ECE 385

Spring 2023
Experiment #3

16-bit Adders

Max Ma & Dhruv Kulgod
NL / Friday 9:15 AM - 9:30 AM
Nick Lu

Purpose of Circuit:

In this lab, we designed and explored the efficiency, speed, and area of three simple combinational adder designs: carry ripple adder, carry lookahead adder, and carry select adder. These three adders were designed to add two 16-bit unsigned numbers, each producing the same result when treated as a black box. Using SystemVerilog, we designed the three adders and simulated them using a testbench. Experiments #2 and #3 have laid the foundation of how to design a simple Arithmetic Logic Unit (ALU) for any processor architecture. After completing this experiment, we learned that each adder has strengths and weaknesses, which we will continue to explore in this report.

Written Description of Circuit:

Background:

Each of the three adder designs have area, complexity, and performance tradeoffs; however, they are all built from the most basic building block: the full adder. The full adder is a combinational logic circuit that has inputs for A, B, and a carry. It produces a sum between 0 and 3, which are the output in the sum (low) and carry (upper). The truth table and K-maps are shown in Figure 1 and Figure 2 respectively.

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1: Truth Table for Full Adder Circuit

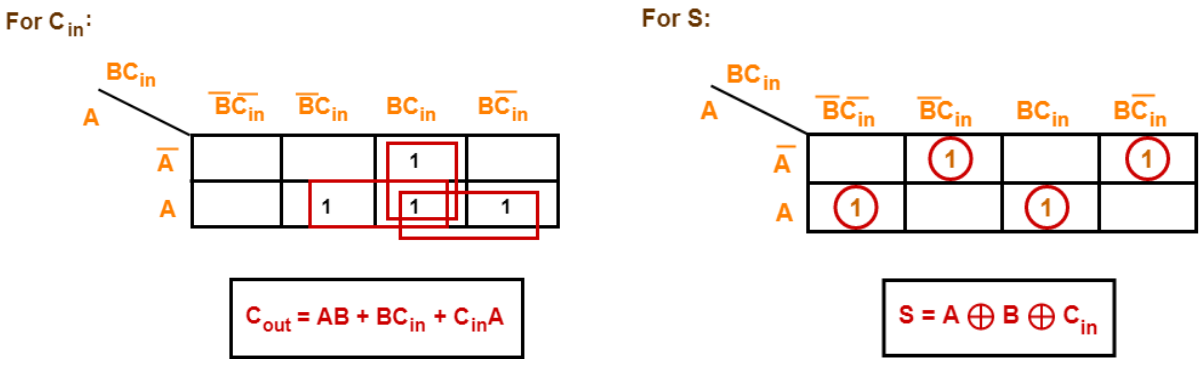

Figure 2: K-map for Full Adder Circuit

Carry Ripple Adder (CRA):
At a high level, the CRA takes the least amount of area, is the simplest in design, and slowest in performance. When implementing the CRA, the designer intends to save physical gate area by giving up computational speed. In more detail, the CRA chains or ripples N full adders together to make a N-bit CRA. Specifically, the carry in and carry out of each full adder is linked together as shown in Figure 8. This naive approach has a crippling result in terms of performance. Although the A and B bits are ready instantaneously, the next full adder cannot compute the appropriate sum and carry out bits because it is dependent on the previous full adder's carry out bit. This results in rippling of carry bits from previous full adders making the design take the computational time of N full adders: O(N).

Carry Lookahead Adder (CLA):
At a high level, the CLA takes a large amount of area, is complex in design, and fastest in performance. When implementing the CLA, the designer intends to increase the physical gate area of his/her design to speed up computational speed. In more depth, the CLA combats the rippling problem of the CRA by computing two additional bits using two-level combinational logic. These bits are the propagation (P) and generation (G) bits. A full adder will propagate the carry in if A or B is logic level 1: $P(A,B) = A \oplus B$. A full adder will generate a carry out bit if both A and B are logic level 1: $G(A,B) = A \cdot B$. The P and G logic bits are used to predict the carry in for the next full adder. One drawback of this performance gain is that it takes up a large physical area, especially for larger N-bit CLAs. This is shown in Figure 3: the logic to predict the next carry in quickly increases in complexity and size making is impossible to implement.

$$C_0 = C_{in}$$
$$C_1 = C_{in} \cdot P_0 + G_0$$
$$C_2 = C_{in} \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1$$
$$C_3 = C_{in} \cdot P_0 \cdot P_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + G_1 \cdot P_2 + G_2$$

