# Computer Organization and Architecture Laboratory

Assignment 3

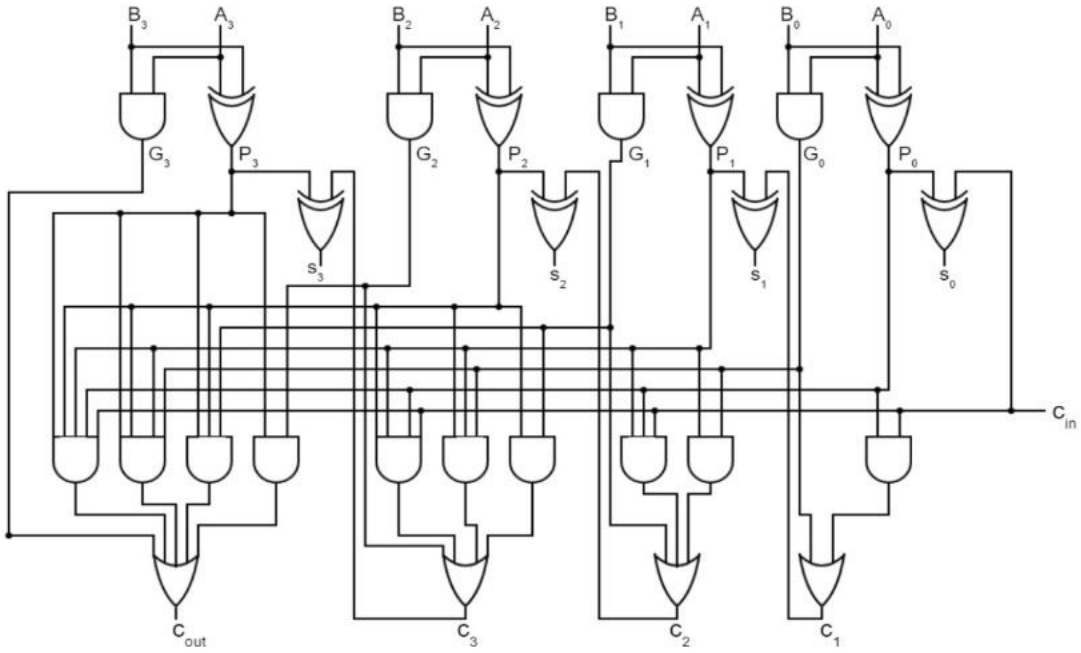
**Group 22**

Nikhil Saraswat(20CS10039)

Amit Kumar (20CS30003)

# PART 2: Carry Look-ahead Adders

## 1. 4-bit carry look-ahead adder




The carry lookahead adder reduces the delay in calculating the sum by calculating the carries concurrently rather than waiting for the carry from the previous block to ripple, as is the case with the ripple carry adder.
The carry look-ahead adder's logic is as follows:

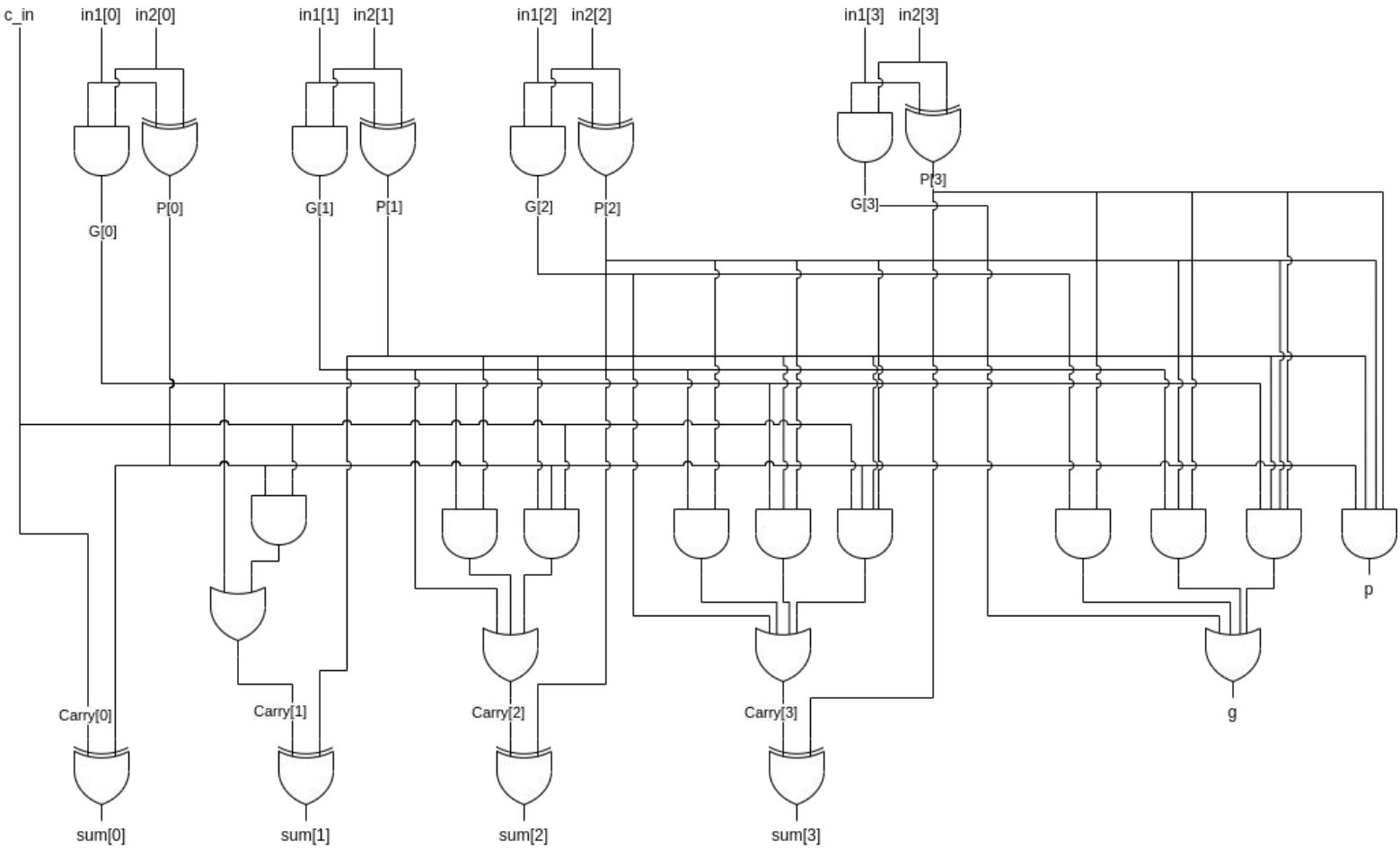
```
For -1 < i < 4 ,

G[i]=in1[i]&in2[i]
P[i] = in1[i] ^ in2[i]

Take c_in to be carry[0] then:

For -1 < i < 4 ,
sum[i] = P[i] ^ carry[i]
carry[i] = G[i-1] | (P[i-1] & carry[i-1])
```

```
After Recursively expanding :

carry[1] = G[0] | (P[0] & carry[0]) = G[0] | (P[0] &  c_in)

carry[2] = G[1] | (P[1] & carry[1]) = G[1] | (P[1] & G[0]) | (P[1] &
P[0] & c_in)

carry[3] = G[2] | (P[2] & carry[2]) = G[2] | (P[2] & G[1]) | (P[2] &
P[1] & G[0]) | (P[2] & P[1] & P[0] &  c_in)

carry[4] = G[3] | (P[3] & carry[3]) = G[3] | (P[3] & G[2]) | (P[3]  &
P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1]  &
P[0] & c_in)
```

## 2. 4-bit carry look-ahead adder (augmented)

Instead of generating the carry out, we give the block propagate and generate as output, which is then used by the carry lookahead unit.
This results in a modular design that allows us to create 16, 32, and 64 bit adders by combining the block propagate and generate from lower levels rather than rippling the carry out every time.
The other logic is the same as in a standard 4-bit CLA.
The propagation and generation of blocks are calculated as follows:

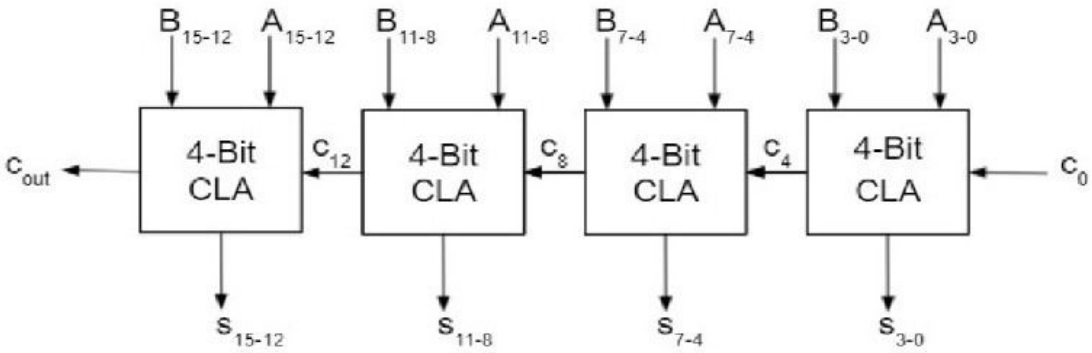
```
p = P[3] & P[2] & P[1] & P[0]
g = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1]
& G[0])
```


c_in
in1[0]
in2[0]
in1[1]
in2[1]
in1[2]
in2[2]
in1[3]
in2[3]
G[0]
P[0]
G[1]
P[1]
G[2]
P[2]
G[3]
P[3]
Carry[0]
Carry[1]
Carry[2]
Carry[3]
g
p
sum[0]
sum[1]
sum[2]
sum[3]

## 3. 16-bit carry look-ahead adder (ripple carry out)

This 16-bit adder is constructed by cascading four 4-bit CLAs and rippling the carry out from one block to the next, as illustrated in the figure.

Here, carry[2:0] are internal carries being rippled from one block to another.




## 4. 16-bit carry look-ahead adder (look-ahead carry unit)

Let P[3:0] and G[3:0] represent the block propagate and generate of the four 4-bit CLAs, respectively.
Instead of rippling the carry out from one block to the next, we can reduce the delay in the circuit by adding an additional lookahead unit that calculates these carries simultaneously.
This eliminates the need for subsequent blocks to wait for the previous block's carry.
This also leads to a modular design in which we can calculate the block propagation and generation of the entire 16-bit CLA and use it later for higher order adders.

The logic for the lookahead carry unit is as follows:

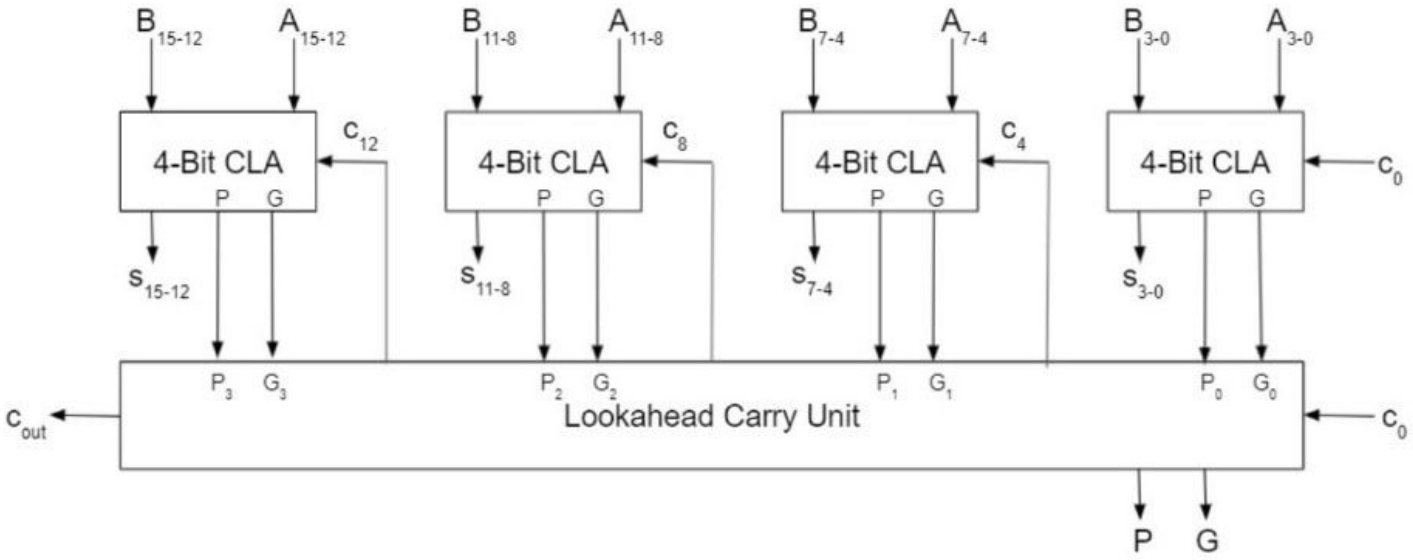
```
Take c_in to be carry[0] then
carry[i] = G[i-1] | (P[i-1] & carry[i-1]), 1 <= i <=  4
```

