Adder

Carry Look-Ahead Adder

- Main drawback of ripple-carry adder: (RCA) :

– The total delay is proportional to the number of bits n.

– Performance degradation for large values of n.

– The main bottleneck is the carry, which propagates sequentially from one stage to the next.

- How this can be overcome?

– Generate all the carry bits in parallel before the actual addition starts.

– After the initial delay, all the additions can be done in parallel.

Why RCA is slow?

Because S_3 Depends on C_3

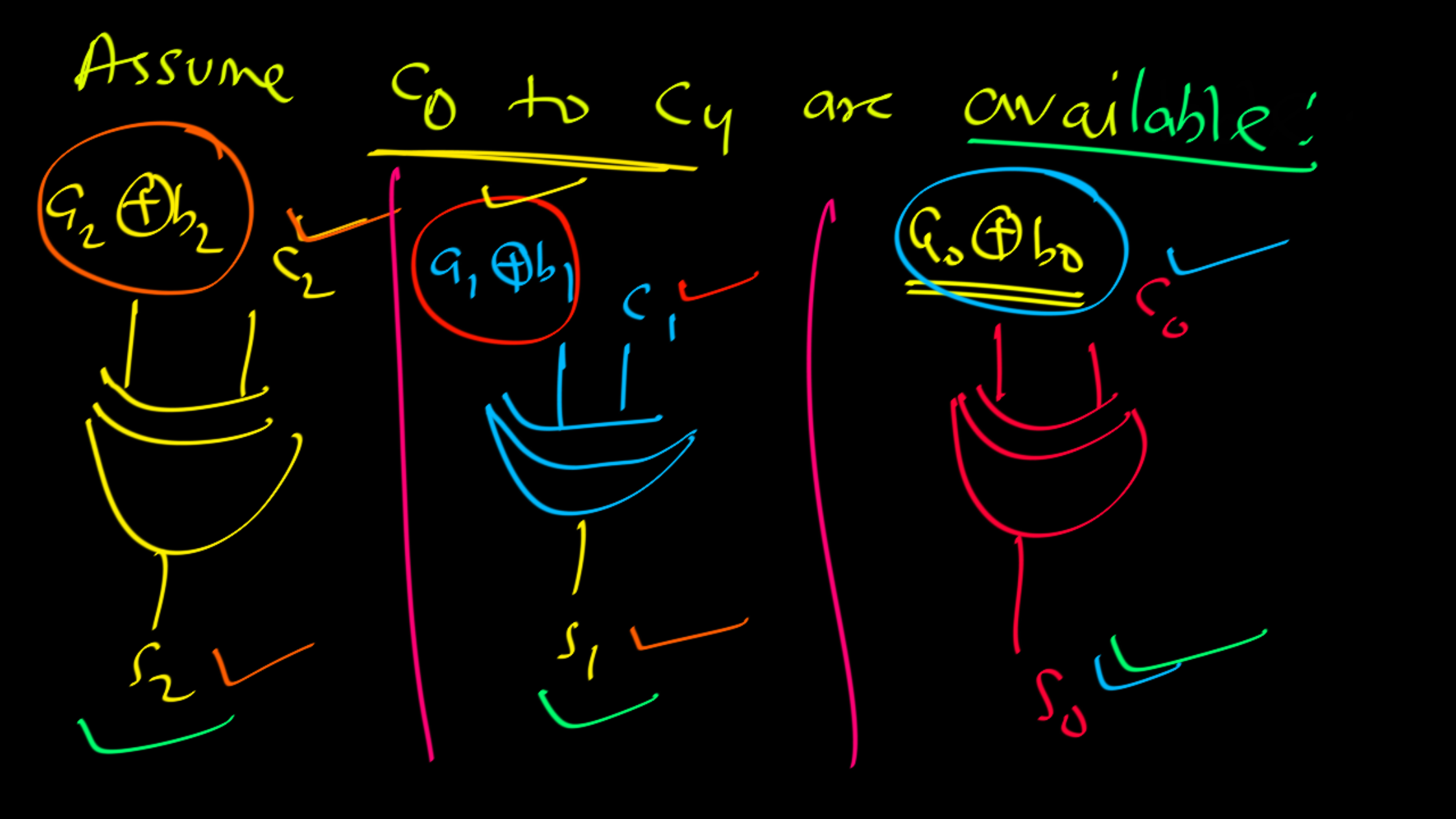
C_3 Depends on C_2

C_2 Depends on C_1

C_1 Depends on C_0

How to make our Adder fast?

Calculate Carry of each adder Directly. initially; first Calculate Carry of each adder: Now all carry are available.


Assume
c_0 to c_4 are
available!
$a_2 \oplus b_2$
c_2
s_2
$a_1 \oplus b_1$
c_1
s_1
$a_0 \oplus b_0$
c_0
s_0

Carry Look-ahead Adder (CLA)

Carry Look-ahead Adder (CLA)

- The propagation delay of an n-bit ripple carry order has been seen to be proportional to n.

– Due to the rippling effect of carry sequentially from one stage to the next.

- One possible way to speedup the addition.

– Generate the carry signals for the various stages in parallel.

– Time complexity reduces from O(n) to O(1).

– Hardware complexity increases rapidly with n.

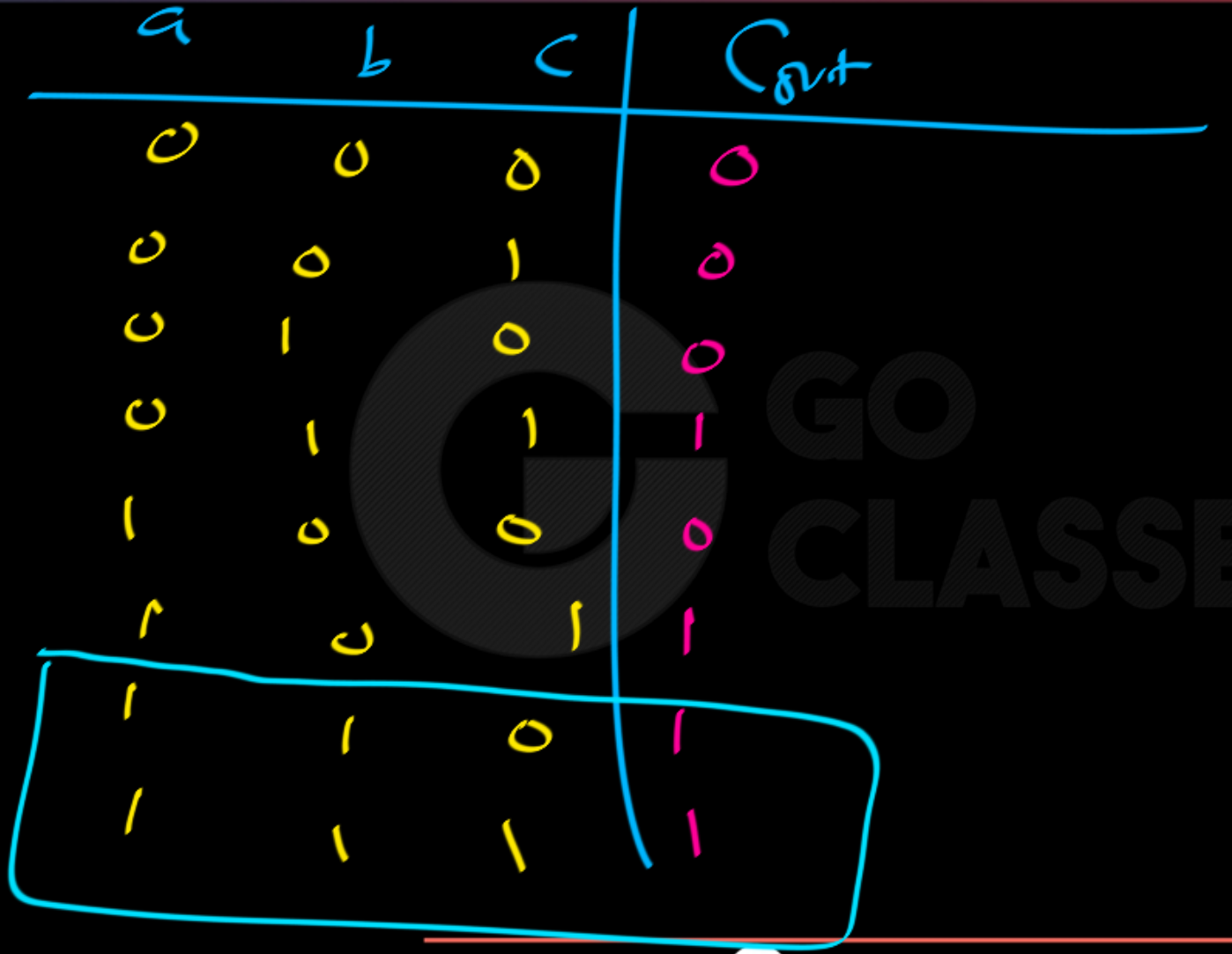
Initially, we have :

$a_3\ a_2\ a_1\ a_0$

$b_3\ b_2\ b_1\ b_0$

c_0

Now we will create a separate Circuit for Carry-Calculation


a
b
c
Cout

$a=1$; $b=1$

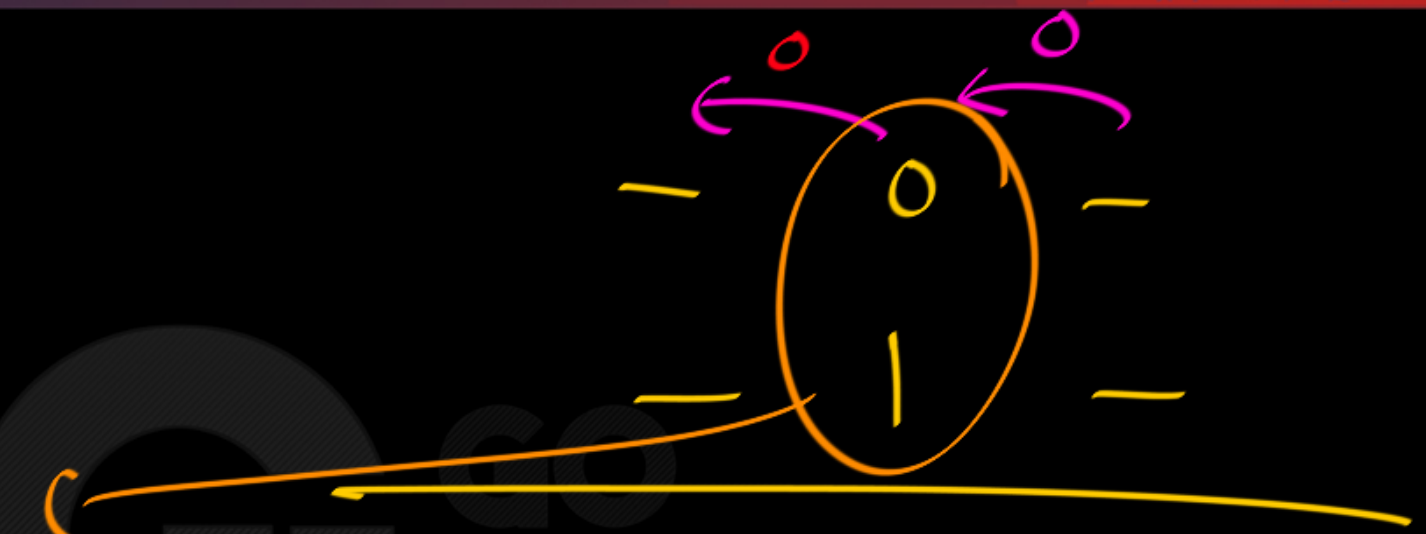
Carry Generator

$G = ab$

↓

Carry Generate

Generating Carry



Simply propagates incoming carry from previous stage to the next stage.

