University of Illinois at Urbana-Champaign

ECE385
Digital Systems Laboratory

Lab3: Introduction to SystemVerilog, FPGA, EDA, and 16-bit Adders

TA: Abigail Wezelis, Harris Mohamed

Yan Miao, Guangxun Zhai
yanmiao2, gzhai5

February 2021

Contents

1 Introduction

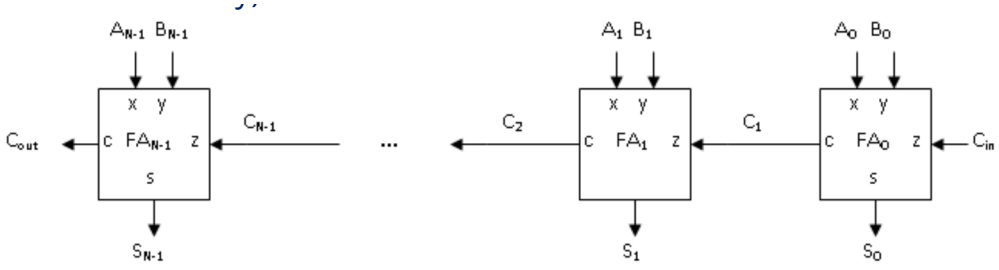
In this lab, we keep focusing on the hardware coding and use SystemVerilog to write three different codes to implement binary adding. The three adders in our design are a carry-ripple adder(CRA), a carry-lookahead adder(CLA), and a carry-select adder(CSA). Then, we study the differences and performances of those adders.

2 Adders

In this section, we are going to describe how we write the code of three adders as well as compare the differences and tradeoffs between those implements.

2.1 Ripple Carry Adder

A Ripple Carry Adder is basically a combination of several full-adders. we combine those full-adders by linking the original c_out of the former full-adder with the original c_in of the next full-adder and collecting the outputs of those adders. For the design of full-adder, we simply come up with the logic in Boolean expressions using a truth table.



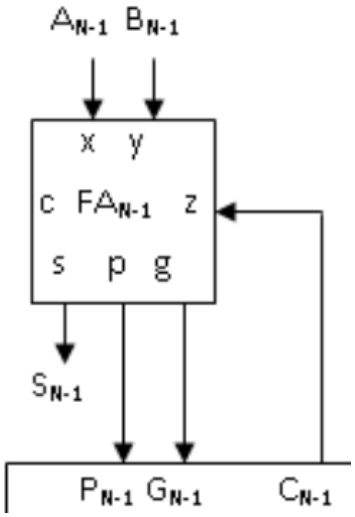
Block Diagram of Ripple Carry Adder

2.2 Carry Lookahead Adder

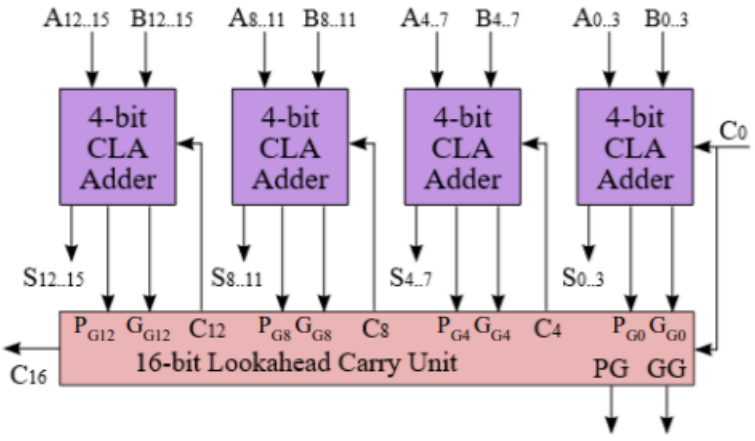
Compared with the Ripple Carry Adder, Carry Lookahead Adder has a more effective speed for doing calculations since we introduce two new values (P/G bits) to speed up instead of waiting for each carryout provided by the former full-adder. In the design, we also develop a Carry Lookahead Unit to collect those P/G values and send the carry-out bits back to the full-adders.

G = A*B, which means that G = 1 only when A and B are both 1. When both A and B are 1, the adder must provide a carry-out. P = A XOR B, which means that P = 1 while either A or B is 1. When either A or B is 1, the adder produces a carry-out if the former carry-out is 1 and does not produce a carry-out if the former carry-out is 0. Therefore, the values of P/G determine the value of each carry-out.

