ECE 385

Fall 2021

Experiment # 4

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

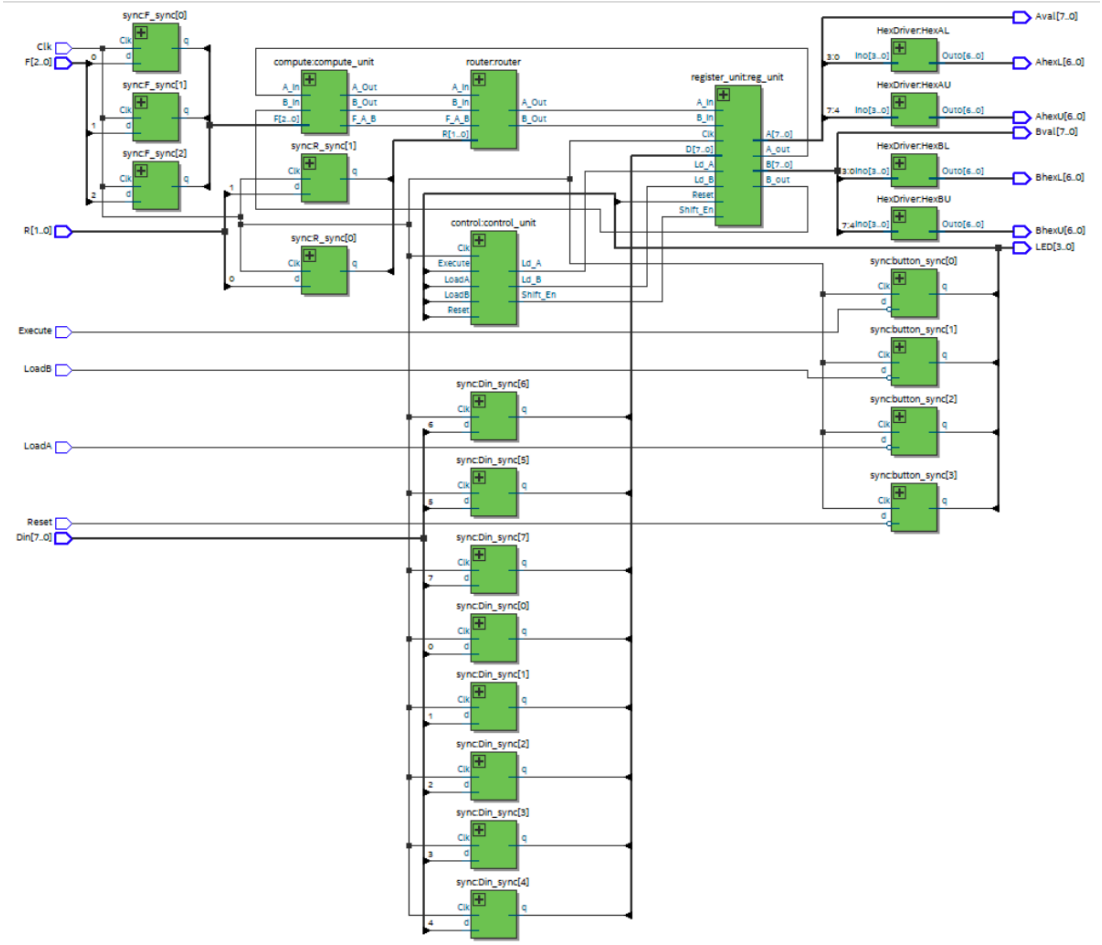
Xu Ke / Zhu Xiaohan

LA4/Thursday & 18:00-20:50 Huang Tianhao

I. Introduction

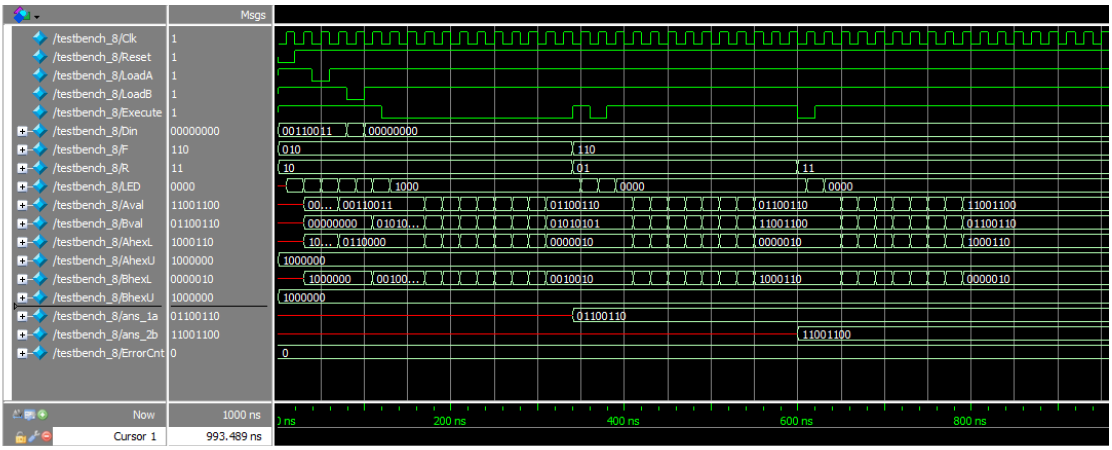
In this experiment, we come to understand the basic syntax and constructs of SystemVerilog, as well as acquire the basic skill required to operate Quartus Prime, a CAD tool for FPGA synthesis and simulation. Specifically, we extend a 4-bit logical processor into 8-bit using System Verilog. And this processor will be used for 8-bit logic operations of AND, OR, XOR, etc. We also use System Verilog to make three types of adders that perform 16-bit arithmetic binary adding operation, including a ripple-adder, a carry-lookahead adder, and a carry-select adder, which have the same operations but are relatively different in design and behavior.

