ECE 385

Fall 2020

Experiment 4

Intro to System Verilog

Zhicong Fan/Xin Jin
AB2/Online
Yucheng Liang

Introduction:

In this lab, we designed an 8 bits serial logic processor which simulated the functionality the lab3 circuits, and three types of adders which are ripple adder, look-ahead adder, and select adder. The last two adders were designed to optimize the runtime. The design was accomplished by using system Verilog on FPGA.

Serial Logic Processor:

The provided 4 bits serial logic processor was extended to 8 bits by first changing the inputs of the processor from 4 bits to 8 bits. Then the storing unit register A and B should also be changed to 8 bits by extending 4 additions bits after the original 4 bits. Accordingly, since we changed 4 bits to 8 bits, we would need to add four additional states to the control unit by extending one start states, 4 computing states and one finish state to one start states, 8 computing states and a finish state.

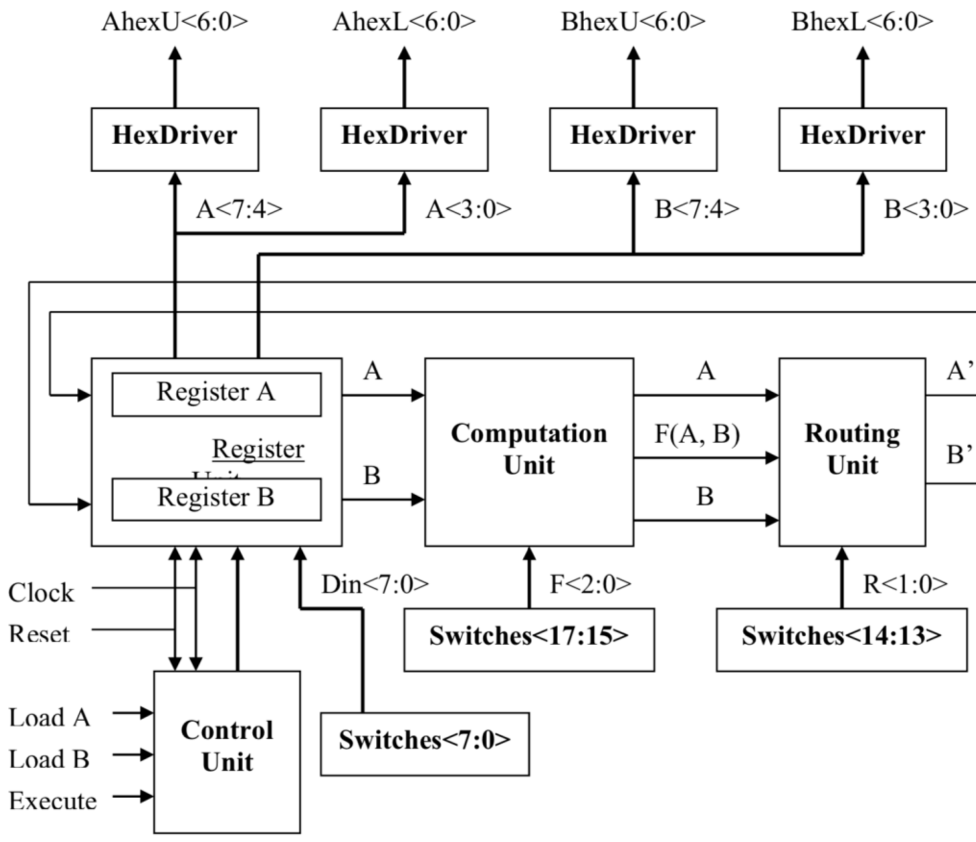
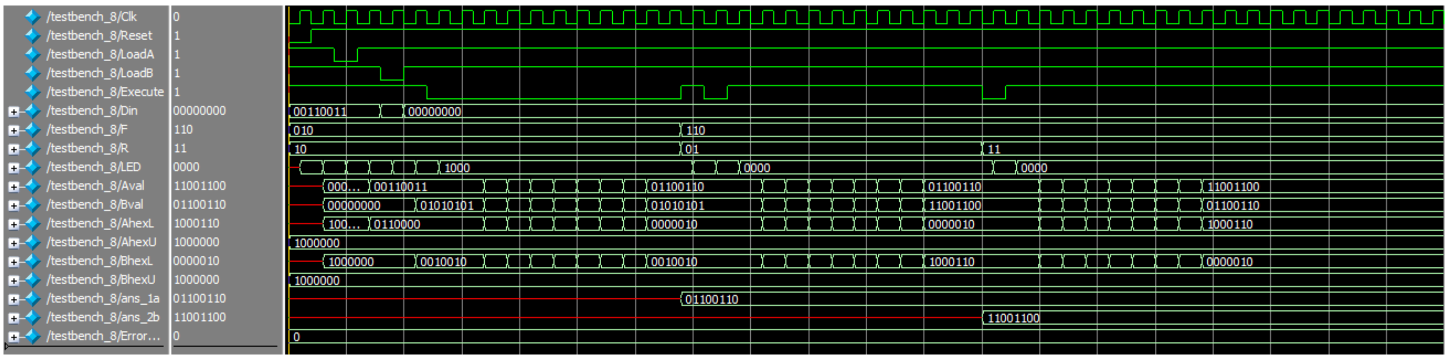
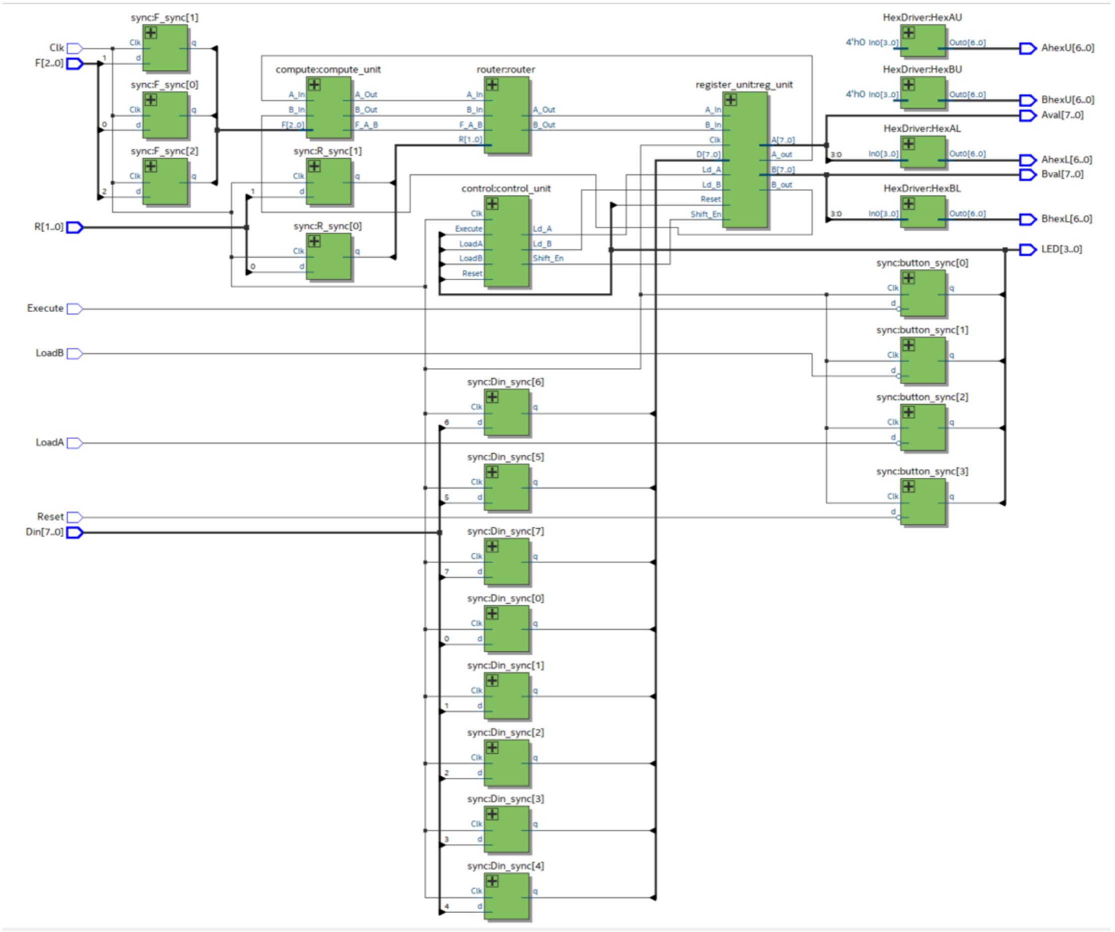


Figure 1 Simulation graph


```
/testbench_8/Clk        0
/testbench_8/Reset      1
/testbench_8/LoadA      1
/testbench_8/LoadB      1
/testbench_8/Execute    1
/testbench_8/Din        00000000
/testbench_8/F          110
/testbench_8/R          11
/testbench_8/LED        0000
/testbench_8/Aval       11001100
/testbench_8/Bval       01100110
/testbench_8/AhexL      1000110
/testbench_8/AhexU      1000000
/testbench_8/BhexL      0000010
/testbench_8/BhexU      1000000
/testbench_8/ans_1a     01100110
/testbench_8/ans_2b     11001100
/testbench_8/Error...   0
```

As we can see in the graph, the inputs are stored in Aval and Bval. Ans_1a and ans_2b denote the output of the calculation. In this graph, an XOR operation was performed and followed by a swap operation.

