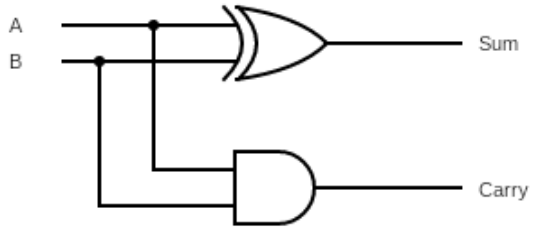
Computer Organization and Architecture Laboratory, Autumn 2021

Assignment 1

Neha Dalmia (19CS30055) and Rajat Bachhawat(19CS10073)

Problem 1

1a : Half Adder

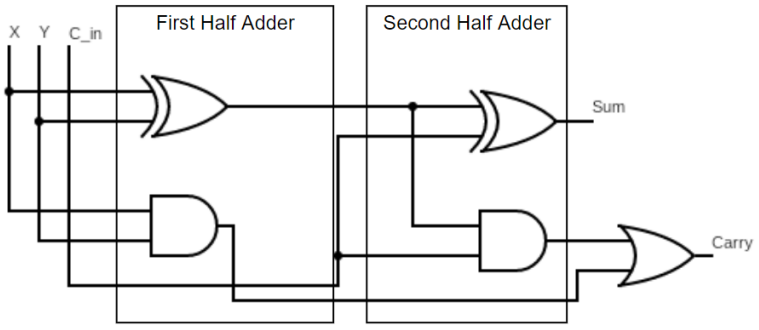


Circuit Diagram of Half Adder

Truth Table for Half Adder

a	b	sum = a ^ b	carry = a & b
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

1b : Full Adder

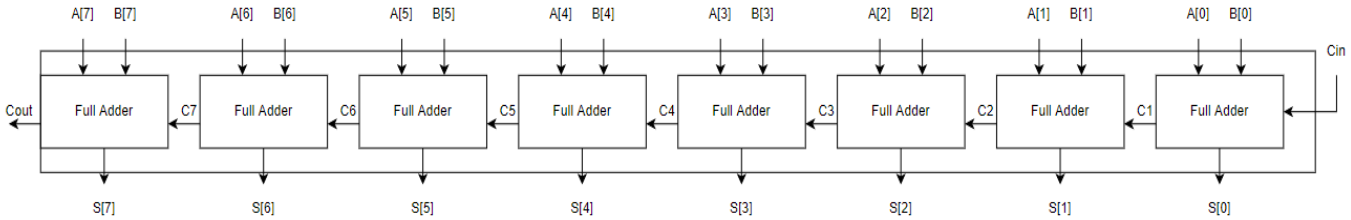


Circuit Diagram for Full Adder implemented using Half Adders

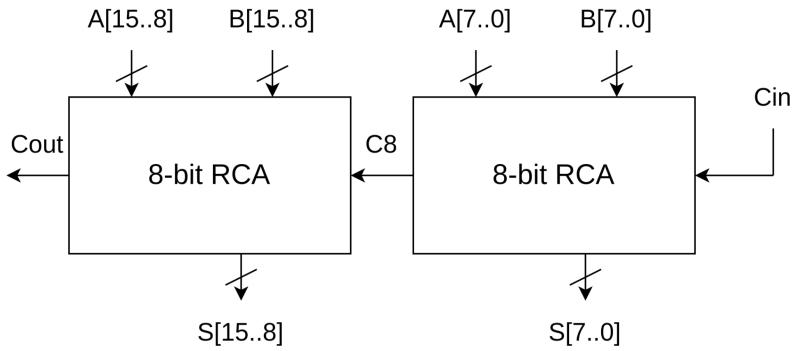
Truth Table for Full Adder

a	b	carry_in	sum = a ^ b ^ C_in	carry = (a & b) | (C_in & (a ^ b))
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