II. Schematic block diagram of the bit-serial logic processor



III. Annotated design simulations of the bit-serial logic processor

In this part, we extend the *shift-register* from 4 bits to 8 bits. And in order to fit the states that *shift-register* functions, we also modify the control unit by adding 4 new states and change the transition process. Besides, we modify the *Hex Drivers* to fit the 8-bits condition. And for the *top-level processor*, we change it to 8-bits and also retake testbench_8.sv as the testbench instead of the previous testbench_4.sv.



IV. Written description, purpose, and operation

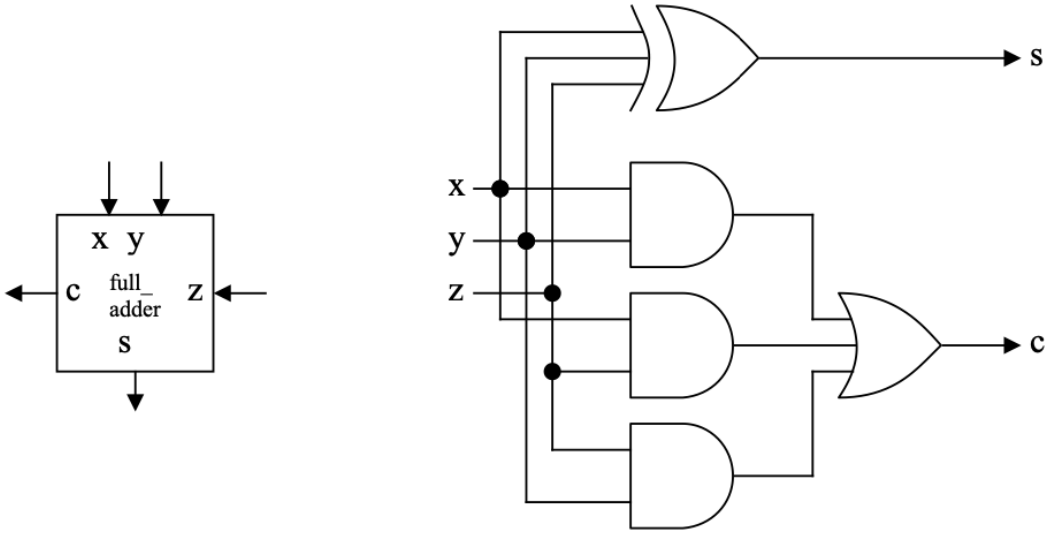


Figure 2: Full-Adder Block Diagram

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = (AB) + (BC_{in}) + (AC_{in})$$

a) Carry-Ripple Adder

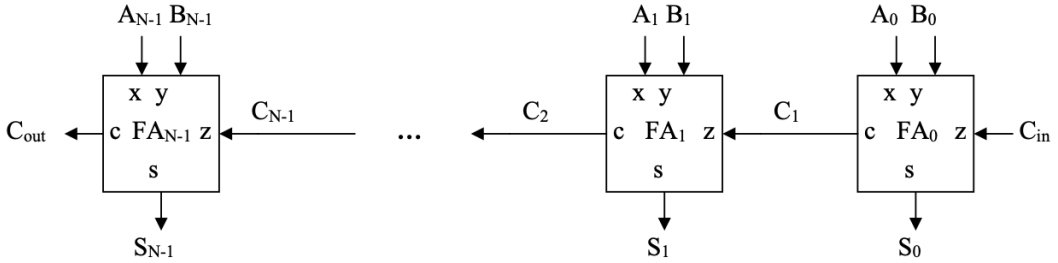


Figure 3: *N*-bit Carry-Ripple Adder Block Diagram

As the figure shown above, we build our *Carry-Ripple Adder* based on *Full-Adders*, where we have three binary bits (i.e. *A, B* and C_{in}) inputted through a set of logic gates to produce a single-bit sum (i.e. *S*) and a single-bit carry-out (C_{out}). The N full- adders are then linked together in series through the carry bits, forming an N-bit binary adder. When the binary inputs are provided, the *Full-Adder* of the least significant bit (*LSB*) will produce a sum (S_0) and a carry-out (C_1). The carry-out is fed to the carry-in of the second Full-Adder, which then produces a second sum (S_1) and a second carry-out (C_2). The process ripples through all N bits of the adder and settles when the Full-Adder of the most significant bit (*MSB*) outputs its sum (*SN-1*) and carry-out (C_{out}).

