ECE 385

Fall 2019

Experiment #4

Introduction to SystemVerilog, FPGA, EDA, and Adders

Xinglong Sun Churan He
ABD T 8AM
Mihir Iyer

Introduction (High level function of the circuit)

a. Logic Processor

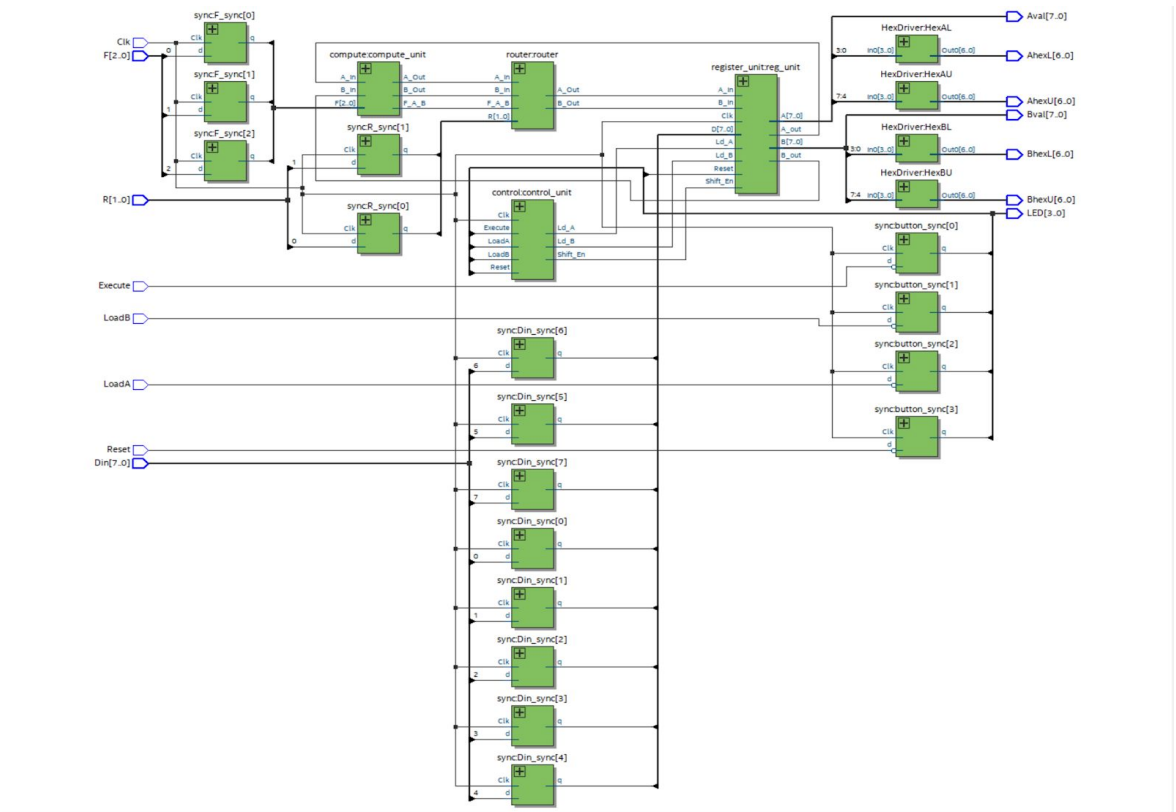
The 8-bit serial processor is very similar to the logic processor we designed in the previous lab. It's able to perform bitwise logical operations on two values stored in two 8-bit shift registers. The logical operations that can be performed by it include NAND, NOR, XNOR, 0, AND, OR, XOR, and 1.

b. Three types of adders

The 16 bit binary adder is able to take two 16-bit data values and compute their sum. User can input the values from the switches on the FPGA board. The two input values will be shown in decimal on the board LED Display, and the sum will be shown by the LEDs once the user hit the RUN button. In this lab, we design three types of adder: Carry Ripple Adder (CRA), Carry Lookahead Adder (CLA), and Carry Select Adder (CSA). They all accomplish the same high-level tasks, but each adder is of different architecture and performance.