Figure 3: Computation for Carry In Bits for a 4-bit Carry Look Ahead Adder

To make the CLA a viable option as opposed to the CRA, we need to resort to a hierarchical 4x4 CLA design. This design chains 4-bit CLAs (shown in Figure 9) together into a larger 4x4 or 16-bit CLA shown in Figure 10. The benefit of this design is that we will abstract each 4-bit CLA into a group and treat the resulting hierarchy as another 4-bit CLA. Each 4-bit CLA will produce group P and G bits for the hierarchy to implement the next group carry in computation. It is evident that the group carry in equations (Figure 5) take on the same logic as shown in Figure 3. This is the hierarchical nature of the 4x4 CLA design, which can be extended to NxN bit Carry Lookahead Adders.

$$P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$
$$G_G = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

Figure 4: Group Propagation and Generation Bits for 4x4 CLA Design

$$C_4 = G_{G0} + C_0 \cdot P_{G0}$$
$$C_8 = G_{G4} + G_{G0} \cdot P_{G4} + C_0 \cdot P_{G0} \cdot P_{G4}$$
$$C_{12} = G_{G8} + G_{G4} \cdot P_{G8} + G_{G0} \cdot P_{G8} \cdot P_{G4} + C_0 \cdot P_{G8} \cdot P_{G4} \cdot P_{G0}$$
$$\dots$$

Figure 5: Group Carry In Bits for 4x4 CLA Design

Carry Select Adder (CSA):

At a high level, the CSA takes a very large amount of area with a semi-complex design. In terms of performance, the CSA falls in between the CRA (slowest) and CLA (fastest) designs. When implementing the CLA, the designer intends to increase the physical gate area of his/her design to speed up computational speed. Diving into the details, the CSA tries to compute all the possible outcomes to eliminate the rippling of the CRA. As shown in Figure 11, the CSA is composed of 4-bit CRAs with some glue logic. The first 4-bit CRA will one-by-one ripple to produce the corresponding carry out. Meanwhile, the other 4-bit groups are computing the possibility that their carry in is either logic low or logic high. These speculative computations happen in parallel with the first CRA making their rippling negligible. The glue logic will select the appropriate sum based on the output of the previous group. Hence, the rippling is divided by four after the result of the first CRA making it faster while increasing the physical area.

High Level Block Diagram:

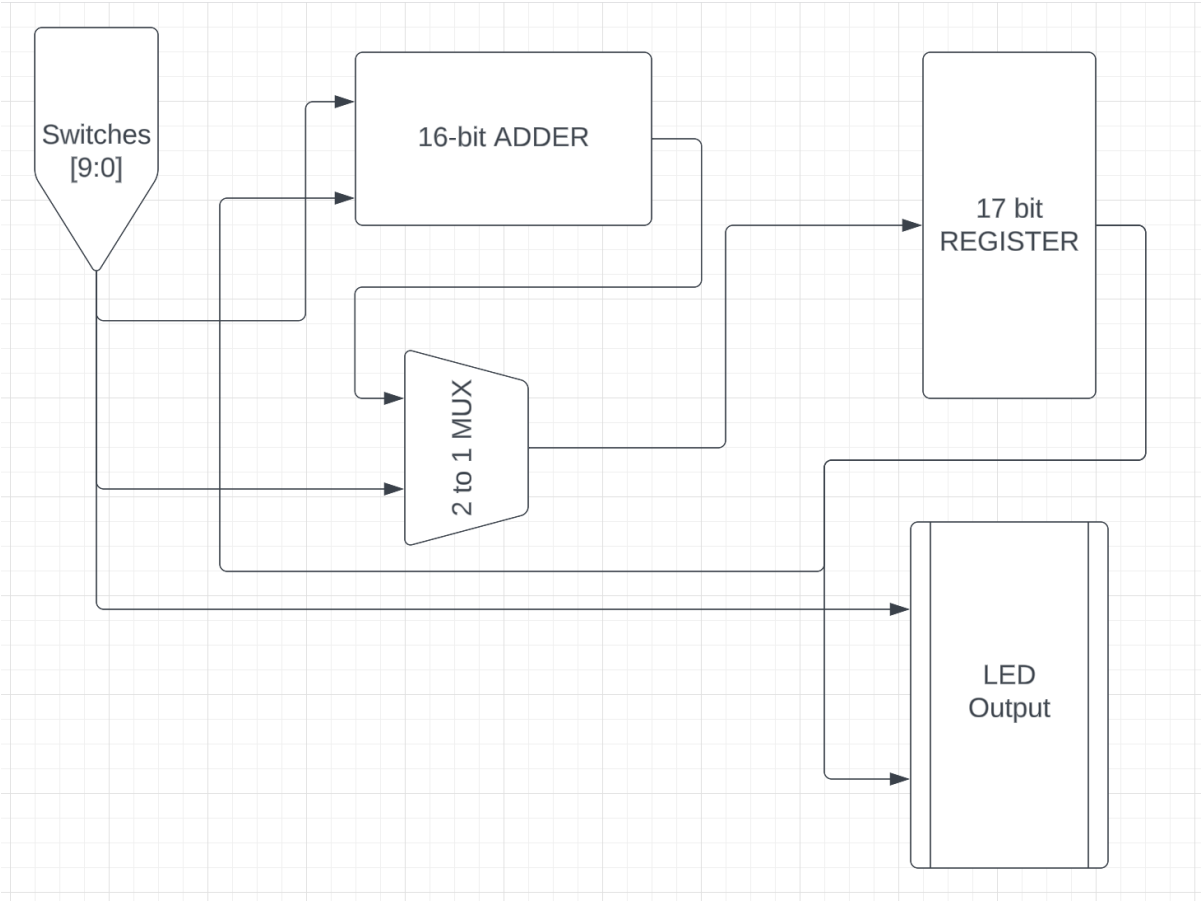

Figure 6: High level block diagram

The diagram above showcases the logical flow of data in our design. We first store a value into the register using the switches (selecting the correct line in the MUX). This value can be seen in our HEX display as well. Then, we select a new value with our switches. When the “ADD” button is pressed, the adder block adds the values in the register and switches, and stores it in the register again. We see the final value displayed in hex.

Logic Diagrams:

The logic diagrams for the full adder, carry ripple adder, carry lookahead adder, and carry select adder are shown below in Figures 7-11 respectively.