The hierarchical 4x4 adder in the Carry Lookahead Adder design is a combination of four full-adders with some small changes. These four 4-bit CLAs parrallel loads the bits value of A/B and process them to output the propagate group and generate group (which includes P/G values). Next, we use those P/G group to generate right c_in for the next 4-bit CLA.



Block Diagram Inside A Single CLA (4-bits)

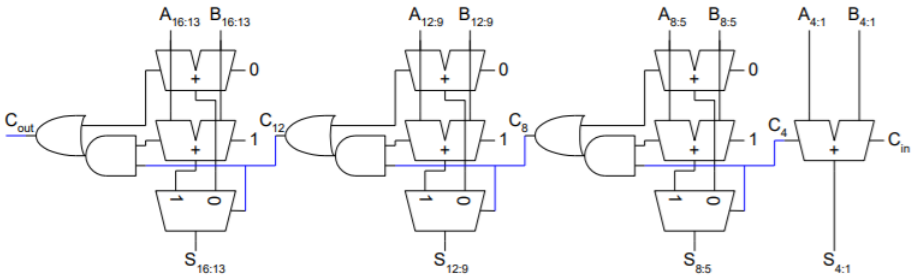


Block Diagram of How Each CLA Was Chained Together

2.3 Carry Select Adder

Similar to the Carry Lookahead Adder, Carry Select Adder improves its efficiency by not waiting for the carry-outs calculated by each full-adders. Carry Select Adder uses a double amount of full-adders and pre-compute the results by assuming c_in to be 1 and 0. Then it passes the results into 2-to-1 MUXs to select the final answer.

It normally has two conditions that will lead to the carry-out to be 1. The first condition is both A and B are 1. Under this circumstance, the carry-out has to be 1 no matter the value of the former carry-out is 1 or 0, which is the logic of OR gate in the block diagram. Carry-out would equal 1 when the c_out going from the top full-adder (with c_in = 0) is 1. The second condition is when A/B has one 1, and the former carry-out is also 1. From the block diagram, we can see when the former carry-out is 1, two inputs going through the AND gate are all 1, and this output "1" of this AND gate determined the carry-out is also 1.



Block Diagram of Carry Select Adder

2.4 Description of all .SV modules

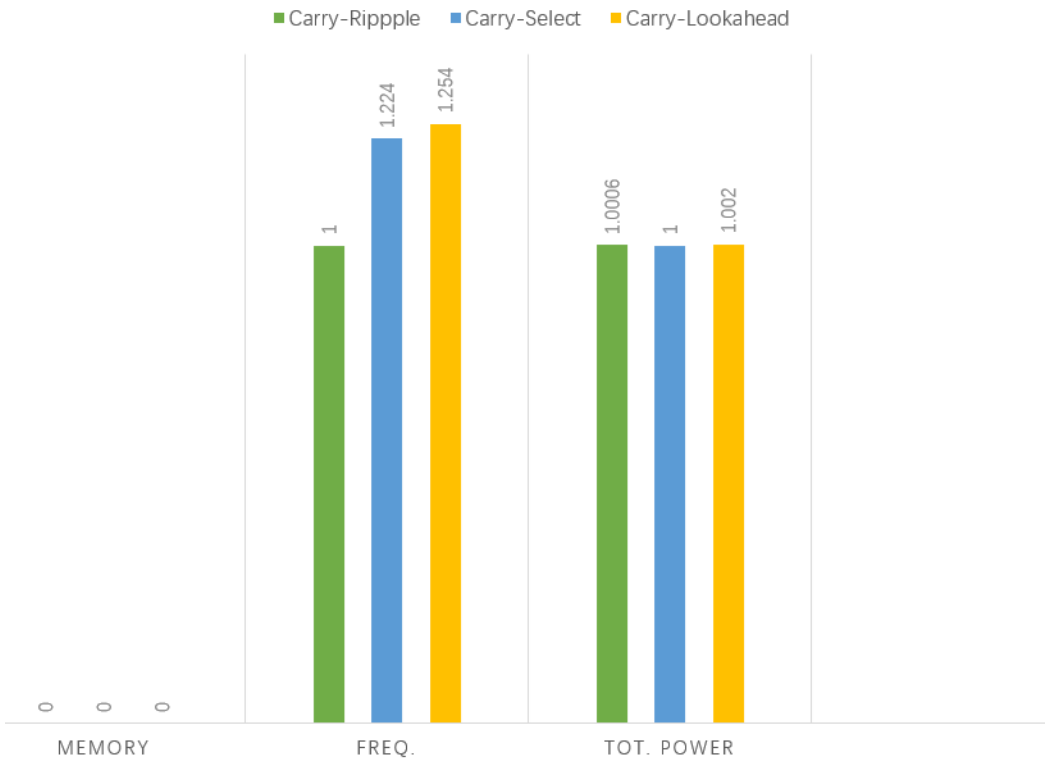
1. **Module: ripple_adder**
    - Inputs: [15:0] A, [15:0] B, c_in
    - Outputs: [15:0]S, c_out
    - Description: ripple-adder is just a linear combination of multiple full-adders. The c_in for the first full-adder is 0 and each former full-adder transmits its c_out to the c_in of next full-adder. Since the full-adders inside need to wait for c_in delivered by the former adder, the operation takes longer time and is not very effective.
    - Purpose: Ripple-adder can do addition of longer bits of input numbers with easy design but ineffective speed.
2. **Module: four_ripple_adder**
    - Inputs: [3:0] A, [3:0] B, c_in
    - Outputs: [3:0]S, c_out
    - Description: this module is just a tiny part of the 16-bit-ripple-adder. It does exactly the same function and have same weakness as the 16-bit-ripple-adder.
    - Purpose: same as the ripple-adder top.
3. **full_adder**
    - Inputs: A, B, c_in
    - Outputs: S, c_out
    - Description: we use the given logic of addition between inputs and out outputs
    - and assign the correct value to S and c_out.
    - Purpose: Because full-adder is included in any of the three complicated n-bit adders, this module is used as a basic file for doing adding calculation of A/B.
4. **lookahead_adder**
    - Inputs: [15:0] A, [15:0] B, c_in
    - Outputs: [15:0]S, c_out
    - Description: the lookahead-adder introduces two new value P and G to compute c_in for the next full-adder so that the whole implement has faster operating speed.
    - Purpose: Lookahead-adder can do addition of longer bits of input numbers with harder design but more effective speed.

