1) A copy of the new Verilog code you used to implement the design. The Verilog code should be complete with useful comments (5 pts).

```
- `timescale 1ns / 1ps
- //////////////////////////////////////////////////////////////////////////////////
- // Company:
- // Engineer:
- //
- // Create Date: 09/29/2025 09:49:06 AM
- // 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, //switch inputs
-     output [8:0] led, //led outputs
-     input btnC //instantiate center button
-     );
-
-     wire [7:0]a = sw[7:0]; //split up the switches into two sets
-     wire [7:0]b = sw[15:8];
-     wire [7:1]c; //carry over
-
-     fulladd stage0 (btnC, a[0], b[0], led[0], c[1]); //addd the least sig bits out put to led if needed and set c to something
-     fulladd stage1 (c[1], a[1], b[1], led[1], c[2]); //the rest of the lines do the same as the one above
-     fulladd stage2 (c[2], a[2], b[2], led[2], c[3]);
-     fulladd stage3 (c[3], a[3], b[3], led[3], c[4]);
-     fulladd stage4 (c[4], a[4], b[4], led[4], c[5]);
-     fulladd stage5 (c[5], a[5], b[5], led[5], c[6]);
-     fulladd stage6 (c[6], a[6], b[6], led[6], c[7]);
-     fulladd stage7 (c[7], a[7], b[7], led[7], led[8]);
```

```
-
-   endmodule
-
-   module fulladd(
-       input Cin, x, y, //inputs carry, one bit and another bit to add
-       output s, Cout //should we turn on the led, is there a carry out
-       );
-
-       xor (s, x, y, Cin); //make the logic
-       and (z1, x, y),
-           (z2, x, Cin),
-           (z3, y, Cin);
-       or (Cout, z1, z2, z3);
-   Endmodule
```

Extra Credit Code - I removed the button this time around

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/29/2025 09:49:06 AM
// 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, //switch inputs
    output [8:0] led //led outputs
```

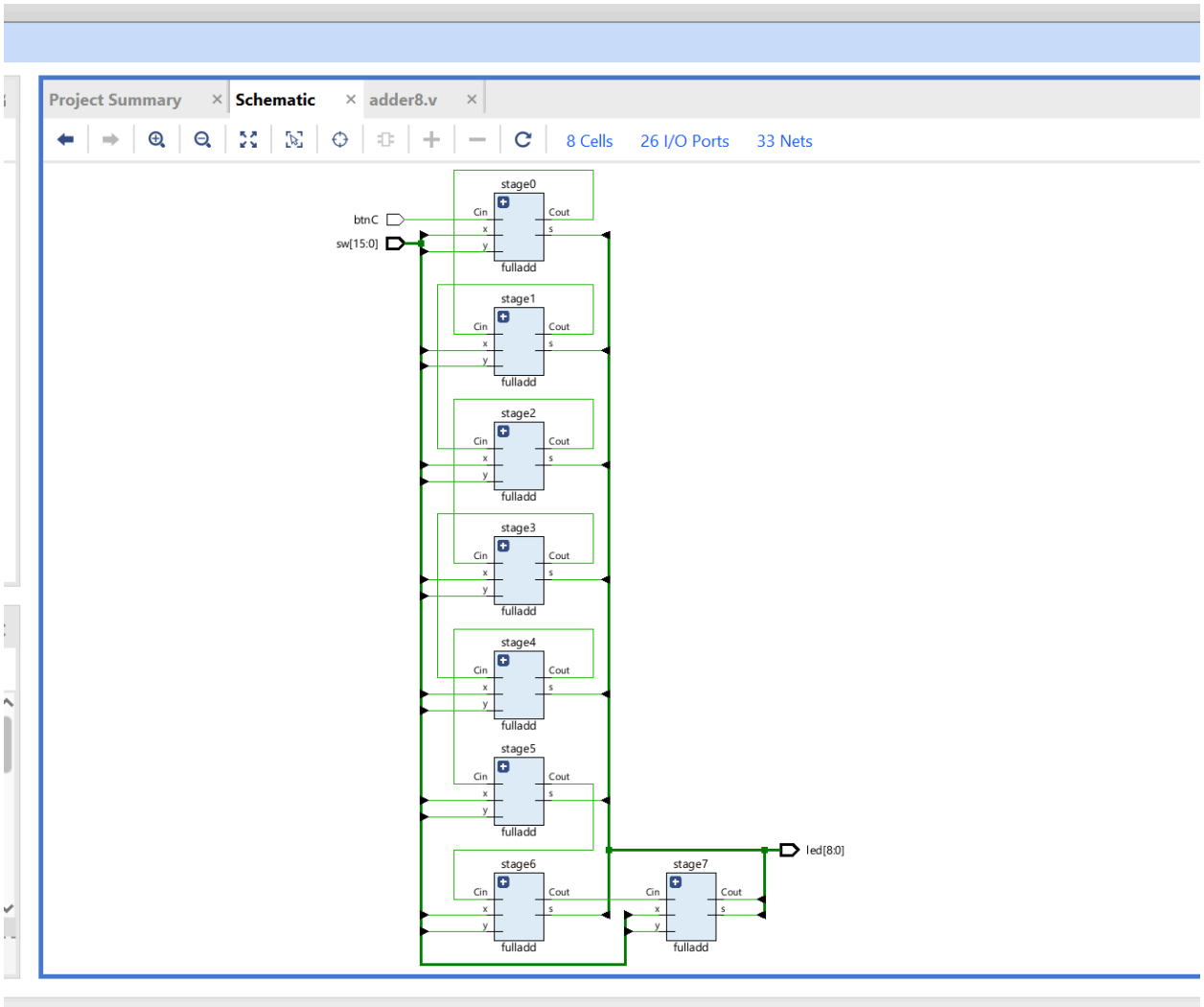
```
    );

