ECE 385 Lab Report

Spring 2023

Lab4: Introduction to SystemVerilog, FPGA, CAD and 16-bit Adders

Name: Zhu Hanggang, Hong Jiadong

Student ID: 3200110457, 3200110970

Prof. Chushan Li, Prof. Zuofu Cheng
ZJU-UIUC Institute
April 6, 2023

1 Introduction

The 8-bit Serial Logic Processor implementation is to modify the former 4-bit Serial Logic Processor code. There is two main change for the 8-bit implementation. The first change is the change of the register structure, the second change is the Finite State Machine that controls the register and computation unit.

The three adders' main functionality is to add two 16-bit values and output the sum. The ripple adder is simply connected by 16 1-bit full adders in series. The Carry-Lookahead Adder supports the calculation of P, G, which is used to output carry bits. The Carry-Select Adder supports the functionality of selecting between two sums.

2 Part 1 - Serial Logic Processor

2.1 High-level RTL schematic

We complete the IQT 8-bit Serial Logic Processor implementation by modifying the former 4-bit Serial Logic Processor code. There is two main change for the 8-bit implementation. The first change is the change of the register structure, the second change is the Finite State Machine that controls the register and computation unit.

We mainly implement a new eight-bit register module and change the corresponding I/O functions, the Data In and Data Out is working properly in the 0-160 nanoseconds simulation, the result is shown in the figure1

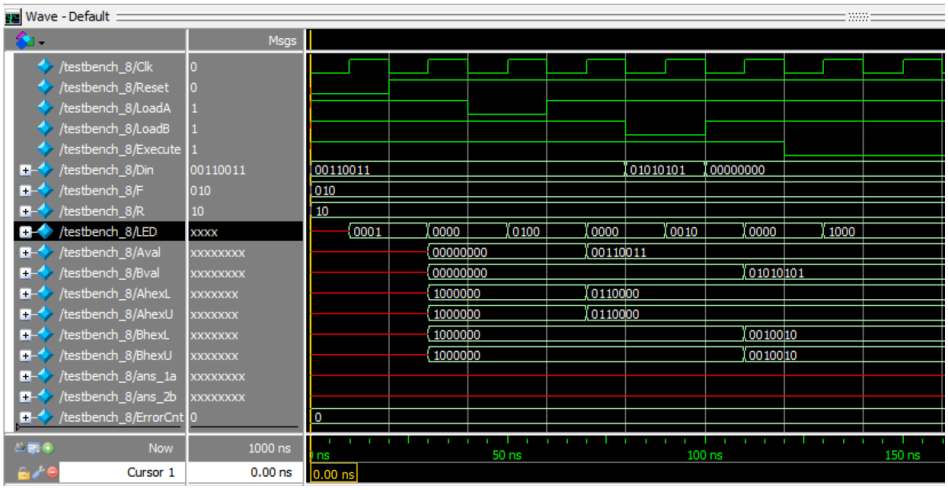

Figure 1: 0-160ns Wave Form

For the second important change in FSM design, the main difference is that we should increase the state number from 4 to 8 to execute the whole 8-bit calculation.

See figure2.

2.2 Code Implementation

For the 8-bit processor implementation, we should mainly implement two additional logic, the first one is the data structure of the 8-bit shift register.

The second main part is the control unit of the Finite State Machine, in which we should implement four more states to completely implement the 8-bit computation.

