ECE 385
Lab 4
Introduction to SystemVerilog, FPGA, EDA, and 16-bit Adders

Li, Ziying
Kang, Xingjian

March 15, 2024

1 Introduction

This experiment focuses on teaching the principles of RTL (register-transfer level) design on a Field-Programmable Gate Array (FPGA) using the programming language SystemVerilog. The performance analysis and optimization features of Quartus Prime will be utilized to create three different types of adders: a 16-bit carry-ripple adder, a 16-bit carry-look-ahead adder, and a 16-bit carry-select adder; we also build an 8-bit Serial Logic Processor based on the 4-bit Serial Logic Processor built in our previous lab.

2 Part 1 - Serial Logic Processor

2.1 block diagram of the bit-serial logic processor

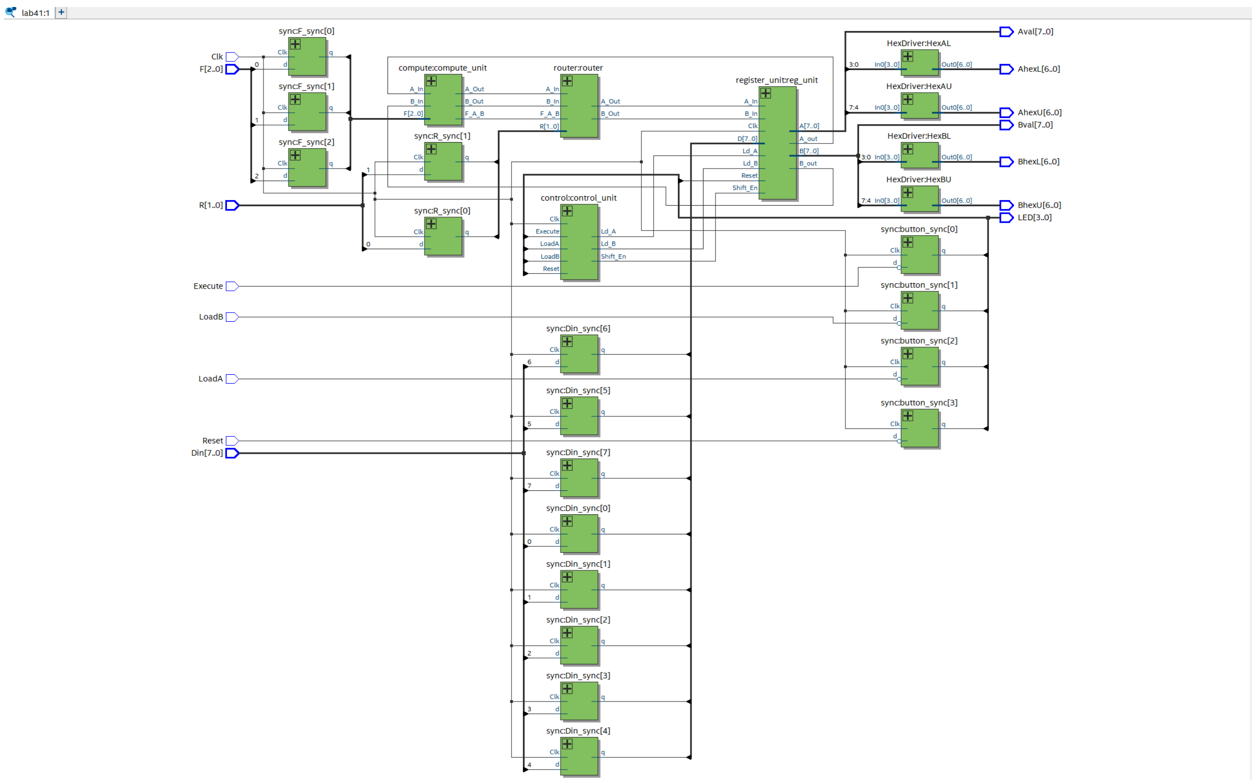
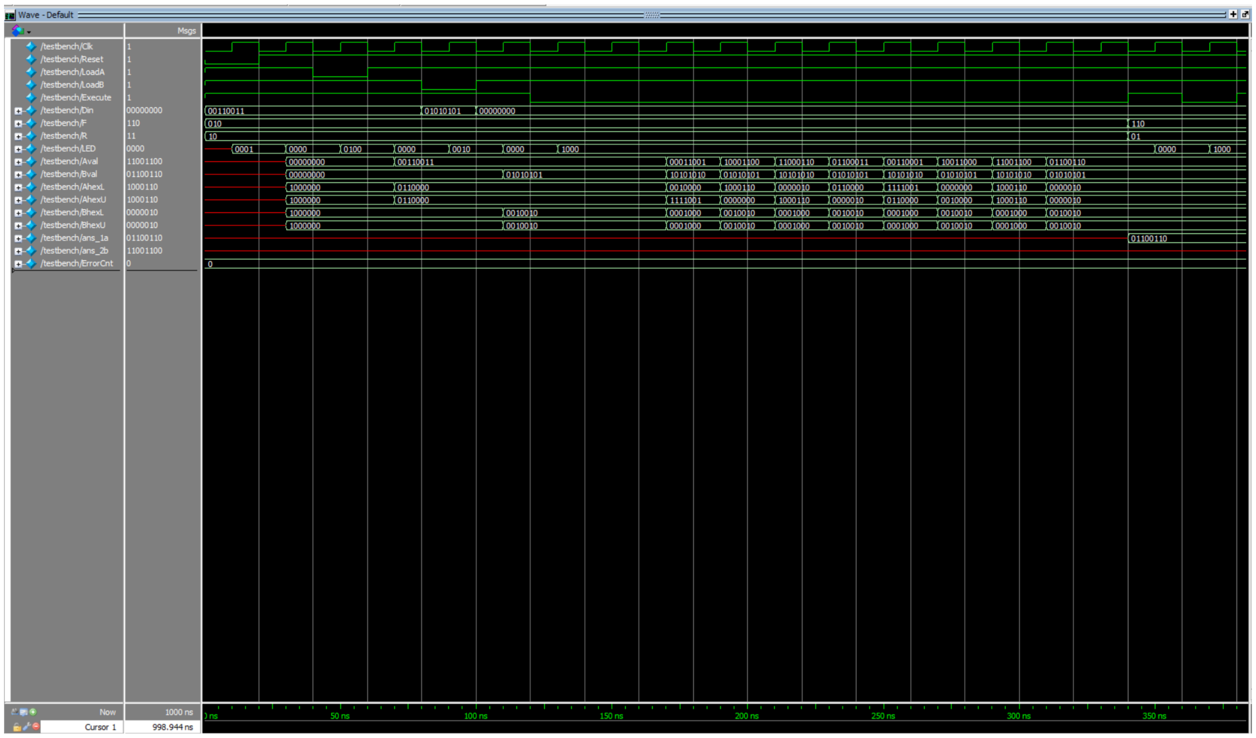


