Course Name: EMBEDDED SYSTEMS I / III

Course Number and Section: **14:332:493:03 / 16:332:579:05**

Year: Spring 2024

Lab Report #: 2

Lab Instructor: Milton Diaz

Student Name and RUID: Ruben Alias 207005068

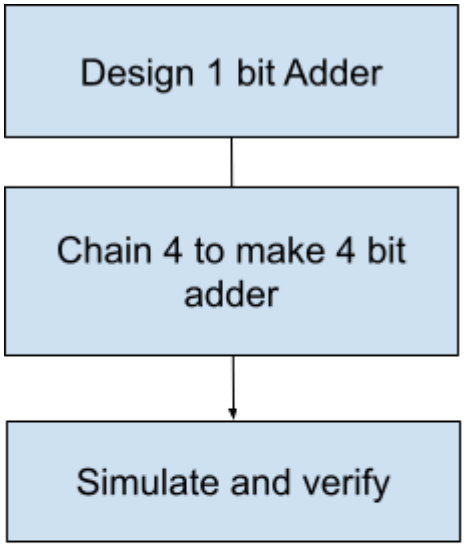
Date Submitted: 03/01/2024

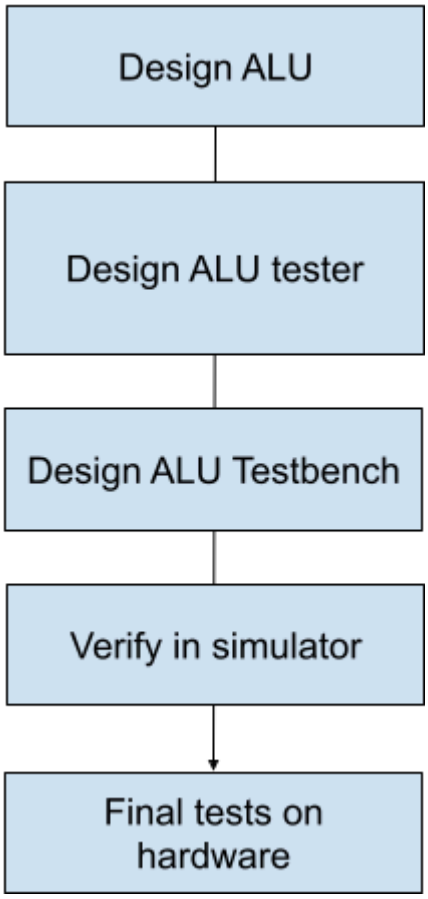
GitHub Link:

https://github.com/embedded-systems-1-spring-2024-labs/lab-2-Herxity

Purpose/Objective: The purpose of this lab was to design an Arithmetic Logic unit capable of 4-bit calculations, 16 total. It required opcode decoding, as well as loading calculations into registers.

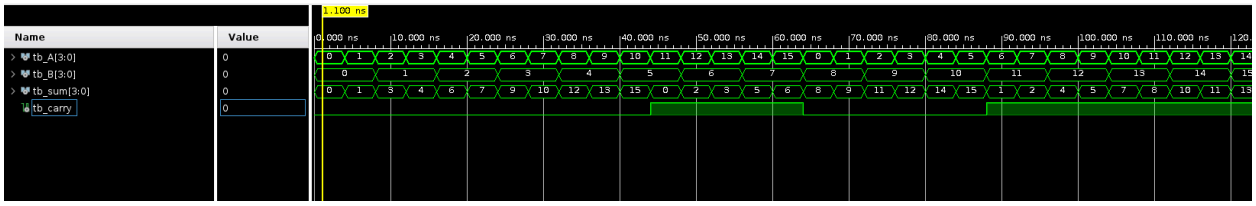
Theory of Operation:





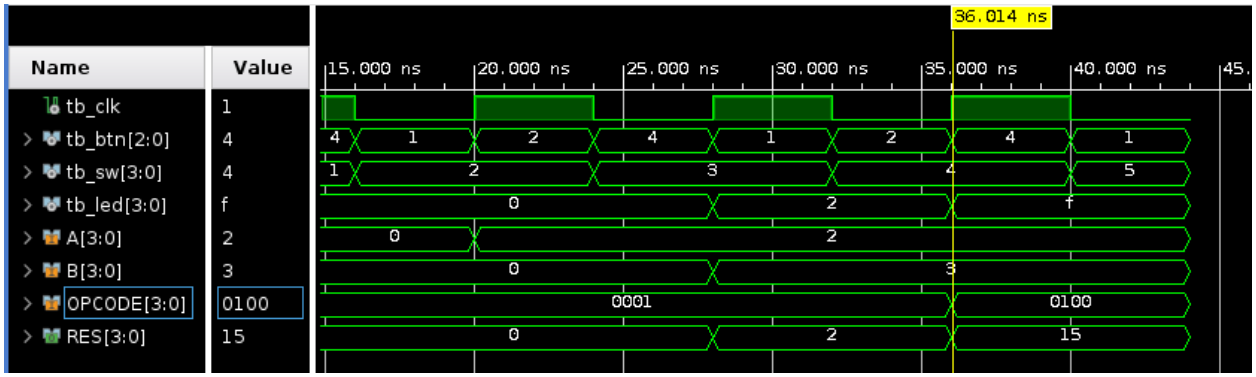
Simulation Waveforms:

Ripple Adder Sim



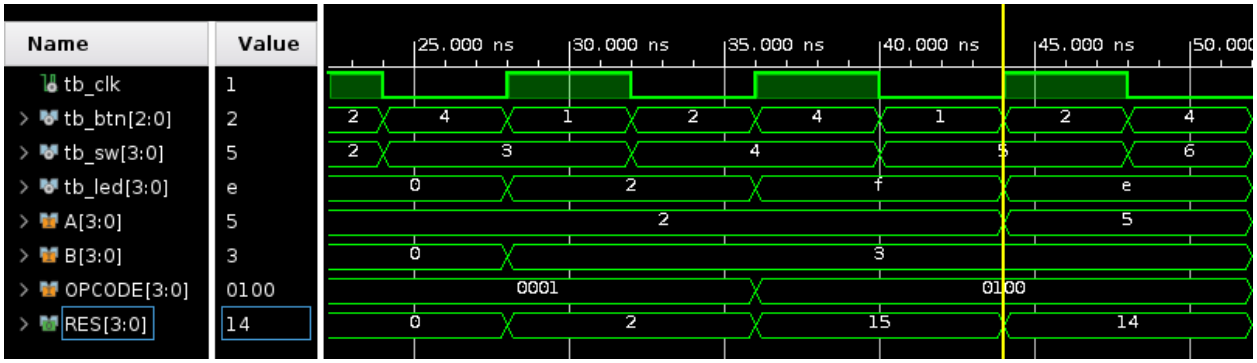
Testbench(This is just a test of basic ability to perform the calculations, in the absence of debouncing due to time):

1. **Subtraction**



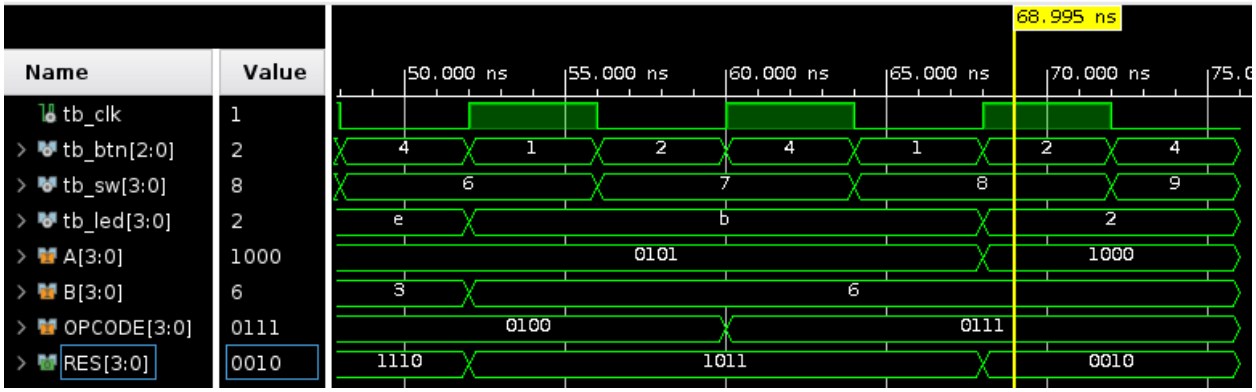
A = 2
B = 3
OPCODE = 0001
RES = 15
0010 - 0011 = 1111