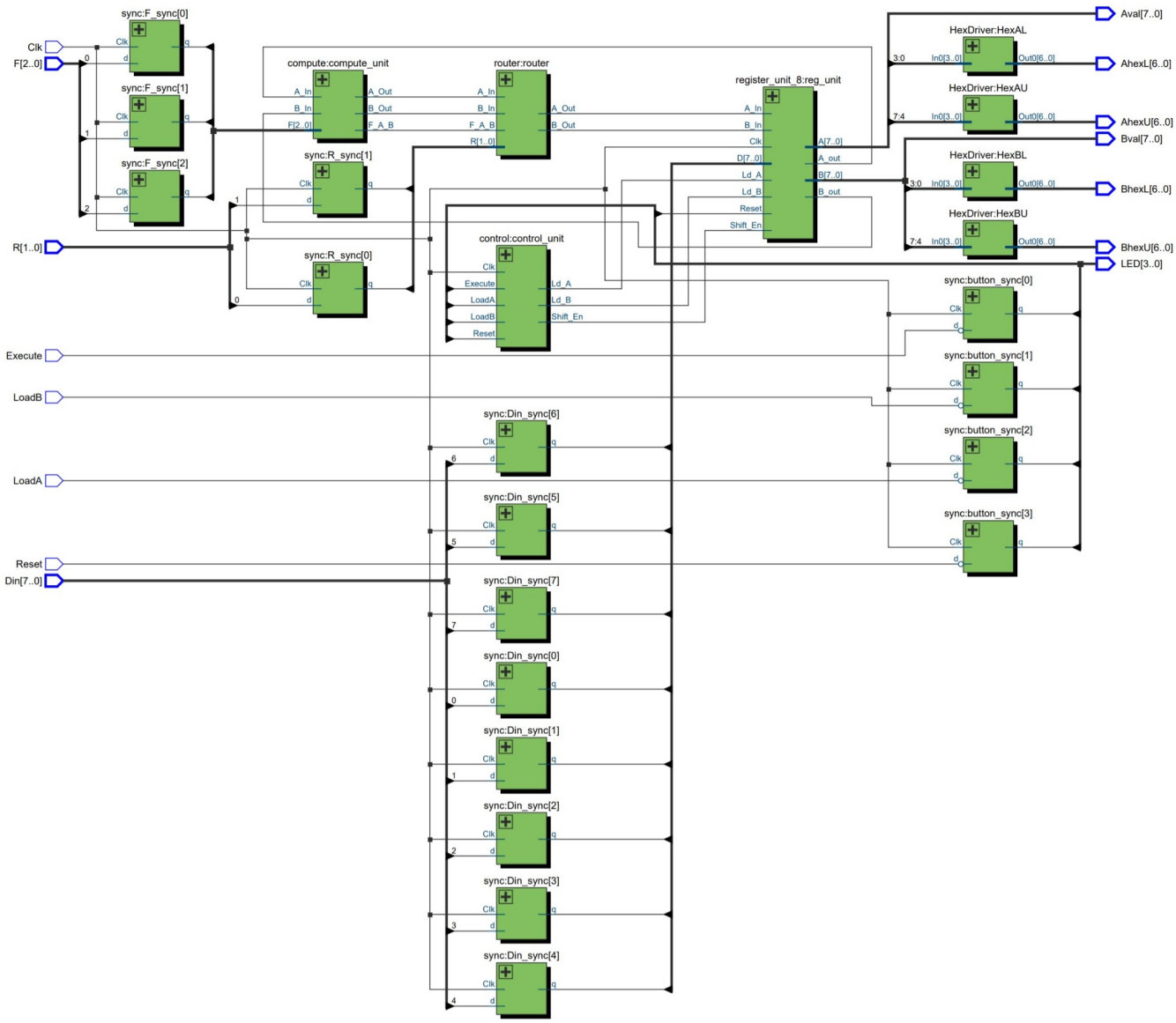

Figure 2: RTL Schematic For 8-bit Logic Processor

2.3 Simulation of the Computation Process

The computation control logic for function output and router output are the same as previous lab. There's no big change here.

The output of the outcome will be stored in the Aval in this example. As you can see in the figure 3. As you can see in the figure 3 and the figure 4, the XOR calculation is executed correctly for all the values in the shift register when the F2-F0 is 010.

The 8-bit logic processor computation control works properly when the F2-F0 changed to 010. The XNOR output is correctly stored in Aval as you can see in the figure 5.

