# Chapter .2

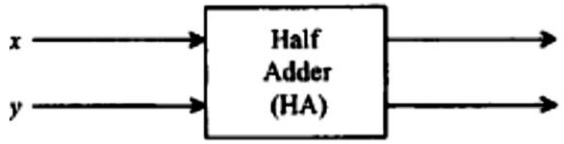
# Algorithms and Design of the Common Fixed-Point Arithmetic Operations

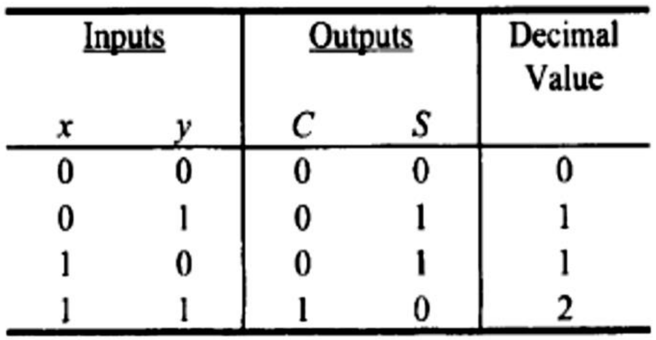
## *Addition Operation*

***Addition: is the most frequent operation performed by ALU. It also used for multiplication and division. Thus the speed of the adder unit is essential to the efficient operation of an execution unit.***

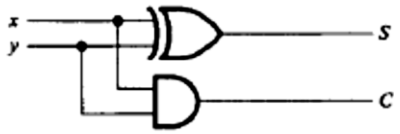
### Half Adder (HA)

Half adder circuit has two inputs: A and B, which add two input digits and generate a carry and sum.





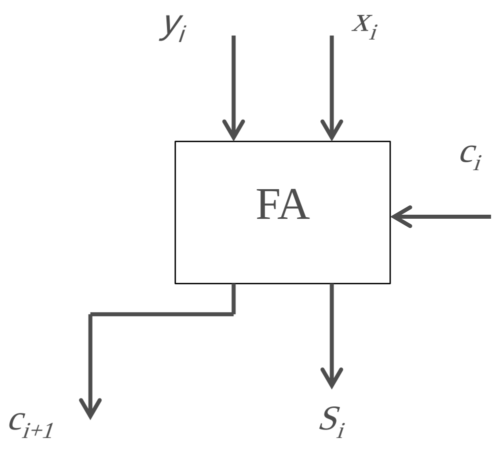
| Inputs | | Outputs | | Decimal Value |
|---|---|---|---|---|
| $x$ | $y$ | $C$ | $S$ | |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 2 |



