Lab 4: A Comparison of Arithmetic Adders using Xilinx Vivado

ENGG3050: Reconfigurable Computing Systems

Instructor:
Dr. Shawki Areibi

Group 42: Tuesday-3:30 Section

Bilal Ayyache: 0988616
Anthony Granic: 0994824

Lab Start Date: October 8, 2019
Lab End Date: October 22, 2019

Table of Contents

1.0- Project Description:

1.1- Problem Statement

The objective of this lab was to create multiple adders and compare their performance, functionality, speed and number of resources used by the Nexys A7 board.

1.2- Constraints and Assumptions

The constraints of this lab were to use VHDL, Vivado, and Nexys A7 100T to simulate the designs. It was assumed that these were functioning correctly. Post-Map Static Timing, and Post-Place & Route Static Timing are tools available in ISE Design Suite, but not in Vivado.

1.3- System Overview & Justification of Design

There were three systems that were designed. The first is a generic ripple carry adder. This was done using its most common design. It is supposed to be the least efficient of the adders made. The second adder made was a carry select adder. This adder is supposed to be the second fastest, but also use more resources than the ripple carry adder. The last and fastest adder made was a carry look ahead adder. This adder also used the most resources.

All of the adders were made using a hierarchal design. This was done for efficiency and readability. The ripple carry adder requires that the previous bit is calculated before the next one can be calculated. The carry select adder computes N bits at a time and uses the carry output as a selector of the answer of the next N bits. The carry look ahead adder uses an equation to choose the output of each bit as it progresses. The logic becomes more complicated with more bits.

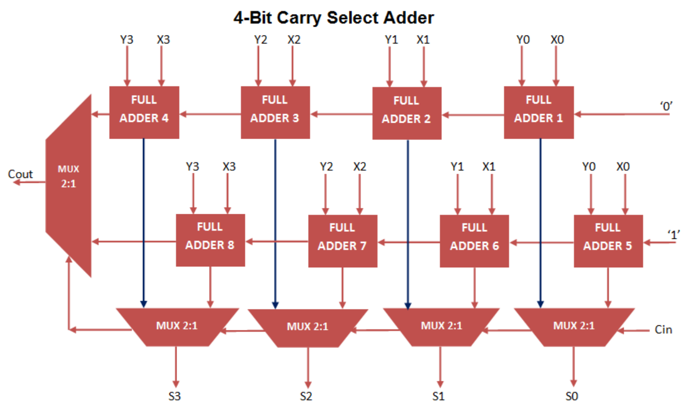


Figure 1: Carry Select Adder Diagram

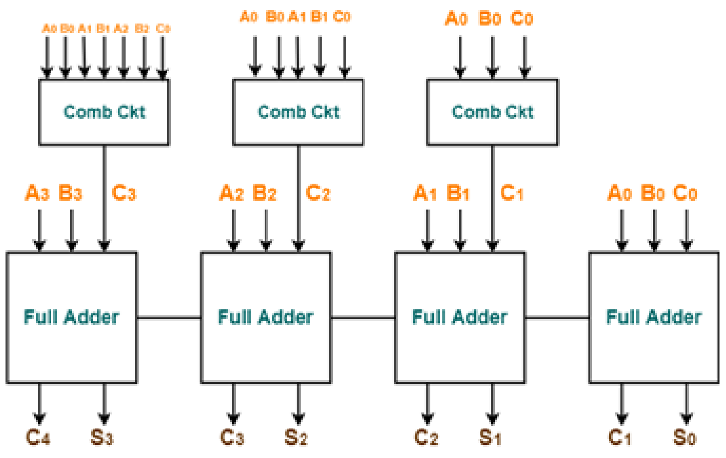


Figure 2: Carry Look Ahead Adder Diagram

2.0- Hardware description:

The schematics below show the descriptions of the different adders. The ripple carry adder uses full adders for each bit and has an extra input for the carry of the previous bit. The carry select adder uses full adders and an assumed carry in for each pair of bits. The answers from these are then chosen by the carry in that is calculated. The carry look-ahead adder uses a full adder for each pair of bits and uses a function to decide the carry in for each full adder.

2.1- Hardware Schematic:

2.1.1- Carry Select Adder:

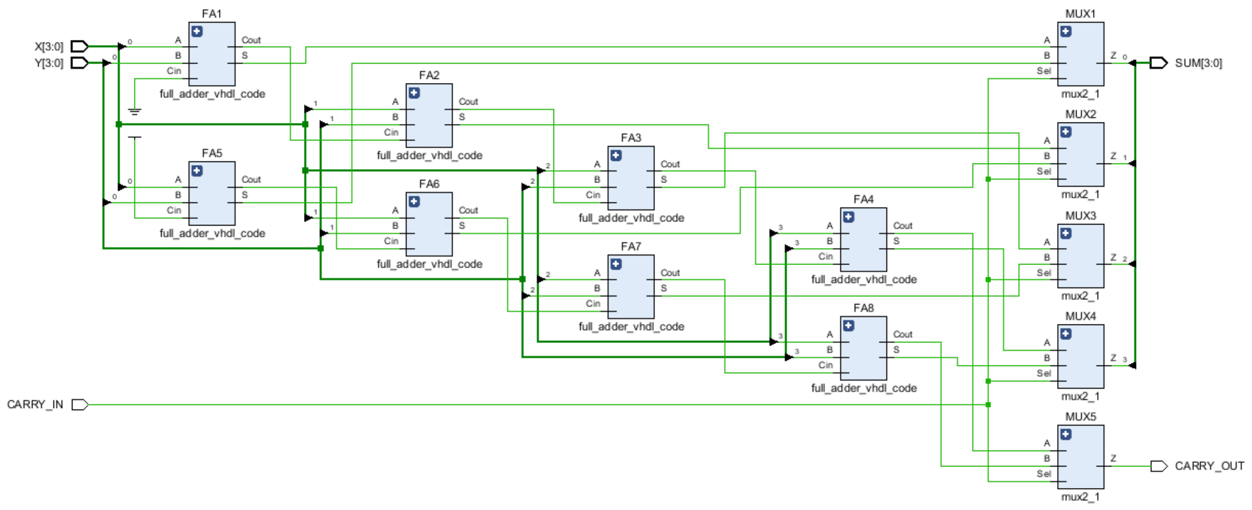


Figure 3 – Carry Select Adder Hardware Schematic

2.1.2- Carry Ripple Adder:

