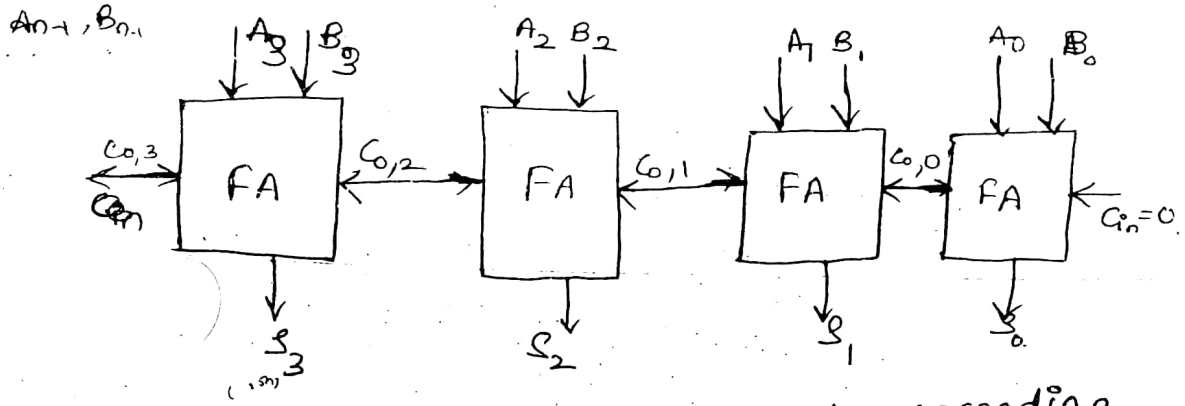
Ripple Carry Adder (or) Parallel Adder

4-bit Ripple Carry Adder.



> N-bit Adder can be constructed by cascading N full-Adders circuits in series, such that first full adder 'Co' is connected to next fulladder carry-in & so on.

This arrangement is known as ripple carry Adder, since carry bit ripples from one stage to another.

> The delay through the circuit depends upon the number of stages that must be traversed & is a function of the applied i/p signals.

> for some input combinations no rippling occurs at all, while for others, the carry has to ripple from LSB to the MSB. point. The propogation delay of such structure is defined as the worst case delay over all possible patterns.

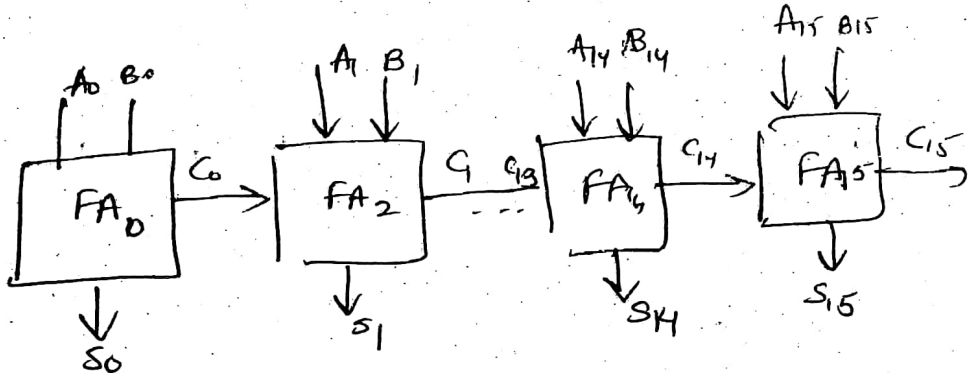
> The delay is then proportional to the no. of bits in the i/p words N' & is approximated by

$$t_{adder} \cong (N-1)t_{carry} + t_{sum}$$

16-bit
$t_{PC} = 12$
$t_S = 15$
$12(15) + t_{sum}$
$12 +$

> The propogation delay of ~~the~~ ripple carry adder is linearly proportional to N. This property becomes very important when designing adders for wide datapaths.

* A 16-bit RCA is realized using 16 identical full adders (FA) as shown in fig. The carry-propogation delay of each FA is 12ns & the sum-propogation delay of each FA is 15ns. The worst case delay of this 16-bit Adder?



$$t_{add} = (N-1)t_{pc} + t_{sum}$$

$$= (16-1)t_{pc} + t_{sum}$$

$$= 15 \times 12n + 15ns.$$

$$= 195ns$$

The ripple carry adder has a finite delay in obtaining the final o/p, as the carry has to be transmitted from one stage to another. In normal adder of 4 or 8-bits, this delay is insignificant, but the delay becomes prominent in case of 64 (or) 128 bit adders.

