Computer Organization and Architecture Laboratory

Assignment 1

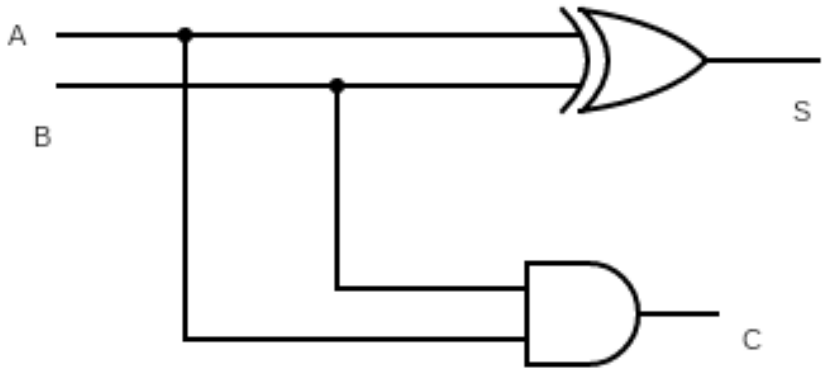
Group 1

Animesh Jha (19CS10070)

Nisarg Upadhyaya (19CS30031)

PART 1: Ripple Carry Adders

1. Half adder

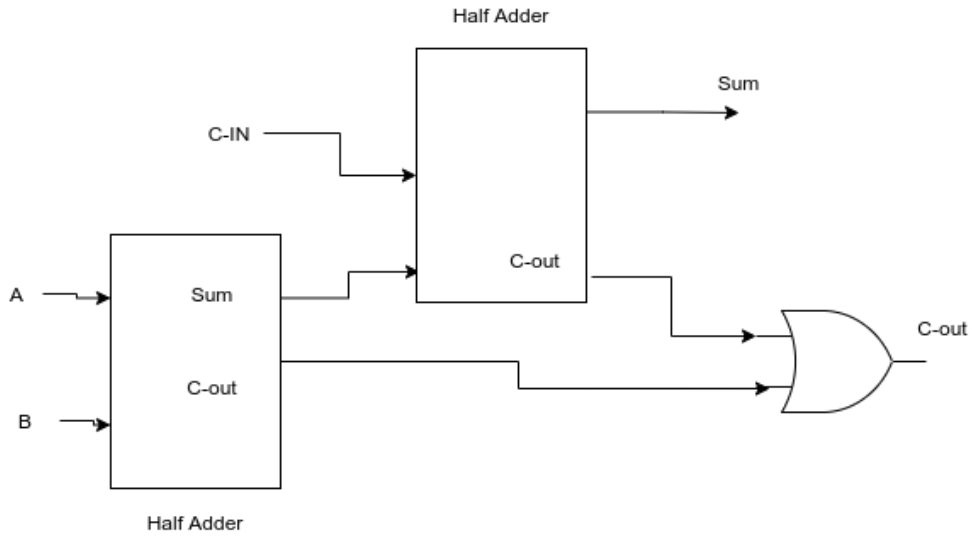


Inputs		Outputs	
A	B	Sum	C_out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logic:

```
c_out = a & b
 sum = a ^ b
```