2. **0-A**



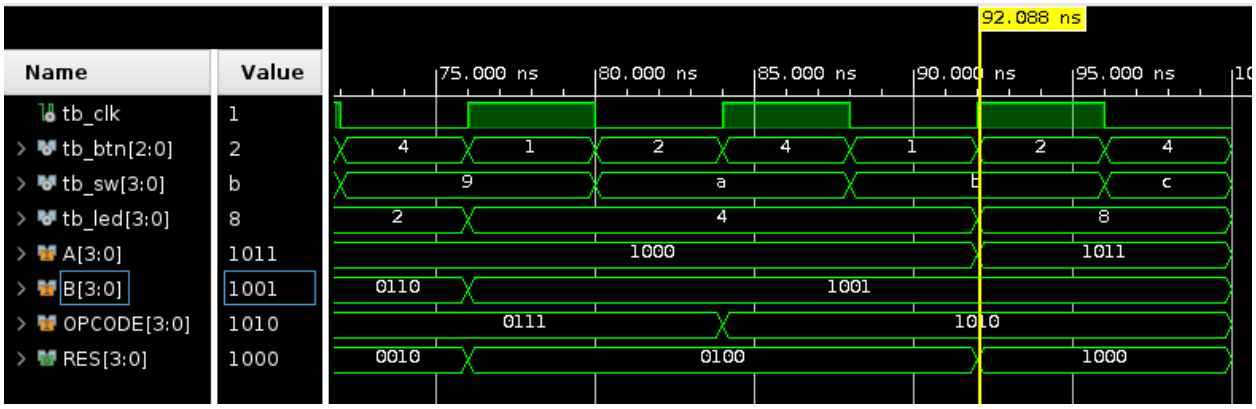
A = 2
B = 3
OPCODE = 0100
0-2=14

3. **Shift Right**



A = 0101
B = 0110
OPCODE = 0111
RES = 0010
Shifting right the bits of 0101 by 1, and padding a 0, yields 0010

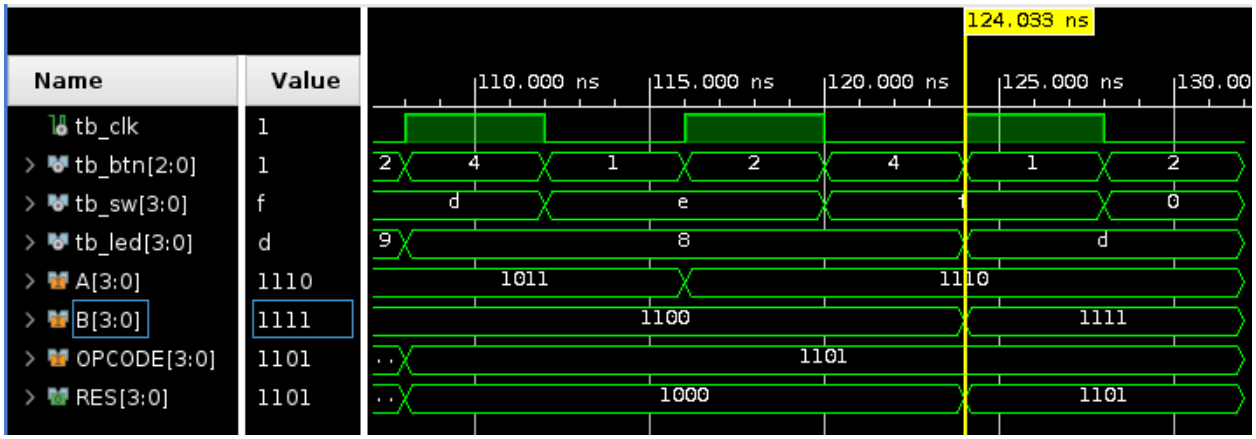
4. **Bitwise AND**



A = 1000
B = 1001

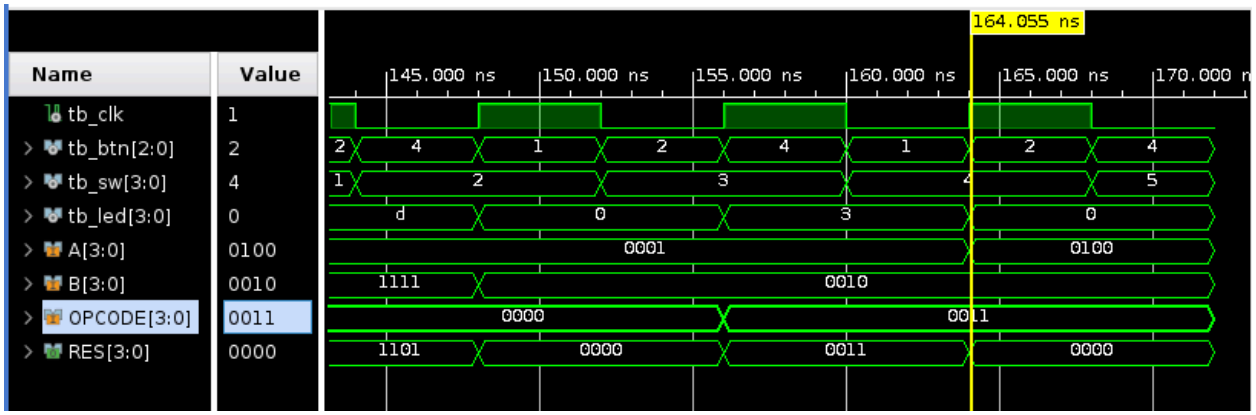
OPCODE = 1010
RES = 1000
Bitwise and of 1000 and 1001 would yield 1000 as the 1's of both bits match then

5. **Bitwise XNOR**



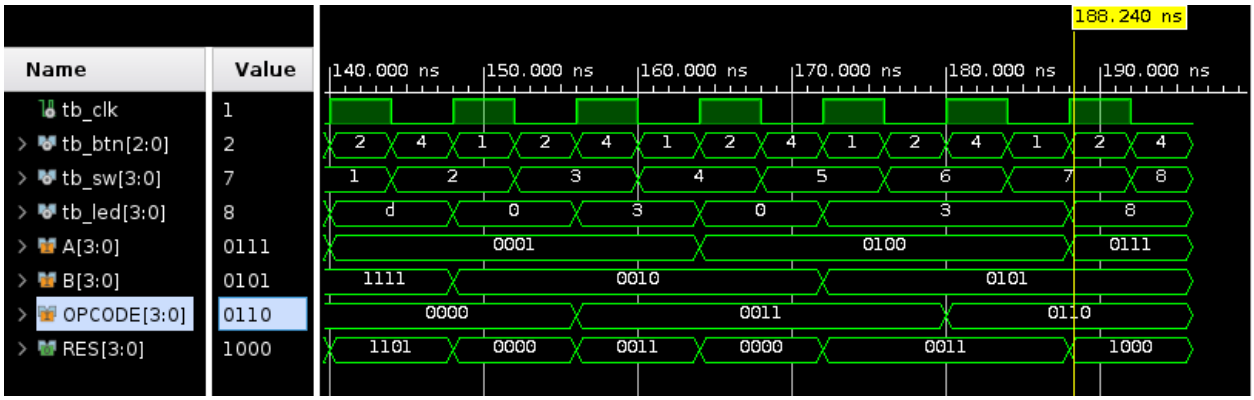
A = 1110
B = 1100
OPCODE = 1101
RES = 1101
Bitwise XNOR of 1110 and 1100 is 1101 as 1 xnor 1 =1 and 0 xnor 0 is 1, all other cases being 0.

