ECE 385

Fall 2023

Experiment #3

Introduction to System Verilog, FPGA, EDA, and 16-bit Adders

Yunxuan Yang, Yuqi Wen
2023/9/20
TA: Tianhao Yu

1) Introduction

This lab delves into the design and performance analysis of three fundamental binary adder architectures: Ripple Carry Adder, Carry Lookahead Adder, and Carry Select Adder. Each adder has unique characteristics concerning speed, complexity, and resource utilization. We build the three adders based on the basic full adder.

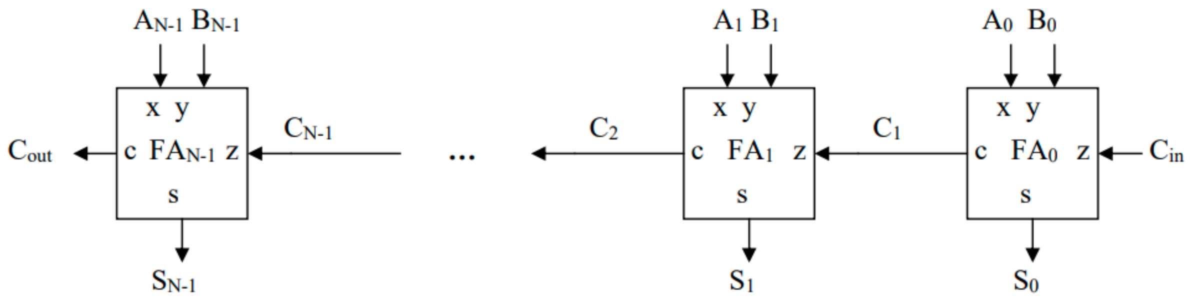
2) Adders

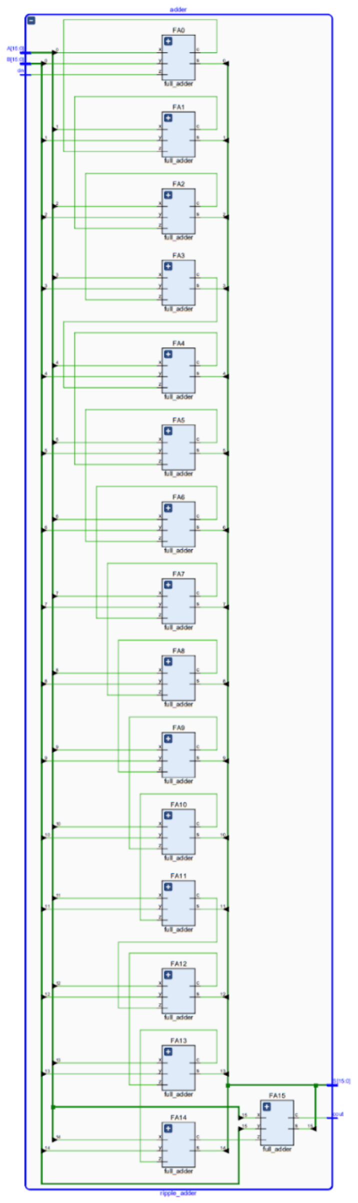
a. Ripple Carry Adder

i. Written description of the architecture of the adder

The Ripple Carry Adder operates by sequentially summing every bit, beginning from the least significant bit, and rippling the carry over to the next bit. A 16-bit Ripple Carry Adder consists of 16 cascaded 1-bit full adders. Each of these 1-bit full adders can add two input bits and a carry bit from the previous stage to generate a sum bit and a carry-out bit.

ii. Block diagram





b. Carry Lookahead Adder

i. Written description of the architecture of the adder

A Carry Lookahead Adder is an advanced type of adder that accelerates the binary addition process by overcoming the latency introduced by the ripple-carry mechanism. Rather than waiting for the carry to propagate from one stage to the next, the CLA "predicts" the carry base on the “G” and “P” logics from the carry-lookahead unit and computes it in parallel using dedicated logic.

ii. Describe how the P and G logic are used

The Generate (G) and Propagate (P) logic components form the foundational elements of a Carry Lookahead Adder (CLA). Their primary role is to determine the generation and propagation of the carry bit, enabling the CLA to compute carry values in advance and thus eliminate the ripple effect seen in traditional ripple carry adders.