Simply propagates the incoming carry (from previous stage) to next stage.

$a=0$; $b=1$

or

$a=1$; $b=0$

these combinations are called carry propagator.

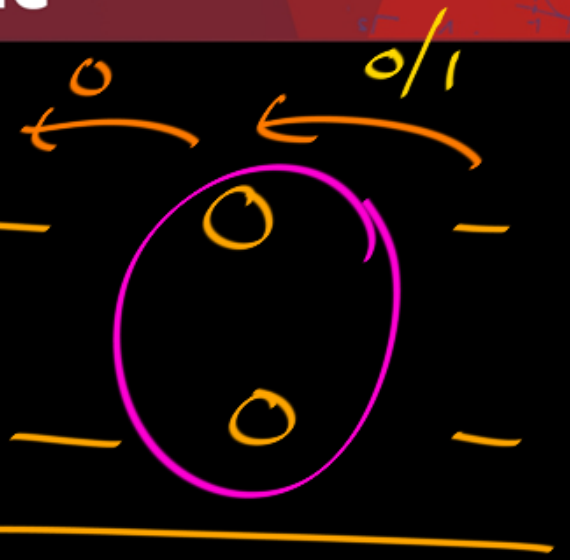
$P = a \oplus b$

↓

carry propagate

$a = 0$; $b = 0$

Carry killer



Carry kill $= \bar{a}\,\bar{b}$

So we Define two new variables:

Carry Generate $= \overline{G} = ab$

" Propagate $= \overline{P} = a \oplus b$

Note: To Calculate G, P, we Don't have to wait for anything.

$G_0 = a_0 b_0$; $P_0 = a_0 \oplus b_0$

$G_1 = a_1 b_1$; $P_1 = a_1 \oplus b_1$

$G_2 = a_2 b_2$; $P_2 = a_2 \oplus b_2$

$G_3 = a_3 b_3$; $P_3 = a_3 \oplus b_3$

We can calculate Q_i; P_i Directly.

Q: If only basic gates can be used; AND each basic gate $\Rightarrow$ delay of then Delay of P_3?

Delay of any $G_i = g_i$ b_i

If

Delay of any $P_i \Rightarrow g_i \oplus b_i$

3δ

Not → AND → OR

Initially, at $t = 0$; $\underbrace{q_i, b_i, c_0}$ available;

So after $f \Rightarrow c_i$

after $3f \Rightarrow p_i$

Q: If variables available in original, Complementary form : then

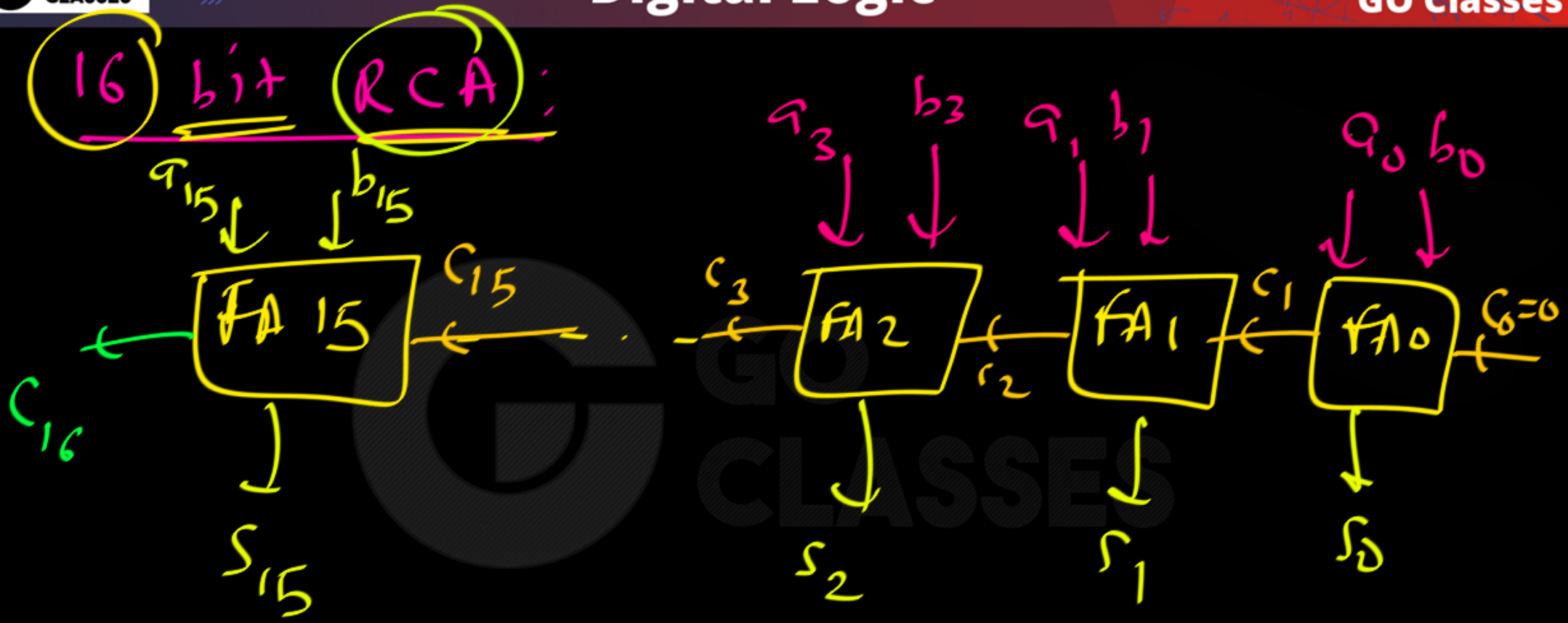
Delay for $q_i \Rightarrow 1f$

" " $p_i \Rightarrow 2f$

Q: If every logic gate have delay 1;

Delay of G_i $\Rightarrow$ 1 (AND)

" " P_i $\Rightarrow$ 1 (Ex-OR)


16 bit RCA :
a_{15}
b_{15}
FA 15
C_{15}
C_{16}
S_{15}
a_3
b_3
a_1
b_1
a_0
b_0
C_3
FA 2
C_2
FA 1
C_1
FA 0
$C_0=0$
S_2
S_1
S_0

fast Adder:

Idea: Generate / Calculate the Carry of each FA initially. ⇒ $C_0, C_1, C_2, \ldots C_{16}$

behind Carry-look-Ahead Adder

We Define two variables;

G_i = Carry Generate = $a_i b_i$

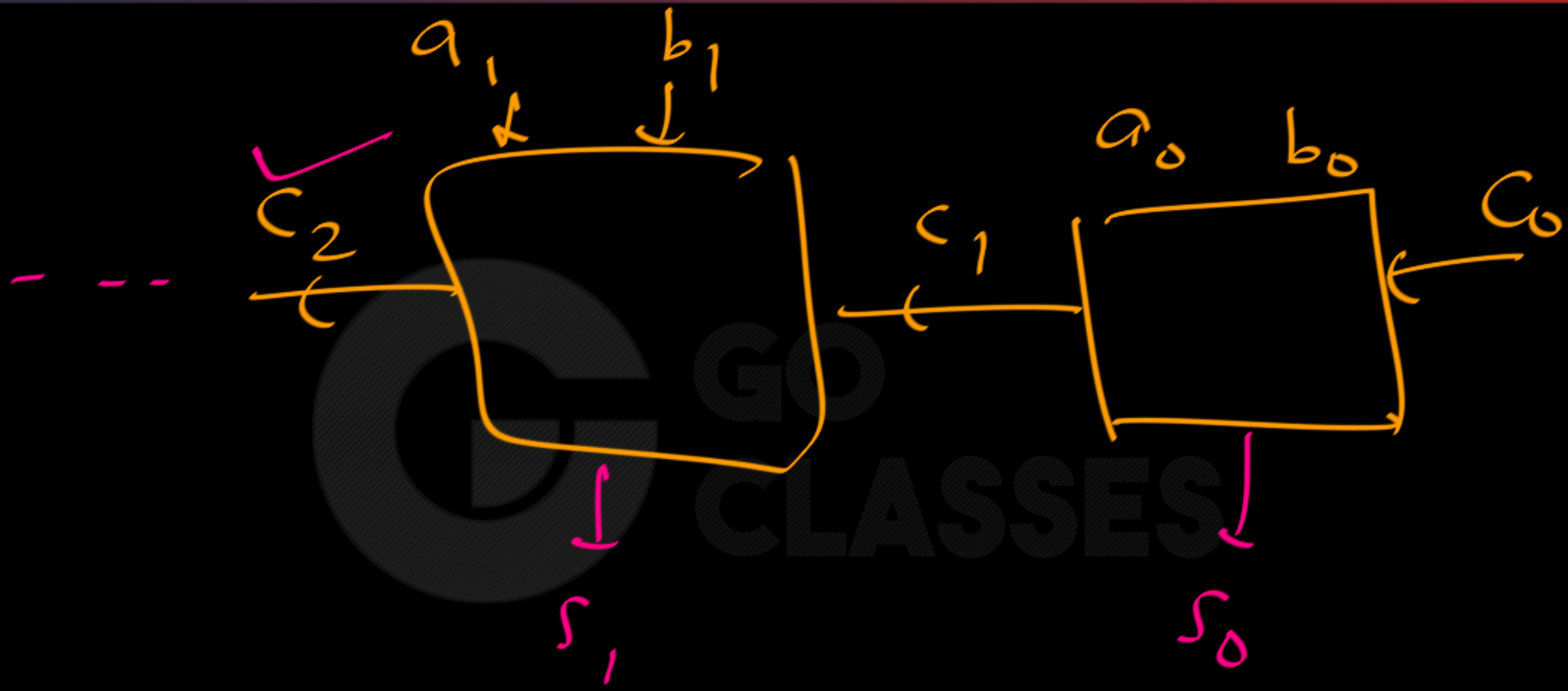
P_i = Carry Propagate = $a_i \oplus b_i$