6. **A -1**



A = 0001
B = 0010
OPCODE = 0011
RES = 0000
0001 - 1 = 0000

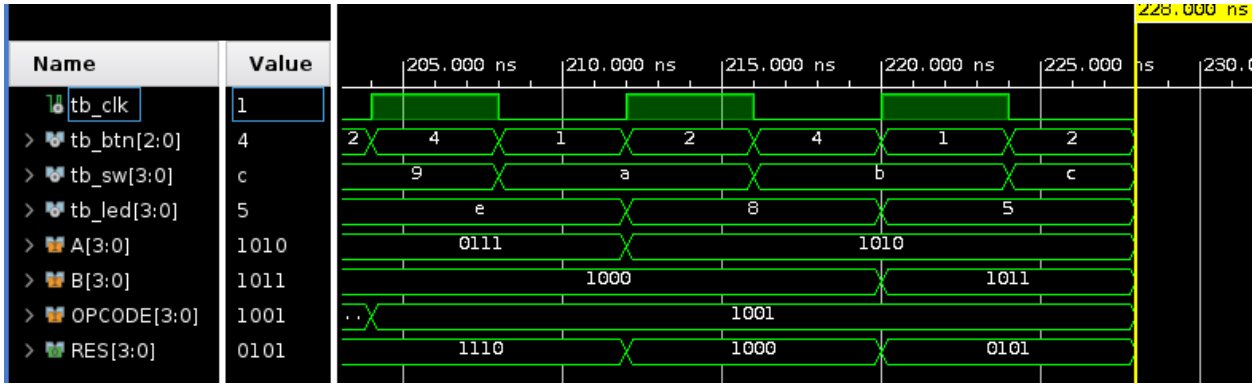
7. **Shift Left**



A = 0100
B = 0101
OPCODE = 0110
RES = 1000

Shifting 0100 to the left by 1 bit and padding a 0 yields 1000

8. **Not A**

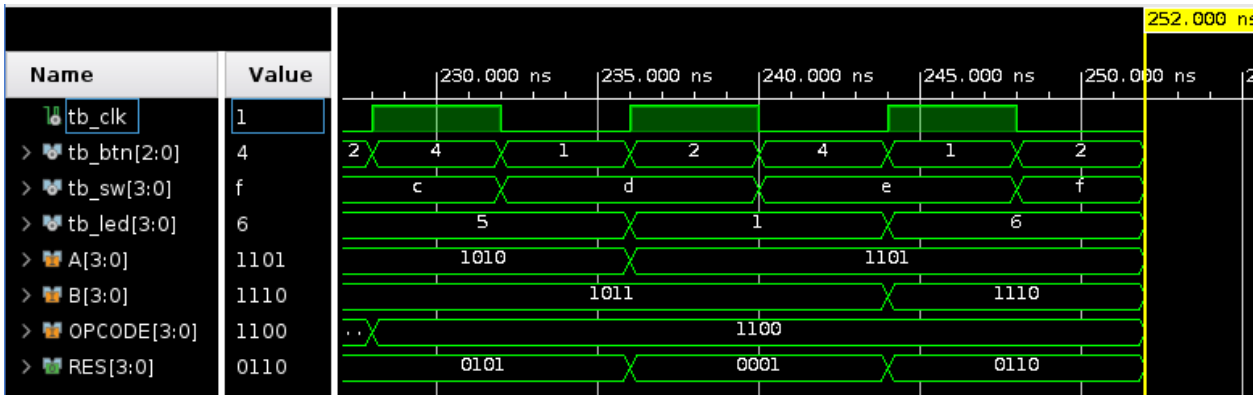


A = 1010
B = 1000

OPCODE = 1001
RES = 0101

Inverting all the bits of A will convert each 0 to a 1 and vice versa.

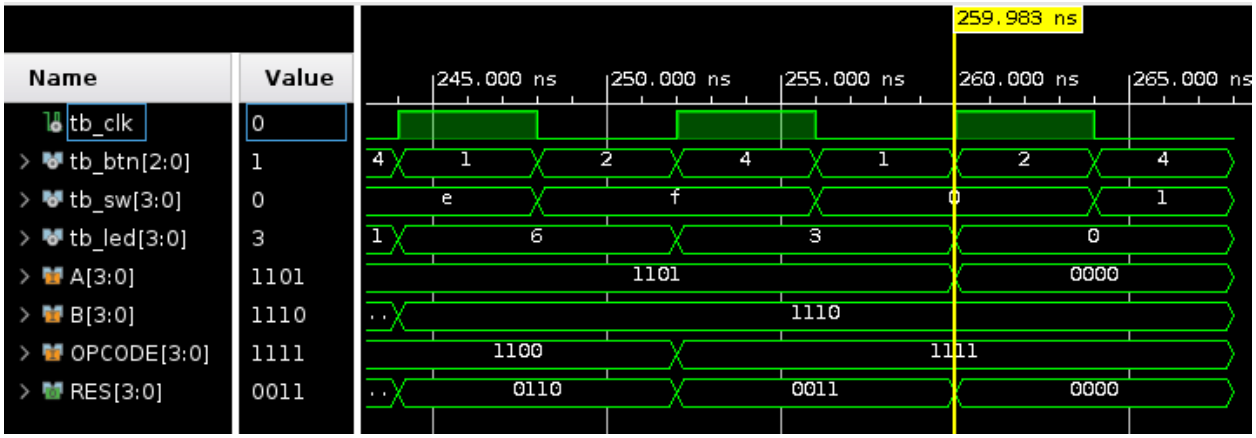
9. **A xnor B**



A = 1101
B = 1011
OPCODE = 1100
RES = 0110

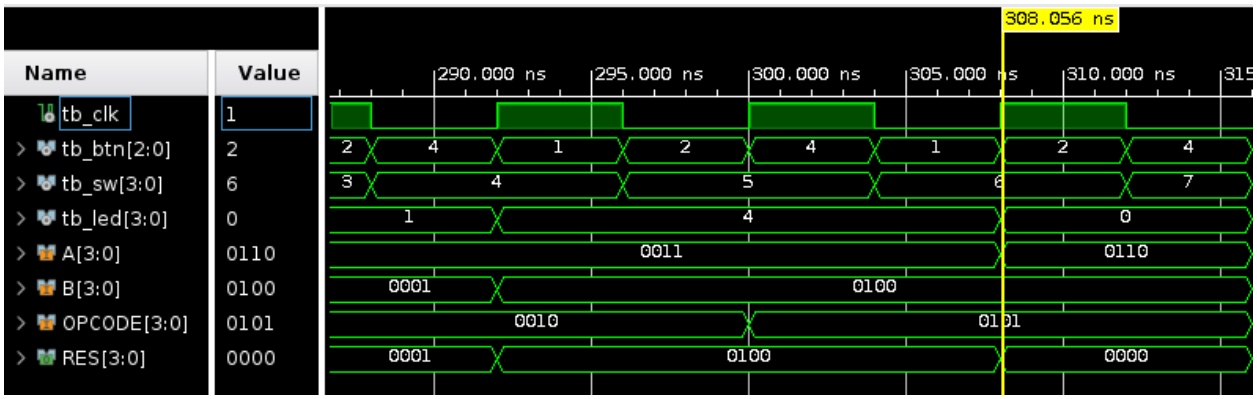
The 1's on the ends will turn to 0s in the output, whereas the alternating 10 and 01 in the middle of each input will be converted to 1's, hence 0110

10. **A nor B**



A = 1101
B = 1110
OPCODE = 1111
RES = 0000
Because there are no indices where both bits are 0 in A or B, all the output bits are 0.