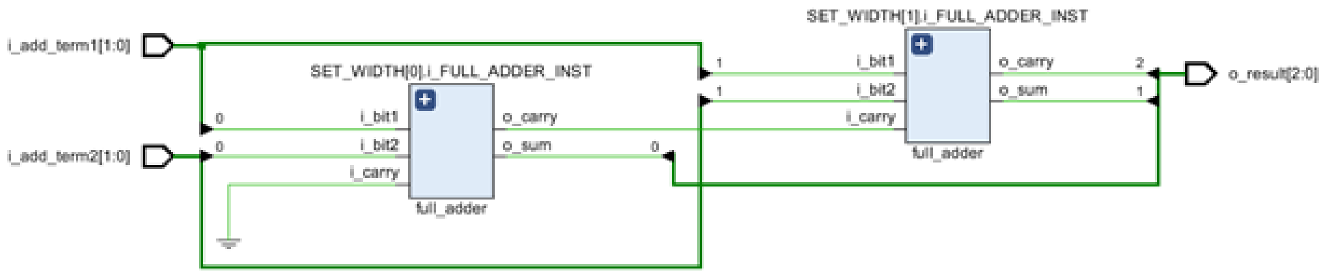


Figure 4 – Carry Ripple Hardware Schematic (1 Bit for Simplicity)

2.1.3- Carry Look-Ahead Adder:

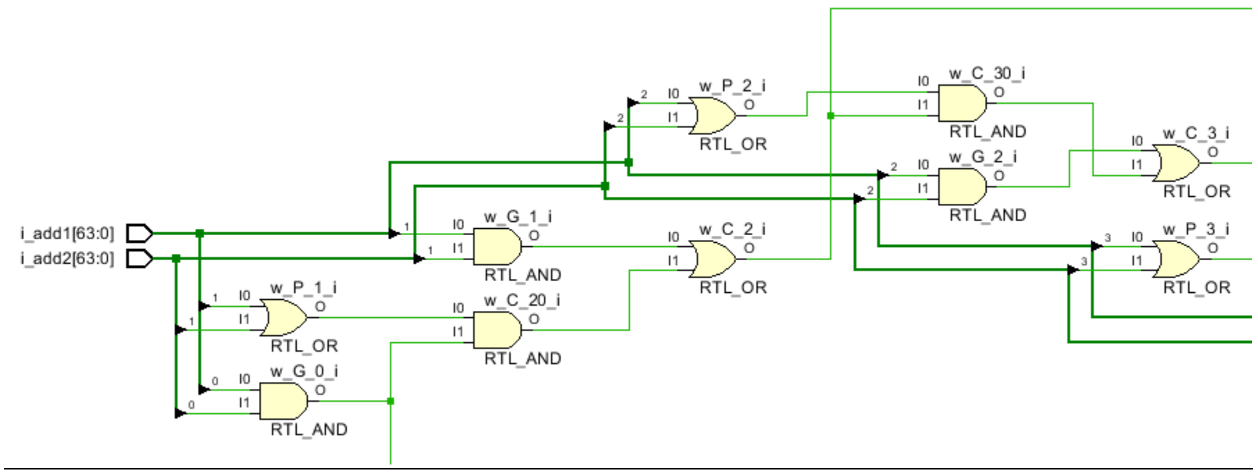


Figure 5 – Carry Look-Ahead Hardware Schematic (Part 1)

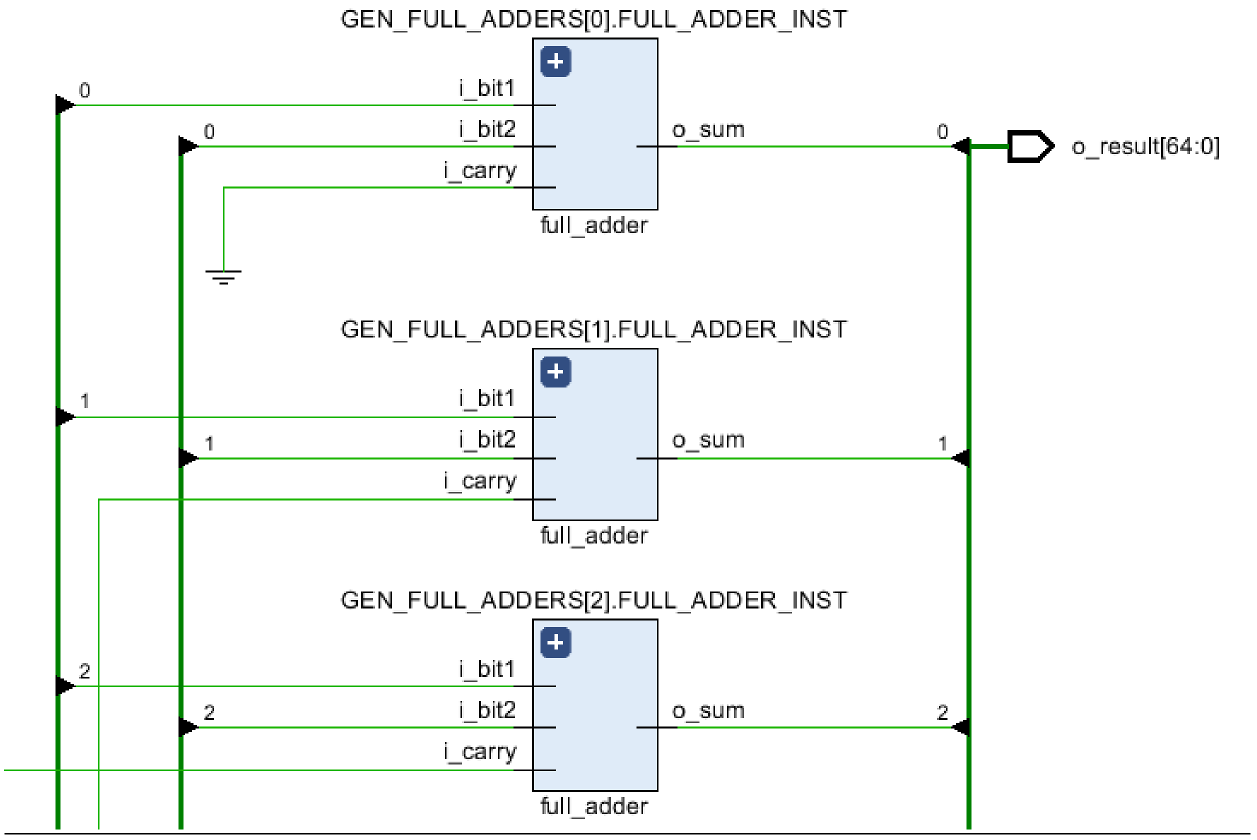


Figure 6 – Carry Look-Ahead Hardware Schematic (Part 2)

2.1.4- Full Adder:

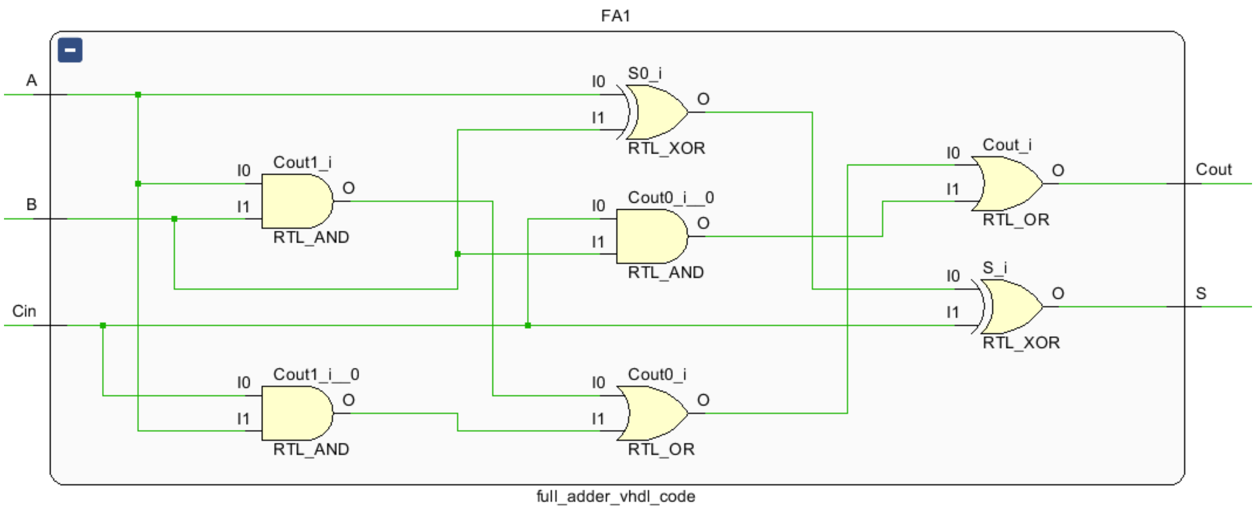
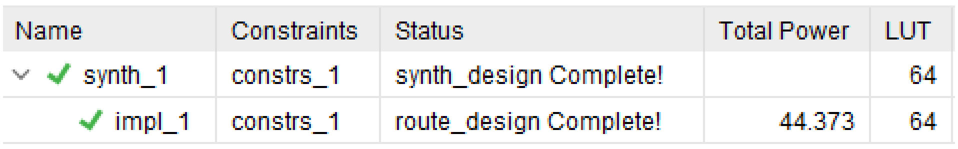


Figure 7 – Full Adder (1 Bit for Simplicity)

2.2- Resources used:

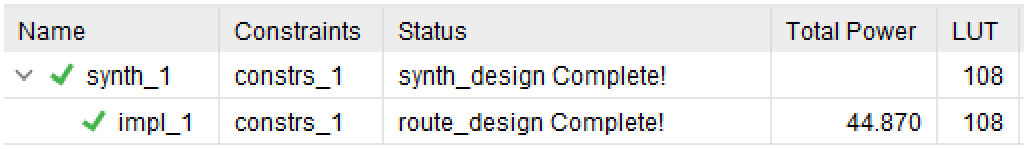
As expected, the ripple carry uses less resources than the other two designs and is also slower. This is shown in the diagram below.

2.2.1- LUT and Power Usage Analysis:



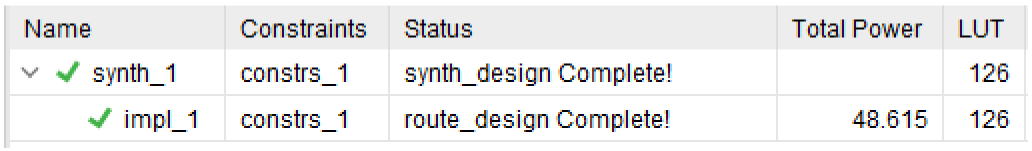
Name	Constraints	Status	Total Power	LUT
✓ synth_1	constrs_1	synth_design Complete!		64
✓ impl_1	constrs_1	route_design Complete!	44.373	64

Figure 8 – LUT usage (Ripple Carry Adder)



Name	Constraints	Status	Total Power	LUT
✓ synth_1	constrs_1	synth_design Complete!		108
✓ impl_1	constrs_1	route_design Complete!	44.870	108

Figure 9 – LUT usage (Carry Select Adder)



Name	Constraints	Status	Total Power	LUT
✓ synth_1	constrs_1	synth_design Complete!		126
✓ impl_1	constrs_1	route_design Complete!	48.615	126

Figure 10 – Full Adder (Carry Look-ahead Adder)

2.2.2- Visual Representation Of Space Occupied:

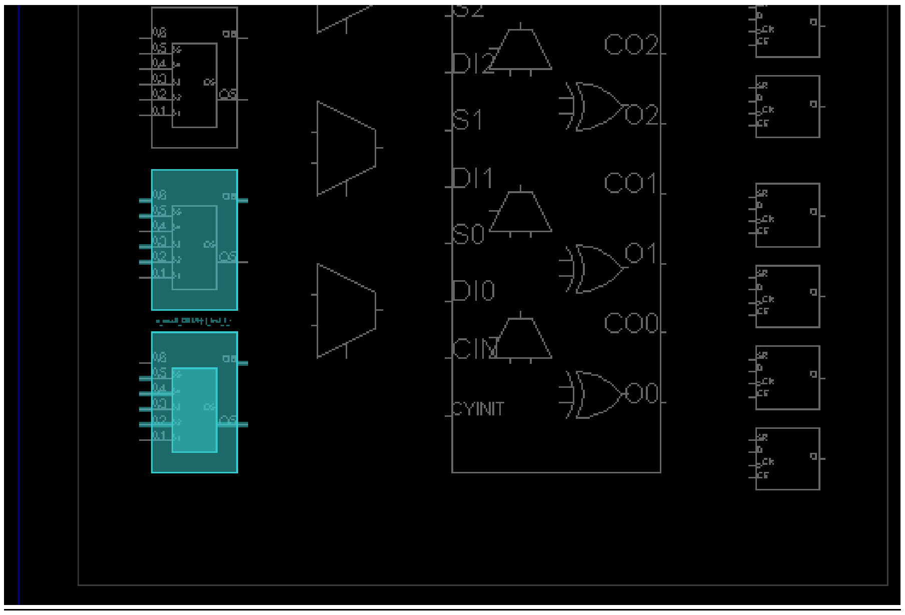


Figure 11– Space Occupied (Ripple Carry Adder)