5. **CLU**
    - Module: CLU
    - Inputs: [3:0] P, [3:0] G, c_in
    - Outputs: c1, c2, c3, c_out
    - Description: this module uses values of P/G to calculate the c_in for next adder
    - Purpose: CLU is a component of the lookahead-adder
6. **four_CLA**
    - Inputs: [3:0] A, [3:0] B, c_in
    - Outputs: [3:0]S, c_out, pg, gg
    - Description: this module uses A/B (the number we used for adding) to compute P/G values and store them into propagate and generate groups for module CLU to further computing c_in.
    - Purpose: The introduction of P/G largely decrease the computation time and makes it unnecessary to wait for the lower-bit neighbor to produce a carry-out (like in ripple-adder).
7. **select_adder**
    - Inputs: [15:0] A, [15:0] B, c_in
    - Outputs: [15:0]S, c_out
    - Description: this type of adder is composed of full-adders and MUX so that it can compute the carry-out prior to time by assuming a carry-in value.
    - Purpose: Select-adder can do addition of longer bits of input numbers with harder design but more effective speed.
8. **2-to-1 MUX**
    - Inputs: [3:0] S_c0, [3:0] S-c1, c_in
    - Outputs: [3:0]S-out
    - Description: This module is a 2-to-1 MUX, which has 2 inputs, one select bit, and one output. When the select bit is 0, MUX will transmit one of its input values to its output and when the select bit is 1. it will output another input value.
    - tPurpose: A multiplexer makes it possible for several input signals to share one device or resource.

2.5 Area, Complexity, and Performance Tradeoffs

CRA is the simplest design among those 3 adders, but the long computation time is its drawback. Every full-adder inside has to wait for the lower-bit neighbor to produce a carry-out before it can correctly compute its sum and carry-out, which increasing its propagation delay with N. Compared to CRA, either CLA or CSA optimizes processing carry-out and reduces the computation time. Lower cost of time brings higher operating frequency, and the additional logic gates in the CLA and CSA design results in an increment of both area and power consumption of the adders. Furthermore, CLA and CSA are more complicated than CRA in design.

2.6 Performance of Each Adder



Design Analysis of the Three Adders

2.7 Critical Path Analysis(Extra Credit)

As is introduced in the lecture, $F = 1/t\_max$ the maximum frequency is determined by the longest delay of all paths. We call this deciding path the critical path, and have included the datapath diagram below. The critical path for all 3 adders matches our expectation that the longest path would come from inputs(SW) to one of the output(reg_17 in our adder design).