Part 1 - Serial Logic Processor

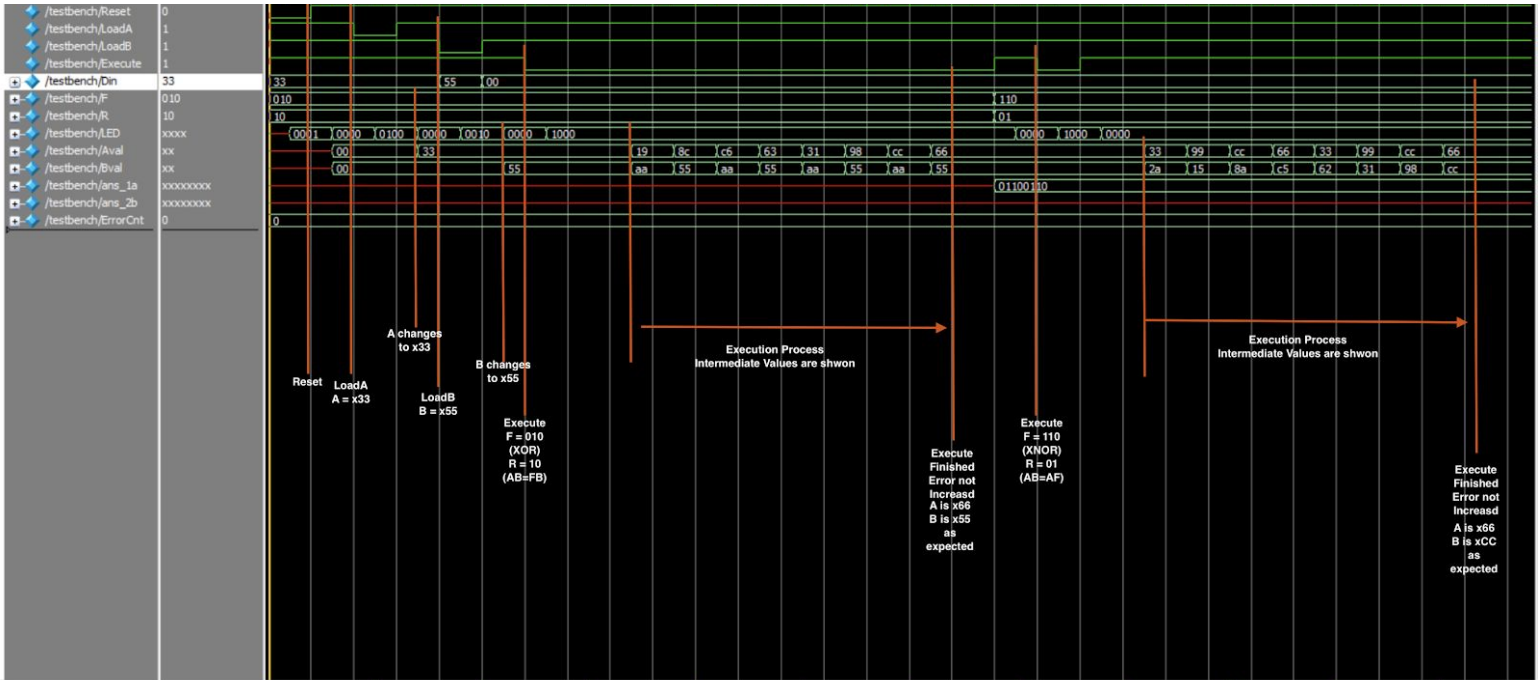
a. Block Diagram



b. Modifications Taken on the 4-bit Processor

In order to extend the 4-bit Processor to a 8-bit Processor, there are mainly two major modifications taken. We first need to enlarge our 4-bit register to a 8-bit one in order to properly store the 8-bit data values and computation result. This is achieved very easily by replacing four bit signal vectors([3:0]) into eight bit ones([7:0]). Another important modification we need to make is to increase the states of the FSM. Previously, we had four states acting as a counter to count the four serial shift our circuit needs to perform. After transforming the circuit into a 8-bit one, the number of our "Count" states also needs to become eight. We then increase the number of HexDrives which allows FPGA board to display more decimals.

