University of California, Irvine

Team Superheroes in Training

EECS 31L Midterm Project

Design Report

Date: October 30, 2016. 11:00PM

Gerry Su	16325043
Joshua Lazaro	57177939
JiaRui Zhu	75042047
Christopher Kevin Bravo	23748994

Tasks Done by Each Member:

- Gerry Su
    - Creating the descriptions for each module in a way that is easy to visualize
    - Working on Design Report and providing explanations for how each module works both independently and in conjunction with others.
    - Come up with ingenious ideas for the code
- Joshua Lazaro
    - Taking descriptions of each module and translating them into Systemverilog code
    - Created Source files and testbenches to test each module
- JiaRui Zhu
    - Helping with understanding and briefing of project goals and design.
    - Helped make the initial version of the full adder, mux, and and other arithmetic components
    - Help with testing and anaylsis of waveform and schematic data, and helped with compiling the project report.
- Christopher Kevin Bravo
    - Help brainstorm the design project
    - Co-operate in the development of sources for the midterm code
    - Format the final draft of the report
    - Make sure data was accurately reported in the final draft of the report

128-Bit ALU

Purpose: Design basic 128 bit ALU in terms of structural design. 128 1-bit ALUs will be cascaded.

Primary Blocks: Arithmetic, logic, 1-Bit ALU, C_in generator, flag generator

The **1-Bit ALU** consists of:

The **Arithmetic block** is created with 3 modules: an 8to1 input B manipulator mux, a 8to1 carry handler mux, and a 1-Bit full adder. Unused ports are set to default values.

Logic block is created with 2 modules: 8to1 mux for logic operations, and a 8to1mux carry handler. Unused ports are set to default values.

To decide which of the arithmetic's or logic block's result & carry should be passed, we used two 2 to 1 muxes.

C_in[0] generator: Generates the carry_in (that feeds to the first 1-bit ALU) based on the opsel signal. There are only three operations where the carry-in must be 1: Sub, Increment, Add & increment. The rest of the operations should begin with C_in set to 0.

Flag Generator: Generates flags based on the output.

C-flag: output is 1 when c_out[127] = 1;

Z-flag: output is 1 when result[127:0] = 0;

S-flag: output is result[127];

O-flag: output is 1 when cout[127] = 1; (same as C-flag for this ALU)

In this case we only assumed the O-flag addressed overflows, not underflows.

Problems Encountered

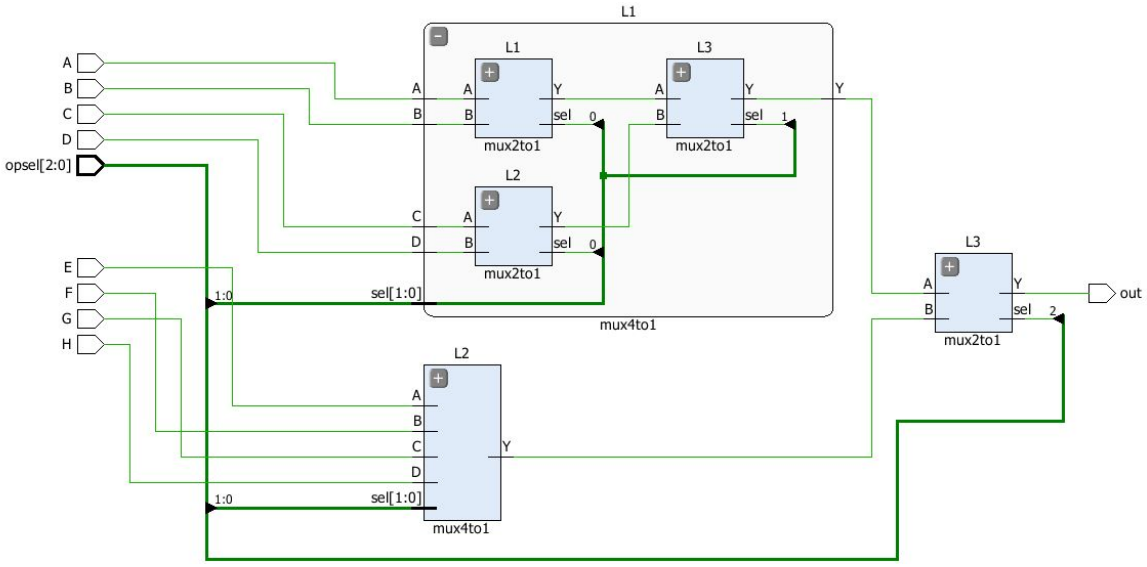
How do we handle the C_in from the previous ALUs? E.g. C_in for ALU[0].

We needed to create the C_in generator with output of 1 for 3 of the 12 operations, else output 0.

We also had difficulty with the O-Flag. We think our assumption and implementation of the O-flag is incorrect. For a proper and correct o-flag, it should address both overflows and underflows. The O-flag should only be evaluated for arithmetic operations like add, add & increment, subtract, subtract with carry, and increment/decrement, and set the O-flag to 0 for logical and other operations. To calculate the O-flag, we should compare the most significant bit of the input(s) and the result. If the MSB differs between the result and the input, then there is sign of a underflow/overflow (assuming the numbers are in 2's complement signed integers). We could not manage to implement this due to the difficulty and complexity of evaluating this O-flag in Vivado. We assumed the O-flag addressed overflows, not underflows.

Detailed Architecture of modules

8to1mux: We used three 2to1 muxes together to create 4to1 mux and two 4to1 mux with 2to1mux to create an 8to1mux. The 8to1mux is used in various components throughout our design.

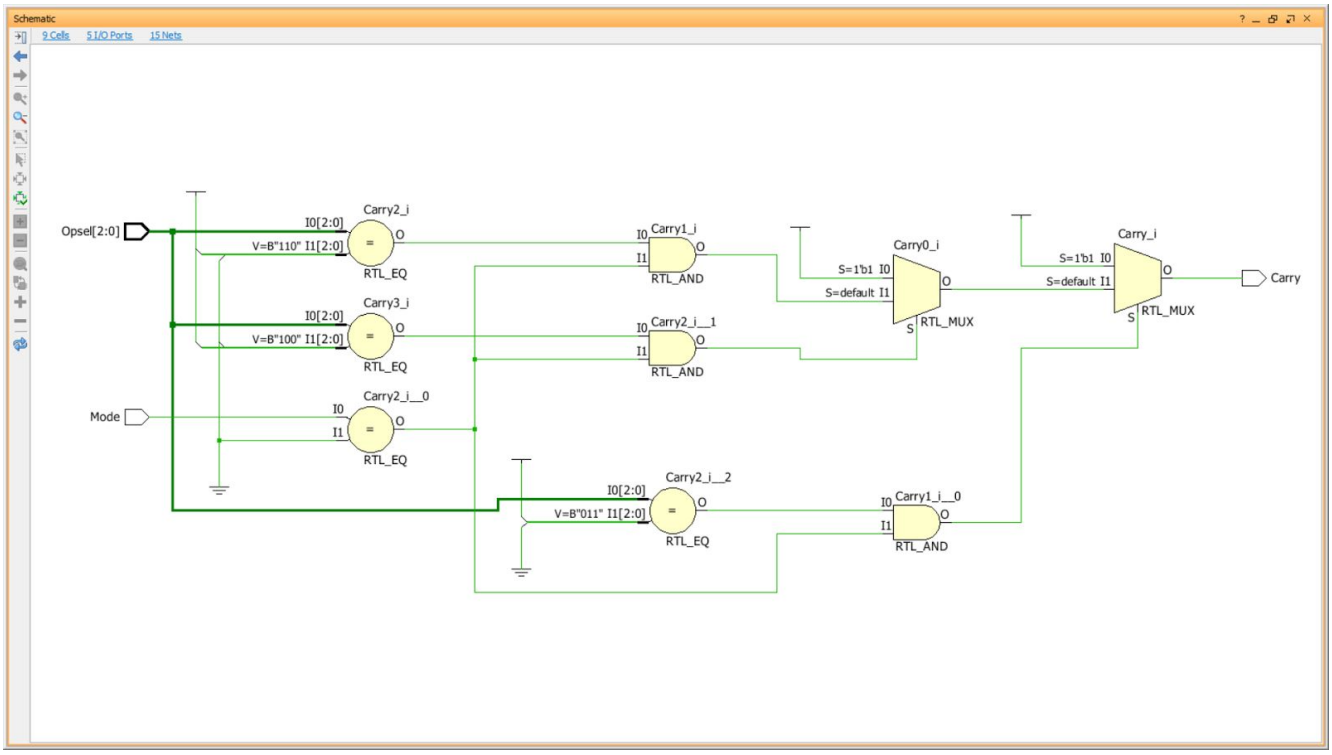


C_in[0] generator:

Input: Opsel, Mode

Output: C_out for (c_in[0])

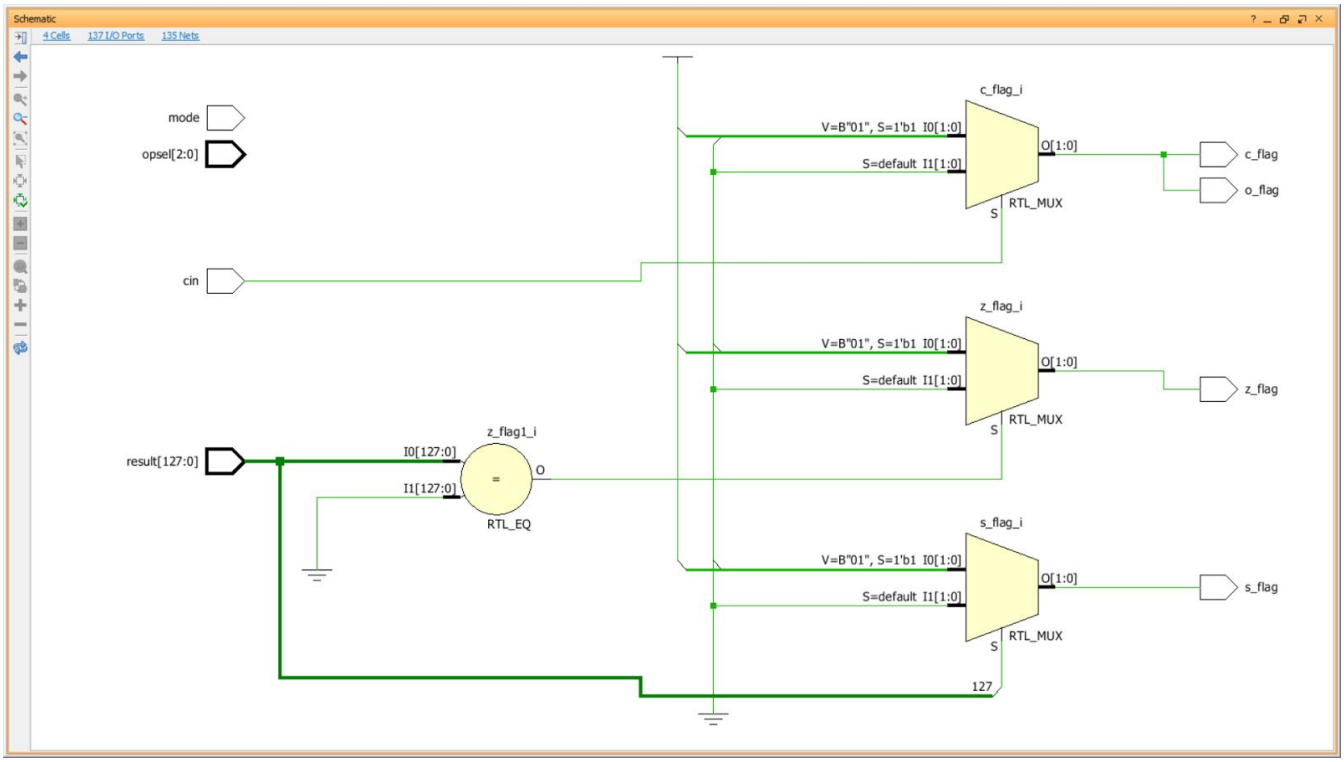
Returns 1 when opsel == 3'b011, 3'b100, 3'b110



