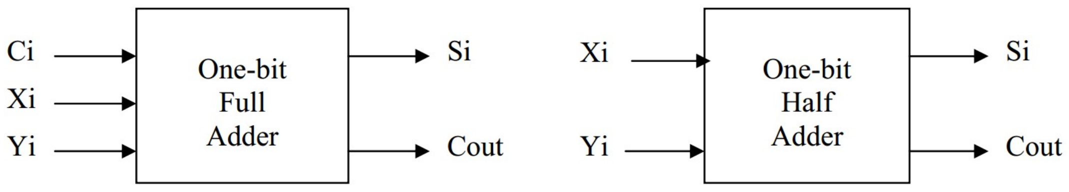
Justin VanWinkle
EEE 3342C
Experiment #4
1 July 2015

Objective

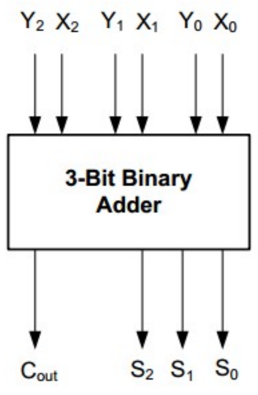
This experiment is intended to create and analyze a parallel binary adder gate using the Xilinx ISE design tools. The logic gate will be analyzed using the Xilinx ISE simulation tool as well as on a BASYS 2 FPGA development board.

Diagrams

The following is a block diagram that shows an over-arching view of the half and full adders required to build a parallel adder for this experiment.



A parallel adder is represented by the following:


Y_2 X_2 Y_1 X_1 Y_0 X_0
3-Bit Binary Adder
C_{out} S_2 S_1 S_0

Equipment

The equipment used for this experiment included a computer with the Xilinx ISE design suite and Digilent ExPort installed as well as a Digilent BAYSYS 2 development board.