Hence, to increase the speed of the adder, faster carry generation techniques are made use of. These techniques improve the speed of operation of the adder, but on another hand increase the floor area as well.

(i) Carry Look Ahead Adder.

In RCA, the final carry-o/p cannot be obtained without the propogatn of carry-in through the previous adder cells. Using the generate & propogate functn, this carry propogatn can be avoided & the carry o/p's can be computed directly.

$$C_i = G_i + P_i C_{i-1}$$

Inputs A_i, B_i, C_{in}

$i = 0$

$C_0 = G_0 + P_0 C_{-1}$ — (1)

$i = 1$

$C_1 = G_1 + P_1 C_0$ — (2)

$i = 2$ $\quad G_1 + P_1 (G_0 + P_0 C_{-1})$

$C_2 = G_2 + P_2 C_1$

$= G_2 + P_2 (G_1 + P_1 (G_0 + P_0 C_{-1}))$

$i = 3$.

$C_3 = G_3 + P_3 C_2$

$= G_3 + P_3 \{ G_2 + P_2 G_1 + P_1 P_2 G_0 + P_0 P_1 P_2 C_{-1} \}$

A	B	C_{in}	S	Cout	
0	0	0	0	0	No carry generation i.e Cout = 0
0	0	1	1	0	
0	1	0	1	0	
0	1	1	0	1	Carry Propogate Cout = Cin
1	0	0	1	0	
1	0	1	0	1	
1	1	0	0	1	Cout = 1.
1	1	1	1	1	

G_i $\quad P_i$

$C_0 = \underbrace{A \cdot B}_{\text{Carry generatn}} + \underbrace{(A \oplus B)}_{\text{Carry propogatn}} C_{in}$