Flag generator:

Input: C_in[127], mode (unused), opsel (unused), result[127:0]

Output: c_flag, z_flag, o_flag, s_flag



Arithmetic Block:

Input: A, B, Cin, Opsel

Output: Cout, Result

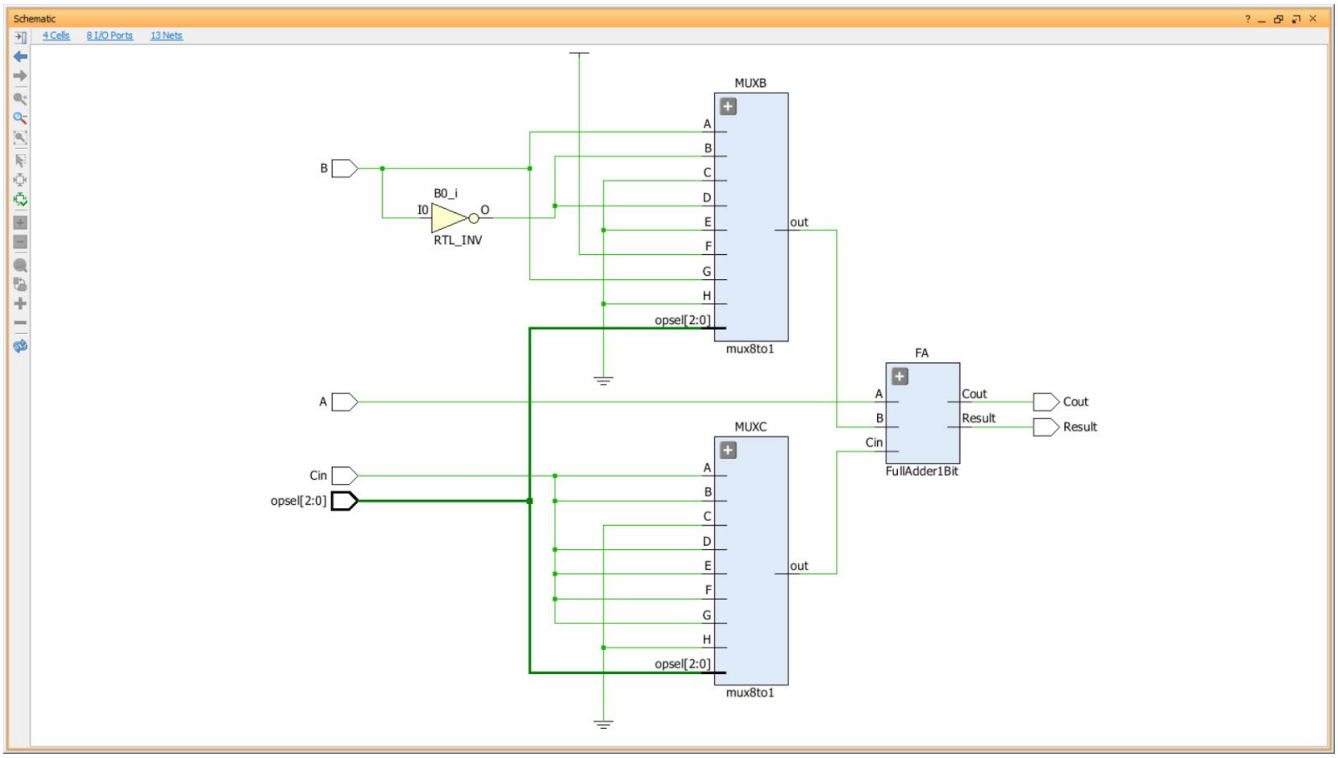
The top 8to1 mux selects the correct input B for a given opsel. We named the inputs in order according to the operations in the assignment.

'A' refers to Add, +b

'B' refers to $+\overline{b}$

'C' refers to move, etc.

The bottom 8to1 mux handles the Cin. The only operation that doesn't take Cin into consideration is move.

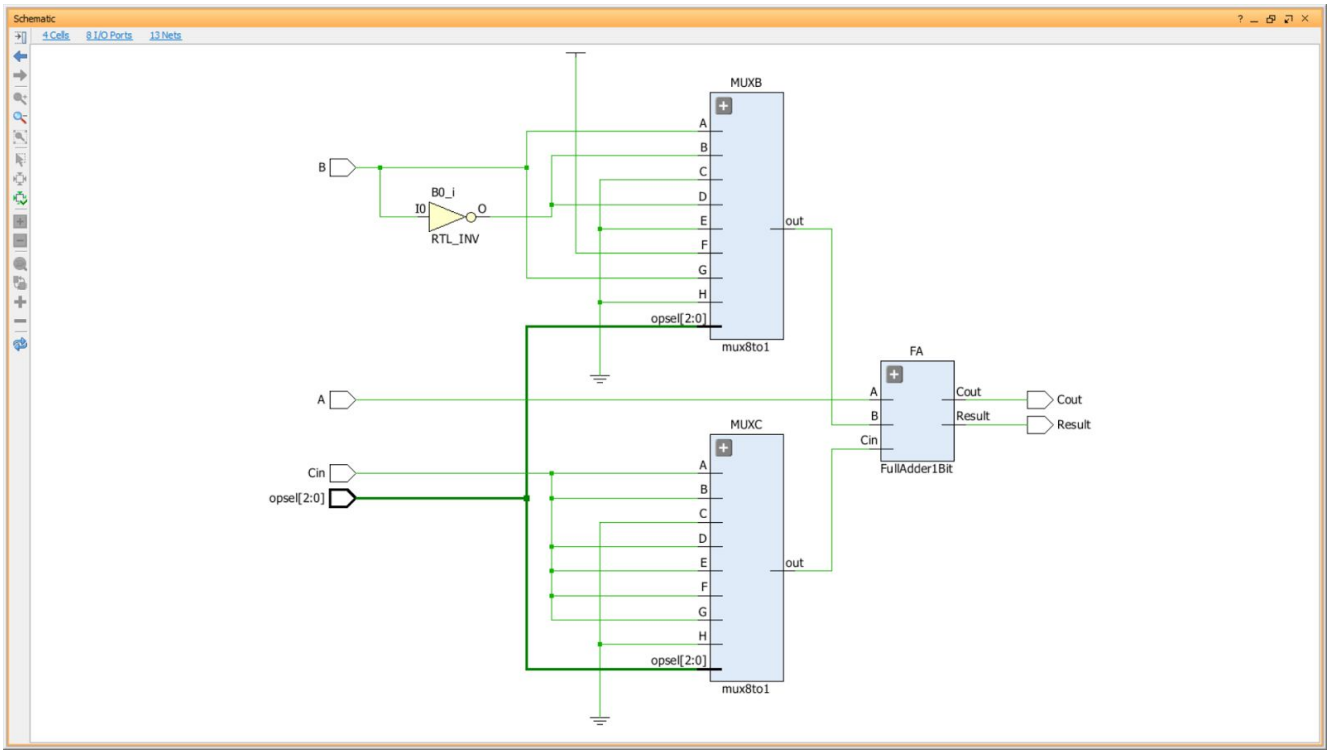


Full Adder:

Input: A, B, Cin

Output: Cout Result

Arithmetic Block schematic



Logic Block

Input: A, B, Cin, Opsel

Output: Cout, Result

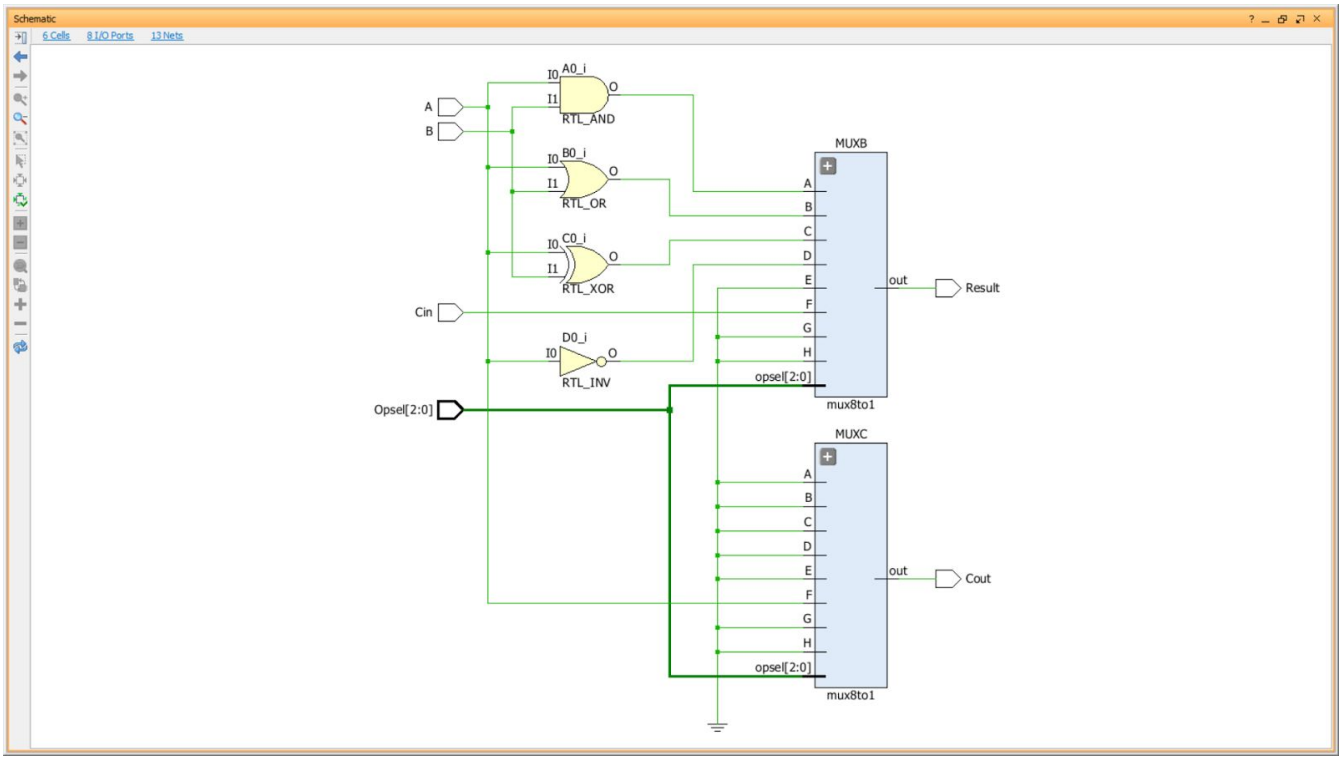
The top 8to1 mux selects the correct input for a given opsel.