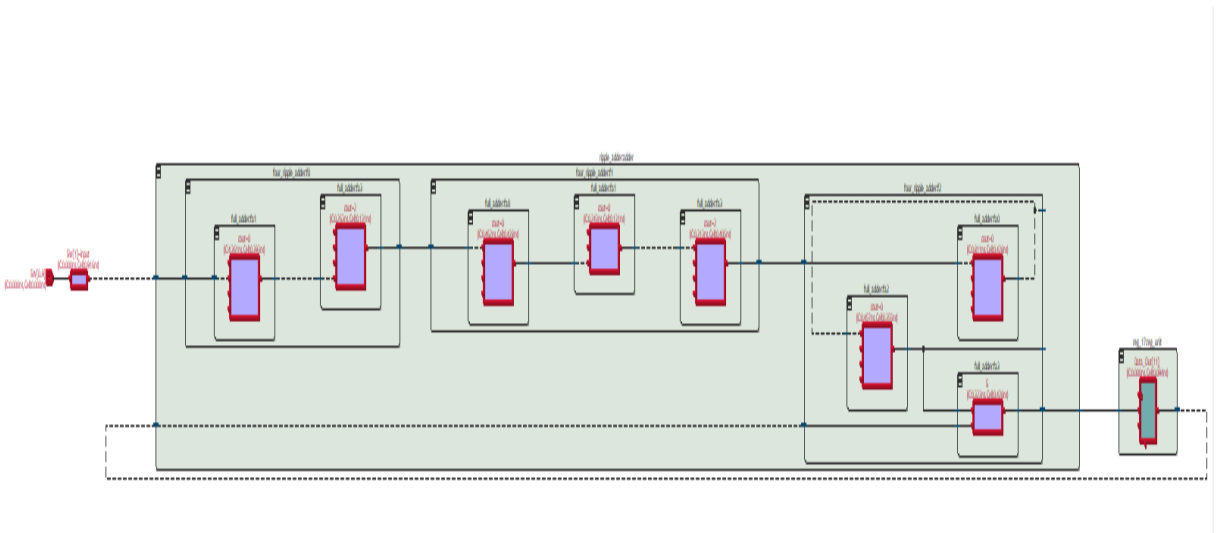
Path #99: Setup slack is 9.450

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	0.000	0.000					clock path
1	0.000	0.000	R				clock network delay
3	3.000	3.000	F	iExt	1	PIN_C11	SW[1]
4	13.942	10.942					data path
1	3.000	0.000	FF	IC	1	IOIBUF_X51_Y54_N22	SW[1]~input|i
2	3.916	0.916	FF	CELL	9	IOIBUF_X51_Y54_N22	SW[1]~input|o
3	8.277	4.361	FF	IC	1	LCCOMB_X77_Y46_N2	adder|f0|fa1|cout~0|datab
4	8.670	0.393	FF	CELL	3	LCCOMB_X77_Y46_N2	adder|f0|fa1|cout~0|combou
5	8.933	0.263	FF	IC	1	LCCOMB_X77_Y46_N20	adder|f0|fa3|cout~2|datad
6	9.064	0.131	FF	CELL	2	LCCOMB_X77_Y46_N20	adder|f0|fa3|cout~2|combou
7	9.526	0.462	FF	IC	1	LCCOMB_X76_Y46_N24	adder|f1|fa0|cout~0|dataa
8	9.954	0.428	FF	CELL	2	LCCOMB_X76_Y46_N24	adder|f1|fa0|cout~0|combou
9	10.197	0.243	FF	IC	1	LCCOMB_X76_Y46_N26	adder|f1|fa1|cout~0|datad
10	10.328	0.131	FF	CELL	3	LCCOMB_X76_Y46_N26	adder|f1|fa1|cout~0|combou
11	10.641	0.313	FF	IC	1	LCCOMB_X76_Y46_N28	adder|f1|fa3|cout~2|dataa
12	11.046	0.405	FF	CELL	2	LCCOMB_X76_Y46_N28	adder|f1|fa3|cout~2|combou
13	11.857	0.811	FF	IC	1	LCCOMB_X76_Y42_N14	adder|f2|fa0|cout~0|dataa
14	12.285	0.428	FF	CELL	4	LCCOMB_X76_Y42_N14	adder|f2|fa0|cout~0|combou
15	12.742	0.457	FF	IC	1	LCCOMB_X77_Y42_N22	adder|f2|fa2|cout~0|dataa
16	13.097	0.355	FF	CELL	3	LCCOMB_X77_Y42_N22	adder|f2|fa2|cout~0|combou
17	13.420	0.323	FF	IC	1	LCCOMB_X77_Y42_N2	adder|f2|fa3|S|dataa
18	13.848	0.428	FF	CELL	1	LCCOMB_X77_Y42_N2	adder|f2|fa3|S|combout
19	13.848	0.000	FF	IC	1	FF_X77_Y42_N3	reg_unit|Data_Out[11]|d
20	13.942	0.094	FF	CELL	1	FF_X77_Y42_N3	reg_17:reg_unit|Data_Out[11