Figure 1: Schematic block diagram of 8-bit processor

2.2 what was done with the provided code to extend it from 4 bits to 8 bits

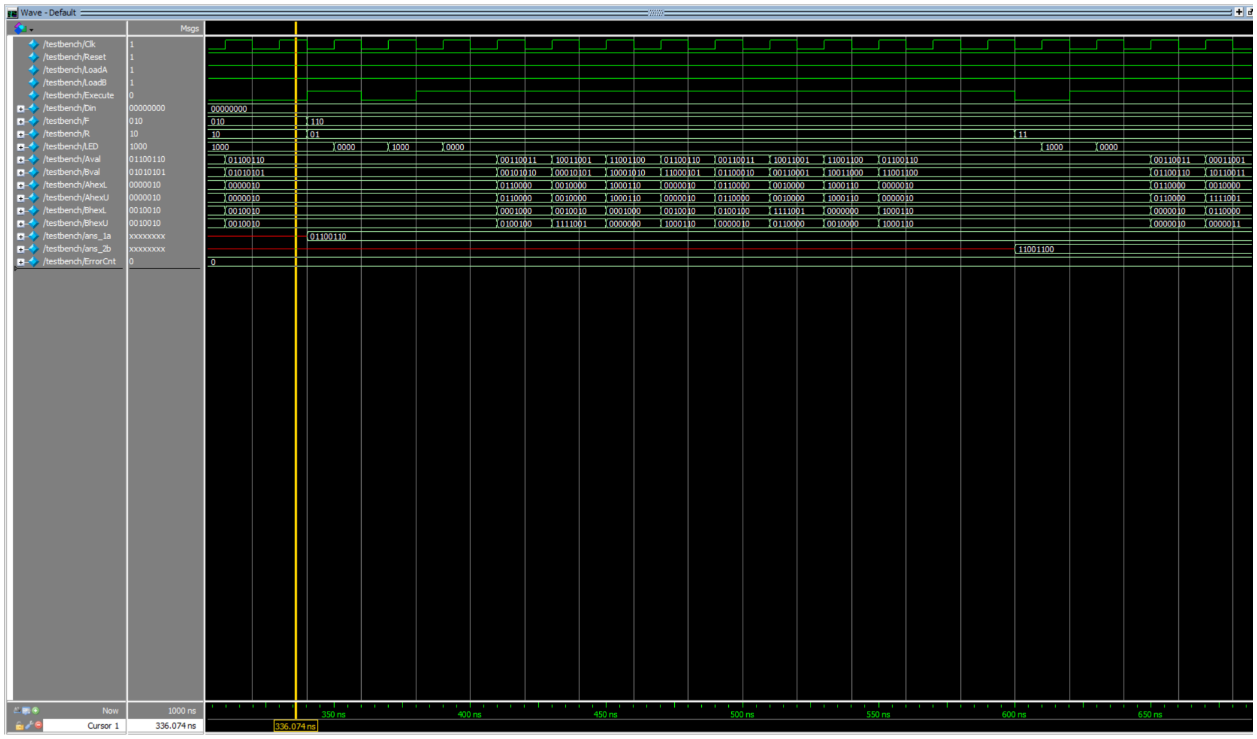
First, we expand all the arrays that were used for A, B, Din from [3:0] to [7:0]. Then we modify the control unit to let it have other 4 state for the 4 more bits. Last, we add the hexdrivers to display the new bits.