'A' refers to A AND B

'B' refers to A OR B, etc.

The bottom 8to1 mux handles the Cin. The only operation that we need Cin for is Shift Left.

Logic block schematic

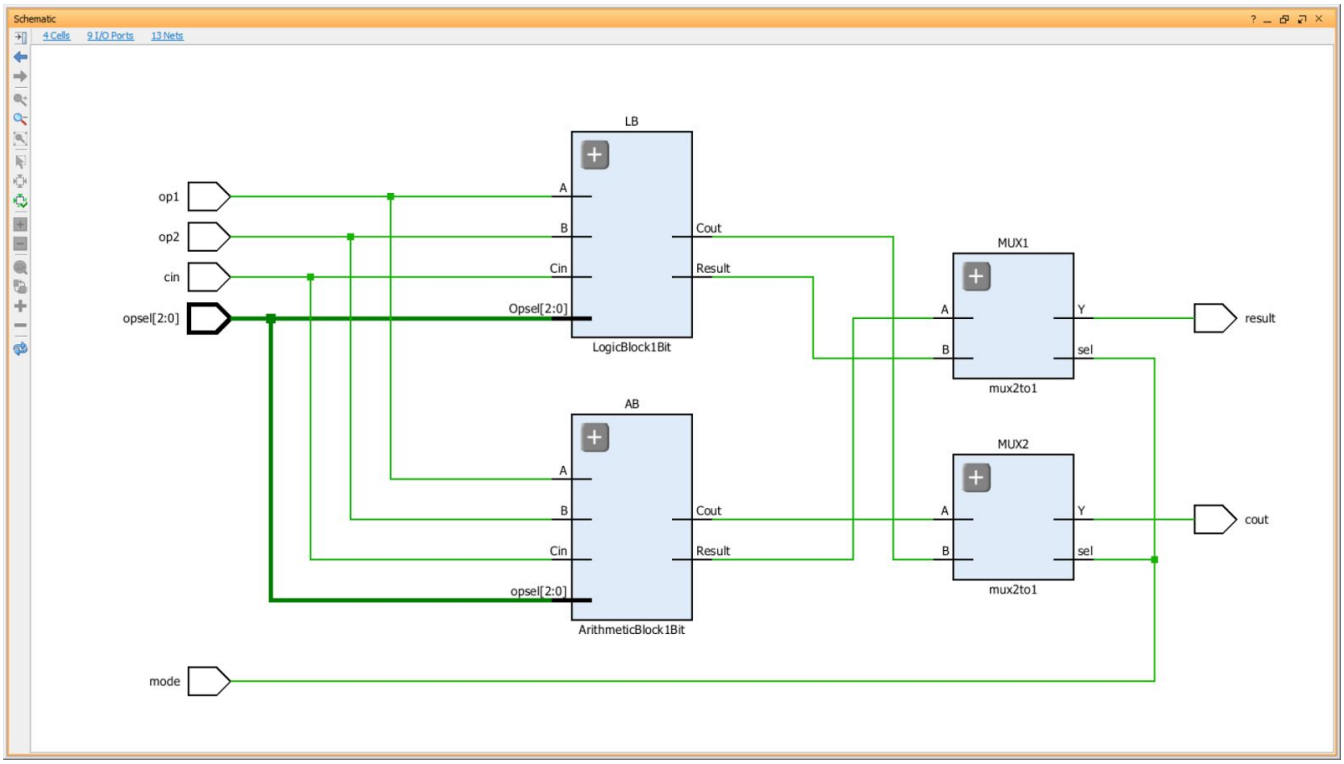


1Bit ALU:

Input: op1, op2, Cin, opsel[2:0], mode

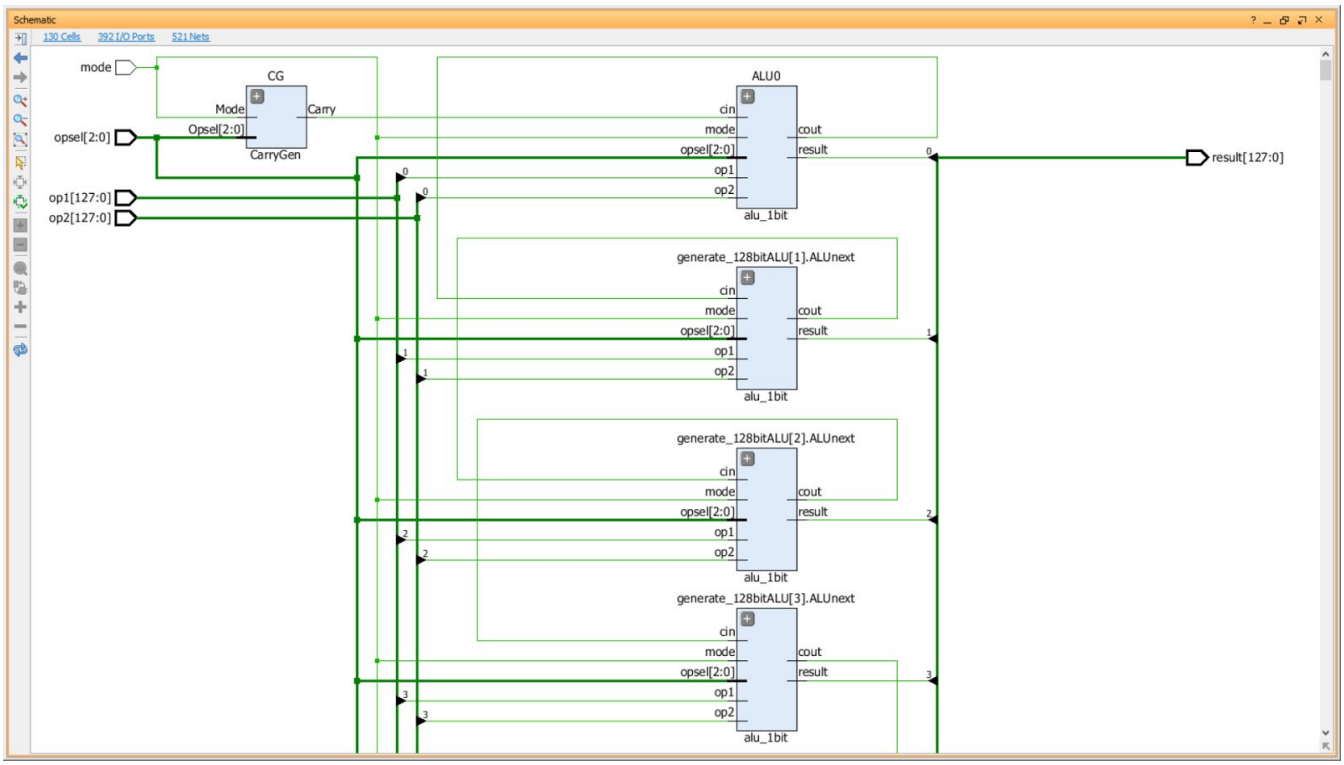
Output: result, Cout

We used two 2to1 muxes to decide which of the arithmetic's or logic block's result & carry should be passed. **1Bit ALU schematic**

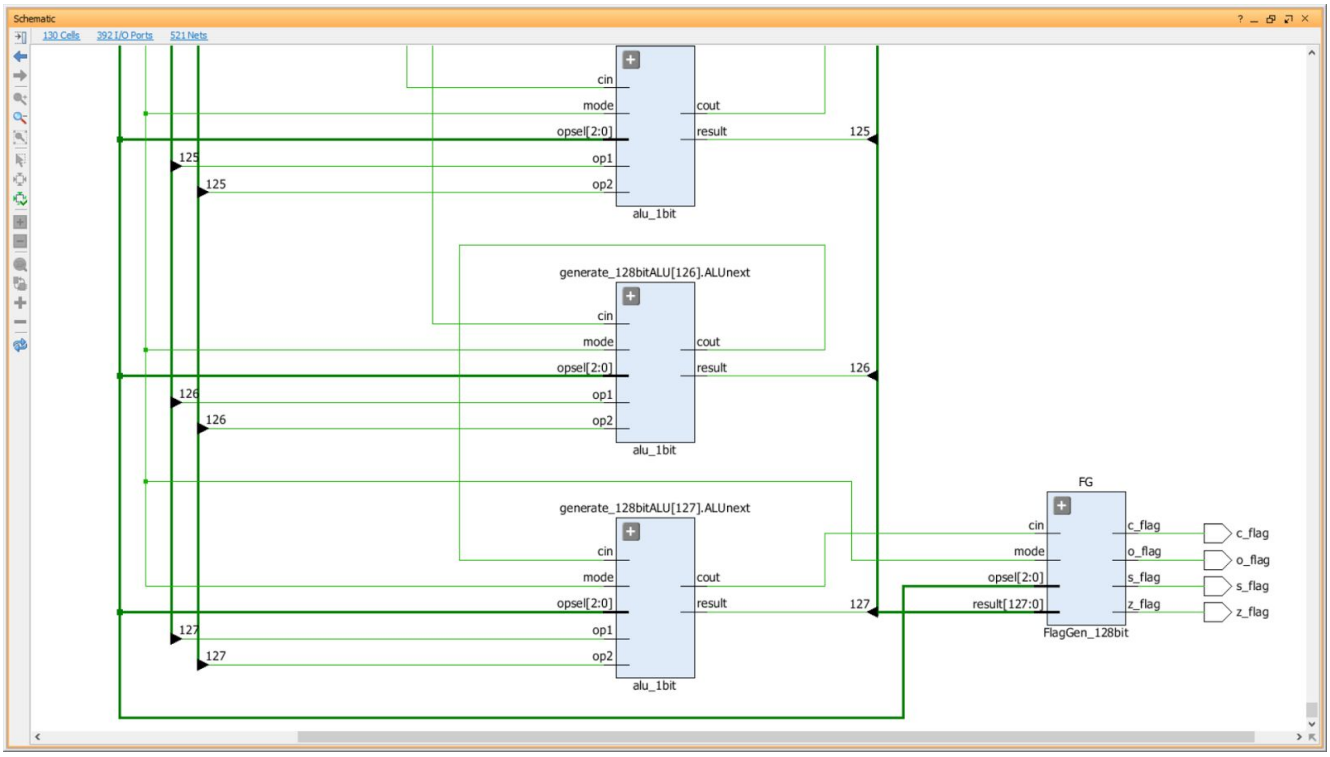


128bit ALU schematic

Beginning of 128 bit ALU, shows carry[0] generator, ALU[0] and cascading of ALUs.