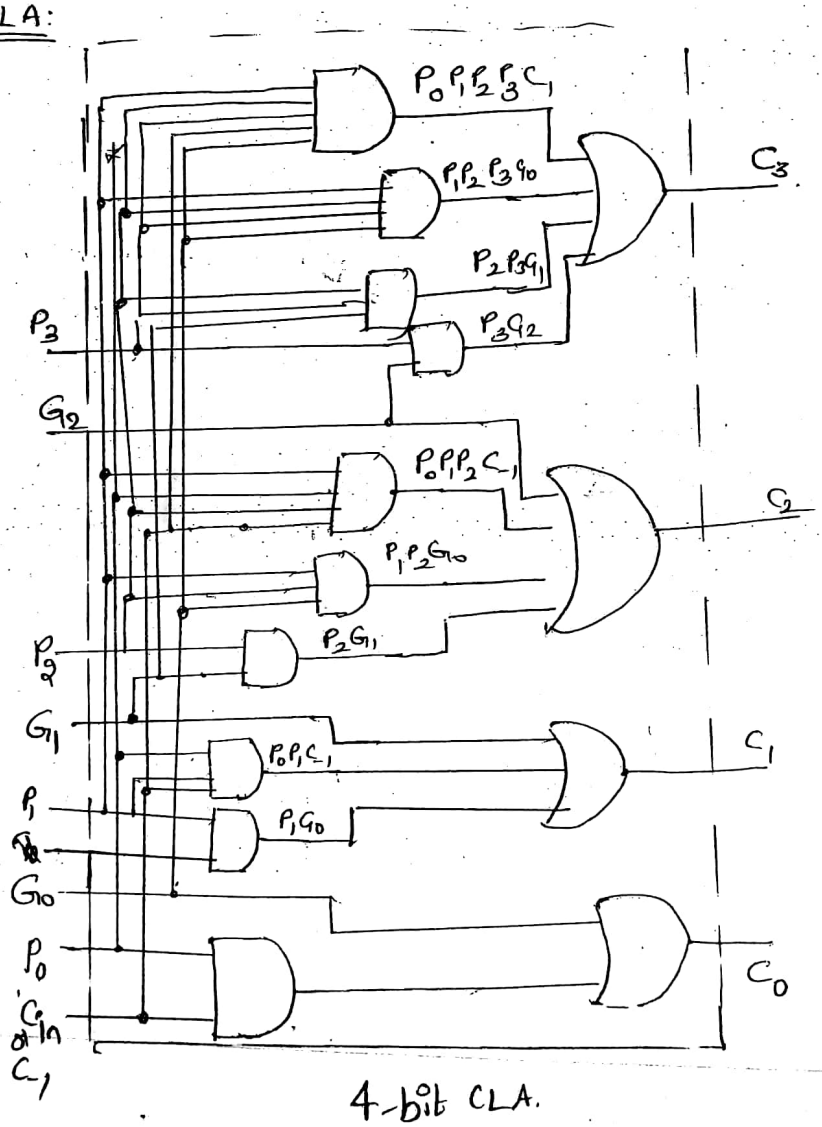
$C_0 = G_i + P C_{in}$

> These equations indicate that carry outputs can be computed directly, without the need for propogating through the adder cells

> Hence, as there is no ripple effect, the speed of the adder is increased.

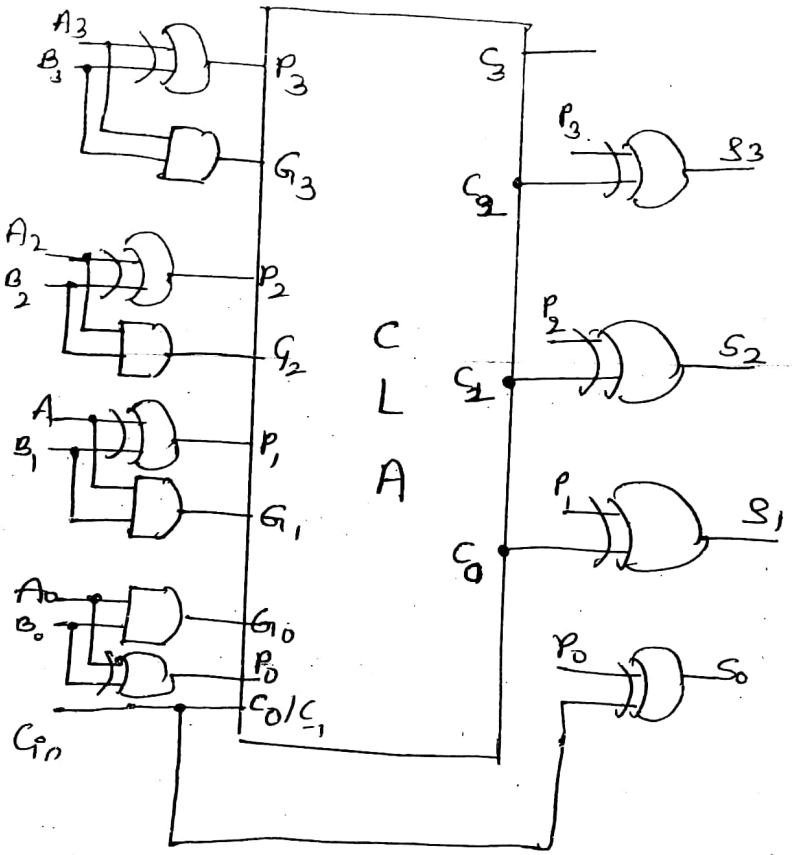
> But the limitation is that with increased no. of carry outputs, the expression becomes much larger & logic becomes quite complexity.

> CLA:



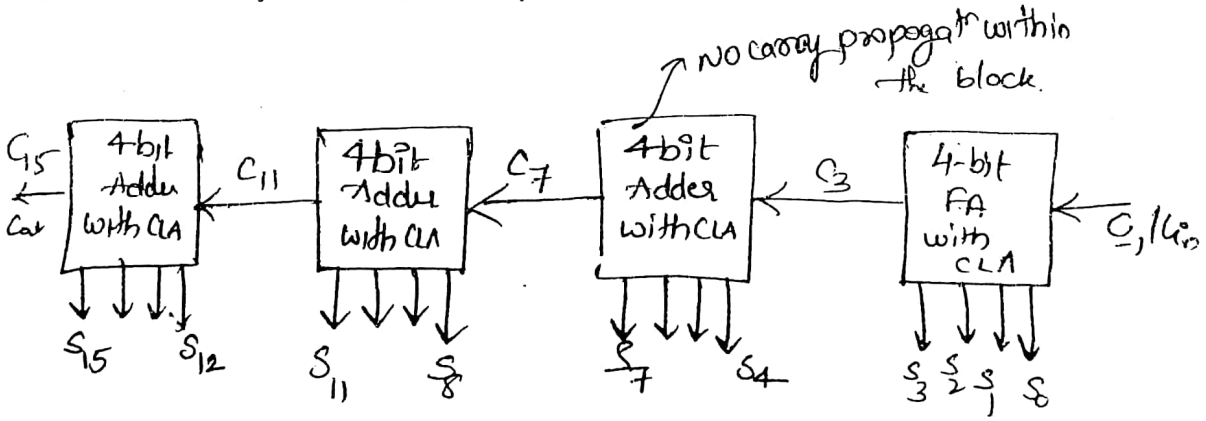
4-bit CLA.

4 bit full Adder Using CLA.



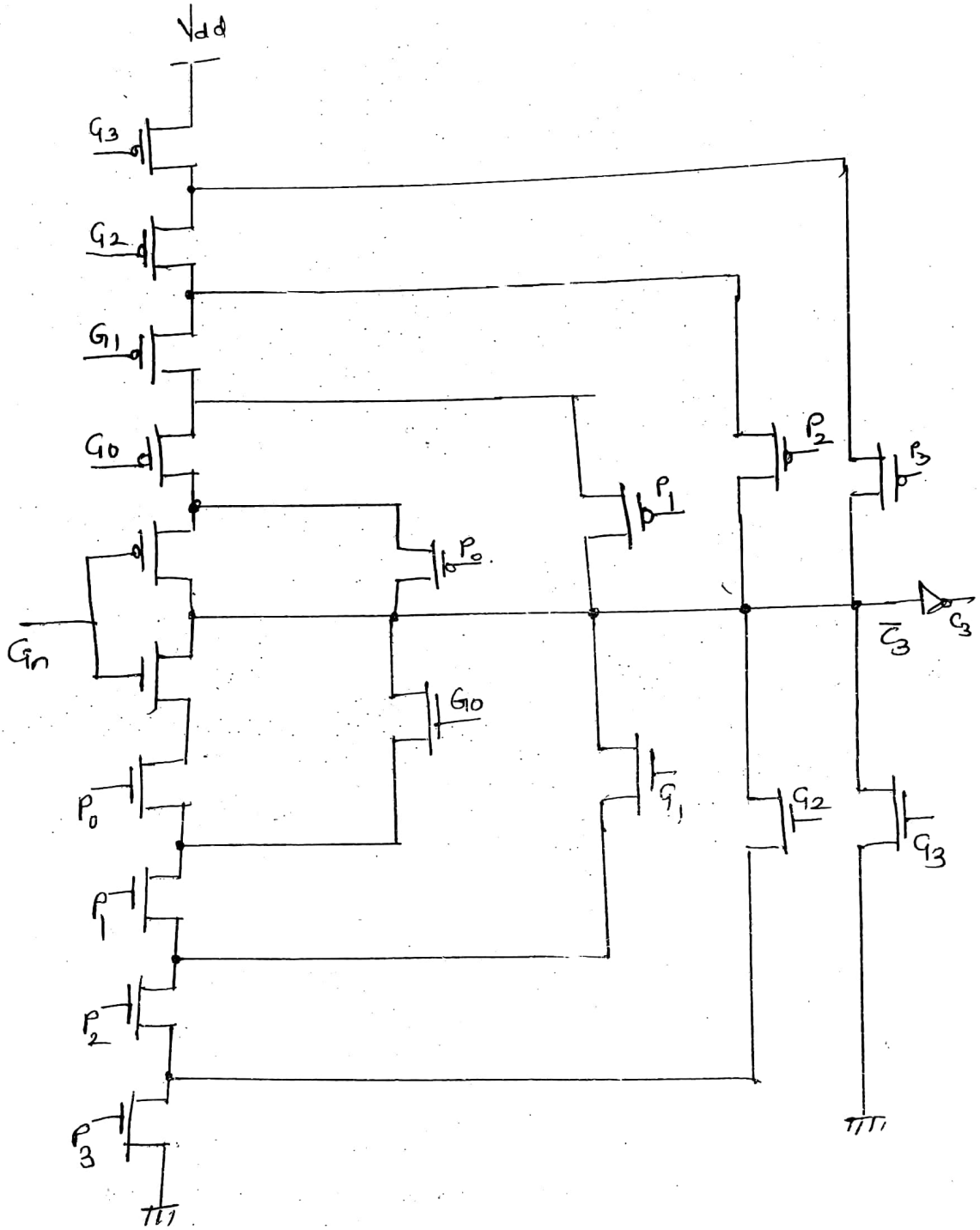
This unit produces the four carry outputs simultaneously However, C_0, C_1, C_2 are required only for producing the sum bits. These carry o/ps will not ripple through.