2.3 simulations of the bit-serial logic processor



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

Figure 2: Aval is expected to be 8'h33 XOR 8'h55, Bval is expected to be the original 8'h55



```
/testbench/Clk        1
/testbench/Reset      1
/testbench/LoadA      1
/testbench/LoadB      1
/testbench/Execute    0
/testbench/Din        00000000
/testbench/F          010
/testbench/R          10
/testbench/LED        1000
/testbench/Aval       01100110
/testbench/Bval       01010101
/testbench/AhexL      0000010
/testbench/AhexU      0000010
/testbench/BhexL      0010010
/testbench/BhexU      0010010
/testbench/ans_1a     xxxxxxxx
/testbench/ans_2b     xxxxxxxx
/testbench/ErrorCnt   0
```

Figure 3: Aval is expected to stay the same, Bval is expected to be the answer of 1st cycle XNOR 8'h55

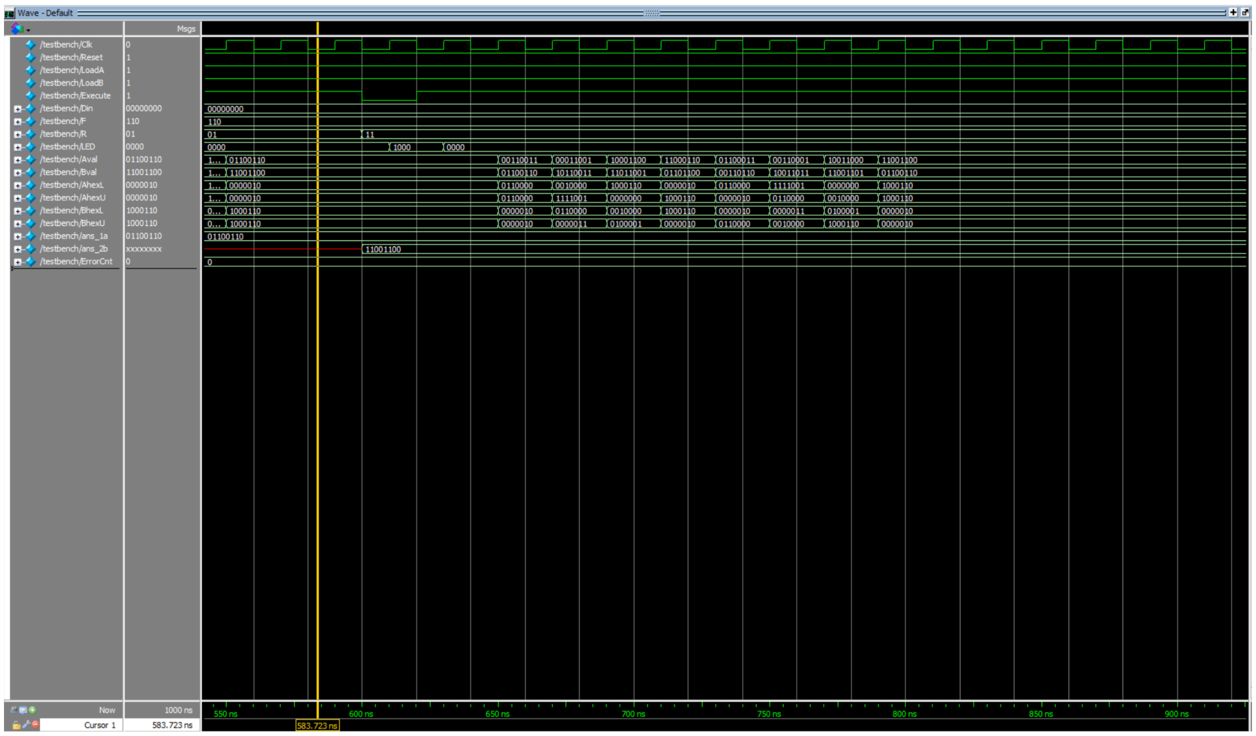


Figure 4: Aval and Bval are expected to swap