The Generate logic determines whether the given bit position will produce or "generate" a carry. This carry is the direct result of adding the bits A_i and B_i at that position. Specifically, for each bit position i: $G_i = A_i \times B_i$

The Propagate logic determines if a carry from the previous bit position will be passed or "propagated" to the next bit position. Specifically, for each bit position i: $P_i = A_i \oplus B_i$

With P and G defined, the Boolean expression for the carry-out C_{i+1} is $C_{i+1} = G_i + (P_i \cdot C_i)$.

$$
\begin{aligned}
&C_0 = C_{in} \\
&C_1 = C_{in} \cdot P_0 + G_0 \\
&C_2 = C_{in} \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1 \\
&C_3 = C_{in} \cdot P_0 \cdot P_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + G_1 \cdot P_2 + G_2 \\
&\ldots
\end{aligned}
$$

iii. Describe how you created the hierarchical 4x4 adder

In the foundational model, we leverage the illustrated design to craft a standard 4-bit CLA. Yet, to align with our hierarchical structure, it's imperative to formulate new group propagate (P_G) and the group generate (G_G) expressions tailored for accurate computation across four digits.

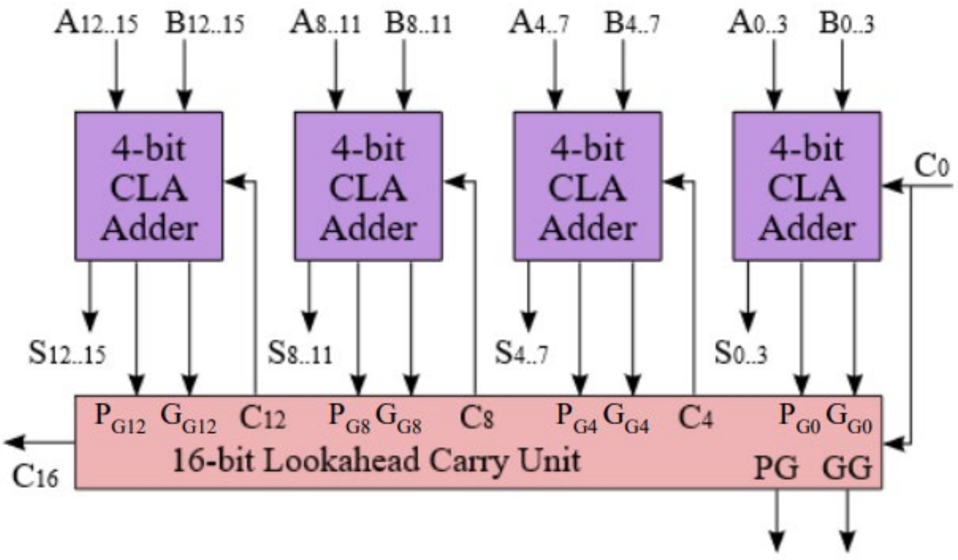
$P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$

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

Then, we generate Cins from the 4-bit CLA using P_G and G_G with the formula shown below:

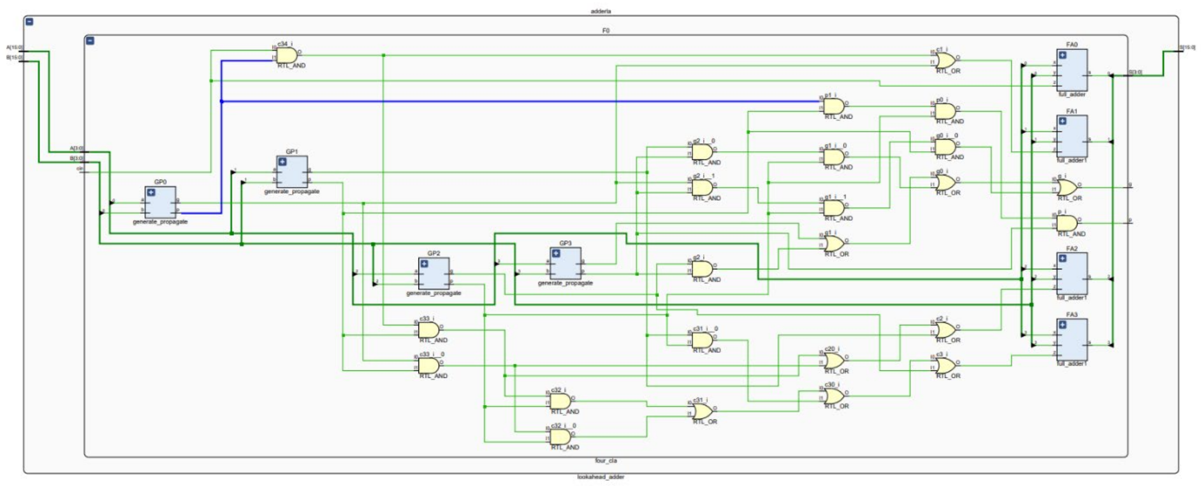
$$
\begin{aligned}
&C_4 = G_{G0} + C_0 \cdot P_{G0} \\
&C_8 = G_{G4} + G_{G0} \cdot P_{G4} + C_0 \cdot P_{G0} \cdot P_{G4} \\
&C_{12} = G_{G8} + G_{G4} \cdot P_{G8} + G_{G0} \cdot P_{G8} \cdot P_{G4} + C_0 \cdot P_{G8} \cdot P_{G4} \cdot P_{G0}
\end{aligned}
$$