Data Path of the Ripple Adder



Path Diagram of the Ripple Adder

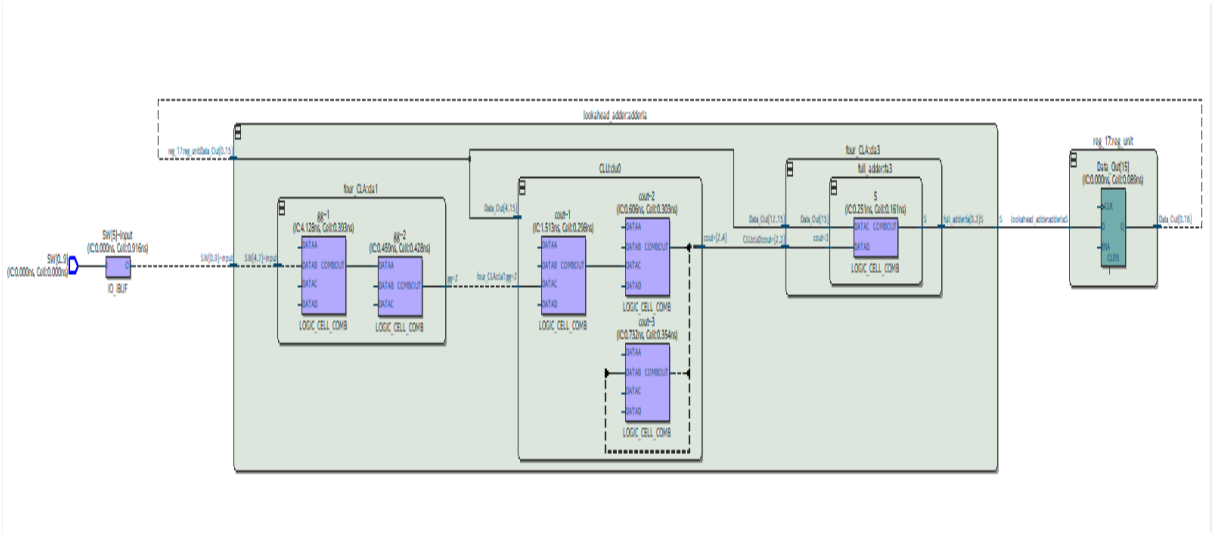
Path #152: Setup slack is 9.751

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	⌄ 0.000	0.000					clock path
1	0.000	0.000	R				clock network delay
3	3.000	3.000	F	iExt	1	PIN_B12	SW[5]
4	⌄ 13.631	10.631					data path
1	3.000	0.000	FF	IC	1	IOIBUF_X49_Y54_N1	SW[5]~input|i
2	3.916	0.916	FF	CELL	12	IOIBUF_X49_Y54_N1	SW[5]~input|o
3	8.044	4.128	FF	IC	1	LCCOMB_X65_Y50_N12	adderla|cla1|gg~1|datab
4	8.437	0.393	FF	CELL	1	LCCOMB_X65_Y50_N12	adderla|cla1|gg~1|combout
5	8.896	0.459	FF	IC	1	LCCOMB_X66_Y50_N30	adderla|cla1|gg~2|dataa
6	9.324	0.428	FF	CELL	3	LCCOMB_X66_Y50_N30	adderla|cla1|gg~2|combout
7	10.837	1.513	FF	IC	1	LCCOMB_X74_Y46_N2	adderla|clu0|cout~1|datac
8	11.135	0.298	FF	CELL	1	LCCOMB_X74_Y46_N2	adderla|clu0|cout~1|combou
9	11.741	0.606	FF	IC	1	LCCOMB_X77_Y46_N14	adderla|clu0|cout~2|datac
10	12.044	0.303	FF	CELL	4	LCCOMB_X77_Y46_N14	adderla|clu0|cout~2|combou
11	12.776	0.732	FF	IC	1	LCCOMB_X74_Y46_N28	adderla|clu0|cout~3|datab
12	13.130	0.354	FF	CELL	2	LCCOMB_X74_Y46_N28	adderla|clu0|cout~3|combou
13	13.381	0.251	FF	IC	1	LCCOMB_X74_Y46_N10	adderla|cla3|fa3|S|datad
14	13.542	0.161	FR	CELL	1	LCCOMB_X74_Y46_N10	adderla|cla3|fa3|S|combout
15	13.542	0.000	RR	IC	1	FF_X74_Y46_N11	reg_unit|Data_Out[15]|d
16	13.631	0.089	RR	CELL	1	FF_X74_Y46_N11	reg_17:reg_unit|Data_Out[15]