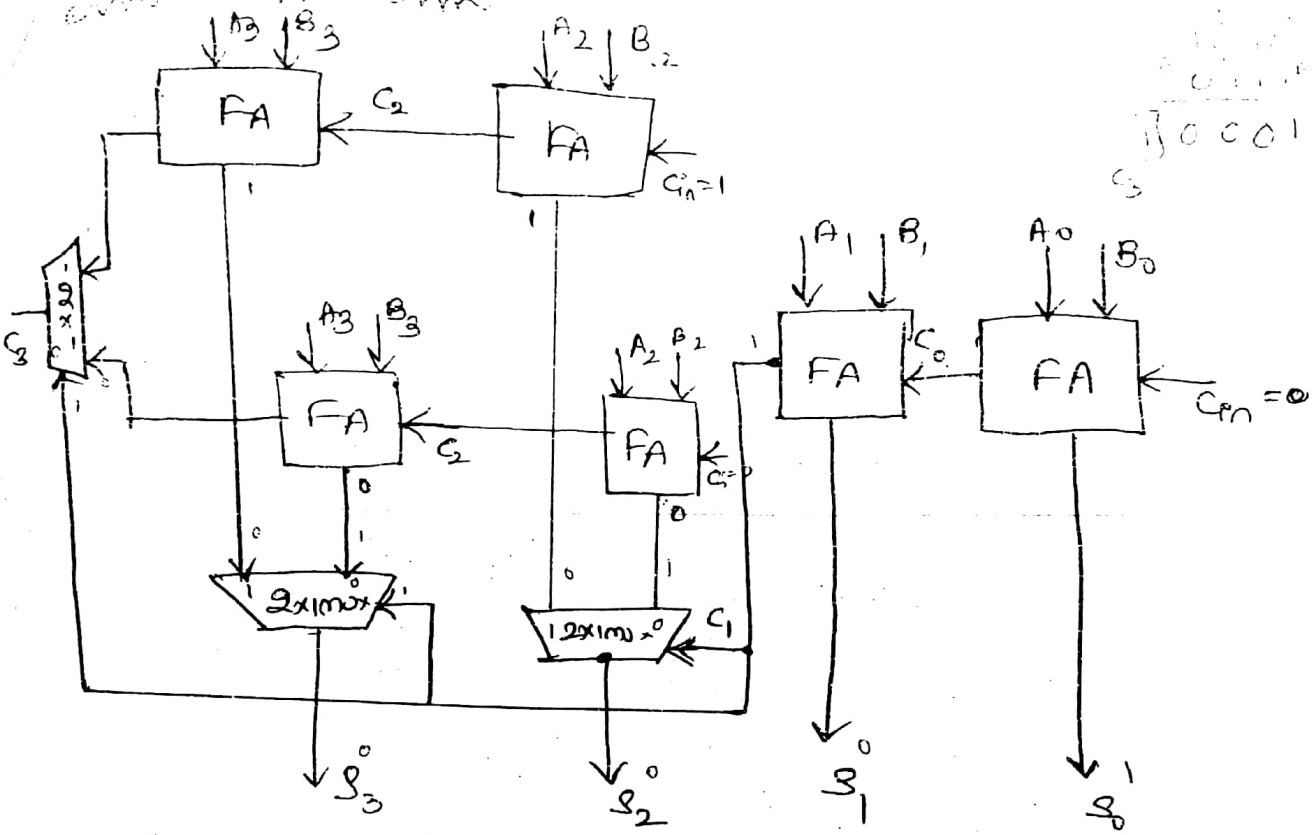
* 16-bit full-Adder using 4-bit FullAdder.