$S = \bar{x}.y + x.\bar{y} \equiv x \quad y$ $\qquad C = x.y$

**Figure: Half-adder**

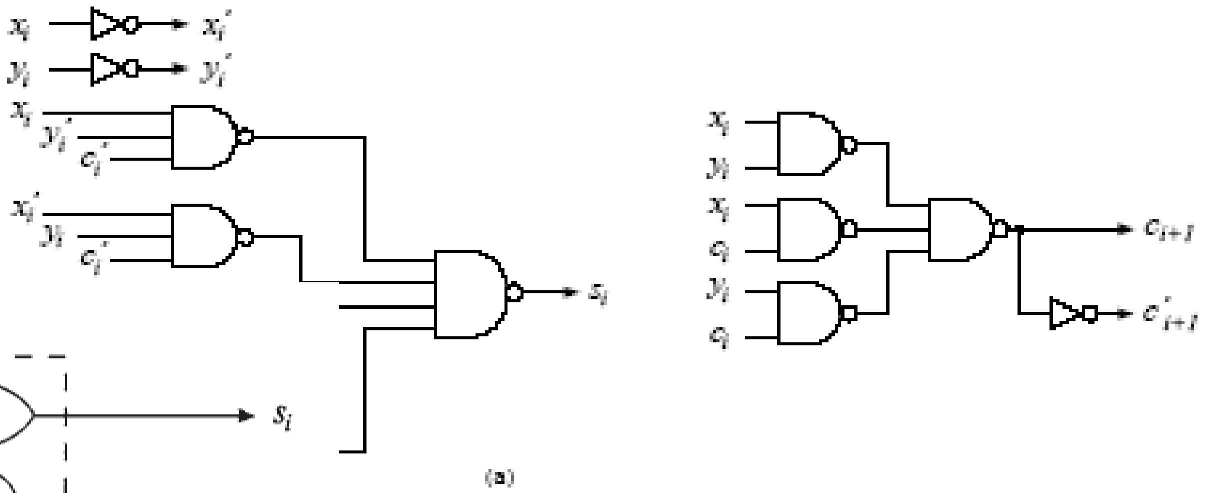
***Full Adder (FA):*** ***is a combinational digital circuit with I/P bits $x_i$ and $y_i$ and incoming carry bit $c_i$ and O/P sum bit $s_i$ and outgoing carry bit $c_{i+1}$***



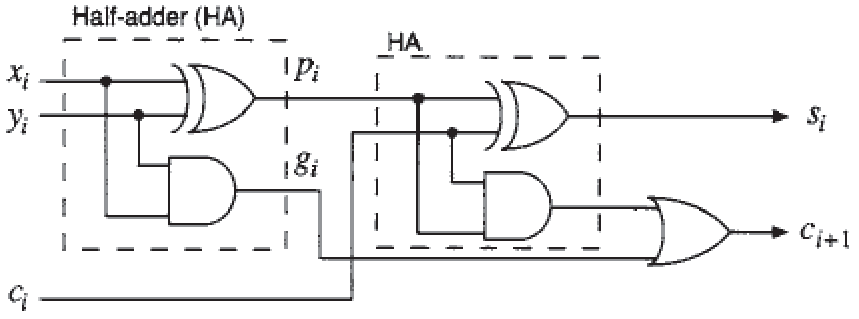
| $x_i$ | $y_i$ | $c_i$ | $c_{i+1}$ | $s_i$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

$s_i = x_i\, y_i' c_i' + x_i' y_i\, c_i' + x_i' y_i' c_i + x_i\, y_i\, c_i \equiv x_i \quad y_i \quad c_i$
$c_{i+1} = x_i\, y_i + x_i\ c_i + y_i\, c_i$

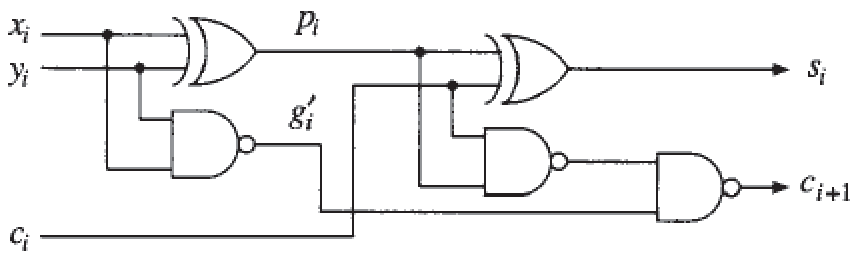
## Implementations of FA



(a)



(b)