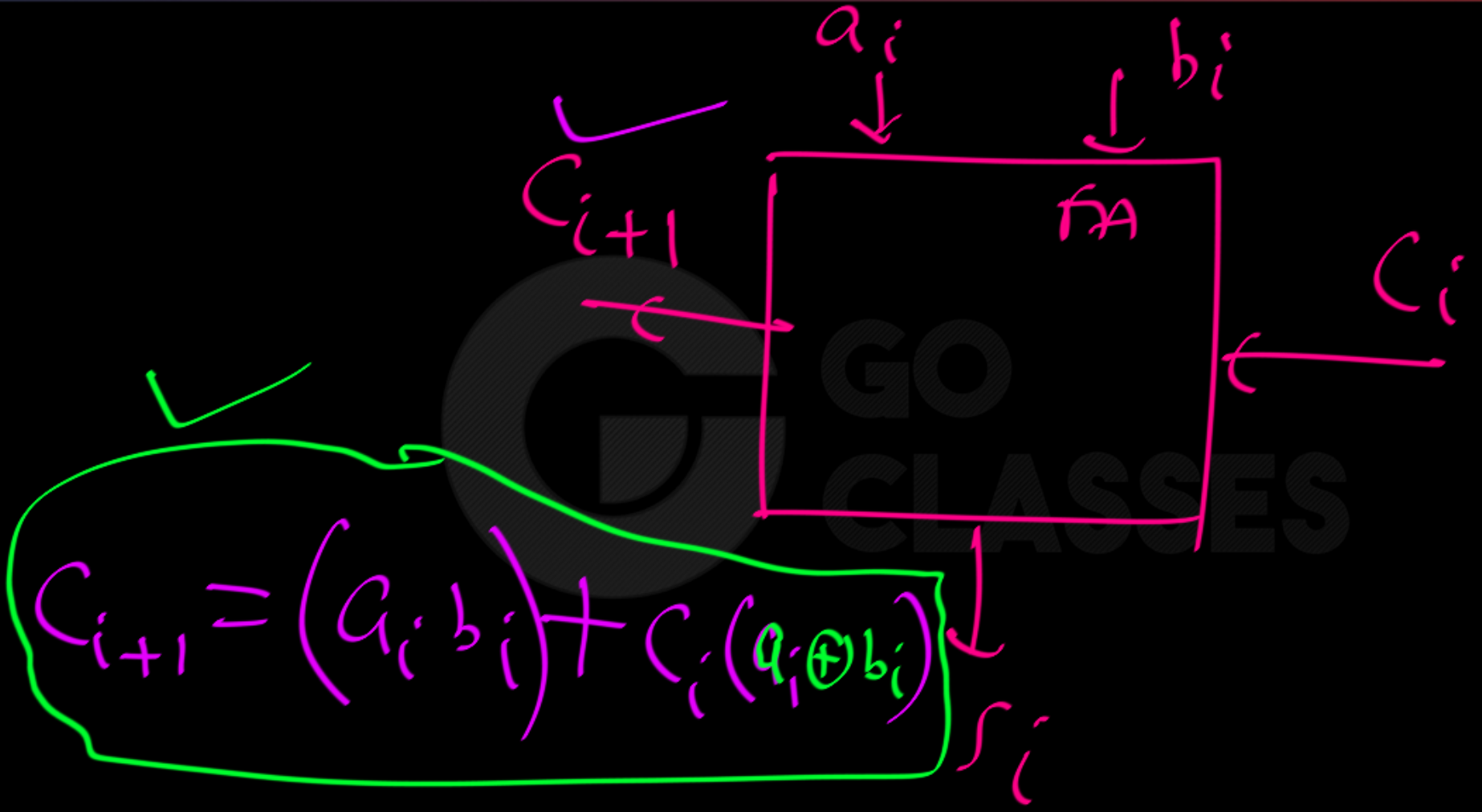
Can be calculated in the beginning.

Once we have G_i, P_i for each i:

Now we can calculate C_{i+1}

C_{i+1} = Carry out of stage i


a_1
b_1
a_0
b_0
c_2
c_1
c_0
s_1
s_0

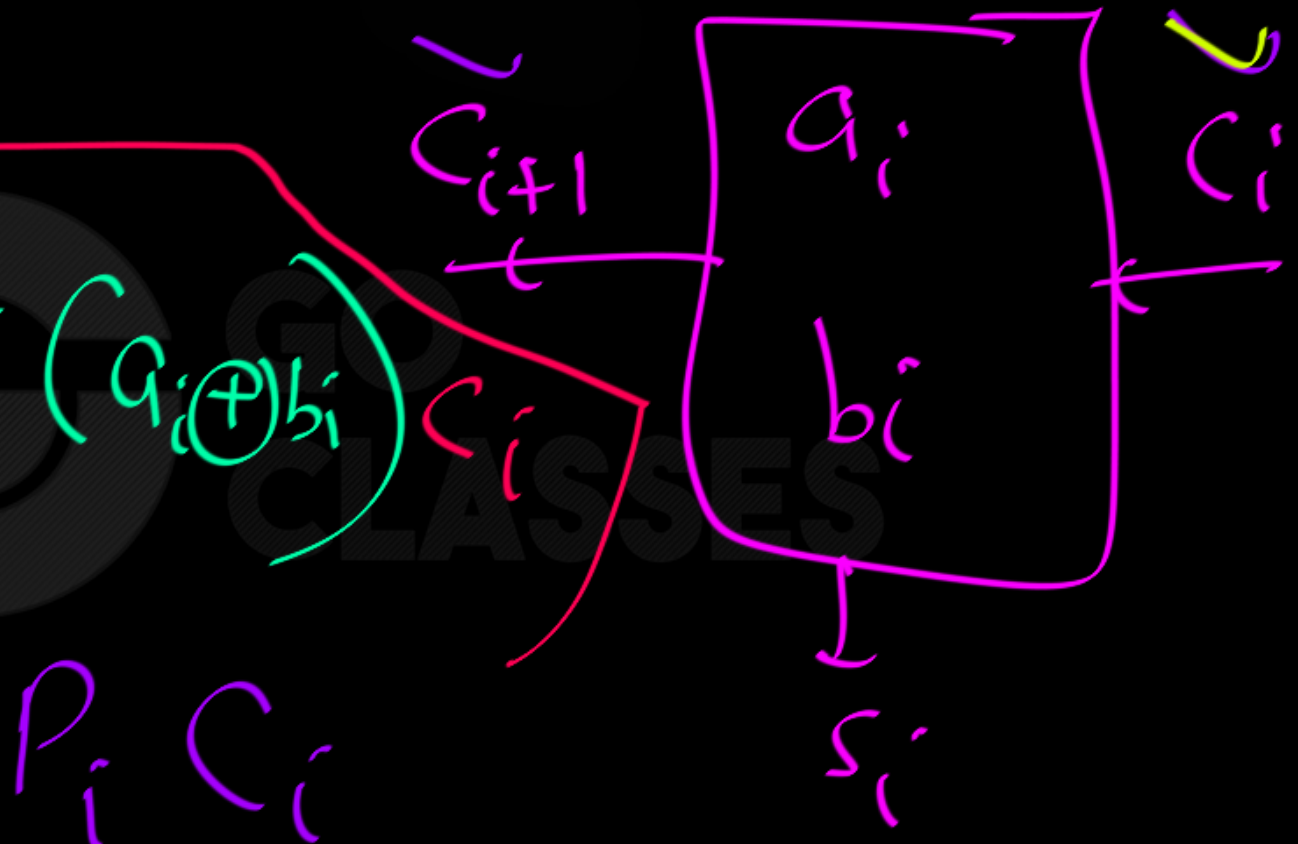

a_i
b_i
c_{i+1}
FA
c_i
s_i
$c_{i+1} = (a_i \cdot b_i) + c_i(a_i \oplus b_i)$

$$C_{i+1} = a_i b_i + c_i (a_i \oplus b_i)$$

$$C_{i+1} = g_i + c_i p_i$$

$$C_{i+1} = G_i + (a_i \oplus b_i) C_i$$

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



C_{i+1} is 1 when either a_i, b_i generate C_{i+1} (means $a_i = b_i = 1$)

OR a_i, b_i propagate C_i AND C_i is 1.

C_{i+1} is 1 when either g_i, b_i generate it OR g_i, b_i propagate C_i.