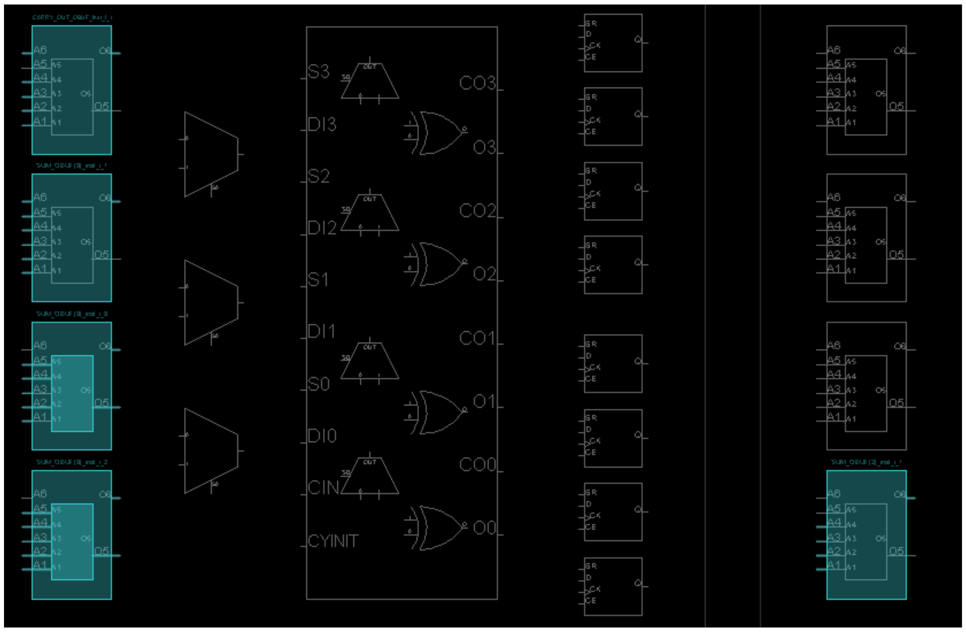


Figure 12 – Space Occupied (Carry Select Adder)

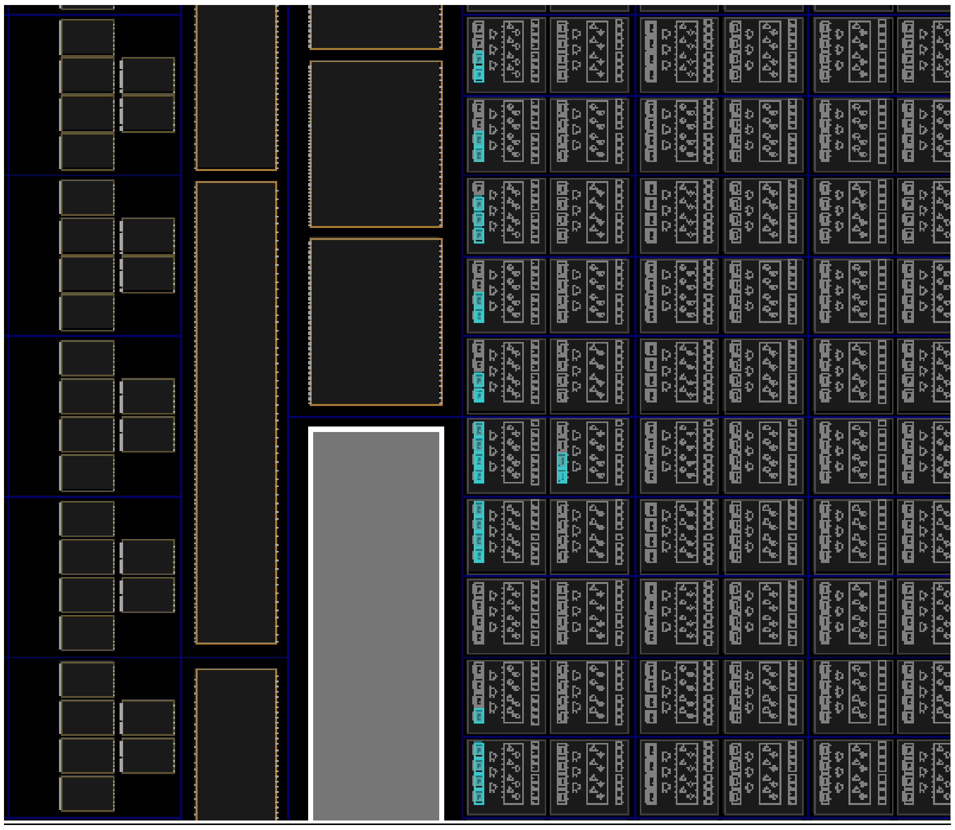
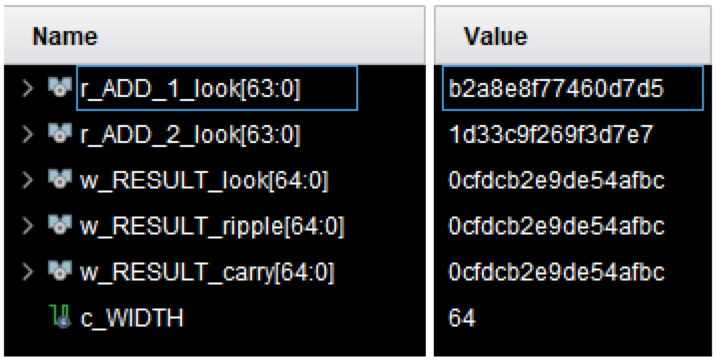


Figure 13 – Space Occupied (Carry Look-ahead Adder)

3.0- Implementation:

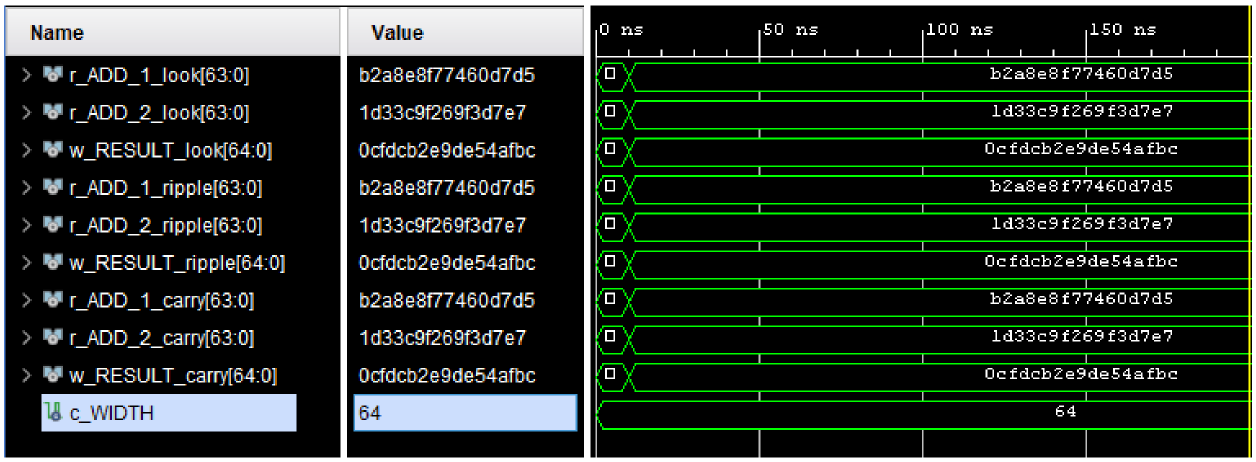
Before implementing the ALU designs, the designs where tested to ensure full functionality. A 64-bit implementation was designed for all ALUs. The figure above proves that the designs are fully functional. The same procedure was made when working with the 32-bit adders.