- b) $\overline{C_3}$ expression – CMOS implementation

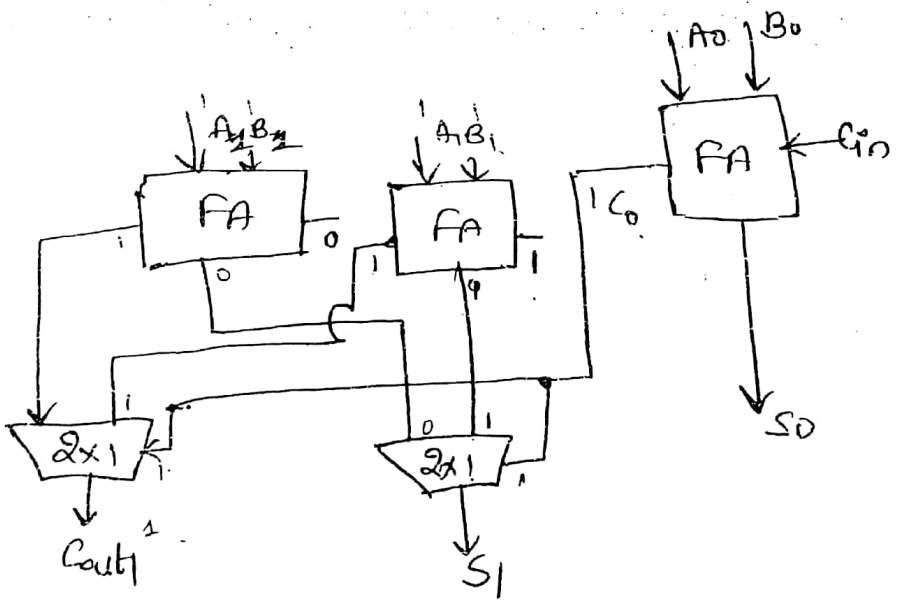
$$\overline{C_3} = G_3 + P_3G_2 + P_2P_3G_1 + P_1P_2P_3G_0 + P_0P_1P_2P_3C_{-1}$$





4-bit CSA.

2-bit CSA:



Cin=0

A1	A0
1	1
B1	B0
1	1
1	0

1|9
4, 7, 13, 21, 25, 29, 30
31, 33, 39, 38, 37, 43, 45
57, 49, 54

* Carry Select Adder.

① A standard logic design technique to accelerate the critical path is to pre-compute the o/ps for both possible i/ps, & then use a multiplexer to select btw the two o/p choices.

② CSA consists of two ripple carry adders & a multiplexer.

③ The calculation is done twice with an assumption that the carry i/p is '0' & another adder with an assumption that the carry i/p is 1.

④ After the two results are calculated, the correct sum, as well as the correct carry-out, is then selected with the multiplexer once the carry-in is known.

* Improvement in Speed.

Let us consider the RCA; for an n-bit adder, the computation time is given by $T = nk_1$, where k_1 is the delay through one adder cell.

→ for, Carry Select Adder, if the adder is divided into blocks, with each block containing two adder cells, then

$$T = 2k_1 + (n-2)k_2$$

where k_2 is the delay through the mux.

→ Hence it is observed that, if not planned properly, then 'k_2' may ~~practically~~ there may not be any great advantage.

Let the n-bit adder be divided into 'M' blocks & each block contain 'p' adder cells in series i.e. $n = MP$ (ex 16-bit → 2 blocks → each block has 8 adder cells). The completion time (T) is the sum of the propogation delay through the first block & propogatn delay through the mux es.

i.e. $T = Pk_1 + (M-1)k_2 \quad -(1)$

To obtain minimum T eq① has to be differentiated w.r. to 'M' & the result to be equated to zero.

$n = MP$

$P = n/M$

i.e. $\frac{dT}{dM} = \frac{d}{dM}\left\{\frac{n}{M}k_1 + (M-1)k_2\right\} = 0$

$nk_1\left(-1/M^2\right) + k_2 - 0 = 0$

(or) $\frac{nk_1}{M^2} = k_2 \Rightarrow \boxed{M = \sqrt{\frac{nk_1}{k_2}}}$

Ex:- for a 64-bit CSA, given $k_1 = 4ns$ & $k_2 = 1ns$. find the no. of blocks & no. of adder cells in each block for achieving min. T.