Figure 2 RTL View

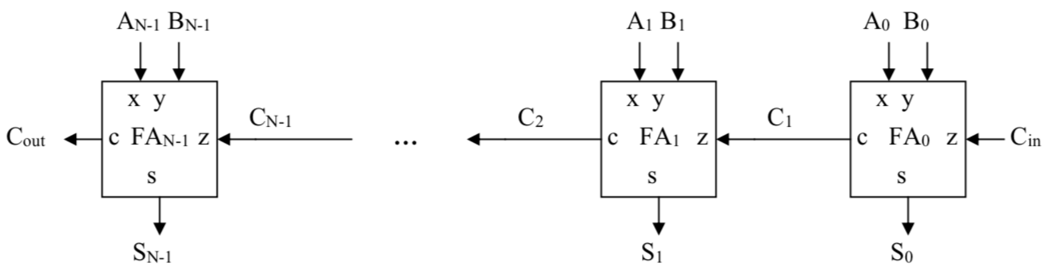


Adders:

Ripple Carry Adder:

Description:

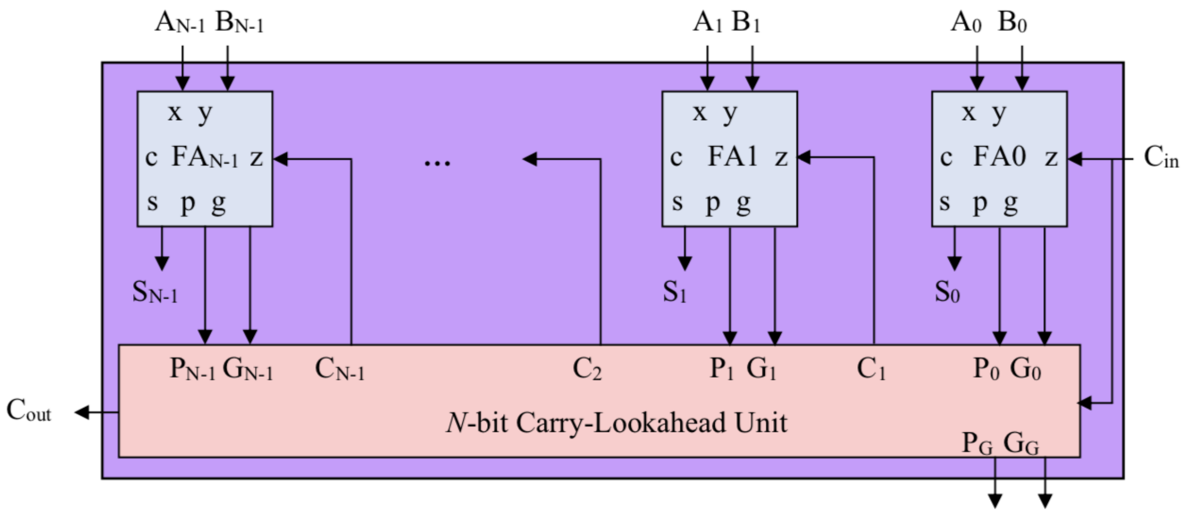
The ripple adder was accomplished by connecting several full adders together. Each full adder would be responsible for one bit. The i^{th} full adder would take the i^{th} bit of A and the i^{th} bit of B, and a carry in bit. The output should be the i^{th} bit of the result and the carry in bit for the $(i+1)^{th}$ full adder. Since every full adder expect the first one needed the carry out bit of the previous full adder carry out, the n^{th} full adder would need the previous n-1 full adders to finish, so the run time of the ripple carry adder was very slow.



Carry Lookahead Adder:

Description:

The N-bit carry lookahead adder was consisted of N full adders and a N bit carry lookahead unit. Instead of waiting for the previous full adder carry out bit like RCA, the carry lookahead adder would compute two internal bits generating (G) and propagation(P) using the current available inputs. Then G and P would be used to calculate the next full adder's carry in bit. By doing this, we could compute the output of each bit in parallel, which improved the runtime.