Procedure

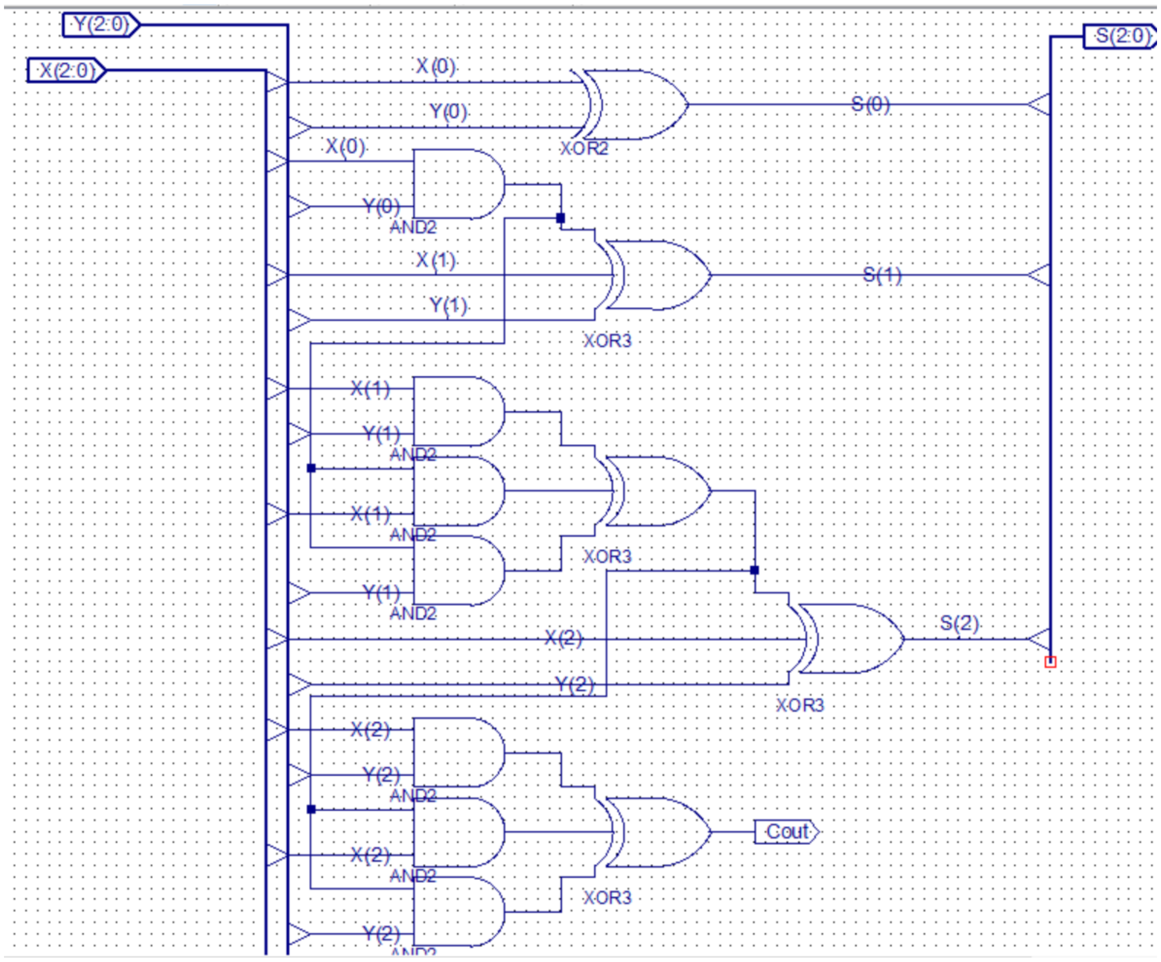
Part 1

The first step in this experiment should be analyzing and understanding what our expected output should be based on any given set of inputs. This will give us a better understanding of how to build the schematic of the circuit. The below table is a comprehensive view of the logic possibilities in the circuit at any given moment, matched with their respective output

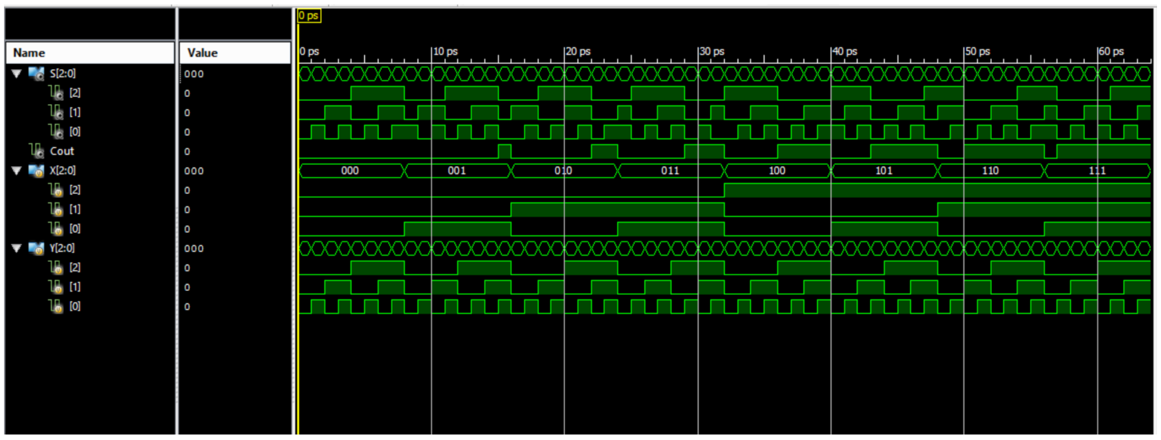
X	Y	X_2	Y_2	X_1	Y_1	X_0	Y_0	C_{out}	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	1
0	2	0	0	0	1	0	0	0	0	1	0
0	3	0	0	0	1	0	1	0	0	1	1
0	4	0	1	0	0	0	0	0	1	0	0
0	5	0	1	0	0	0	1	0	1	0	1
0	6	0	1	0	1	0	0	0	1	1	0
0	7	0	1	0	1	0	1	0	1	1	1
1	0	0	0	0	0	1	0	0	0	0	1
1	1	0	0	0	0	1	1	0	0	1	0
1	2	0	0	0	1	1	0	0	0	1	1
1	3	0	0	0	1	1	1	0	1	0	0
1	4	0	1	0	0	1	0	0	1	0	1
1	5	0	1	0	0	1	1	0	1	1	0
1	6	0	1	0	1	1	0	0	1	1	1
1	7	0	1	0	1	1	1	1	0	0	0
2	0	0	0	1	0	0	0	0	0	1	0
2	1	0	0	1	0	0	1	0	0	1	1
2	2	0	0	1	1	0	0	0	1	0	0
2	3	0	0	1	1	0	1	0	1	0	1
2	4	0	1	1	0	0	0	0	1	1	0
2	5	0	1	1	0	0	1	0	1	1	1
2	6	0	1	1	1	0	0	1	0	0	0
2	7	0	1	1	1	0	1	1	0	0	1
3	0	0	0	1	0	1	0	0	0	1	1
3	1	0	0	1	0	1	1	0	1	0	0
3	2	0	0	1	1	1	0	0	1	0	1
3	3	0	0	1	1	1	1	0	1	1	0
3	4	0	1	1	0	1	0	0	1	1	1