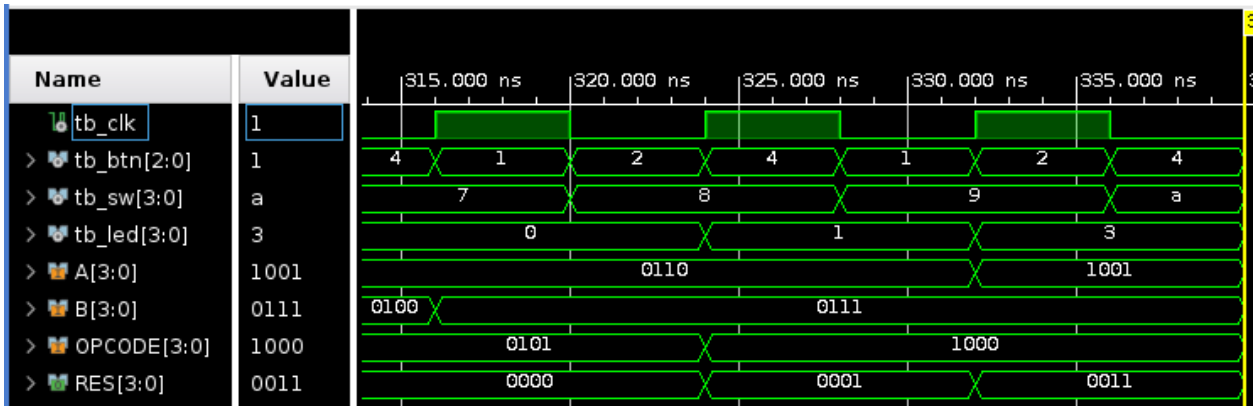
11. A > B



A = 0011
B = 0100
OPCODE = 0101
RES = 0000

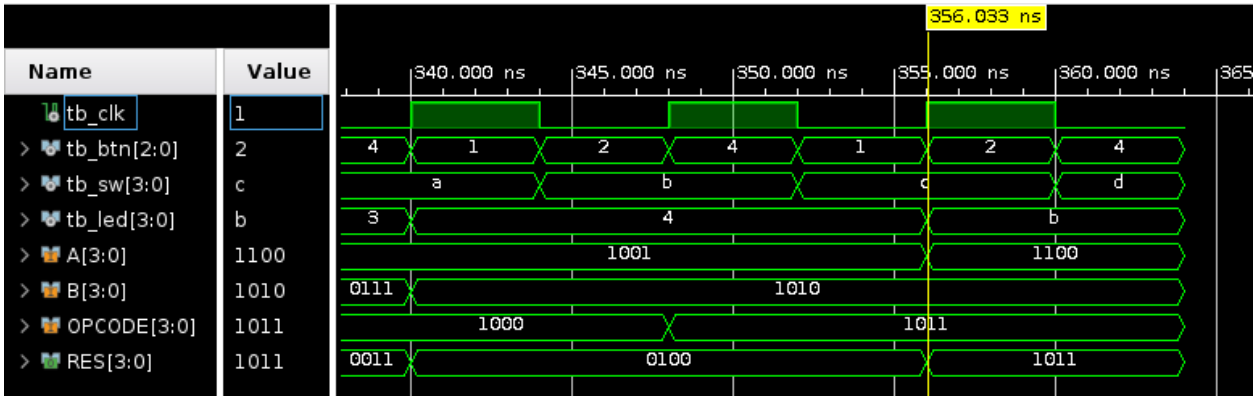
Since A is not greater than B, the output is 0000 rather than 0001

12. Shift right arithmetic



A = 0110
B = 0111
OPCODE = 1000
RES = 0011
The output is 0011, which is both shifted right and the sign of the number is preserved as well.

13. A OR B

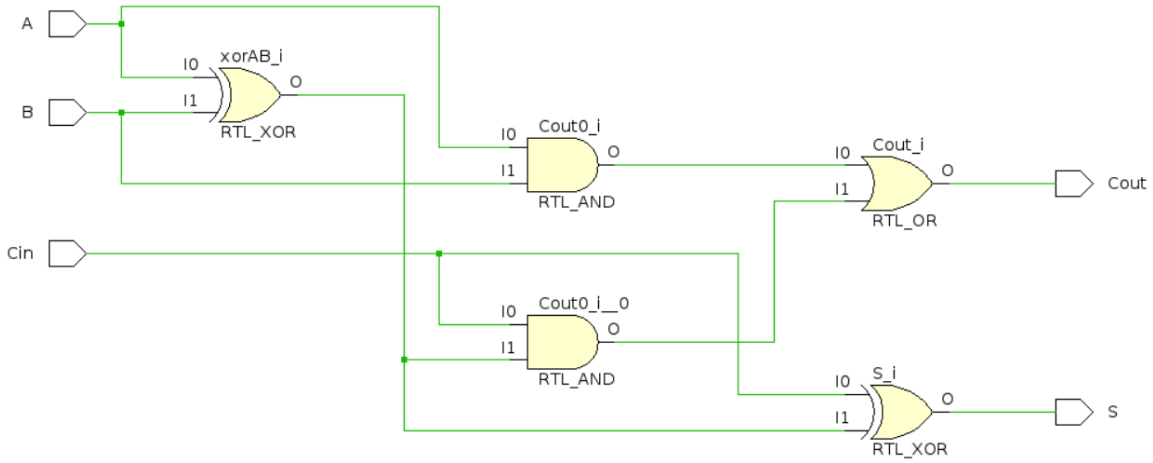


A = 1001
B = 1010
OPCODE = 1011
RES = 1011
1001 OR 1010 = 1011

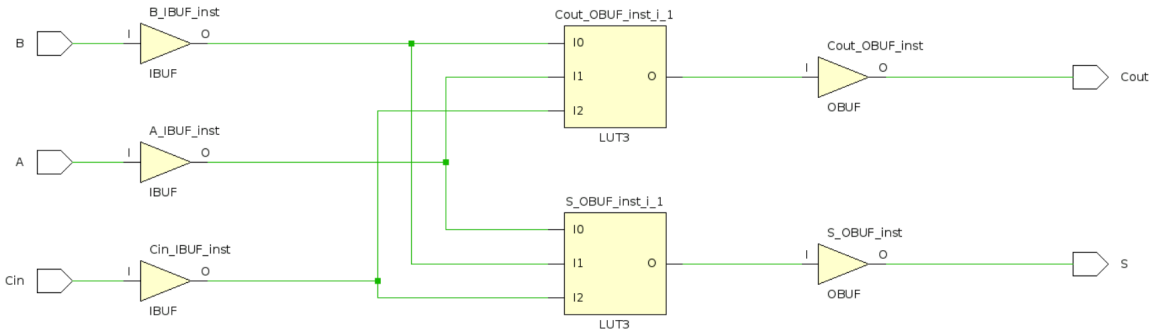
Vivado Schematics:

One-Bit Full Adder

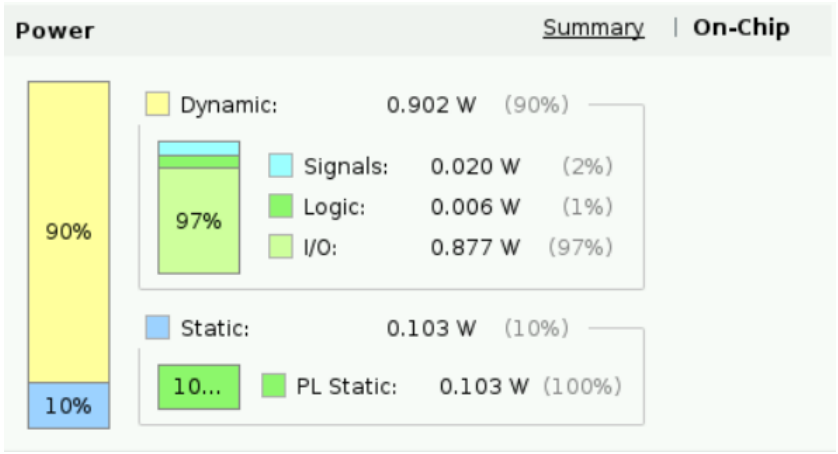
RTL Schematic



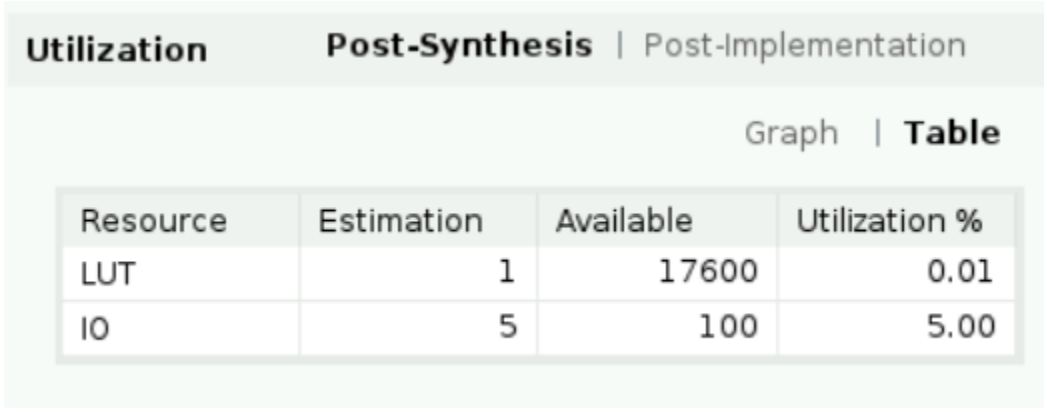
Synthesis Schematic



On-Chip Power Graph



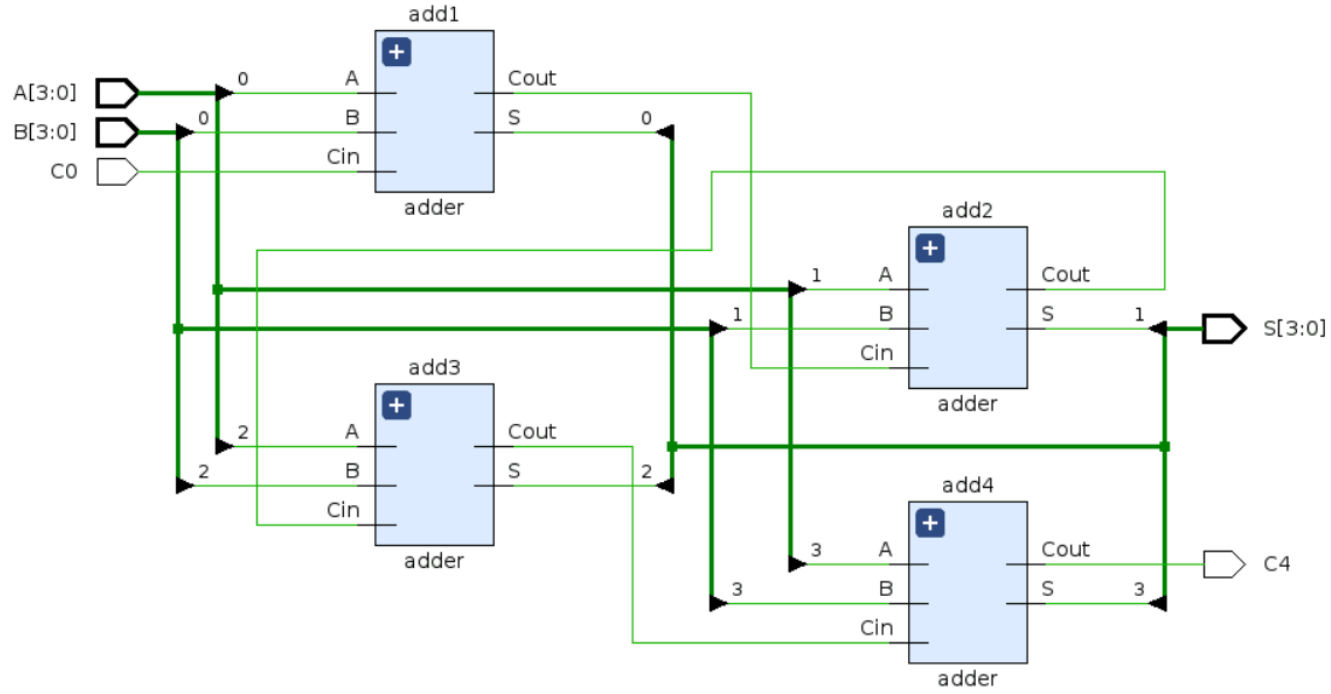
Post Synthesis Utilization



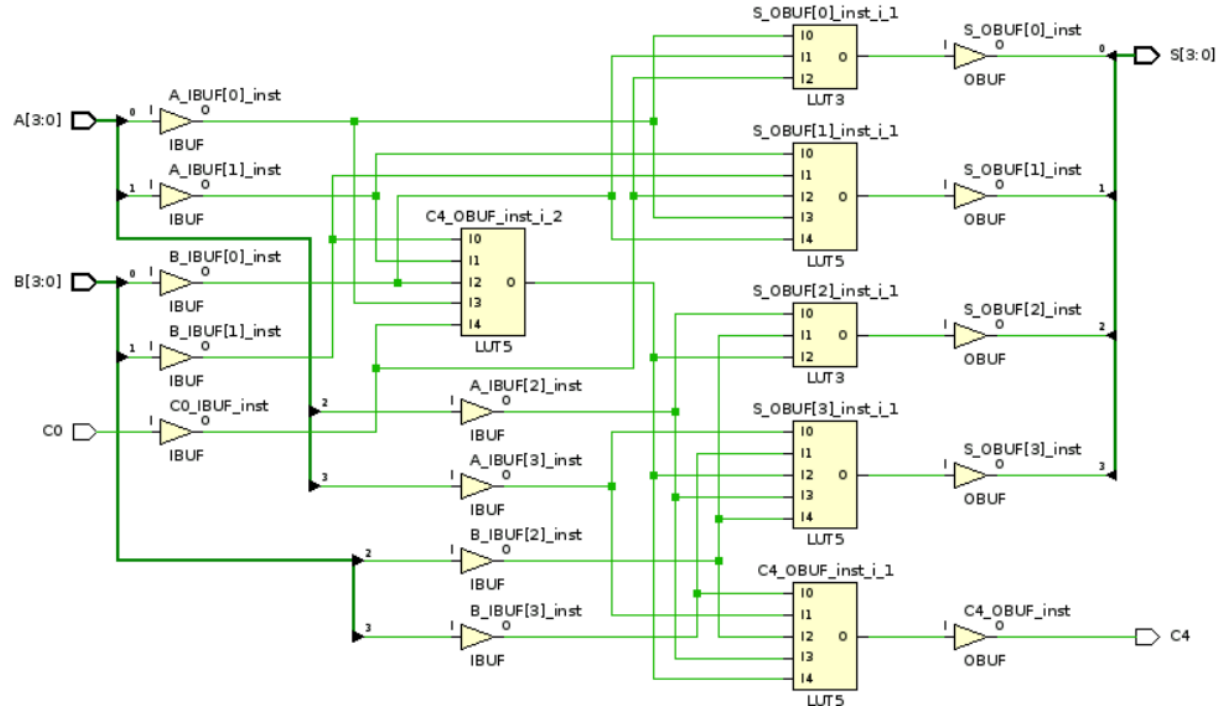
Resource	Estimation	Available	Utilization %
LUT	1	17600	0.01
IO	5	100	5.00