We complete our design as below:

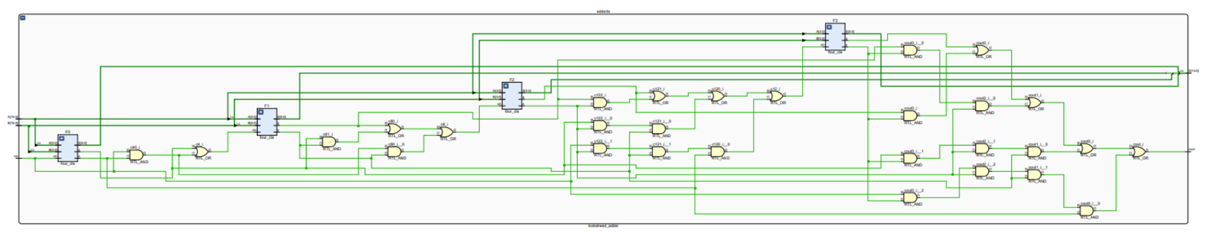


iv. Block diagram

1. Block diagram inside a single CLA (4-bits)



2. Block diagram of how each CLA was chained together



c. Carry Select Adder

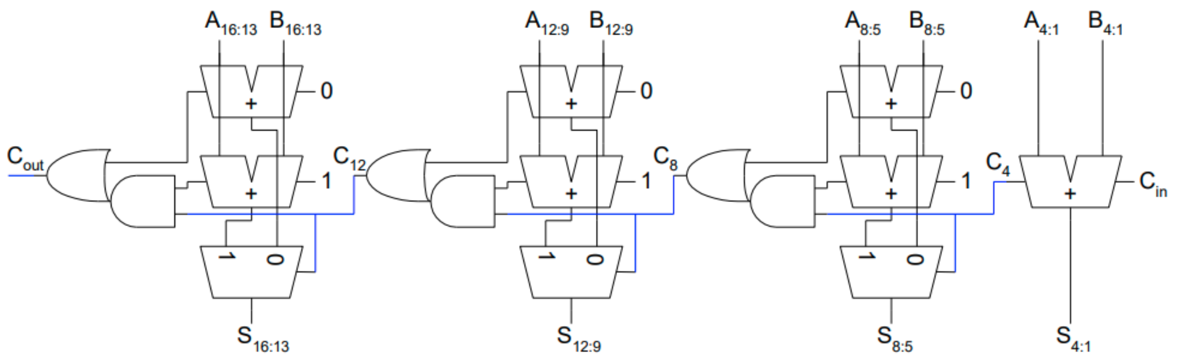
i. Written description of the architecture of the adder