Description of the P and G logic:

The ith P and ith G were calculated based on the following equations:

$$P_i(A_i, B_i) = A_i \oplus B_i \qquad G(A_i, B_i) = A_i \cdot B_i$$

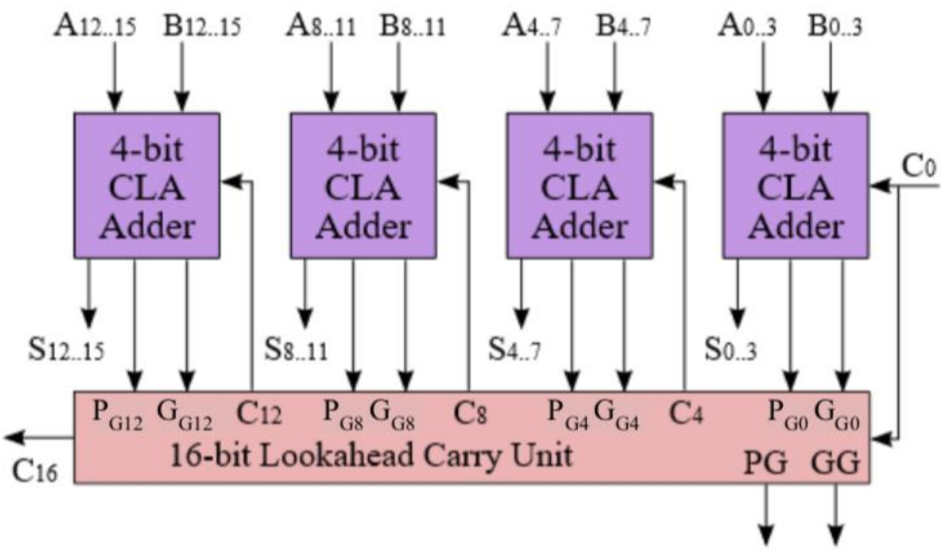
Then, the (i+1)th carry in bit could be calculated by $C_{i+1} = G_i + (P_i \cdot C_i)$. However, if C_{i+1} was depended on C_i, the runtime was not improved at all. Instead, we should expend the C_{i+1} as followings in 4-bit CLA unit:

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

Description of hierarchical 4x4 adder:

To design a 16-bit carry lookahead adder, we would use four 4-bit carry lookahead adders in a hierarchical structure. In specific, each carry 4-bit carry would generate a P and G, and they would be used to generate the next 4-bit CLA adder carry in. The carry in bit for each 4-bit CLA adder was calculated by the followings in 16-bit CLA unit:

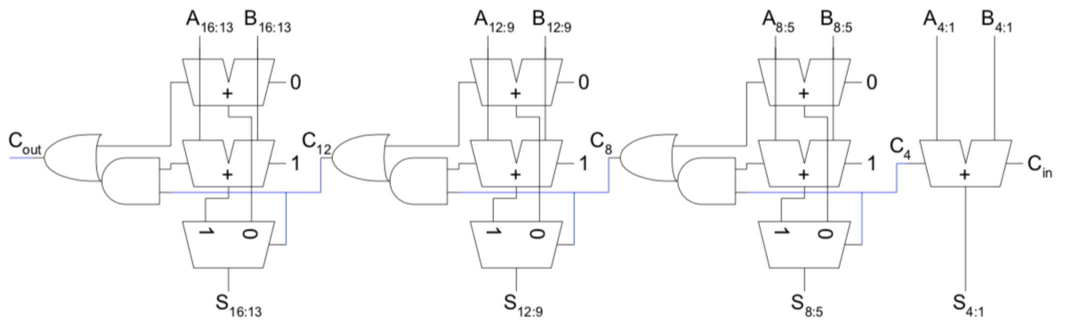
$$C_4 = G_{G0} + C_0 \cdot P_{G0}$$
$$C_8 = G_{G4} + G_{G0} \cdot P_{G4} + C_0 \cdot P_{G0} \cdot P_{G4}$$
$$C_{12} = G_{G8} + G_{G4} \cdot P_{G8} + G_{G0} \cdot P_{G8} \cdot P_{G4} + C_0 \cdot P_{G8} \cdot P_{G4} \cdot P_{G0}$$
$$\dots$$



Carry Select Adder:

Description:

Carry select adder dealt with the carry in bit by having all possible combinations of carry in bits ready, and when the doing the actual computation using the input data, the actual carry in bits were selected based on the current inputs.