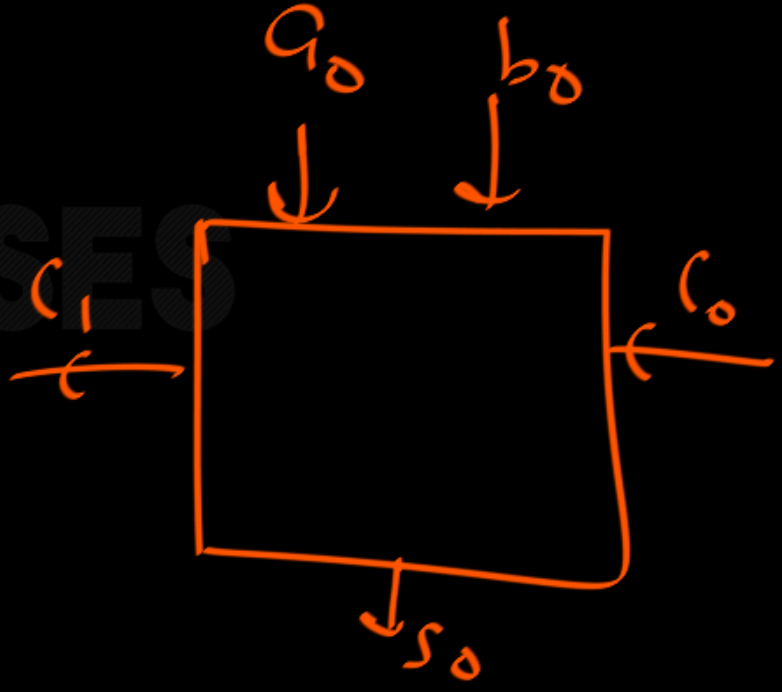
$$C_{i+1} = g_i + p_i c_i$$

initially; $a_0\ a_1\ a_2\ a_3$; $b_0\ b_1\ b_2\ b_3$; C_0

C_0 = Given initially

$C_1 = G_0 + P_0 C_0$

Can be calculated with initial Data.