3 Part 2 - Adders

3.1 Ripple Carry Adder

3.1.1 Written description of the architecture of the adder

The fundamental concept is implemented by linking together full adders in a cascading manner. More specifically, an 16-bit Carry-Ripple Adder can be constructed by connecting 16 full adders in a series, and the carry signal is transmitted sequentially from one full adder to the next full adder via a traveling wave.

3.1.2 Block diagram

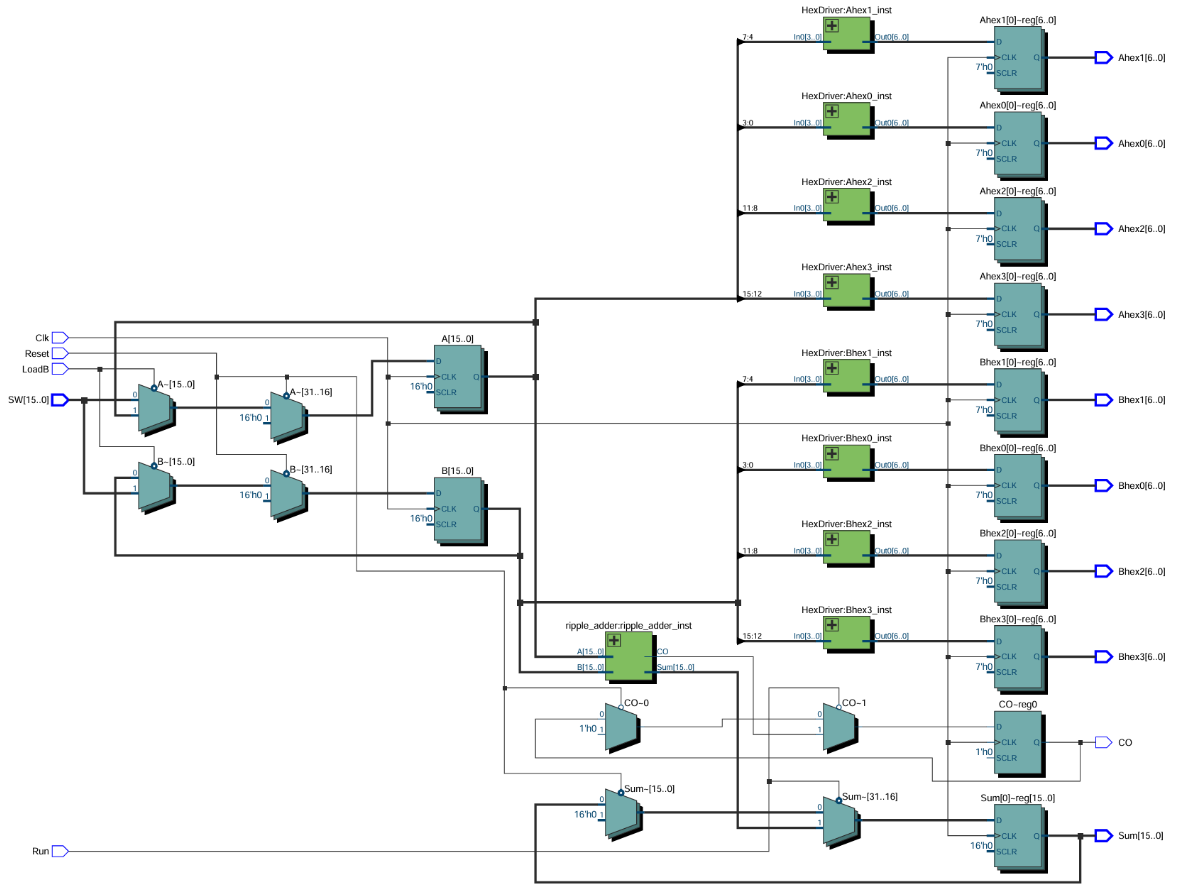


Figure 5: CRA

3.2 Carry Lookahead Adder

3.2.1 Written description of the architecture of the adder

In Carry-Ripple Adders, the two bits that are to be added in each adder block are immediately accessible. Nevertheless, every adder block remains idle until it receives the carry signal from the preceding block. So, it is not possible to generate the sum and carry of any block until the input carry is known.