c. Annotated Simulation Waveform



Part 2 - Adders

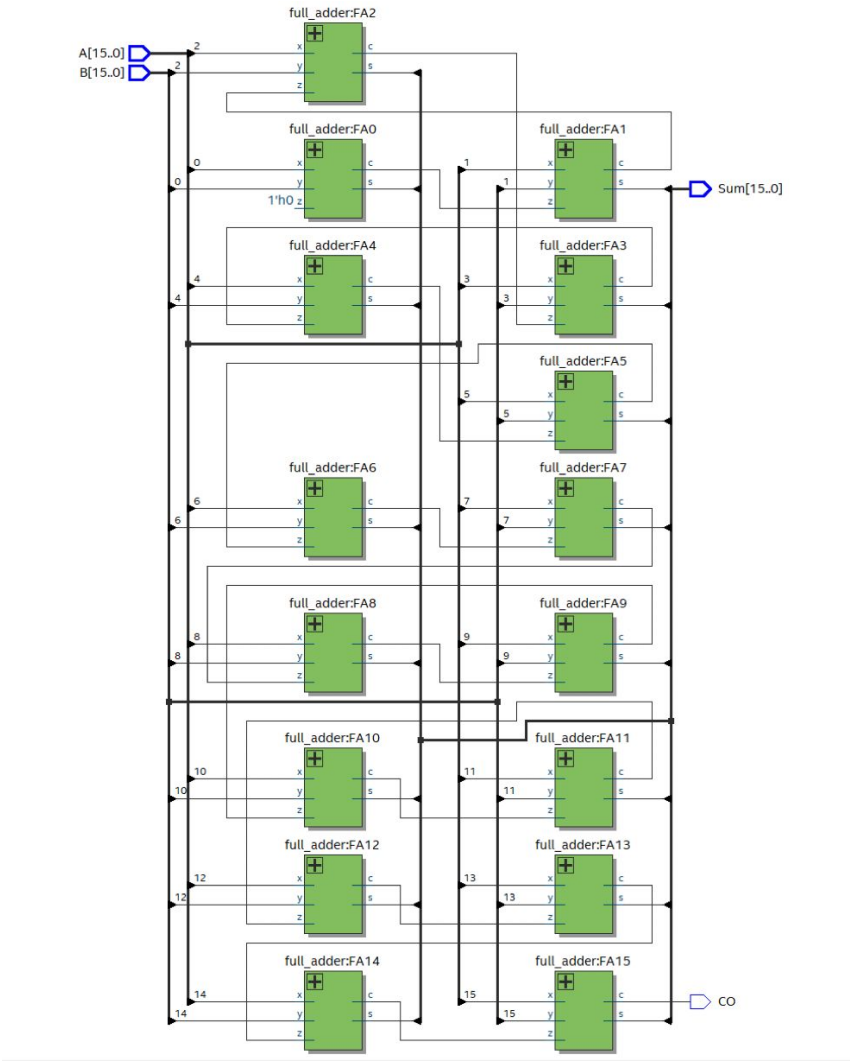
a. Carry Ripple Adder

i. Written Description of Architecture

In this part of the lab, we use SystemVerilog to implement a 16-bit Carry Ripple Adder (CRA) on our FPGA board. The 16-bit Carry Ripple Adder contains 16 Full Adder Units connected in a series, each for 1 bit. For these 16 Full Adder Units, the previous unit's carry-out bit would be the next unit's carry-in bit. Therefore, each individual bit in the output result is the

sum of the specific bit from input A, B and also the carry-in bit. In this case, the carry-in bit would influence the output result, and then in turn influence all the more significant bits. The output result would be valid from the right-most bit, one by one, to the left-most bit (most significant). This design uses the least number of logics, but takes the longest time for the valid result.

ii. Block Diagram.

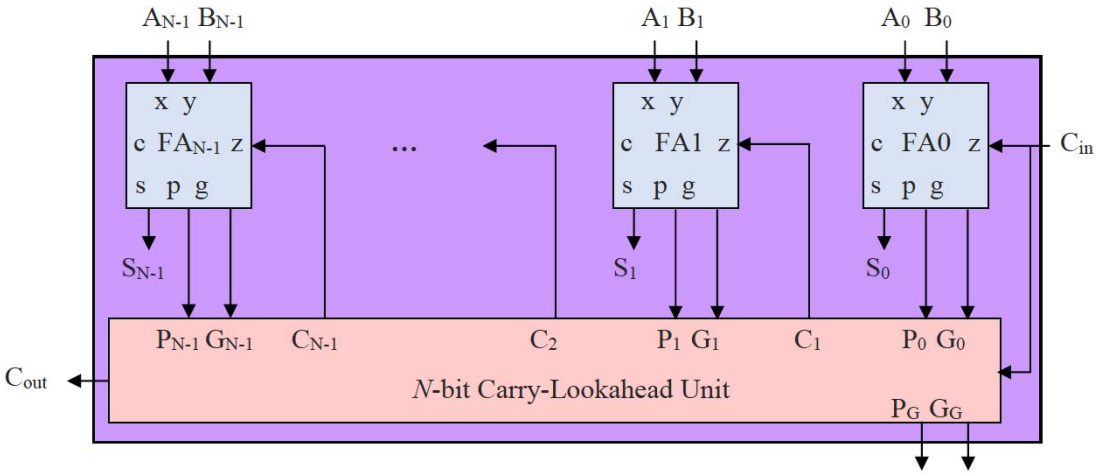


b. Carry Lookahead Adder

i. Written Description of Architecture

Instead of the naive Carry Ripple design, we can make predictions for the carries of each adder based on the immediate available inputs (A and B). Therefore, instead of all the full adders

being connected serially, each full adder's carry-in port is connected to a Carry-Lookahead Unit that determines the carries for each adder. How we accomplish it is explained in detail in the next subsection. The following diagram is a high level structure of CLA:



ii. P and G

In order to calculate carry-ins immediately as the inputs are fed in, we are going to find some relation between the inputs A,B and the carry-in C. Suppose we have a full adder, with two operands denoted as Ai and Bi, carry in denoted as C_in, and carry out denoted as C_out. Through a careful observation and analysis, we found that when Ai and Bi are both 1, no matter what C_in is, C_out is always 1. When either one of Ai and Bi is 1, C_out is the same as C_in. When both Ai and Bi are 0, C_out is always 0. To transform the above description into elegant boolean algebra expressions, we use two intermediate variables, Generate($G(A,B) = A \,.\, B$) and Propagate($P(A,B)=A \; XOR \; B$). Therefore,

$$C\_out = C\_in \,.\, P + G$$

For a 4-bit adder, carry-ins can be shown in expansion as:

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

iii. Hierarchical Design

A very critical limitation of CLA design is that the number of gates involved will grow quickly to a very large number as N(input operand bits) increases. As a concrete example, by merely observing our four-bit adder above, we can see that the expression of *C3* has already involved lots of logic. Therefore, if we just follow the "Flat" approach using the explicit expansion of *Ci*, our design will soon become impractical. A hierarchical design is necessary in this case to simplify the logic and enhance the circuit performance. In this lab, we choose to adopt a 4 x 4 hierarchical design.

