Objective

In this lab, a fast-adder circuit for two-opperand addition was designed called carry-lookahead addition. The design consists of using a combination of dataflow and structural Verilog. The designs were then simulated on the Xilinx ISE to check for correctness. The propagation delays for for a 4-bit and 16-bit carry-lookahead adder were also analyzed in order to compare to similar sized ripple-carry adders.

Design

Experiment 1

To start off the experiment, the following code was written for the most part as part of the prelab. A few modifications were made into order to function properly. The first three blocks of code are of the modules (*generate-propagate-unit, carry-lookahead-unit,* and *summation-unit*) used in *Code Block 4* for the top level module *carry-lookahead-4bit*

Code Block 1: Generate Propagate Unit (4-Bit)

```
1  `timescale 1ns / 1ps
2  module generate_propagate_unit(G, P, X, Y);
3    // Ports are wires as we will use dataflow
4    output wire [3:0] G, P;
5    input wire [3:0] X, Y;
6
7    // Generate Propagate Unit Logice
8    assign G = X & Y;
9    assign P = X ^ Y;
10 endmodule
```

Code Block 2: Carry Lookahead Unit (4-Bit)

```
1  `timescale 1ns / 1ps
2  module carry_lookahead_unit(C, G, P, C0);
3          // Ports are wires because we will use dataflow
```

```
4     output wire [4:1] C; // C4, C3, C2, C1
5     input wire [3:0] G, P; // Generates and propagates
6     input wire C0; // input carry
7           // Carry Lookahead Unit Logic
8     assign C[1] = G[0] | (P[0] & C0);
9     assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C0);
10    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0])
11          | (P[2] & P[1] & P[0] & C0);
12    assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1])
13          | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1]
14          & P[0] & C0);
15 endmodule
```

Code Block 3: Summation Unit (4-Bit)

```
1  `timescale 1ns / 1ps
2  module summation_unit(S, P, C);
3    // Ports are wires becasue we will use dataflow
4    output wire [3:0] S; // sum vector
5    input wire [3:0] P, C;
6
7    // Summation Unit Logic
8    assign S = P ^ C;
9  endmodule
```

Code Block 4: Top Level Carry Lookahead (4-Bit)

```
1  `timescale 1ns / 1ps
2  module carry_lookahead_4bit(Cout, S, X, Y, Cin);
3    // Ports are wires because we are using structual
4    output wire Cout; // C4 for a 4-bit adder
5    output wire [3:0] S; // final 4-bit sum vector
6    input wire [3:0] X, Y; // the 4-bit addends
```

```
7   input wire Cin; // input carry
8
9   // Intermediete wires
10  wire [3:0] G, P;
11  wire [4:0] C;
12  assign C[0] = Cin;
13
14  // generate_propagate_unit(G, P, X, Y);
15  generate_propagate_unit gpu(G, P, X, Y);
16
17  // carry_lookahead_unit (C, G, P, C0);
18  carry_lookahead_unit clu(C[4:1], G, P, Cin);
19
20  // summation_unit (S, P, C);
21  summation_unit su(S, P, C[3:0]);
22
23  // Assign Cout to wire C
24  assign Cout = C[4];
25
26 endmodule
```

For the next part, gate delays were modified in the following modules.

Code Block 5: Generate Propagate Unit (4-Bit with Delay)

```
8   assign #2 G = X & Y;
9   assign #2 P = X ^ Y;
```

Code Block 6: Carry Lookahead Unit (4-Bit with Delay)

```
8   assign #2 C[1] = G[0] | (P[0] & C0);
9   assign #2 C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C0);
10  assign #2 C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0])
11          | (P[2] & P[1] & P[0] & C0);
```

```
12    assign #2 C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1])
13            | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1]
14            & P[0] & C0);
```

Code Block 7: Summation Unit (4-Bit with Delay)

```
8     assign #2 S = P ^ C;
```

Experiment 2

In the next experiment, a block carry lookahead unit was designed. The module interface is as follows:

Code Block 8: Block Carry Lookahead Unit

```
1  `timescale 1ns / 1ps
2  module block_carry_lookahead_unit(G_star, P_star, C, G, P, C0);
3    // Ports are wires because we will use dataflow
4    output wire G_star, P_star;
5    output wire [3:1] C; // C3, C2, C1
6    input wire [3:0] G, P;
7    input wire C0; // Input carry
8
9    // Block Carry Lookahead Unit Logic
10   assign #2 G_star = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1])
11     | (P[3] & P[2] & P[1] & G[0]);
12
13   assign #2 P_star = P[3] & P[2] & P[1] & P[0];
14
15   assign #2 C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0])
16     | (P[2] & P[1] & P[0] & C0);
17
18   assign #2 C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C0);
19
20   assign #2 C[1] = G[0] | (P[0] & C0);
```

```
21
22 endmodule
```

Next, the generate propegate unit and summation unit were modified in order to take in 15 bit wires as shown below. The gate delays were also removed.