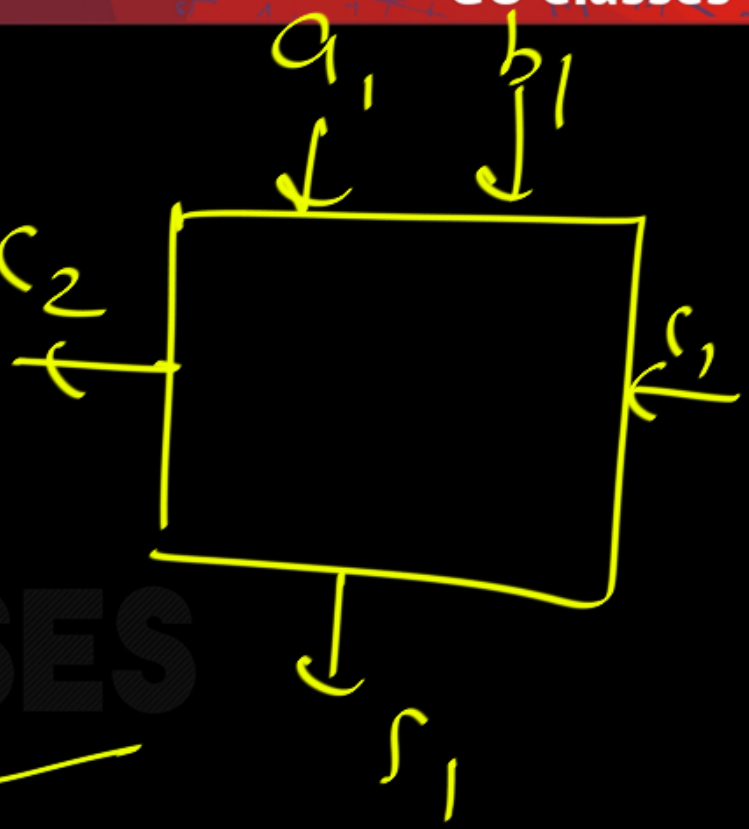


$$C_2 = G_1 + P_1 C_1$$

$$= G_1 + P_1 (G_0 + P_0 C_0)$$

$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

Can be calculated using initial Data



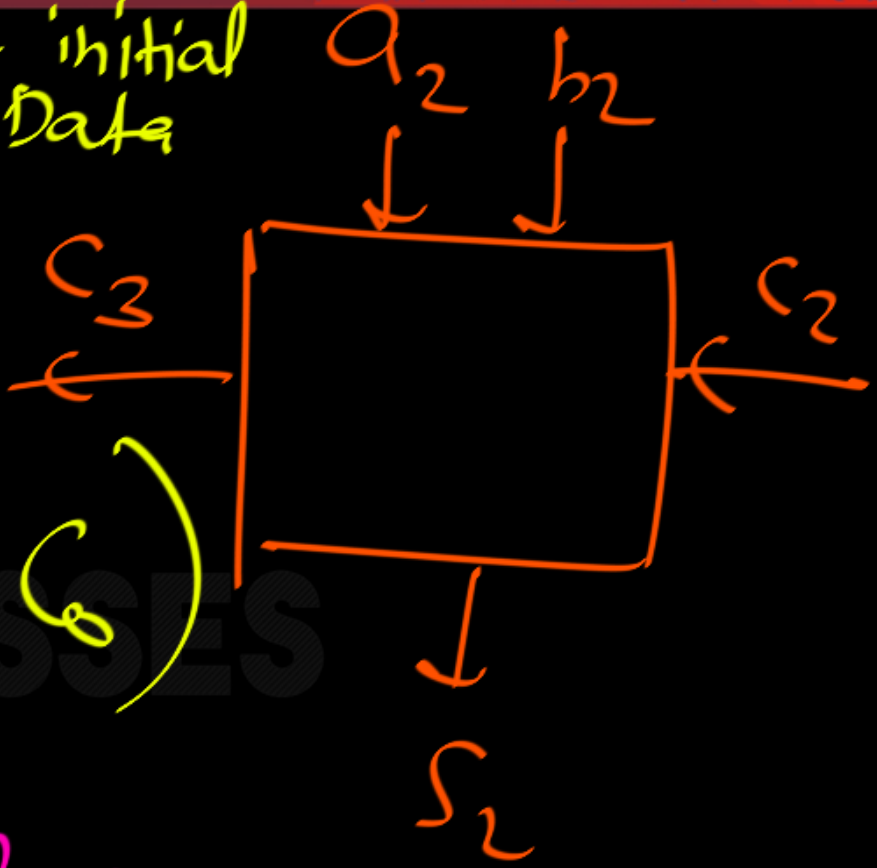
$$C_3 = G_2 + P_2 C_2$$

Not initial Data

$$= G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

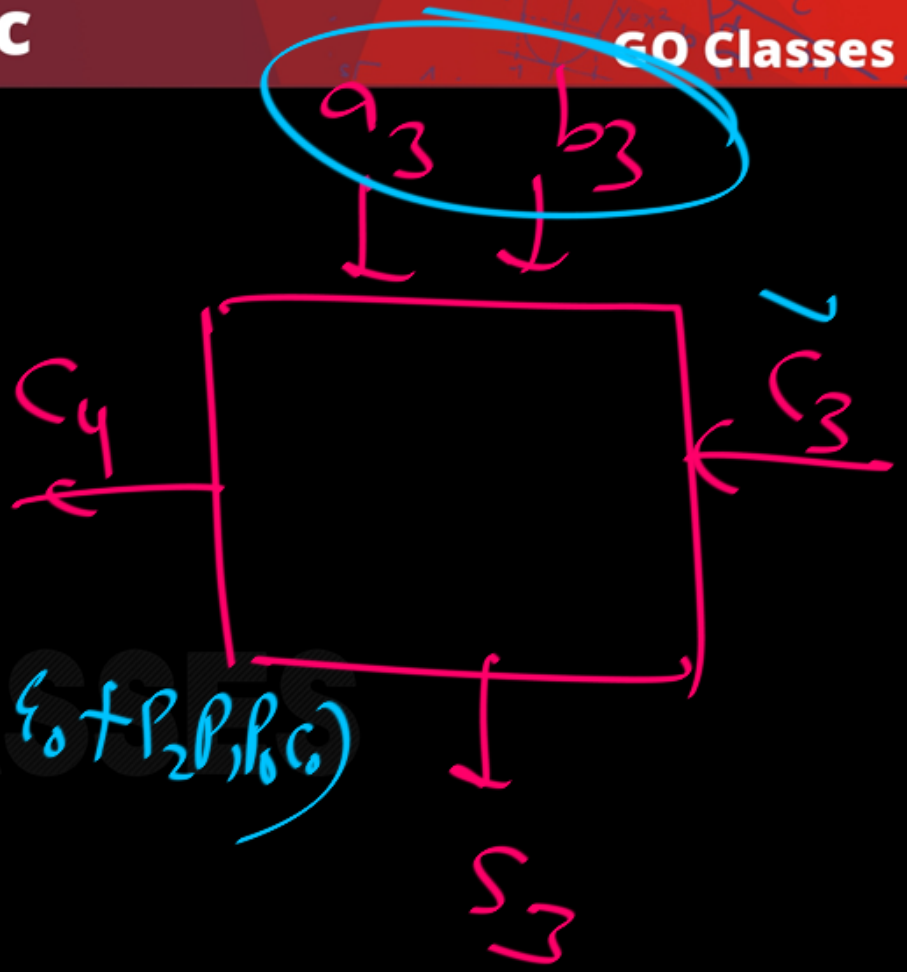
Can be calculate with initial Data



$$C_4 = G_3 + P_3 C_3$$

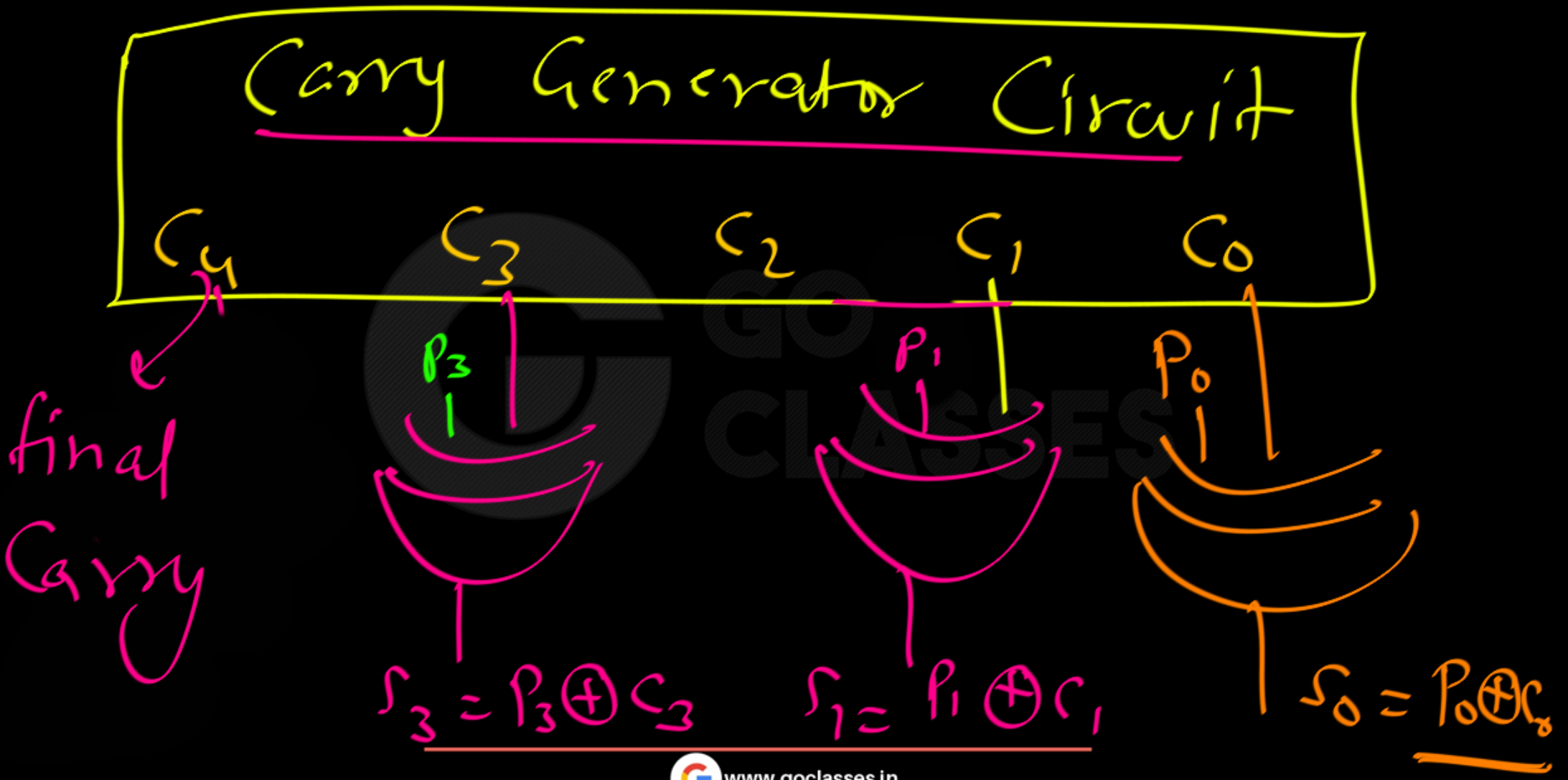
$$C_4 = G_3 + P_3(G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$$

$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$



What we have done :

We can Calculate carry of each adder from initial Data. So we Dont have to wait for anything.


Carry Generator Circuit
C4
C3
C2
C1
C0