The technique the Carry-Lookahead Adder uses is called lookahead technique. More specifically, instead of waiting on the actual carry-in values, it makes use of the Propagating (P) and Generating (G) logics to predict the carry-out bit for every pair of input bits A and B. A carry-out is generated (G) only when both accessible inputs (A and B) are set to 1, independent of the carry-in. The equation is G(A, B) = A AND B. Alternatively, a carry-out can be propagated (P) if either A or B is equal to 1. This condition can be expressed as P(A, B) = A XOR B.

$$P_G = P_0P_1P_2P_3$$
$$G_G = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$$

To achieve a 16-bit CLA, we build a 4-bit Carry Look-Ahead Unit and cascade 4 of it together, i.e., there are four groups of four bits each created from the 16-bit inputs A and B. The detailed logic is as follows:

3.2.2 Block diagram

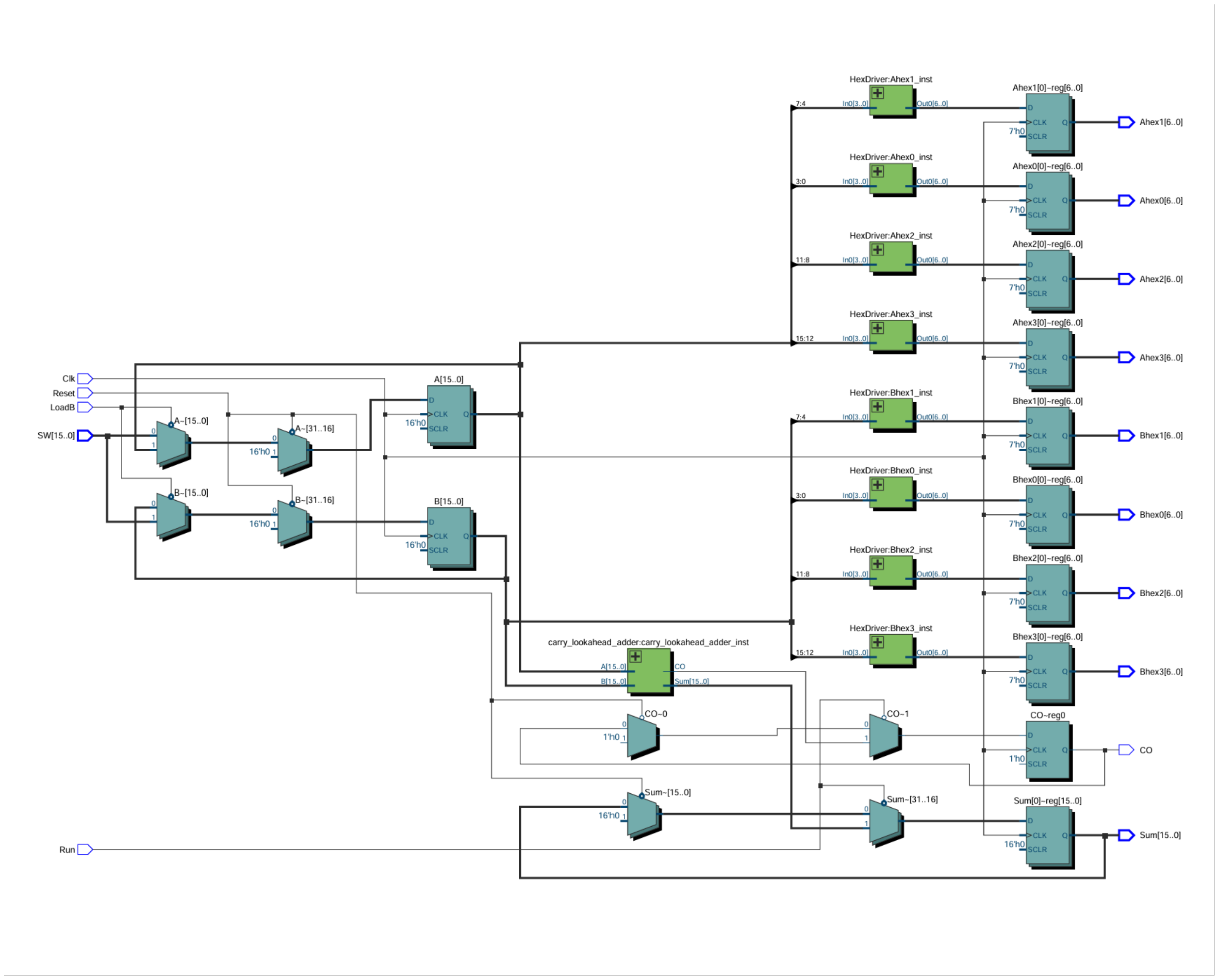


Figure 6: CLA

3.3 Carry Select Adder

3.3.1 Written description of the architecture of the adder