Code Block 9: Generate Propagate Unit (16-Bit)

```
4   output wire [15:0] G, P;
5   input wire [15:0] X, Y;
```

Code Block 10: Summation Unit (16-Bit)

```
4   output wire [15:0] S; // sum vector
5   input wire [15:0] P, C;
```

Finally the past modules were implemented into a toplevel module for a 16-bit carry lookahead adder.

Code Block 11: 16-Bit Carry Lookahead Adder

```
1  `timescale 1ns / 1ps
2  module carry_lookahead_16bit(Cout, S, X, Y, Cin);
3    // Ports are wires as we will use structual
4    output wire Cout; // C_16 for a 16-bit adder
5    output wire [15:0] S; // final 16-bit sum vector
6    input wire [15:0] X, Y; // the 16-bit addends
7    input wire Cin;
8
9    // Intermediate nets
10   wire [16:0] C; // 17-bit carry vector
11   wire [15:0] P, G; // 16 bit generate and propagate vectors
12   wire [3:0] P_star, G_star; // block gens and props
13
14   assign C[0] = Cin;
15
```

```
16    // Instantiate Modules
17    generate_propagate_unit gpu (G, P, X, Y);
18
19    block_carry_lookahead_unit BLCAU0(
20      .G_star (G_star[0]),
21      .P_star (P_star[0]),
22      .C (C[3:1]),
23      .G (G[3:0]),
24      .P (P[3:0]),
25      .C0 (C[0])
26      );
27
28    block_carry_lookahead_unit BLCAU1(
29      .G_star (G_star[1]),
30      .P_star (P_star[1]),
31      .C (C[7:5]),
32      .G (G[7:4]),
33      .P (P[7:4]),
34      .C0 (C[4])
35      );
36
37    block_carry_lookahead_unit BLCAU2(
38      .G_star (G_star[2]),
39      .P_star (P_star[2]),
40      .C (C[11:9]),
41      .G (G[11:8]),
42      .P (P[11:8]),
43      .C0 (C[8])
44      );
45
46    block_carry_lookahead_unit BLCAU3(
```

```
47      .G_star (G_star[3]),
48      .P_star (P_star[3]),
49      .C (C[15:13]),
50      .G (G[15:12]),
51      .P (P[15:12]),
52      .C0 (C[12])
53      );
54
55    carry_lookahead_unit CLAU(
56      .C ({C[16], C[12], C[8], C[4]}),
57      .G (G_star),
58      .P (P_star),
59      .C0 (C[0])
60      );
61
62    summation_unit su(S,P,C[15:0]);
63
64    assign Cout = C[16];
65  endmodule
```

Results

Experiment 1

Below is the waveform for the 4-Bit carry lookahead adder.

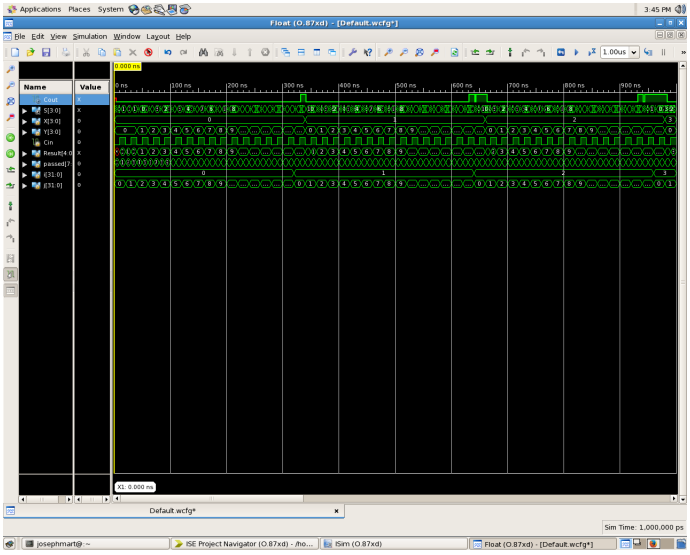

Figure 1: *Full waveform*

Below are the test results for the same module where all the tests results passed.

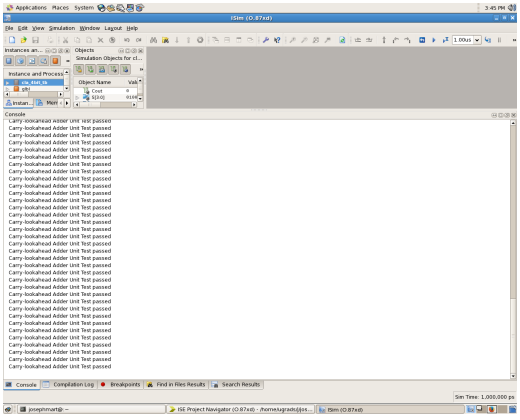

Figure 2: *Experiment 1 Test Results*

Below is the waveform with the gate delays added.