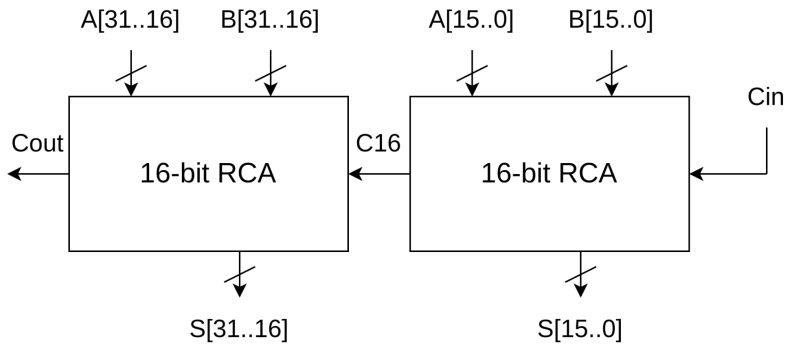
1c : n-bit Ripple Carry Adder Speeds



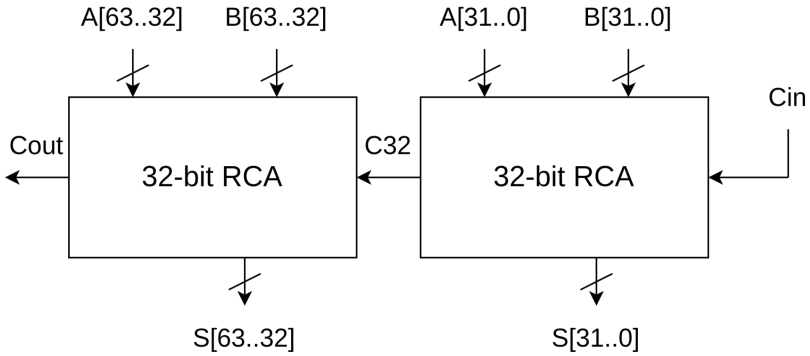
Circuit Diagram for 8-bit Ripple Carry Adder implemented using full adders



Circuit Diagram for 16-bit Ripple Carry Adder implemented using 8-bit RCAs



Circuit Diagram for 32-bit Ripple Carry Adder implemented using 16-bit RCAs



Circuit Diagram for 64-bit Ripple Carry Adder implemented using 32-bit RCAs

Comparison of the various n-bit RCAs

	8-bit RCA	**16-bit RCA**	**32-bit RCA**	**64-bit RCA**
Longest Delay	9.949ns	18.717ns	36.253ns	71.325ns
Number of Slice LUTs	40	80	160	320
Number of Bonded IO Buffers	26	50	98	194
Levels of Logic	36	70	138	274

As the number of bits increases, the delay also increases mainly because of the **critical path - the rippling of the carry.**