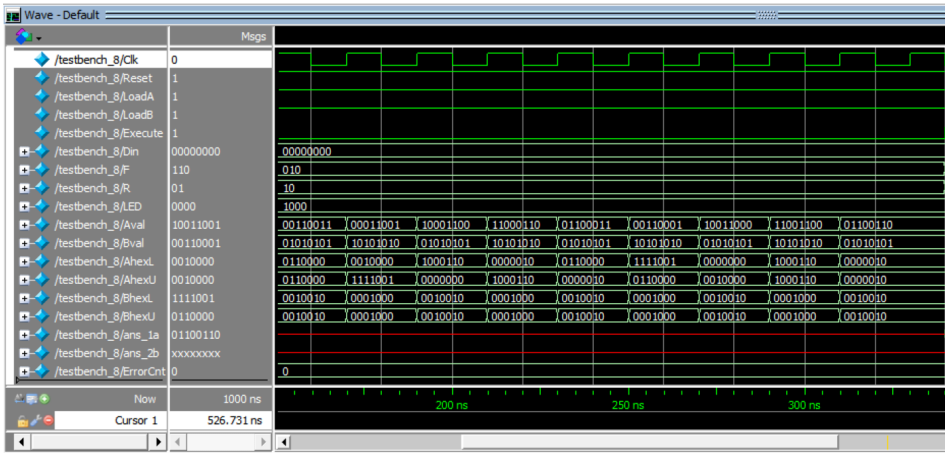

Figure 3: 150-340ns Wave Form

Figure 4: 340-560ns Wave Form

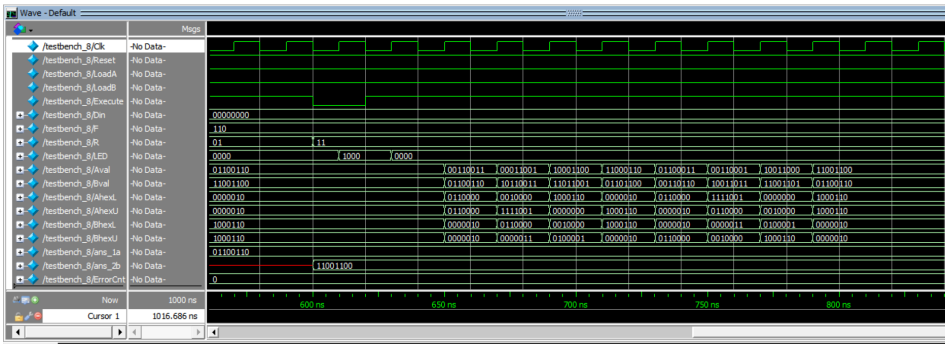

Figure 5: 560-840ns Wave Form

3 Part 2 - Adders

3.1 Ripple Carry Adder

The basic unit of the ripple-carry adder is a 1-bit full adder. Link 16 1-bit full adder together in series through the carry bits The carry-out of a full adder is connected to the carry-in of another full adder. By doing this, The 16-bit ripple-carry adder is completed.

The block diagram of the ripple-carry adder is shown in figure 6. One should notice that it's simply 16 1-bit full adders wired in series.

