Computer Organization and Architecture Laboratory

Verilog Assignment 1

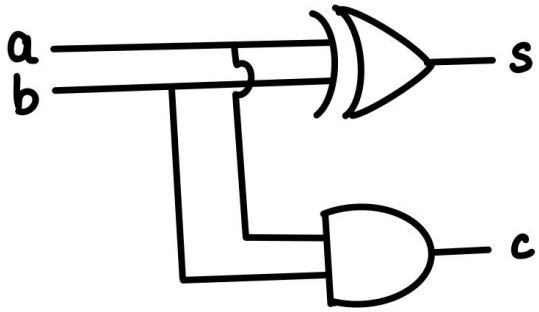
Group 15

Sai Shree Pradhan (22CS10089)

B Pranesh Vijay (22CS10013)

Ripple Carry Adders

1. Half Adder

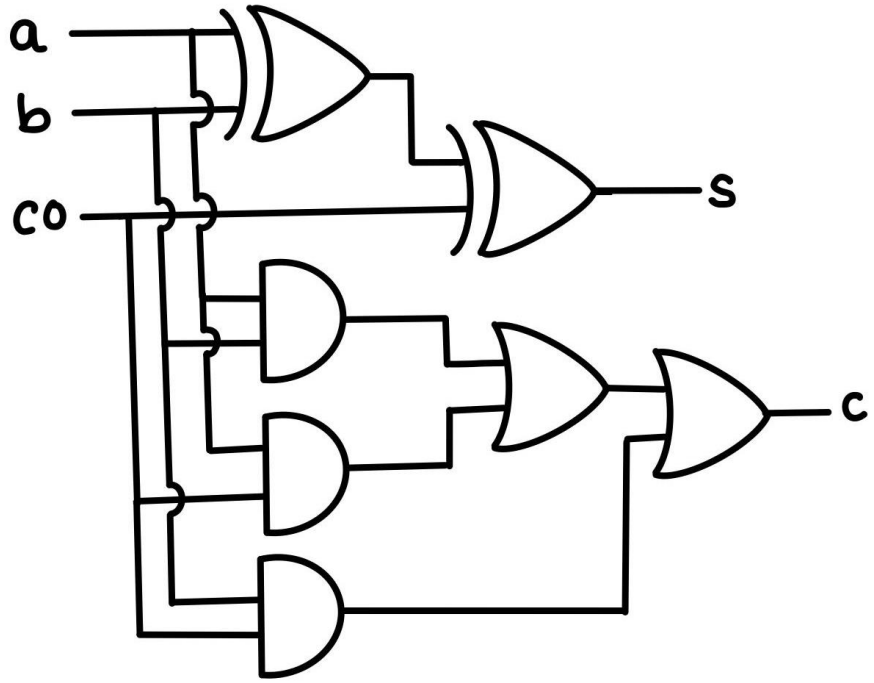


a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Logic:

```
s = a ^ b
c = a & b
```