The 16-bit inputs A and B are divided into groups of 4 bits which are fed into four 4-bit CLAs. We define two additional variables group propagate(*P_G*) and group generate(*G_G*) as:

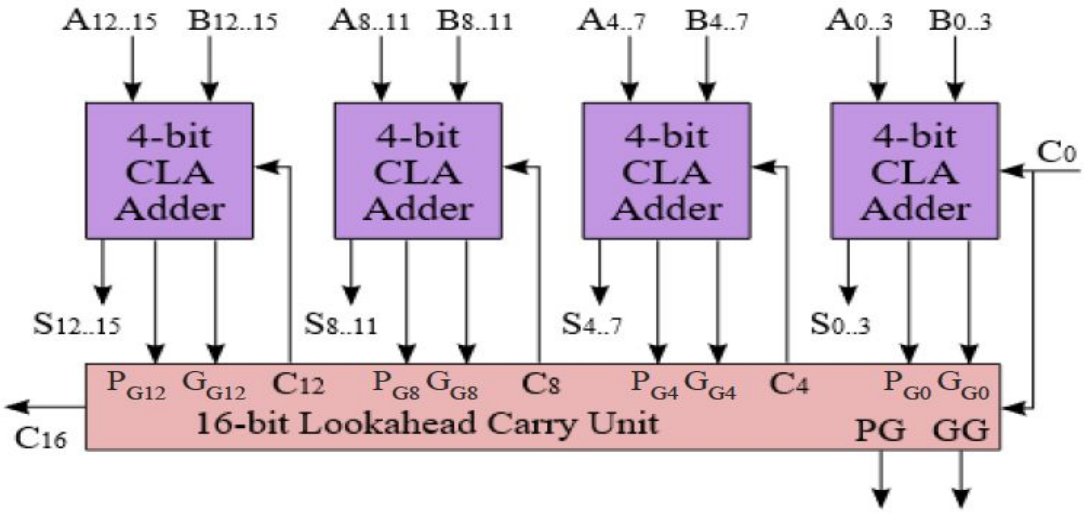
$$P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$
$$G_G = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

The meaning of these two variables is that, if *P_G* is equal to 1, then C_in of our current 4-bit CLA unit will be propagated to the C_in of the next 4-bit CLA unit; if *G_G* is equal to 1, then C_in of the next 4-bit CLA unit is 1.

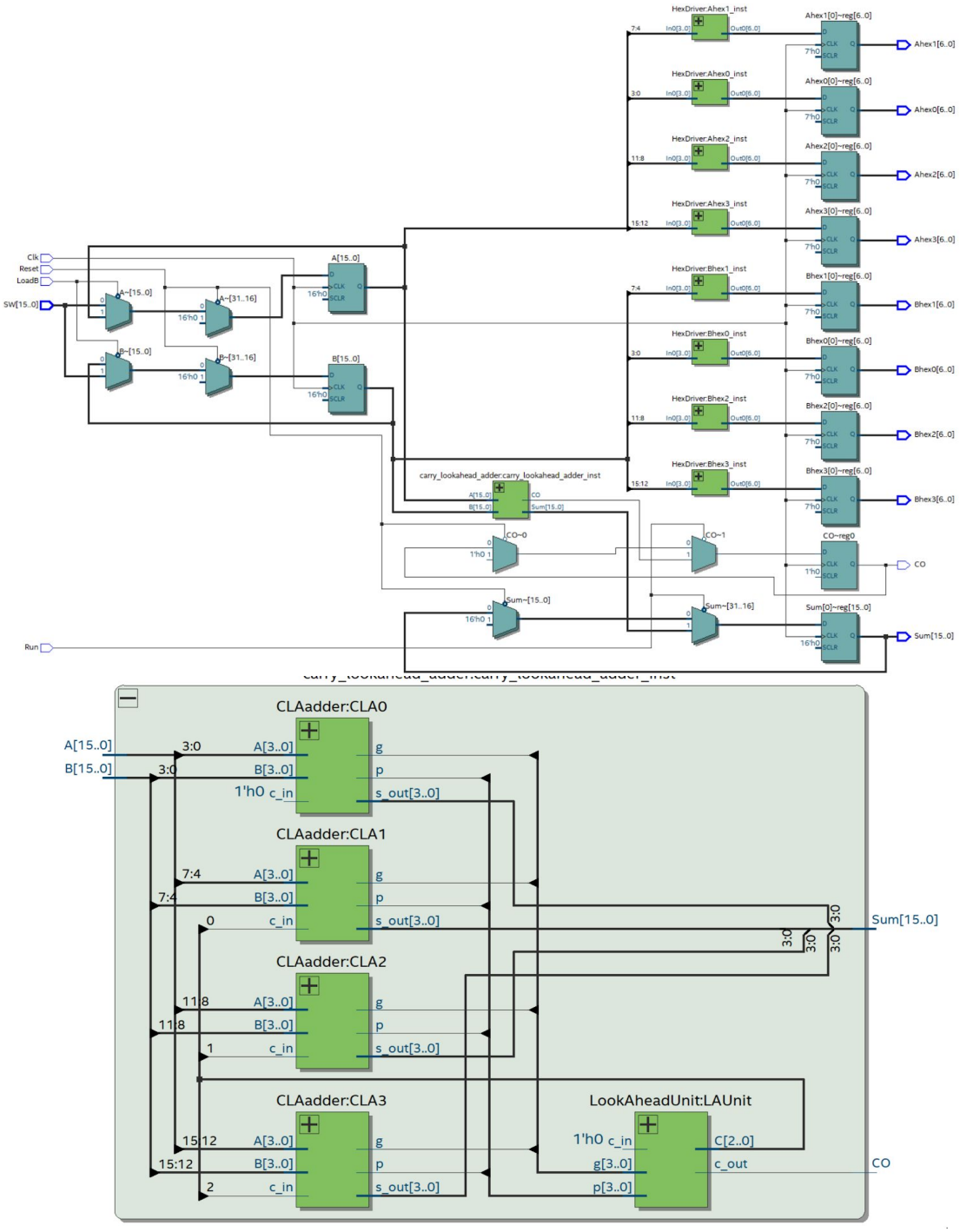
Given these two variables, instead of connecting four 4-bit CLA units serially (which is essentially a high-level ripple design), we can feed the *P_Gs* and *G_Gs* into another Lookahead Unit, as shown by the formulas below,