After Recursively expanding

```
carry[1] = G[0] | (P[0] & carry[0]) = G[0] | (P[0] &  c_in)

carry[2] = G[1] | (P[1] & carry[1]) = G[1] | (P[1] & G[0]) | (P[1] &
P[0] & c_in)

carry[3] = G[2] | (P[2] & carry[2]) = G[2] | (P[2] & G[1]) | (P[2] &
P[1] & G[0]) | (P[2] & P[1] & P[0] &  c_in)

carry[4] = G[3] | (P[3] & carry[3]) = G[3] | (P[3] & G[2]) | (P[3]  &
P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1]  &
P[0] & c_in)
```

Block propagate p and generate g are calculated as:

```
p = P[3] & P[2] & P[1] & P[0]
g = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1]
& G[0])
```



# SYNTHESIS SUMMARY

| | 16-bit carry look-ahead adder (ripple carry out) | 16-bit carry look-ahead adder (look-ahead carry unit) |
|---|---|---|
| Delay (in ns) | 6.200 | 5.013 |
| Number of Slice LUTs | 25 out of 63400 | 44 out of 63400 |
| Number of bonded IOBs | 50 out of 210 | 52 out of 210 |
| Levels of Logic | 11 | 8 |

A CLA with an additional lookahead carry unit outperforms simply rippling the carry at the expense of more LUTs.

- Comparison with 4-bit RCA

| 4-bit | Delay (in ns) | Number of Slice LUTs | Logic Levels |
|---|---|---|---|
| CLA | 2.123 | 6 out of 63400 | 4 |
| RCA | 3.194 | 8 out of 63400 | 10 |

- Comparison with 16-bit RCA

It is clear that a CLA is significantly faster than an RCA. In addition, the LUTs used are nearly half that of an RCA.