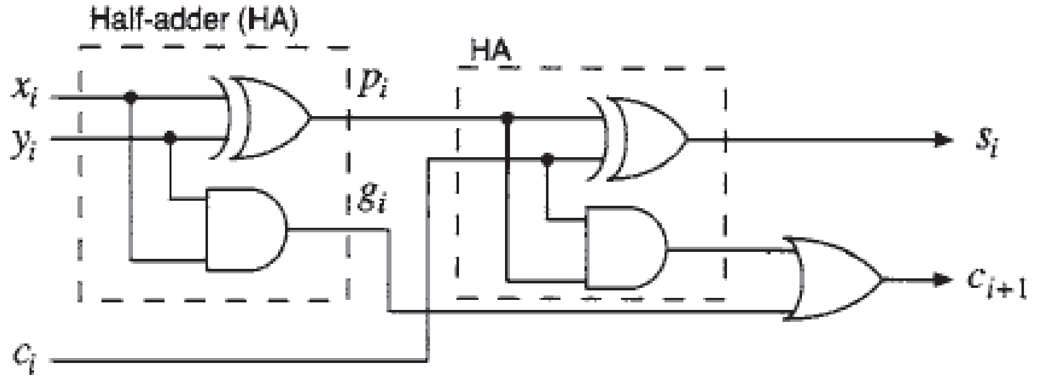
(c)

**Delay of a FA**

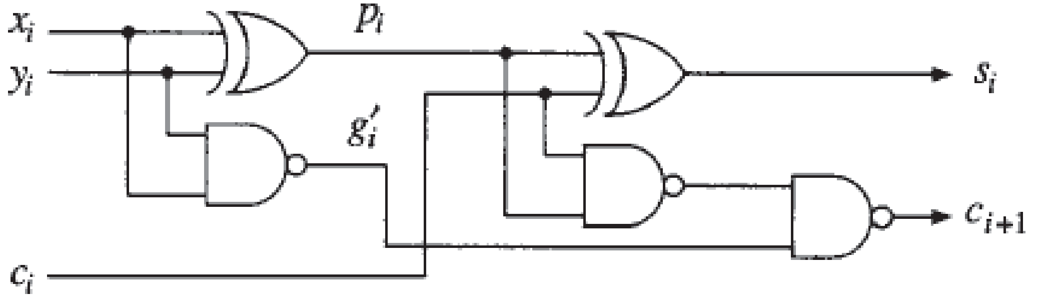
$$T_{FA} = max\ (t_{c_{i+1}},\ t_{s_i})$$

$$= max\ (2\ t_{NAND}\ ,\ t_{XOR}) + t_{XOR} = 2t_{XOR}$$

## Implementations of FA (continue)



(b)



(c)

### Delay of a FA

$$T_{FA} = max\ (t_{c_{i+1}},\ t_{s_i})$$
$$= max\ (2\ t_{NAND}\ ,\ t_{XOR}) + t_{XOR} = 2t_{XOR}$$

## *Carry Ripple (propagate) Adder (CRA) or CPA*

Ripple effect observed at sum outputs of adder until carry propagation is complete.