Ripple Adder

Ripple Adder RTL



Ripple Adder Synthesis Schematic



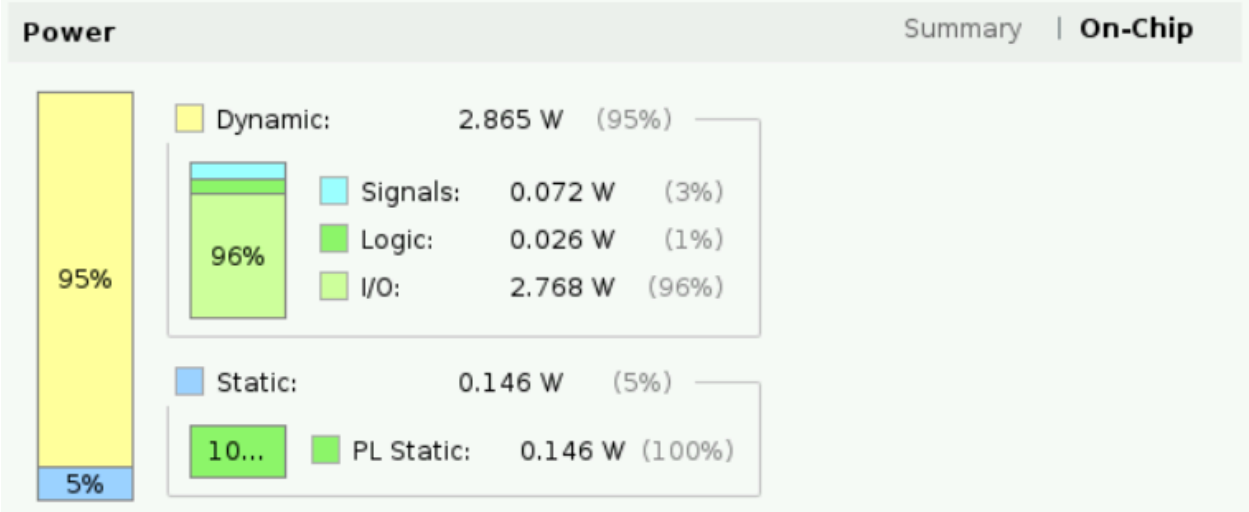
Ripple Adder Utilization Table

Utilization | **Post-Synthesis** | Post-Implementation

Graph | **Table**

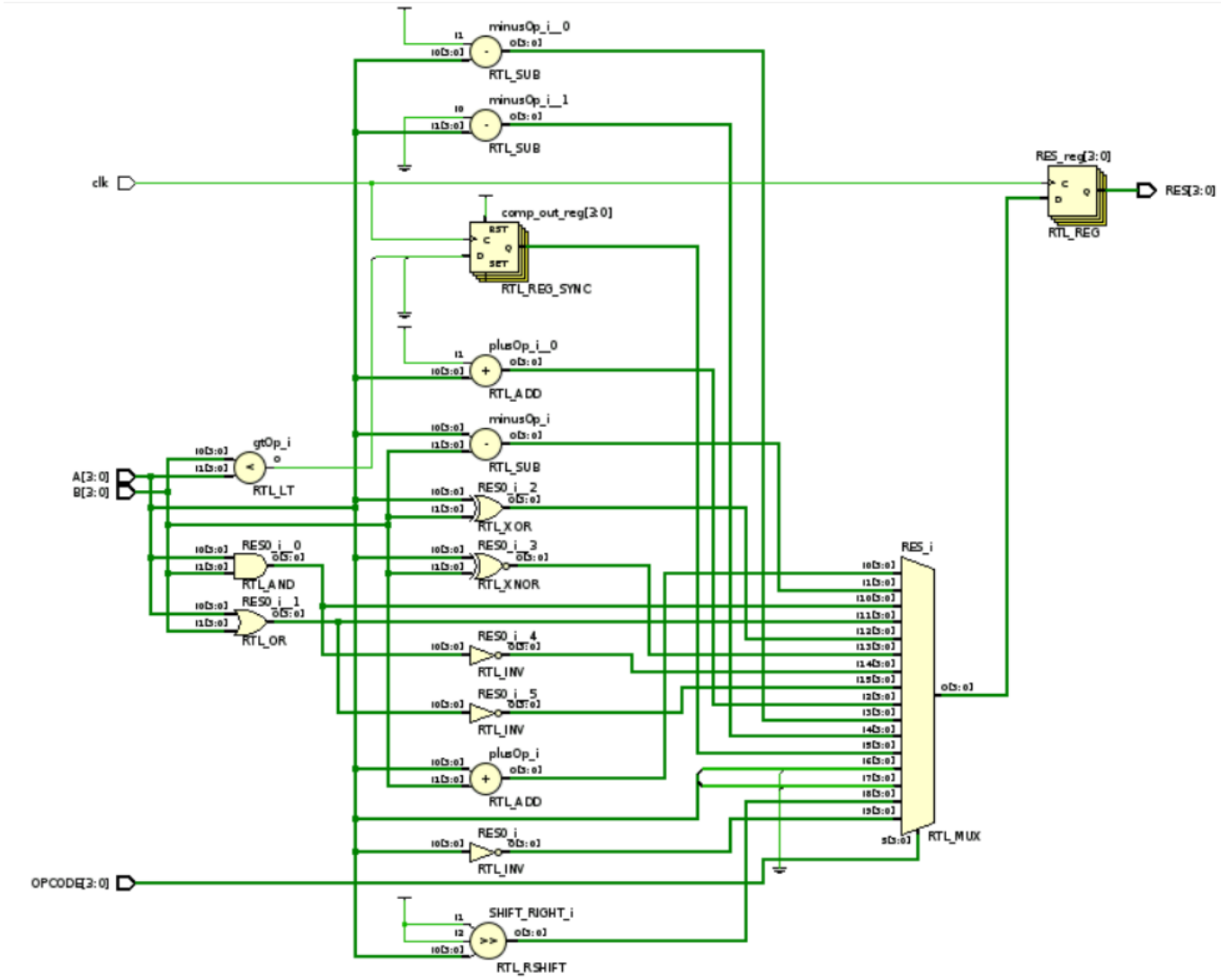
Resource	Estimation	Available	Utilization %
LUT	4	17600	0.02
IO	14	100	14.00