    wire [7:0]a = sw[7:0]; //split up the switches into two sets
    wire [7:0]b = sw[15:8];
    wire [7:1]c; //carry over

    generate //make a block of code that could create structures
        genvar i;
        for (i = 0; i < 8; i = i + 1) begin : ripple_carry_stage //we have two sets of 8 switchs so loop
8 times. Give it a name so we can access the created structures
            fulladd stage ( //call the fulladd module and while specifying the variable to change give
them a value
                .Cin(c[i]),
                .x(a[i]),
                .y(b[i]),
                .s(led[i]),
                .Cout(c[i+1])
            );
        end
    endgenerate

    assign led[8] = c[8]; // we can't do this in the loop so we must assign the final carryout to a
carry out bit LED

endmodule

module fulladd(
    input Cin, x, y, //inputs carry, one bit and another bit to add
    output s, Cout //should we turn on the led, is there a carry out
    );

    xor (s, x, y, Cin); //make the logic
    and (z1, x, y),
        (z2, x, Cin),
        (z3, y, Cin);
    or (Cout, z1, z2, z3);
endmodule
```

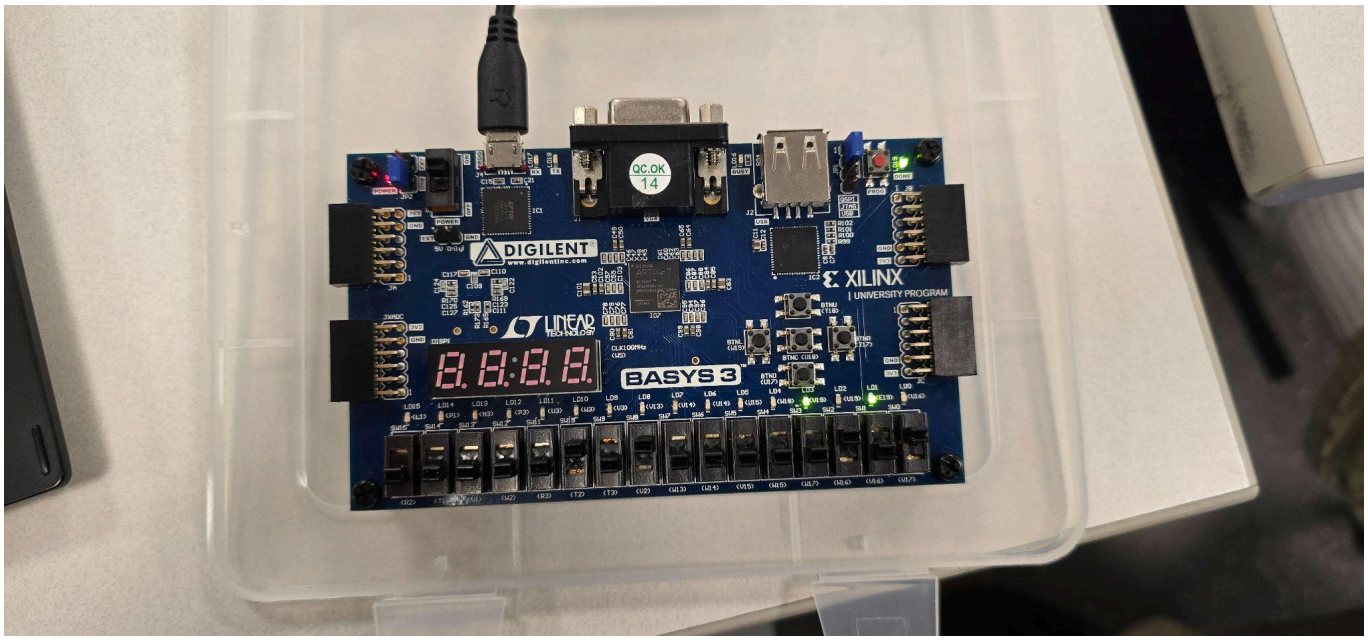
2) A brief description of the unique (new) tools, technologies, or methods used to implement this lab (5 pts).

- Compared to the last lab there wasn't a lot of new things that we used. The main thing we implemented was a 8 bit adder. And I used similar things as the last lab like making and and or gates. I did instantiate a button to determine if there was an initial carry-in which was new. I also used wires in order to split up the 16 switch input so there could be two sets of numbers.