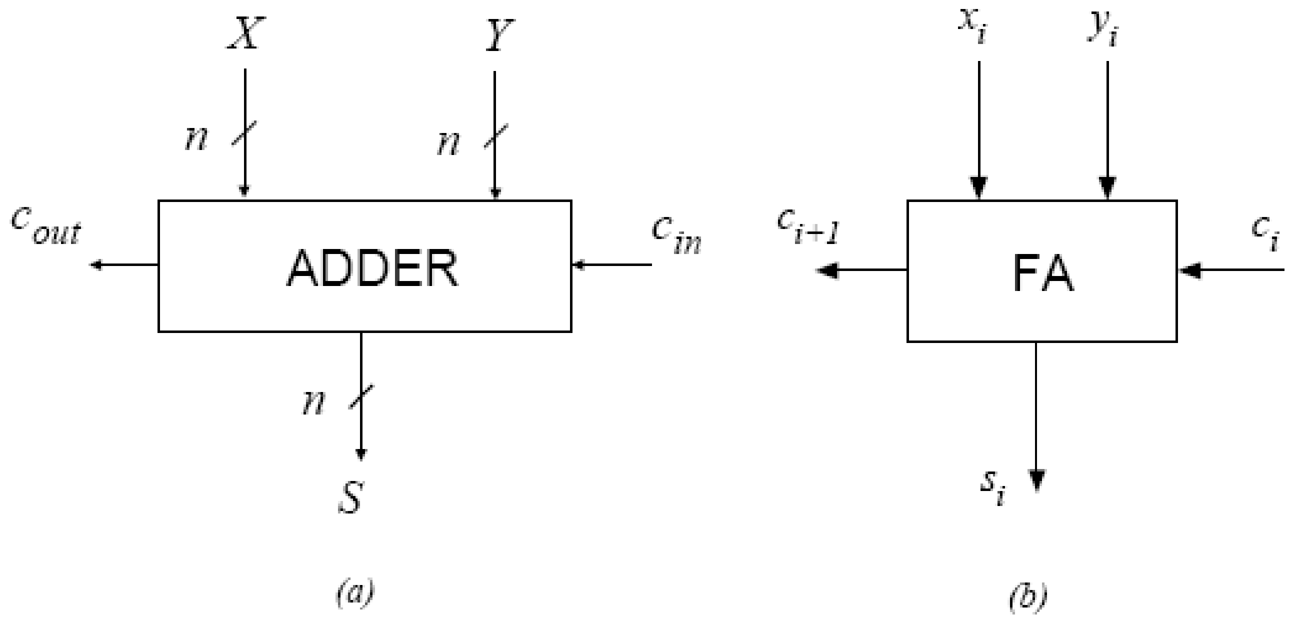


*Fig.(2) a: An n- bit adder    b: 1- bit adder (FA)*

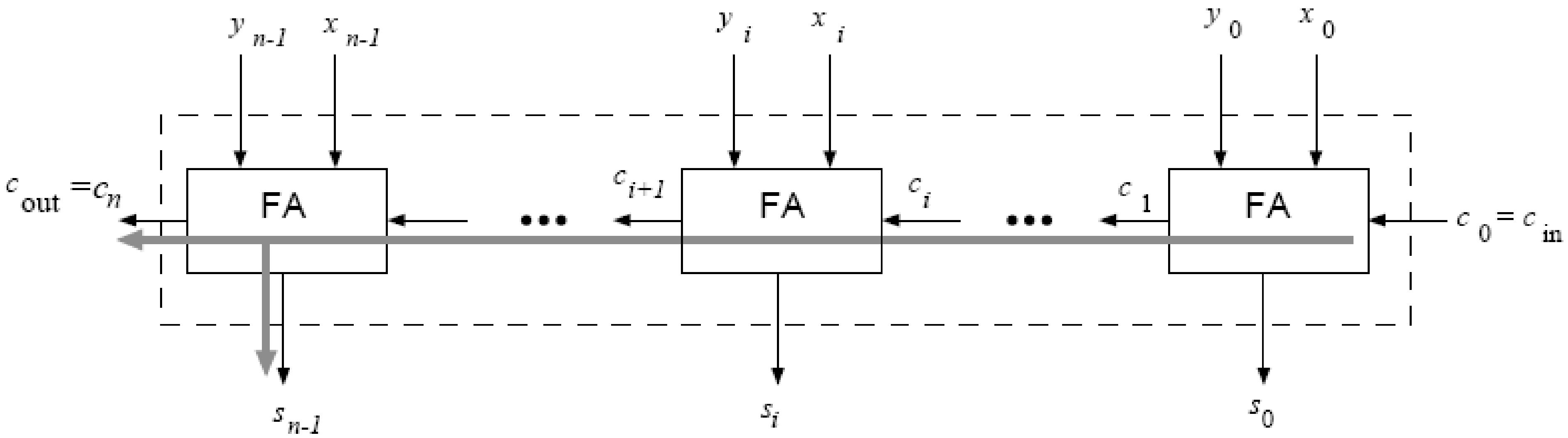


*Figure: n- bit carry ripple adder (CRA)*

The two (n- bit) operands (X and Y) are available at the same time.