2. Full Adder

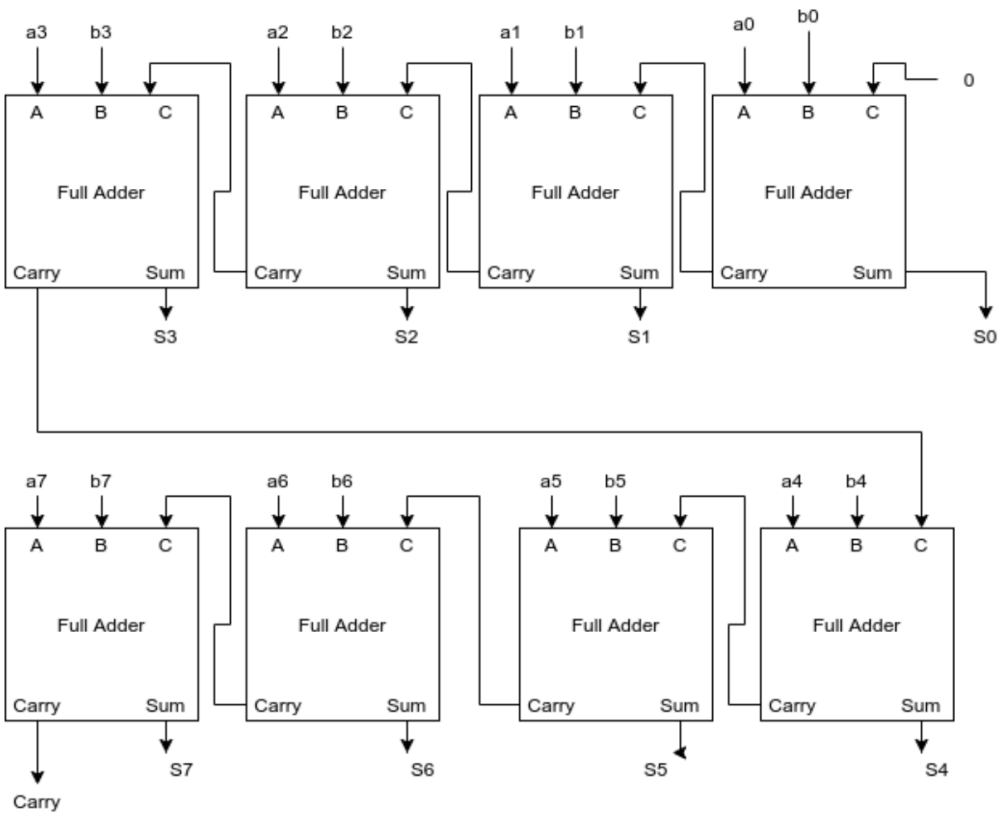


a	b	c0	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

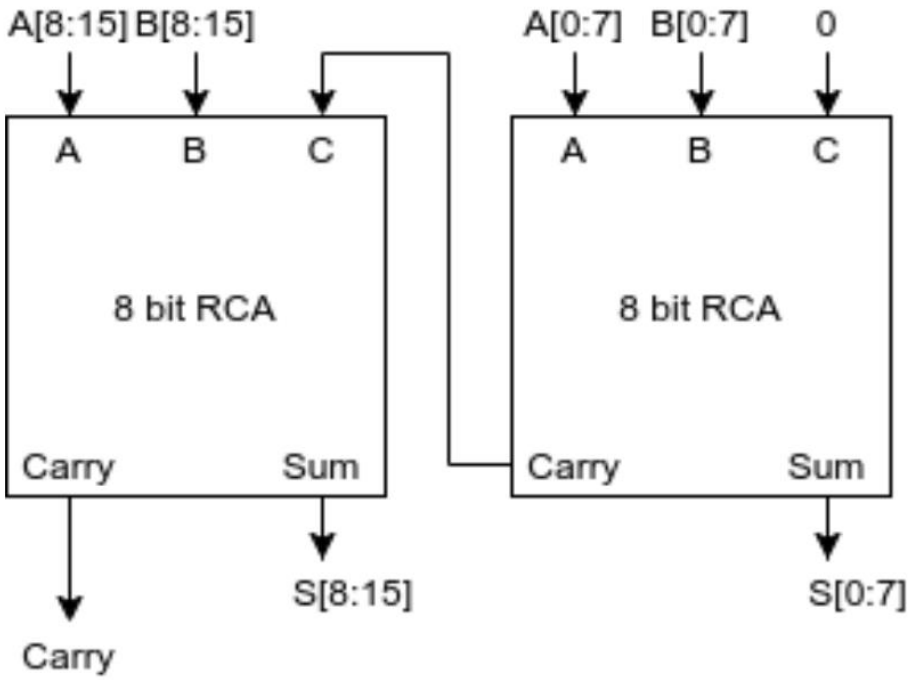
Logic:

```
s = a ^ b ^ c0
c = (a & b) | (b & c) | (c & a)
```