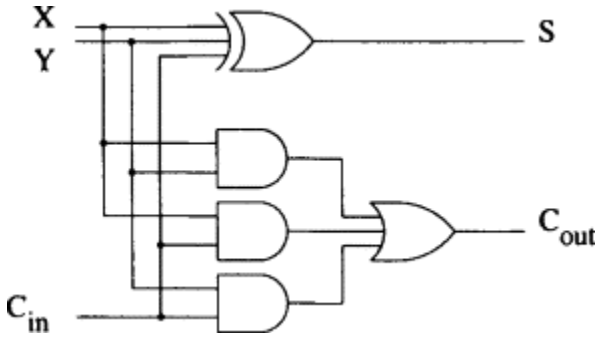

Figure 7: Full Adder Logic Diagram

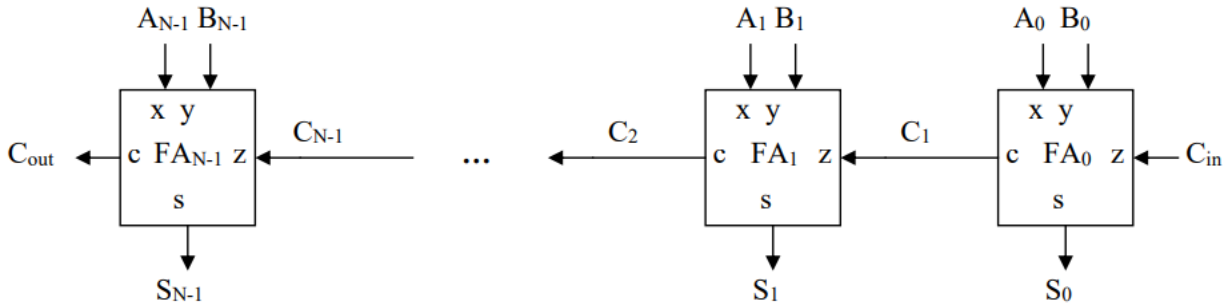

Figure 8: Carry Ripple Adder Logic Diagram with N = 16

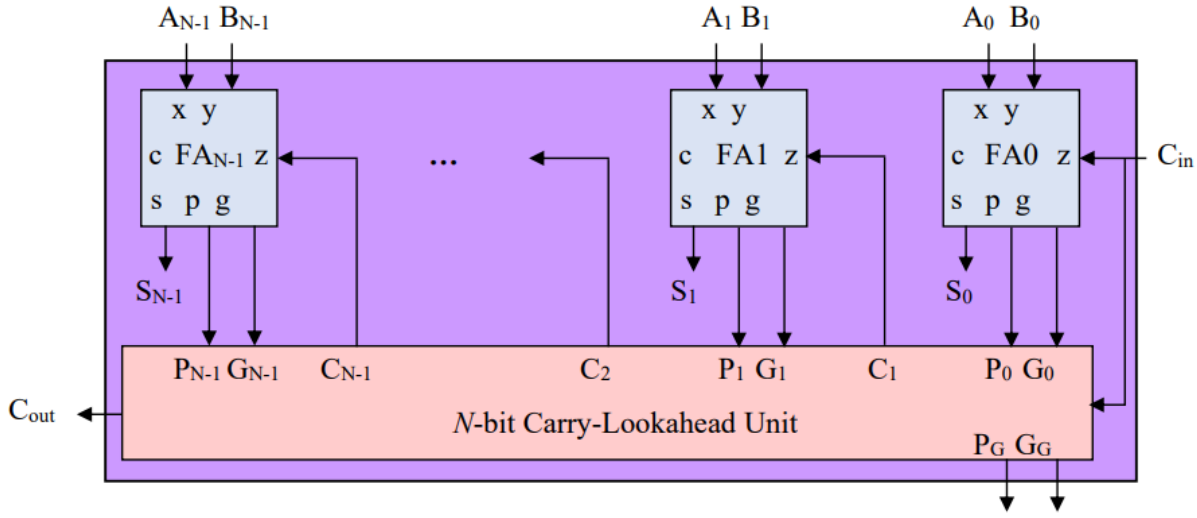

Figure 9: Single 4-bit Carry Lookahead Adder Logic Diagram

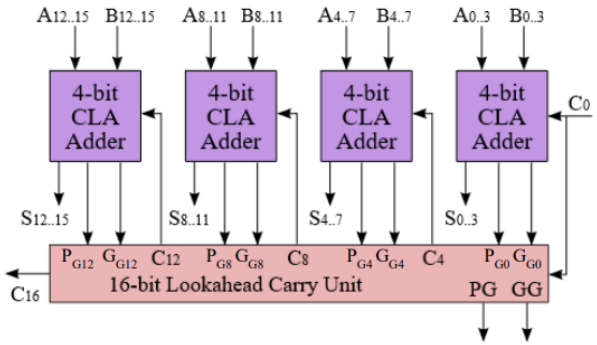

Figure 10: 16-bits 4x4 Hierarchical Carry Lookahead Adder Logic Diagram

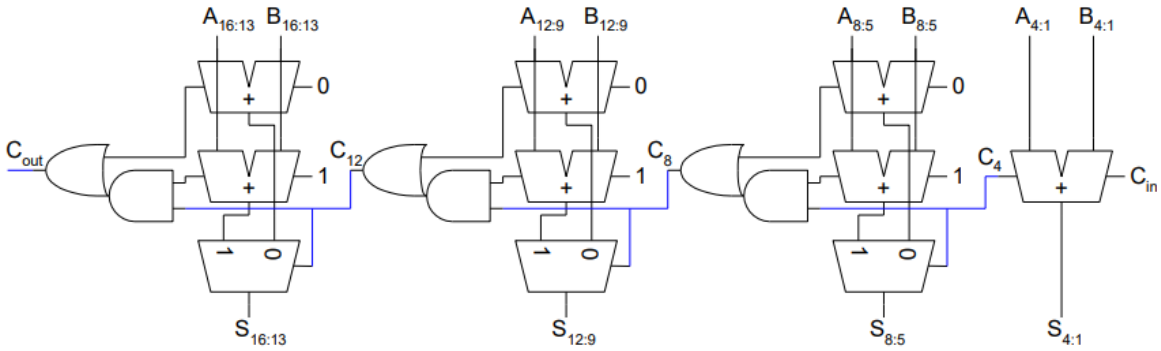

Figure 11: Carry Select Adder Logic Diagram with Glue Logic

Answers to Pre-Lab:

1.

Carry Ripple Adder		Carry Lookahead Adder		Carry Select Adder	
Memory	0	Memory	0	Memory	0
Frequency (MHz)	1	Frequency (MHz)	1.002	Frequency (MHz)	1.014
Total Power (mW)	1	Total Power (mW)	1.0002	Total Power (mW)	0.998