3	5	0	1	1	0	1	1	1	0	0	0
3	6	0	1	1	1	1	0	1	0	0	1
3	7	0	1	1	1	1	1	1	0	1	0
4	0	1	0	0	0	0	0	0	1	0	0
4	1	1	0	0	0	0	1	0	1	0	1
4	2	1	0	0	1	0	0	0	1	1	0
4	3	1	0	0	1	0	1	0	1	1	1
4	4	1	1	0	0	0	0	1	0	0	0
4	5	1	1	0	0	0	1	1	0	0	1
4	6	1	1	0	1	0	0	1	0	1	0
4	7	1	1	0	1	0	1	1	0	1	1
5	0	1	0	0	0	1	0	0	1	0	1
5	1	1	0	0	0	1	1	0	1	1	0
5	2	1	0	0	1	1	0	0	1	1	1
5	3	1	0	0	1	1	1	1	0	0	0
5	4	1	1	0	0	1	0	1	0	0	1
5	5	1	1	0	0	1	1	1	0	1	0
5	6	1	1	0	1	1	0	1	0	1	1
5	7	1	1	0	1	1	1	1	1	0	0
6	0	1	0	1	0	0	0	0	1	1	0
6	1	1	0	1	0	0	1	0	1	1	1
6	2	1	0	1	1	0	0	1	0	0	0
6	3	1	0	1	1	0	1	1	0	0	1
6	4	1	1	1	0	0	0	1	0	1	0
6	5	1	1	1	0	0	1	1	0	1	1
6	6	1	1	1	1	0	0	1	1	0	0
6	7	1	1	1	1	0	1	1	1	0	1
7	0	1	0	1	0	1	0	0	1	1	1
7	1	1	0	1	0	1	1	1	0	0	0
7	2	1	0	1	1	1	0	1	0	0	1
7	3	1	0	1	1	1	1	1	0	1	0
7	4	1	1	1	0	1	0	1	0	1	1
7	5	1	1	1	0	1	1	1	1	0	0
7	6	1	1	1	1	1	0	1	1	0	1
7	7	1	1	1	1	1	1	1	1	1	0

The above logic diagram shows the expected output of the parallel adder based on 6 total inputs (X_0**,** X_1, X_2, Y_0, Y_1, and Y_2).



After completion of the above schematic capture, the operator then synthesized and implemented the design. Using the isim interface, the operator next simulated and tested the schematic design for its accuracy based on all possible combinations of input and function selects. The results of this simulation is as follows:

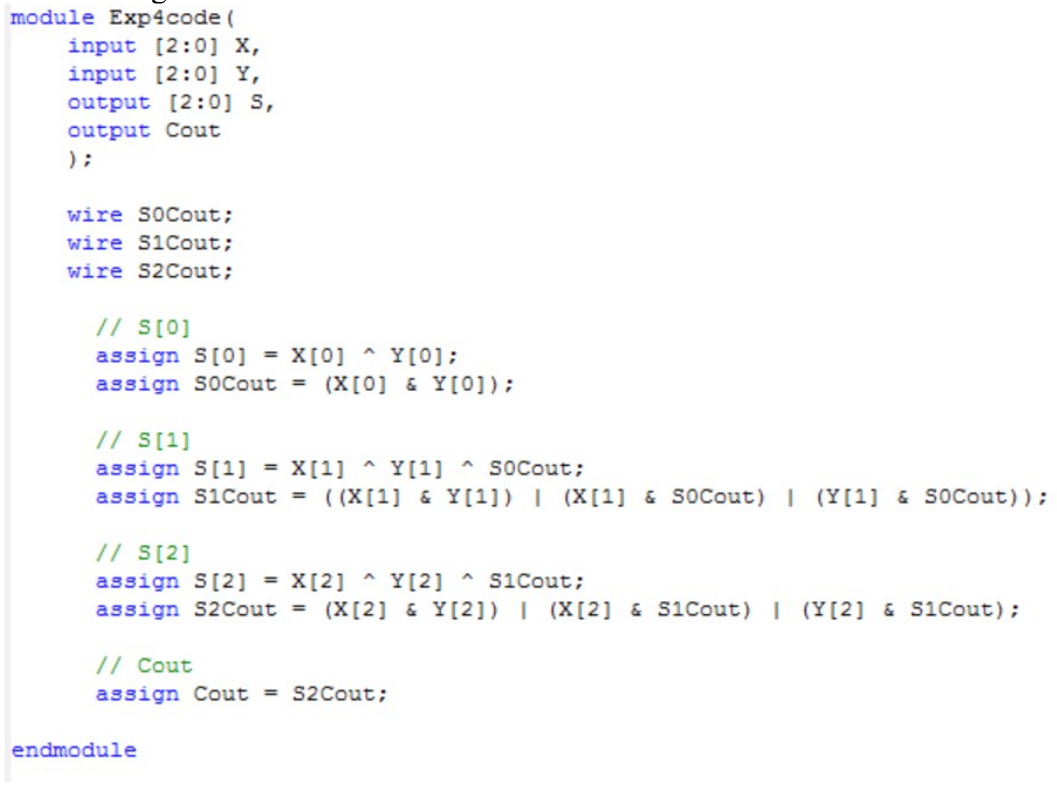


Once the simulation had been proven to provide proper output, the operator was ready to transfer the implementation over to the BASYS 2 development board. To do this, the operator used the ISE design tools to create area constraints such that the inputs were tied to individual switches on the BASYS 2 development board that represented high and low inputs into the gate and the output was tied to a single LED to show if the output was high or low (The chosen switches and LED can be seen in the expected results section). Next, the operator re-implemented the design with the new physical I/O assignments. Once the design was implemented, the ISE design tools were used to generate the programming file that is used to program the BASYS 2 development board. Doing this produced a *.bit file that was used for programming the board. To transfer the file to the BASYS 2 board, Digilent Adept was used. Once in Adept, with the BASYS 2 board powered on, the operator selected the initialize chain button to establish communication between the computer and the BASYS 2 board. Next, the operator used the browse button next to the "FPGA" text to select the newly created *.bit file for programming the BASYS 2 baord. Finally, the "program chain" button was used to push the program to the BASYS 2 board and the operator could test the program on the board for proper output.

Part 2

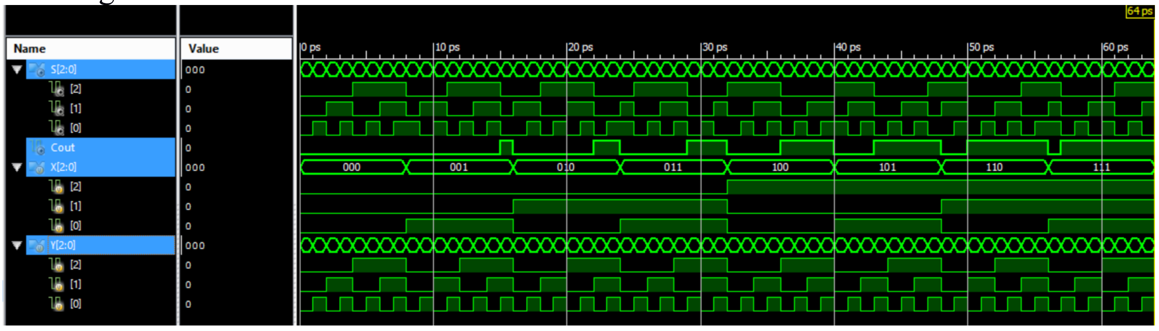
For the second part of the experiment, the operator repeated the first part using the VERILOG module in place of the schematic capture. This very slightly changed the process up to the point of creating a simulation, but beyond that, the process is identical.