2. Full adder

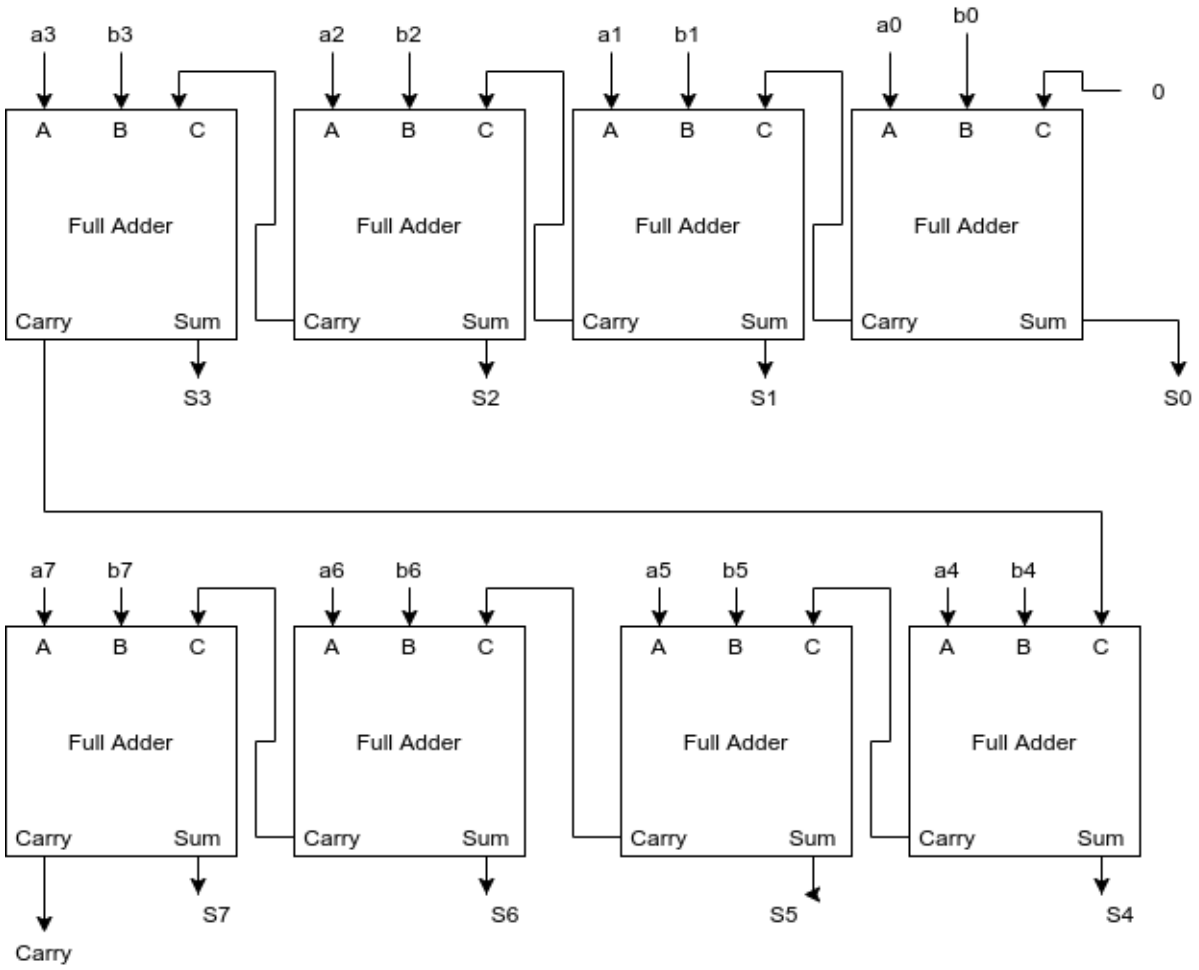


Inputs			Outputs	
A	B	C_in	Sum	C_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

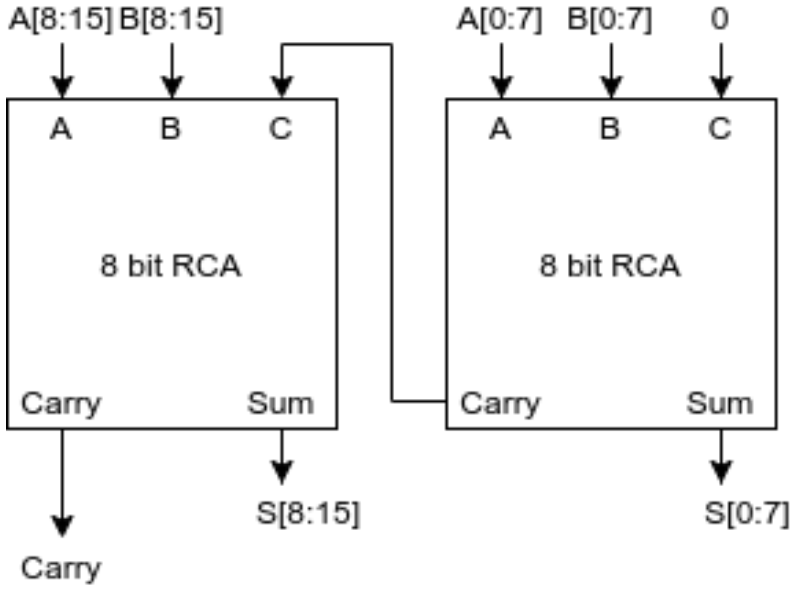
Logic:

```
sum = a ^ b ^ c_in
c_out = (a & b) | (c & (a ^ b))
```