1d : Subtraction using RCA

In order to perform a - b , we can write it as

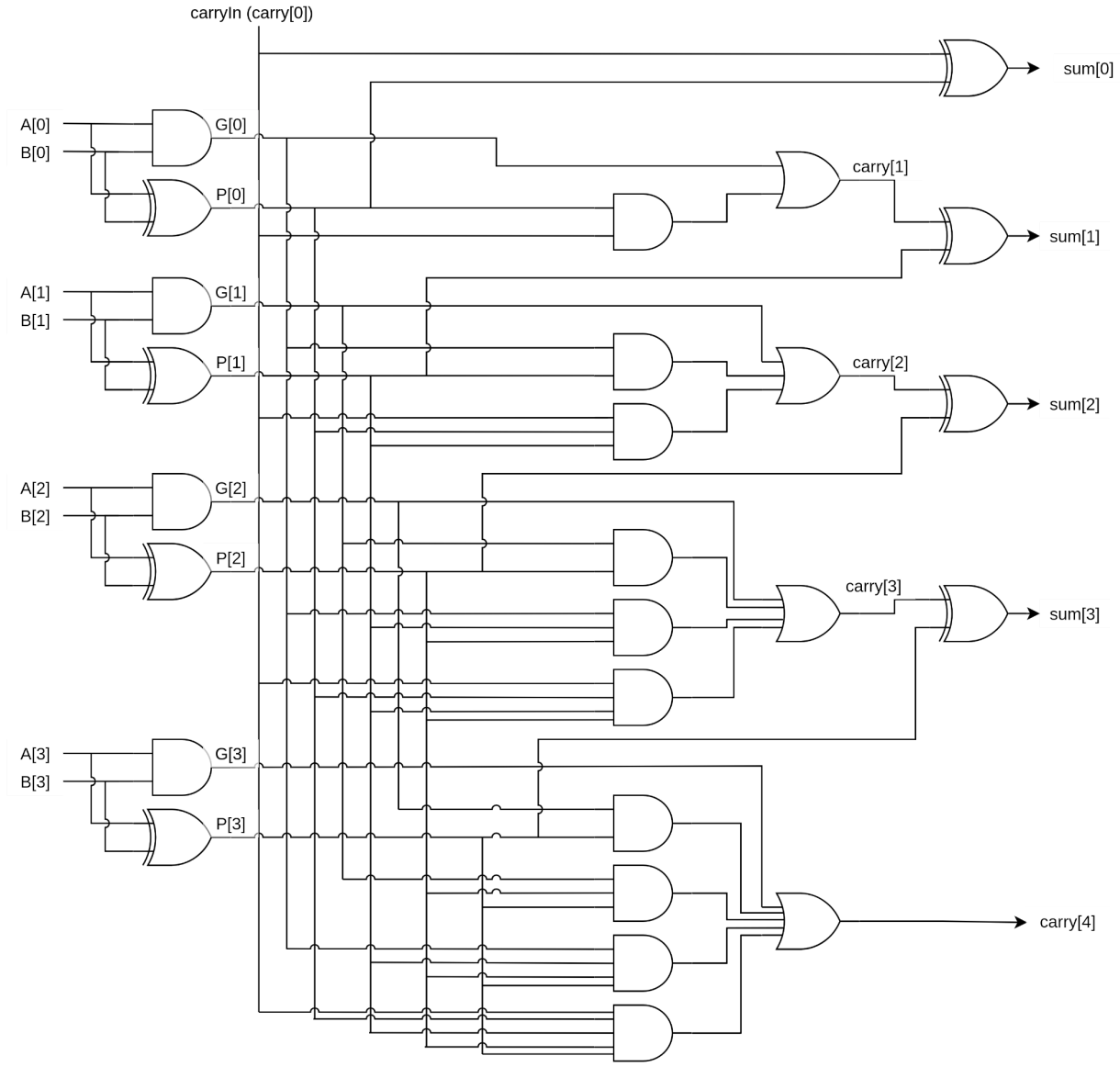
a + (-b)
= a + (2s complement of b)
= **a + (1s complement of b) + 1**

where the 1s complement of b is obtained by inverting every bit of b.

Thus, we pass every bit of b through a NOT gate to get the inverse of b. In order to do the addition of 1, we can set the **initial carry to 1** in our RCA. This way, we get a + (inverse of b) + 1 using an RCA.
In order to perform a subtraction of two n bit numbers, we can ripple in a series of n full adders to make an n bit RCA, and use it to perform the addition of a + (1s complement of b) + 1 by setting the input carry to 1 instead of 0 and completenting bits of b using NOT gates. Thus, subtraction can be performed using addition using the 2s complement method. We do not have to perform two additions to add the extra 1 as it can be passed as the carry.

Problem 2

2a : 4-bit Carry Lookahead Adder



Design of the 4-bit CLA

Inputs:

A[3..0], B[3..0], C0 (input carry)

Outputs:

Sum[3..0] : Sum of A[3..0] and B[3..0]
C4 (Output carry)

P[0..3] = Propagate signals
G[0..3] = Generate signals
carry[1..4] = Generated carry bits

Equations for the generate and propagate signals along with sum:

P[i] = A[i] ^ B[i], for 0<=i<=3
G[i] = A[i] & B[i], for 0<=i<=3
sum[i] = P[i] ^ carry[i], for 0<=i<=3