Ripple Adder Power Graph

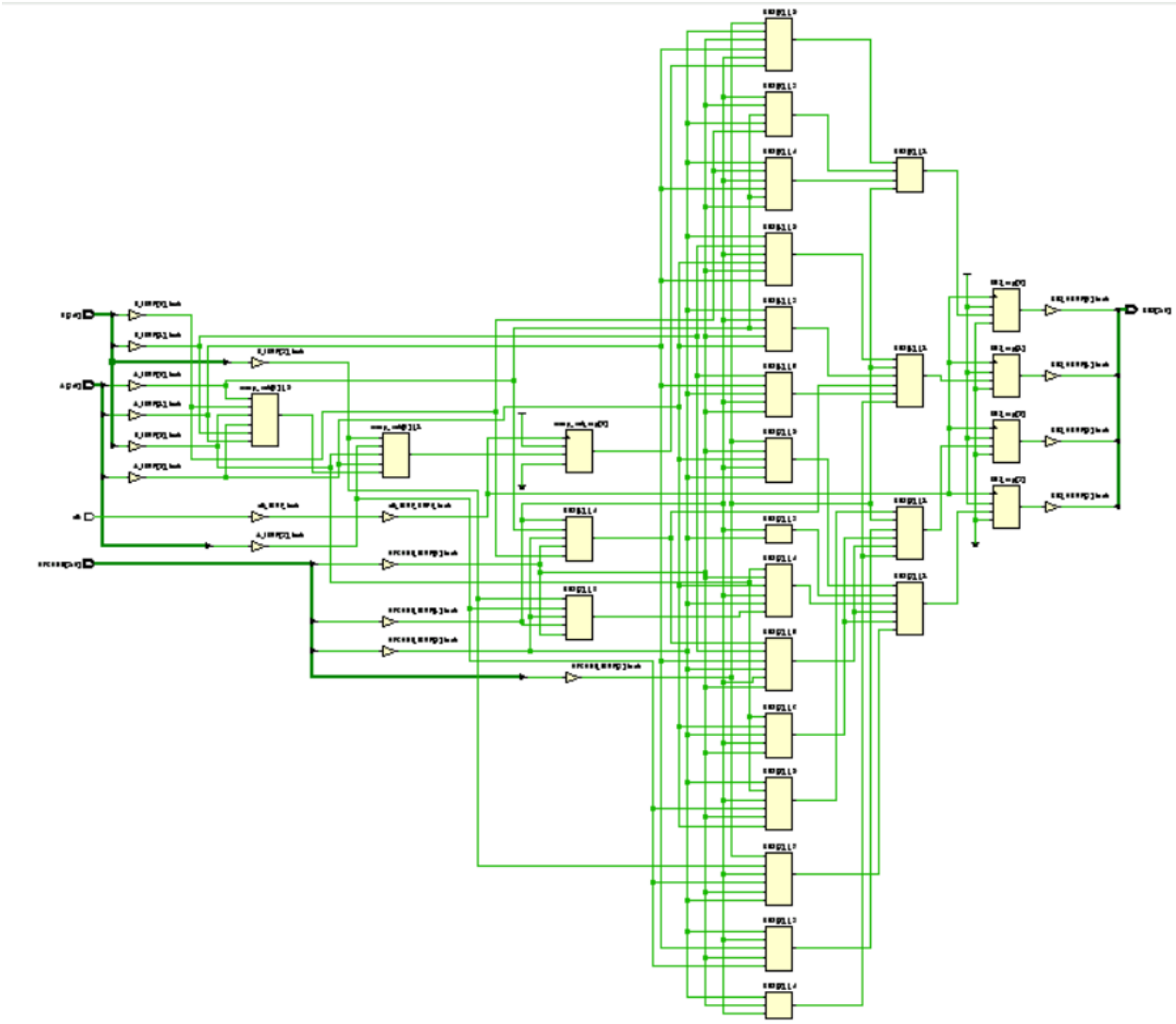


ALU

ALU RTL Schematic



ALU Synthesis Graph



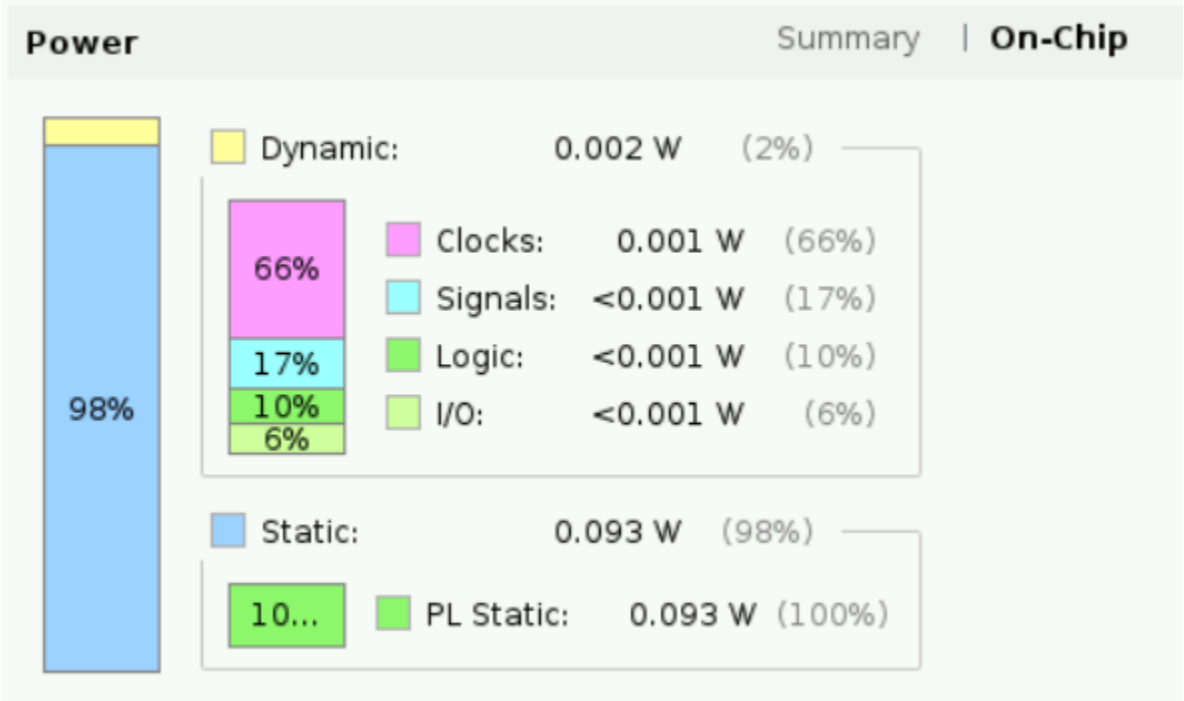
ALU Synthesis Table

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

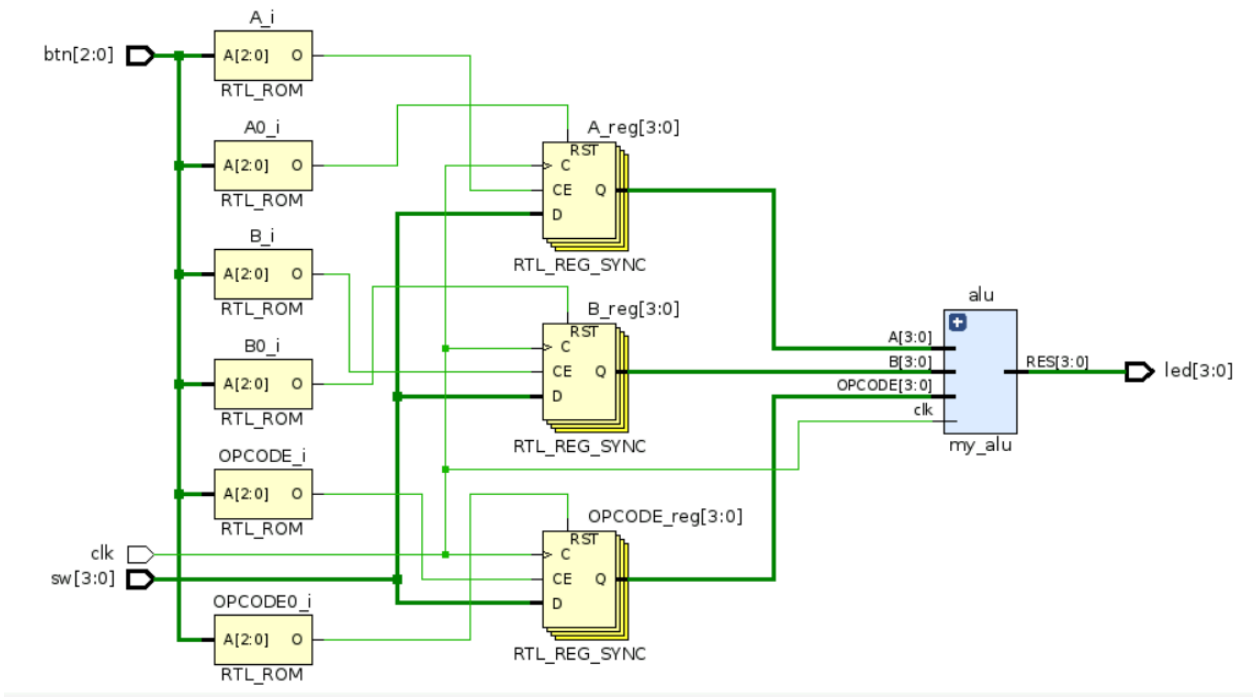
Resource	Estimation	Available	Utilization %
LUT	59	17600	0.34
FF	109	35200	0.31
IO	13	100	13.00
BUFG	1	32	3.13