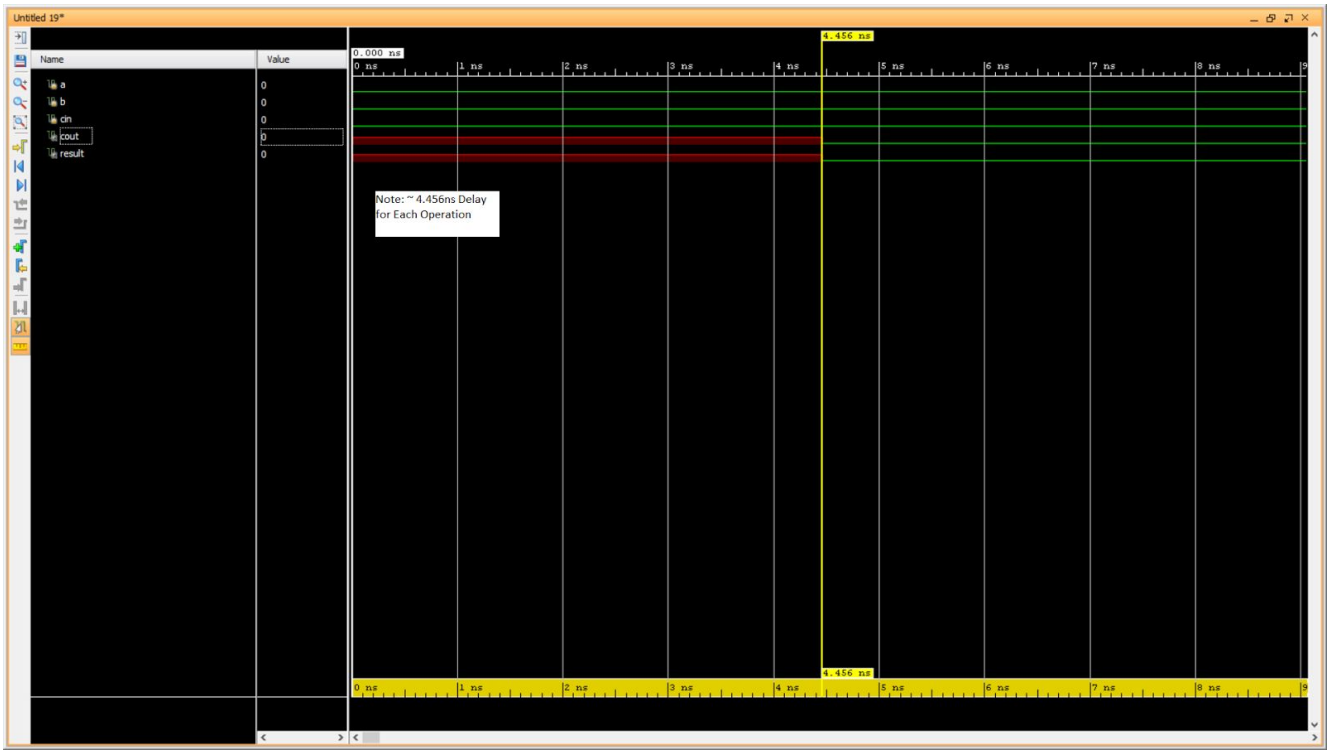


End of the cascade of ALUs, and also the flag generator.



Waveform snapshot for each function w/ brief explanation (Assuming each Gate delay takes 1ns)

Full adder (~4.456ns)

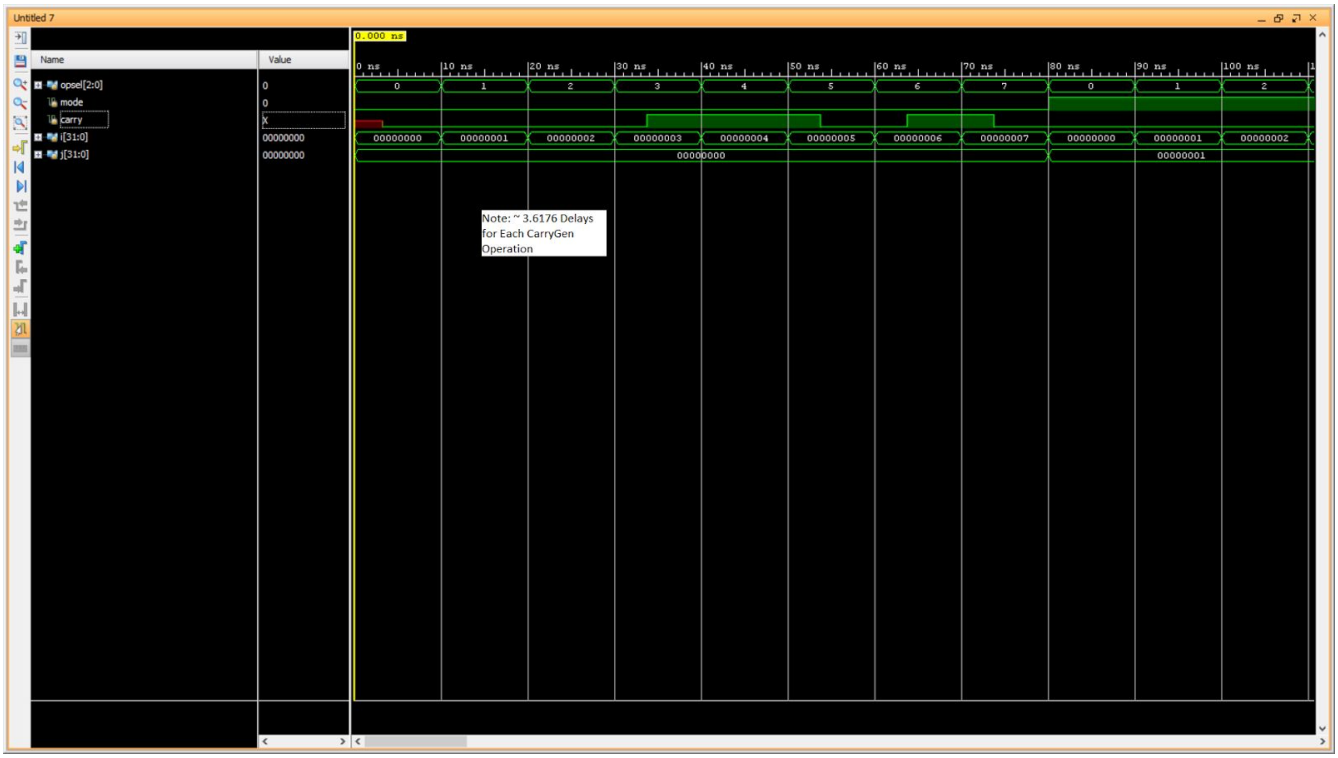


Preliminary Evaluation: The Full Adder involves 3 levels of logic gates. We expected the gate delay to

be around 3ns.

Actual: However, the waveform shows that the delay is actually around 4.456ns.

Carry gen (~3.6176ns)



Preliminary: The Carry Generator involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

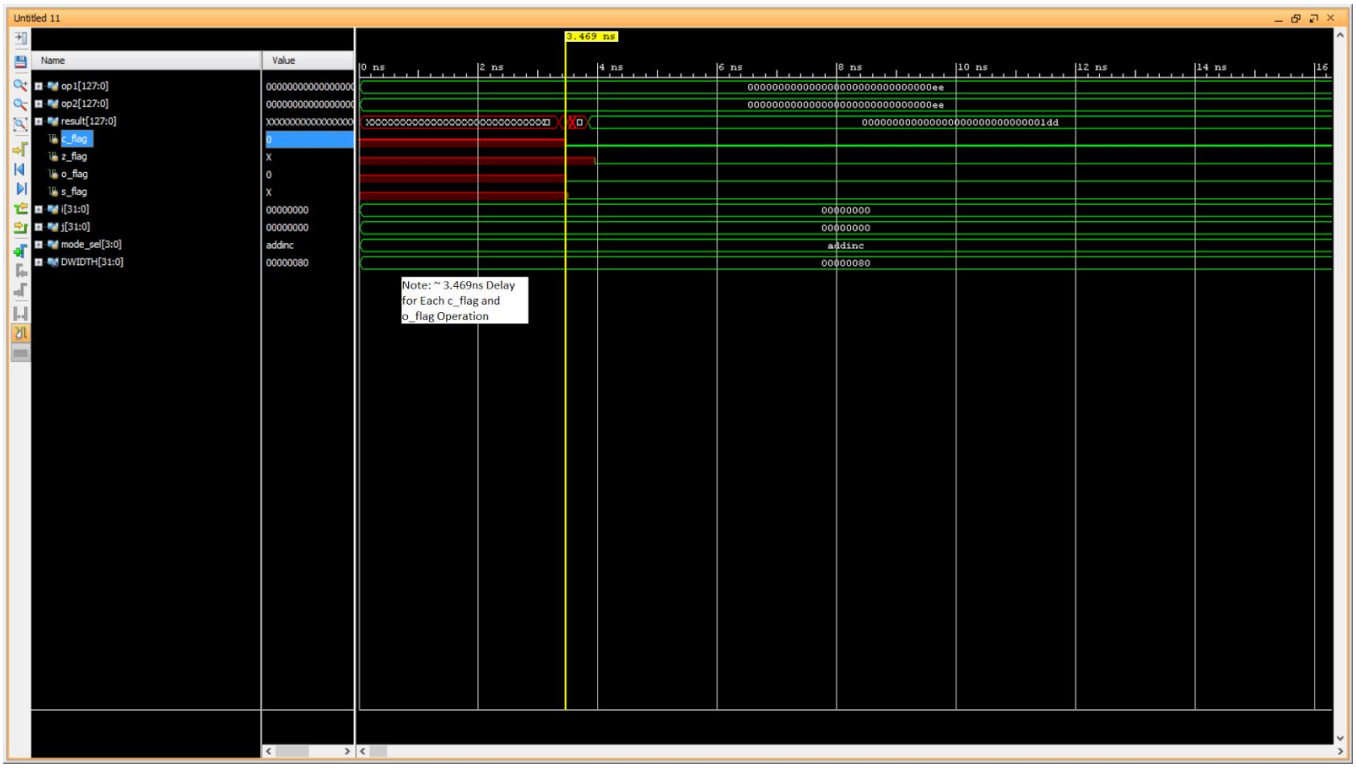
Actual: The waveform shows that the delay is around 3.6176ns, which is not too far from what we expected.