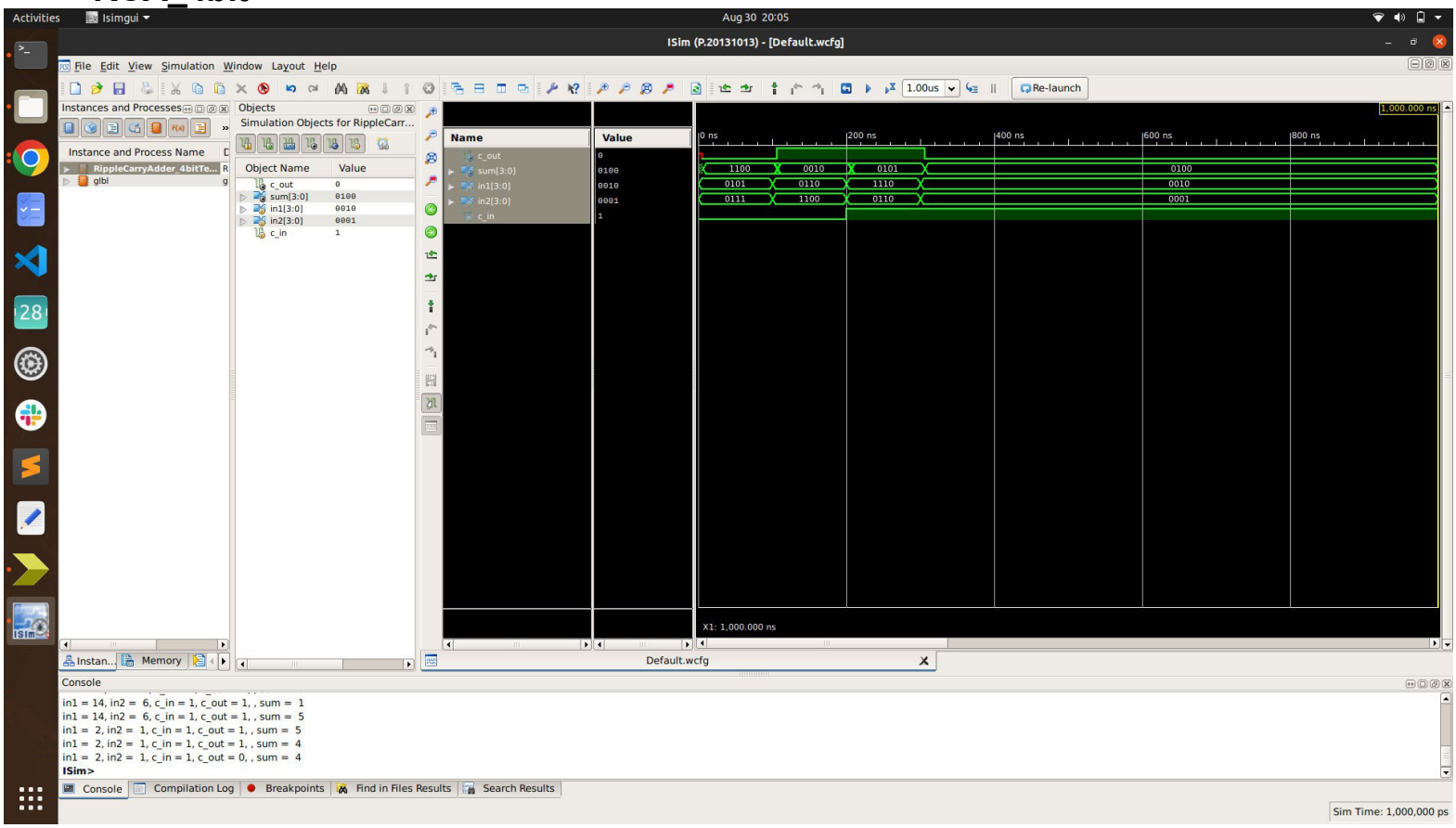
| 16-bit | Delay (in ns) | Number of Slice LUTs | Logic Levels |
|---|---|---|---|
| CLA | 3.768 | 44 out of 63400 | 8 |
| RCA | 6.167 | 24 out of 63400 | 10 |