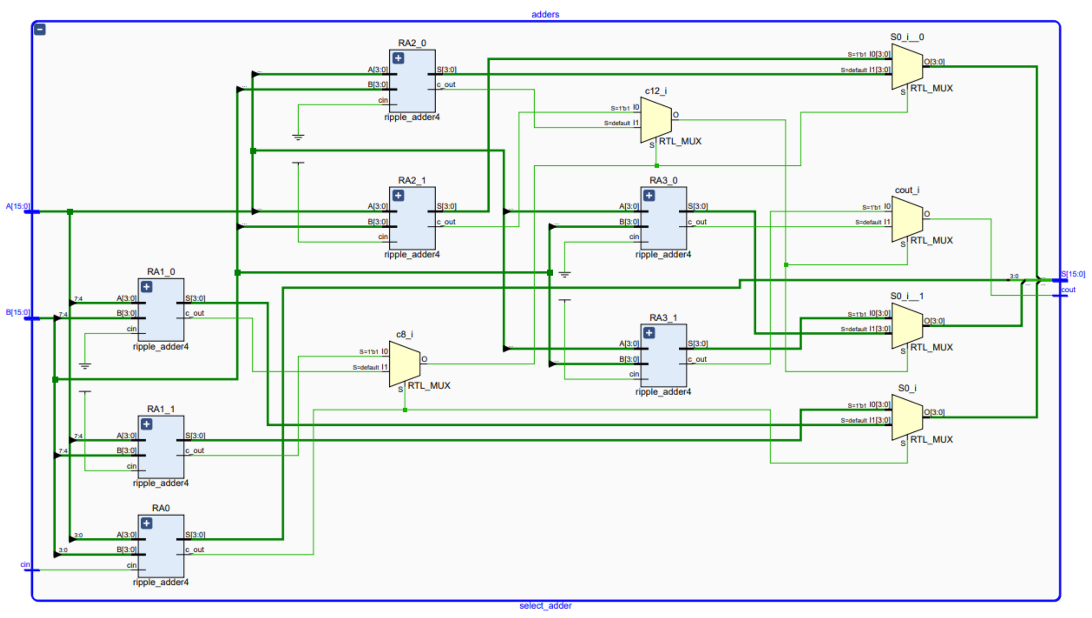
The 16-bit Carry Select Adder (CSA) is an optimized binary adder tailored to enhance computation speeds by tackling the ripple-carry delay common in standard adders. Rather than predictively computing the sum and carry based on potential carry-ins, the CSA simultaneously computes both possible outcomes (carry-in of 0 and carry-in of 1) for every bit, then swiftly chooses the correct results using a multiplexer once the actual carry-in is determined.

ii. Describe at a high level how the CSA speculatively computes multiple sums in parallel and rapidly chooses the correct one later

The CSA achieves a speed advantage by executing parallel computations. In the context of a 4x4 CSA, every 4-bit addition is calculated concurrently for both potential carry-in scenarios (C=0 and C=1). Once these calculations are complete, the actual carry from the previous 4-bit segment determines the correct output through 5 parallel multiplexers. Given that the delay associated with a multiplexer is less than that of a 4-bit adder, the system operates more rapidly overall.

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





d. Written description of all .SV modules

1. Module: Ripple_adder

Inputs: [15:0] A, [15;0] B, cin

Outputs: [15:0] S, cout

Description: This module contains two modules: full adder and ripple adder. Declare full adder first and then use it in ripple adder.

Purpose: This module is used to perform 16-bit ripple add calculation.

2. Module: lookahead_adder:

Inputs: [15:0] A, [15;0] B, cin

Outputs: [15:0] S, cout

Description: This is a carry-lookahead adder with 4 small units of unit adders. It accepts A, B, carry in bit and output its sum and carry out bit co. Different from carry ripple adder, we need P and G of each unit adder to compute carry in and carry out bits between each unit adder.

Purpose: This module is used to perform 16-bit lookahead add calculation.

3. Module: select_adder

Inputs: [15:0] A, [15;0] B, cin

Outputs: [15:0] S, cout

Description: This is a carry-lookahead adder, accepting A, B, carry in bit and output its sum and carry out bit co. A 2-1 MUX is used to select the sum with carry in bit 0 or 1.

Purpose: This module is used to perform 16-bit select add calculation.

4. Module: Reg_17

Inputs: [16:0] Din, Clk, Load, Reset

Outputs: [16:0] Data_Out

Description: This is a 17-bit register with synchronous reset and load capabilities.

Purpose: Store a 17-bit value to a register.

5. Module: Hexdriver

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: This module converts 4bit binary numbers from registers into hexadecimal to display digit on LED.

Purpose: Set the 7-segment LED display on board.

6. Module: Control

Inputs: Clk, Reset, Run

Outputs: Run_O

Description: This is a state machine with asynchronous reset, Only in State B (When input Run is High), the output Run_O will be high.

Purpose: This module is the state machine that gives the execution signal when we press a button and prevents multiple execution if button is not released.