Flag gen

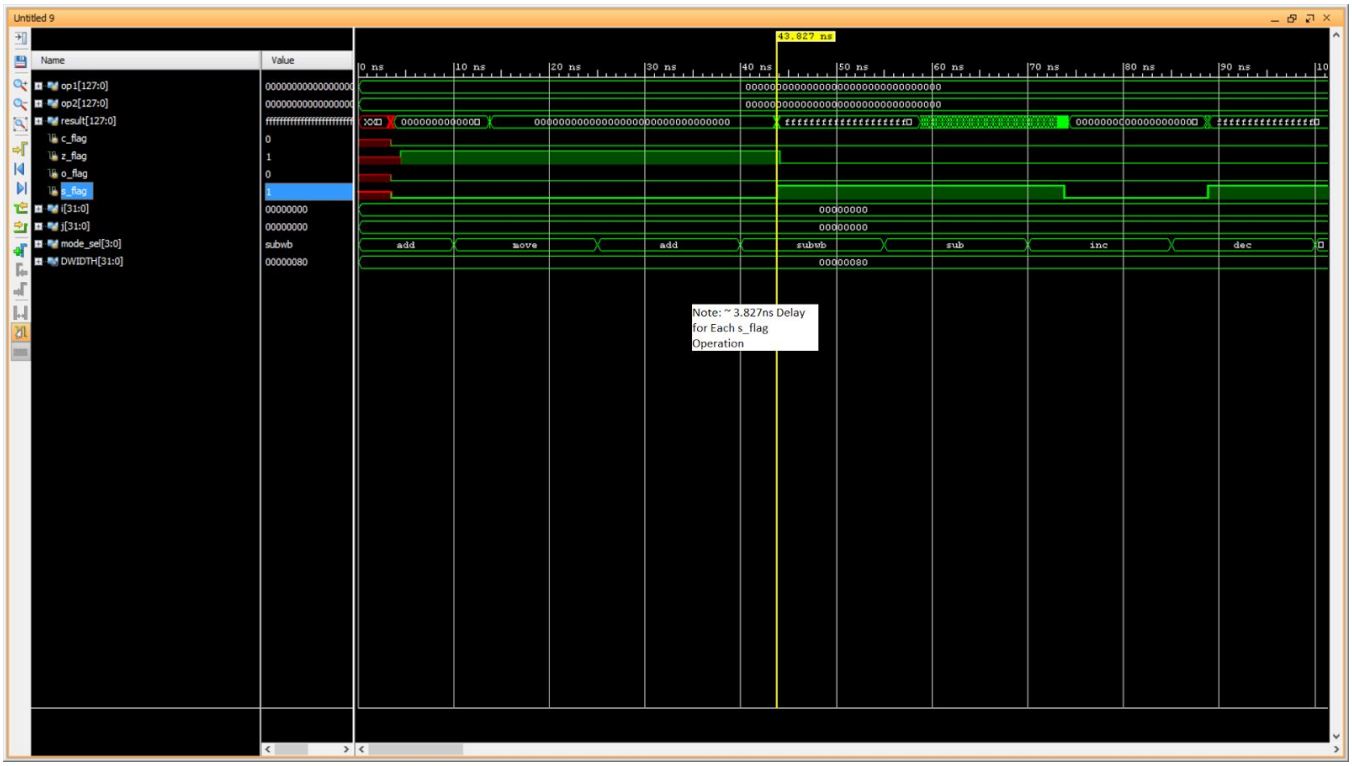
Preliminary: We expected that the z_flag would be calculated much slower than the other flags because the Flag Generator loops through every bit within result[127:0] to find out if result == 0. The Flag Generator involves 2 levels of logic gates with 1 of the level being 2 multiplexers. We expected the delay to be around 2ns for c_flag, s_flag, and o_flag.

Actual: For the c_flag, o_flag, and s_flag, on average, the delay is around 3.648ns. For the z_flag, the delay is 4.433ns, which is indeed the greater delay.

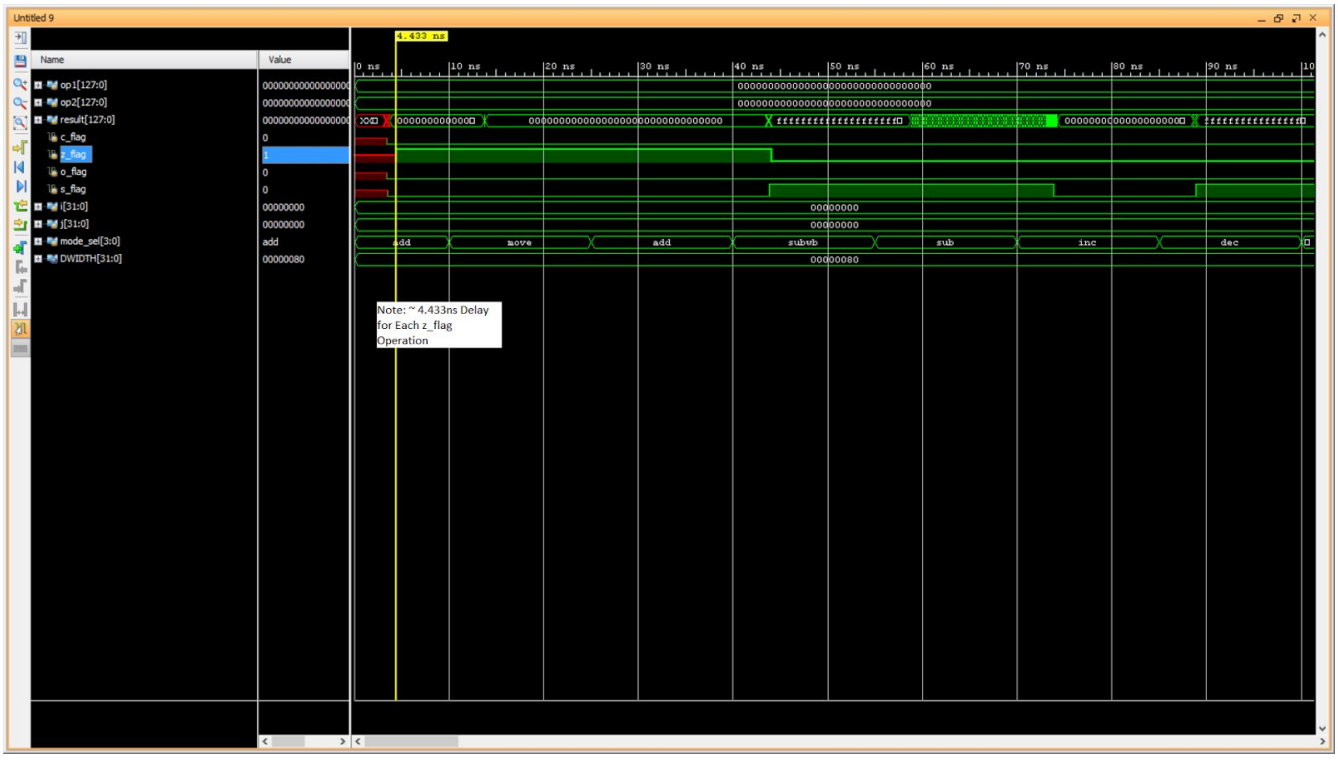
C_flag and O_flag (~3.469ns)



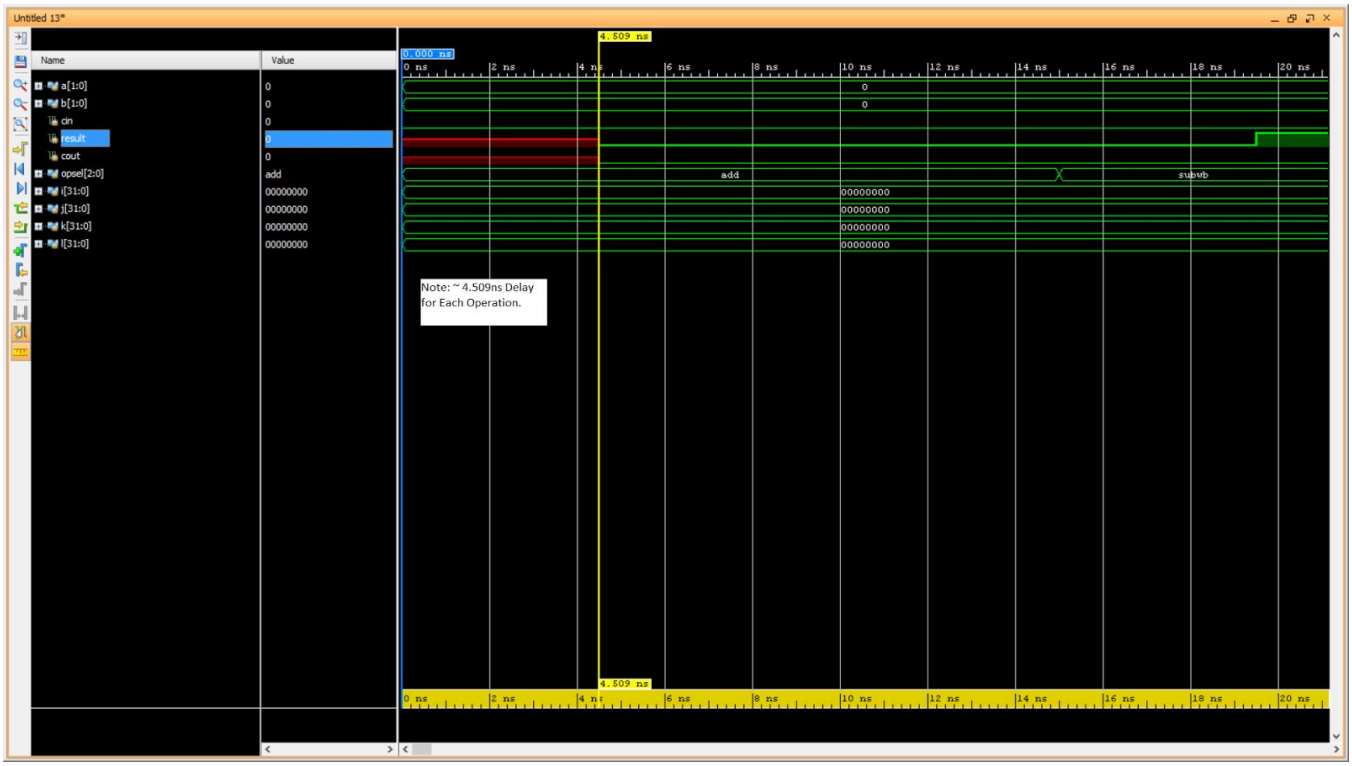
S_flag (~3.827ns)



Z_flag (~4.433ns)



Arithmetic block (~4.509ns)

