ECE 385

Spring 2021

Experiment #3

16-bit Adders

Quanrui Bai, Jarod Partlo

ABJ

Xinbo Wu, Kevin Xia

Introduction

This experiment is intended to implement three kind of adders: the ripple carry adder, the carry lookahead adder, and the carry select adder. We then compare the time delay and resource usage of each design. The adders can compute the sum of two 16-bit values. Because there are not enough on-board switches to represent 16-bit numbers, we have extended the bits in the SystemVerilog files.

Adders

a. Ripple Carry Adder

Architecture: The ripple carry adder uses sixteen full adders to get the result. We designed a 4x4 hierarchy to implement the Ripple Carry Adder. However, the propagation delay is the most time-consuming element among these three adders, because every full adder will have to wait for last full adder's Carry-out bit to feed its Carry-in. The first Carry-in is obtained from user's input, and the last Carry-out indicates overflow.

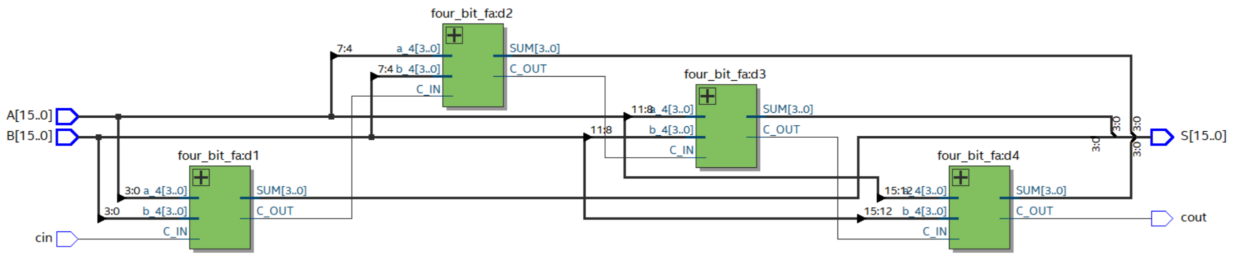


Figure 1: The 16-bit hierarchy block diagram of Ripple Carry Adder

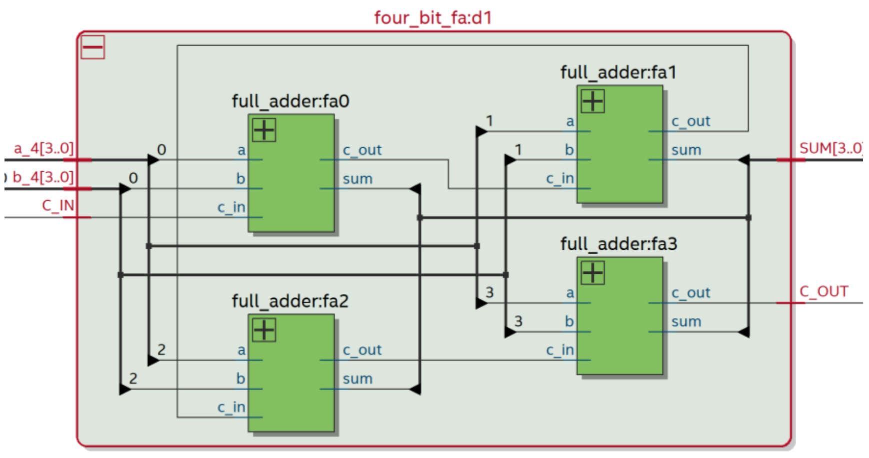


Figure 2: The internal block diagram for bitwise full adder

b. Carry Lookahead Adder

Architecture and P, G logic: The Carry Lookahead Adder contains two important logic parameters, Propagating (P) and Generating (G). Instead of using a standard full adder, the CLA will correspond each carry bit with P and G. Every bit of the CLA will make predictions about the Carry-out and Carry-in. There will be a Carry-out of 1 only when inputs A and B are 1, such that the formula for G = A & B. And the Carry-out will be propagated when either A or B is 1, so the formula for P = A xor B. Then we can get the Carry-out $C_{i+1} = G_i + (P_i \cdot C_i)$. Because the C_{i+1} can be derived from C_i, we can avoid the slow rippling of the carry bit by plug the first Carry-in (C_{in}) in the Boolean expression.