Boolean equations of the Look-ahead carry generation for the 4 carry bits, **C1, C2, C3, and C4** :

carry[0] = input carry
carry[i+1] = G[i] | (P[i] & carry[i]), for 0<=i<=3

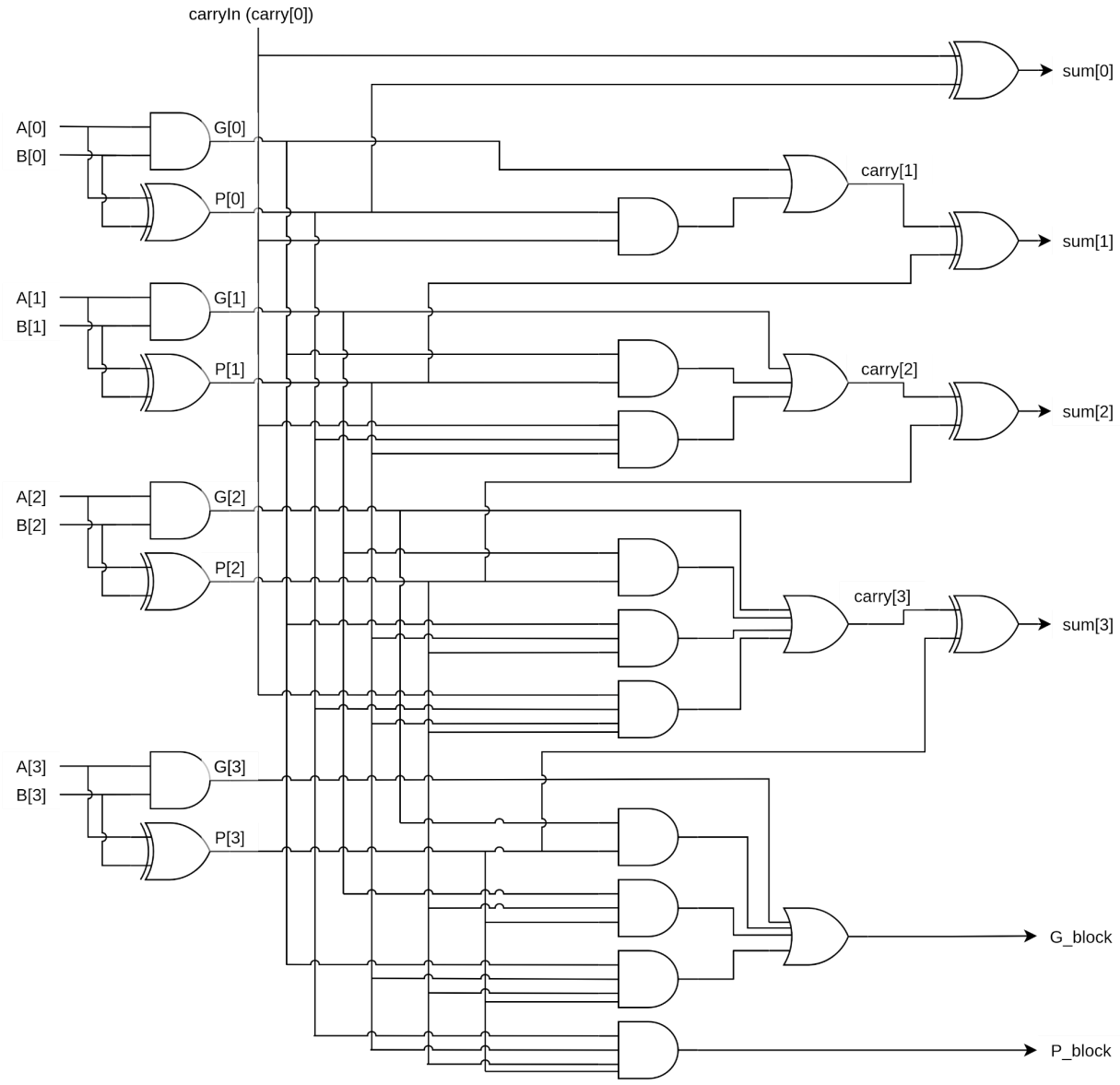
More specifically, on expanding:
carry[1] = G[0] | (P[0] & carry[0])
carry[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & carry[0])
carry[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & carry[0])
carry[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & carry[0])