- Extra Credit. I learned how to use the generate block and generate variables. At first it didn't make sense why I had to use it. After all couldn't I just use a for loop but doing that way wouldn't create blocks it would just do behavioral or testing things no instantiating. I also learned how you need to name the loop so you can access the fulladd instantiated modules. I also learned that the best way to set the inputs when calling a module is to access the variables using a period then in parenthesis add the value you want to assign to it.

3) Images of implemented schematic or other useful images.


Project Summary
Schematic
adder8.v
8 Cells
26 I/O Ports
33 Nets
btnC
sw[15:0]
led[8:0]
stage0
stage1
stage2
stage3
stage4
stage5
stage6
stage7
Cin
Cout
x
y
s
fulladd


DIGILENT
XILINX
UNIVERSITY PROGRAM
BASYS 3

4) The report should be professional quality—meaning it will be neat and use proper English (5 pts).

5) In your conclusion statement, discuss your results, the method of testing, and include the level of functionality of the lab (10 pts).

- My results were positive. I was successfully able to complete the code and get my program to run correctly on the board. I was able to click a button to determine whether or not there was an initial carry-in value. I was also able to have two sets of switches that controlled the bit count in each set to which my program would then calculate the addition of both sets and display that using LEDS. I ran into several errors only to realize that I hadn't enabled my clock and had some logic issues so the main tests I used were those I had to go through in order to run a bitstream like synthesizing.

- Extra credit:
    - Once again my results were positive. I was able to get the same result using a generate block which was super handy because it demonstrated to me that I didn't have to write each thing by hand. It was also very nice because now it is very easy to modify it because I would just have to change one or two things rather than lines and lines of code which makes it much more handy to change or grow.