$$C_1 = C_{in} \cdot P_0 + G_0$$

$$C_2 = C_{in} \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1$$

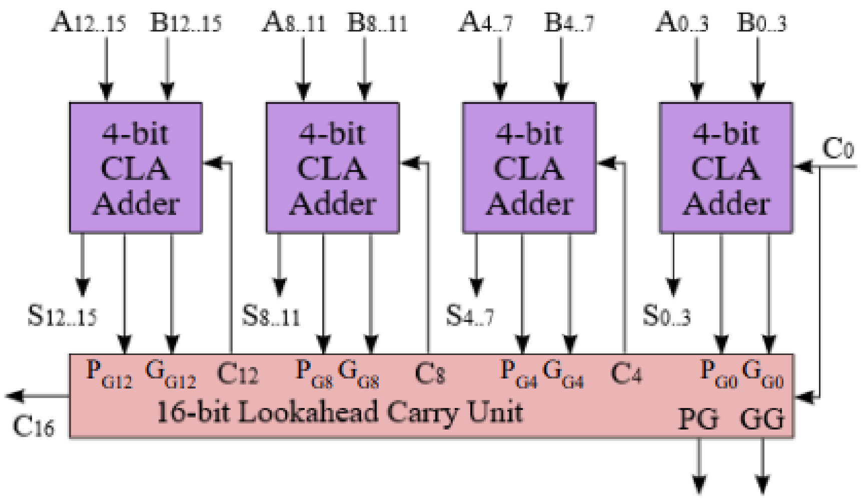


Figure 3: A 4x4-bit Hierarchical CLA Block Diagram from lab manual

We can compute every bit's carry-in/-out through the trend. However, as the number of bits increases, the number of gates needed grows dramatically, to the point where the design is impractical. Because of this limitation, we can instead design the 4-bit CLA, and use the 4-bit architecture to create a larger CLA (16-bits) in a hierarchical fashion. The inputs A and B will be divided into 4 groups with 4-bits. In the next layer, we need to compute the group value of Pg, Gg, and C.

$$Pg = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$

$$Gg = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

Then, we can take these group values as input for each 4-bit CLA, and compute their carry-out based on the formula above for the bitwise Carry-out by replacing G/P with group values.