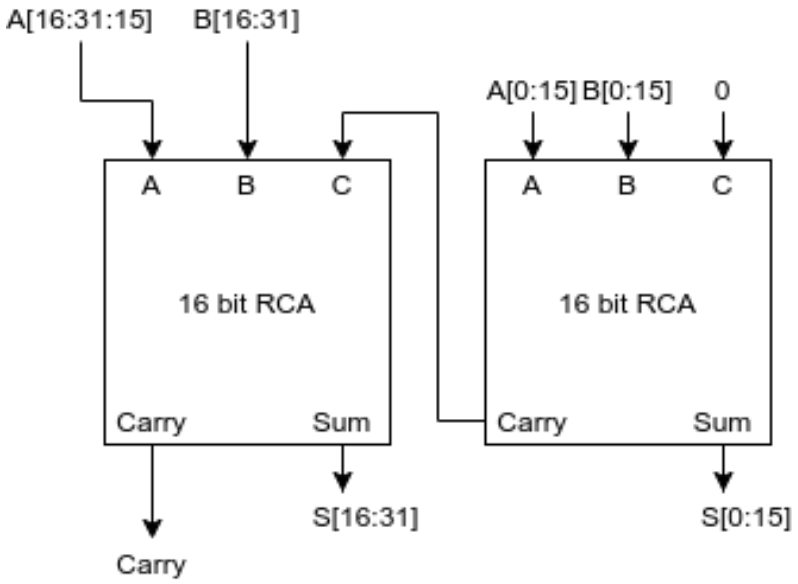
3. 8-bit adder



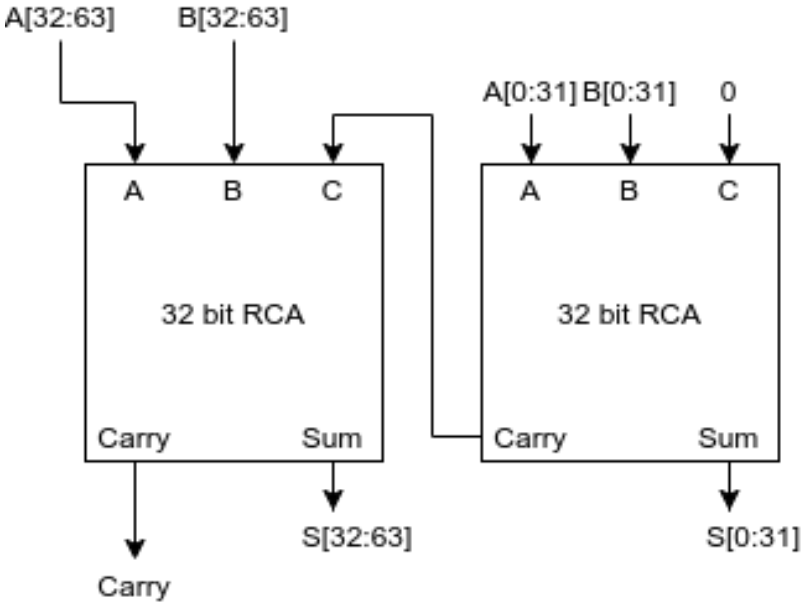
4. 16-bit adder



5. 32-bit adder



6. 64-bit adder



SYNTHESIS SUMMARY

Circuit	Delay (in ns)	Logic Levels	Number of Slice LUTs	Number of bonded IOBs
8-bit RCA	9.949	36	40	26
16-bit RCA	18.717	70	80	50
32-bit RCA	36.253	138	160	98
64-bit RCA	71.325	274	320	194