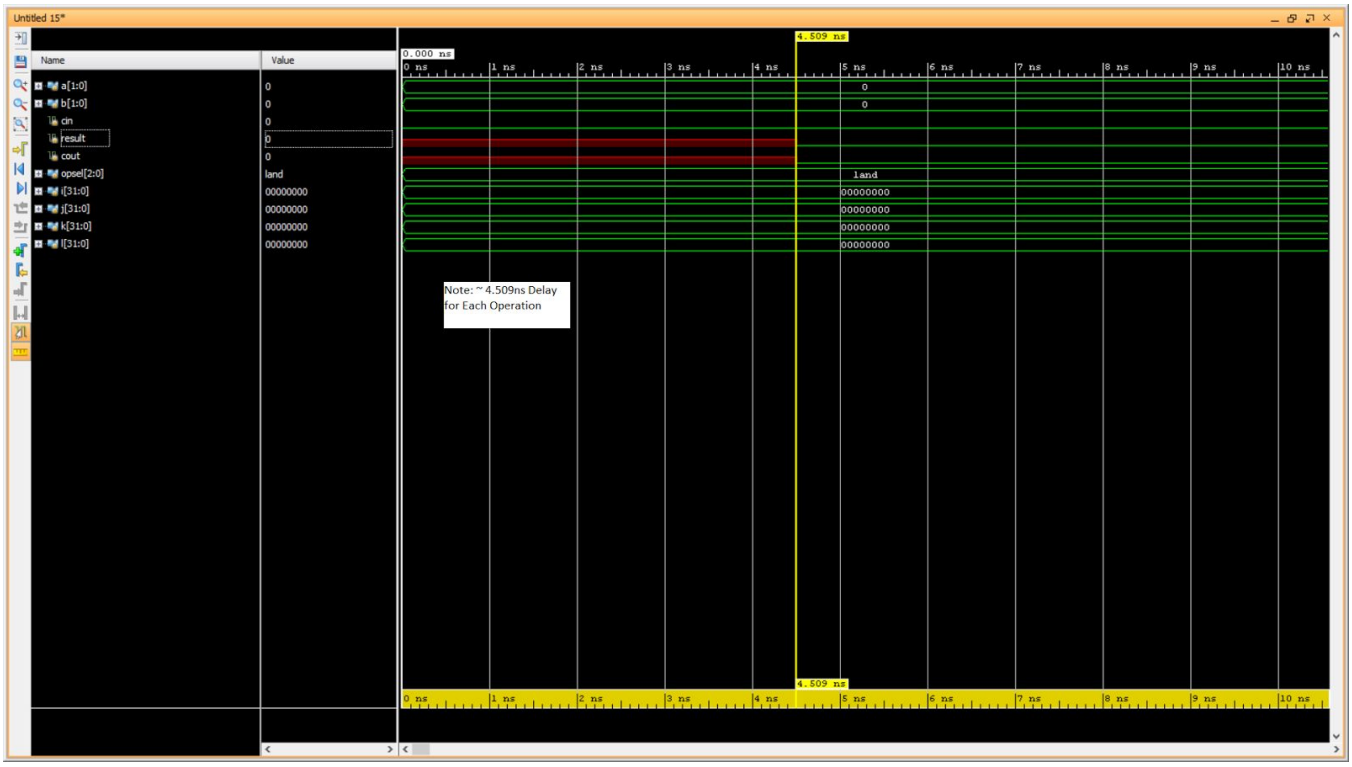


Preliminary: The Arithmetic Block involves 4 levels of logic gates: three muxes and 1 not-gate. We

expected the gate delay to be around 4ns.

Actual: The waveform shows that the delay is around 4.51ns, which is not too far from what we expected.

Logic block (~4.509ns)



Preliminary: The Logic Block consists of 3 levels of logic gates: three muxes We expected the gate delay to be around 3ns.

Actual: The waveform shows that the delay is around 4.51ns, which is significantly different than what we expected.

1-bit ALU (~4.750 ns)

Preliminary: The 1-bit ALU consists of 5 levels of logic gates: four muxes and other logic gates. Because of this, we expected the delay to be around 5ns.

Actual: The waveform shows that the delay for the longest operation in the 1-bit ALU was 4.750 ns, which was close to our expectations.

128-bit ALU

Preliminary: The 128-bit ALU consists of 5 levels of logic gates: four muxes and other logic gates. Because of this, we expected the delay to be greater than 5ns because of the carry delay.

Actual: The waveform shows that the delay is around 5.899 ns for the longest operation (add and then increment), which was within our predictions. There was a longer delay for the 128-bit ALU compared to the 1-bit ALU due to the time which the carry needs to be cascaded down all 128 ALUs, and also due to the additional number and layer of logic gates in the 128-bit ALU. There are some operations that perform faster than the others. For example, the increment operation was the fastest, at 0.167ms for one operation. This was due to the simplicity of this operation and lack of reliance on the carry travelling down the cascaded 128bit ALUs.

Add (~4.052 ns):

Preliminary: The addition operation consists of 5 levels of logic gates: four muxes and other logic gates. Because of this, we expected the delay to be around 5ns.

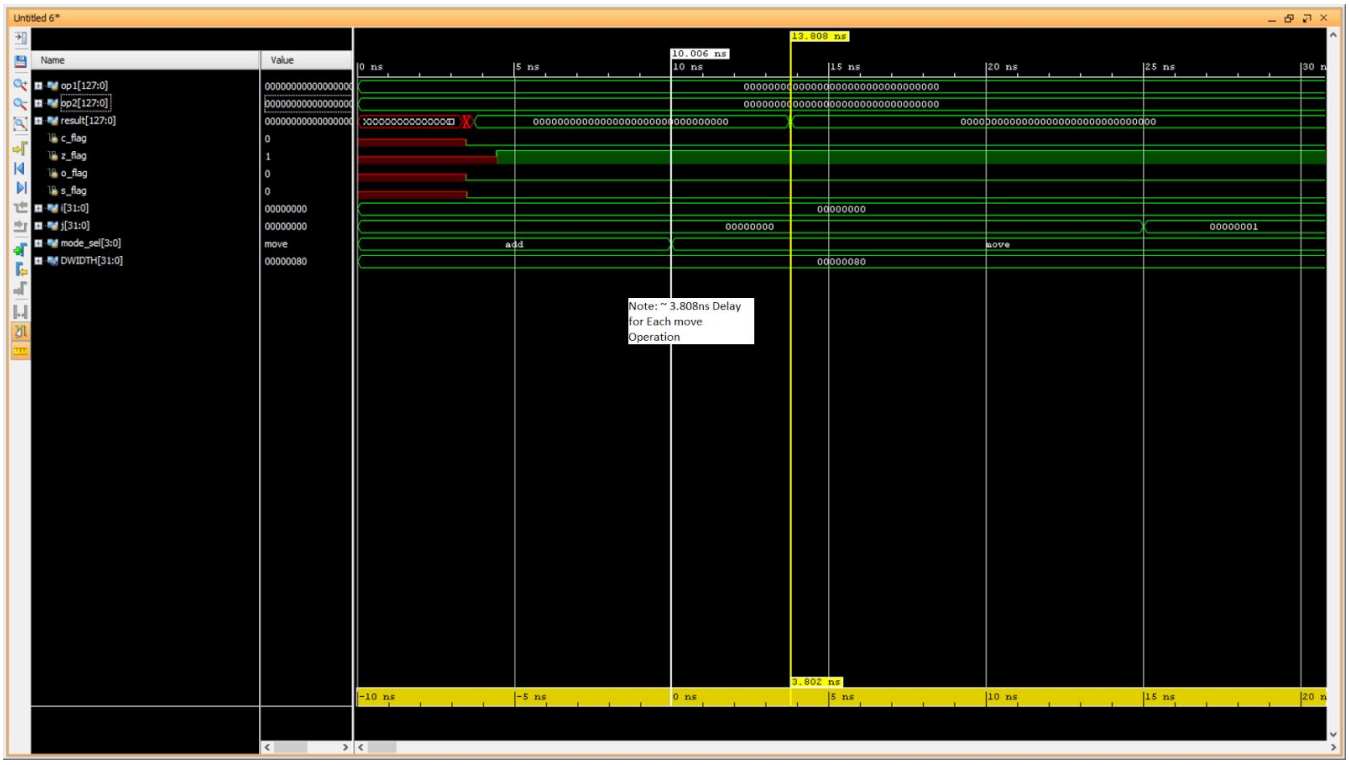
Actual: The waveform shows that the delay is actually around 4.052 ns.

Subwb (~3.721 ns):

Preliminary: The subtraction operation with borrow consists of 5 levels of logic gates: four muxes and other logic gates. Because of this, we expected the delay to be around 5ns.

Actual: The waveform shows that the delay is actually around 3.721 ns.

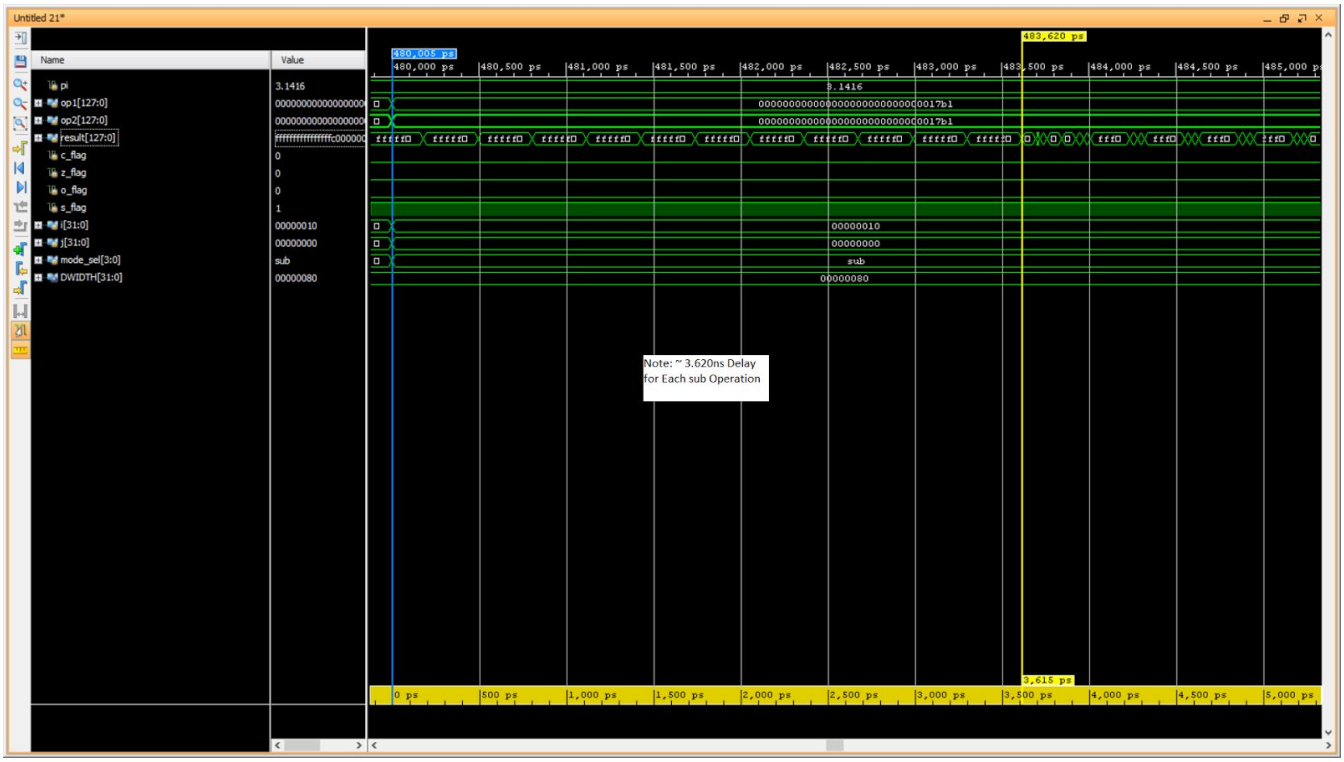
Move (~3.808 ns):



Preliminary Evaluation: The move operation involves 3 levels of logic gates. We expected the gate delay to be around 3ns.