b) Carry-Lookahead-Adder

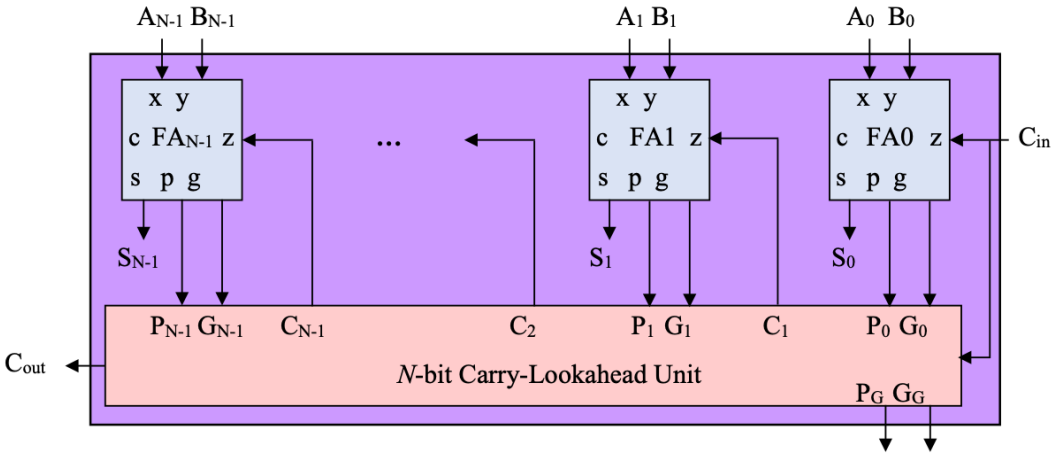


Figure 4: *N*-bit Carry-Lookahead Adder Block Diagram

Carry-Lookahead Adder (CLA) uses the concept that every bit of the *CLA* makes predictions using its immediate available inputs (A and B), and predicts what its carry-out would be for any value of its carry-in. A carry-out is generated (G) if and only if both available inputs (A and B) are 1, regardless of the carry-in.

$$G(A, B) = A \cdot B$$

On the other hand, a carry-out has the possibility of being propagated (P) if either A or B is 1:

$$P(A, B) = A \oplus B.$$

With P and G defined, the Boolean expression for the carry-out Ci+1 giving a potential Ci is then

$$C = G + (P \cdot C)$$

C_{i+1} can be expressed in terms of Ci which in turn can be expressed in terms of C_{i-1} . However, if Ci still depends on C_{i-1} , it will behave like a ripple adder without giving any gain in speed.

In this way, the computation time of the CLA is much faster than that of the CRA, resulting in a higher operating frequency. The downside of the CLA is its additional logic gates, which increases both the area and power consumption of the adder.

In the 4x4-bit hierarchical CLA design, the 16-bit inputs A and B are divided into groups of 4 bits. First, each group of 4 bits go through a 4-bit CLA. Next, a tempting

design is to cascade the four 4-bit CLAs by connecting the C_{out} from the previous 4-bit CLA to the C_{in} of the next 4-bit CLA, but in this way we will be trapped by the slow rippling of these carry bits again. Therefore, instead of using the Cout from the previous 4- bit CLA, we generate the C_{in}s of the 4-bit CLAs using the PGs and GGs, shown as the figure below.

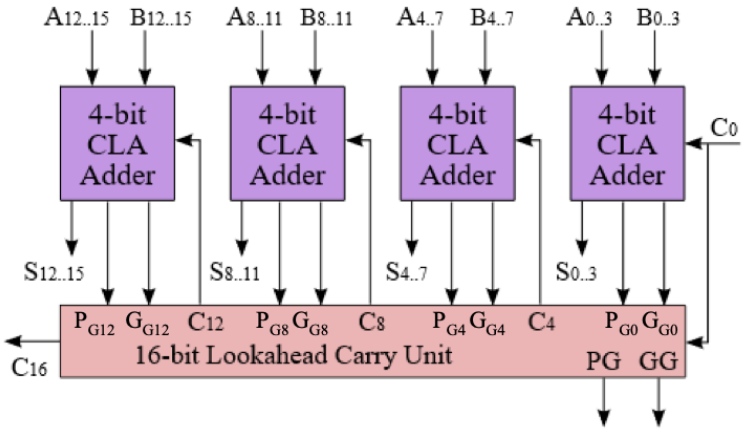


Figure 5: A 4x4-bit Hierarchical Carry-Lookahead Adder Block Diagram

c) Carry-Select-Adder

Carry-Select Adder (CSA) consists of two full adders (or CRAs if multiple bits are grouped) and a multiplexor. One adder computes the sum and carry-out based on the assumption that the carry-in is 0, and the other assumes that the carry-in is 1. In this way, both possible outcomes are pre-computed. Once the real carry-in arrives, the corresponding sum and carry-out is selected to be delivered to the next stage. By paying the price of almost twice the numbers of adders, we gain some speedup.

We design a 16-bit CSA with 4x4-bit hierarchical structure and for each group of 4-bit inputs, we use two CRAs to calculate two versions of the results, one with carry-in bit assumed to be 0 and the other to be 1. Therefore, eventually the 16-bit CSA will contain seven 4-bit CRAs.