The Carry-Select Adder unit is composed of two Ripple Carry Adders (RCAs) and one selector. One of the RCAs assumes a carry-in value of 0, while the other assumes a carry-in value of 1. To achieve a 16-bit CSA, we build a 4-bit Carry-Select Unit and cascade them together, i.e., there are four groups of four bits each created from the 16-bit inputs A and B. Note that for the first four bits, there is no need for a carry choose block because carry-in is known at the start of the computation. This adder will have four complete adder delays and three MUX delays.

3.3.2 Block Diagram of the whole CSA circuit containing adders, multiplexers,and glue logic

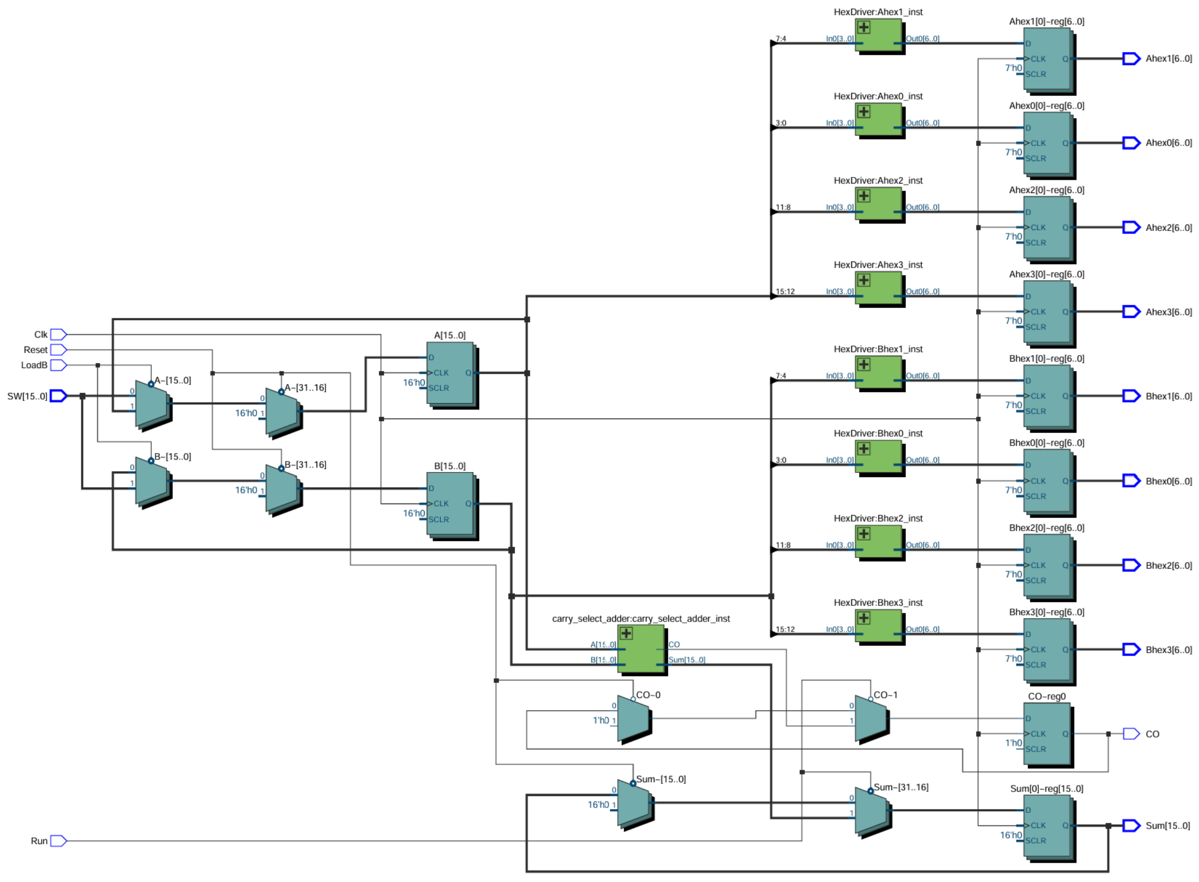


Figure 7: CSA

3.4 Written Description of all .sv Modules

3.4.1 Carry-Ripple Adder (CRA)

1) **Module name:** $full\_adder$

   Input: A, B, c_in

   Output: Sum, CO

   Purpose: Adds two binary numbers and sums them by means of gate circuits.

   operation: Takes 1-bit A, B and 1 carry-in bit c_in as inputs and provides a carry-out bit CO and the result Sum.