For the 16-bit CSA, we used 4x4 hierarchical structure, the first 4 bits were calculated using a simple 4 bit carry ripple adder because the carry in bit was available. For the next 4 bits, we would calculate both results when carry in bit was 1 and 0 from first 4 bits. Then, we would use a 2-to-1 MUX to select which one was the actual result by using the carry out bit from first four bits CRA C_4 as the select bit. Then, the carry out of the current four bits was selected using a simple circuit logic as showed in the graph. If the carry in of the current four bits was 1, then the carry out would be selected to be the carry out of 4-bit CRA with carry in of 1, and vice versa if the carry in of the current four bits was 0. The remaining bits followed the same structure as showed in the graph above.

CSA was able to speed up the runtime because it had all possible combinations of carry in bits ready by doubling the number of hardware. Thus, when the inputs came to the circuit, all of the possible results were computed in parallel and the actual outputs would be ready based on the actual inputs.

Area, complexity, and performance tradeoffs between the adders:

Area:

CSA would need significantly more areas comparing to the other two adders, due to the reason that it used two separate sets of hardware to compute the outputs for both possible carry in bits. On the other hand, CRA required least amount of area since it only required N full adders for N bits inputs and on additional logic gates are required. CLA required additional hardware for the carry lookahead unit comparing to CRA, but it still required less hardware than CSA.

Complexity:

CLA was the most complex design amount three kinds of adders, since it needed the developer to carefully design the carry lookahead unit to deal with the Ps and Gs. In contrast, CRA was the simplest one since it just simply connected all full adders in serial. CSA needed to deal with double amounts of hardware, but it was relatively easy to design and implement.

Performance:

CSA would be the one with best performance by trading of the number of hardware. CLA also had fair performance thanked to its hieratical structure. CRA was the slowest one among three adders.

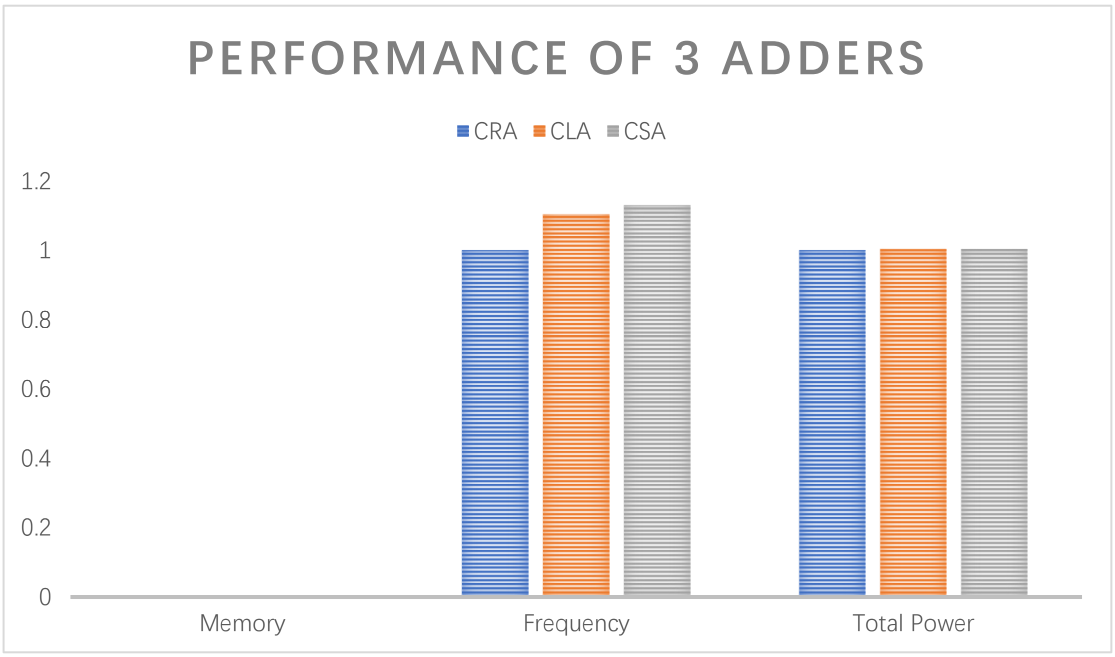


Figure 3 Performance (Memory are all 0)

Post-Lab Question:

Q: 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?

	8-bit Logic Processor
LUT	73
DSP	0
Memory (BRAM)	0
Flip-Flop	43
Frequency	363.77MHZ
Static Power	89.94mW
Dynamic Power	0mW
Total Power	98.71mW

A: In lab 4, we only used a few chips for the serial processor whereas the System Verilog used much more resources than the TTL design. This was because the chip usage of TTL was design specifically for the circuit and lots of the truth tables and the logic designs were accomplished by human. However, these designs were generated by the System Verilog compiler based on our codes, and there may be some optimization and tradeoffs for the better performance. Thus, the TTL design is better in terms of the resource usage and System Verilog is better in terms of the difficulties and performance.