*The carries propagate from the FA in position 0 (with inputs are $x_0$ and $y_0$) to position i before that position produces correct sum and carry out bits.

The worst case delay ($T_{CRA)}$ of n-bit CRA

$$T_{CRA} = (n-1)t_c + \max(t_{c,} t_s)$$

$$= 2(n-1)t_{NAND} + max(2t_{NAND}, t_{XOR}) + t_{XOR}$$

This is the delay of first XOR gate in the sum circuit

**Example:** Design an 8-bit CRA to add the following 2's complement numbers. Perform the addition on your design: $A = -93_{10}$ $B = 126_{10}$

$A = +93_{10} = (01011101)_2$ **Take 1's complement of A:** $A' = (10100010)$

$B = 126_{10} \equiv (01111110)_2$

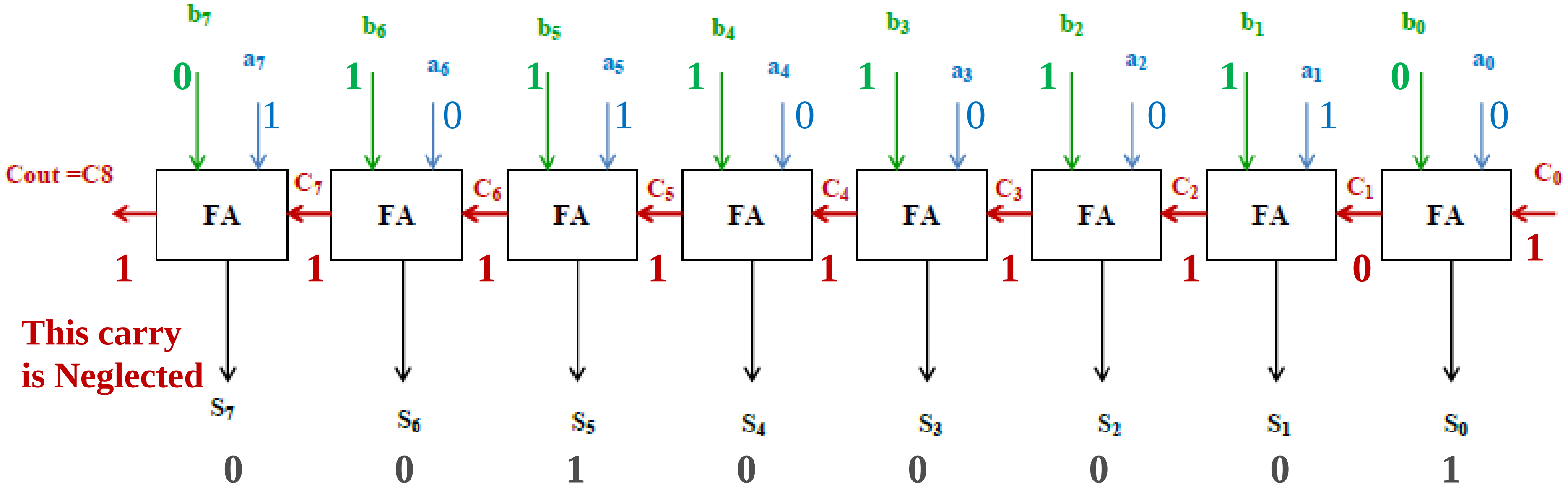


Figure: 8- bit CRA

## Reducing the Adder Delay

The delay of the CRA can be reduced by the following approaches:

- Reducing the carry delay $t_c$ . This is achieved in the switched CRA called Manchester adder.

- Changing the linear factor (n) to a smaller factor (such as *n/k or log*$_n$). This is achieved by the carry skip adder, carry lookahead adder, prefix adder, carry select adder, and conditional sum adder.

- Changing the number representation system.

## Carry Lookahead Adder (CLA)

The basic idea of CLA is to compute several carries, simultaneously.
In the extreme, all carries could be computed at the same time.