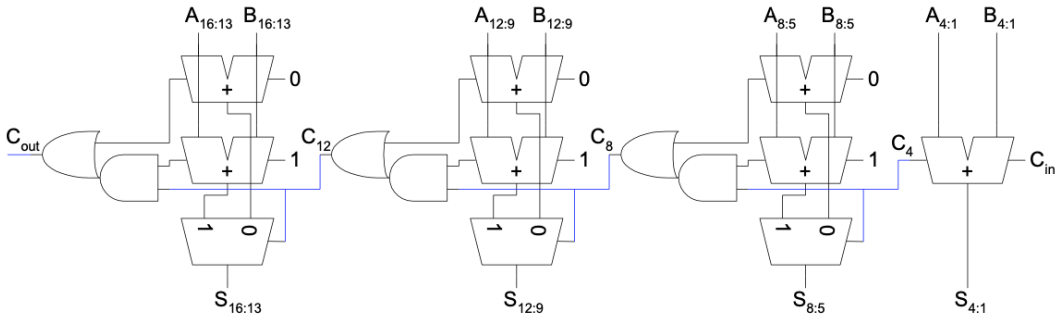


Figure 5: 16-bit Carry-Select Adder Block Diagram

V. Area, complexity, and performance tradeoffs

a) Carry-Ripple Adder

Carry-Ripple Adder takes up the smallest space, i.e. the simplest layout and the lowest complexity is the biggest advantage of Carry-Ripple Adder. However, the higher effective unit must wait for the lower effective unit, causing a longer time-delay.

b) Carry-Lookahead-Adder

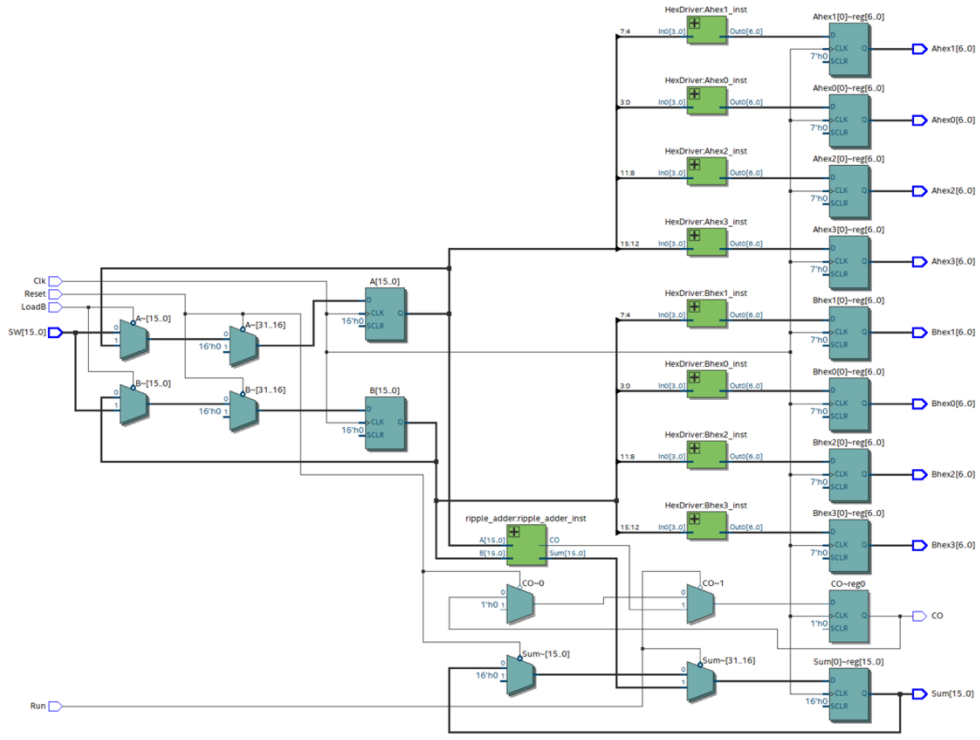
The time delay required for the Carry-Lookahead-Adder to calculate C_{i+1} is fixed by the time-delay of the third-stage gate, rather than the number of bits of operation of the adder, which greatly improves the time efficiency of Carry-Lookahead-Adder. However, the complexity of Carry-Lookahead-Adder design will explode rapidly since the number of calculation bits will increase, resulting in high area and design complexity.