# Post Route Simulation Screenshots:

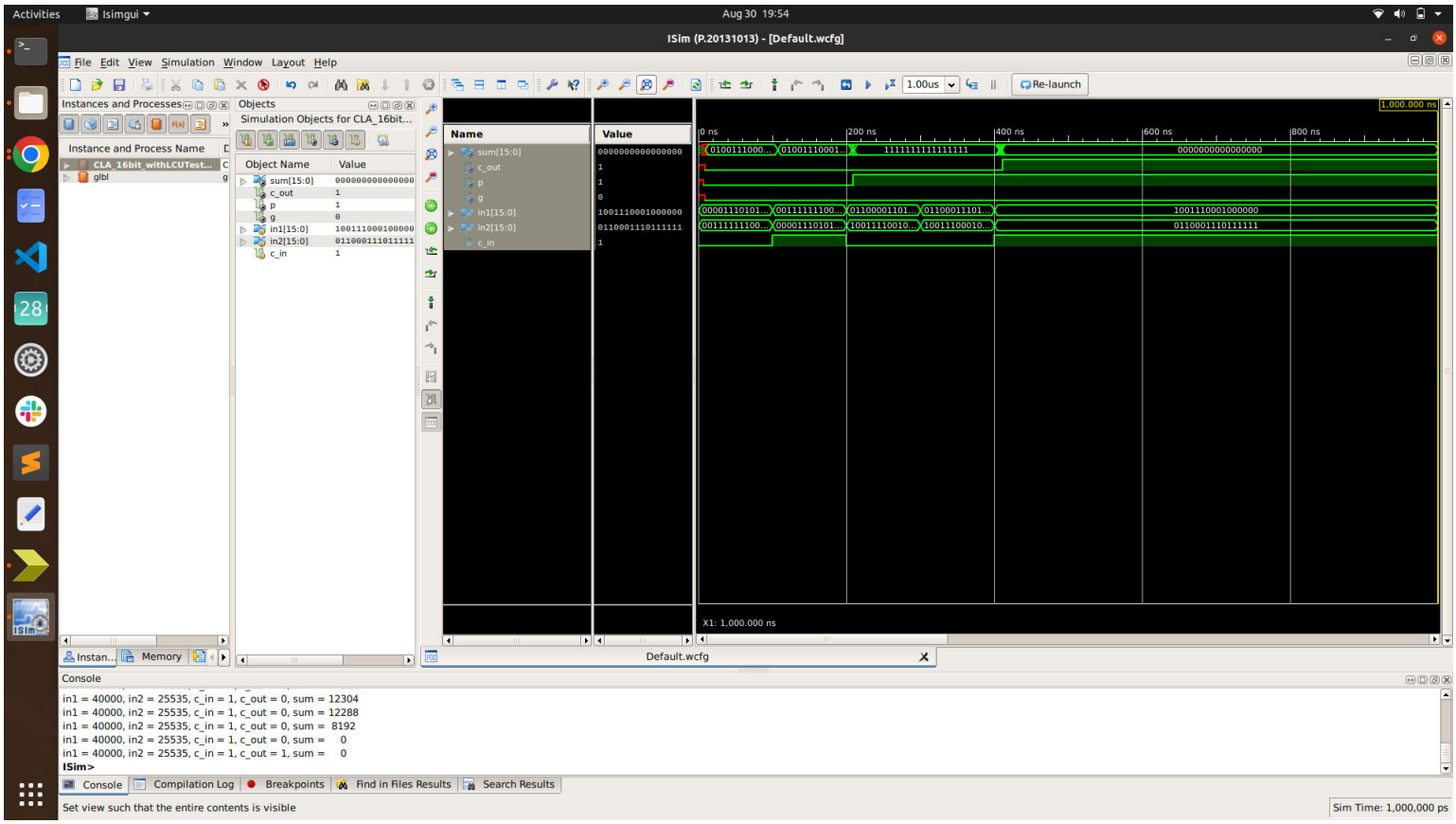
- **wrapper**

| | Met | Constraint | Check | Worst Case Slack | **Best Case Achievable** | Timing Errors | Timing Score |
|---|---|---|---|---|---|---|---|
| **1** | Yes | Autotimespec constraint for clock net clk_BUFGP | SETUP<br>HOLD | 0.202ns | 3.768ns | 0 | 0<br>0 |