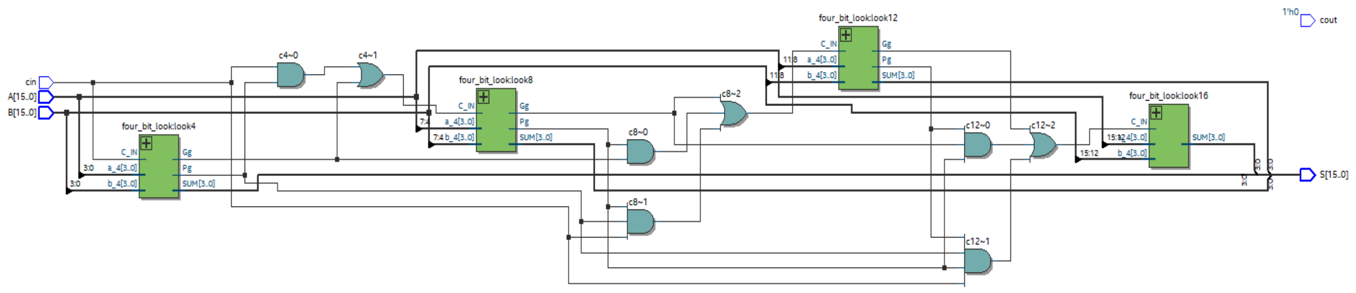


Figure 4: The sixteen-bit hierarchy block diagram of Carry Lookahead Adder

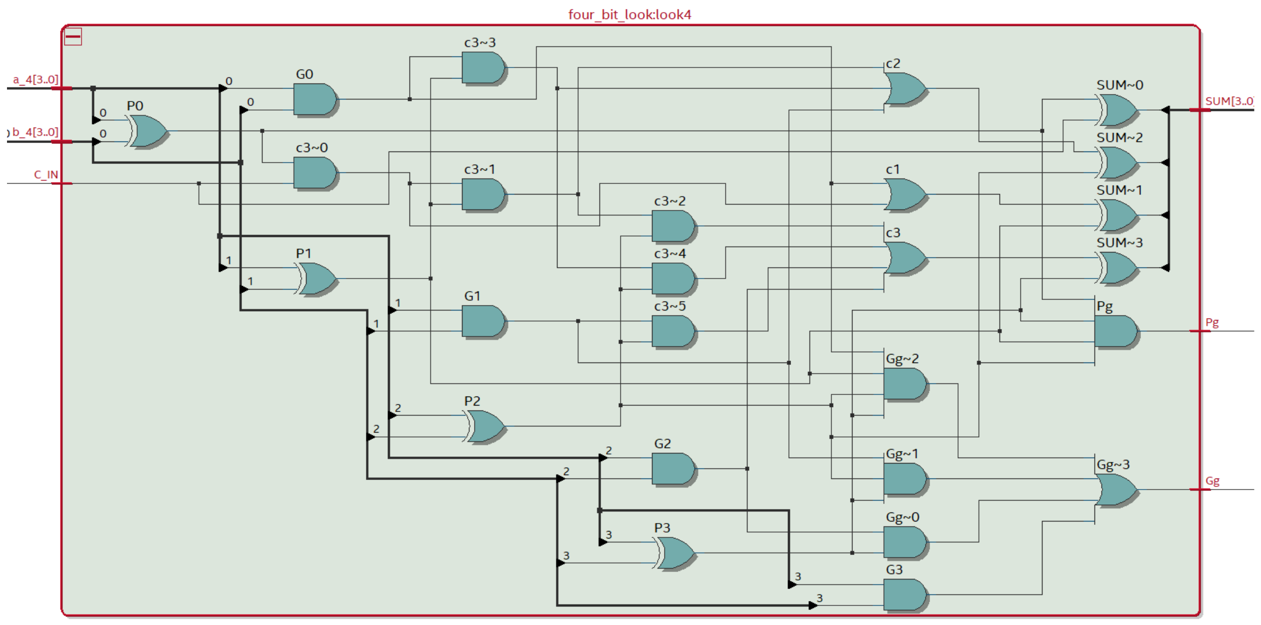


Figure 5: The internal block diagram for bitwise lookahead adders

c. Carry Select Adder

Architecture: The Carry Lookahead Adder computes two results in parallel. First we divide the 16-bit CSA into four 4-bit CSAs. The 4-bit CSA will use a ripple carry adder to compute two different 4-bit sums using different values for the Carry-in (0 or 1).

Then, the results will be fed into a multiplexer to select which result and carry-out is correct for the next 4-bit CSA. If the Carry-in from the previous CSA is zero, then the adder outputs the corresponding sum and carry-out which have already been computed. If it is 1, then it chooses the other pair of outputs. However, for the least significant adder, the Carry-in is always zero, so we only need one ripple carry adder for the first 4-bits. The rest of the 4-bit CSAs need two 4-bit ripple carry adders and one MUX.

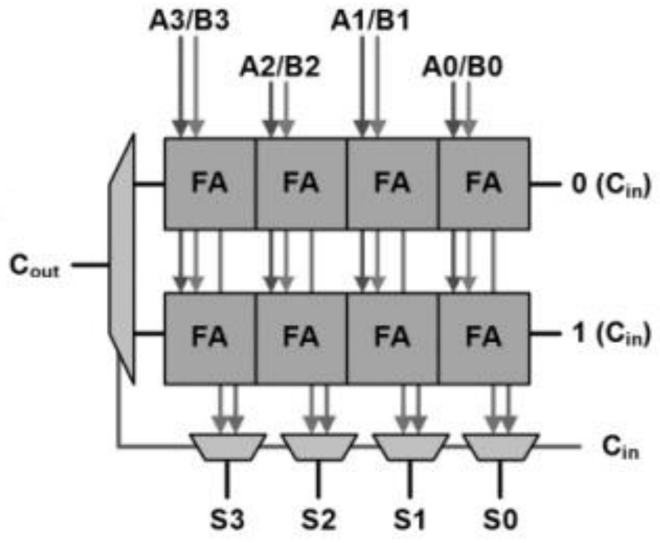


Figure 6: 4-bit CSA Block Diagram

Because each 4-bit ripple carry adder calculates its own portion of the result simultaneously and because the only additional time delay comes from the four multiplexers, this design computes the result faster than the 4x 4-bit ripple carry adder (16-bit RCA).