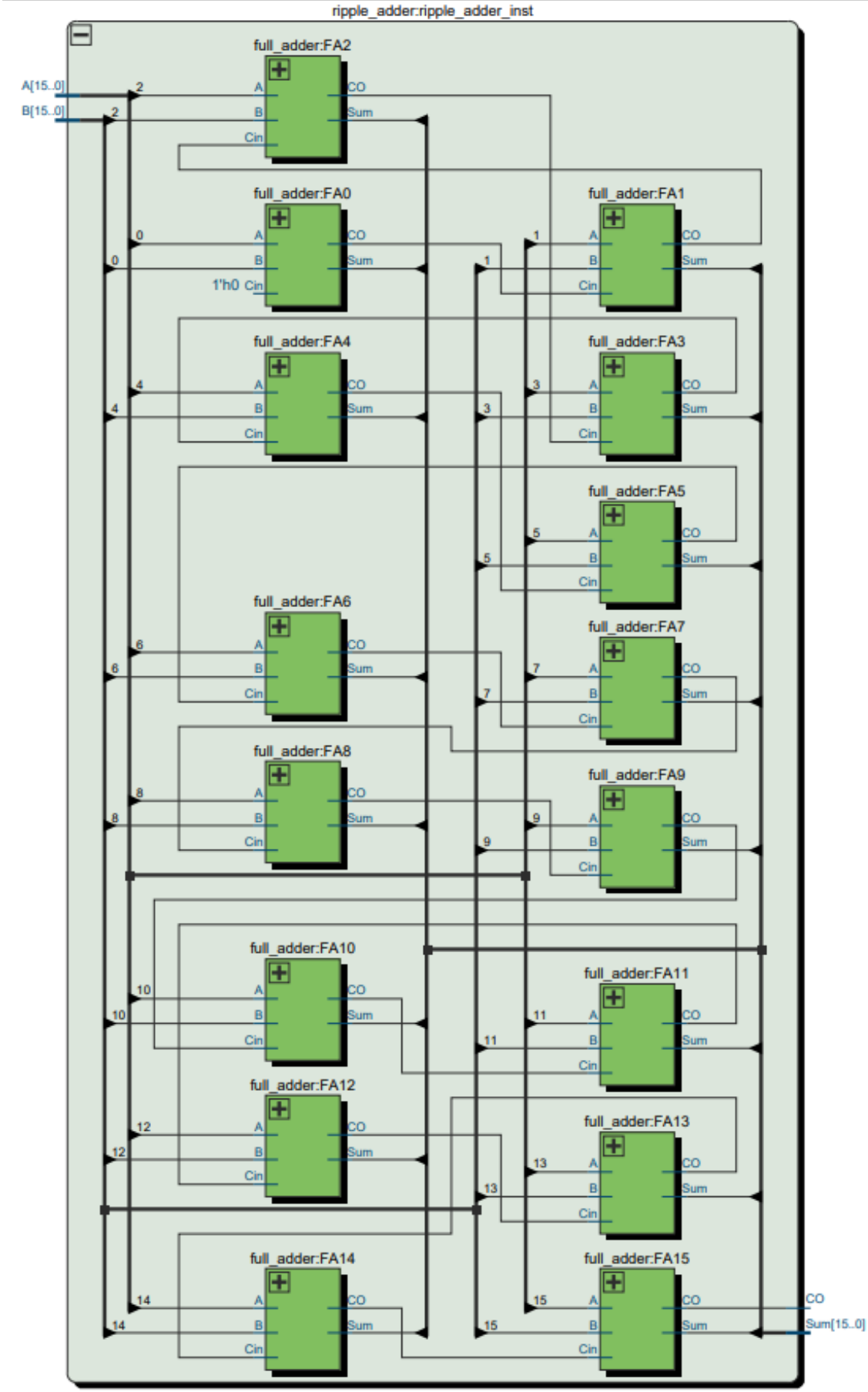

Figure 6: Ripple Carry Adder

3.2 Carry Lookahead Adder

The carry lookahead adder calculates all carry bits without depending on any output from another adder. The basic architecture contains 4-bit carry lookahead module (used to calculate the sum) and the carry-lookahead unit module(used to calculate carry bits). Then create a 16-bit adder by using 4 4-bit CLA. The rippling of 4-bit CLA is avoid by using a similar carry-lookahead module in 16-bits. So one can see the design is hierarchical.

In order to create teh hierarchical 4x4 adder, The following four modules are implemented.

1. **Module:** carry_lookahead_adder4
   Inputs: [3:0]A,[3:0]B, Cin
   Outputs: [3:0]Sum,CO,PG,GG
   Description: Calculate the sum of 4 bit input, without carry-in bits depending on another full-adder. Calculate P and G for the 4-bit carry lookahead unit. It consists of four 1-bit full adders and one 4-bit carry lookahead unit.
   Purpose: To act as a 4-bit adder.

2. **Module:** carry_lookahead_unit4
   Inputs: [3:0]P,[3:0]G,Cin
   Outputs: [3:0]C, PG,GG,CO(unused)
   Description: Take P and G as input and provide carry bits for the 4-bit carry-lookahead adder.
   Purpose: Provide 4 carry bits.

3. **Module:** carry_lookahead_adder
   Inputs: [15:0] A, [15:0] B
   Outputs: [15:0] Sum, CO
   Description: calculate the sum of 16-bit input, without carry-in bits depending on another adder. It consists of four 4-bit carry-lookahead adders and one 16-bit carry-lookahead unit.