When creating a VERILOG module, the user will select inputs and outputs based on the need of the problem and will also select a naming convention for each. The code for the multi-function gate is as follows:



```
module Exp4code(
    input [2:0] X,
    input [2:0] Y,
    output [2:0] S,
    output Cout
    );

    wire S0Cout;
    wire S1Cout;
    wire S2Cout;

      // S[0]
      assign S[0] = X[0] ^ Y[0];
      assign S0Cout = (X[0] & Y[0]);

      // S[1]
      assign S[1] = X[1] ^ Y[1] ^ S0Cout;
      assign S1Cout = ((X[1] & Y[1]) | (X[1] & S0Cout) | (Y[1] & S0Cout));

      // S[2]
      assign S[2] = X[2] ^ Y[2] ^ S1Cout;
      assign S2Cout = (X[2] & Y[2]) | (X[2] & S1Cout) | (Y[2] & S1Cout);

      // Cout
      assign Cout = S2Cout;

endmodule
```

After completion of each of the above VERILOG module, the operator then synthesized and implemented the design. Next, the operator used the newly written code to simulate the intended circuit for its accuracy based on all possible combinations of inputs. The resulting simulation is as follows:



Once the simulation had been proven to provide proper output, the operator was ready to transfer the implementation over to the BASYS 2 development board. To do this, the operator used the ISE design tools to create area constraints such that the inputs were tied to individual switches on the BASYS 2 development board that represented high and low inputs into the gate and the output was tied to a single LED to show if the output was high or low (The chosen switches and LED can be seen in the expected results section). Next, the operator re-implemented the design with the new physical I/O assignments. Once the design was implemented, the ISE design tools were used to generate the programming file that is used to program the BASYS 2 development board. Doing this produced a *.bit file that was used for programming the board. To transfer the file to the BASYS 2 board, Digilent Adept was used. Once in Adept, with the BASYS 2 board powered on, the operator selected the initialize chain button to establish communication between the computer and the BASYS 2 board. Next, the operator used the browse button next to the "FPGA" text to select the newly created *.bit file for programming the BASYS 2 baord. Finally, the "program chain" button was used to push the program to the BASYS 2 board and the operator could test the program on the board for proper output.

Expected Results of Circuit

The expected results of this circuit can be seen in the logic table previously created. To further verify proper output, the operator made assertions that checked for errors during each simulation. The code used for simulations is as follows:

```
module exp4test;

	// Inputs
	reg [2:0] X;
	reg [2:0] Y;

	// Outputs
	wire [2:0] S;
	wire Cout;
```

```
    // Instantiate the Unit Under Test (UUT)
    Exp4code uut (
        .X(X),
        .Y(Y),
        .S(S),
        .Cout(Cout)
    );

    initial begin
        // Initialize Inputs
        X = 0;
        Y = 0;


        // This will run every possible combination
        X = 0;
        Y = 0;
        // increment thru all Y per X
        repeat(8) begin
            Y = 0; // reset Y

            // increment Y inside X
            repeat(8) begin
                #1 // 1ps delay

                ////////////////////////////////////////////
                // Check for proper output
                if(X+Y != S && X+Y > 7) begin
                    if(X+Y-8 == S && Cout == 1) begin
                    end
                    else begin
                        $display("Assertion Error 1");
                    end
                end
                else if (X+Y != S) begin
                    $display("Assertion Error 2");
                end
                ////////////////////////////////////////////

                Y = Y + 1;
            end
            X = X + 1;
        end
    end

endmodule
```

And the code used for I/O mapping is as follows:

```
NET "X[0]" LOC = "P11";
NET "X[1]" LOC = "L3";
NET "X[2]" LOC = "K3";
NET "Y[0]" LOC = "F3";
NET "Y[1]" LOC = "E2";
NET "Y[2]" LOC = "N3";
NET "S[0]" LOC = "M5";
NET "S[1]" LOC = "M11";
NET "S[2]" LOC = "P7";
NET "Cout" LOC = "G1";
```

Design Specification Plan

The results of this experiment draw a sound conclusion that the objective has been met by the execution of the procedure. This is given by the fact that simulations and test results match verbatim to that of the expected results based on the calculations performed beforehand. The methodology chosen for this experiment followed a calculation-first route such that the operator would know what to expect. This also provided simplicity for the operator when designing the schematic as well as the VERILOG module. Given that a successful attempt was made to design, simulate, and implement a parallel adder, which was the objective of this experiment, this experiment has met all of its requirements.