Q: 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?)

A: We can further improve the performance of 4x4 hierarchy. If we want the best performance, we can do a 16x1 hieratical structure, which means we will implement all possible combinations of inputs. However, this needs significantly more hardware and power. To balance the performance, the power consumption, and the number of hardware, we could start with an 1x16 hierarchy structure, which was simply a 16-bit carry ripple adder. We could then do 2x8 structure, 4x4 structure, 8x2 structure, and then finally 16x1 structure. Every time, we kept the information of LUT, DSO, BRAM, number of Flip-Flop, frequency, and the power consumption. Finally, we could compare those data and find the one that best fit our needs of performance, power consumption, and etc.

Q: For the adders, refer to the Design Resources and Statistics in IQT.16-18 and

complete the following design statistics table for each adder. This is more comprehensive than the above design analysis and is required for every System Verilog circuit.

	CRA	CLA	CSA
LUT	77	87	82
DSP	0	0	0
Memory (BRAM)	0	0	0
Flip-Flop	20	20	20
Frequency	86.36MHZ	95.27MHZ	97.65MHZ
Static Power	89.97mW	89.97mW	89.97mW
Dynamic Power	1.45mW	1.68mW	1.76mW
Total Power	105.2mW	105.49mW	105.57mW

Q (cnt): 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?

A: From the table, we could see that the CRA has the least number of LUT while CLA has greatest number of LUT. This was expected because CRA has the easiest logic while CLA has the most complicated logic design. This also reflected in the total power consumption. What's surprising is that even though CLA uses more LUTs than the CSA, its power consumption is less than that of CSA. We could also see that CSA has the highest operating frequency, while CLA has the lowest frequency. This was also expected because CSA needed to do much more operations than CRA and CLA, and it is the fastest among these three while CRA is the slowest.

Conclusion:

Bug Encountered:

1. When extending 4-bit processor to 8-bit processor, wrong bits were entered, so we had errorcnts in the output. Also, we forgot to update the number of states, which results in the inconsistency within the code.
2. When we were designing the CLA, we forgot to use the Ps and Gs to correctly calculate the carry outs. Instead, we used the carry outs of the previous CLA directly. In result, only the first 4 bits were calculated correctly.

Lab Manual Issues:

Pro:

1. The video tutorials of for the CRA and testbench are very useful.
2. The documents for the three adders are explained very well.

Additional:

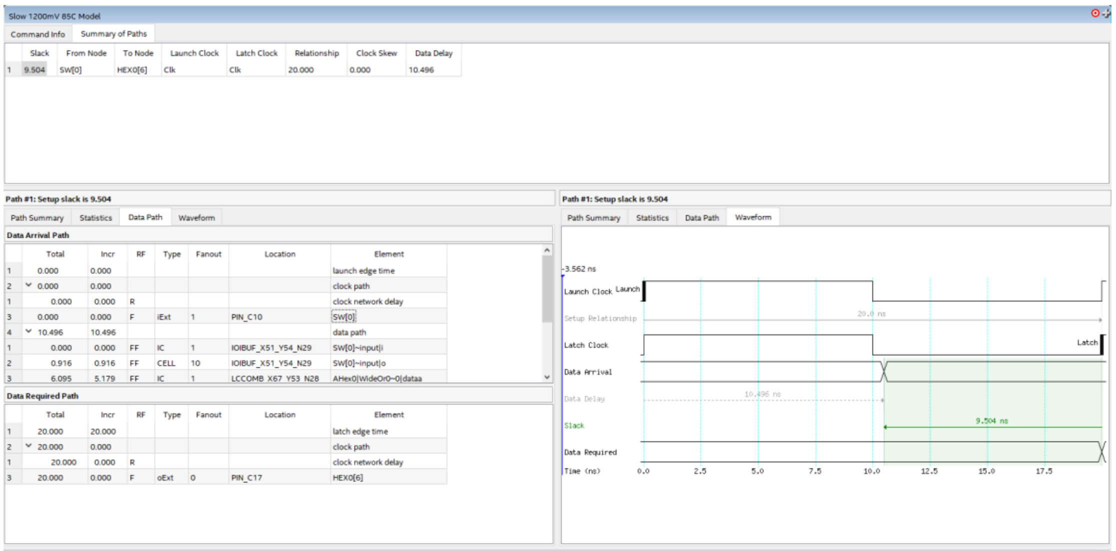
1. The use of testbench can be explained more.