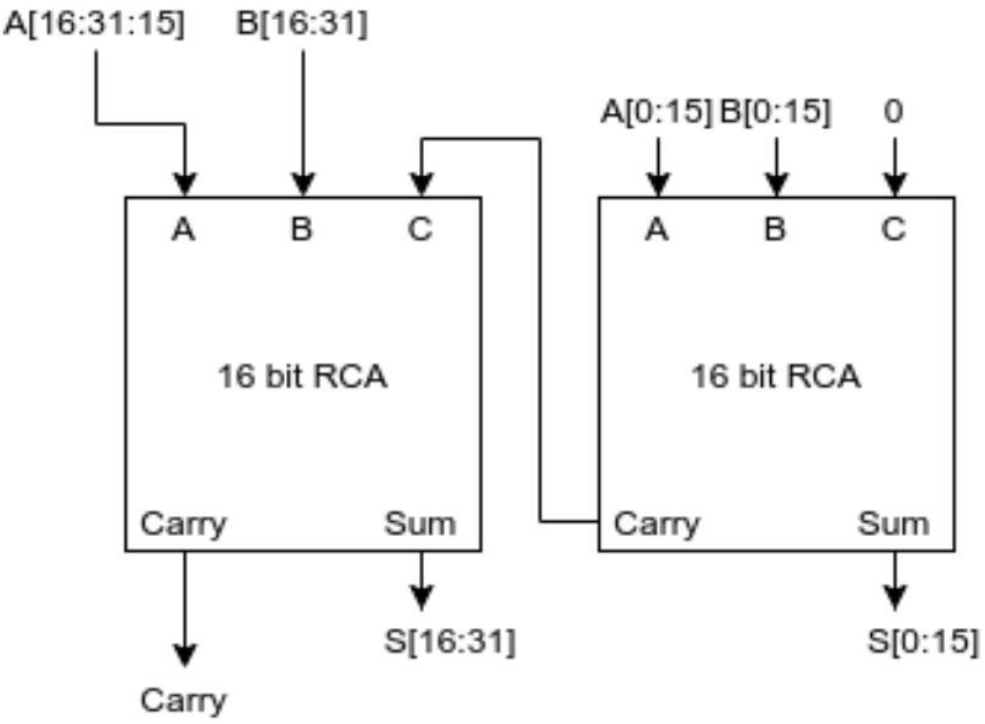
3. 8 – bit Adder



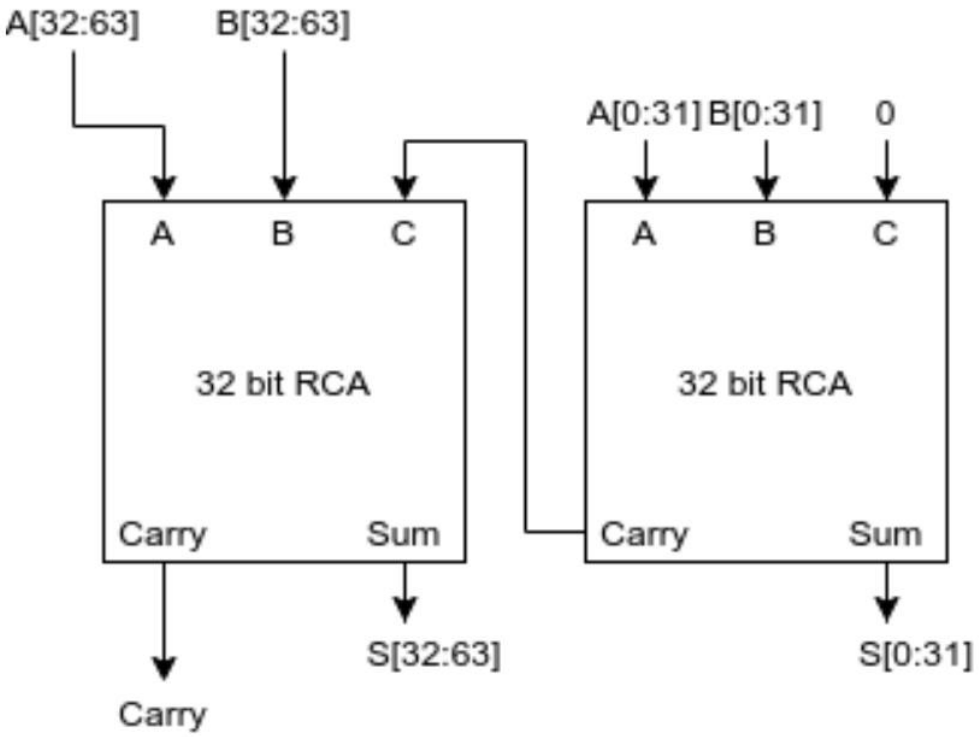
4. 16 – bit Adder



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	2.523	36	12	26
16-bit RCA	4.467	70	24	50
32-bit RCA	8.356	138	48	98
64-bit RCA	16.134	274	96	194

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

Given that $a - b = a + (-b)$, we can observe that $-b$ is the 2's complement of b. The 2's complement of b is calculated as:

$$-b = \sim b + 1$$

where $\sim b$ represents the 1's complement of b (i.e., flipping all bits of b).