final Carry
P3
P1
P0
S3 = P3 ⊕ C3
S1 = P1 ⊕ C1
S0 = P0 ⊕ C0

$$S_i = a_i \oplus b_i \oplus c_i$$

$$S_i = P_i \oplus c_i$$

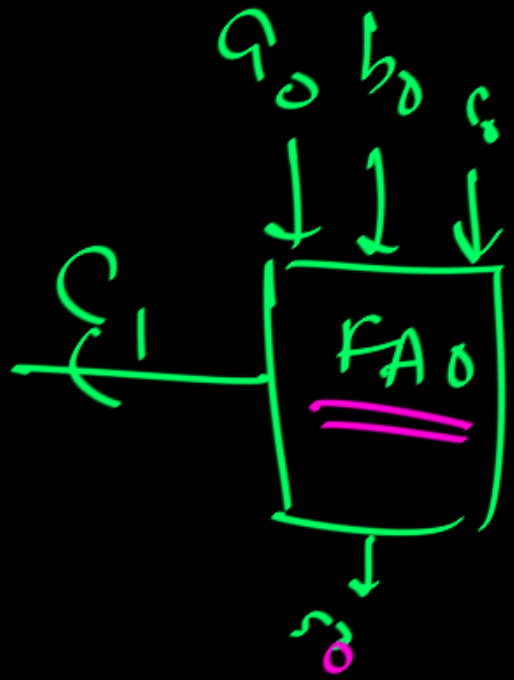
$S_0 = P_0 \oplus C_0$; $S_1 = P_1 \oplus C_1$

$S_2 = P_2 \oplus C_2$

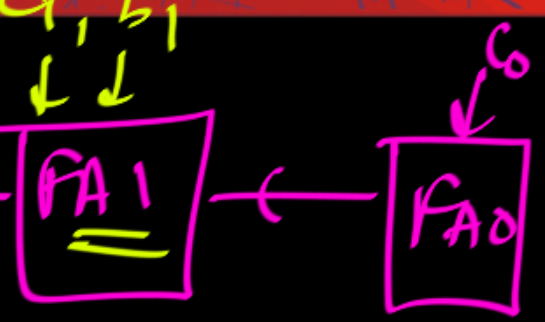
Another way of looking at the equations :

C_0 : Given initially.

$C_1 : G_0 + P_0 C_0$



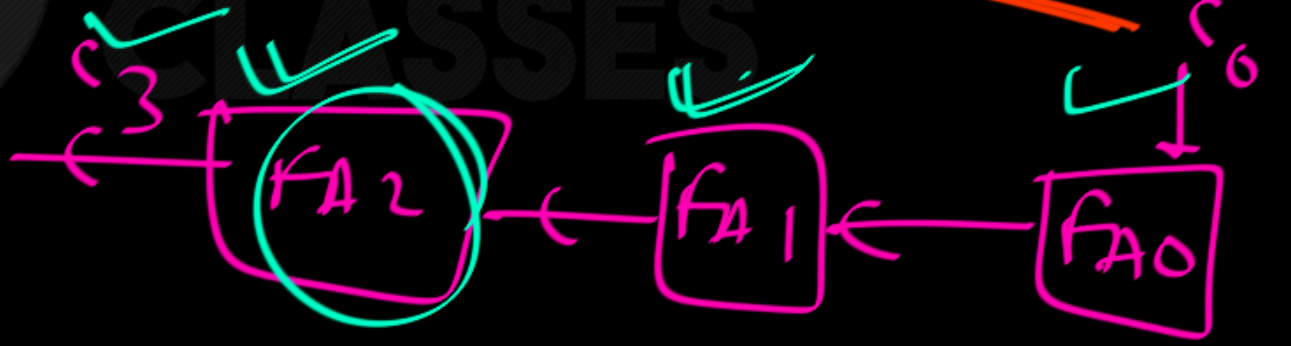
$$C_2 : G_1 + P_1 (G_0 + P_0 C_0)$$



$$: G_1 + P_1 G_0 + P_1 P_0 C_0$$

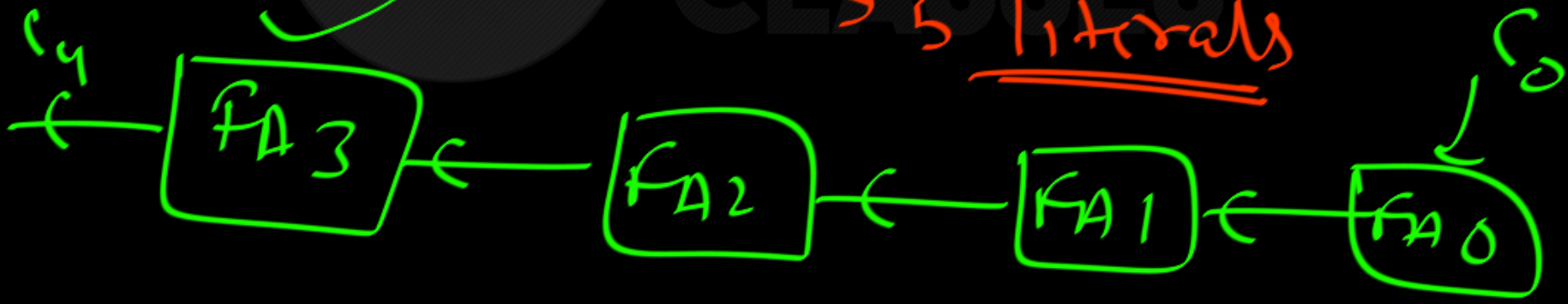
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

4 literals



$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

5 literals



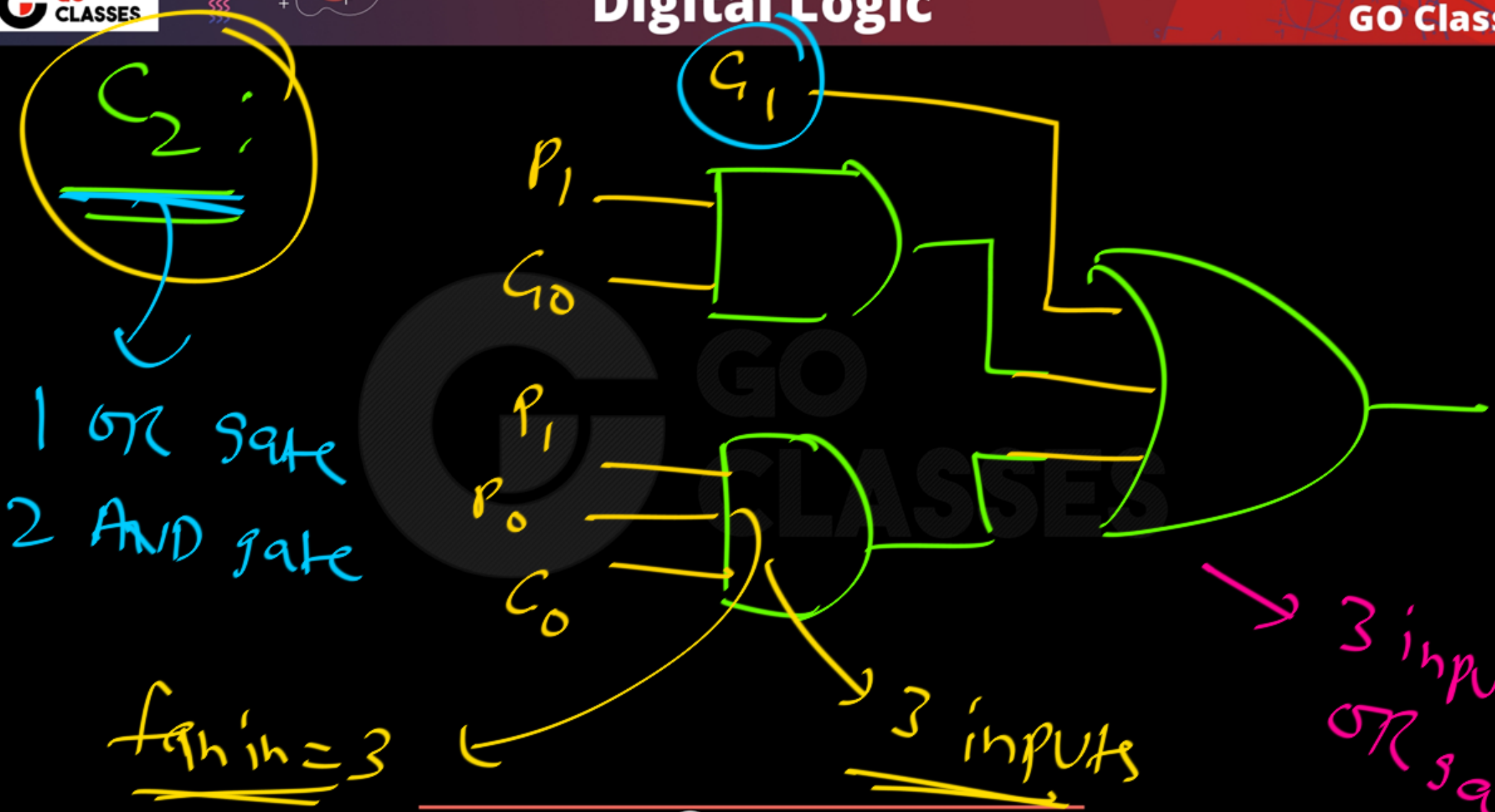
Some Observations about Carry equations

(1) C_i has $(i+1)$ product terms.