Test Plan

This experiment should be tested and verified by the following means. The tester should obtain a BASYS 2 development board with this program loaded on it. Once the tester has set up the board and powered it on, he should test for output based on the provided logic table. Additionally, the tester can run a simulation at any time and watch for console output informing him of wrong outputs.

Results

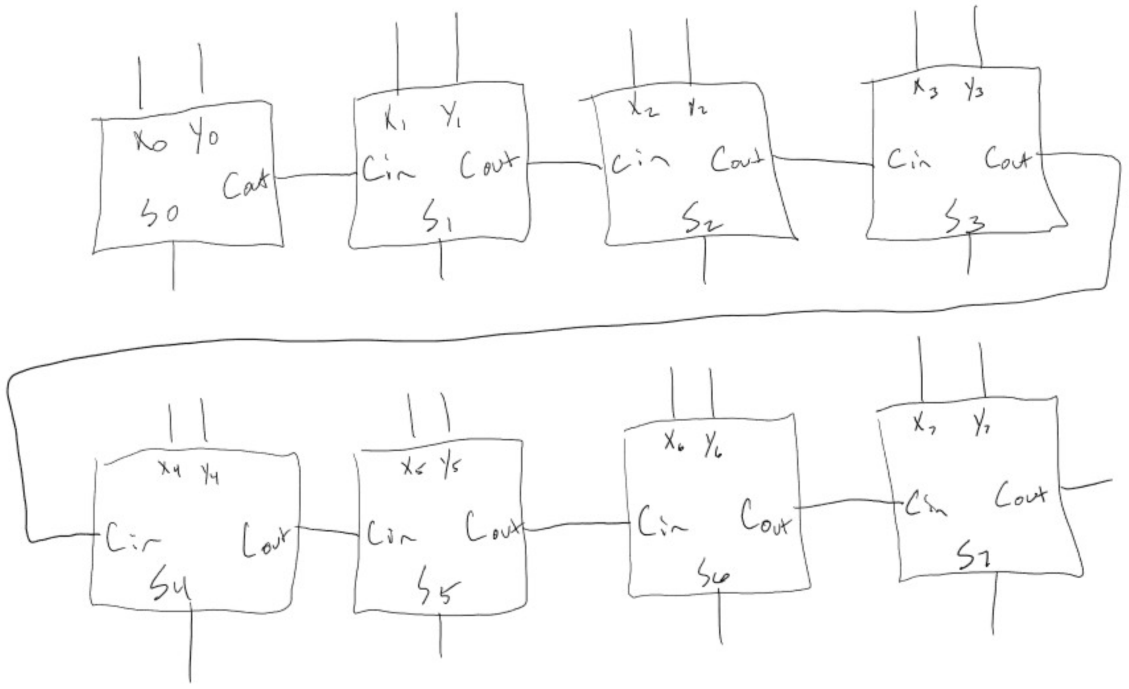
Please see the section entitled procedures for simulation results.
The outcome of this experiment accurately matched that of the calculations performed and presented in the expected results section of this report. This means that a parallel adder was successfully designed and implemented, because when it was given two numbers, the parallel adder was able to produce and return the sum of the given values.

Conclusion

In conclusion, this experiment suggests that an adder of any size can be created such that it would produce and return the sum of two numbers of any size.

Questions

1. Draw an 8-bit adder diagram



2. Comment on the fasibility of designing an 8-bit adder using the brute force method.

   This would be incredibly overwhelming due to size and complexity.

3. Identify the advantages and disadvantages of the brute force method.

   The brute force method requires a greater amount of design work but might be more easily adaptable to specific design needs.

4. Identify the advantages and disadvantages of the iterative cell method.

   The iterative cell method provides a simple and rapid design approach for an adder.

5. Have I met all of the requirements of this lab?

   Yes

6. How should my design be tested?

   This information can be found in the section above entitled “Test Plan”. Additionally, a simulation can be run with the test code provided in this document and the outputs, which are verified to be accurate can be verified with the outputs of a Basys 2 board containing the CP132 package processor.