c) Carry-Select-Adder

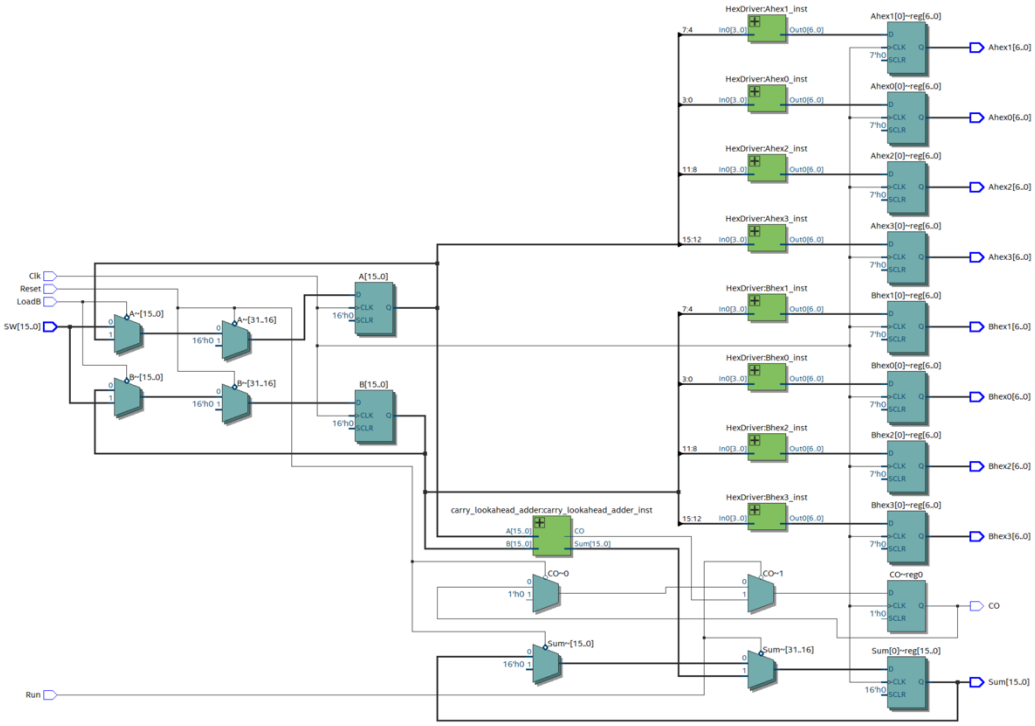
The Carry-Select-Adder uses almost twice the number of full-adders as used in Carry-Ripple Adder, leading to a larger area and higher complexity in structure. However, for the condition that has large number of operation bits, the complexity and area of Carry-Select-Adder will be smaller than that of Carry-Lookahead-Adder. Also for Carry-Select-Adder, pre-calculation and parallel computing can make it highly time-saving.

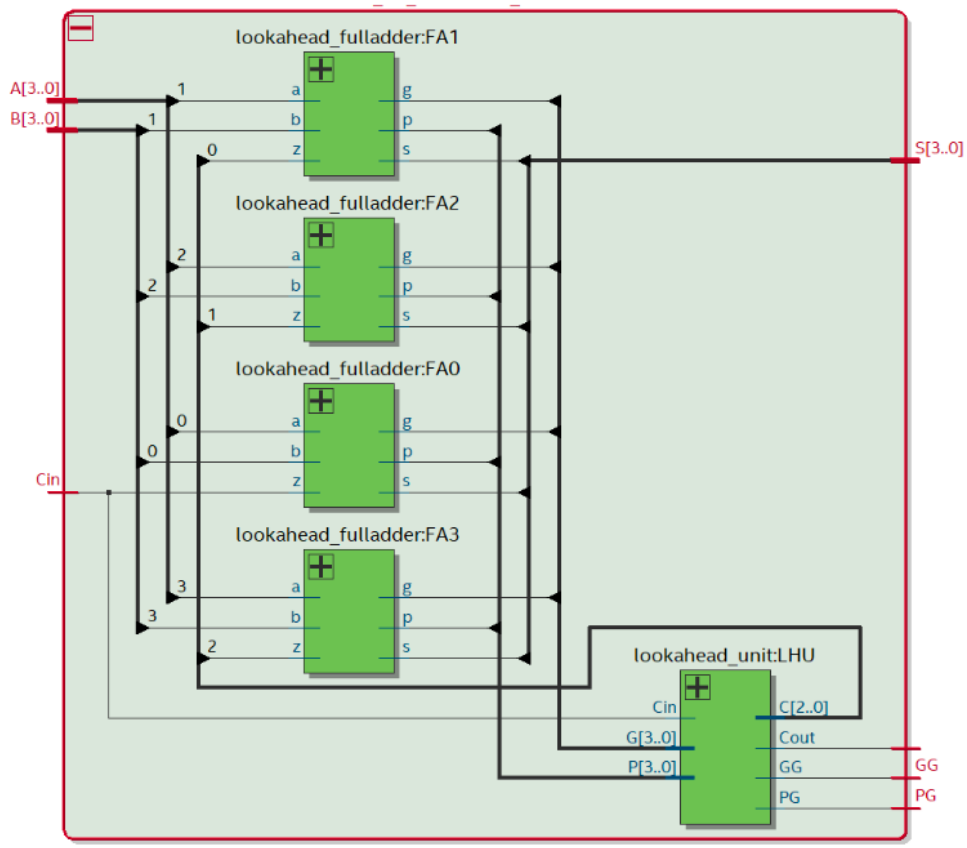
VI. Schematic block diagrams with components, ports, and interconnections labeled