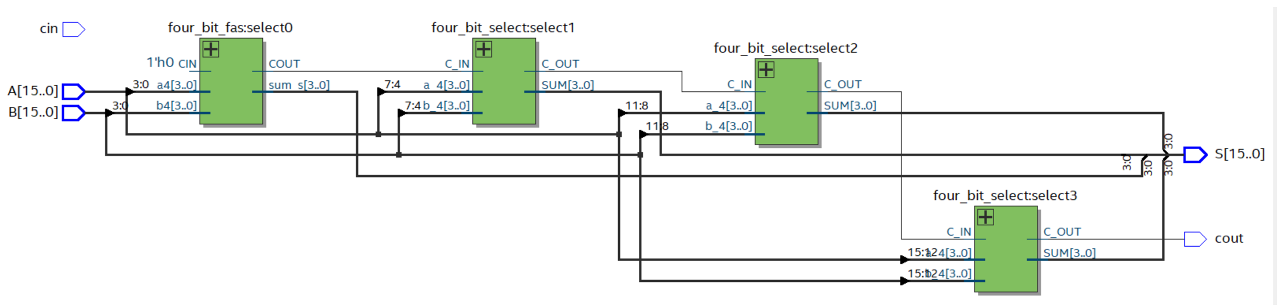


Figure 7: The sixteen-bit hierarchy block diagram of Carry Select Adder

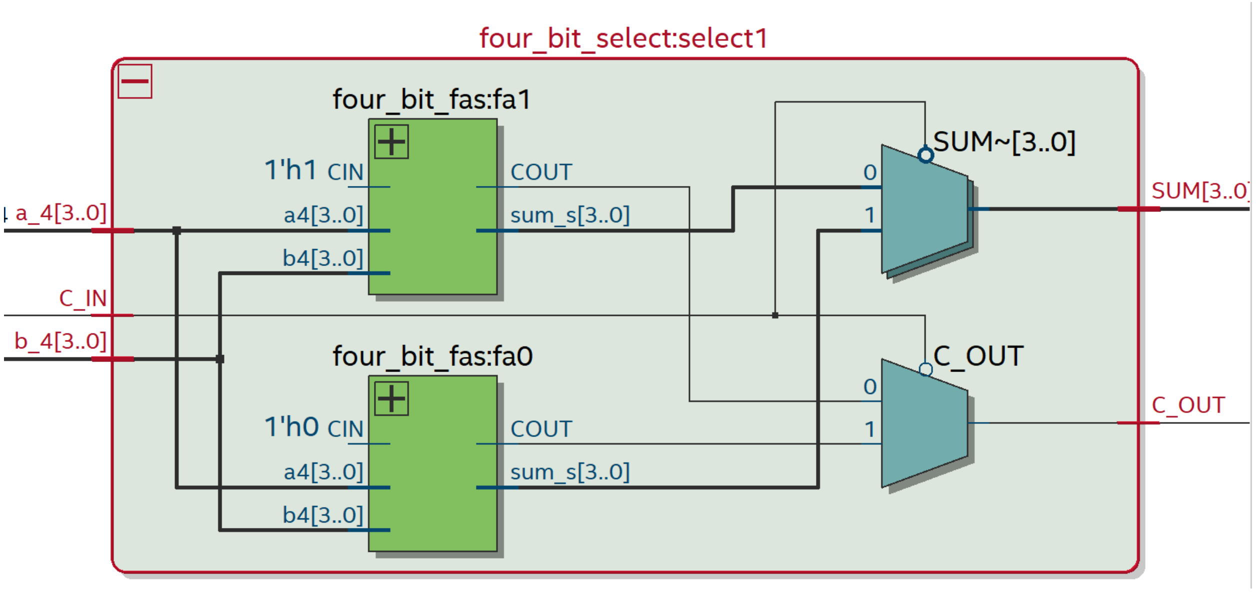


Figure 8: The internal block diagram of four-bit select adder.