Summary:

In this lab, we designed an 8-bit processor and three kinds of adders using system verilog. We compared the hardware use, performance, and the power complexity of three kinds of the adders. In addition, we used the testbench to simulate the hardware computation in software.

Extra Credit:

Time Analysis For Ripple Adder:


Slow 1200mV 85C Model

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	9.504	SW[0]	HEX0[6]	Clk	Clk	20.000	0.000	10.496

Path #1: Setup slack is 9.504

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	0.000	0.000	F	iExt	1	PIN_C10	SW[0]
4	10.496	10.496					data path
1	0.000	0.000	FF	IC	1	IOIBUF_X51_Y54_N29	SW[0]~input|i
2	0.916	0.916	FF	CELL	10	IOIBUF_X51_Y54_N29	SW[0]~input|o
3	6.095	5.179	FF	IC	1	LCCOMB_X67_Y53_N28	AHex0|WideOr0~0|dataa

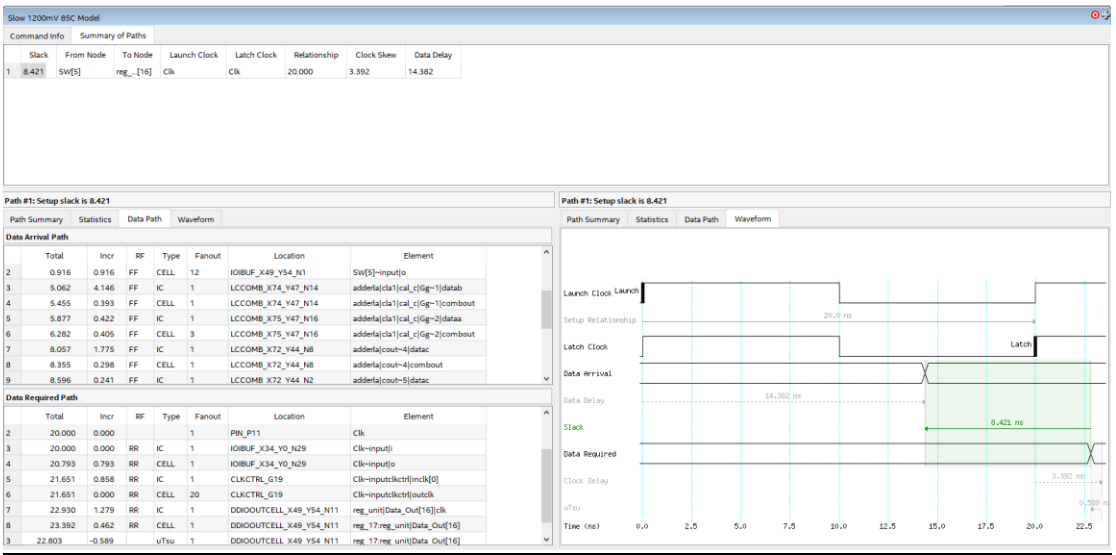
Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	20.000	20.000					latch edge time
2	20.000	0.000					clock path
1	20.000	0.000	R				clock network delay
3	20.000	0.000	F	oExt	0	PIN_C17	HEX0[6]

Path #1: Setup slack is 9.504

Path Summary | Statistics | Data Path | Waveform

Time Analysis For Lookahead Adder:



	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	8.421	SW[5]	reg_...[16]	Clk	Clk	20.000	3.392	14.382