Data Path of the Lookahead Adder



Path Diagram of the Lookahead Adder

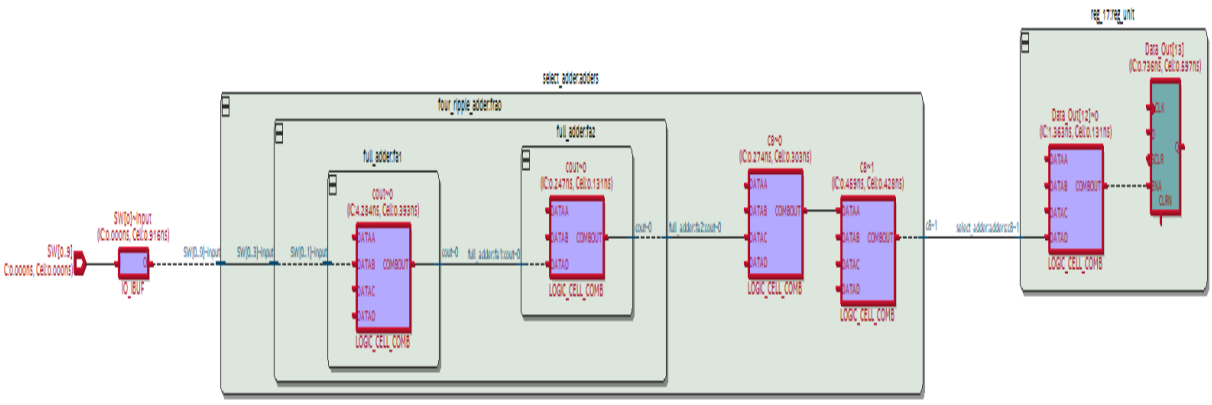
Path #162: Setup slack is 10.131

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	⌄ 0.000	0.000					clock path
1	0.000	0.000	R				clock network delay
3	3.000	3.000	F	iExt	1	PIN_C10	SW[0]
4	⌄ 13.262	10.262					data path
1	3.000	0.000	FF	IC	1	IOIBUF_X51_Y54_N29	SW[0]~input|i
2	3.916	0.916	FF	CELL	10	IOIBUF_X51_Y54_N29	SW[0]~input|o
3	8.200	4.284	FF	IC	1	LCCOMB_X71_Y50_N16	adders|fra0|fa1|cout~0|datab
4	8.593	0.393	FF	CELL	2	LCCOMB_X71_Y50_N16	adders|fra0|fa1|cout~0|comb
5	8.840	0.247	FF	IC	1	LCCOMB_X71_Y50_N26	adders|fra0|fa2|cout~0|datac
6	8.971	0.131	FF	CELL	3	LCCOMB_X71_Y50_N26	adders|fra0|fa2|cout~0|comb
7	9.245	0.274	FF	IC	1	LCCOMB_X71_Y50_N12	adders|c8~0|datac
8	9.548	0.303	FF	CELL	1	LCCOMB_X71_Y50_N12	adders|c8~0|combout
9	10.007	0.459	FF	IC	1	LCCOMB_X72_Y50_N8	adders|c8~1|dataa
10	10.435	0.428	FF	CELL	6	LCCOMB_X72_Y50_N8	adders|c8~1|combout
11	11.798	1.363	FF	IC	1	LCCOMB_X77_Y43_N4	reg_unit|Data_Out[12]~0|data
12	11.929	0.131	FF	CELL	4	LCCOMB_X77_Y43_N4	reg_unit|Data_Out[12]~0|com
13	12.665	0.736	FF	IC	1	FF_X77_Y41_N17	reg_unit|Data_Out[13]|ena
14	13.262	0.597	FF	CELL	1	FF_X77_Y41_N17	reg_17:reg_unit|Data_Out[13]