Actual: However, the waveform shows that the delay is actually around 3.808ns.

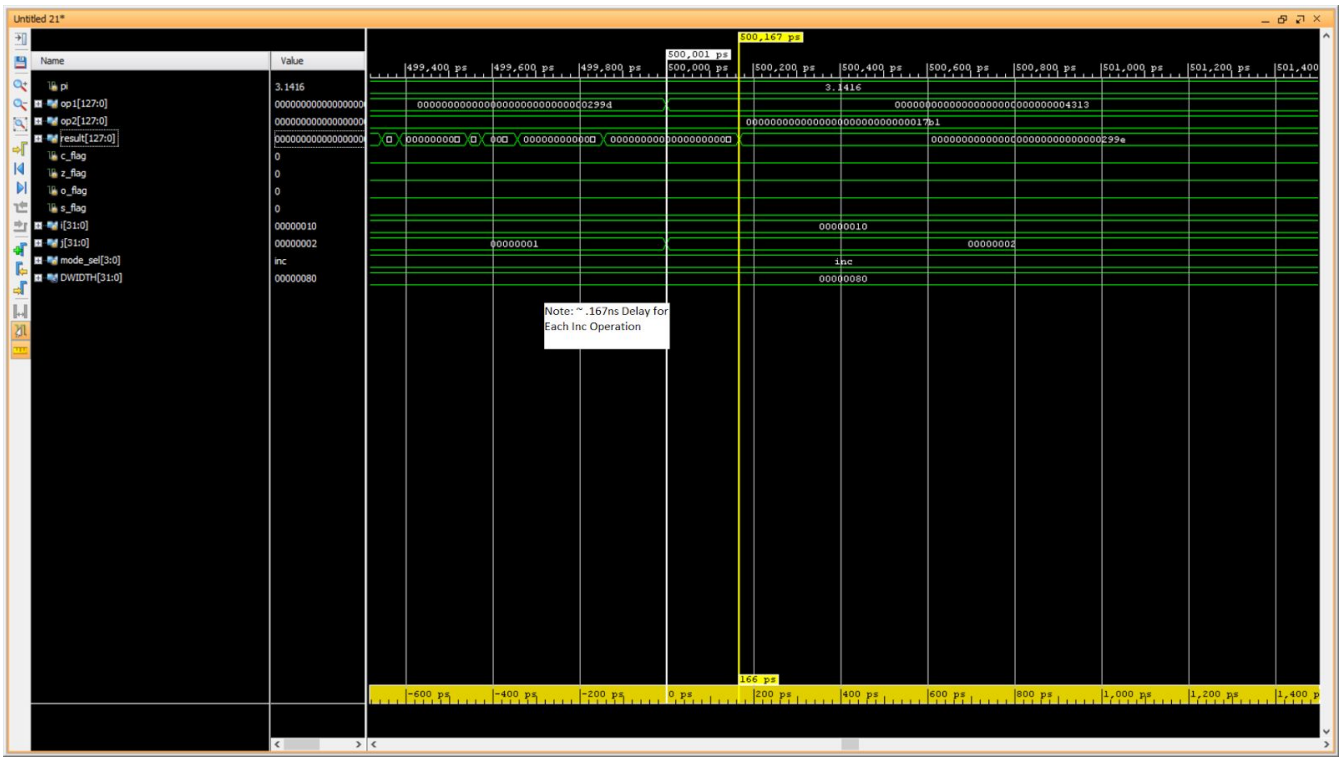
Sub (~3.620 ns):



Preliminary Evaluation: The subtract operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: The waveform shows that the delay is actually around 3.602ns, which is close to our approximation.

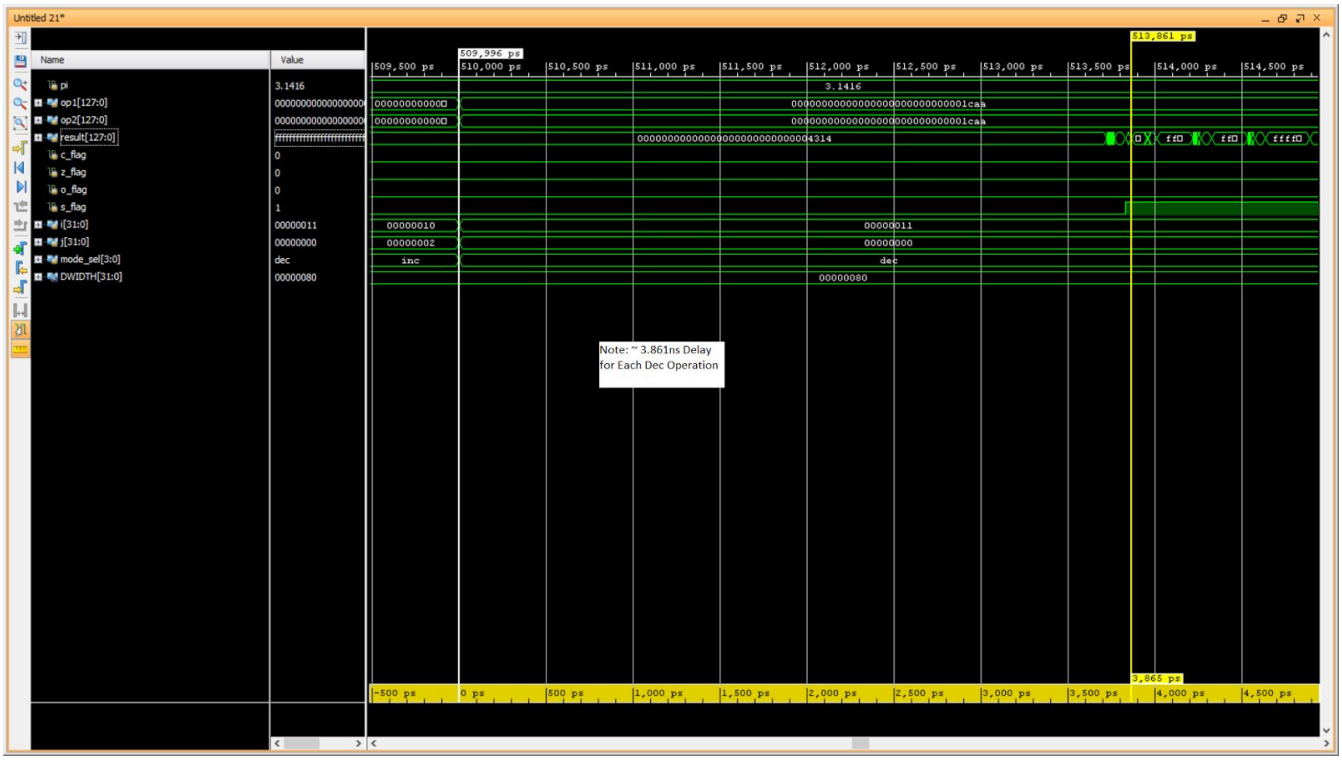
Inc (~.167 ns):



Preliminary Evaluation: The increment operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 0.167ns, which was way less than we initially expected. This is probably because the carry does not need to travel down all 128 bits of ALUs, and that only a few ALUs were involved with handling a carry of 1 in the incrementing process. Also, the increment function would have a lesser delay than the decrement is because that it does not have to subtract and borrow. In essence, the timing of the increment function depends on the input a. If the input is a small value, then in theory the carry bit will travel through less ALUs as a 1 than if the input was a large value, thus reducing the timing needed to perform additions on the FA and carrying a 1 to the subsequent ALU.

Dec (~3.861ns):



Preliminary Evaluation: The decrement operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

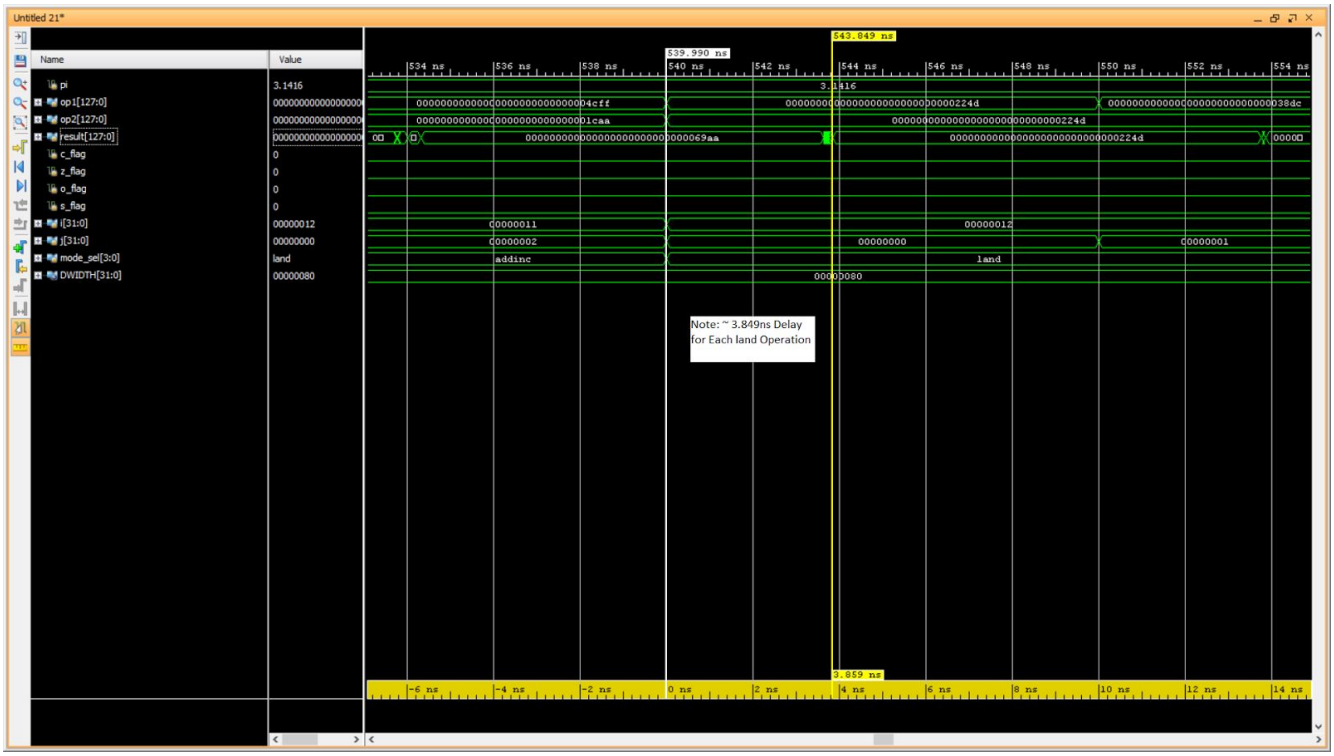
Actual: However, the waveform shows that the delay is actually around 3.861ns, which was close to our approximations.