Path #1: Setup slack is 8.421

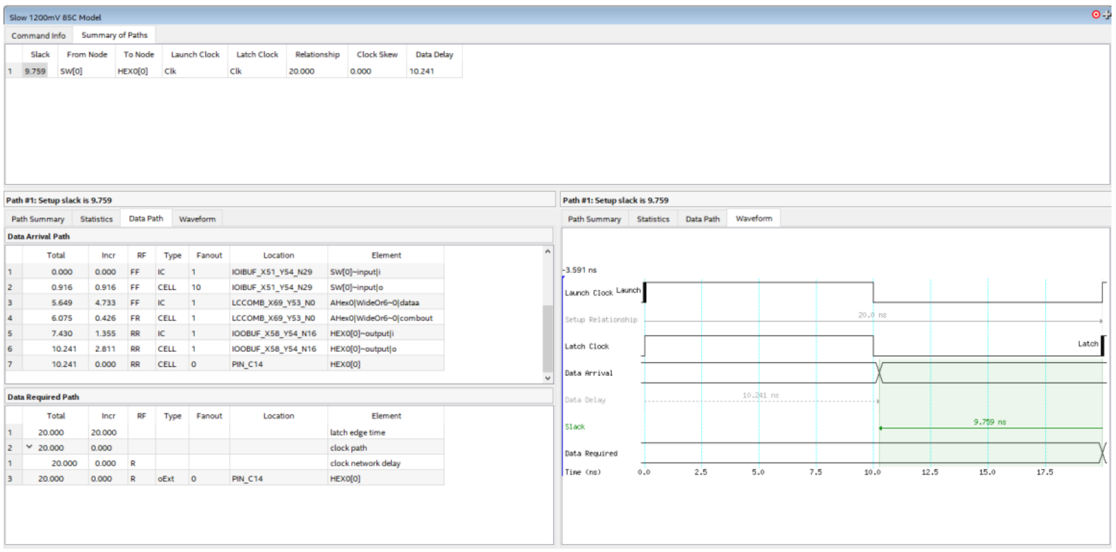
Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
2	0.916	0.916	FF	CELL	12	IOIBUF_X49_Y54_N1	SW[5]~input|o
3	5.062	4.146	FF	IC	1	LCCOMB_X74_Y47_N14	adderla|cla1|cal_c|Gg~1|datab
4	5.455	0.393	FF	CELL	1	LCCOMB_X74_Y47_N14	adderla|cla1|cal_c|Gg~1|combout
5	5.877	0.422	FF	IC	1	LCCOMB_X75_Y47_N16	adderla|cla1|cal_c|Gg~2|dataa
6	6.282	0.405	FF	CELL	3	LCCOMB_X75_Y47_N16	adderla|cla1|cal_c|Gg~2|combout
7	8.057	1.775	FF	IC	1	LCCOMB_X72_Y44_N8	adderla|cout~4|datac
8	8.355	0.298	FF	CELL	1	LCCOMB_X72_Y44_N8	adderla|cout~4|combout
9	8.596	0.241	FF	IC	1	LCCOMB_X72_Y44_N2	adderla|cout~5|datac

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
2	20.000	0.000			1	PIN_P11	Clk
3	20.000	0.000	RR	IC	1	IOIBUF_X34_Y0_N29	Clk~input|i
4	20.793	0.793	RR	CELL	1	IOIBUF_X34_Y0_N29	Clk~input|o
5	21.651	0.858	RR	IC	1	CLKCTRL_G19	Clk~inputclkctrl|inclk[0]
6	21.651	0.000	RR	CELL	20	CLKCTRL_G19	Clk~inputclkctrl|outclk
7	22.930	1.279	RR	IC	1	DDIOOUTCELL_X49_Y54_N11	reg_unit|Data_Out[16]|clk
8	23.392	0.462	RR	CELL	1	DDIOOUTCELL_X49_Y54_N11	reg_17:reg_unit|Data_Out[16]
3	22.803	-0.589		uTsu	1	DDIOOUTCELL_X49_Y54_N11	reg_17:reg_unit|Data_Out[16]

Time Analysis For Carry Select Adder:



	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	9.759	SW[0]	HEX0[0]	Clk	Clk	20.000	0.000	10.241

Path #1: Setup slack is 9.759

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000	FF	IC	1	IOIBUF_X51_Y54_N29	SW[0]~input|i
2	0.916	0.916	FF	CELL	10	IOIBUF_X51_Y54_N29	SW[0]~input|o
3	5.649	4.733	FF	IC	1	LCCOMB_X69_Y53_N0	AHex0|WideOr6~0|dataa
4	6.075	0.426	FR	CELL	1	LCCOMB_X69_Y53_N0	AHex0|WideOr6~0|combout
5	7.430	1.355	RR	IC	1	IOOBUF_X58_Y54_N16	HEX0[0]~output|i
6	10.241	2.811	RR	CELL	1	IOOBUF_X58_Y54_N16	HEX0[0]~output|o
7	10.241	0.000	RR	CELL	0	PIN_C14	HEX0[0]

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	20.000	20.000					latch edge time
2	20.000	0.000					clock path
1	20.000	0.000	R				clock network delay
3	20.000	0.000	R	oExt	0	PIN_C14	HEX0[0]

The Carry Lookahead adder uses least slack for the data to arrive while the carry select adder uses most slack for the data to arrive. This may be caused by the fact that the carry select adder have to compute all the conditions while carry lookahead adder only computes the combination of G and P which are fewer than those in carry select adder.