   Purpose: The complete 16-bit carry-lookahead adder

4. **Module:** carry_lookahead_unit16
   Inputs: [3:0] PG, [3:0] GG, Cin
   Outputs: [3:0] C4
   Description: take PG and GG from 4-bit carry-lookahead adder as inputs and provide $C4, C8, C12$ carry bits for 16-bit carry-lookahead adder.
   Purpose: Provide carry bits $C4, C8, C12$

P and G logic are generated within the 4-bit carry lookahead adder module using the logic expression $P = A \oplus B$ and $G = AB$. They are passed to the 4-bit carry lookahead unit, in which each carry-bits are calculated as well as two additional outputs PG and GG. Besides, the carry-out bit of the 4-bit adder is also generated for completeness, but they are not used as we don't ripple carry bits here. To calculate the carry bits, we use the logic expression provided in the lab manual as follows. The carry bits C_4, C_8, C_12 are evaluated similarly using PGs and GGs.

$$C_0 = C_{in}$$
$$C_1 = C_{in}P_0 + G_o$$
$$C_2 = C_{in}P_0P_1 + G_0P_1 + G_1$$
$$C_3 = C_{in}P_0P_1P_2 + G_0P_1P_2 + G_1P_2 + G_2$$

The diagram of a 4-bit CLA is shown in figure 7. Notice how the 4-bit carry lookahead unit provides the carry-in bit for the full adders. Figure 8 shows the diagram of the carry lookahead adder. One can see that each 4-bit CLA is connected in parallel. The carry-in bit is from the 16-bit carry lookahead unit.