7. Module: mux2 _1_17

Inputs: S, [15:0] A_In, [16:0] B_In

Outputs: [16:0] Q_Out

Description: This is a 17-bit 2-1 MUX which select 4-bit value A or B by input bit 0 or 1.

Purpose: This module accepts carry in bit from last group and select the correct sum with this carry in bit.

8. Module: adder_toplevel

Inputs: Clk, Reset_Clear, Run_Accumulate, [15:0] SW

Outputs: sign_LED, [7:0] hex_segA, [3:0] hex_gridA, [7:0] hex_segB, [3:0] hex_gridB

Description: This is a top-level module containing the specific adder functions to be called.

Purpose: The module is used as a top-level module to connect each module in this project, receive input signal, compute it with adder, change it to hex representation, and display it on the board.

e. area, complexity, and performance tradeoffs between the adders

Ripple Carry Adder:

Area: RCA uses the least amount of hardware resources among the three. It's composed of simple full adders chained together.

Complexity: Low. It's a linear structure with carries rippling through each stage.

Performance: Slowest among the three due to the ripple effect. The carry from one stage must propagate through all subsequent stages, which leads to a delay that grows linearly with the number of bits.

Carry Lookahead Adder:

Area: Uses more hardware than RCA because of the additional logic required for the carry lookahead function. It computes carries using the generate (G) and propagate (P) logic.

Complexity: Moderate. It introduces the concepts of propagate and generate to compute the carry for each bit position in parallel.

Performance: Much faster than RCA as it can determine the carry values more quickly without waiting for them to ripple through. The delay grows logarithmically with the number of bits.

Carry Select Adder:

Area: Uses the most hardware among the three since it essentially computes results for both possible carry input values (0 and 1) in parallel.

Complexity: High. It includes multiple ripple carry adders and additional multiplexer logic to select between the results.

Performance: Faster than RCA but typically slower than CLA. It avoids the ripple delay by speculatively computing sums for both possible carry values and then selecting the correct one once the carry is known.

Tradeoffs:

If area (and potentially power) is the primary concern and performance isn't critical, then RCA might be the best choice.

If maximum performance is essential and area isn't a major concern, then CLA is likely the best choice.

CSA represents a middle ground, providing a performance boost over RCA with a lesser area penalty than CLA.

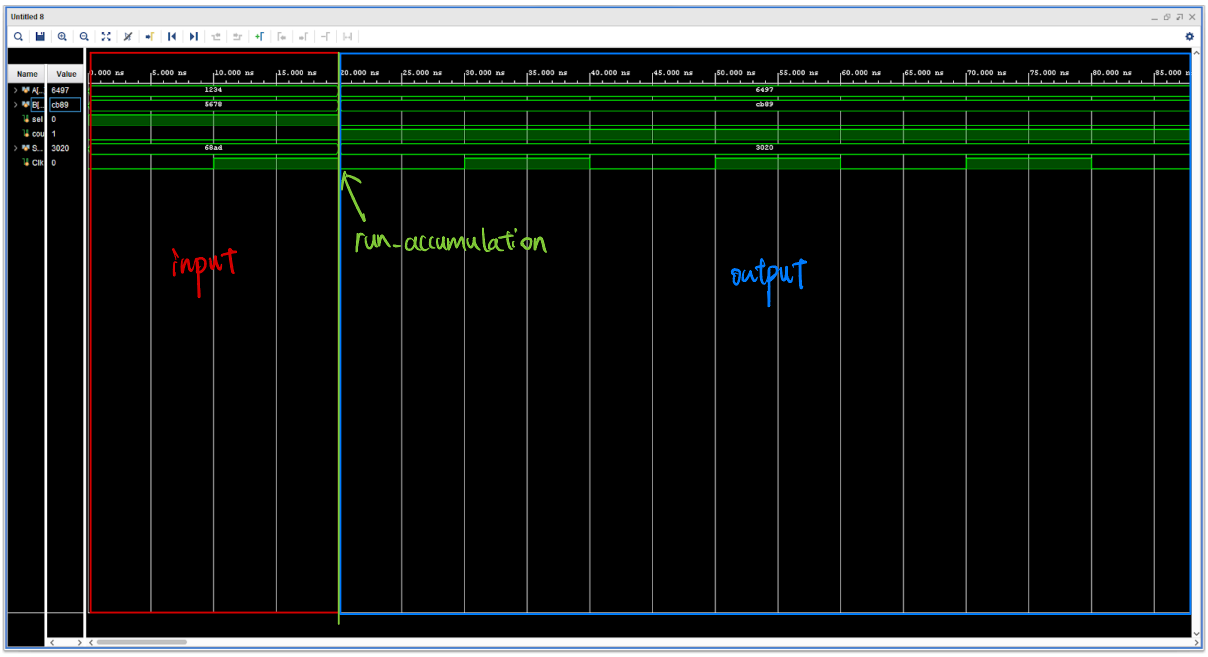
f. Document the performance of each adder