3.1- Post Synthesis Results:



Name	Value
r_ADD_1_look[63:0]	b2a8e8f77460d7d5
r_ADD_2_look[63:0]	1d33c9f269f3d7e7
w_RESULT_look[64:0]	0cfdcb2e9de54afbc
w_RESULT_ripple[64:0]	0cfdcb2e9de54afbc
w_RESULT_carry[64:0]	0cfdcb2e9de54afbc
c_WIDTH	64

Figure 14 – Functionality Testing

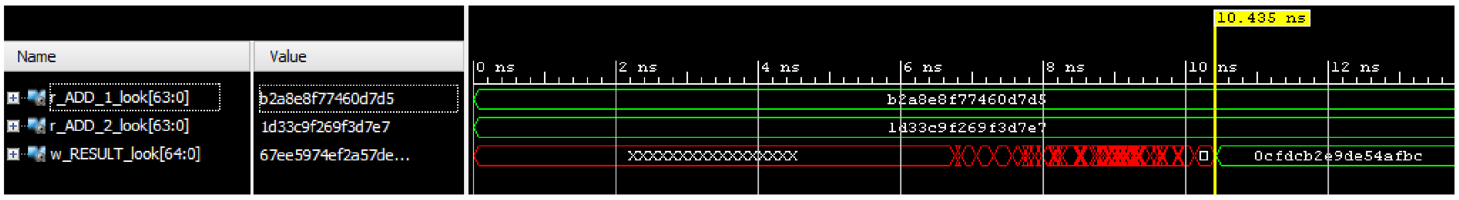


Name	Value
r_ADD_1_look[63:0]	b2a8e8f77460d7d5
r_ADD_2_look[63:0]	1d33c9f269f3d7e7
w_RESULT_look[64:0]	0cfdcb2e9de54afbc
r_ADD_1_ripple[63:0]	b2a8e8f77460d7d5
r_ADD_2_ripple[63:0]	1d33c9f269f3d7e7
w_RESULT_ripple[64:0]	0cfdcb2e9de54afbc
r_ADD_1_carry[63:0]	b2a8e8f77460d7d5
r_ADD_2_carry[63:0]	1d33c9f269f3d7e7
w_RESULT_carry[64:0]	0cfdcb2e9de54afbc
c_WIDTH	64

Figure 15 –Pre-Implementation Simulation

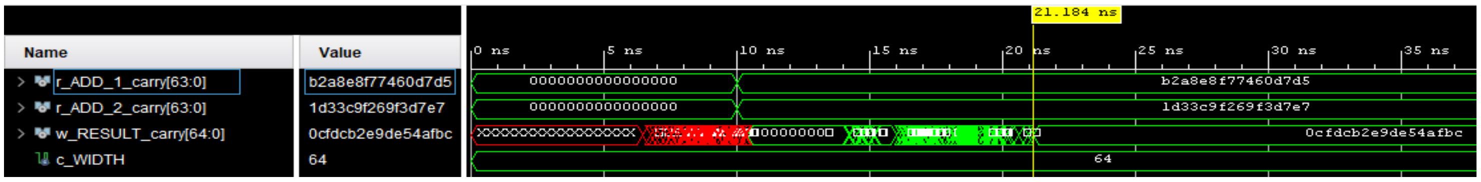
3.2- Post Implementation Timing Analysis

3.2.1- Simulation Results:



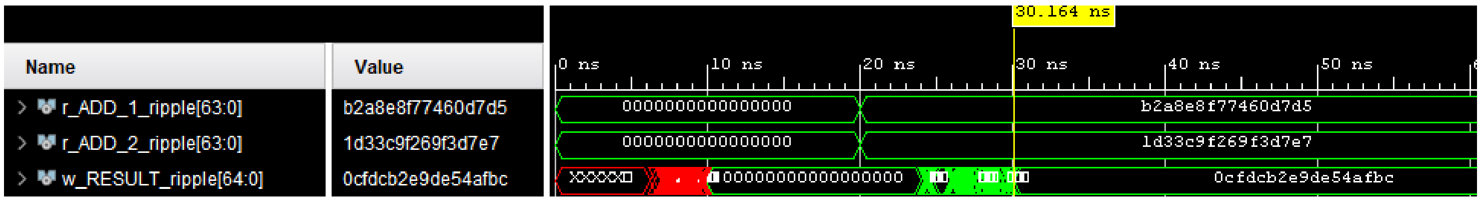
Name	Value
r_ADD_1_look[63:0]	b2a8e8f77460d7d5
r_ADD_2_look[63:0]	1d33c9f269f3d7e7
w_RESULT_look[64:0]	67ee5974ef2a57de...

Figure 16 – Carry Look-Ahead Adder Simulation Result (10.435ns)



Name	Value
r_ADD_1_carry[63:0]	b2a8e8f77460d7d5
r_ADD_2_carry[63:0]	1d33c9f269f3d7e7
w_RESULT_carry[64:0]	0cfdcb2e9de54afbc
c_WIDTH	64

Figure 17 – Carry Select Adder Simulation Result (21.184ns)



Name	Value
r_ADD_1_ripple[63:0]	b2a8e8f77460d7d5
r_ADD_2_ripple[63:0]	1d33c9f269f3d7e7
w_RESULT_ripple[64:0]	0cfdcb2e9de54afbc

Figure 18 – Carry Ripple Simulation Result (30.164 ns)

3.3- Result Analysis:

In the implementations of these designs the simulations below show the speed of the designs after implementation on the Nexys board. These simulations show the input, output, and speed of the different adders. The ripple carry adder took 30.164 ns (figure 18) to complete the addition algorithm, the carry select adder took 21.184 ns (figure 17) and the carry look ahead adder took 10.435 ns (figure 16). These results prove that the Carry Look-ahead Adder achieves the fastest computation time compared to the carry select adder and the carry ripple adder. When analyzing the 32 Bit adders, the carry look-ahead adder achieved a time of 16.69ns (Appendix 9) to complete the algorithm. The carry select's calculations were completed after 19.12ns (Appendix 6) and the ripple carry adder was done after 20.097ns (Appendix 3). The 32-bit analysis can be found in the appendix section. The table below summarizes the lab experiment:

Design Implemented	Time for Calculation	Power Consumption	LUTS Used
32-Bit Carry Ripple	20.097ns (A3)	22.09 Watts (A2)	32 (A2)
64-Bit Carry Ripple	30.164ns (F18)	44.373 Watts (F8)	64 (F8)
32-Bit Carry Select	19.12ns (A6)	22.92 Watts (A5)	52 (A5)
64-Bit Carry Select	21.184ns (F17)	44.870 Watts (F9)	108 (F9)
32-Bit Look-Ahead	16.69ns (A9)	23.712 Watts (A8)	62 (A8)
64-Bit Look-Ahead	10.435ns (F16)	48.615 Watts (F10)	128 (F10)

Table 1 – Result Summary (F= figure, A= Appendix)

3.4- Error Analysis

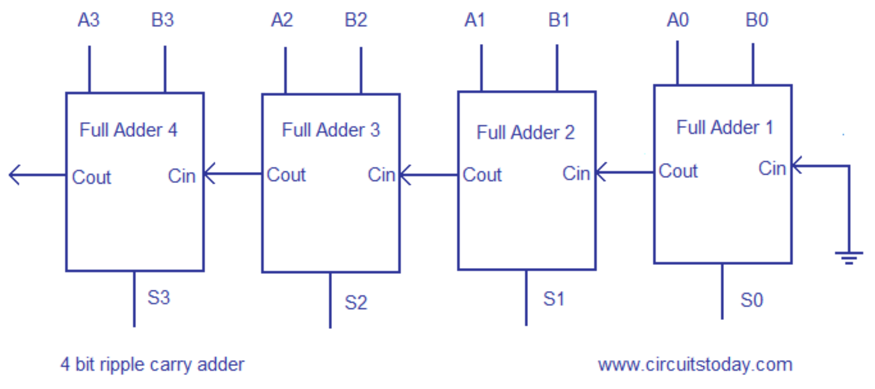
The errors that occurred were unexpected times of addition for the different adders. It is believed that this was caused due to using different computers to run Vivado simulations. The results were therefore inconclusive. A requirement in the lab is to produce a Post-Map Static Timing, Post-Place & Route Simulation Model, and Post-Place & Route Static timing. Those requirements where not met as Vivado does not offer these tools. Those tools were available in ISE design tool.

VHDL CODE REFERENCES:

Code used to make analysis was referenced from the following sources. The VHDL code was modified to enable 64bit and 32 bit addition.

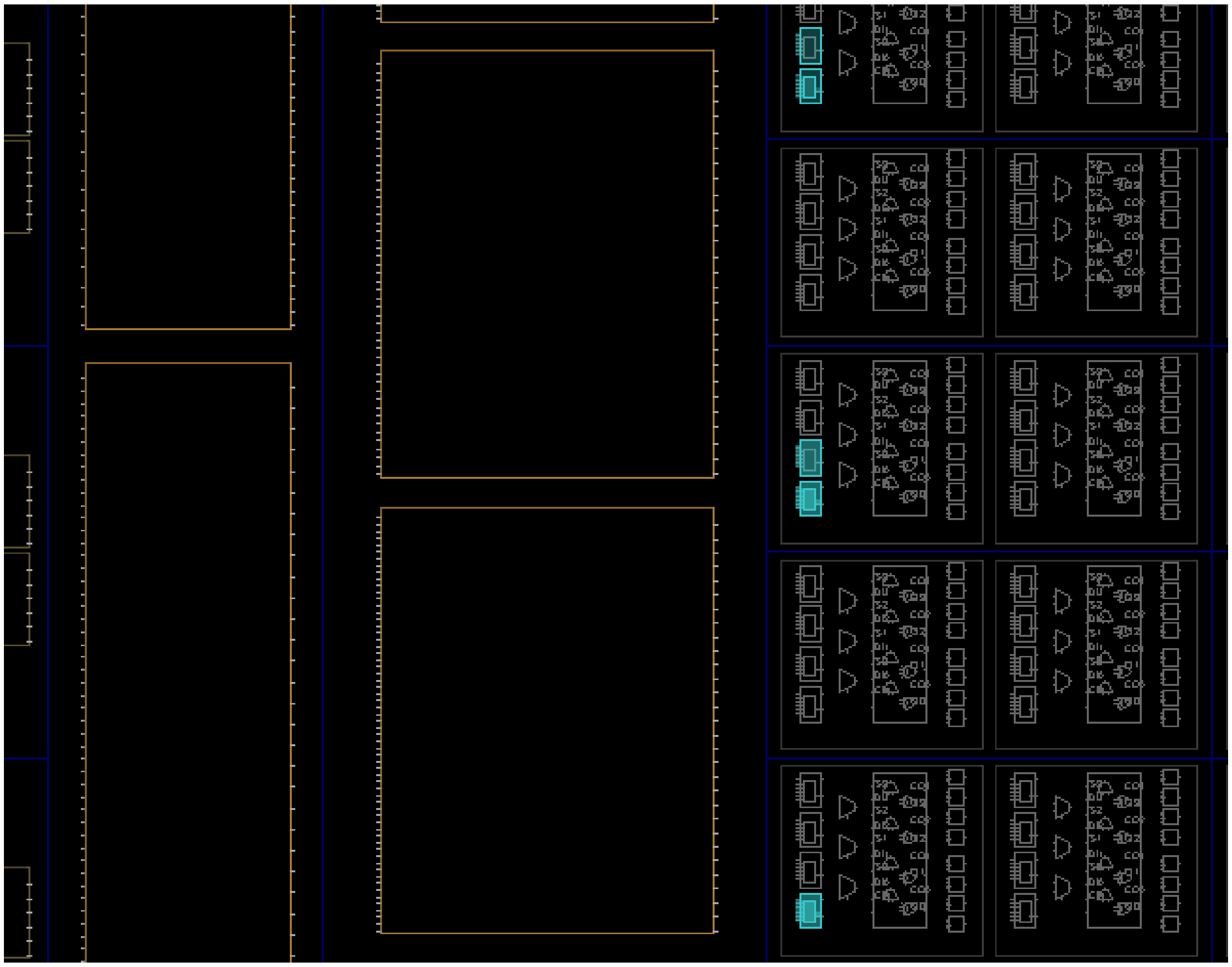
1- https://www.nandland.com/vhdl/modules/carry-lookahead-adder-vhdl.html
2- https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6514477
3- https://allaboutfpga.com/carry-look-ahead-adder-vhdl-code/
4- https://allaboutfpga.com/4-bit-ripple-carry-adder-vhdl-code/

APPENDIX:

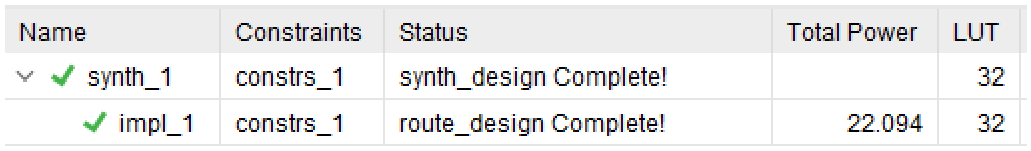


Appendix 0 – Ripple Carry Diagram

A.1- 32-Bit Ripple Carry Adder:

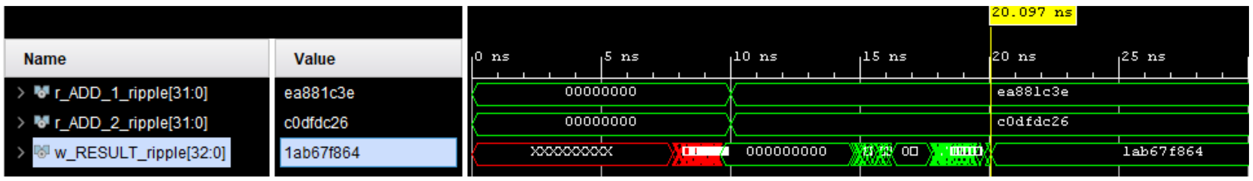


Appendix 1 – 32Bit Ripple Carry Hardware Usage



Name	Constraints	Status	Total Power	LUT
✓ synth_1	constrs_1	synth_design Complete!		32
✓ impl_1	constrs_1	route_design Complete!	22.094	32

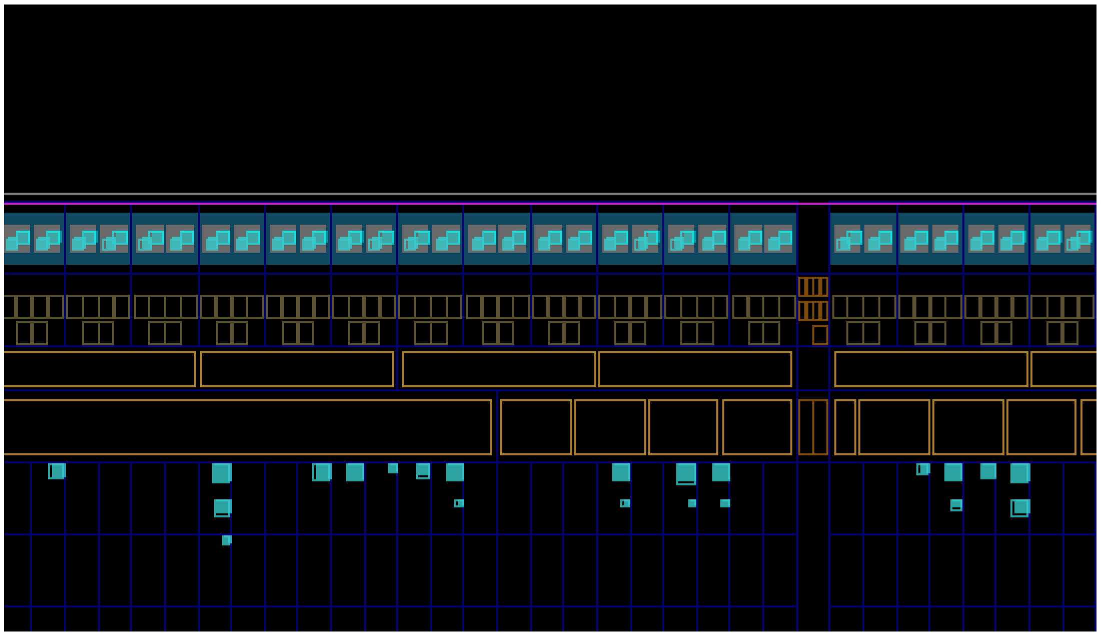
Appendix 2– 32Bit Ripple Carry LUT and Power Usage



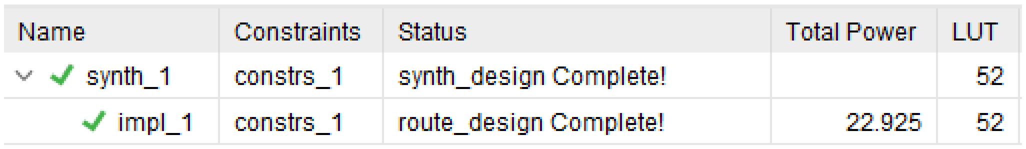
Name	Value
r_ADD_1_ripple[31:0]	ea881c3e
r_ADD_2_ripple[31:0]	c0dfdc26
w_RESULT_ripple[32:0]	1ab67f864

Appendix 3 – 32Bit Ripple Carry Execution Time (20.097)

A.2- 32-Bit Carry Select Adder:

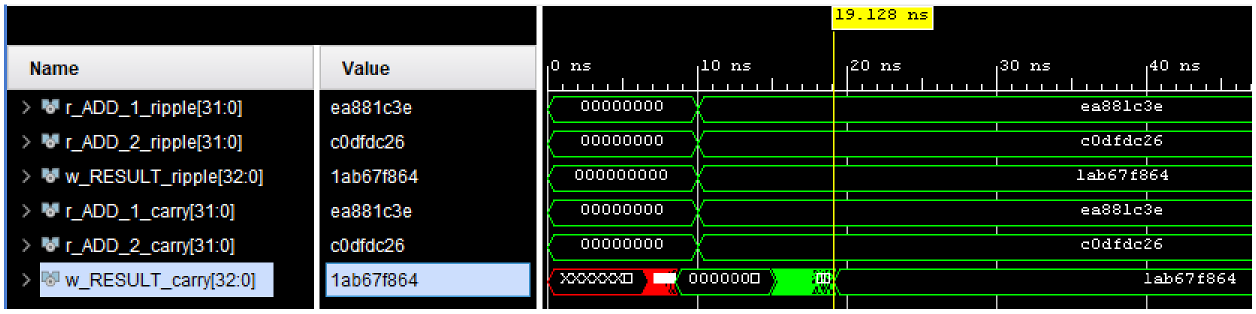


Appendix 4 – 32Bit Carry Select Hardware Usage



Name	Constraints	Status	Total Power	LUT
✓ synth_1	constrs_1	synth_design Complete!		52
✓ impl_1	constrs_1	route_design Complete!	22.925	52