Figure 12: Normalized data for the each adder - Carry Ripple as baseline

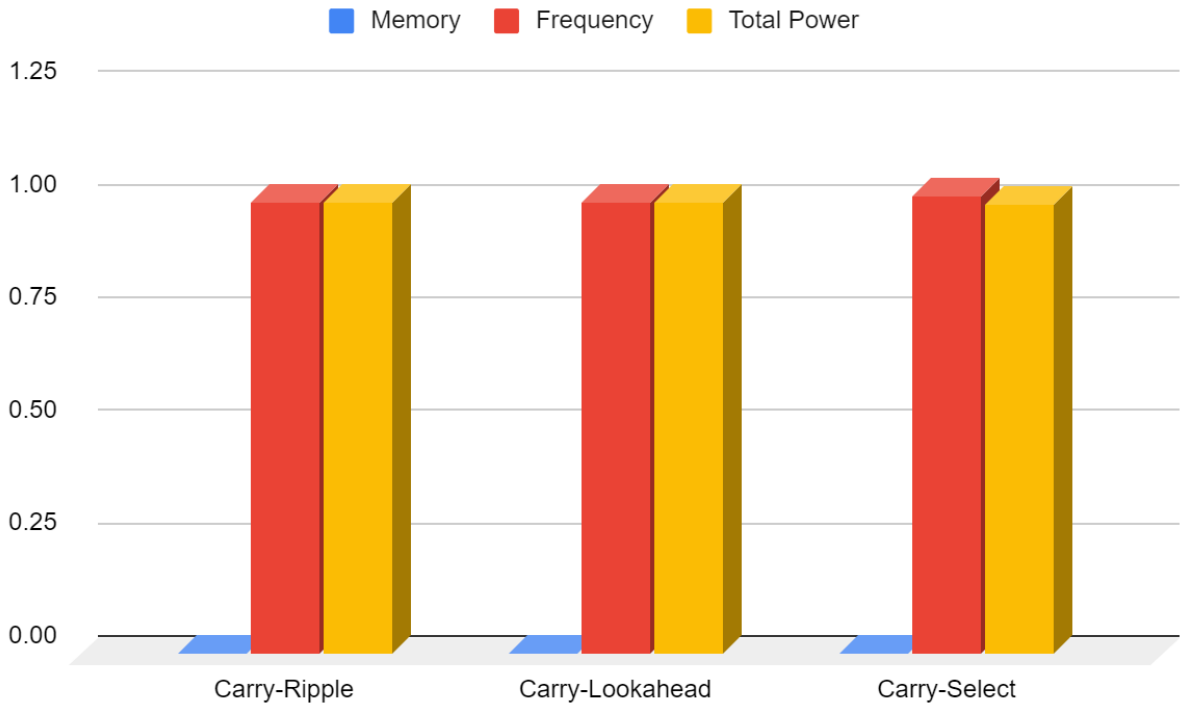

Figure 13: Normalized Bar Graph

We can see that for just 16 bits, the differences between the different adders are not great. Carr-Select does run at a higher frequency than the rest, probably thanks to the parallelized nature of its operation. However, even miniscule differences here would blow up at the number of operations that occur in any modern machine. Thus, comparing statistics like this is a very important step while designing hardware systems.

Answers to Post-Lab:

2. To answer if the 4x4 hierarchical design is ideal, there are some ideas that need to be considered. First, the ideal ratio between supporting glue logic resources and computational resources needs to be determined. By increasing the design to 8x2, you will reduce the amount of glue logic; however, you will also decrease your ability to compute in parallel. On the other hand, going with a 2x8 design will increase the glue logic required as well as increase the parallel compute power. Will this additionally glue logic increase the number of delays by more than you save with more parallelism? Probably. In the end, the 4x4 hierarchy design is ideal because it costs roughly 10 delays, which is less compared to the 8x2 and 2x8 hierarchical design.
3. The following table illustrates the design statistics for each adder. The breakdown comparison is indicative of the results mentioned in this report. For example, the CRA is smallest in physical area, while the CLA is largest in physical area. Additionally, it is observed that the CLA is the fastest as explained in the written description. The CLA also

consumes the most total power. This is expected since it has the largest physical area, which means that it will need more power to drive more LUTs.

Carry Ripple Adder		Carry Lookahead Adder		Carry Select Adder	
LUT	79	LUT	97	LUT	85
DSP	0	DSP	0	DSP	0
Memory	0	Memory	0	Memory	0
Flip-Flop	20	Flip-Flop	20	Flip-Flop	20
Frequency (MHz)	67.7	Frequency (MHz)	67.84	Frequency (MHz)	68.65
Static Power (mW)	89.97	Static Power (mW)	89.97	Static Power (mW)	89.97
Dynamic Power (mW)	1.57	Dynamic Power (mW)	1.47	Dynamic Power (mW)	1.44
Total Power (mW)	105.3	Total Power (mW)	105.33	Total Power (mW)	105.15

Table 14: Detailed Design Statistics for CRA, CLA, and CSA

FPGA Labs: Block Diagram:

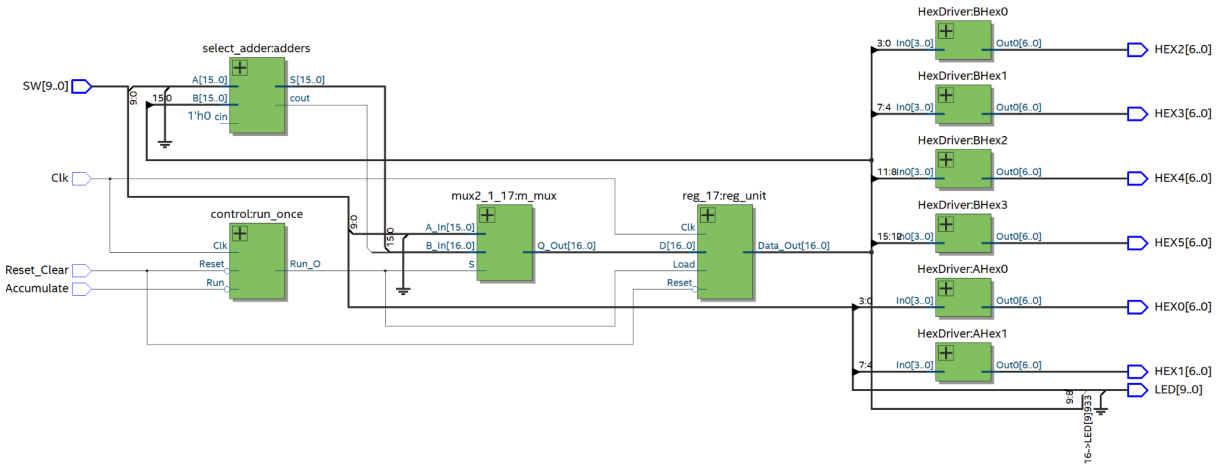

Figure 15: Top level RTL block diagram

The top level RTL shows a similar structure to the high-level block diagram. The adder module in the figure (top left) is the Carry Select adder, but all three adders of this lab can be interchanged here, as their high-level inputs and outputs are the same.

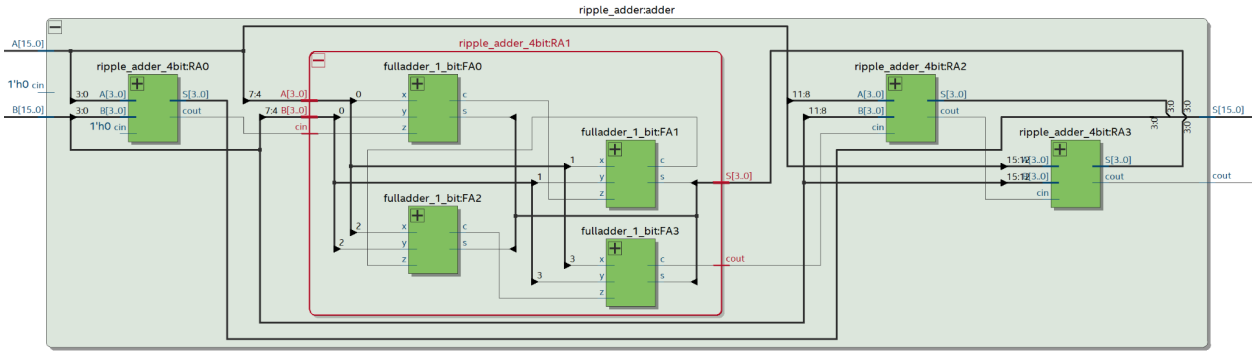

Figure 16: Carry-Ripple Adder RTL

The diagram consists of 4 four-bit ripple adder modules chained together. One of these four-bit modules is expanded to show they in turn consist of 4 one-bit full adders.

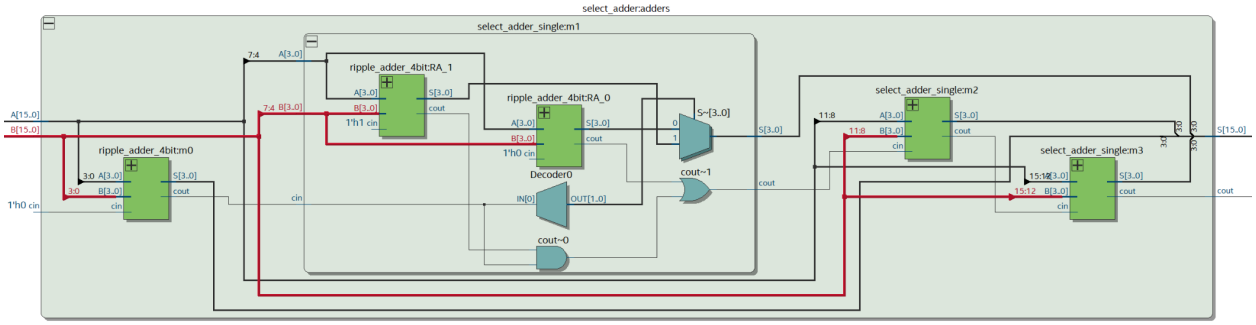

Figure 17: Carry-Select Adder RTL

The carry select adder has 4 internal modules as well. The first is a four-bit ripple adder. The next three are 4 bit carry select adders: each essentially two ripple adders in series to calculate the sum for a 0 and 1 bit carry in simultaneously.