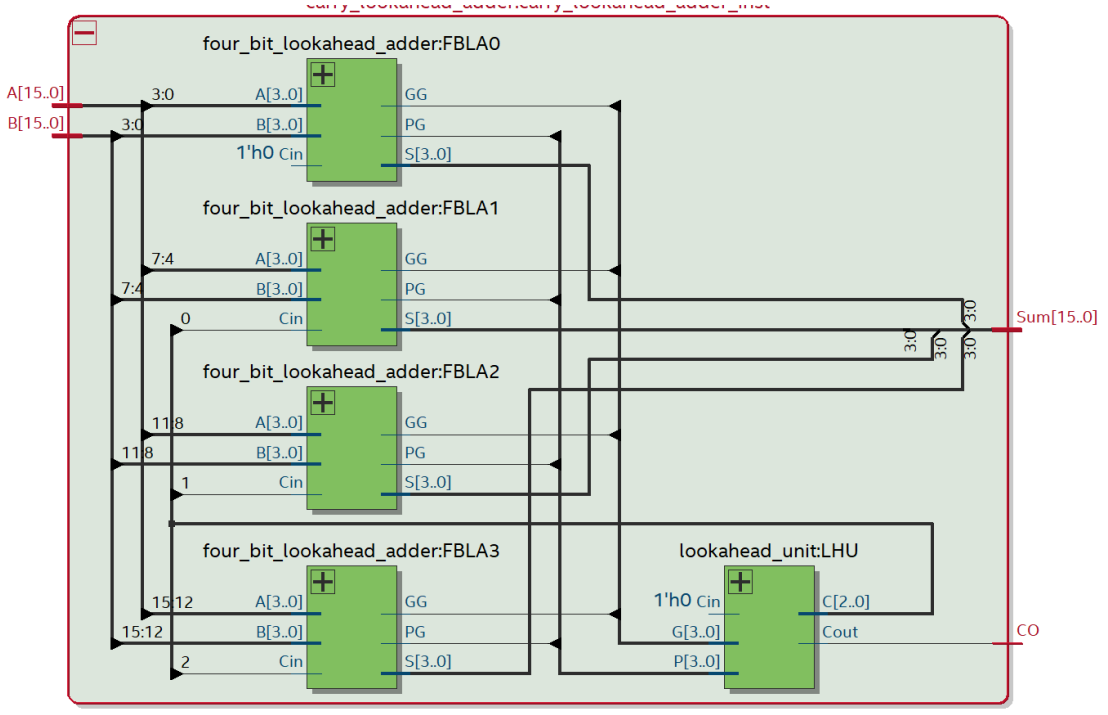
a) Carry-Ripple Adder

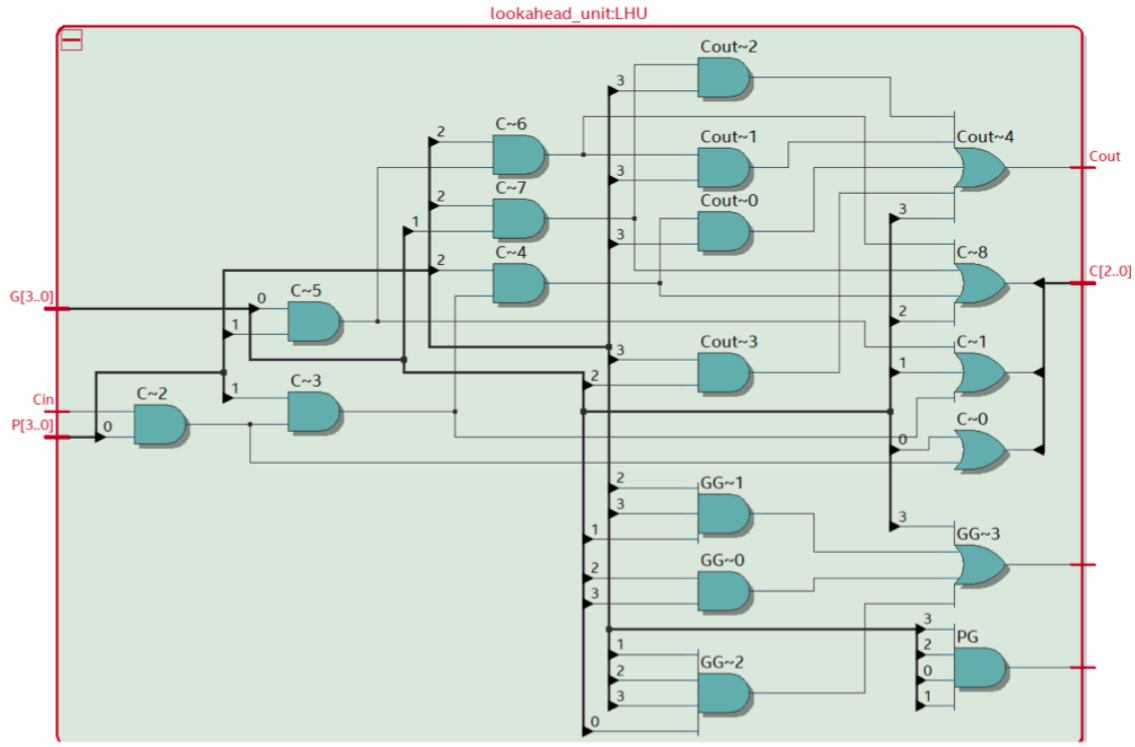


b) Carry-Lookahead-Adder

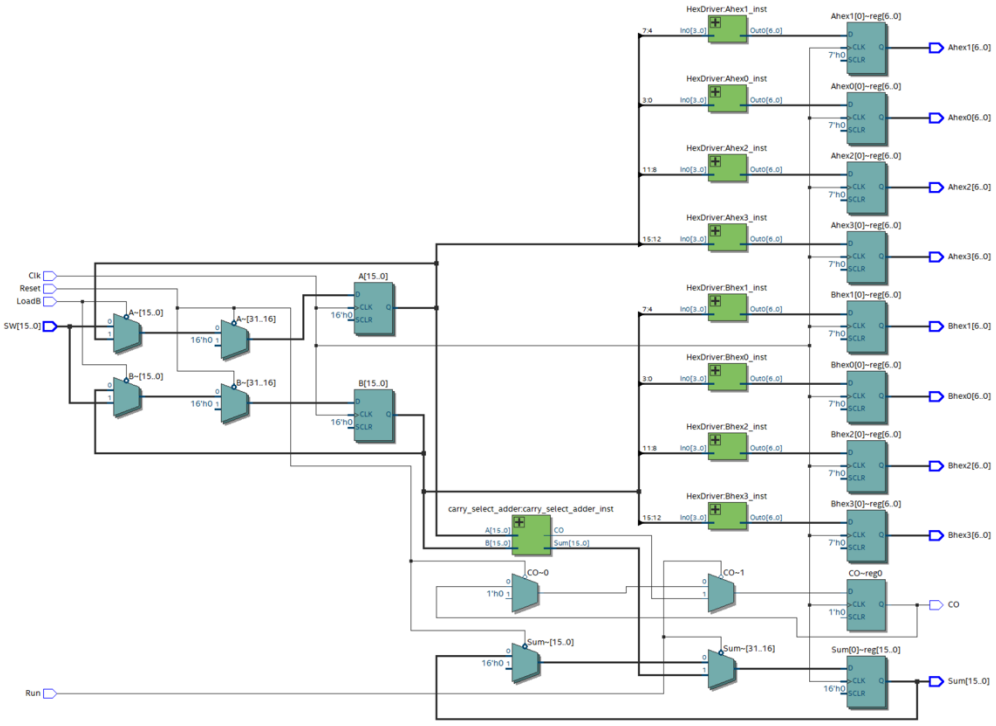


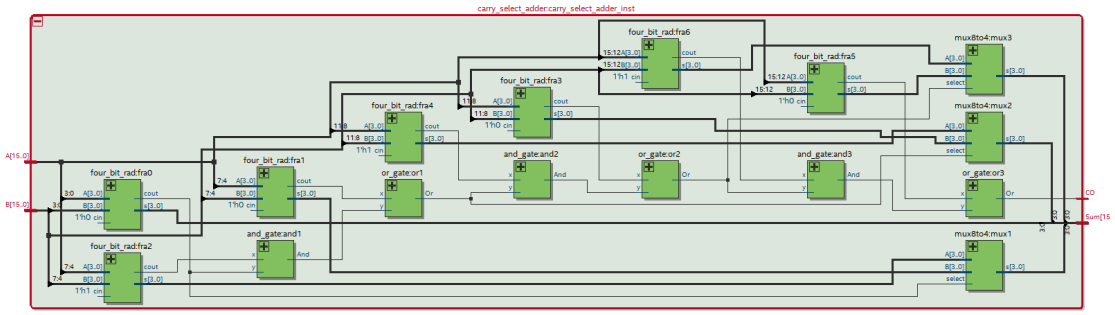



four_bit_lookahead_adder:FBLA0
four_bit_lookahead_adder:FBLA1
four_bit_lookahead_adder:FBLA2
four_bit_lookahead_adder:FBLA3
lookahead_unit:LHU
A[15..0]
B[15..0]
A[3..0]
B[3..0]
Cin
S[3..0]
GG
PG
1'h0 Cin
C[2..0]
G[3..0]
P[3..0]
Cout
Sum[15..0]
CO


lookahead_unit:LHU
Cout~2
Cout~1
Cout~0
Cout~3
Cout~4
C~6
C~7
C~4
C~5
C~2
C~3
C~8
C~1
C~0
GG~1
GG~0
GG~2
GG~3
PG
G[3..0]
P[3..0]
Cin
Cout
C[2..0]