2) **Module name:** $ripple\_adder$

   Input: $[15:0]A, [15:0]B$

   Output: $[15:0]Sum, CO$

   Purpose: Adds 16-bit numbers using 16 full adders.

operation: Takes 16-bit A, B as inputs and provides a carry-out bit CO and the 16-bit result $[15 : 0]Sum$.

3.4.2 Carry-Lookahead Adder (CLA)

1) **Module name:** bit_full_adder

   Input: A, B, C

   Output: P, G, S

   Purpose: Adds two binary numbers and sums them by means of gate circuits.

   operation: Takes 1-bit A, B and 1 carry-in bit C as inputs and provides a carry-out bit CO and the result Sum.

2) **Module name:** CLU

   Input: $P, G, C0$

   Output: $PG, GG, [4 : 1]C1to4$

   Purpose: A 4-bit CLA unit.

   operation: Outputs group propagate PG, the group generate GG, and the next carry bits based on formulas.

3) **Module name:** $ahead\_adder\_4bit$

   Input: $[3 : 0]A, [3 : 0]B, Ci$

   Output: $[3 : 0]Y, PG, GG, Co$

   Purpose: Works as a 4-bit Look-Ahead Adder unit for 16-bit one. Uses 4 full adders for each bit, and each full adder will give a P and G values for predicting the carry-out bit.

   operation: Takes 4-bit A, B and 1 carry-in bit ci as inputs and provides a carry-out bit Co, the 4-bit result Y, and PG and GG values for predicting the carry-out bit.

4) **Module name:** $carry\_lookahead\_adder$

   Input: $[15 : 0]A, [15 : 0]B$

   Output: $[15 : 0]Sum, CO$

   Purpose: Works as a 16-bit Look-Ahead Adder. Uses 4 4-bit Carry-Lookahead Adders for each 4-bit sequence, and each Carry-Lookahead Adder will give values predicting the carry-out bit.

   operation: Takes 16-bit A, B as inputs and provides a carry-out bit CO and the 16-bit result Sum.

3.4.3 The Carry-Select Adder, CSA

1) **Module name:** $full\_adder$

   Input: A, B, c_in

   Output: Sum, CO

   Purpose: Adds two binary numbers and sums them by means of gate circuits.

   operation: Takes 1-bit A, B and 1 carry-in bit c_in as inputs and provides a carry-out bit CO and the result Sum.

2) **Module name:** $multiplexer2$

   Input: $i0, i1, sel$

   Output: $bitout$

   Purpose: Selects which input bit to choose depending on the selection bit.

   operation: Takes 1-bit $i0, i1$ and 1 selection bit sel as inputs and provides a selected bit $bitout$.

3) **Module name:** $carry\_select\_adder4$

   Input: $[3:0]A, [3:0]B, Cin$

   Output: $[3:0]S, cout$

   Purpose: Works as a 4-bit Carry Select Adder unit for 16-bit one.

   operation: Takes 4-bit A, B and 1 carry-in bit Cin as inputs and provides carry out bit $cout$ and the 4-bit result S.

4) **Module name:** $carry\_select\_adder$

   Input: $[15:0]A, [15:0]B$

   Output: $[15:0]Sum, CO$

   Purpose: Works as a 16-bit Carry Select Adder.

   operation: Takes 16-bit A, B as inputs and provides a carry-out bit CO and the 16-bit result Sum.

3.5 Describe at a high level the area, complexity, and performance tradeoffs between the adders

Carry Ripple Adder is the one with least complexity and lowest performance. Carry Lookahead Adder used complex mathmatic equations to accelerate the calculation to middle complexity and middle performance. Carry Select Adder used lots of full-adders and multiplexers to reach high performance with high complexity.