$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

3 terms

So the fan-in of OR gate requires for C_i :


C2 :
C1
P1
C0
P1
P0
C0
1 OR gate
2 AND gate
fan in = 3
3 inputs
3 inputs
OR gate

(2) for any (i, j) what is the maximum fan-in of AND gate required?

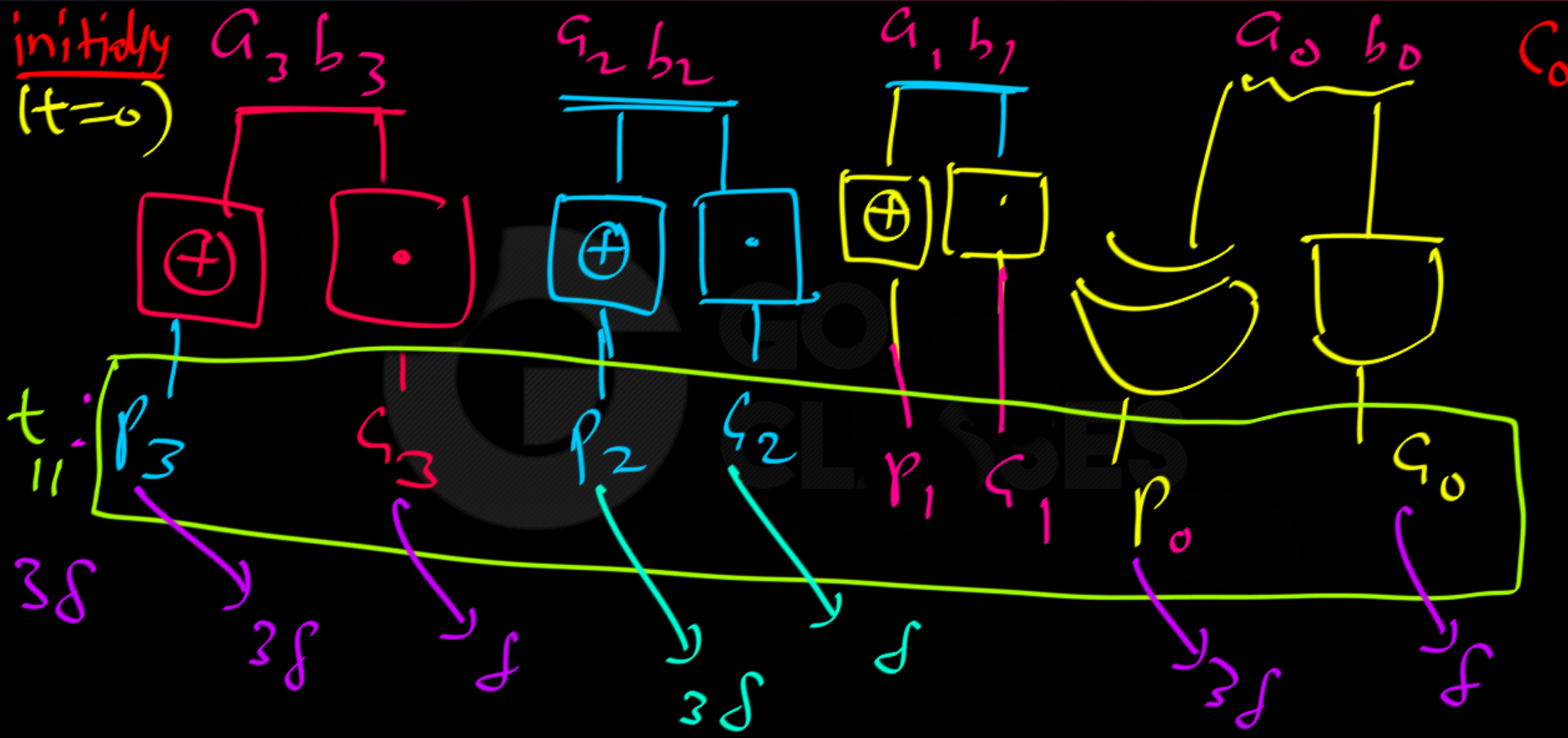
= i + 1

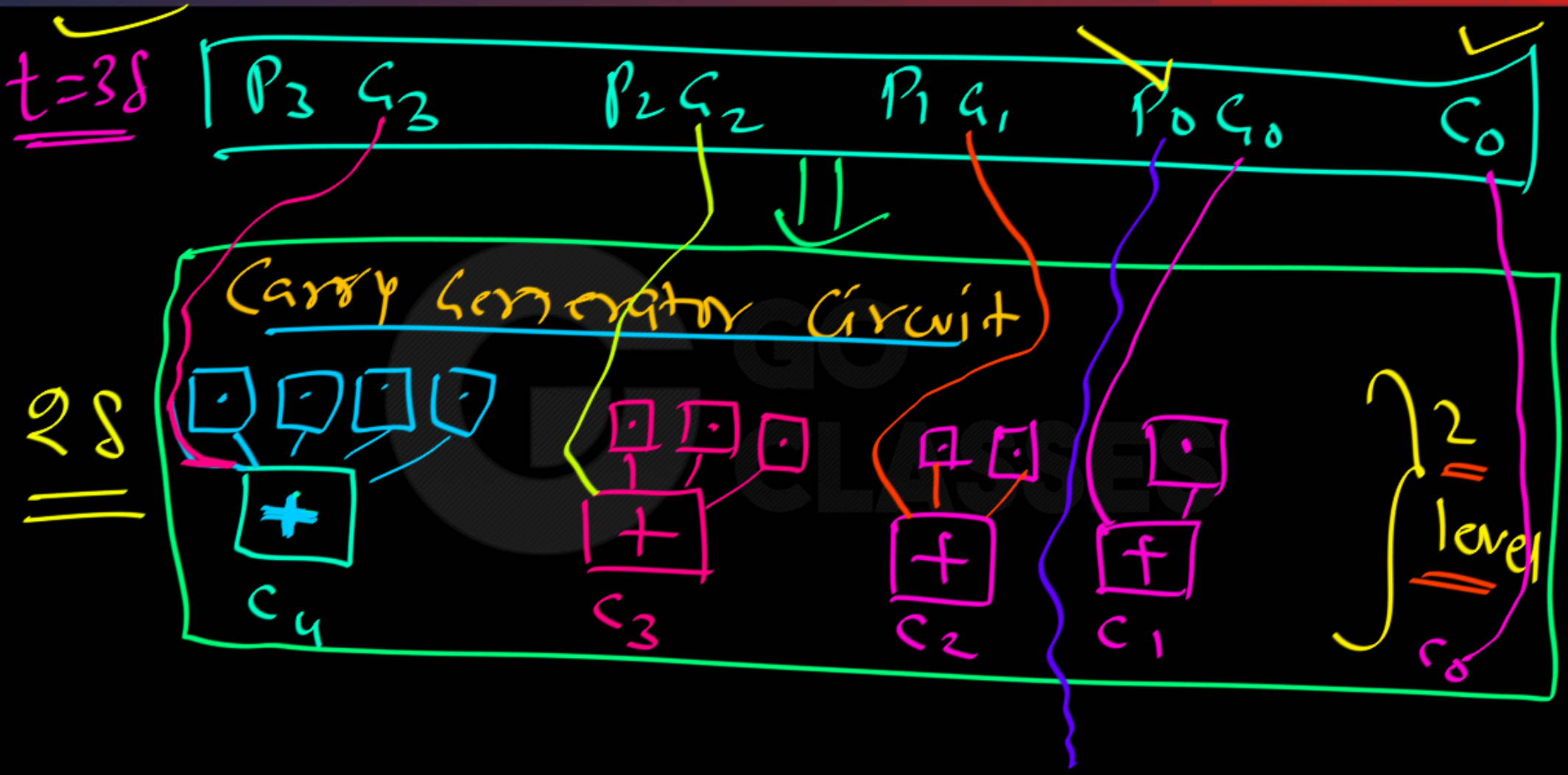
for (i, 1) OR gate
: i AND gates

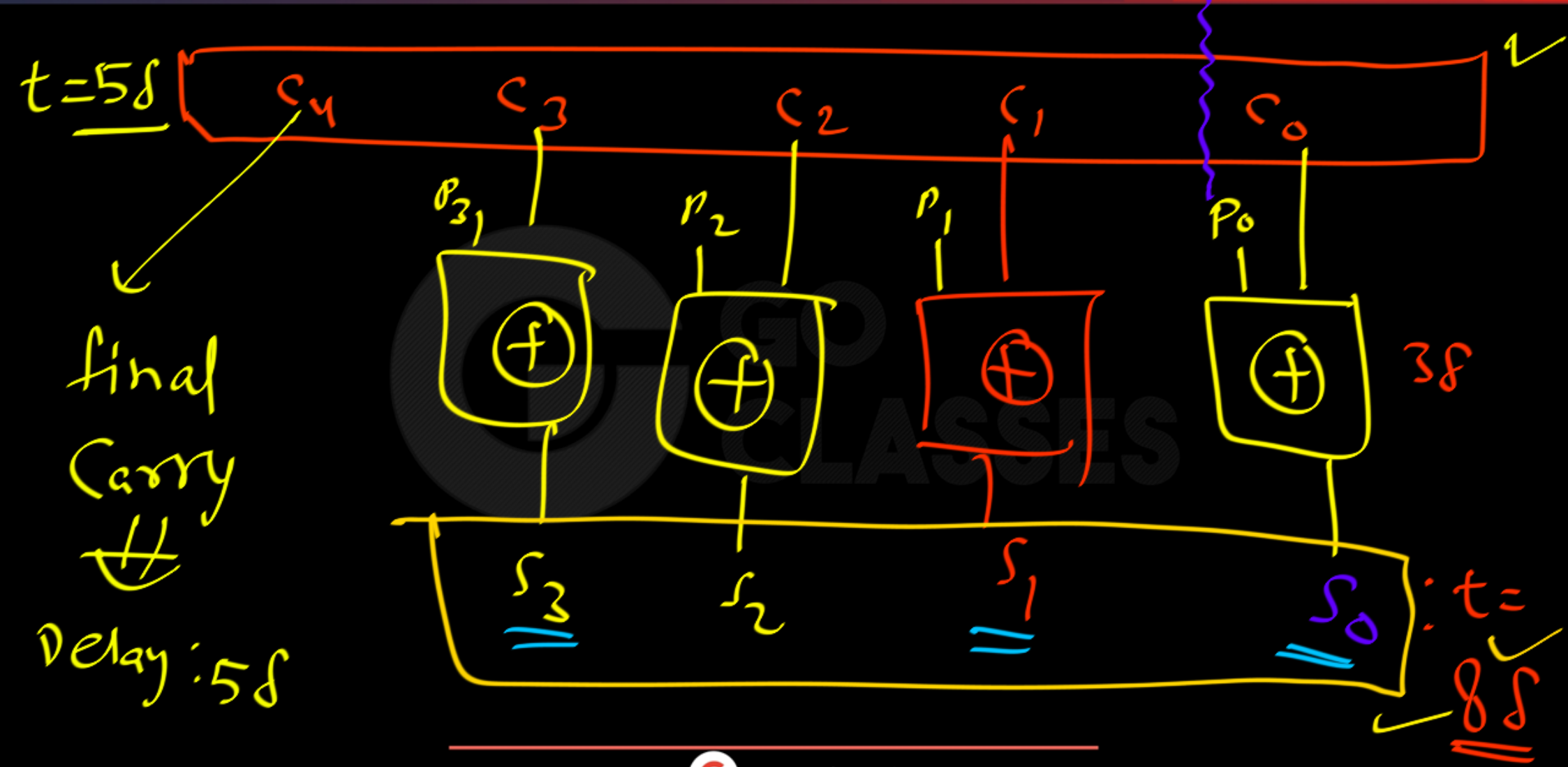
CLA adder :

1. When fan-in is unlimited (of any gate)

Assume Basic gates (AND, OR, NOT,) Delay = δ


initially
(t=0)
a_3 b_3
a_2 b_2
a_1 b_1
a_0 b_0
C_0
t = 11
P_3
G_3
P_2
G_2
P_1
G_1
P_0
G_0
3δ
3δ
δ
3δ
δ
3δ
δ


t=3δ
P3 G3
P2 G2
P1 G1
P0 G0
C0
Carry Generator Circuit
2δ
C4
C3
C2
C1
2 level
C0


t=5δ
C4
C3
C2
C1
C0
P3
P2
P1
P0
final Carry
Delay: 5δ
S3
S2
S1
S0
3δ
t=8δ

Every Carry is Available out 5g.

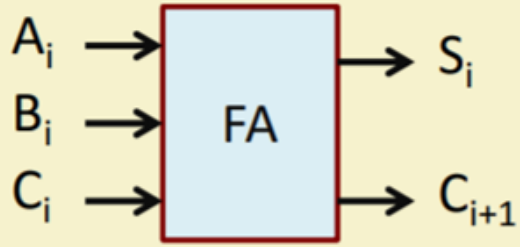
Ex OR using basic gates (OR, AND, NOT)

- Consider the i-th stage in the addition process.
- We define the *carry generate* and *carry propagate* functions as:

$$G_i = A_i.B_i$$

$$P_i = A_i \oplus B_i$$

  - G_i = 1 represents the condition when a carry is generated in stage-*i* independent of the other stages.
  - P_i = 1 represents the condition when an input carry C_i will be propagated to the output carry C_{i+1}.
- We can generate the output carry in terms of G_i and P_i.



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

(using carry lookahead logic)

$$P_i = A_i \oplus B_i$$

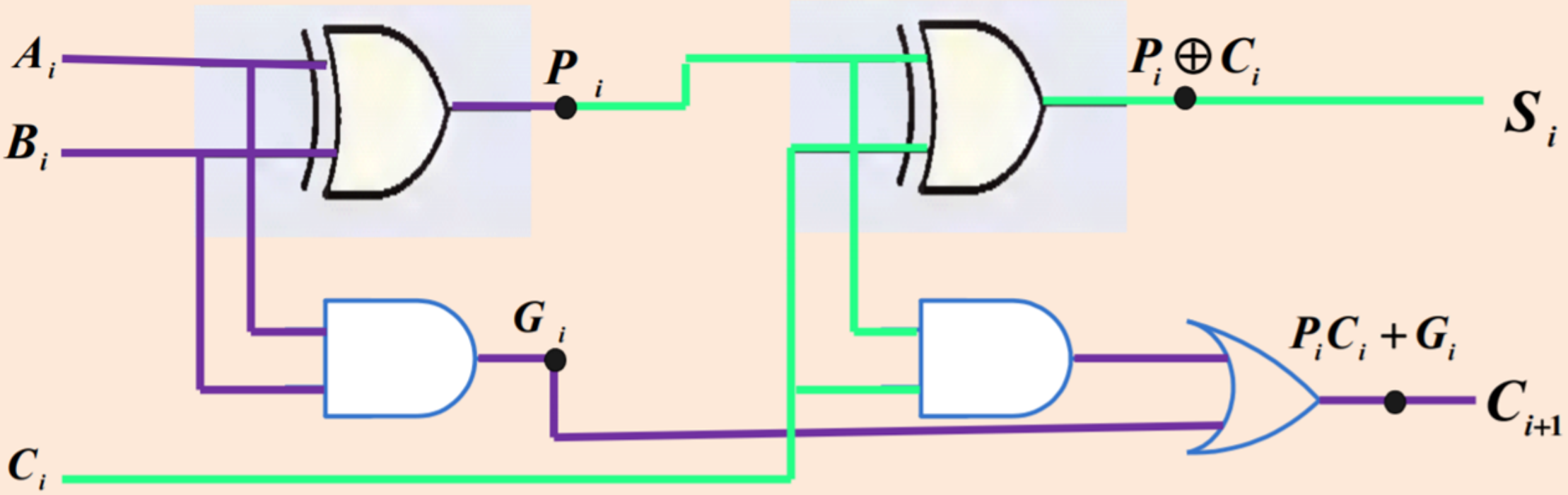
$$G_i = A_i B_i$$

The output sum and carry are:

$$S_i = P_i \oplus C_i$$

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

- ✓ G_i-called a ***carry generate***, and it produces a carry of ***1*** when both A_i and B_i are ***1.***
- ✓ P_i-called a ***carry propagate***, it determines whether a carry into stage i will propagate into stage $i + 1$.



Full Adder with P and G

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i. P_i is called a *carry propagate*, because it determines whether a carry into stage i will propagate into stage $i + 1$ (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

- ✓ The ***Boolean function*** for the carry outputs of each stage and substitute the value of each C_i form the previous equations:

$$\begin{cases} C_0 = input\ carry \\ C_1 = G_0 + P_0C_0 \\ C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) \\ \quad = G_1 + P_1G_0 + P_1P_0C_0 \\ C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) \\ \quad = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \end{cases}$$

Design of 4-bit CLA Adder

$$C_4 = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$$
$$C_3 = G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$$
$$C_2 = G_1 + G_0P_1 + C_0P_0P_1$$
$$C_1 = G_0 + C_0P_0$$