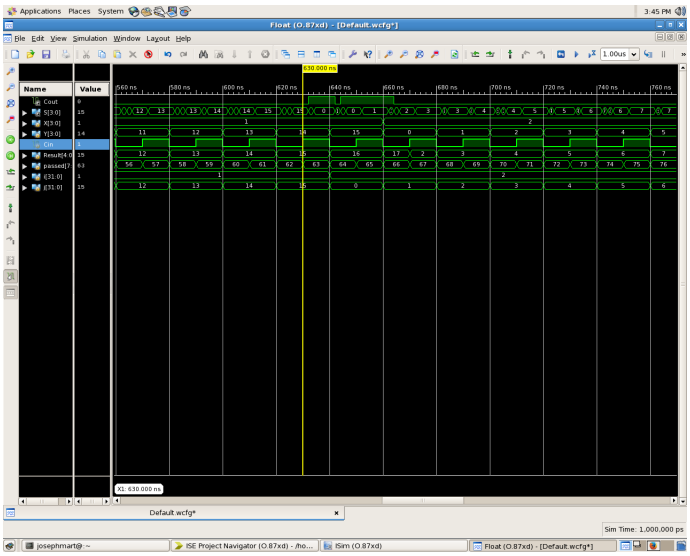

Figure 3: *Difference 6ns*

Experiment 2

Below is the first time the tests failed for 16 bit carry lookahead adder. The TEST_DELAY was set to 9 ns

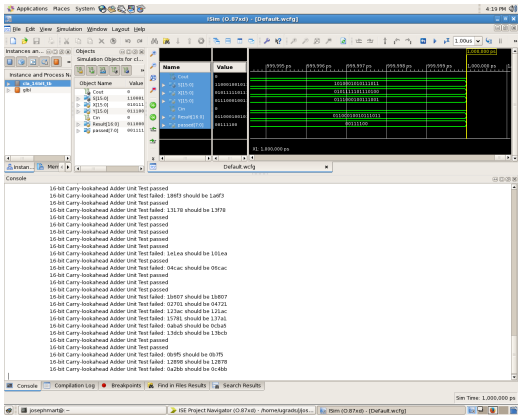

Figure 4: *Experiment 3 Failed Tests*

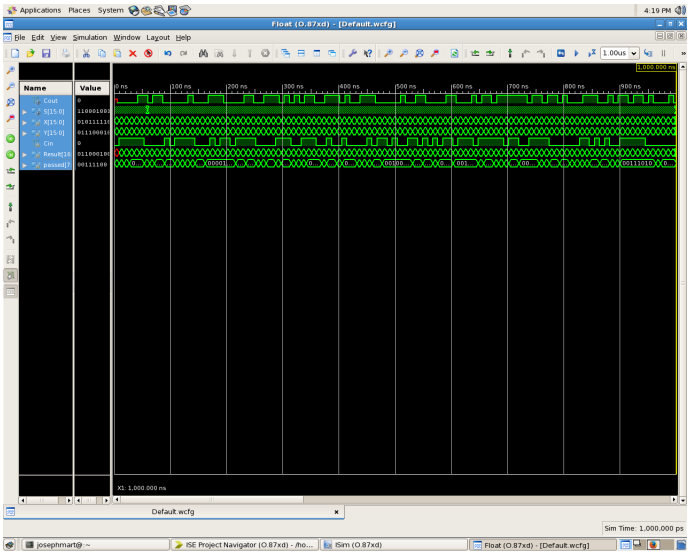

Figure 5: *Failed Waveform*

Conclusion

In conclusion, I learned how faster adders than ones previously created are designed. This lab also allowed me to build upon my ability to create basic modules using both structural and dataflow Verilog. After creating a 4-Bit adder, I was able to create a 16-bit lookahead adder using and modifying previously created modules.

Questions

1. **Include the source code with comments for all modules in lab. you do not have to include test bench code. Code without comments will not be accepted!** *In report*

2. **Include the simulation screenshots requested in the above experiments in addition to the corresponding test bench console output.** *In report*

3. **Answer all questions through the lab manual: If your calculations in the prelab are correct and you correctly added delays to your sub-modules, you should find that the computed delay matches the measure delay. Is**

this the case? No they do not match. I believe the issue is with how gate delays were not assigned correctly.

4. **How does the gate-count of the 16-bit carry-looahead adder compare to that of a ripple-carry adder of the same size.**

   The gate-count of a 16-bit-carry-lookahead adder compared to a ripple-carry of the same size is that CLA has 82-gates and RC has 80 gates. Also the CLA functions faster than the RC.

5. **How does the propagation delay of the 4-bit carry-lookahead adder compare to that of a ripple-carry adder of the same size. Similarly, how does the 16-bit carry-lookahead adder compare to that of a ripple-carry adder of the same size.**

   The propagation delay of a 16-bit carry lookahead adder compared to a ripple carry adder is that the 16-bit CLA has a bigger delay than the RC. This is due to how the CLA uses parallel circuitry rather than a long string of full adders in series like the RC.

Student Feedback

1. **What did you like most about the lab assignment and why? What did you like least about it and why?** I liked building upon my knowledge of verilog.

2. **Were there any section of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?** The lab was pretty clear

3. **What suggestions do you have to improve the overall lab assignment?** This was the easiest lab so far so I do not have any suggestions.