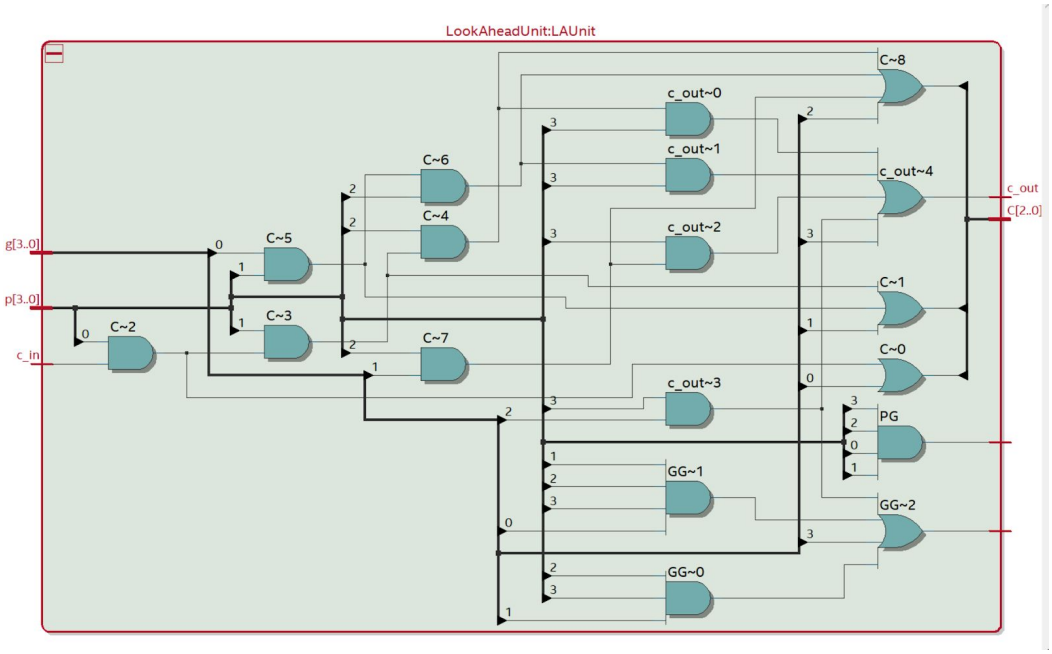
$$C_4 = G_{G0} + C_0 \cdot P_{G0}$$
$$C_8 = G_{G4} + G_{G0} \cdot P_{G4} + C_0 \cdot P_{G0} \cdot P_{G4}$$
$$C_{12} = G_{G8} + G_{G4} \cdot P_{G8} + G_{G0} \cdot P_{G8} \cdot P_{G4} + C_0 \cdot P_{G8} \cdot P_{G4} \cdot P_{G0}$$

An architecture diagram is shown as follows:

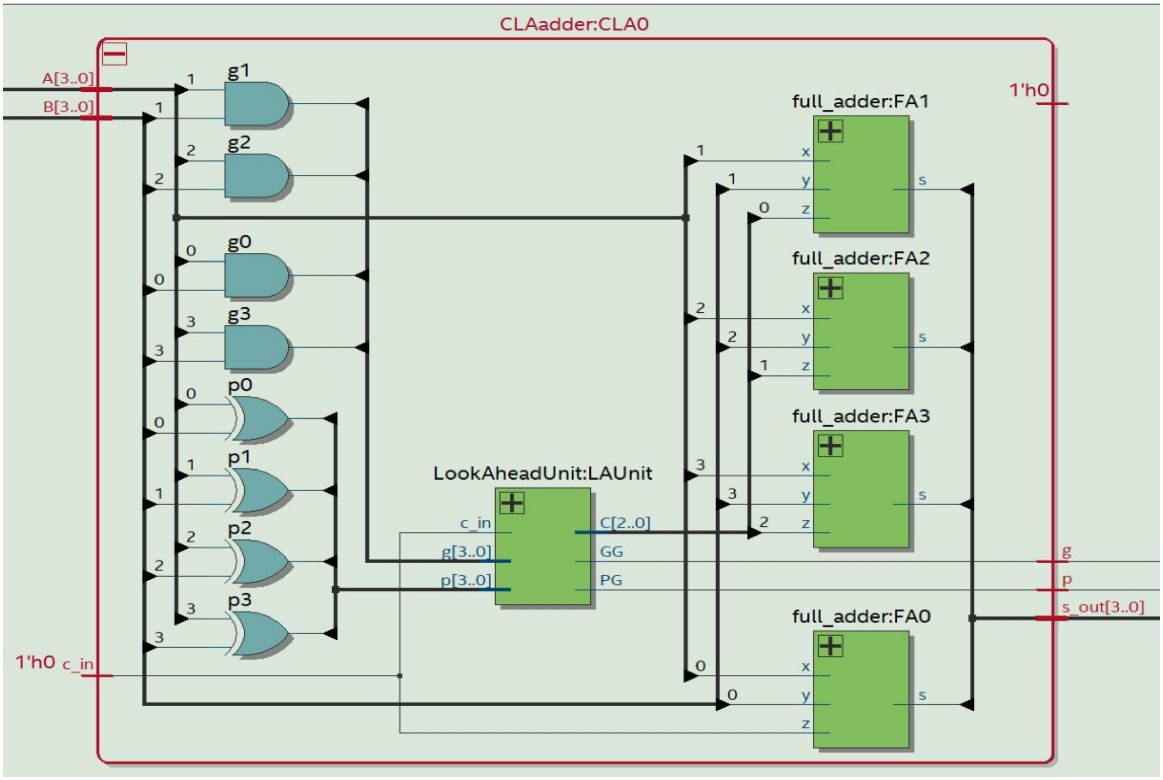


iv. Block diagram





v. Block Diagram inside a single CLA (4-bits)



c. Carry Select Adder

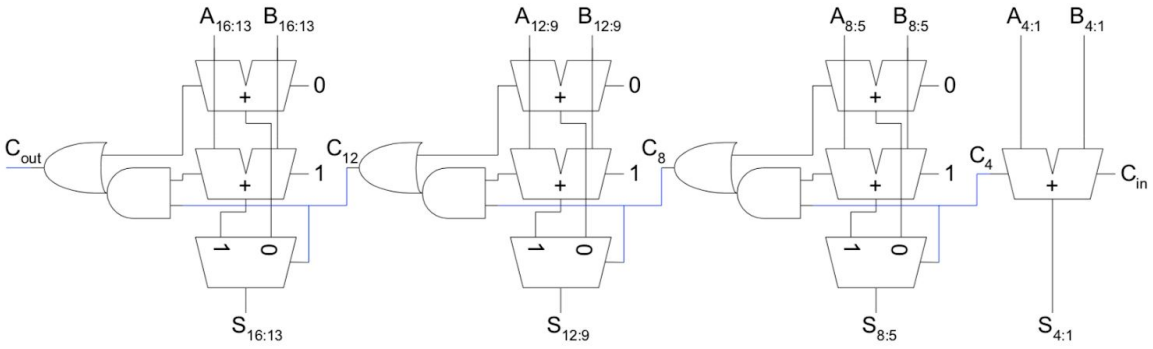
i. Written Description of Architecture

The function of the Carry Select Adder (CSA) is to calculate the results of a group of consecutive bits with that the carry-in bit is 0 and 1 at the same time, then the carry-in bit of the previous would control the MUX to select the valid value. In this lab, we build a 16-bit Carry Select Adder with a 4 * 4 bits hierarchical structure, with one 4 bits Carry Ripple Adder Unit for the right-most 4 bits and three 4 bits Carry Select Adder Unit for the other bits. Each Carry Select Adder Unit also contains two Carry Ripple Adders, the logic to determine its carry-out bit, and the MUX to select the output result.

ii. How CSA selects value

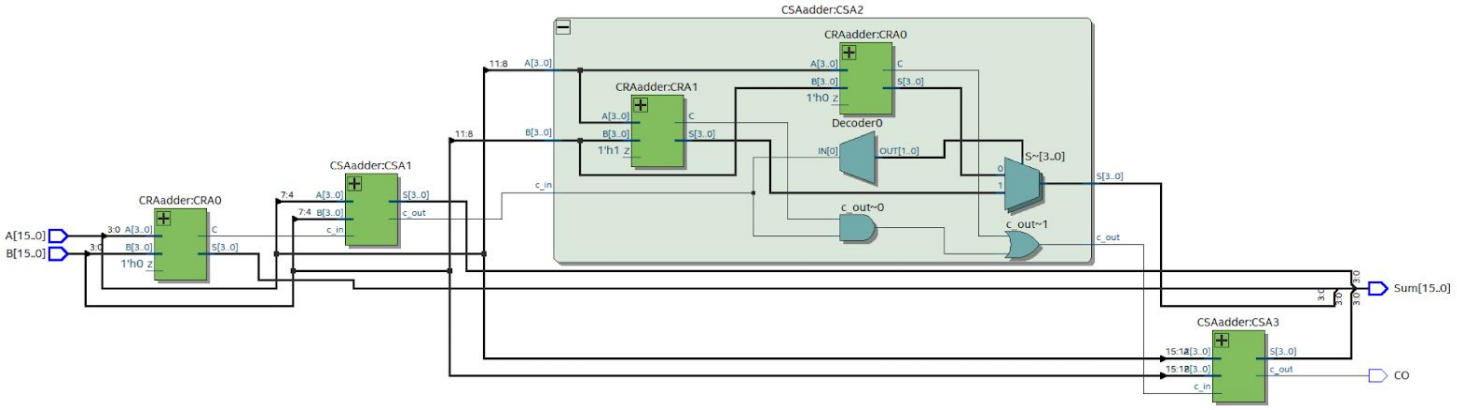
During the operation, the 7 Carry Ripple Adders can calculate the result at the same time, then the carry-out from the adder unit on the right could select the valid result with a MUX based on the carry-in value. The MUX's "0" input is the result from the Carry Ripple Adder Unit with carry-in 0, and the "1" input is the result from the unit with carry-in 1, and the select signal is the actual carry-in value. The carry-in value would also be a determined factor of the carry-out of this Carry Select Adder Unit.The carry-out of a Carry Select Adder Unit is