d. Summary of Modules

Module: adder2.sv
Inputs: [9:0] SW, Clk, Reset, Clear, Run_Accumulate
Outputs: [9:0] LED, [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5
Description: This is the top-level design entity which will initialize the extended 16-bit SW, and set the output value for LED display.
Purpose: This will test our three adders.

Module: ripple_adder.sv
Inputs: [15:0] A, [15:0] B, cin
Outputs: [15:0] S, cout
Description: This is the 16-bit adders with four 4-bit RCA. This adder will wait the Carry in from last adder, so the propagation time is very slow.
Purpose: This module will do the 16-bit addition.

Module: lookahead_adder.sv
Inputs: [15:0] A, [15:0] B, cin
Outputs: [15:0] S, cout
Description: This is the 16-bit carry lookahead adders with four 4-bit CLA (4x4 hierarchy). This adder will make predictions on bits' carry out, so it will have logic Propagating (P), and Generating (G) for computation.
Purpose: This module will do the 16-bit addition.

Module: select_adder.sv
Inputs: [15:0] A, [15:0] B, cin
Outputs: [15:0] S, cout
Description: This is the 16-bit carry select adders with four 4-bit CSA (4x4 hierarchy). This adder will compute two results simultaneously based on the Carry in, 0 or 1. And then it will decide which one is the correct by a MUX. The first 4-bit's carry in is always 0, so it will only need a full adder. The rest of 4-bit need two full adder and one MUX.
Purpose: This module will do the 16-bit addition.

Module: reg_17.sv
Inputs: [16:0] D, Clk, Reset, Load
Outputs: [16:0] Data_Out
Description: When reset is pressed, the result will become 0.
Purpose: This module is to reset the computation result to 0.

Module: HexDriver.sv
Inputs: [3:0] In0
Outputs: [6:0] Out0
Description: This module has hardcoded values that convert 4bit binary numbers from registers into hexadecimal
Purpose: It will support the display hexadecimal onto FPGA board.

Module: control.sv
Inputs: Clk, Reset, Run
Outputs: Run_O
Description: This module is to control the states of adders.
Purpose: It will assign outputs based on the state

Module: rounter.sv
Inputs: R, [15:0] A_In, [16:0] B_In,
Outputs: [16:0] Q_Out
Description: This module is 17-bit parallel multiplexer implemented using case statements
Purpose: It is the routing

e. Area, Complexity, and Performance tradeoffs between the adders.

Carry Ripple Adder:

Area: The CRA needs the smallest area among the three adders analyzed in this experiment, it is only compromised of 16 full adders. The CRA has the simplest design of the three adders. N full-Adders are linked together in series through the carry bits, forming an N-bit binary adder.

Complexity: The CRA has the worst time efficiency in the group. Every full adder or 4-bit CRA must wait on the Carry-in from last adders before it can compute its result.

Performance: The CRA's power consumption was technically the highest, but marginally so.

Carry Lookahead Adder:

Area: The CLA has the most complex logic of any of the adders. It takes the largest area because it needs a great number of logic gates to implement the Boolean expressions for the propagating and generating logic. It needs logic to compute every bit’s G and P, as well as every 4-bit group’s G and P.

Complexity: The CLA performs the best in terms of time complexity. It is 26% faster than the CRA and 5% faster than the CSA (1.26 vs 1.20 normalized).

Performance: The CLA also has the lowest power consumption according to our analysis. The differences between the power performance of the adders was small, however.

Carry Select Adder:

Area: The CSA uses the second-highest amount of space in the LUT based on the resource summary, so it uses the second most area among three adders. It requires two sets of adders as well as several multiplexers, but still takes less space for logical operations that the CLA. Its resource usage is directly in the middle of that of the two other adder designs (CSA: 82, CLA: 87, CRA: 77).