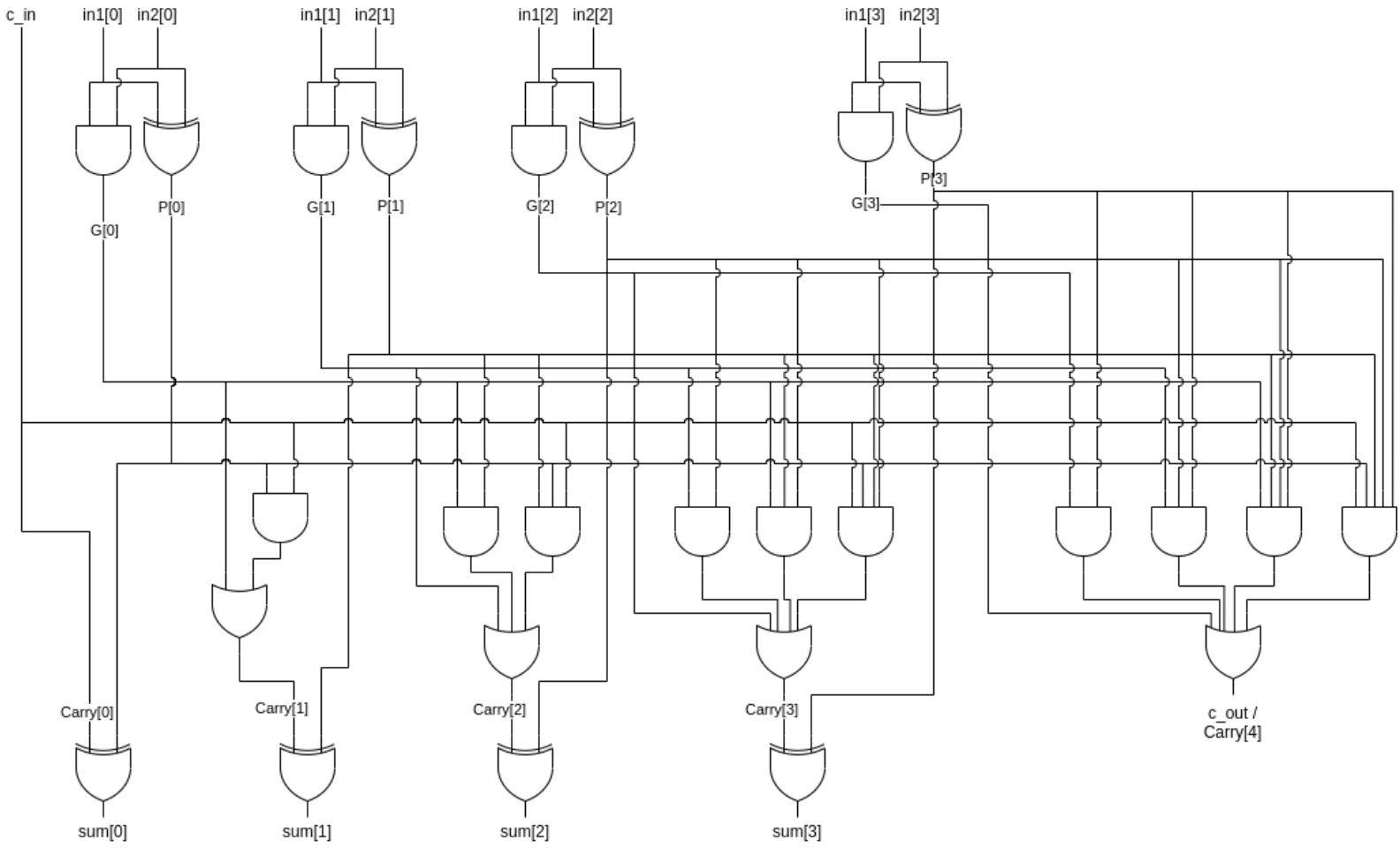
Q. How can you use the above circuit, to compute the difference between two n-bit numbers?

```
=> a - b = a + (-b)
=> observe that (-b) is the 2's complement of b
=> -b = ~b + 1, ~b = 1's complement of b
=> so a - b = RCA(a, ~b, 1) where 1 is the carry in and a and ~b are
the two inputs

In the circuit we can put a switch which will be connected to the
carry in of the adder and also with XOR gates with each of the input
bits of b. When the switch is turned on it will make the carry in 1
and simultaneously flip all the bits of b (XOR with 1 flips  the bit
value). This way we will get a-b in output. When the switch is off it
will not change anything and we will get a+b in output.
```