Data Path of the Select Adder



Path Diagram of the Select Adder

3 Post-Lab Questions

(a) **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, what experiments would you do to figure out?)**

When it comes to design choice, there's no perfect choice for everything. We have to consider the trade-off in power, computing capacity and speed between different designs.

In this case, we could potentially come up with a 2*8 hierarchy design (2 ripple adders per unit, 8 units in total)with better speed but higher power dissipation. Compared to the original 4*4 design, we could get a speed increase because we utilize more parallelism(8-unit) and less waiting on the c_out for previous ripple adders(2 ripple adders compared to 4 before). However, with this faster design, it also comes up with more logical units or LUT, because we are doing more calculations than before; therefore, it certainly would lead to more power consumption.

Admitted this is just theoretical prediction, to verify that this is indeed the case, we still need to modify our HDL adders to a 2*8 structure and re-run all the analysis(Frequency, Total Power, LUT) we did in the table below.

(b) **For the adders, refer to the Design Resources and Statistics and complete the following design statistics table for each adder. This is more comprehensive than the above design analysis and is required for every SystemVerilog circuit.**

	Select-Adder	Lookahead-Adder	Ripple-Adder
LUT	82	87	78
DSP	0	0	0
Memory (BRAM)	0	0	0
Flip-Flop	20	20	20
Frequency	67.8 MHz	68.01 MHz	64.47 MHz
Static Power	89.97 mW	89.97 mW	89.97 mW
Dynamic Power	1.58 mW	1.41 mW	1.39 mW
Total Power	105.34 mW	105.28 mW	105.11 mW

Design Resources and Statistics

(c) **Observe the data plot and provide explanation to the data, i.e., does each resource breakdown comparison from the plot makes 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?**

Overall, we think our data makes sense since when we compare data for either adder, it fits our expectation.

Let's first take a look at CSA and CRA, since we calculated all the potential results in CSA, we would definitely use more LUT than CRA, and this leads to more power consumption given more computation. It's faster(higher frequency) because instead of waiting on the previous c_out, we pre-calculated the result.

Then let's also take at CLA and CRA, since we introduced CLU to calculate additional(compared to CRA) P and G signals for us, it also makes sense we would use more LUT, which would give us more computing overhead. Again, this would be faster(higher frequency) because it doesn't need to wait on all previous c_out and can use CLU(parallel calculation) logic to generate the c_out. And it also fits our expectation that we need more power because it does more computation than CRA.

A final note on this analysis, as suggested by Professor Chen, our power analysis has low confidence because this adder design is too simple and we only use a small portion of the FPGA(<0.01), therefore it's possible that the resolution of the power analysis tool isn't that great.

4 Conclusion

This lab is not a long one and we just finish the whole procedure by following all the instructions given by lectures and those PDF. We individually finish three SystemVerilog files and set some parameters on Quarters, which gave us a brief view of how to write a hardware code by connecting each module.

4.1 Bugs and Countermeasures

(a) **CLA: flat c_out instead of using another CLU**

In our original design, we directly connect the c_out of a 4-bit CLA to the c_in of the next 4-bit CLA; however, it lowers the efficiency in that we didn't utilize hierarchy. To solve this problem, we created another instance of the CLU and generate all c_out for the 4-bit CLAs using the P, G values they provided.

(b) **CSA: 1st level only 2-adder**

Initially, we used 8 4-bit CRA to implement CSA; but later we discovered that we only need 1 4-bit CRA in the first level because the first c_in is already known and doesn't need extra computation. Therefore, we optimized our design by removing the extra MUX and the extra CRA.

4.2 Suggestions to Lab-Manual and Given Document

I felt the instructions on the manual are pretty straightforward and easy to follow. For instance, the block diagrams provided in the documents help our group a lot on coding the three adders, and the video from the website also gives us a brief idea of the format of this hardware code and also how to write a testbench (which is very useful for the next lab). Besides, we hope to see more graphs to explain some complicated topic in future since reading long pages of texts are always painful while graphs are easier to understand.