Addinc (~5.899 ns):

Preliminary Evaluation: The add and increment operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 5.899ns. This is surprising, because the normal add operation in the 128-bit ALU takes 4.052ns, and the increment takes around 0.167ns. Even with both of these operations combined, they do not add up to the 5.8999ns figure that we had in this waveform. Perhaps that the delay for the increment in this case became very long because of the need for the carry to traverse through a large amount of ALUs, which can take a longer time.

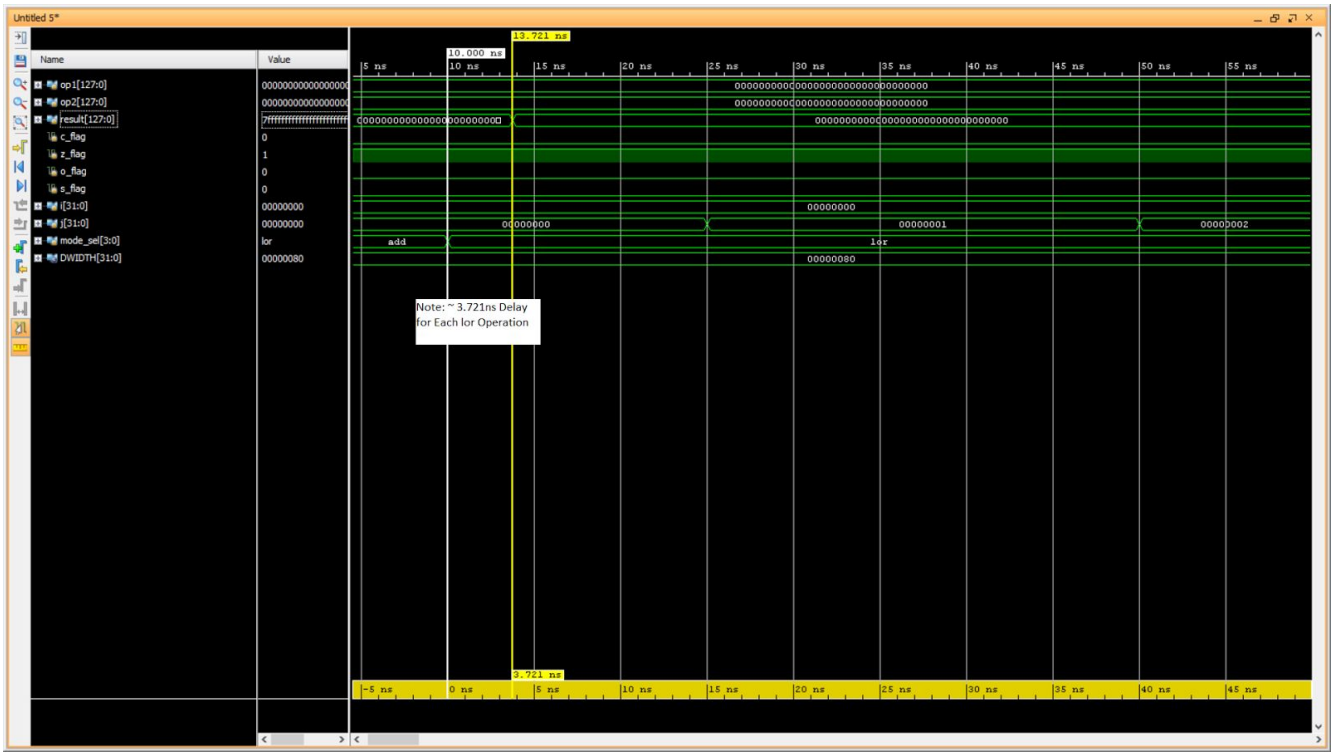
Logical AND (~3.849 ns):



Preliminary Evaluation: The logical AND operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 3.849ns. This is within our expectations.

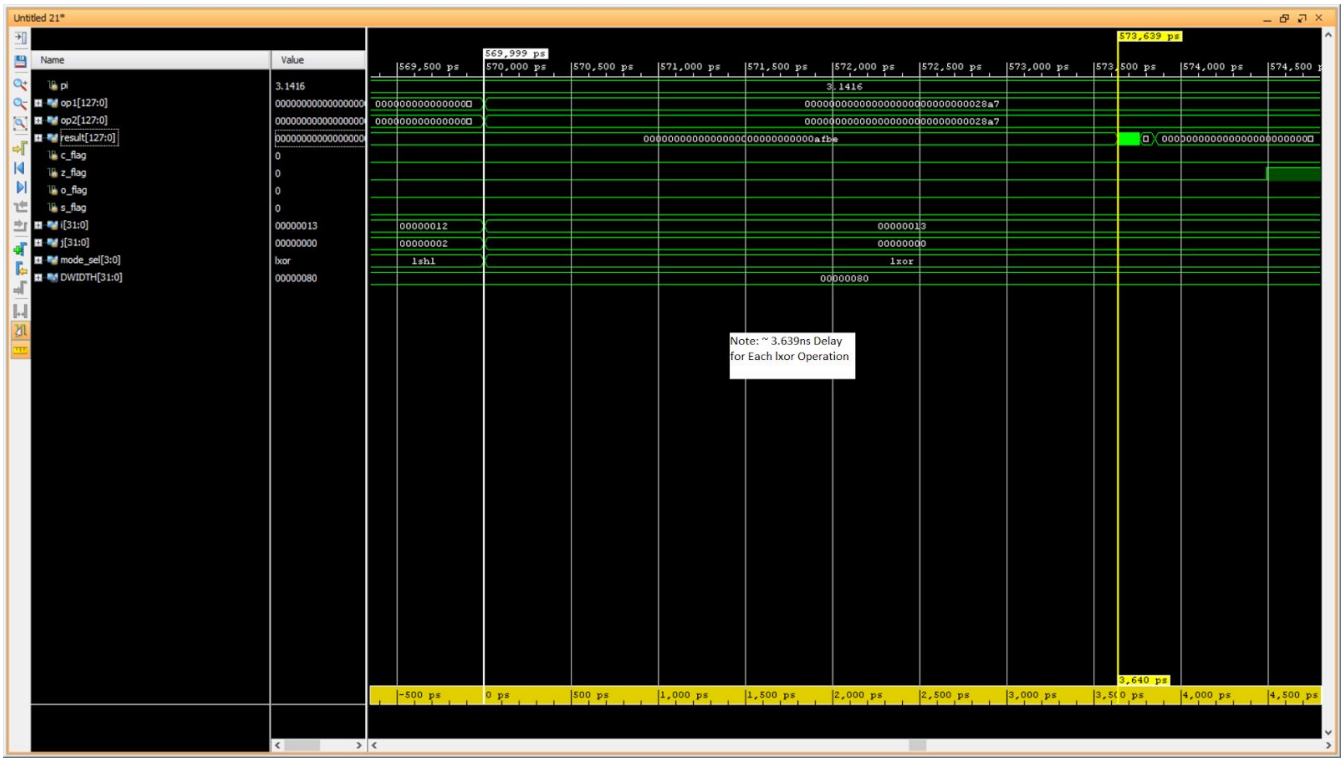
Logical OR (~3.721 ns):



Preliminary Evaluation: The logical OR operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 3.721ns. This is within our expectations.

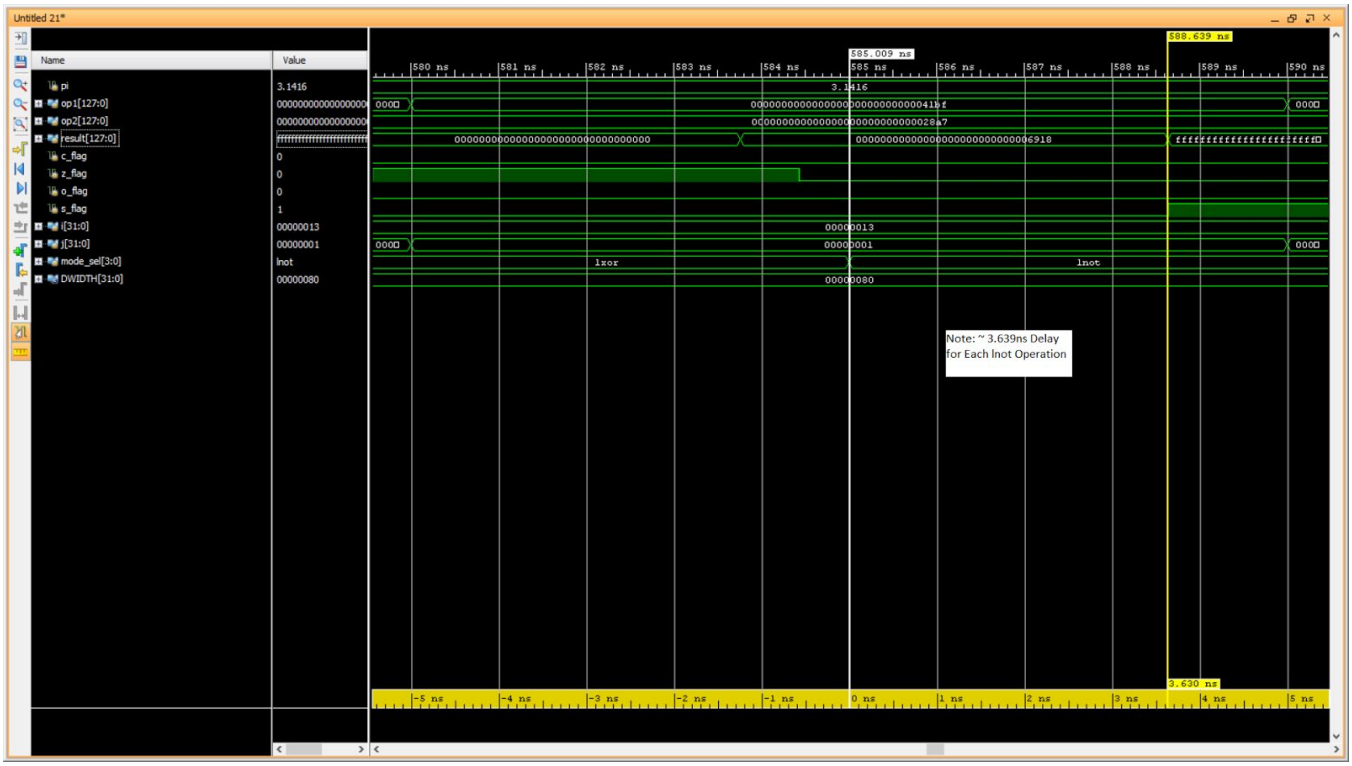
Logical XOR (~3.639 ns):



Preliminary Evaluation: The logical XOR operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 3.639ns. This is within our expectations.

Logical NOT (~3.639 ns):



Preliminary Evaluation: The logical NOT operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 3.639ns. This is within our expectations.

Left SHIFT (~3.639 ns):

Preliminary Evaluation: The logical SHIFT operation involves 4 levels of logic gates. We expected the gate delay to be around 4ns.

Actual: However, the waveform shows that the delay is actually around 3.639ns. This is within our expectations.