PAGE INTENTIONALLY LEFT BLANK

PART 2: Carry Look-ahead Adders

1. 4-bit carry look-ahead adder



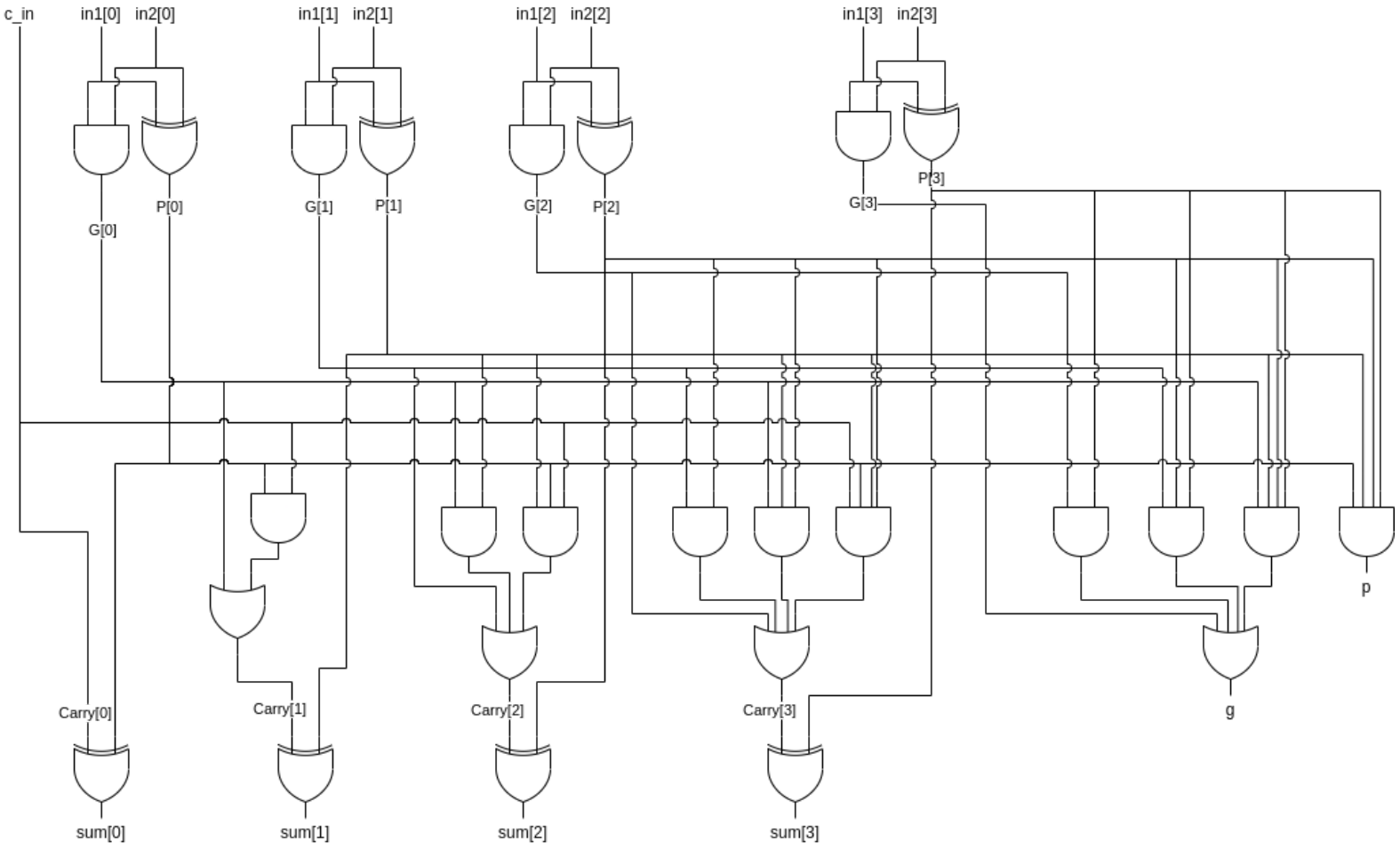
The carry lookahead adder reduces the delay in calculating the sum by simultaneously calculating the carries instead of waiting for the carry from the previous block to ripple in which is the case for ripple carry adder. The logic for the carry look-ahead adder is as follows:

```
G[i] = in1[i] & in2[i], 0 <= i <= 3
P[i] = in1[i] ^ in2[i], 0 <= i <= 3

Take c_in to be carry[0] then:

sum[i] = P[i] ^ carry[i], 0 <= i <= 3
carry[i] = G[i-1] | (P[i-1] & carry[i-1]), 1 <= i <= 4
