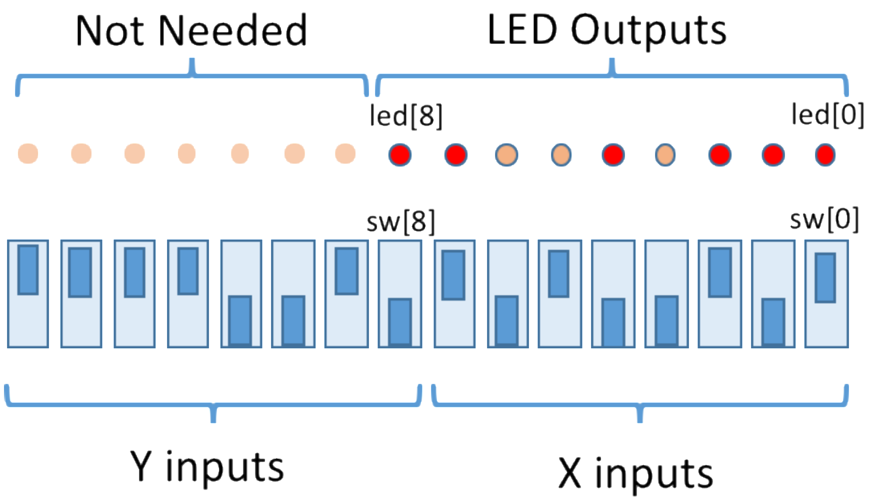
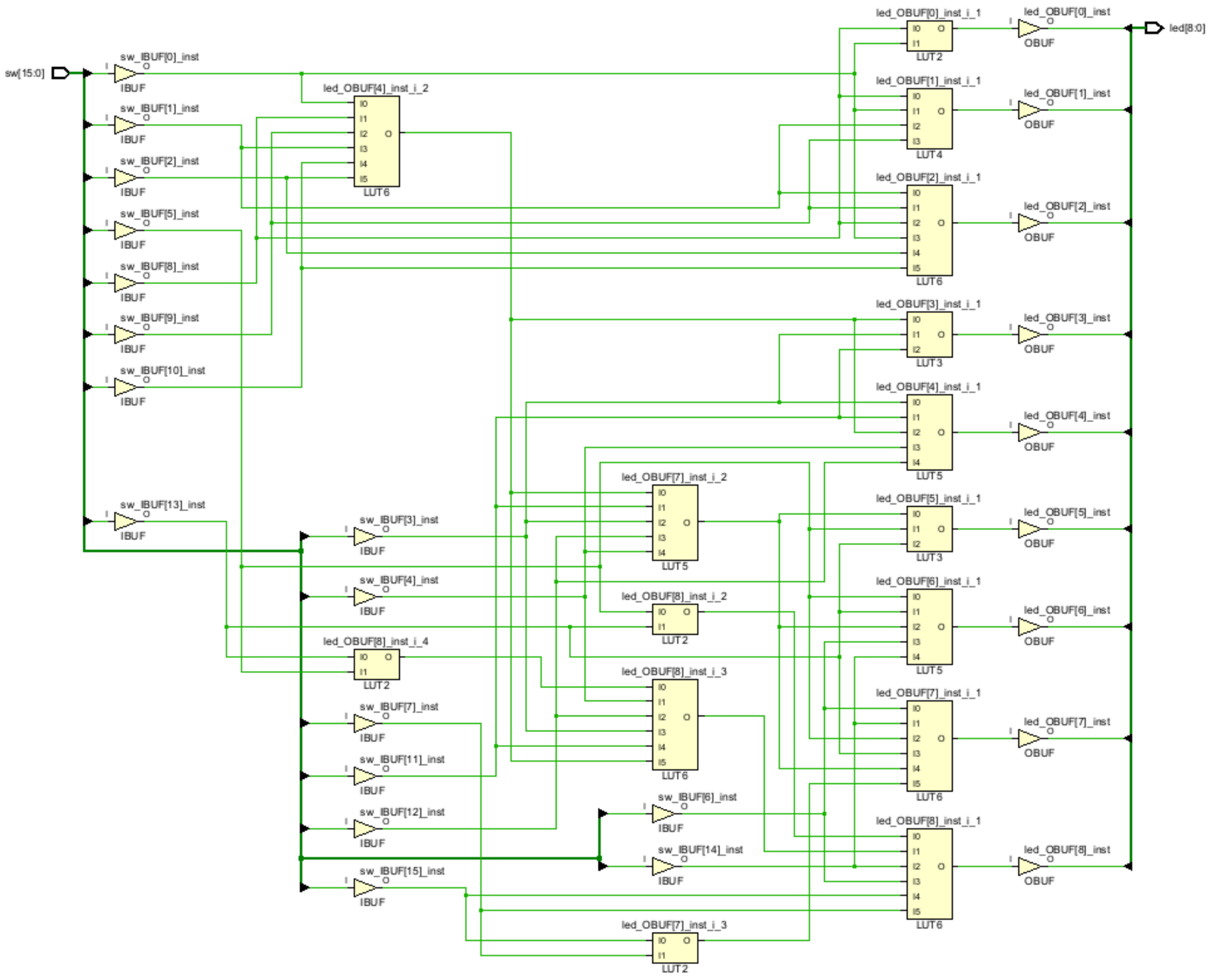