Complexity: The CSA is a significant improvement on the CRA in terms of time efficiency. By our analysis, it arrives at results 20% faster.

Performance: Power consumption was almost exactly the same as the CRA (there was a negligible decrease in power consumption) and slightly more than the CLA.

	Carry-Ripple	Carry-Select	Carry-Lookahead
Memory(BRAM)	0	0	0
Frequency	1.0 (78.15 MHz)	1.20 (93.57 MHz)	1.26 (98.16 MHz)
Total Power	1 (105.36 mW)	1 (105.34 mW)	1 (105.16 mW)

Table 1: Design Analysis for Three Adders

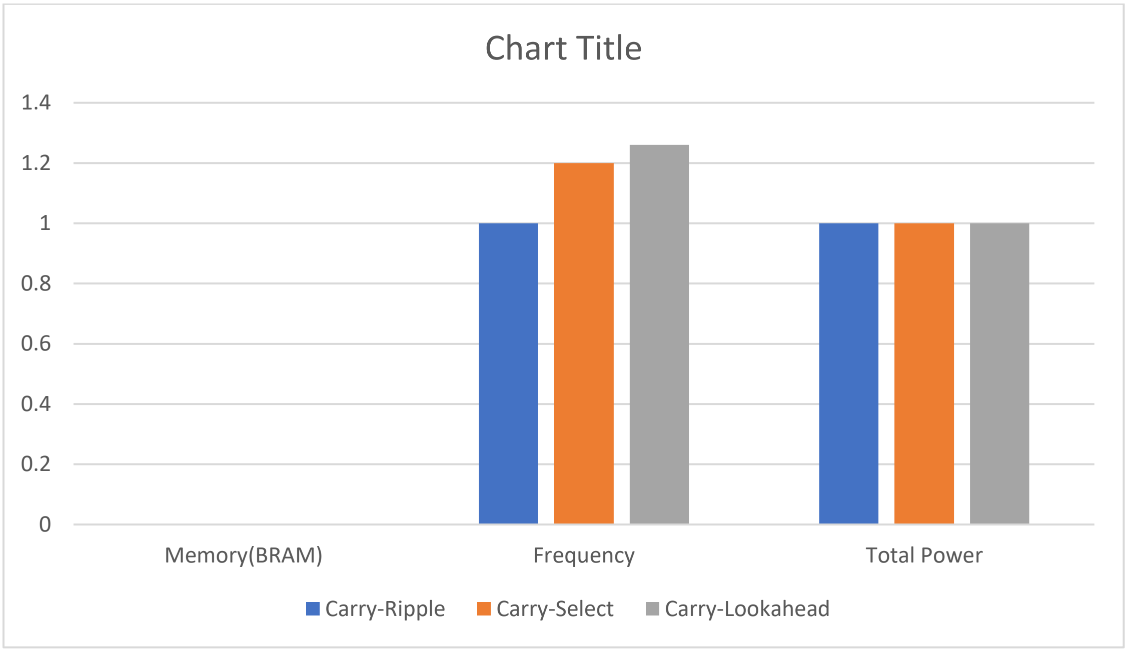
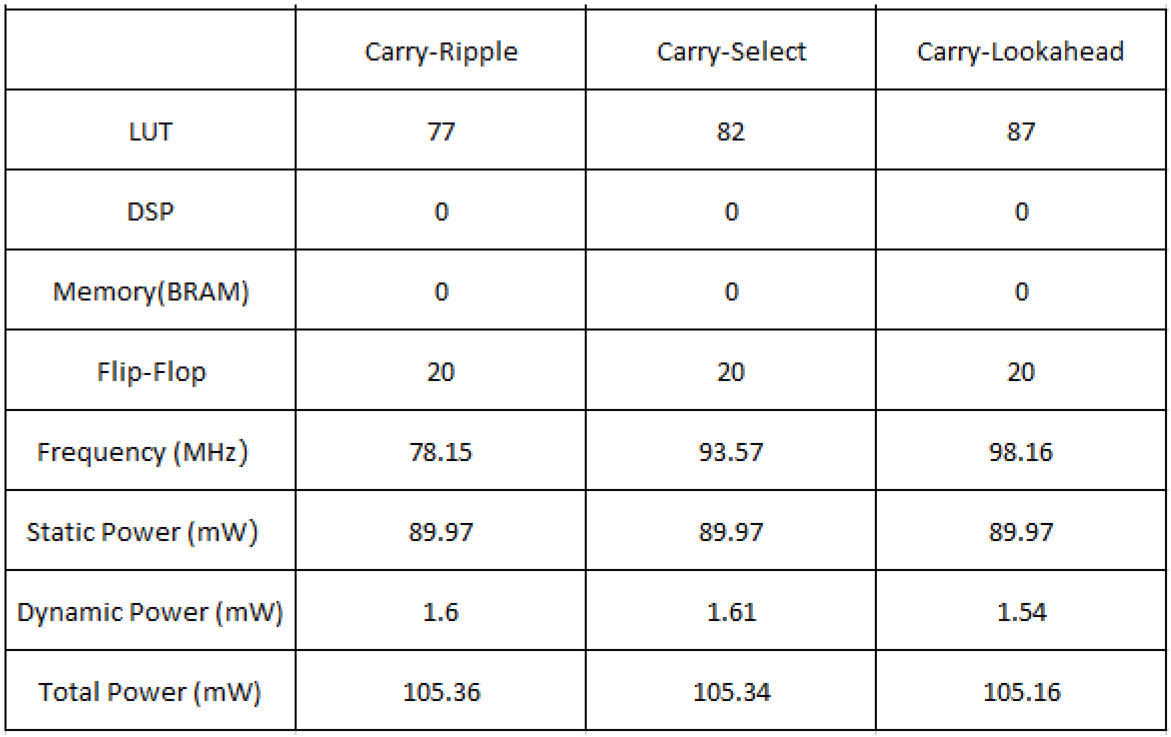