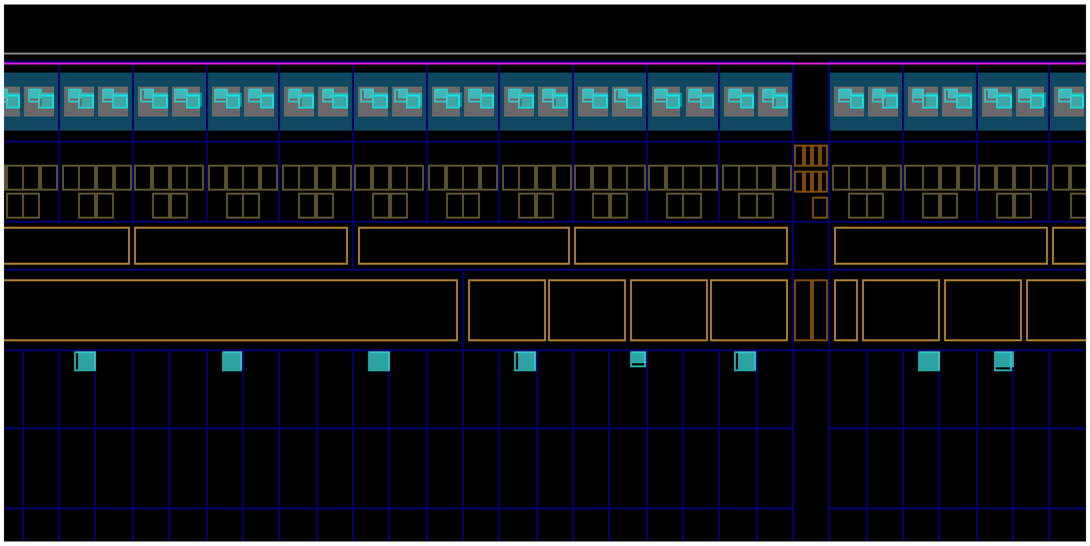
Appendix 5 – 32Bit Carry Select Hardware Usage



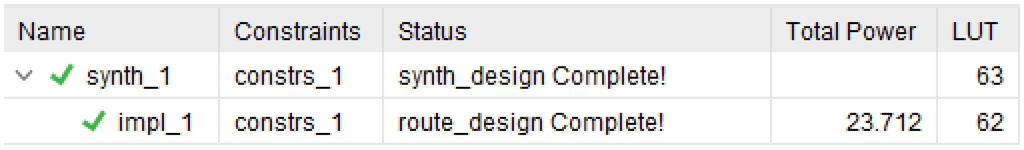
Name	Value
r_ADD_1_ripple[31:0]	ea881c3e
r_ADD_2_ripple[31:0]	c0dfdc26
w_RESULT_ripple[32:0]	1ab67f864
r_ADD_1_carry[31:0]	ea881c3e
r_ADD_2_carry[31:0]	c0dfdc26
w_RESULT_carry[32:0]	1ab67f864

Appendix 6 – 32Bit Carry Select Execution Time (19.120ns)

A.2- 32-Bit Carry Look-ahead Adder:

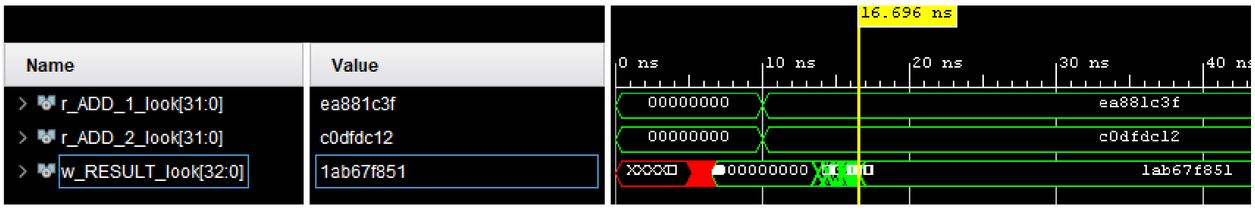


Appendix 7 – 32Bit Carry Look-ahead Hardware Usage



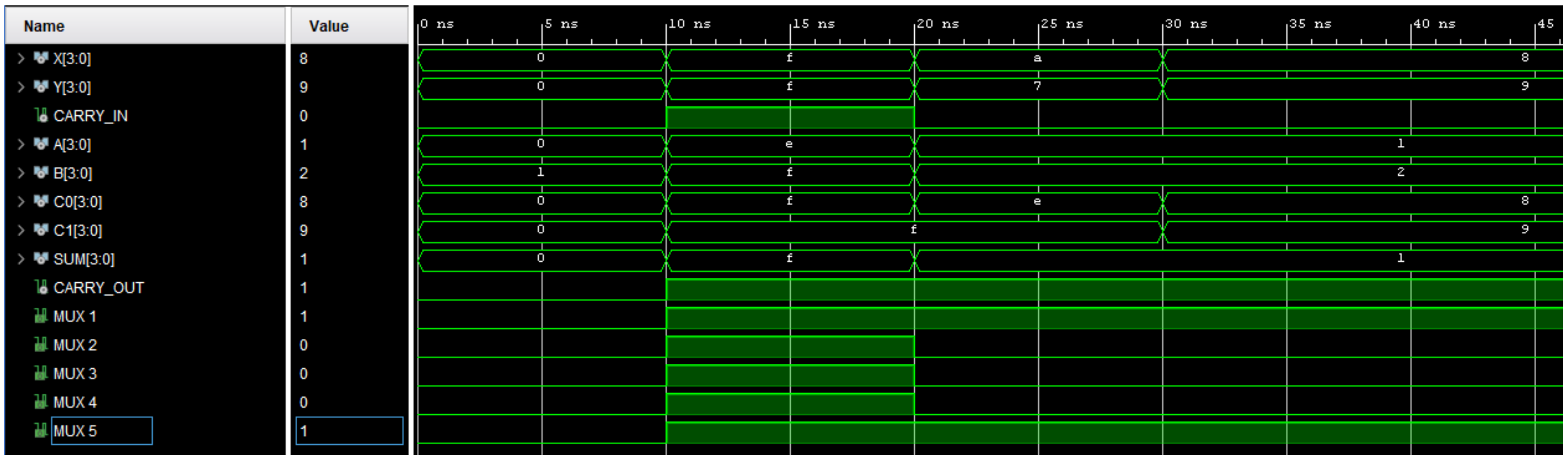
Name	Constraints	Status	Total Power	LUT
✓ synth_1	constrs_1	synth_design Complete!		63
✓ impl_1	constrs_1	route_design Complete!	23.712	62

Appendix 8 – 32Bit Carry Look-Ahead Hardware Usage



Name	Value
r_ADD_1_look[31:0]	ea881c3f
r_ADD_2_look[31:0]	c0dfdc12
w_RESULT_look[32:0]	1ab67f851

Appendix 9 – 32Bit Carry Look-Ahead Execution Time (16.696ns)



Appendix 10 – 32Bit Carry Select Post synthesis Simulation in Details