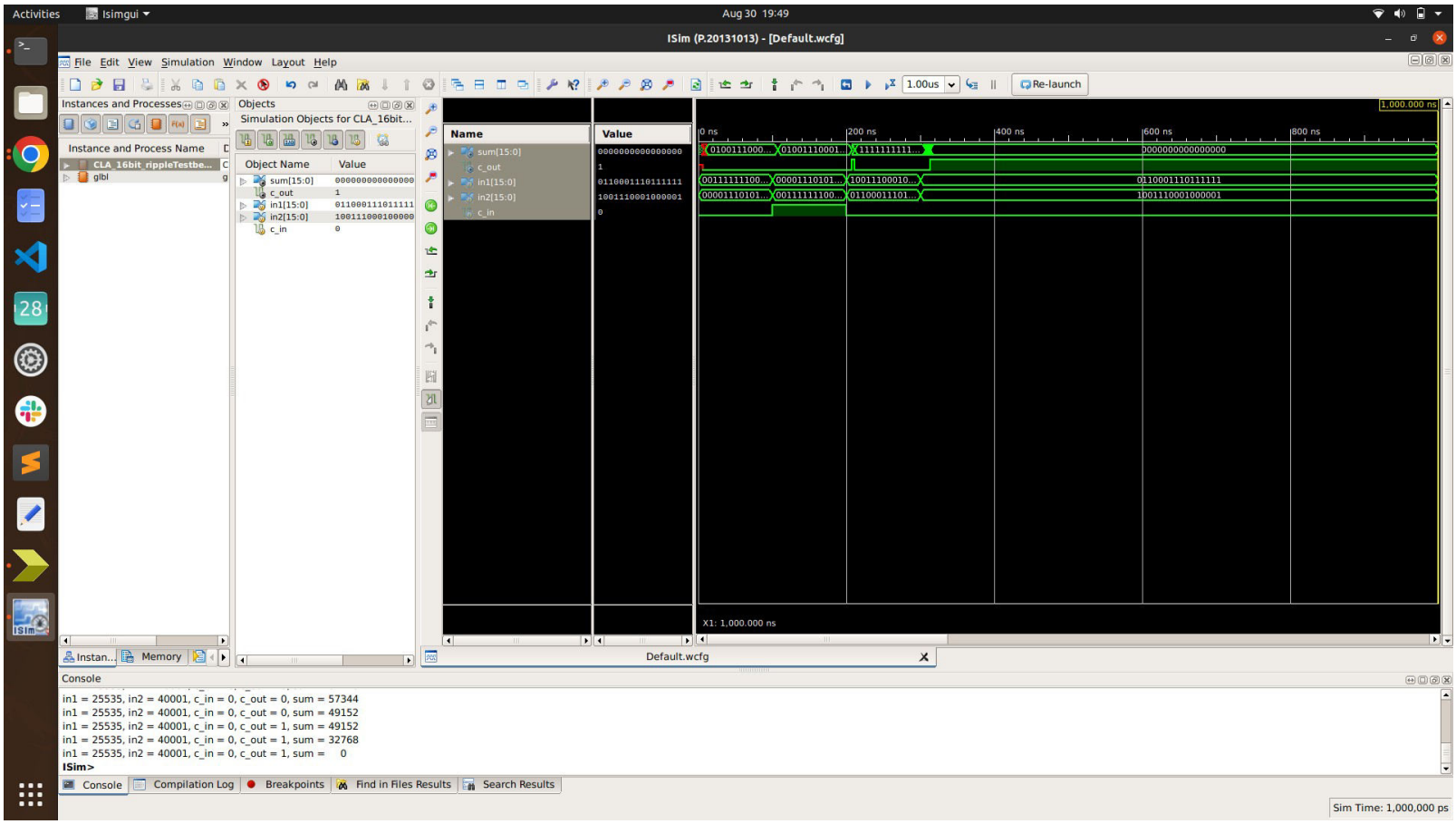
- **RCA_4bit**



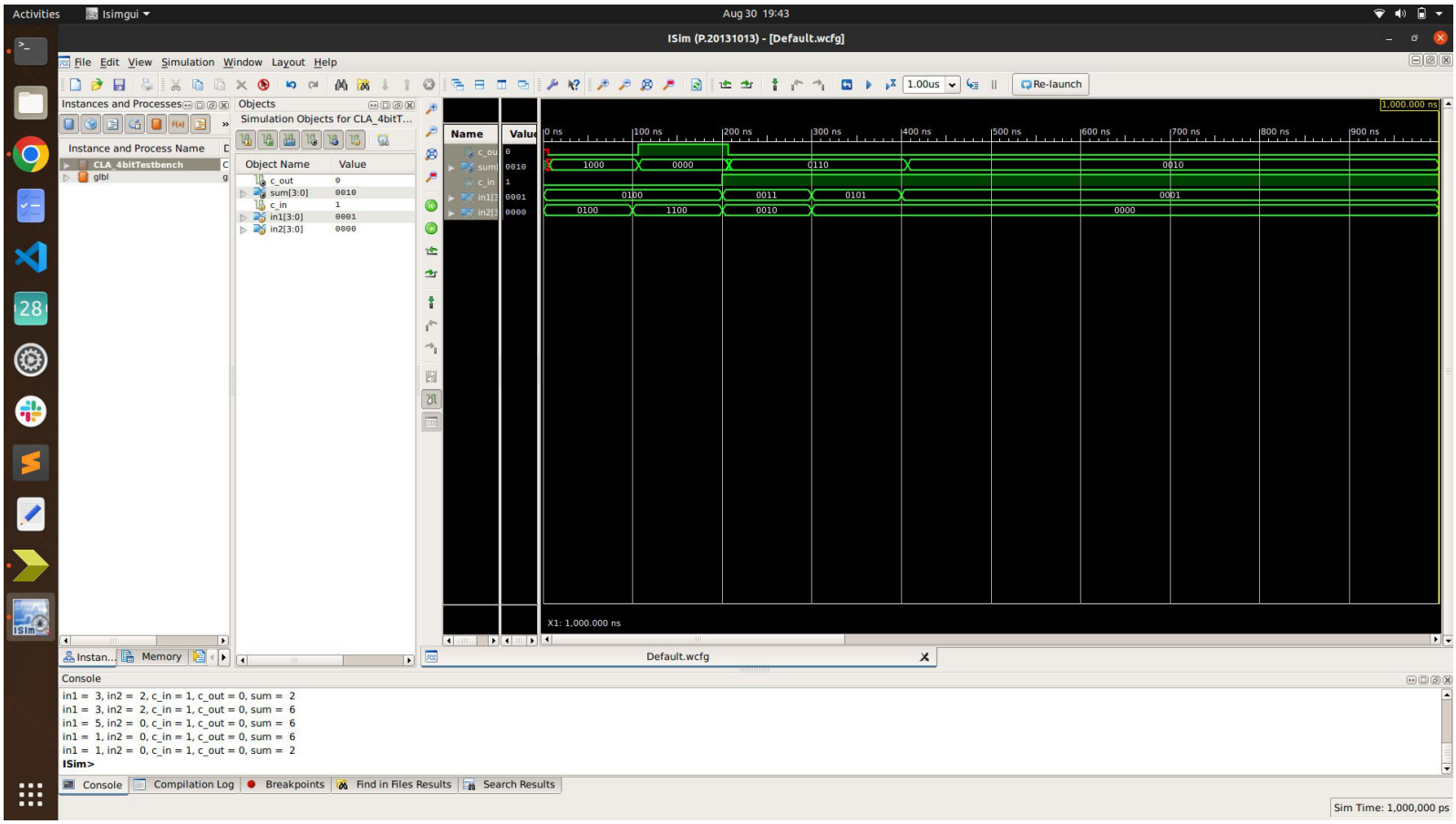

- **CLA_16bit_with LCU**




- **LCA_16bit_ripple**


Activities
Isimgui
Aug 30 19:49
ISim (P.20131013) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
Instances and Processes
Objects
Simulation Objects for CLA_16bit...
Instance and Process Name
CLA_16bit_rippleTestbe...
glbl
Object Name
Value
sum[15:0]
c_out
in1[15:0]
in2[15:0]
c_in
Name
Value
1,000.000 ns
0 ns
200 ns
400 ns
600 ns
800 ns
X1: 1,000.000 ns
Default.wcfg
Console
in1 = 25535, in2 = 40001, c_in = 0, c_out = 0, sum = 57344
in1 = 25535, in2 = 40001, c_in = 0, c_out = 0, sum = 49152
in1 = 25535, in2 = 40001, c_in = 0, c_out = 1, sum = 49152
in1 = 25535, in2 = 40001, c_in = 0, c_out = 1, sum = 32768
in1 = 25535, in2 = 40001, c_in = 0, c_out = 1, sum = 0
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Sim Time: 1,000,000 ps

- **4-bit-CLA**


Activities
Isimgui
Aug 30 19:43
ISim (P.20131013) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Re-launch
1.00us
Instances and Processes
Instance and Process Name
CLA_4bitTestbench
glbl
Objects
Simulation Objects for CLA_4bitT...
Object Name
Value
c_out 0
sum[3:0] 0010
c_in 1
in1[3:0] 0001
in2[3:0] 0000
Name
Value
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000.000 ns
1000
0000
0110
0010
0100
0011
0101
0001
0100
1100
0010
0000
X1: 1,000.000 ns
Instan...
Memory
Default.wcfg
Console
in1 = 3, in2 = 2, c_in = 1, c_out = 0, sum = 2
in1 = 3, in2 = 2, c_in = 1, c_out = 0, sum = 6
in1 = 5, in2 = 0, c_in = 1, c_out = 0, sum = 6
in1 = 1, in2 = 0, c_in = 1, c_out = 0, sum = 6
in1 = 1, in2 = 0, c_in = 1, c_out = 0, sum = 2
ISim>
Console
Compilation Log
Breakpoints
Find in Files Results
Search Results
Sim Time: 1,000,000 ps