Figure 2 – Diagram representing the input switches and the output LEDs.

<u>Code and schematic for part 1:</u>

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Brodric Young & Karl Richards
//
// Create Date: 01/21/2025 01:55:07 PM
// Design Name:
// Module Name: adder8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module adder8(
    input [15:0] sw,
    output [7:0] led
    );

    wire [7:0] x;
    wire [7:0] y;
    wire [7:0] sum;
    wire [7:0] carry;
    wire cout;

    // right half of switches are x input, left half are y input
    assign x = sw[7:0];
    assign y = sw[15:8];

    // full adder for each of the 8 bits
    fullAdder FA0 (.A(x[0]), .B(y[0]), .Cin(0), .sum(sum[0]), .Cout(carry[0]));
    fullAdder FA1 (.A(x[1]), .B(y[1]), .Cin(carry[0]), .sum(sum[1]), .Cout(carry[1]));
    fullAdder FA2 (.A(x[2]), .B(y[2]), .Cin(carry[1]), .sum(sum[2]), .Cout(carry[2]));
    fullAdder FA3 (.A(x[3]), .B(y[3]), .Cin(carry[2]), .sum(sum[3]), .Cout(carry[3]));
    fullAdder FA4 (.A(x[4]), .B(y[4]), .Cin(carry[3]), .sum(sum[4]), .Cout(carry[4]));
    fullAdder FA5 (.A(x[5]), .B(y[5]), .Cin(carry[4]), .sum(sum[5]), .Cout(carry[5]));
    fullAdder FA6 (.A(x[6]), .B(y[6]), .Cin(carry[5]), .sum(sum[6]), .Cout(carry[6]));
    fullAdder FA7 (.A(x[7]), .B(y[7]), .Cin(carry[6]), .sum(sum[7]), .Cout(cout));

    // combines cout and sum to display the correct leds
    assign led = {cout, sum};
endmodule
module halfAdder(
    input A, B,
    output S1, S0
);
    xor (S1, A, B);
    and (S0, A, B);
endmodule


module fullAdder(
    input A, B, Cin,
    output sum, Cout
    );

    wire sum1, carry1, carry2;

    halfAdder HA1(
        .A(A),
        .B(B),
        .S1(sum1),
        .S0 (carry1)
    );

    halfAdder HA2(
        .A(sum1),
        .B(Cin),
        .S1(sum),
        .S0(carry2)
    );