3.6 Design analysis comparison results from pre-lab

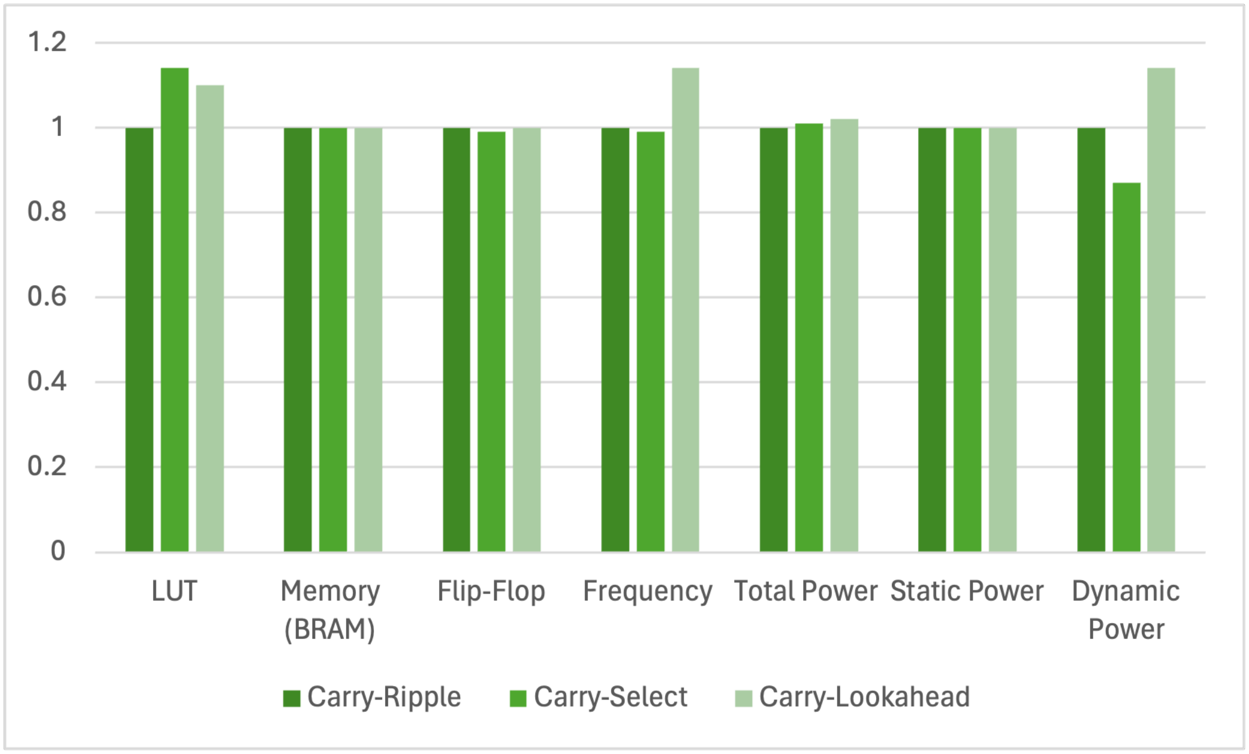


Figure 8: Normalized comparison results graph

4 Answers to post-lab questions

4.1 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?

Because of its programmability and reconfigurability, the FPGA-based bit-serial logic processor may be a superior option if the application calls for flexibility, scalability, and simplicity of design iteration.

On the other hand, the TTL-based 8-bit processor may be more appropriate if the application calls for real-time processing, low power consumption, and cost-effectiveness, particularly for jobs where processing eight bits simultaneously in parallel is beneficial.

4.2 Design statistics table for each adder

	Carry-Ripple Adder	Carry Look-Ahead Adder	The Carry Select Adder
LUT	114	130	125
DSP	0	0	0
Memory (BRAM)	0	0	0
Flip-Flop	105	104	105
Frequency	86.04MHz	85.29MHz	98.21MHz
Static Power	98.58mW	98.58mW	98.59mW
Dynamic Power	3.66mW	3.20mW	4.18mW
Total Power	166.77mW	168.90mW	169.70mW

4.3 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?

The data plot is shown as 8, and both the plot and the table above do make sense because of the following reasons:

- **LUT:** A Carry-Ripple Adder should have the smallest LUT, since it has a less complex structure.
- **Frequency:** A Carry-Look-ahead Adder should have the largest frequency, since the look-ahead technique it uses enables it to reduce the delay, thus having faster operation.

However, there is one aspect that's not as what we expected:

- **Total power:** Originally, we expected that a Carry-Ripple Adder should consume the largest total power, since each bit's carry depends on the carry propagation from the previous bit, which should consumes the largest amount of energy. However, it turns out that the Carry-Look-ahead Adder consumes the most amount of total energy. We think that's because of additional logic we use to generate the values predicting the carry-out bits.

5 Conclusion: A conclusion regarding what worked and what didn't, with explanations of any possible causes and the potential remedies.

In this lab, we construct an 8-bit Serial Logic Processor in addition to three ways to realize 16-bit adder: carry-ripple, carry-look-ahead, and carry-select adders. Processors and all adders work successfully.