**Aside**: The carry- out from a FA is: $c_{i+1} = x_i y_i + x_i c_i + y_i c_i$

$$\boxed{c_{i+1} = x_i y_i + (x_i + y_i) c_i}$$

From this equation, we can see that there are three **mutually exclusive cases**. These cases depends only on the input operands bits ($x_i$ *and* $y_i$).

***Case 1: carry*** $c_{i+1}$ ***generate if*** $x_i = y_i = 1 \longrightarrow g_i = x_i . y_i$

***Case 2: carry*** $c_i$ ***is propagated if*** $x_i$ ***or*** $y_i \longrightarrow p_i = x_i \oplus y_i$

***Case 3: carry*** $c_i$ ***is killed if*** $x_i = y_i = 0 \longrightarrow k_i = x_i'.y_i'$

***Or is alive if*** $a_i = k_i'$

**** Rewrite the carry- out of a FA:***

$$\boxed{c_{i+1} = g_i + a_i c_i \quad \text{or} \quad c_{i+1} = g_i + p_i c_i}$$

**Return**

## (n- bit) CLA Addition

Using the previous equations of $c_{i+1}$ $\longrightarrow$ all carries can be determined in parallel from the input data X and Y and forced carry $C_0$ .

*Example: Design a 4- bit CLA* $\longrightarrow$ *the carry equations are:*

$c_1 = g_0 + c_0 a_0$

$c_2 = g_1 + c_1 a_1 = g_1 + g_0 a_1 + c_0 a_0 a_1$

$c_3 = g_2 + c_2 a_2 = g_2 + g_1 a_2 + g_0 a_1 a_2 + c_0 a_0 a_1 a_2$

$c_4 = g_3 + c_3 a_3 = g_3 + g_2 a_3 + g_1 a_2 a_3 + g_0 a_1 a_2 a_3 + c_0 a_0 a_1 a_2$