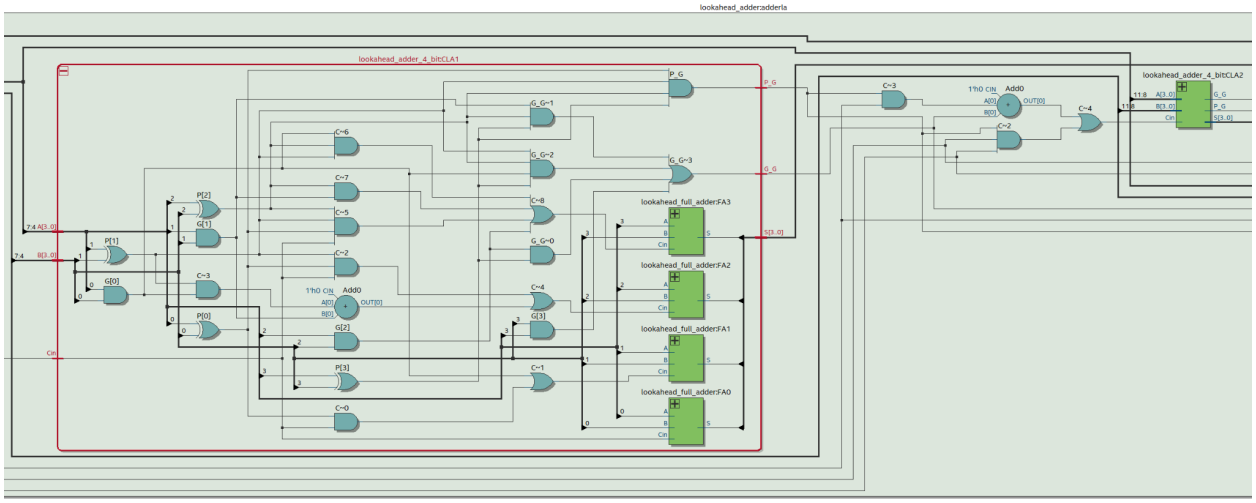

Figure 18: Internal RTL of a 1 bit CLA adder.

The Carry-Lookahead adder has the most complicated internal structure. Again, consisting of 4 four-bit modules, each module calculates the sum according to the formula discussed earlier in the report. Following the hierarchical design to prevent slow rippling, each of these 4 modules is treated like individual ‘bits’ to calculate the carries between them, giving us the final sum.

FPGA Labs: Design Module Information

Reg_17:

Inputs: Clk, Reset, Load, [16:0] D
Outputs: [16:0] Data_Out
Description: This module describes a 17-bit synchronous register that will reset to 0x0 when the reset signal is logic level high. Otherwise, if the load signal is logic level high it will parallel load the 17-bit input into the register.

Mux2_1_17:

Inputs: S, [15:0] A_In, [16:0] B_In
Outputs: [16:0] Q_Out
Description: This module describes a 17-bit parallel multiplexer with one select bit. For a select bit of logic level low, it will select the A input and append a 1’b0 to the front. Otherwise, for a select bit of logic level high, it will select the B input.

HexDriver:

Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: This module acts as a translator between binary to hexadecimal. A set of four binary bits are taken as input and linked to their corresponding pattern on the 7-Segment Hex Display using a case statement.

Control

Inputs: Clk, Reset, Run
Outputs: Run_O
Description: This module describes a simple FSM using a three always design in SystemVerilog. The state machine will output a load signal with logic level high for one clock cycle once RUN is pressed. It will not be able to execute again until RUN is released or RESET is pressed.

Adder_toplevel:

Inputs: Clk, Reset_Clear, Run_Accumlate, [9:0] SW
Outputs: [9:0] LED, [6:0] Hex [0:5]

Description: This is the top level module for the project. It does all the necessary module instantiations and connects internal nets between the modules: control unit, muxes, registers, adder, and hex drivers. It inverts the active low push-buttons to active high and assigns drivers to the LEDs.

Carry Ripple Adder:

Inputs: [15:0] A, [15:0] B, Cin
Outputs: [15:0] S, Cout
Description: This is the top level module for the CRA, which instantiates four 4-bit CRAs and connects them in series. The carry out of each module feeds into the next bit's carry in. The first carry in for the first module is always zero. Each 4-bit CRA consists of 4 full adders in series.

Carry Lookahead Adder:

Inputs: [15:0] A, [15:0] B, Cin
Outputs: [15:0] S, Cout
Description: This is the top level module for the CLA, which instantiates four 4-bit CLAs and connects them into a hierarchical design. It computes the group carry in bits for each 4-bit CLA using the respective group propagation and generation bits. Additionally, it computes the final carry out (overflow bit). The individual 4-bit CLAs use the full adder module to create a carry lookahead adder according to Figure 9.