    or (Cout, carry1, carry2);
 endmodule
```


sw[15:0]
led[8:0]
sw_IBUF[0]_inst
IBUF
sw_IBUF[1]_inst
IBUF
sw_IBUF[2]_inst
IBUF
sw_IBUF[5]_inst
IBUF
sw_IBUF[8]_inst
IBUF
sw_IBUF[9]_inst
IBUF
sw_IBUF[10]_inst
IBUF
sw_IBUF[13]_inst
IBUF
led_OBUF[4]_inst_i_2
LUT6
sw_IBUF[3]_inst
IBUF
sw_IBUF[4]_inst
IBUF
led_OBUF[8]_inst_i_4
LUT2
sw_IBUF[7]_inst
IBUF
sw_IBUF[11]_inst
IBUF
sw_IBUF[12]_inst
IBUF
sw_IBUF[15]_inst
IBUF
led_OBUF[7]_inst_i_2
LUT5
led_OBUF[8]_inst_i_2
LUT2
led_OBUF[8]_inst_i_3
LUT6
sw_IBUF[6]_inst
IBUF
sw_IBUF[14]_inst
IBUF
led_OBUF[7]_inst_i_3
LUT2
led_OBUF[0]_inst_i_1
LUT2
led_OBUF[1]_inst_i_1
LUT4
led_OBUF[2]_inst_i_1
LUT6
led_OBUF[3]_inst_i_1
LUT3
led_OBUF[4]_inst_i_1
LUT5
led_OBUF[5]_inst_i_1
LUT3
led_OBUF[6]_inst_i_1
LUT5
led_OBUF[7]_inst_i_1
LUT6
led_OBUF[8]_inst_i_1
LUT6
led_OBUF[0]_inst
OBUF
led_OBUF[1]_inst
OBUF
led_OBUF[2]_inst
OBUF
led_OBUF[3]_inst
OBUF
led_OBUF[4]_inst
OBUF
led_OBUF[5]_inst
OBUF
led_OBUF[6]_inst
OBUF
led_OBUF[7]_inst
OBUF
led_OBUF[8]_inst
OBUF

**Part 2 (second lab day).** Implement an 8-bit adder module call "adder8". Instantiate the full adder circuit 8 times to create an 8-bit adder using a style like that used in Figure 3.23 of the textbook.

20% Extra Credit: Instantiate the "fulladd" modules using Verilog's generate capability.

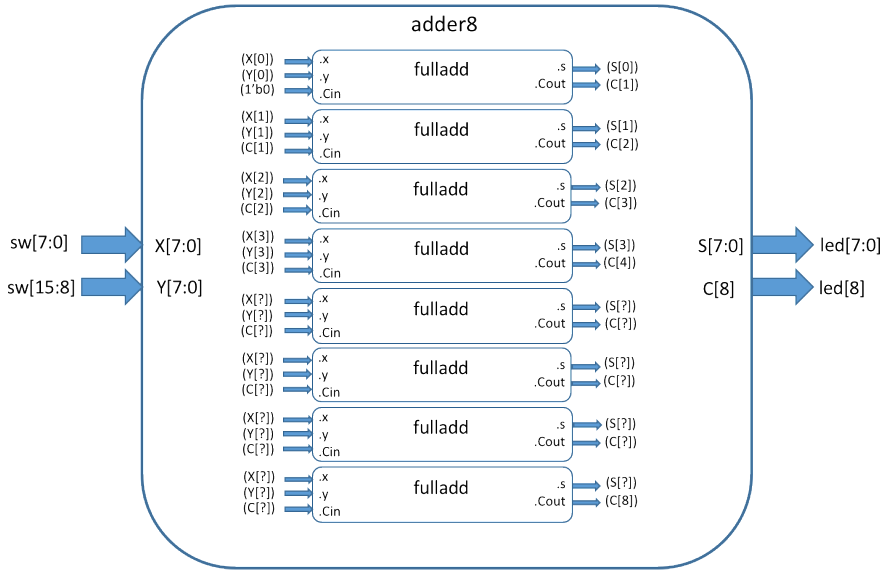


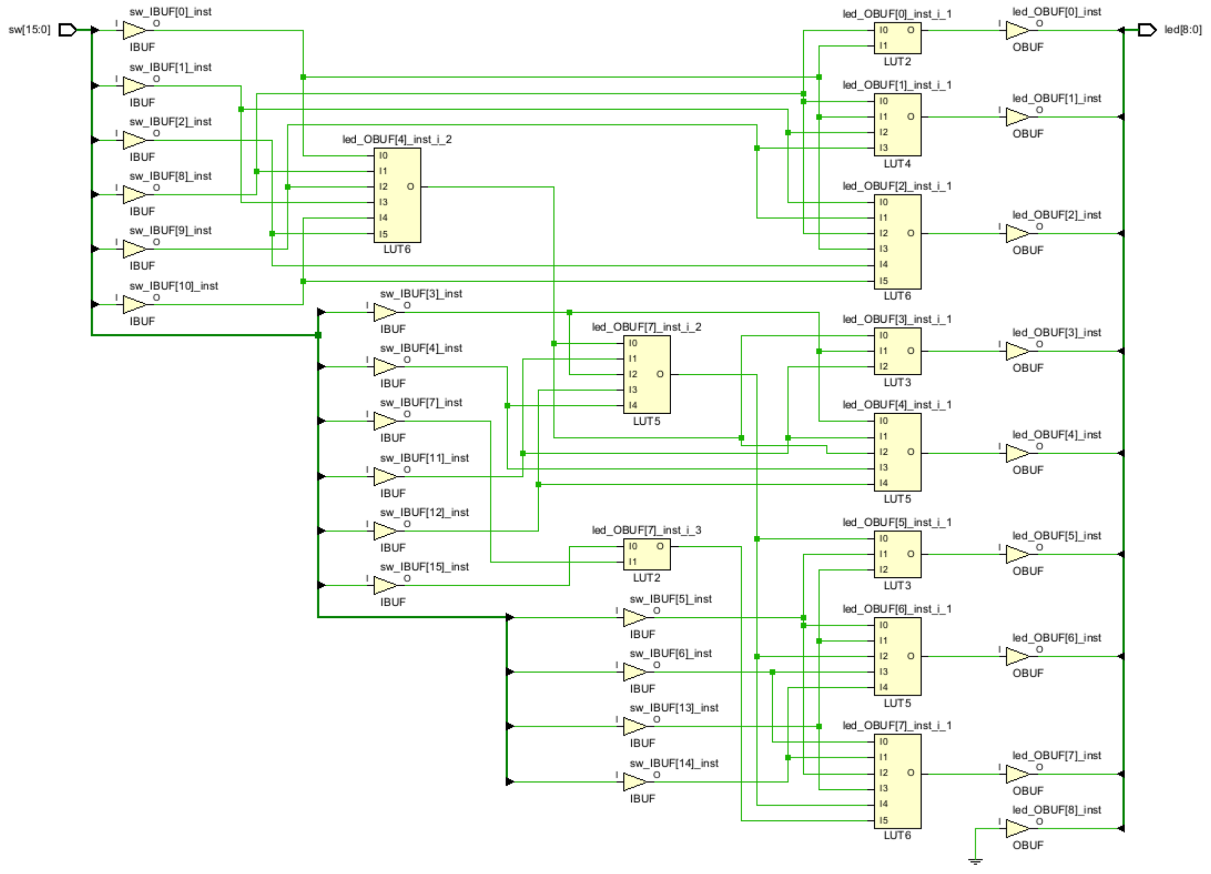
Figure 3 – Diagram representing the completed 8-bit adder circuit with "sw" input ports and "led" output ports.

<u>Code and schematic for part 2:</u>

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Brodric Young & Karl Richards
//
// Create Date: 01/21/2025 01:55:07 PM
// Design Name:
// Module Name: 8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
////////////////////////////////////////////////////////////////////////////////
module adder8(
   input [15:0] sw,
   output [8:0] led
   );

   wire [7:0] x;
   wire [7:0] y;
   wire [7:0] sum;
   wire [7:0] carry;
   wire cout;

   // right half of switches are x input, left half are y input
   assign x = sw[7:0];
   assign y = sw[15:8];