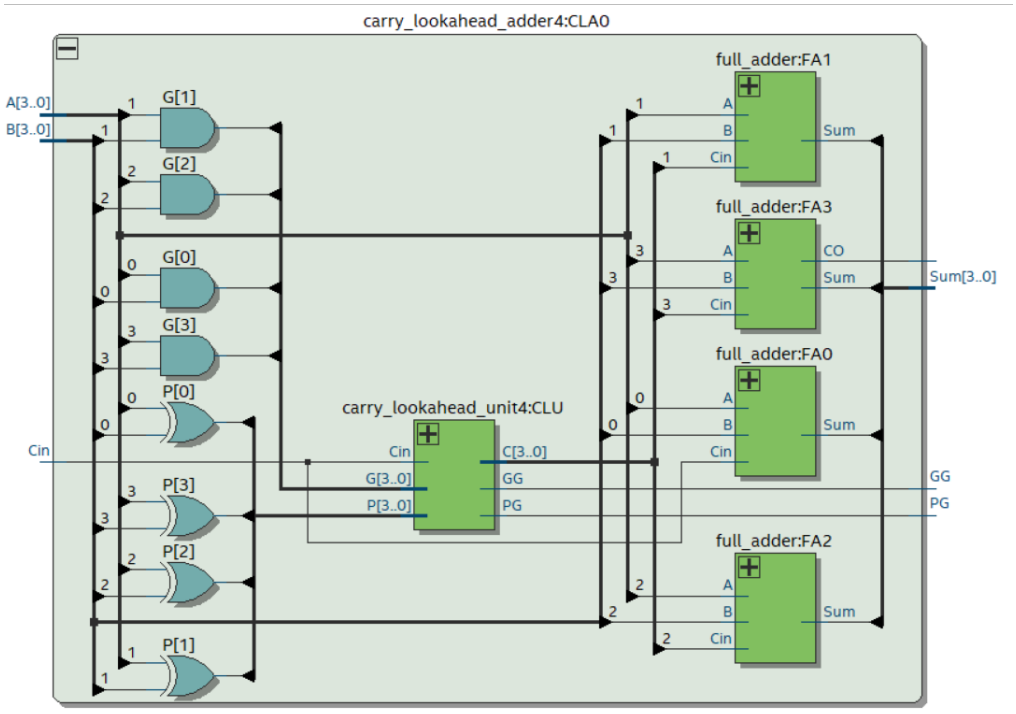

Figure 7: 4-bit Carry Lookahead Adder

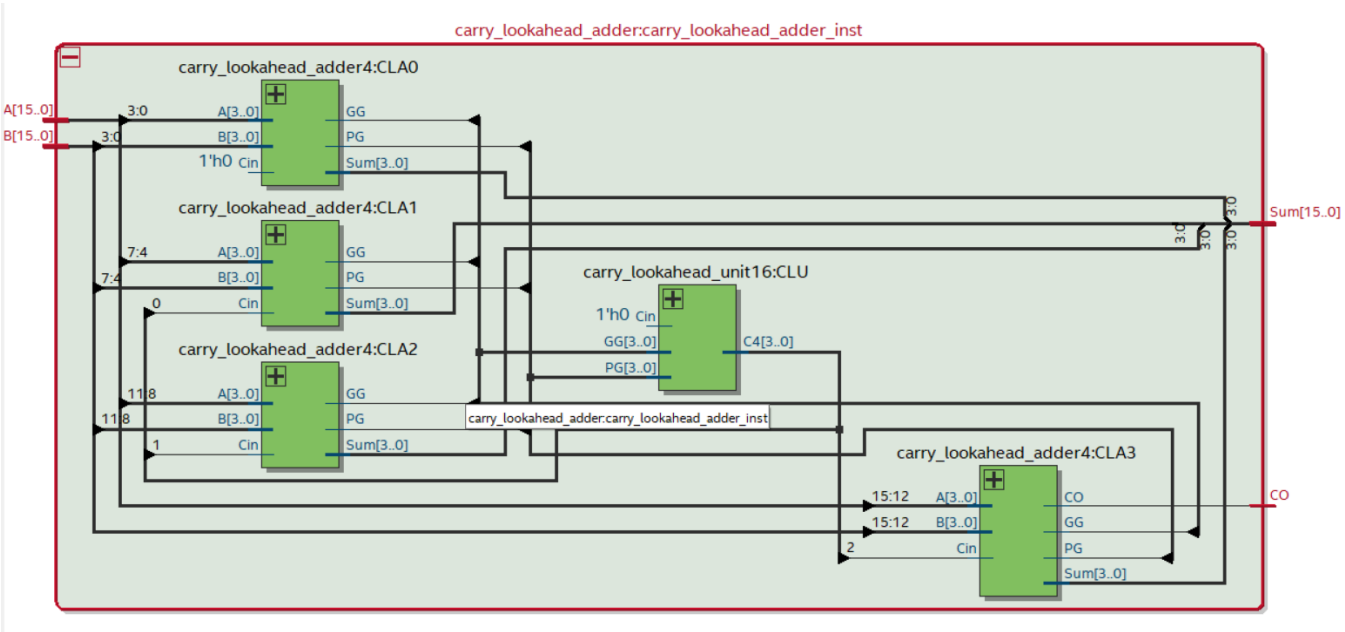

Figure 8: Carry lookahead adder

3.3 Carry Select Adder

The 16-bit carry select adder consists of 7 4-bit full adders. One to compute the lower four bits of sum, three to compute the higher 12 bits of sum with carry in bit 0, and the last three the compute higher 12 bits of sum with carry bit 1. The final sum is decided based on the carry-out bit of former 4-bit adders with some logic combination.

For our CSA, the seven full adders can compute the corresponding sum and carry out bits in parallel, without one interference from another. Based on the carry-out bit, we can choose between two sums. The selection criterion is based on C_4, C_8, C_12, where

$$C_4 = C_{in}$$
$$C_8 = C_4 Cout1[1] + Cout0[1]$$
$$C_{12} = C_8 Cout1[2] + Cout0[2]$$

where, $Couti[2]$ means j^{th} full adder with carry0in bit i. This makes sense because if C_4 is zero, the C_8 will only depend on $Cout0[1]$, as $Cout1[1]$ is actually not going to be used. If C_4 is one, $Cout1[1]$ will possibly affect C_8 as well.

Figure 9 shows the diagram of carrying select adder. One can see all 7 full adders compute in parallel and there are some logic gates and MUXs to determine what the sum is.