(C with carry-in 0) OR ((C with carry-in 1) AND (Carry-in))



With this logic (the glue logic), the carry-out of the Carry Select Adder Unit would be able to select the value of the next Carry Select Adder Unit on its left.

iii. Block Diagram



d. Comparison of Three Types of Adders

In terms of area: CRA and CLA use the same number of full adders. However, CLA has much more logic gates in its Lookahead Unit. Therefore, CLA takes more space than the CRA. CSA uses almost twice the number of full adders, and there are also additional MUXes and gates between different units. Therefore, among these three, CSA occupies the largest area. The same analysis here also applies to the power of three types of adder. CRA should take the least amount of power, and CSA's power consumption should be much more than that of CRA and is the biggest among the three..

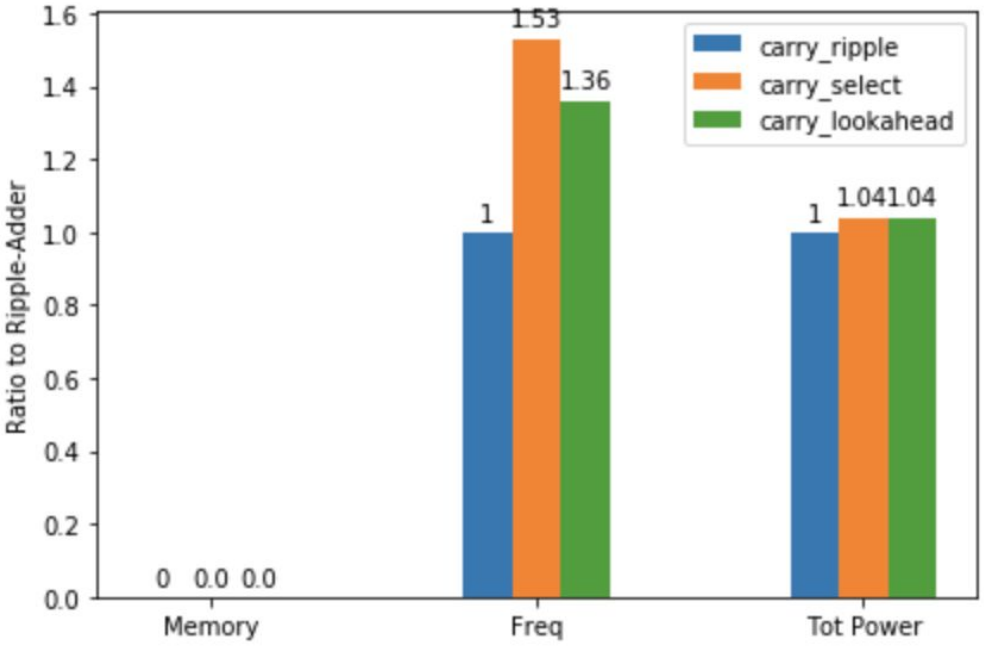
In terms of complexity: The carry ripple design is the most straightforward and simplest among the three. Between the other two, on a high level, CLA's logic is more complex than that of CSA. There are many Boolean Algebra computations taken in CLA's Lookahead Unit to predict the carry-ins, whereas the underlying principles of CSA are based on MUXes to select from two sources of values.

In terms of performance: CRA has the longest gate delay due to its carry rippling nature. The current full adder has to wait for the previous one to finish computation and use the carry-in generated by it to start its own computation. Performance of CLA and CSA are much more superior than that of CRA. CSA is supposed to perform slightly better than CLA, because after the actual computation of the first 4-bit CRA, all the CSA does is to select different values based on the carry_ins.

e. The performance of each adder

	Carry-Ripple	Carry-Select	Carry-Lookahead
Memory	0	0	0
Frequency	62.25MHz	95.18MHz	84.72MHz
Total Power	156.09mW	161.80mW	162.92mW

Data Collected with 50 MHz .sdc File



Answers to the 3 Post-lab Questions

1. Compare the RTL design and the TTL design of the 8-bit processor

The difference between these two designs is the type of finite state machine. In the RTL design, we implement the 8-bit processor by the Moore State Machine, and in the TTL design, we use the Mealy State Machine. Because Mealy State Machine's output can depend on its input, we need only two states to implement the processor. By contrast, the Moore State Machine in the RTL design uses 10 individual states.

Each of the design has its own benefits. The Mealy State Machine only has two states, so the state and transition design could be easier, and then it could reduce the logic gates for gate transition. The Moore State Machine's output only depend on the state, so it can save the time for evaluating the output based on the input comparing to the Mealy State Machine, and therefore achieve a higher performance. The designer should choose the specific design

depending on the actual needs. As for this processor, performance is relatively more important, so using the RTL design with the Moore State Machine is an appropriate choice.