**The internal design of a 4- bit CLA is:**

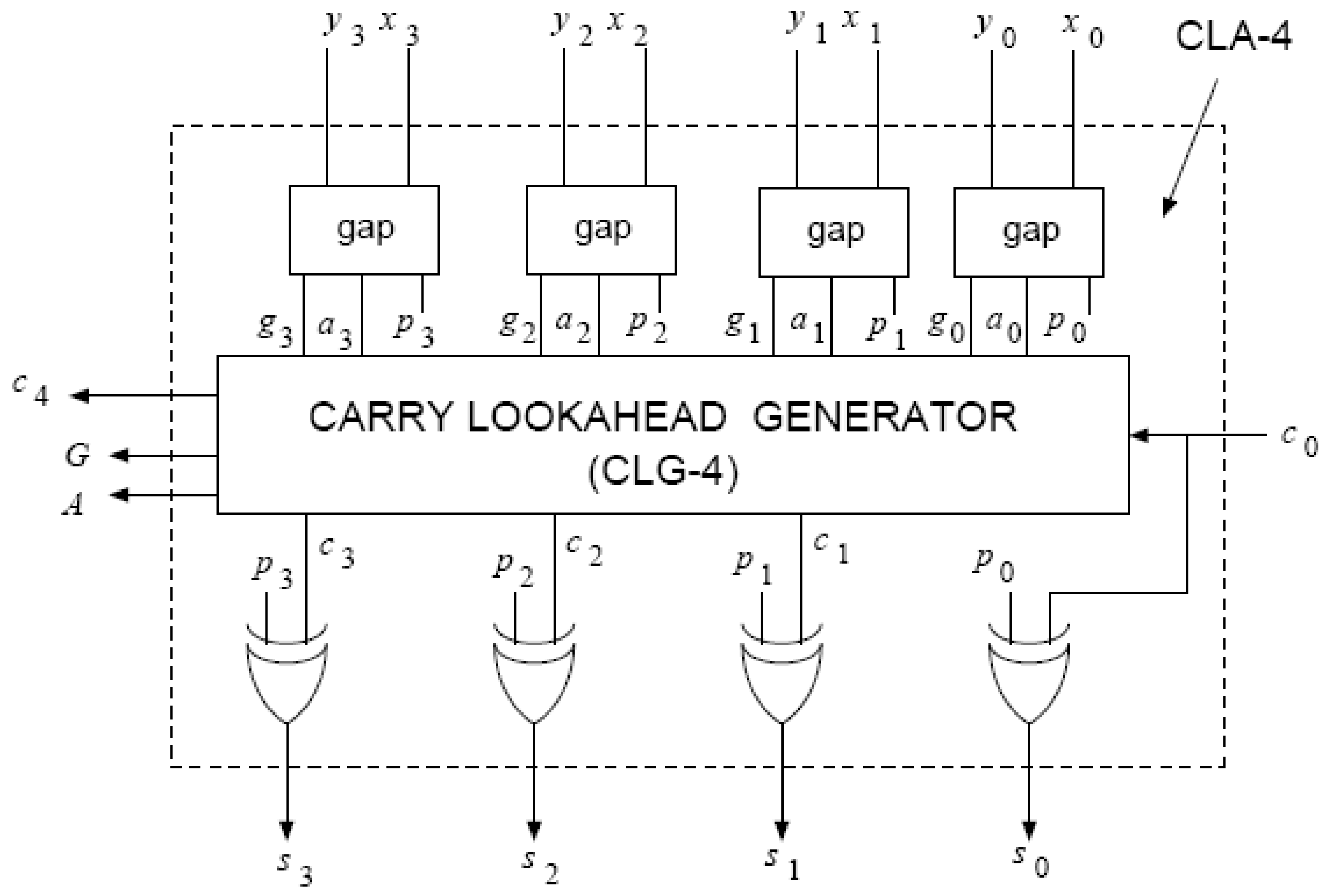
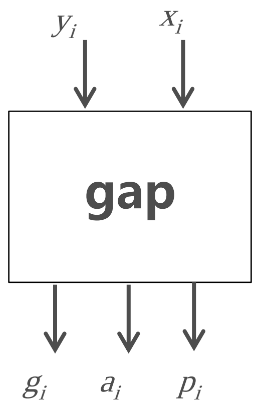


*Fig. (4)- bit CLA module (m=4)*

***Ex.*** ***Add X = $8_{10}$ = $(1000)_2$, and Y = $9_{10}$ = $(1001)_2$***



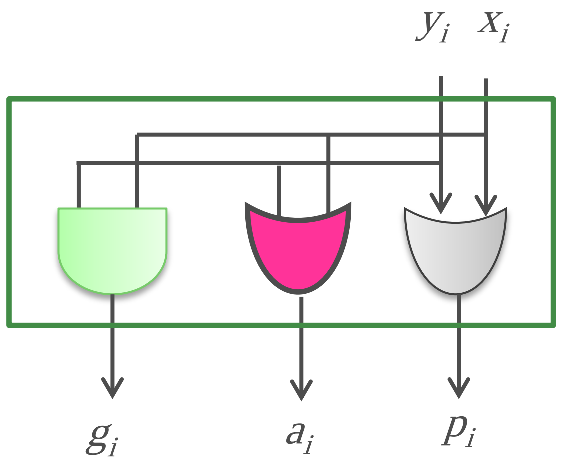


**Figure: Internal logic design of gap circuit.**

Where

A: is a carry alive signal of the group "m" (where m= 4-bit) : $A = \underset{i=0}{\overset{m-1}{AND}}\ a_i$

and G: is a carry generate signal of the group $m$ ($m$= 4-bit) :

$$G = \underset{j=0}{\overset{m-1}{OR}} \left( \underset{i=j+1}{\overset{m-1}{AND}}\ a_i \right) g_i$$