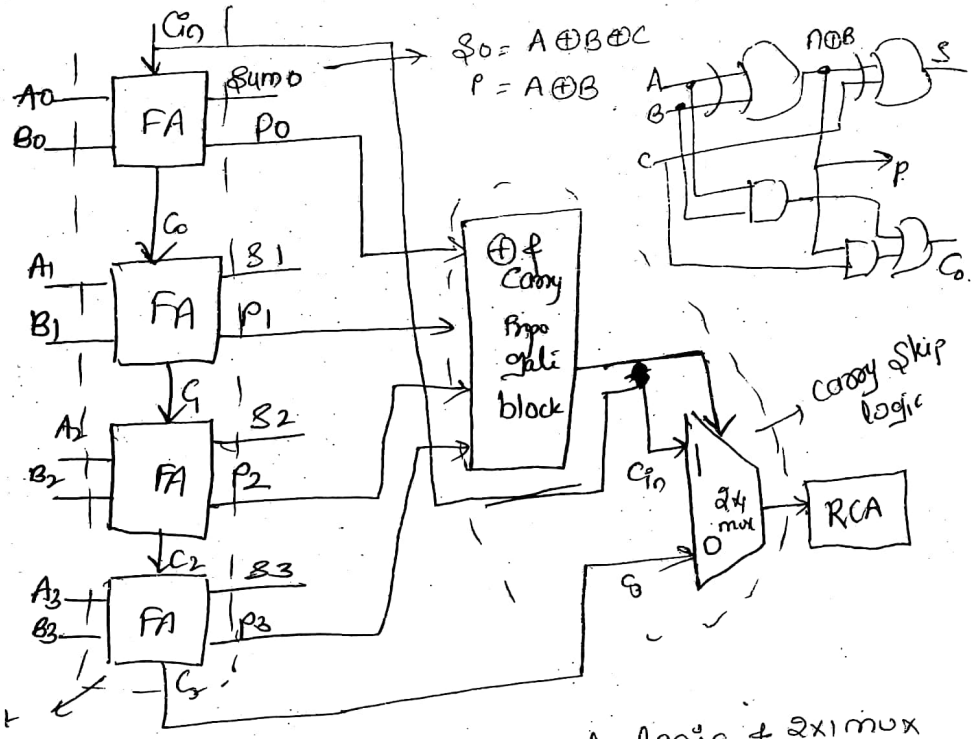
Given $n = 64$, $k_1 = 4ns$, $k_2 = 1ns$

$M = \sqrt{\frac{64 \times 4}{1}} = \sqrt{256} = 16$

$P = \frac{64}{16} = 4$, $T = 16 + 15 = 31ns$

* Carry Skip Adder: / carry-by-pass Adder.

4-bit carry Skip Adder.



Carry Skip Logic:- Using carray propogate logic & 2x1 mux carry is skipped of carry is propogated to next RCA block.

-> carry propogate:- $\overline{BP} = (A_3 \oplus B_0)(A_1 \oplus B_1)(A_1 \oplus B_2)(A_3 + B_3)$

block → if $P = 1$, then no need to wait for 'C_3' generation directly passed C_{in} to next block which eliminates carry generation time i.e. C_3