Figure 9: The chart for the Design Analysis

Post-Lab Questions

1. There are two primary ways to adjust the design to achieve different goals. The 4x4 hierarchy for a 16-bit CSA represents a balance of efficient and concise design. An 8x2 design (eight 2-bit CSAs) will be much faster than the 4x4 hierarchy, but it will consume more power and increase the area somewhat as the number of multiplexers doubles. If we design a 2x8 adder (two 8-bit CSAs), it will consume less power and take a little less area, but it will be much slower than the 4x4 hierarchy.



	Carry-Ripple	Carry-Select	Carry-Lookahead
LUT	77	82	87
DSP	0	0	0
Memory(BRAM)	0	0	0
Flip-Flop	20	20	20
Frequency (MHz)	78.15	93.57	98.16
Static Power (mW)	89.97	89.97	89.97
Dynamic Power (mW)	1.6	1.61	1.54
Total Power (mW)	105.36	105.34	105.16

Table 2: Design Resources and Statistics

2. From the table above, we can observe that the Carry-Lookahead uses the most space in the LUT, but the frequency is also the highest. As we expected, the Carry-Lookahead and Carry-Select adders are significantly faster than the CRA by using more LUT than the CRA. These three adders are all implemented in the FPGA using combinational logic, so they do not use memory or registers with the same number of flip-flops used for the control unit. As far as power consumption, all three adders consume an identical amount of static power, with minimal variability in dynamic power and very similar results for total power, which matches our hypothesis. Since

there will be thermal power for I/O thermal power dissipation, the total power is not the sum of Dynamic Power and Static Power.

Conclusion

Bugs Encountered

When we design the CSA, we don’t know where to set the MUX, and how it works. Later, we figured out that the CSA needs to compute two results simultaneously based on different carry-in’s. Then we knew to set the true carry-in from the last adder as a select bit to choose the correct result.

Lab Manual

This lab was the most straight-forward of the labs thus far. We don’t have any recommended changes.

Summary

In this lab, we implemented a 16-bit adder using three different methods on the DE10-Lite FPGA board: the Carry Ripple Adder, the Carry Select Adder, and the Carry Lookahead Adder. We also learned how to use the Quartus software to identify the performance and resource consumption data of our implementation, as well as how to compare their total power, dynamic power, register usage, and time complexity.