2b : Comparison of 4-bit CLA with 4-bit RCA

	4-bit RCA	**4-bit CLA**
Longest Delay	5.565ns	2.123ns
Number of Slice LUTs	20	9
Levels of Logic	20	4

Hence, the 4-bit CLA is much faster than the 4-bit RCA. The reason for this is that we have eliminated the rippling of carry from one bit to another (which formed the critical path) by introducing generate and propagate signals (and computing them in an optimised way using multiple levels of **lookahead circuit**) which speed up the process of carry generation and propagation through the circuit.

2c (i) : Augmented 4-bit CLA



Design of the augmented 4-bit CLA

In this case, instead of returning the carry out we give the **block propagate and generate** as output which are then used by the carry lookahead unit. This modular design can be used to make 16, 32 and 64 bit adders by combining the block propagate and generate from the lower levels instead of waiting for the carry to ripple out every time.

Inputs:

A[3..0], B[3..0], C0 (input carry)

Outputs:

Sum[3..0] : Sum of A[3..0] and B[3..0]
P_Block : Block Propagate
G_Block : Block Generate

P[0..3] = Propagate signals

G[0..3] = Generate signals
carry[1..4] = Generated carry bits

Equations for the generate and propagate signals:

P[i] = A[i] ^ B[i], for 0<=i<=3
G[i] = A[i] & B[i], for 0<=i<=3

Boolean equations of the Look-ahead carry generation for the 4 carry bits, C1, C2, C3, and C4:

carry[0] = input carry
carry[i+1] = G[i] | (P[i] & carry[i]), for 0<=i<=3
More specifically, on expanding:
carry[1] = G[0] | (P[0] & carry[0])
carry[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & carry[0])
carry[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & carry[0])

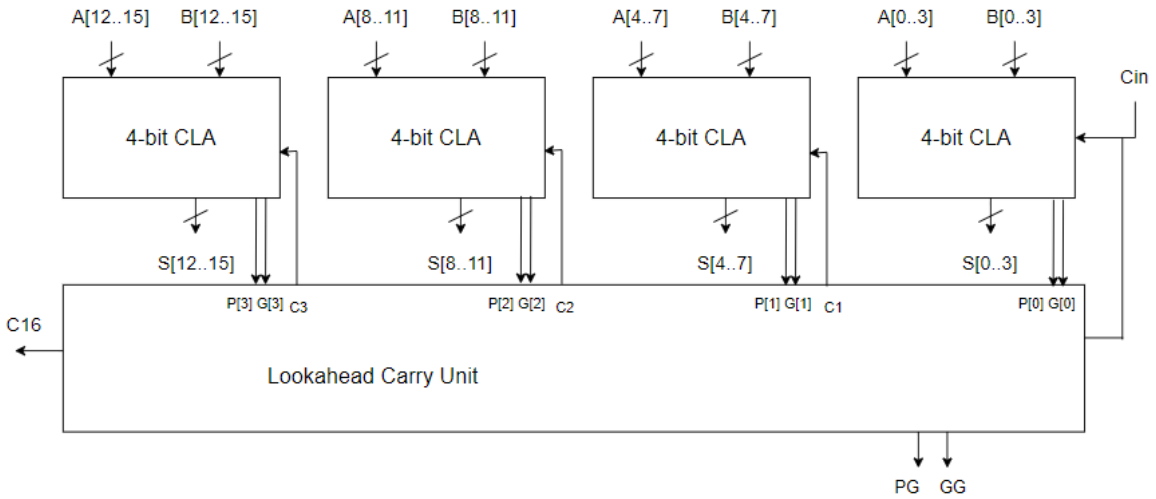
No need to calculate c[4] as the LCU does that using P_Block and G_Block.

sum[i] = P[i] ^ carry[i], for 0<=i<=3

Calculation of Block Propagate and Generate for the entire 4-bit block (independent of carry[0]):