2. **In the CSA for this lab, we asked you to create a 4x4 hierarchy. Is this ideal? If not, how would you go about designing the ideal hierarchy on the FPGA**

In terms of performance, a 4 x 4 hierarchical design is not necessarily the best. Unlike the CLA, in CSA, there's no involved Boolean Logic that get too onerous to be practical when a "Flat" design is adopted. Notice that the primitive 4-bit adder unit we use in the 4 x 4 CSA is CRA. Suppose we replace CRA with a 4-bit CSA, after C_out of the **first 1-bit full adder** is computed out, the circuit just does a series of selections from the pre-calculated sums and carries of the later full adders. Compared to the 4 x 4 CSA where the circuit performs selection after the result of the **first 4-bit CRA** unit is computed out, flat design's overall performance should be increased. Therefore, a 8 x 2 hierarchical design or merely a 16 x 1 flat design could perform better than the 4 x 4 one. However, the selection process in CSA is also a rippling process. Though the actual computation time is shrunk, the time taken to perform the selections is actually increased. Therefore, the tricky part of it is to balance the "Computation Time" inside the basic adder unit and the rippling "Selection Time" to achieve an optimal result. To explore this further, we need to experiment it on the FPGA. We will design different hierarchies and observe the Maximum Operating Frequency on Quartus which might give us some hints of which design is more superior.

3. **Design Resources and Statistics**

CRA:

LUT	114
DSP	0
Memory (BRAM)	0
Flip-Flop	105
Frequency	62.25MHz
Static Power	98.55mW
Dynamic Power	3.09mW
Total Power	156.09mW

CLA:

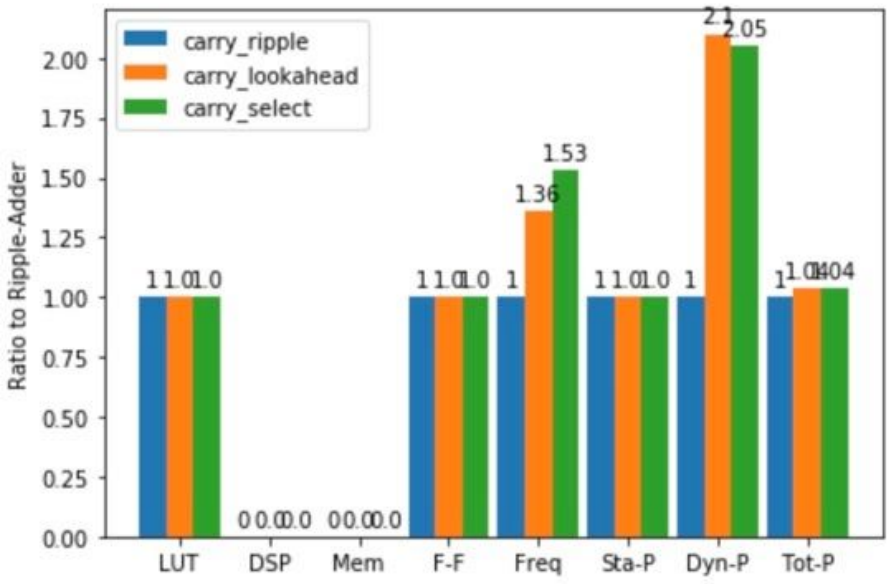
LUT	128
DSP	0
Memory (BRAM)	0
Flip-Flop	105
Frequency	84.72MHz
Static Power	98.57mW
Dynamic Power	6.48mW
Total Power	162.92mW

CSA:

LUT	123
DSP	0
Memory (BRAM)	0
Flip-Flop	105
Frequency	95.18MHz
Static Power	98.57mW
Dynamic Power	6.34mW
Total Power	161.80mW

Data Collected with 50 MHz .sdc File

Overall Comparison



For the number of LUTs used, CRA uses 114, CLA uses 128, and CSA uses 123. It meets our expectations based on the design complexity that the CRA uses the least number of LUTs and the CLA uses the most number of LUTs (more logic in determining Ps and Gs).

For the Fmax, CRA can go up to 62.25MHz, CLA goes to 84.72MHz, and CSA goes to 95.18MHz. It also meets our expectations that CSA's the fastest in getting the valid result, because CSA can calculate every 4-bit group parallelly and doesn't require complex logic to determine carry-in bits. The CLA theoretically can calculate every bit parallel, but the complex logic for P and G would significantly drag the overall speed. The CRA is the slowest due to its serial adders. Every full adder on the left need to wait for the carry-in bit from the right.

However, we also get some results that don't meet our expectations. In the simulation, the power consumption of the CLA is higher than CSA. We expect the CSA consumes the most power, because it contains 28 full adders, MUXes and glue logics, which is more than the CLA. If we implement these designs in TTL, more components means more chips are required, which would in turn be reflected in the power consumption. However, we use LUTs in FPGA, not chips. Complex logics determining Ps and Gs in CLA may utilize more LUTs, which would be reflected in the power consumption. The reason of our simulated results like this may be the different behavior of transistors logic and LUTs, and the design optimization by Quartus. Quartus may help us find the most efficient way to implement the circuit path in the background.

Conclusion

In this lab, we got more familiar with System Verilog, FPGA, and Altera Quartus. We chose to do this lab after a careful read and review of the SystemVerilog and Quartus Introduction. Therefore, we finished this lab pretty quickly and didn't encounter many problems. We also got more familiar with testing the simulation circuit using a testbench. Moreover, we explored different types of adder and did a thorough analysis of their features and comparison.