Eg:-

$A = 1010$

$B = 0110$

$\overline{BP} = P_0 \cdot P_1 \cdot P_2 \cdot P_3$

$= 0$

$\therefore$ Wait for 'C_3' generation

$C_2 \; S \; P \; C_0$

$P = A \oplus B \Rightarrow P_0 - 0$

$P_1 - 0$

$P_2 - 1$

$P_3 - 1$

Eg: 1010
0101
1111
0

$C_{out} = C_{in}$ $P = A \oplus B$

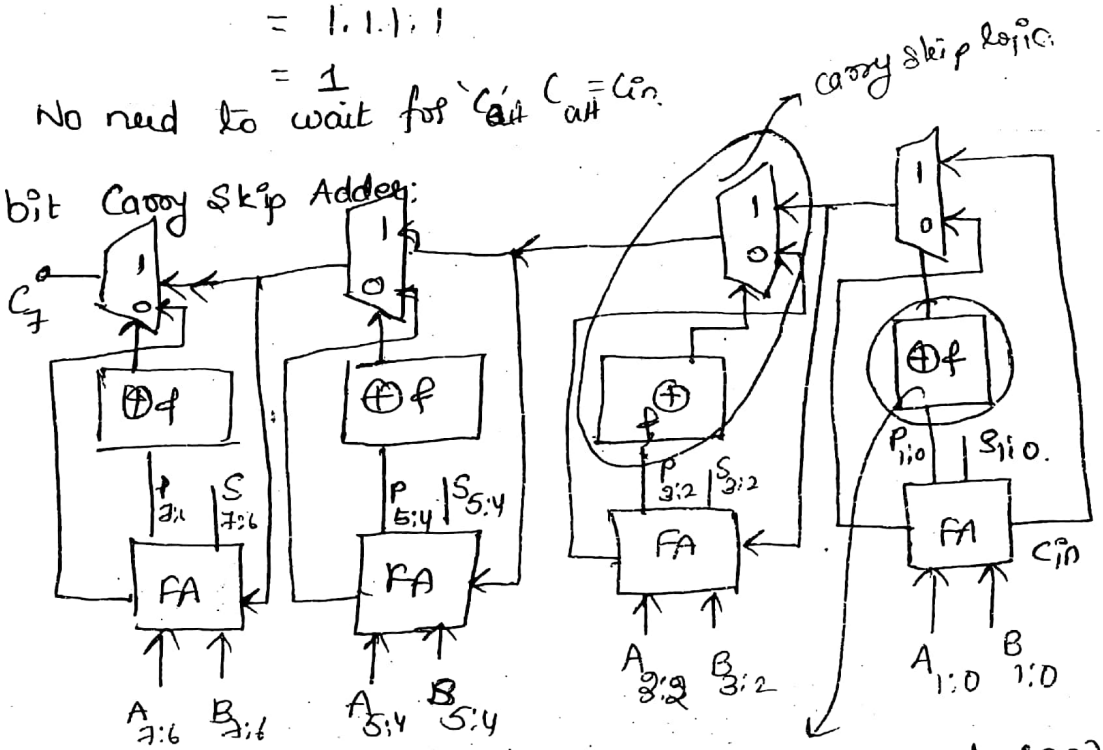
$BP = (A_0 \oplus B_0)(A_1 \oplus B_1)(A_2 \oplus B_2)(A_3 \oplus B_3)$

$= 1.1.1.1$

$= 1$

No need to wait for C_{out} $C_{out} = C_{in}$

* 8-bit Carry Skip Adder:



This adder makes use of the "block propogate" signals.

> In a block, if the bits of the operand A & B are different in all the positions w.r.to each other, then carry doesn't get generated.

Ex:- A = 01101011

B = 10010100

$BP = (A_0 \oplus B_0)(A_1 \oplus B_1) \ldots (A_7 \oplus B_7)$

$= 1.1.1.1.1.1.1.1$

$\therefore C_{out} = C_{in}$ $\therefore$ In such cases carry need not get propogate at all.

> Thus, to save this propogation time a special signal called as "block propogation" is used

> This is defined as ($BP = \prod P_i$) & if this signal is '1', then carry-in need not be propogated through the block

> Instead it can be directly transmitted through a mux, to the next block

Thus, the propogation delay gets minimized & speed of the adder gets increased.

* Case if BP = 0, then it indicates that there can be generation of carry & hence the i/p carry needs to be propogated through the block. Now to reduce, this delay, the choice of the block size becomes necessary, for which the compilation is -

Let 'k_1' be the time needed by the carry signal to propogate thro' the adder cell & 'k_2' - propogation delay of mux. Then compilation time is given by,

$$T = 2(P-1)k_1 + (M-2)k_2 \quad \text{---(1)} \qquad \boxed{n = MP}$$

$$\therefore T_{min} = \frac{dT}{dM} = \frac{d}{dM}\left[2\left\{\frac{n}{M} - 1\right\}k_1 + (M-2)k_2\right] = 0$$

$$\Rightarrow 2n\left\{-1/M^2\right\}k_1 + k_2 = 0 = 0$$

$$\frac{2nk_1}{M^2} = k_2$$

$$\boxed{M = \sqrt{\frac{2nk_1}{k_2}}}$$

$T_2 - 1$ .. Wait T'