ALU Power Graph

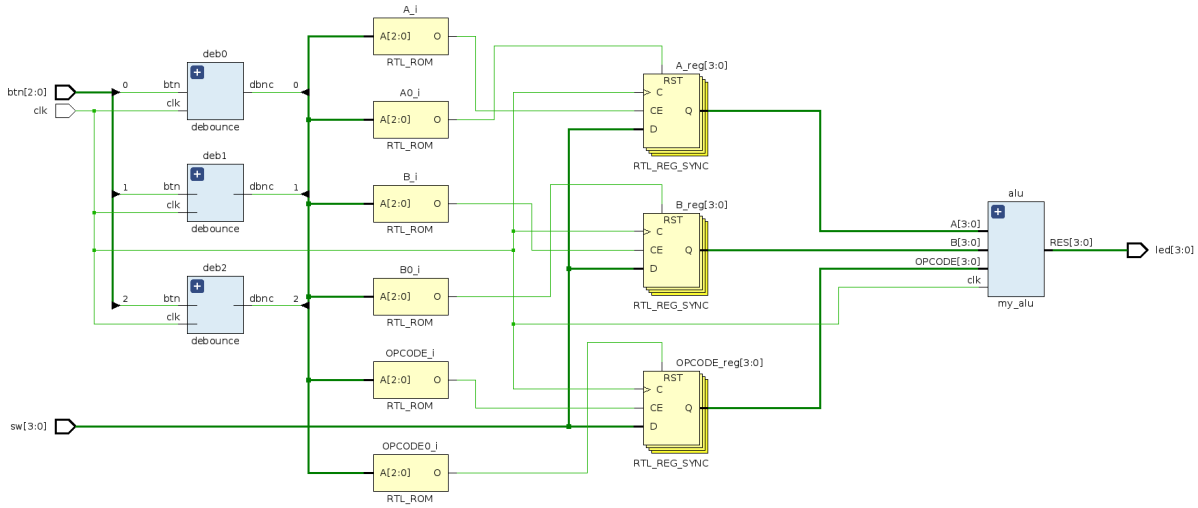


ALU_Tester

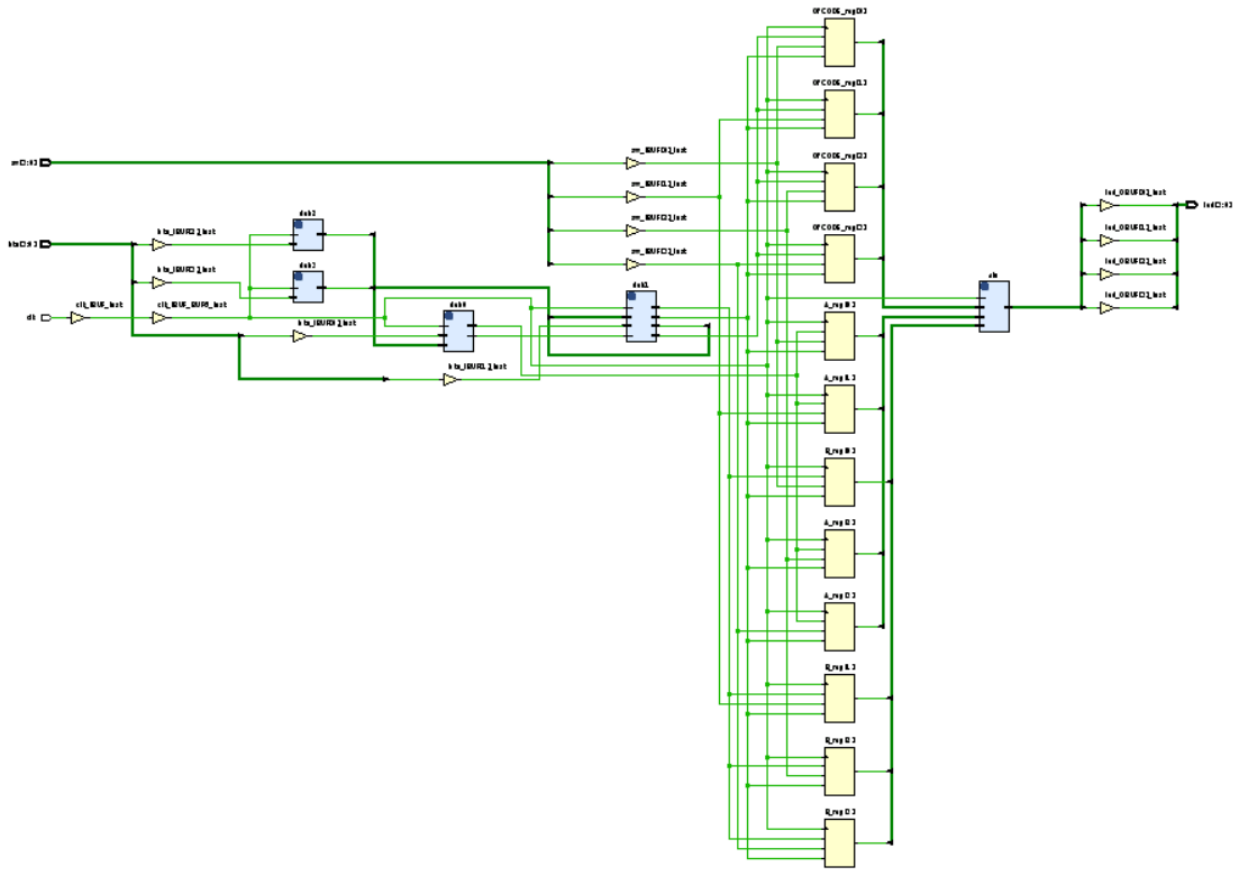
ALU_Tester RTL Used for Simulation, without debouncers



ALU_Tester with Debouncers:



ALU_Tester Synthesis Schematic



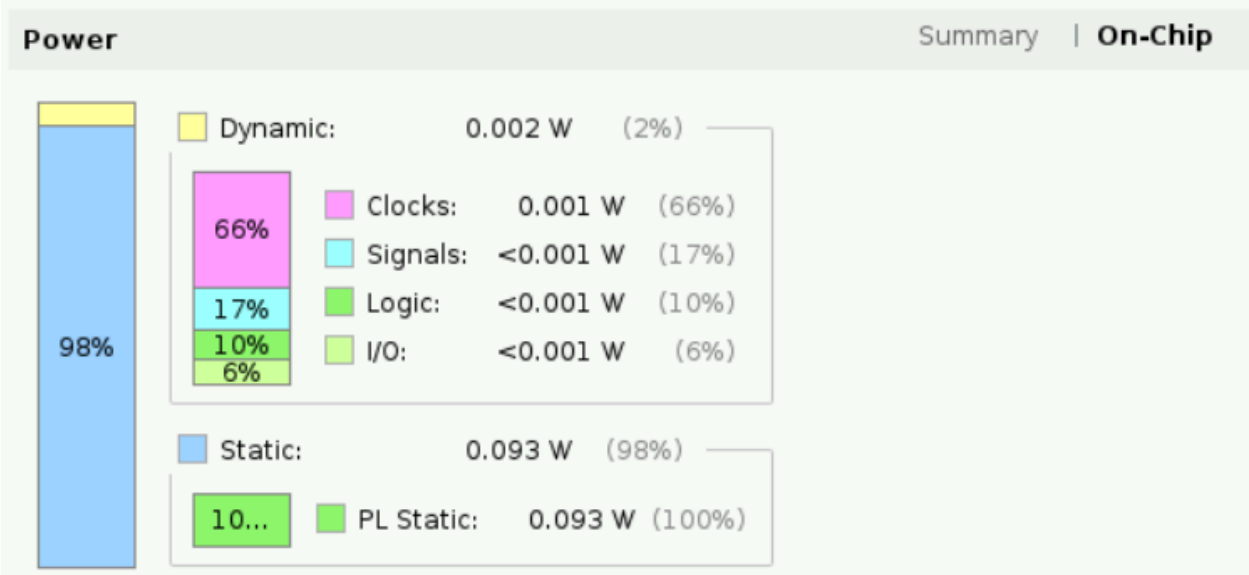
Post Synthesis Utilization Table

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	59	17600	0.34
FF	109	35200	0.31
IO	13	100	13.00
BUFG	1	32	3.13

Post Implementation On Chip Graph



For the XDC file, it was required to uncomment the buttons and switches and clock as inputs, and the LEDs as outputs.

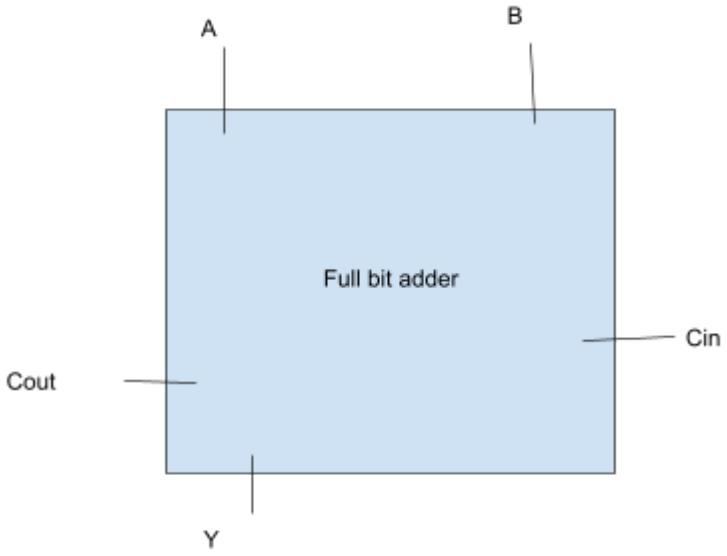
Answers to Additional Questions and Extra Credit:

Pre Lab - Operators and Conversions Review

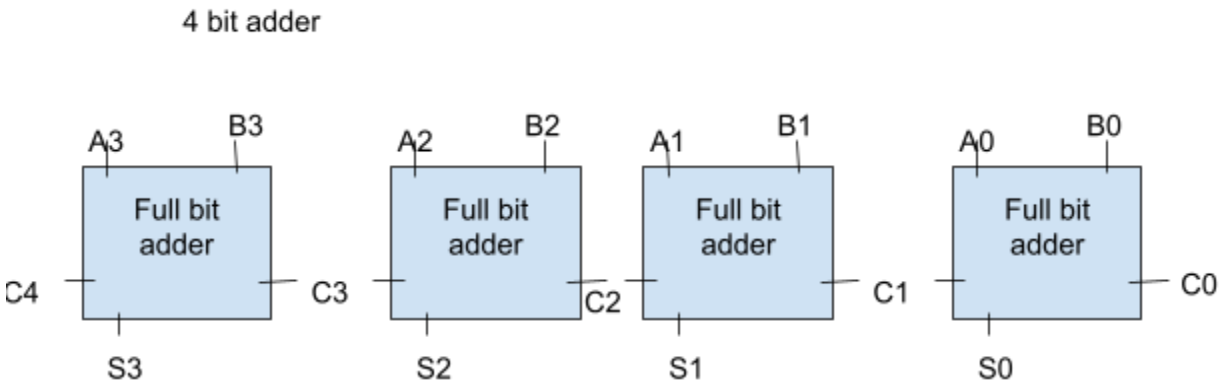
• Write the logic equations for a single bit full adder with inputs A, B, Cin, and outputs Y, Cout.

```
xorAB <=A xor B;
Y <= Cin xor xorAB  ;
Cout <= (A and B) or ( Cin and xorAB );
```

• Draw a black box diagram for the single bit full adder described above.



• Draw a block diagram of a 4-bit ripple-carry adder showing the inner connectivity among the single bit full adders, with 4 bit bus inputs A and B, a single bit input Cin, a 4-bit bus output S, and a single bit output Cout. The ports of the top level diagram are: A, B, S, Cin, Cout.



Conclusion: I learned in this lab that how you code a program influences how state impacts calculations at the level of single clock ticks. In my simulation waveforms, state is only considered from times before the clock tick, not at the time of the clock tick(when state might be updated).
Follow Up: I feel like I understand test benching, but can do better in working with unsigned and signed calculation.