Carry Select Adder:

Inputs: [15:0] A, [15:0] B, Cin
Outputs: [15:0] S, Cout
Description: This is the top level module for the CSA, which instantiates three 4-bit CSA modules and one 4-bit CRA module. This is because the first module has no uncertainty about its carry-in value (it is always zero). Each 4-bit CRA module has two 4-bit CRAs in parallel, one with a carry-in of 1 and the other with 0. A switch case decides which sum to pass to the output depending on the carry in from the previous module.

FPGA Labs: Design Statistics

See Post-Lab Answer 3.

FPGA Labs: Annotated Simulation Waveform

The annotated simulation waveform shown in Figure 19 illustrates four successful test cases for each of the adder designs. The respective test cases are labeled and described in the annotated waveform.

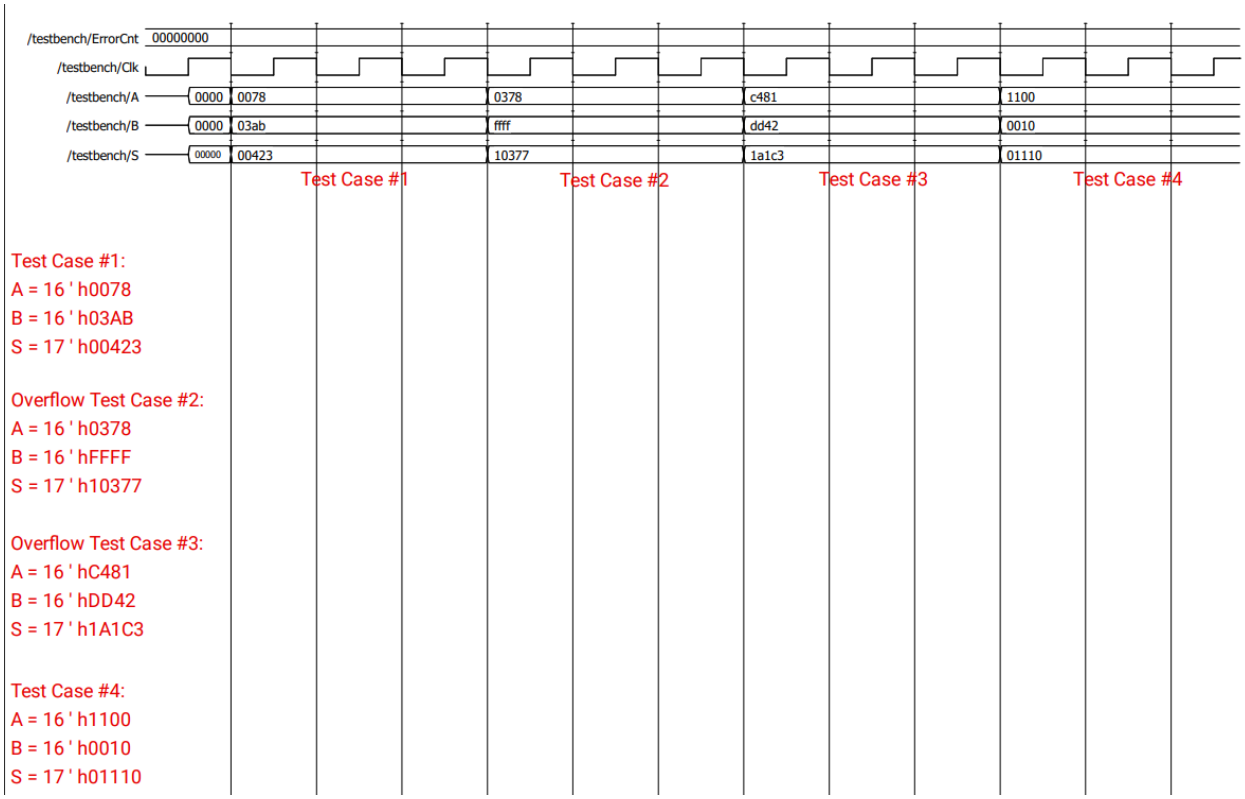

Figure 19: Annotated Simulation Waveform

Conclusions:

In conclusion, this experiment demonstrates that there are design tradeoffs for any digital logic design. Specifically, the adders each different in their characteristic area, complexity, and speed. When designing each of the three adders, bugs in the SystemVerilog code such as missing/incorrect expressions and incorrect port linking between modules occurred. To combat these small problems, we devised a custom test bench with the same test cases to simulate all three adder designs. The diagrams provided in the lab manual were clear and concise as well as the instructions throughout the documentation were well written.