```

```
Recursively expanding we get:

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)

In this case instead of generating the carry out ,i.e., carry[4] we give the block propagate and generate as output which are then used by the carry lookahead unit. This leads to a modular design using which we can make 16, 32 and 64 bit adders by combining the block propagate

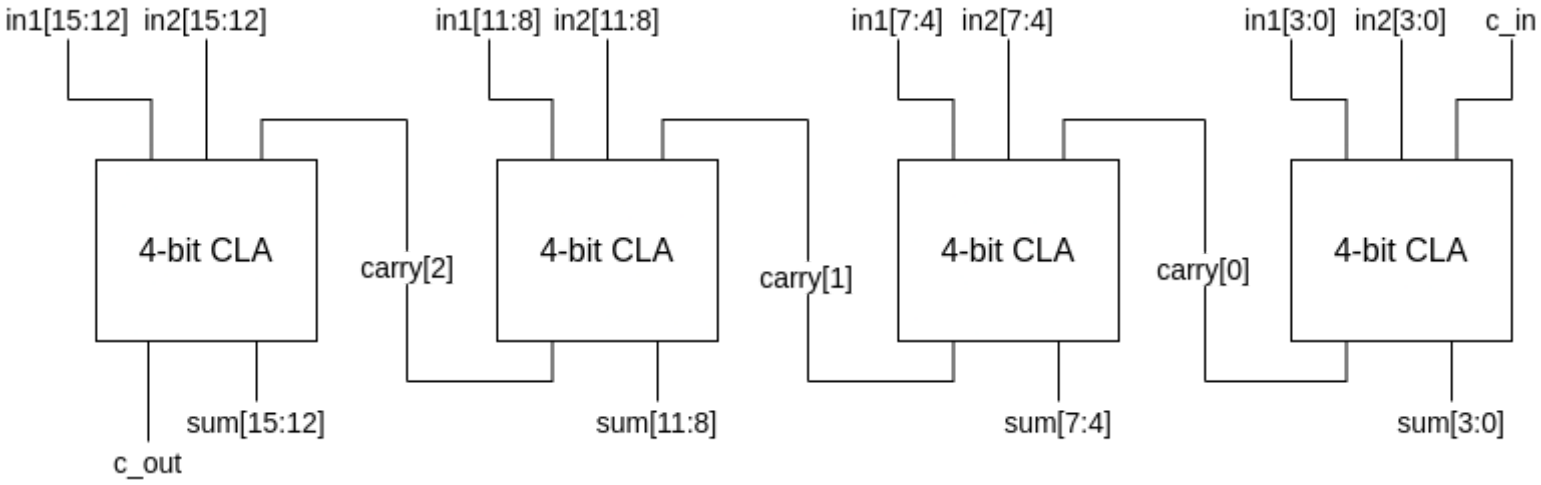


and generate from the lower levels instead of rippling the carry out every time. The other logic remains the same as the normal 4-bit CLA. The block propagate and generates are calculated as follows:

```
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])
```

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

This 16-bit adder is made by cascading four 4-bit CLAs by rippling the carry out from one block to another as shown 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] be the block propagate and generate of the four 4-bit CLAs. Then instead of rippling the carry out from one block to another we can reduce the delay in the circuit by adding an additional lookahead unit which simultaneously calculates these carries. This way the subsequent blocks don't have to wait for the carry from the previous block. Also, this leads to a modular design where we can further calculate the block propagate and generate of this complete 16-bit CLA as a whole and use them later for higher order adders.

The logic for the lookahead carry unit is as follows:

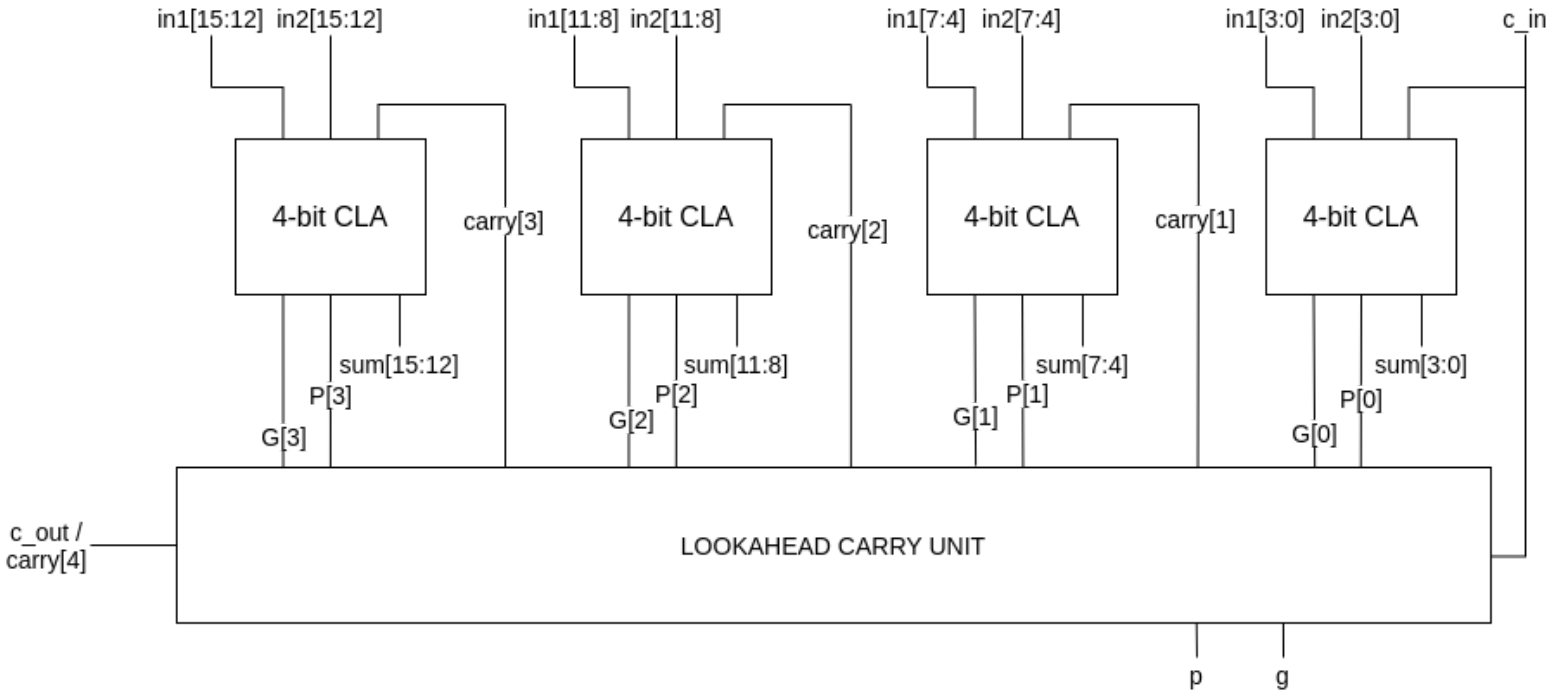
```
Take c_in to be carry[0] then
carry[i] = G[i-1] | (P[i-1] & carry[i-1]), 1 <= i <= 4
```

Recursively expanding we get

```
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)
```

Also 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)	6.167	5.308
Number of Slice LUTs	24	43
Number of bonded IOBs	50	52
Levels of Logic	14	11

A CLA with an additional lookahead carry unit performs better than simply rippling the carry at the cost of additional LUTs.

- Comparison with 4-bit RCA

4-bit	Delay (in ns)	Number of Slice LUTs	Logic Levels
CLA	2.363	9	4
RCA	5.565	20	20

- Comparison with 16-bit RCA

It can be observed that a CLA is significantly (almost 3.5x) faster than a RCA. Also the LUTs used are almost half that for a RCA.

16-bit	Delay (in ns)	Number of Slice LUTs	Logic Levels
CLA	5.308	43	11
RCA	18.717	80	70