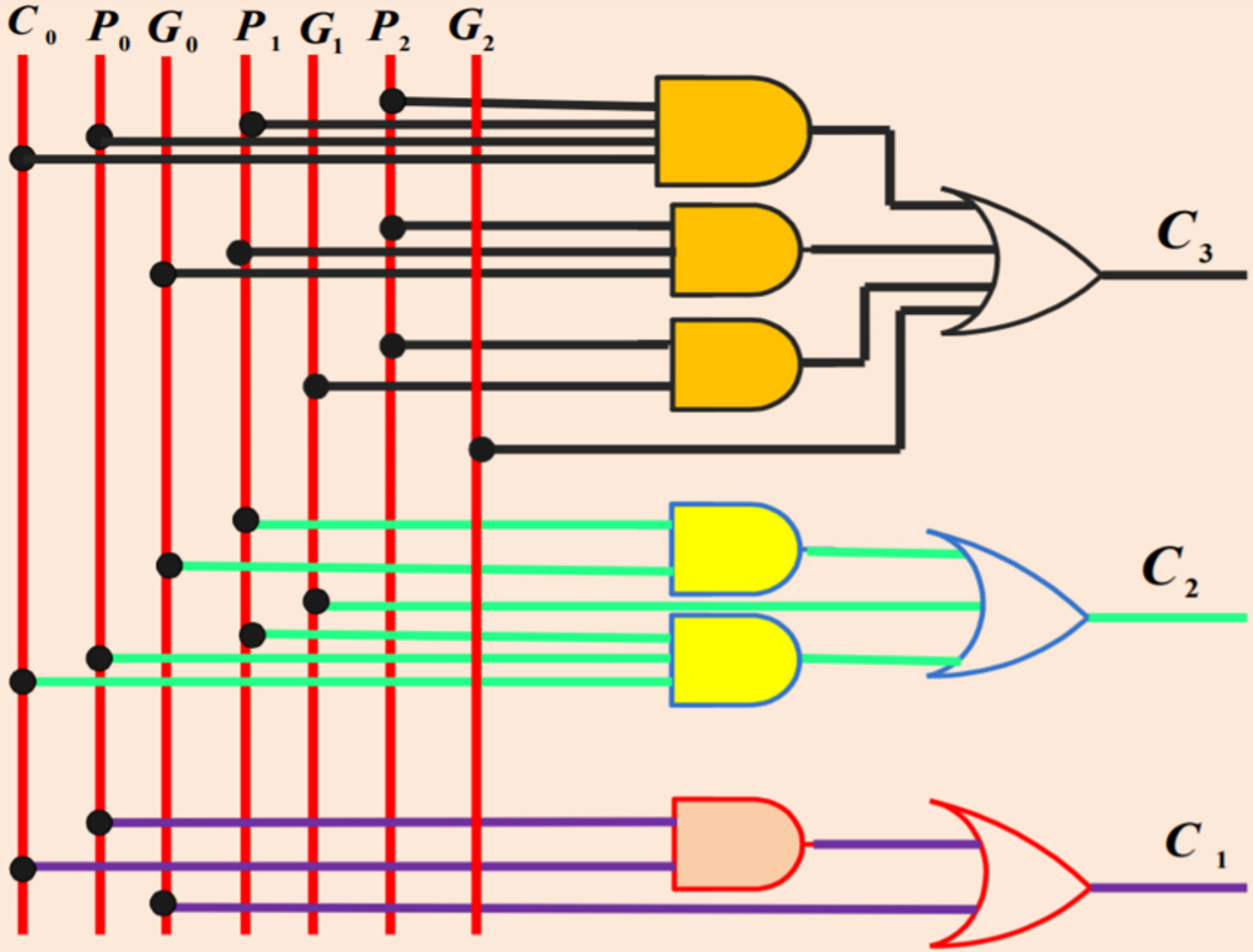
$$S_0 = A_0 \oplus B_0 \oplus C_0 = P_0 \oplus C_0$$
$$S_1 = P_1 \oplus C_1$$
$$S_2 = P_2 \oplus C_2$$
$$S_3 = P_3 \oplus C_3$$

- The three Boolean functions C_1, C_2 and C_3 are implemented in the ***carry lookahead generator.***

> ***The two level-circuit for the output carry C_4 is not shown, it can be easily derived by the equation.***

- C_3 does not have to wait for C_2 and C_1 to propagate, in fact C_3 is propagated at the same time as C_1 and C_2.

Since the Boolean function for each output carry is expressed in sum-of-products form, each function can be implemented with one level of AND gates followed by an OR gate (or by a two-level NAND).



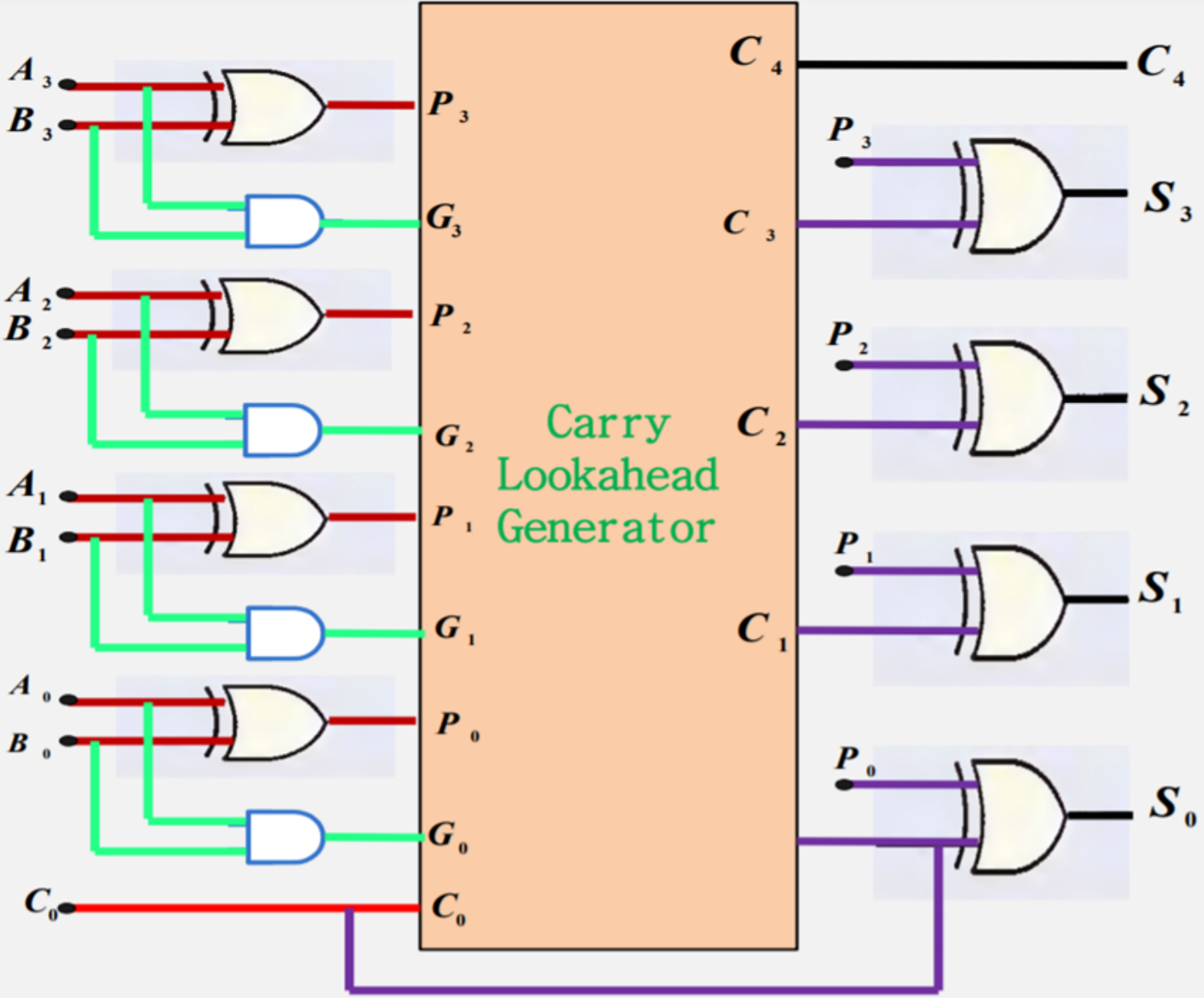
Logic Diagram for Carry Lookahead Generator

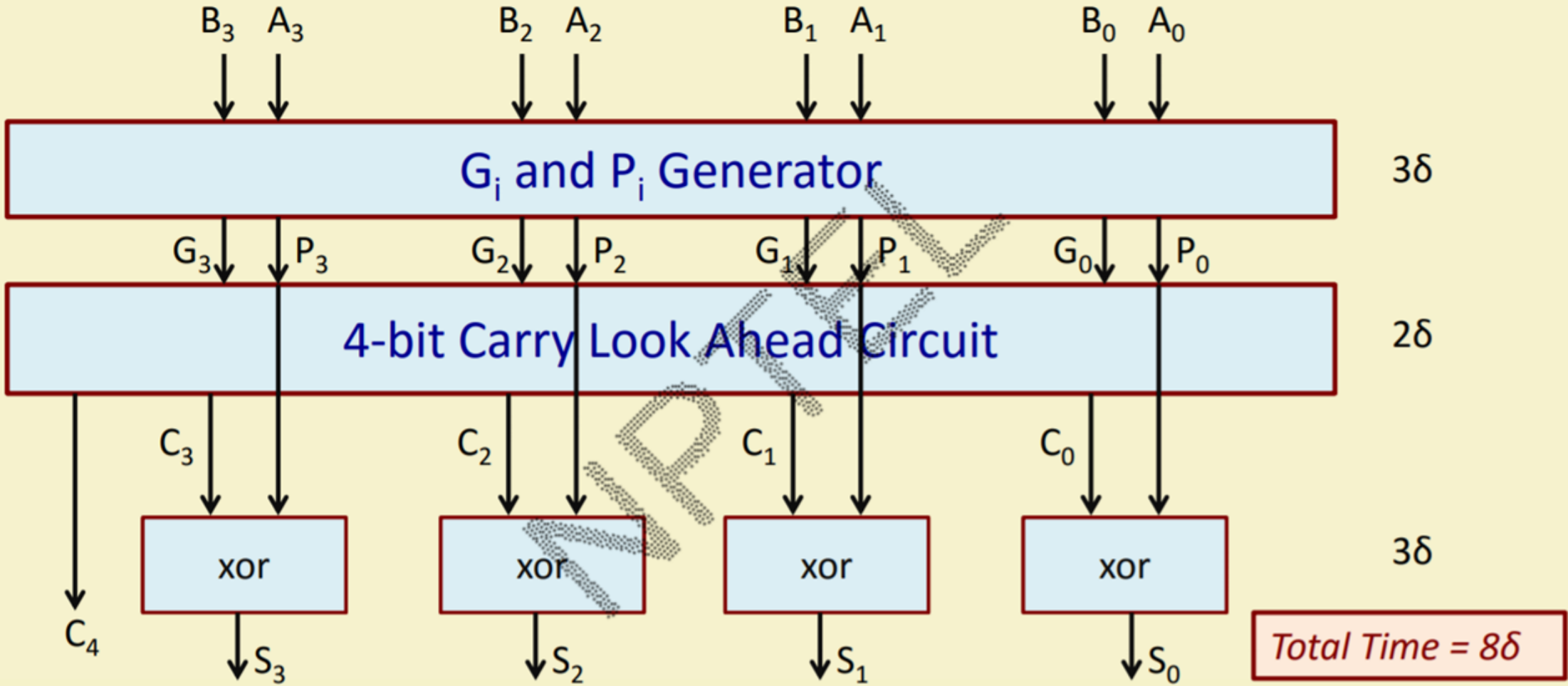
Carry Generator Circuit :

- two levels : AND — OR implementation

The construction of a four-bit adder with a carry lookahead scheme is the following:

four-bit adder with a carry look ahead scheme




B3 A3
B2 A2
B1 A1
B0 A0
Gi and Pi Generator
3δ
G3 P3
G2 P2
G1 P1
G0 P0
4-bit Carry Look Ahead Circuit
2δ
C3
C2
C1
C0
xor
xor
xor
xor
3δ
C4
S3
S2
S1
S0
Total Time = 8δ