P_block = P[3] & P[2] & P[1] & P[0]
G_block = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) | (P[3] & P[2] & P[1] & G[0])

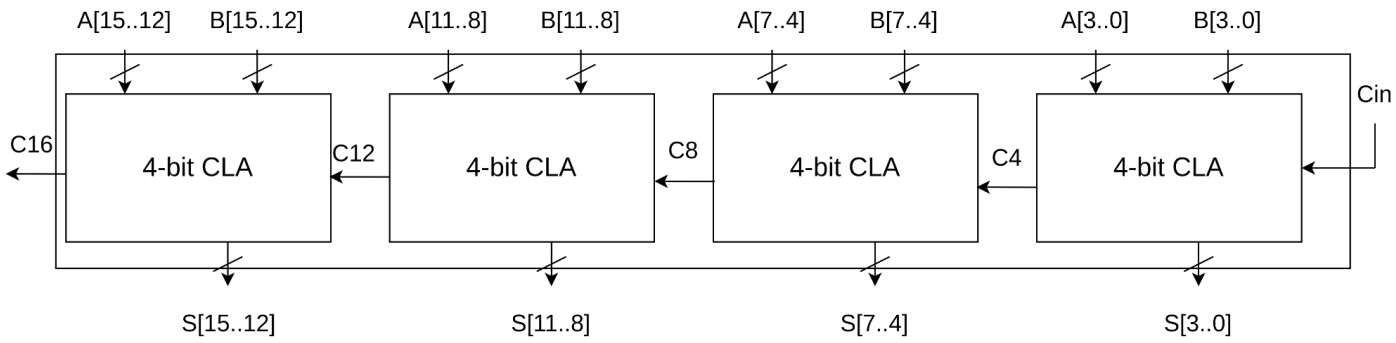
2c (ii) : 16-bit CLA with Lookahead Carry Unit



Design 1 : Design of 16-bit CLA using block propagates and block generates passing through a second layer of lookahead

Instead of waiting for the carry to ripple out from the 4-bit blocks, the LCU simultaneously calculates these carries using the P_Block and G_block from the 4-bit CLAs. Then it creates its own P and G blocks which can be used by higher order adders.

2c (iii) : Comparison of Delays between the two versions of 16-bit CLA



Design 2 : Design of 16-bit CLA by rippling carries from the 4-bit CLA blocks

	16-bit CLA without Lookahead Carry Unit	**16-bit CLA with Lookahead Carry Unit**
Longest Delay	6.167ns	5.243ns
Number of Slice LUTs	16	29
Number of Bonded IO Buffers	50	52
Levels of Logic	14	11

Hence, the inclusion of the Lookahead Carry Unit improves the speed of the 16-bit CLA in comparison to cascaded 4-bit CLA structure where the carry bits are rippled in. The reason is that the former has an extra level of lookahead circuit which optimises the generation/propagation of carry bits even more. As the carry-chain of the RCA forms the critical path for the delay, optimising the generation/propagation of carry bits by using a lookahead circuit reduces the longest delay (as in the CLA).

2c (iv) : Comparison between the 16-bit CLA and the 16-bit RCA

	16-bit Ripple Carry Adder	**16-bit Carry Lookahead Adder**
Longest Delay	18.717ns (Levels of Logic = 70)	5.243ns (Levels of Logic = 11)
Number of Slice LUTs	48 / 63400	29 / 63400

Number of Bonded IO Buffers	50 / 210	52 / 210
Number of LUT-FF Pairs used	48	29
Number of Occupied Slices	48 / 15850	14 / 15850

Hence, we see that:

1. The **lookup table (LUT) cost** of the 16-bit CLA is much **lesser** in comparison to the 16-bit RCA.
2. The **speed** of the 16-bit CLA is much **greater** (as delay is lesser) in comparison to the 16-bit RCA.

The reason for this is that we have eliminated the rippling of carry from one bit to another by introducing generate and propagate signals (and computing them in an optimised way using multiple levels of **lookahead circuit**) which speed up the process of carry generation and propagation through the circuit.