Therefore, the expression a - b can be implemented using a Ripple Carry Adder (RCA) as:

$$a - b = RCA\ (a, \sim b, 1)$$

Here, a and $\sim b$ are the two inputs, and the carry-in is set to 1.

To design a circuit that can perform both addition and subtraction, we can introduce a switch. This switch is connected to the carry-in of the adder and to XOR gates that are applied to each bit of b.

- **When the switch is ON:** The carry-in is set to 1, and the XOR gates flip all the bits of b, effectively computing $a - b$.
- **When the switch is OFF:** The carry-in is 0, and the XOR gates leave b unchanged, resulting in $a + b$ as the output.

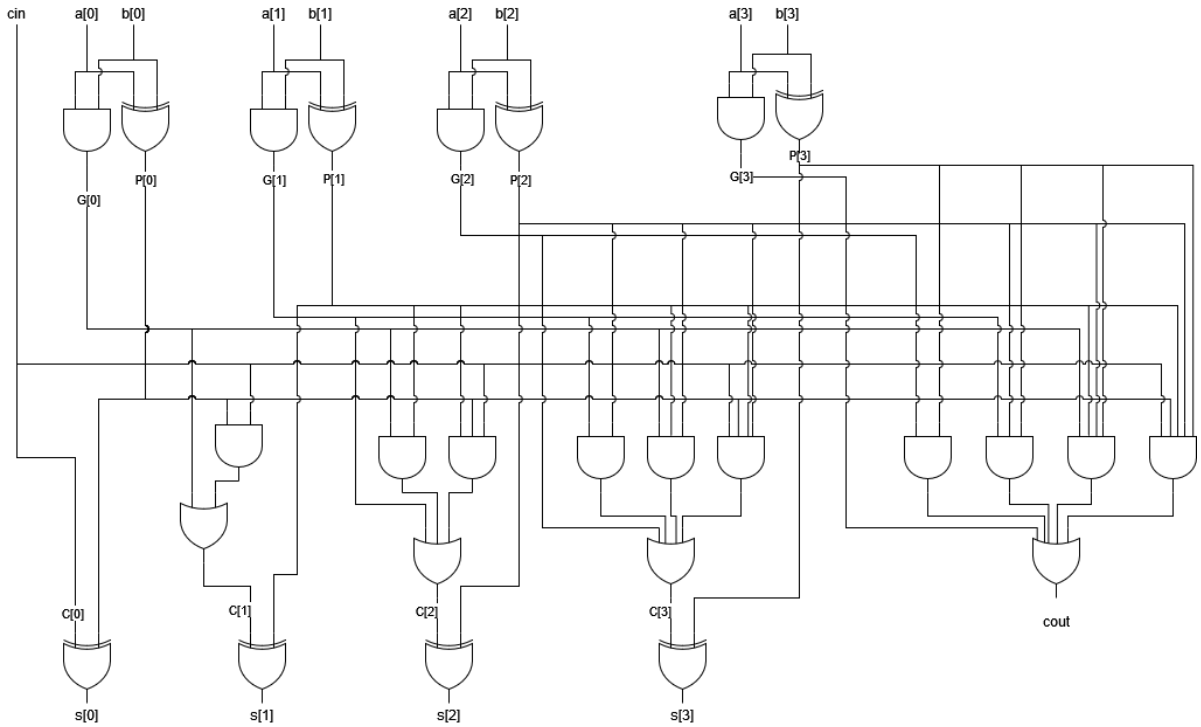
This design allows for the selection between addition and subtraction based on the state of the switch.

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:

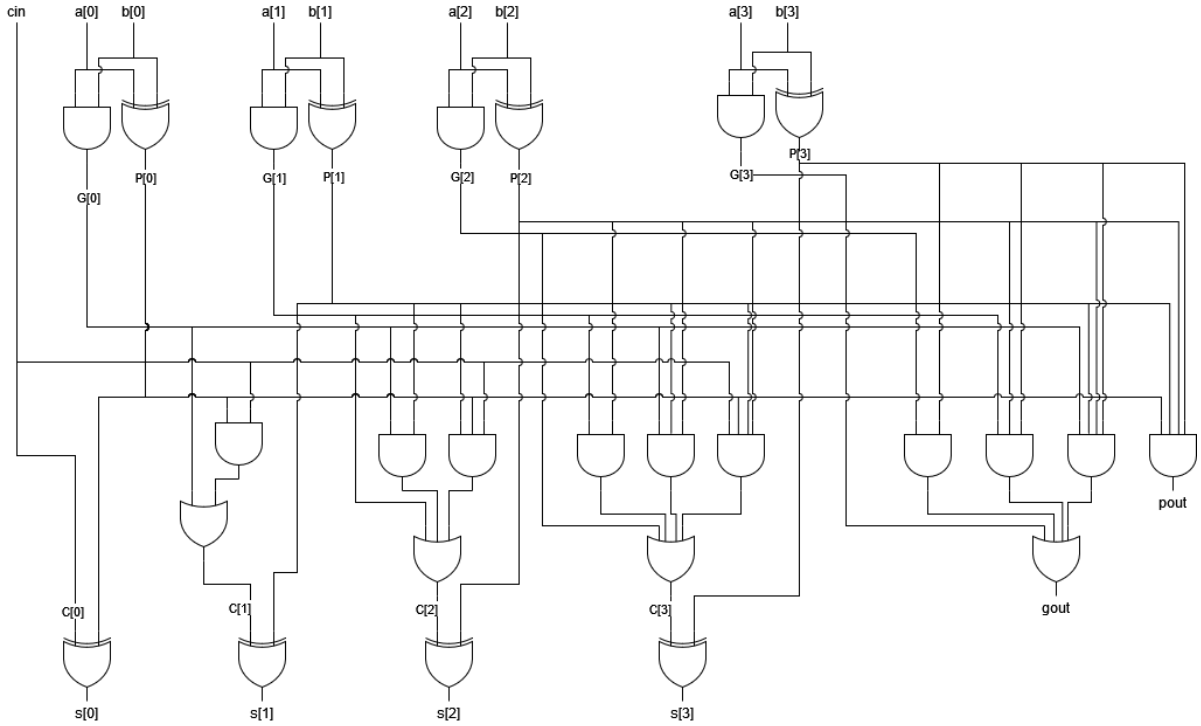
```
G[i] = a[i] & b[i], 0 <= i <= 3
P[i] = a[i] ^ b[i], 0 <= i <= 3

cin = C[0]
s[i] = P[i] ^ C[i], 0 <= i <= 3
C[i] = G[i-1] | (P[i-1] & C[i-1]), 1 <= i <= 4

Recursively expanding we get:

C[1] = G[0] | (P[0] & C[0]) = G[0] | (P[0] & cin)

C[2] = G[1] | (P[1] & C[1]) = G[1] | (P[1] & G[0]) | (P[1] &
P[0] & cin)

C[3] = G[2] | (P[2] & C[2]) = G[2] | (P[2] & G[1]) | (P[2] &
P[1] & G[0]) | (P[2] & P[1] & P[0] & cin)

C[4] = G[3] | (P[3] & C[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] & cin)

cout = C[4]
```



2. 4 – bit Carry Look – Ahead Adder (Augmented)