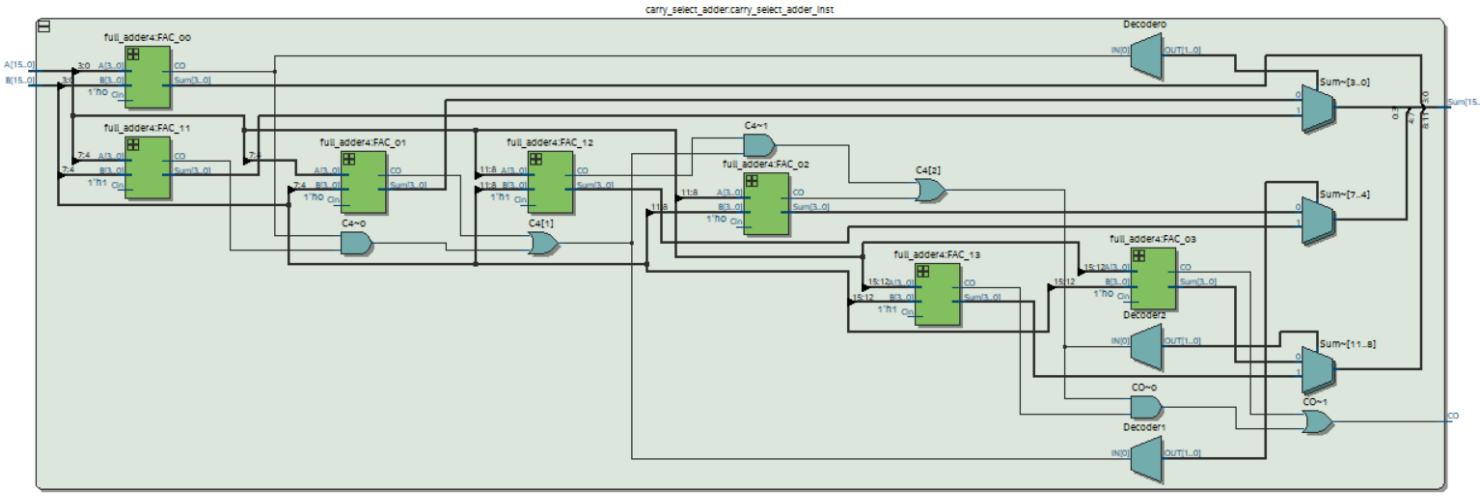

Figure 9: Carry Select Adder

3.4 Area, complexity, performance trade-offs between adders

The ripple-carry adder has the most straightforward design, with the least complexity. It only needs to wire several full adders together so it has the least area. However, one full adder must wait for the output of another full adder and the critical path is the longest. Its performance is not so good, as it has the largest delay.

The carry lookahead adder has some additional logic to compute P and G so it has more area than the ripple carries adder, producing a higher power consumption. It involves most complex logic expressions, involving multiple units. It's also the most complex adder to implement. However, it supports the parallel computation of carrying bits. So it has a higher frequency and better performance.

The carry select adder has nearly two times the number of adders. But based on the data in table 1, it still has less LUT and area than the carry-lookahead adder. The complexity of CSA is fair as it only contains a few logic gates besides adders. The propagation delay of the carry select adder is the smallest of all, with the highest frequency. Overall, It turns out to carry select adder has the best performance.

3.5 Performance graph

The detailed table including data of three adders is shown in table 1 in the post-lab answer section.