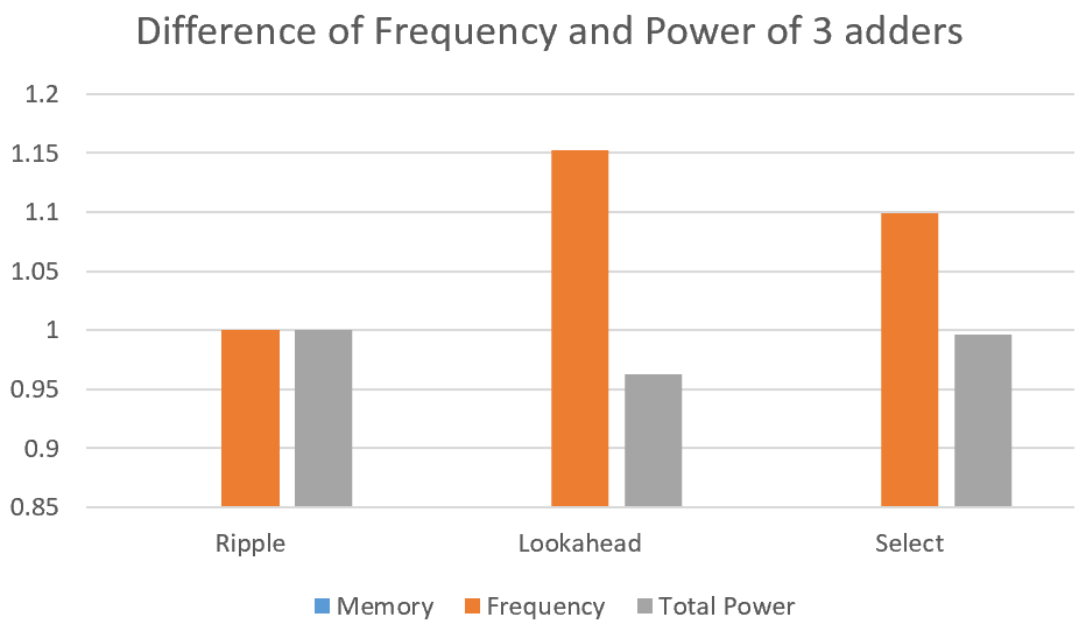
c) Carry-Select-Adder





VII. Design analysis comparison results from pre-lab

	Carry-Ripple Adder	Carry-Lookahead Adder	Carry-Select Adder
Memory	N/A	N/A	N/A
Frequency	89.56Mhz (1)	103.2Mhz (1.1523)	98.43Mhz (1.099)
Total Power	147.33mW (1)	141.79mW (0.9624)	146.79mW (0.9963)



VIII. Answers to post-lab questions

a) Question 1

	Lab 3	Lab 4
LUT	122	72
Memory (BRAM)	0	0
Flip-Flop	21	43

If we extend the design into 8 bits, we need to rebuild the entire logic, but we don't need to add states to the Mealy State Machine and still need 1 kip-flop for the control unit. Since the inputs were extended into 8 bits, we would need four 4-bit shift registers, which is composed of 4 flip-flops. The inside of a 4-bit counter has 4 flip-flops (we will only make use of three of them), thus the total number of 21 flip-flops will be used in the TTL design.

However, the resource information shows that is the SystemVerilog design, we used 43 flip-flops. The reason behind that is we used too many states for the state machine in the control module. The LUT for the SystemVerilog design, in this case, is 72, but we would not need this much even if we extend the design to 8 bits.

In my opinion, I think the TTL design is better since it uses fewer resources but harder to design. The System Verilog design is simple to make but used more resources.

b) Question 2

The 4x4 hierarchy design of the CSA we used for this experiment may be not ideal. We chose one unit of the CSA to be 4 bits, which is possibly large compared to the total of 16 bits. Also, there are other choices we could make since we could put different numbers of full adders in each unit or increase the number of units. We need to know

the time-delays for the gates and MUXs to compute total operation time for the CSA to make sure whether a unit or the selecting logic and the MUXs operate faster. In the ideal case, the adders finish operation when the MUXs finish the operations. We need to do the experiments to test those delays to balance the operation time of MUXs and adders.

c) Question 3

For Carry-Ripple Adder:

LUT	114
DSP	None
Memory (BRAM)	0 bit
Flip-Flop	105
Frequency	89.56 MHz
Static Power	83.37 mW
Dynamic Power	2.15 mW
Total Power	147.33 mW

For Carry-Lookahead Adder:

LUT	123
DSP	None
Memory (BRAM)	0 bit
Flip-Flop	105
Frequency	103.2 MHz
Static Power	82.86 mW
Dynamic Power	2.58 mW
Total Power	141.79 mW

For Carry-Select Adder:

LUT	127
DSP	None
Memory (BRAM)	0 bit
Flip-Flop	105
Frequency	98.43 MHz
Static Power	83.78 mW
Dynamic Power	2.03 mW
Total Power	146.79 mW

The select adder and ripple adder make sense in terms of each other. While the Select Adder is faster than the Ripple-Adder (98.43 MHz compared to 89.56 MHz), it

has higher complexity than the ripple adder (127 LUTs instead of 114) which makes sense to why it uses more power (83.78mW vs 83.37mW). We would similarly expect the lookahead adder to be faster than the ripple adder, with a tradeoff of more complexity and higher power dissipation.

However, we saw that a lower frequency than the ripple adder (103.2 MHz) and essentially not much more power consumption. It was clear after seeing these numbers that even though our carry-lookahead adder was producing the correct answers, we were running into the same limitation of the carry ripple adder in the fact that we were connecting the C_{out} from one module to the C_{in} to the other module. It makes sense that the frequency was even lower than the ripple because not only did we have to wait for the carry bits to propagate, we also need to wait for the logic for the propagation and generate bits. Had we implemented this correctly with the C_{in} generated from the group propagates and group generates, we would have seen higher operating frequency since we wouldn't have to wait for the carry-outs to propagate to the carry-ins, being able to perform more operations per second directly leading to more power dissipation.

IX. What worked and what didn't with explanations of any possible causes and the potential remedies

Honestly, since we are not very familiar with System Verilog, and since group work we make some mistakes that we share some same name, which is not allowed and caused compiler errors. We fixed the problems by moving modules outside and making them independent and changing variable names to avoid repetition.

X. Conclusion

Before the lab session, we do the preparation with the lab manual and tutorials on the Blackboard regarding Quartus and System Verilog and especially the recorded video which is extremely helpful. They provide us with detailed instructions and procedures on how to conduct this lab step by step, in which the pin assignment table is most useful and saves us a lot of time to figure out everything by ourselves. Besides, honestly, this is our very first interactions with FPGA and System Verilog and through this experience we think it is really interesting and hope to explore more in the coming labs. According to me, the most impressive thing in this journey is that I know how to build structures more efficiently, even though many choices can all achieve the given functional goals.