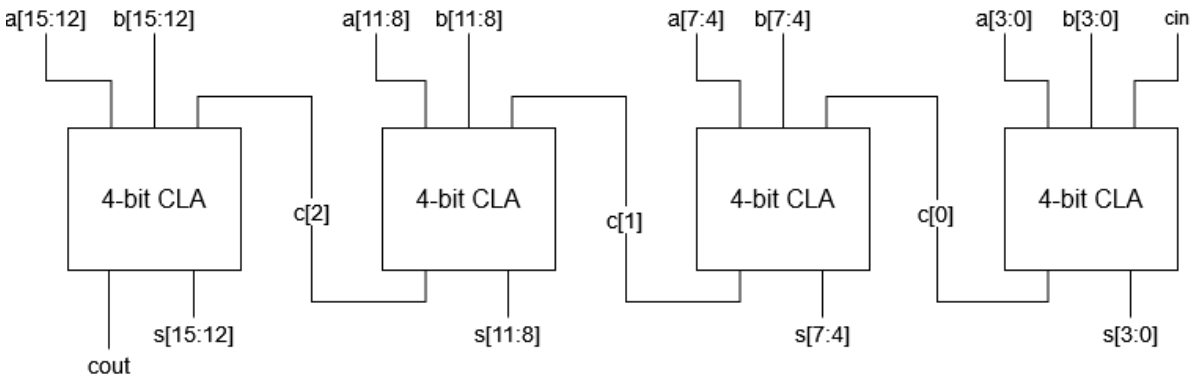
In this case instead of generating the carry out ,i.e., C[4] we give the block propagate and generate as output which are then used by the carry lookahead unit. This leads to a modular design using which we can make 16, 32 and 64 bit adders by combining the block propagate and generate from the lower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

```
pout = P[3] & P[2] & P[1] & P[0]
gout = 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)

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

4. 16 – bit Carry Look – Ahead Adder (Look – Ahead Carry Unit)

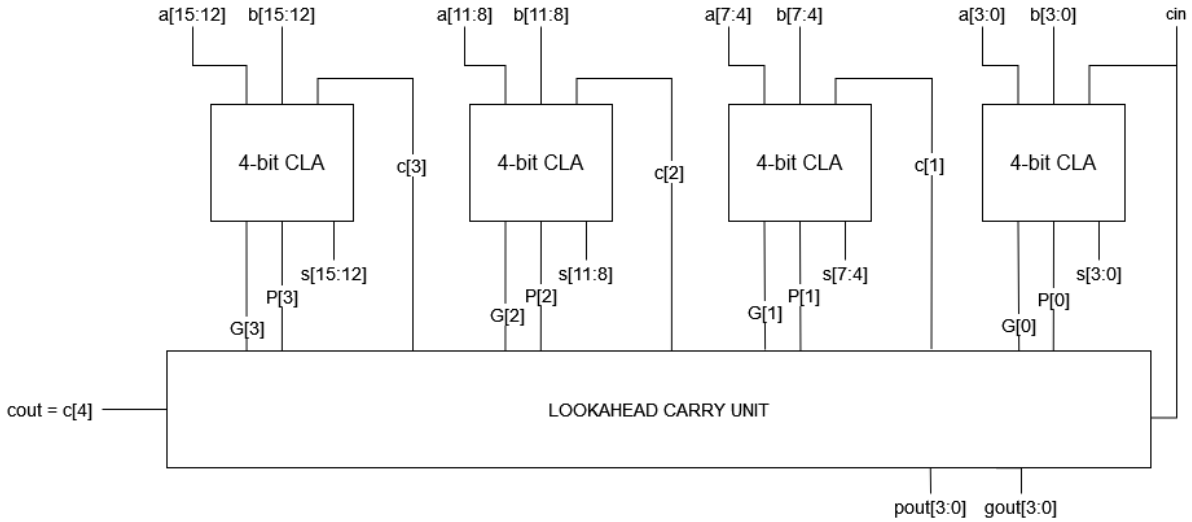
Let P[3:0] and G[3:0] be the block propagate and generate of the four 4-bit CLAs. Then instead of rippling the carry out from one block to another we can reduce the delay in the circuit by adding an additional lookahead unit which simultaneously calculates these carries. This way the subsequent blocks 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 cin to be C[0] then

C[i] = G[i-1] | (P[i-1] & C[i-1]), 1 <= i <= 4

Recursively expanding we get

C[1] = G[0] | (P[0] & C[0]) = G[0] | (P[0] & cin)

C[2] = G[1] | (P[1] & C[1]) = G[1] | (P[1] & G[0]) | (P[1] &
P[0] & cin)

C[3] = G[2] | (P[2] & C[2]) = G[2] | (P[2] & G[1]) | (P[2] &
P[1] & G[0]) | (P[2] & P[1] & P[0] & cin)

C[4] = G[3] | (P[3] & C[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] & cin)

cout = C[4]
```

Also block propagate p and generate g are calculated as:

```
pout = P

gout = G
```



SYNTHESIS SUMMARY

	16 – bit Carry Look – Ahead Adder (Ripple Carry)	16 – bit Carry Look – Ahead Adder (Look – Ahead Carry Unit)
Delay (in ns)	4.467	3.973
Number of Slice LUTs	40	24
Number of bonded IOBs	58	50
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	1.551	6	4
RCA	1.551	6	20

Comparison with 16 – bit RCA

16-bit	Delay (in ns)	Number of Slice LUTs	Logic Levels
CLA	3.973	40	11
RCA	4.467	24	70