   assign carry[0] = 0;  // initial carry in is 0

   // using generate to loop through the fulladder module 8 times for an 8 bit fulladder
   genvar i;
   generate
      for(i=0; i<=7; i=i+1)
         begin
            fullAdder FA_i (.A(x[i]),
                            .B(y[i]),
                            .Cin(carry[i]),
                            .sum(sum[i]),
                            .Cout(carry[i+1])
                            );
         end
   endgenerate

   assign cout = carry[8];  // the final carry out
   assign led = {cout, sum};  // Combine the carry-out and sum into a single output for LEDs.
endmodule


module half_Adder(
   input A, B,
   output S1, S0
);
   xor (S1, A, B);
   and (S0, A, B);
endmodule
```

```
module fullAdder(
    input A, B, Cin,
    output sum, Cout
  );

  wire sum1, carry1, carry2;

  half_Adder HA1(
    .A(A),
    .B(B),
    .S1(sum1),
    .S0 (carry1)
  );

  half_Adder HA2(
    .A(sum1),
    .B(Cin),
    .S1(sum),
    .S0(carry2)
  );
  or (Cout, carry1, carry2);
endmodule
```



## Conclusion:

In this lab, we used Vivado to write verilog code that would implement an 8-bit adder on the FPGA. In part 1, we started off creating a half-adder module and instantiating it twice inside a full-adder module to create a full-adder. In part 2, we created an 8-bit full-adder module in which we took the full-adder module and instantiated it 8 times to create the 8-bit full adder. For extra credit we also modified this 8-bit full-adder module to use verilog's generate statement which allowed us to use a for loop to instantiate the full-adder module 8 times rather than hard coding it 8 times. After completing the 8-bit full-adder module both in part 2 and for the extra credit we tested it by using the right-most 8 switches as one 8-bit number and the left-most 8 bits as another 8-bit number and verified that the correct LED lit up for what the result of the addition of the two numbers should be. After some troubleshooting and correcting pieces of code, it all worked as we expected, 100% functional.