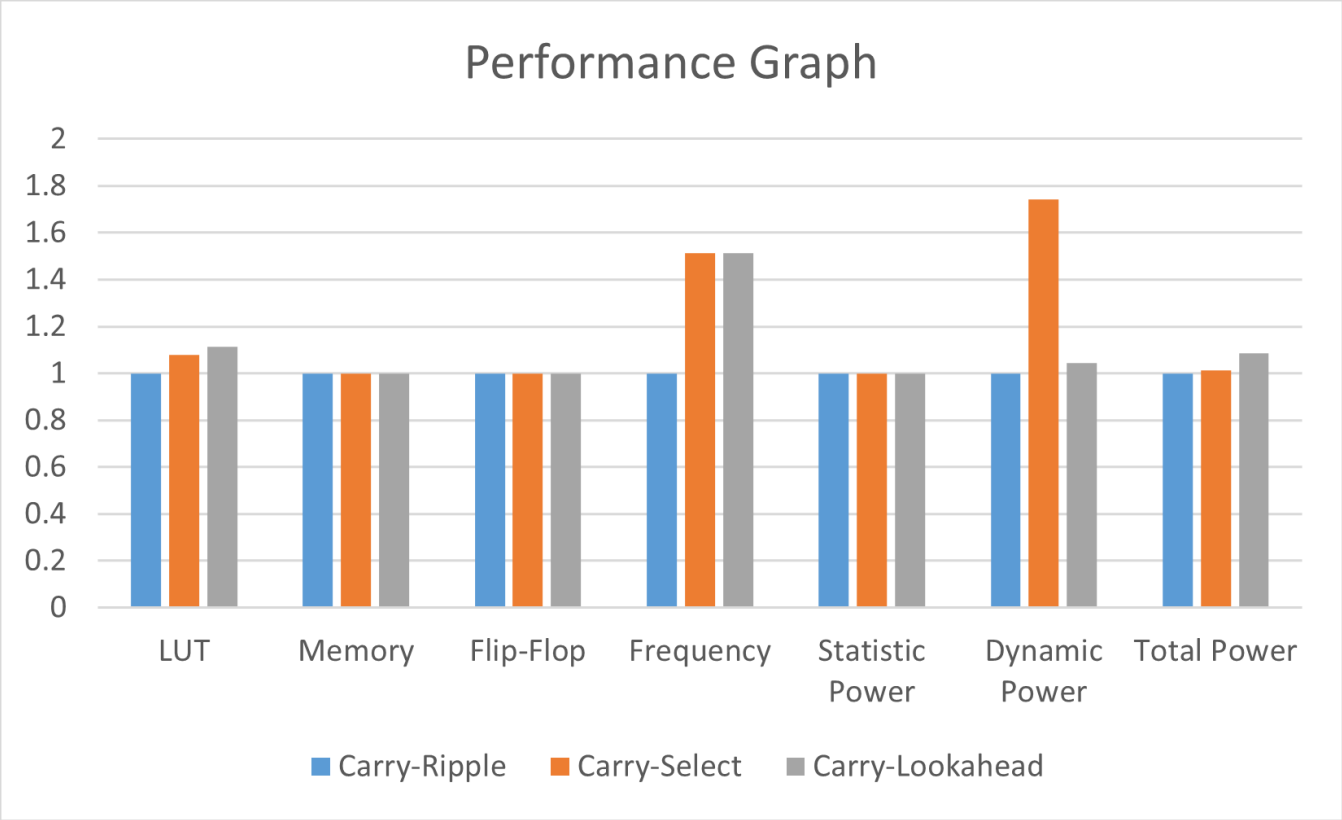

Figure 10: Adder Performance graph

4 Answers to the post-lab questions

1. **Compare the usage of LUT, Memory, and Flip-Flop of your bit-serial logic processor exercise in the IQT with your TTL design in Lab 3. Make an educated guess of the usage of these resources for TTL assuming the processor is extended to 8-bit. Which design is better, and why?**

   The corresponding logic of LUT is more convenient than TTL, but it is not as intuitive as the TTL hardware circuit. I think the quality of the design should be evaluated according to the difficulty of the design and the actual effect. I think LUT design greatly reduces the time of line design and production, and improves efficiency. Compared with TTL design, it is more friendly for industrial applications.

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 (what information would you need, and what experiments would you do to figure out?)**

   For this lab, we only design a 4x4 hierarchy, but it's also possible to design an 8x2 hierarchy or a 2x8 hierarchy. For the 8x2 hierarchy, it will require more logic gates and a higher area but it can be faster. For the 2x8 hierarchy, it will require fewer logic gates and a lower area but it can be slower. To determine which one is best, we can do all these three designs and compare the evaluation parameters between them, just like what we do now to compare CRA, CLA, CSA.

3. **Design statistics table for each adder. Does each resource breakdown comparison from the plot make sense? Are they complying with the theoretical design expectations, e.g., the maximum operating frequency of the carry-lookahead adder is higher than the carry-ripple adder? Which design consumes more power than the other as you expected, why?**

	Carry-Ripple	Carry-Select	Carry-Lookahead
LUT	114	123	127
Memory (BRAM) (bit)	0	0	0
Flip-Flop	105	105	105
Frequency (MHz)	64.93	98.34	98.21
Static Power (mW)	98.55	98.55	98.58
Dynamic Power (mW)	3.10	5.4	3.24
Total Power (mW)	155.99	157.98	169.13

Table 1: Three adder analysis table

The graph of three adders is shown in Figure 10. Overall, the resource comparison from the table is reasonable. The rippling carry adder has the smallest number of LUT as it has the simplest design. The maximum operating frequency of the carry-lookahead adder is higher than the carry-ripple adder as it has a smaller delay. Intuitively, we may think the carry-select adder consumes the most power as it contains nearly twice the number of full adders. But the result shows that the carry-lookahead adder has the most power consumption as well as the most number of LUTs. which is mostly from additional logic gates to compute P, G, and carry bits.

5 Conclusion

5.1 Bugs and countermeasures taken during this lab

1. Carry Select adder passes test 1 but fails test 2 and test 3. The countermeasure is to look at the RTL diagram produced. It can be found that only certain sums are correctly selected. And it turns out that the bug is a typo in the case statement.

2. Lots of syntax errors. It is mainly caused by the unfamiliarity with System Verilog. Most bugs are fixed according to the error reported by Quartus.

5.2 Difficulty in the lab and possible improvement

The Carry Select adder is the most difficult part of this lab. The units of CSA are not explicitly stated. Even though there are formulas for P, G, and Carry, I still need to figure out what's the input and output for each part of the circuit.

5.3 Summary

In this lab, we are starting to get familiar with SystemVerilog, FPGA, and CAD. These tools greatly improve our efficiency in developing circuits. Also, we designed three ways to implement adders. We learned how to evaluate each design and determine tradeoffs and design Finite State Machine between these adders. Such analysis can be quite useful throughout this course and even in industrial design.