	Carry-Ripple	Carry-Select	Carry-Lookahead
LUT	93	99	117
Frequency (MHz)	148.7652484	191.7545542	232.6663564
Total	1.022	1	0.989

g. Table for all the remaining performance metrics

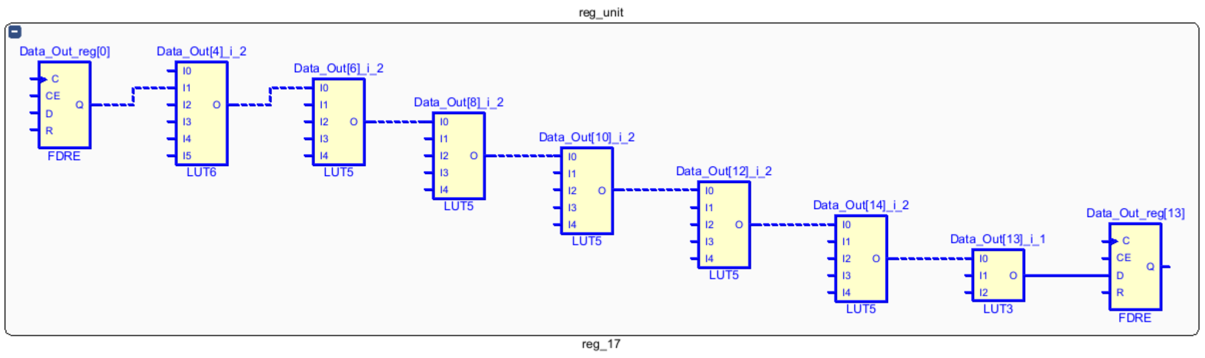
	Carry-Ripple	Carry-Select	Carry-Lookahead
LUT	93	99	117
DSP	0	0	0
Memory	0	0	0
Flip-Flop	53	53	53
Frequency	148.7652484	191.7545542	232.6663564
static Power	72	72	72
Dynamic Power	23	21	20
Total	95	93	92

h. Annotated simulation trace



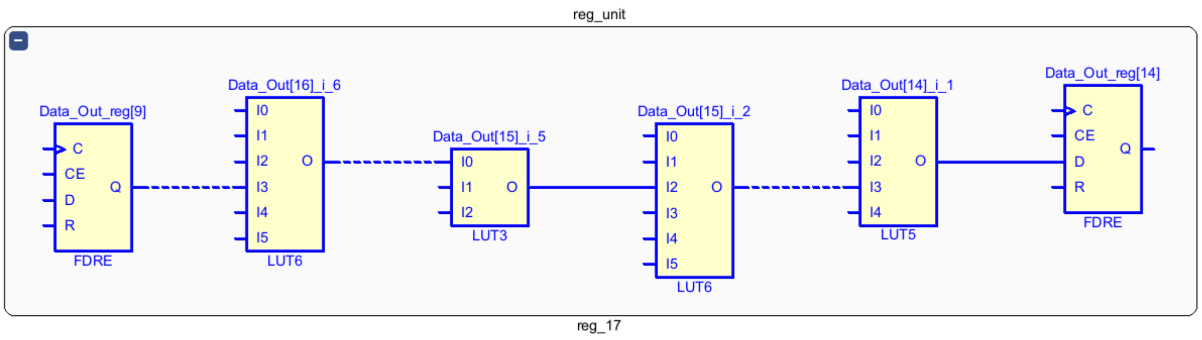
i. critical path analysis

Ripple Carry Adder:

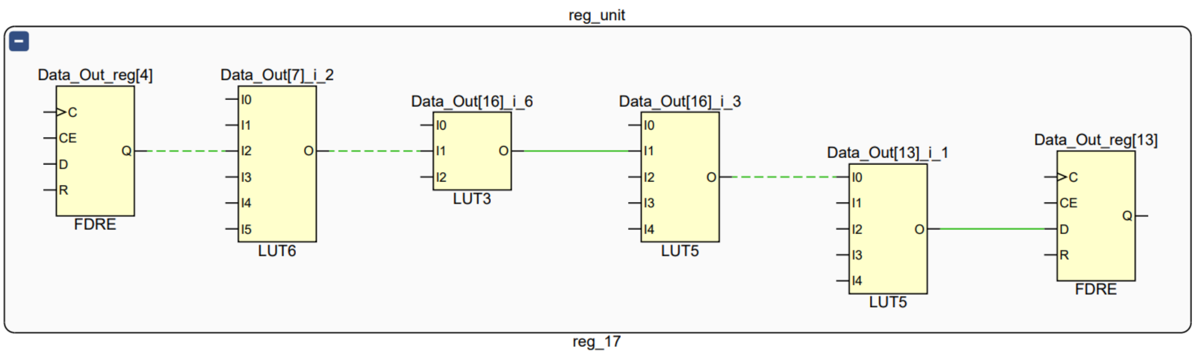


For the Carry Ripple adder, the theoretical critical path should have 16 adders, but in reality, we only have 7 LUTs. The 7 LUTs in the critical path suggest that several of the adder stages might be combined into single LUTs or that only part of the full 16-bit path is the actual critical path. Perhaps only a portion of the ripple-carry adder (7 bits out of 16) is dominating the delay.

Carry Lookahead Adder:



Carry Select Adder:



For Carry Lookahead adder and Carry Select adder, the theoretical critical path is consistent with the actual critical path. They have the same number of adders and LUTs.

3) Answers to the post-lab questions

1. Document design analysis

Fmax =1Tclk - WNS

Based on the findings from the Fitter Resource Usage and Power Analyzer, the Carry Lookahead Adder registers the highest frequency among all adders, clocking in at 232.7 MHz. This elevated frequency indicates a quicker computational capability in the adder. As previously discussed, we expected the Carry Lookahead Adder to be the speediest due to its Propagate and Generate signals. Additionally, we had noted that it occupies more space owing to its increased gate count. This aligns with our observation, as evidenced by its higher LUT, pointing to a greater number of logic elements. Furthermore, the Carry Lookahead Adder uses the most power, likely because it undertakes additional computations, like the Propagate and Generate, leading it to draw more power relative to the other adders.

2. 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?)
The 4x4 configuration for a Carry-Select Adder isn't optimal, given that it accumulates the delays from 4 full adders and 3 multiplexers (MUXes). The absence of a carry-in for the first adder means only 3 MUX delays are considered.

The key is to strike a balance between the MUX delay and the computation time for the addition, ensuring that the carry-out is generated as swiftly as possible. By adjusting the number of adders per block and the MUXes managing the carry bit, we can optimize this. An intriguing discovery was that by increasing the cumulative MUX delays, the full adder delays decreased.

For instance, employing 8 two-bit adders results in 7 MUX delays. Modifying the configuration further, if we use a six-bit adder at the end, the setup will consist of 1 six-bit adder, 5 two-bit adders, and 5 MUXes. Through iterative experimentation, one can determine the best division for the 16-bit adder. For a comprehensive assessment, understanding how to compute the delay based on the total bit addition and discerning the delays for varying full adder lengths would be vital.

3. Table

	Carry-Ripple	Carry-Select	Carry-Lookahead
LUT	93	99	117
DSP	0	0	0
Memory	0	0	0
Flip-Flop	53	53	53
Frequency	148.7652484	191.7545542	232.6663564
static Power	72	72	72
Dynamic Power	23	21	20
Total	95	93	92

4. Critical Path
See 2i

4) Conclusion

a. Describe any bugs and countermeasures taken during this lab

Variable name capital letter inconsistent, causing function not working. We ran the simulation to debug and eventually realized the variable name was wrong.

b. ambiguous, incorrect, or unnecessarily

Generally, this lab’s instructions are very good which help us to understand the use of System Verilog and Vivado quickly.

c. Summary

In this lab, we delved deeper into the adders introduced in ECE 120. We closely examined the bitwise addition process of each adder, gaining insights into their individual workings. This allowed us to discern when and why to opt for a specific adder over another. In essence, the lab's objective, which was to construct the three distinct adders - the Carry Ripple Adder, Carry Lookahead Adder, and Carry Select Adder - was accomplished effectively.