



Combinational Circuits & Sequential Circuits

By:-Gourav Kottawar

Contents

3.1 Half / Full Adder

3.2 Decoder / Encoder

3.3 Multiplexer / Demultiplexer,

3.4 Flip Flops - SR, D, JK, Master – Slave, Edge
Triggered

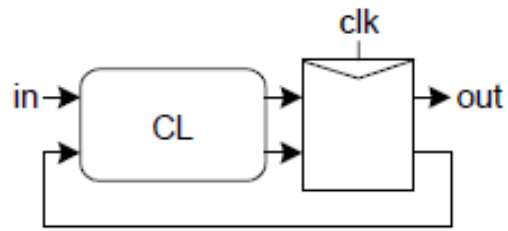
D flipflop with timing diagram

3.5 Shift Registers (Any one type)

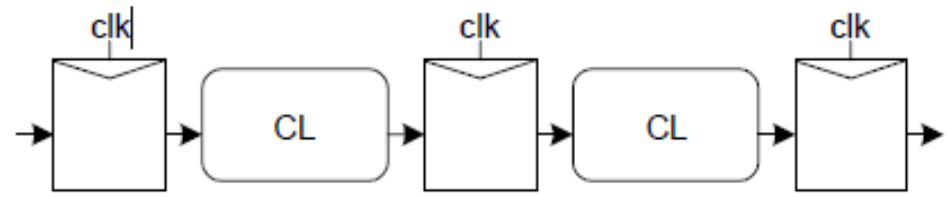
3.6 Introduction to Counters, Synchronous &
Asynchronous counter, Binary counter, mod-10
Counter

Introduction

- ***There are two types of logic circuits***
- ***Combinational Circuits***
- Formed by combination of different logic circuits
- Outputs depend on the current inputs
- Ex- Multiplexer, De multiplexer, Encoder, Decoder
- No memory allocation required
- ***Sequential Circuits***
- Outputs depend on current and previous inputs
- Requires separating previous, current and future
- Called ***states or tokens***
- Example: Finite State Machines (FSMs), Pipelines



Finite State Machine



Pipeline

Difference between Combinational and sequential circuits

Combinational	Sequential
Output of any instance of time depends only upon the input variables	Output is generated dependent upon the present input variables and also on the basis of past history of these inputs
Memory unit is not required. i.e. it doesn't allocate any memory to the elements.	Memory unit is required. . i.e. it allocates any memory to the elements.
Faster	Slower
Easy to design	Difficult
Parallel adder	Serial adder
Ex- Half and full adder Half and full subtractor MUX , DEMUX	Ex- Flip flops Shift registers Binary counters

Adders

- Computers implement arithmetic computations such as addition, subtraction, multiplication, division and many such operations using the concept of combinational circuits.

- Ex-

1	1	0
0	1	1
<hr/>		
1	0	0
1		1

- When addition of two k bits is computed the answer must be of k bits.
- But If the numbers are unsigned binary , the result can be K+1 bits.
- This k+1 bit is carry.
- Three types of Adder

1. Half Addder Adder
- 2.Full Addder
- 3.Ripple carry Adder

1. Half Adder

Half Adder: is a combinational circuit that performs the addition of two bits, this circuit needs two binary inputs and two binary outputs.

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth table

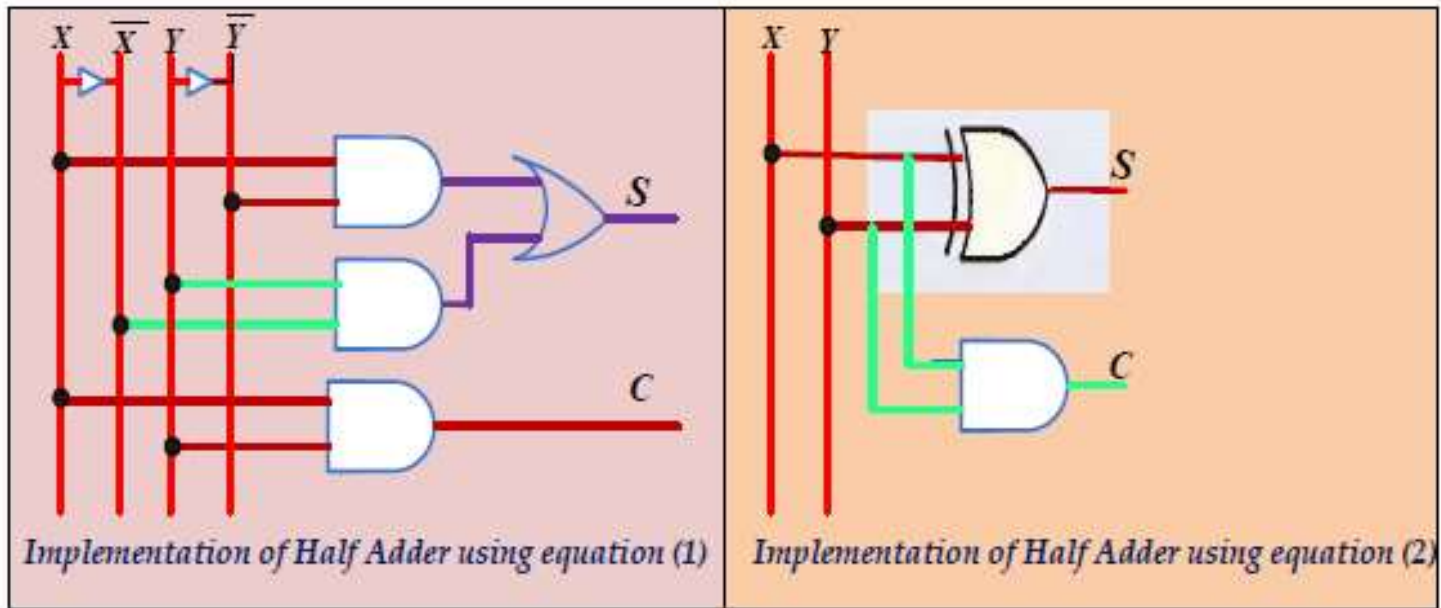
The simplified Boolean function from the truth table:

$$\left. \begin{array}{l} S = \bar{X}Y + X\bar{Y} \\ C = XY \end{array} \right\} \text{(Using sum of product form)}$$

Where **S** is the sum and **C** is the carry.

$$\left. \begin{array}{l} S = X \oplus Y \\ C = XY \end{array} \right\} \text{(Using XOR and AND Gates)}$$

1. Half Adder



- The implementation of half adder using **exclusive-OR and an AND gates** is used to show that two half adders can be used to construct a full adder.
- The inputs to the **XOR gate are also the inputs to the AND gate.**

Half Adder

- Draw backs
 1. Doesn't handle carries
 2. Doesnot do further processing on carries.

2. Full Adder

- **Full Adder** is a combinational circuit that performs the addition of three bits (two significant bits and previous carry).
- It consists of *three inputs and two outputs*, two inputs are the bits to be added, the third input represents the carry from the previous position.
- The full adder is usually a component in a cascade of adders, which add 8, 16 etc binary numbers.

Inputs			Outputs	
X	Y	C_{in}	S	C_{out}
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

Truth table for the full adder

➤ The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1 .

➤ The C_{out} output has a carry 1 if two or three inputs are equal to 1 .

➤ The Karnaugh maps and the simplified expression are shown in the following figures:

$X \backslash YC_{in}$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

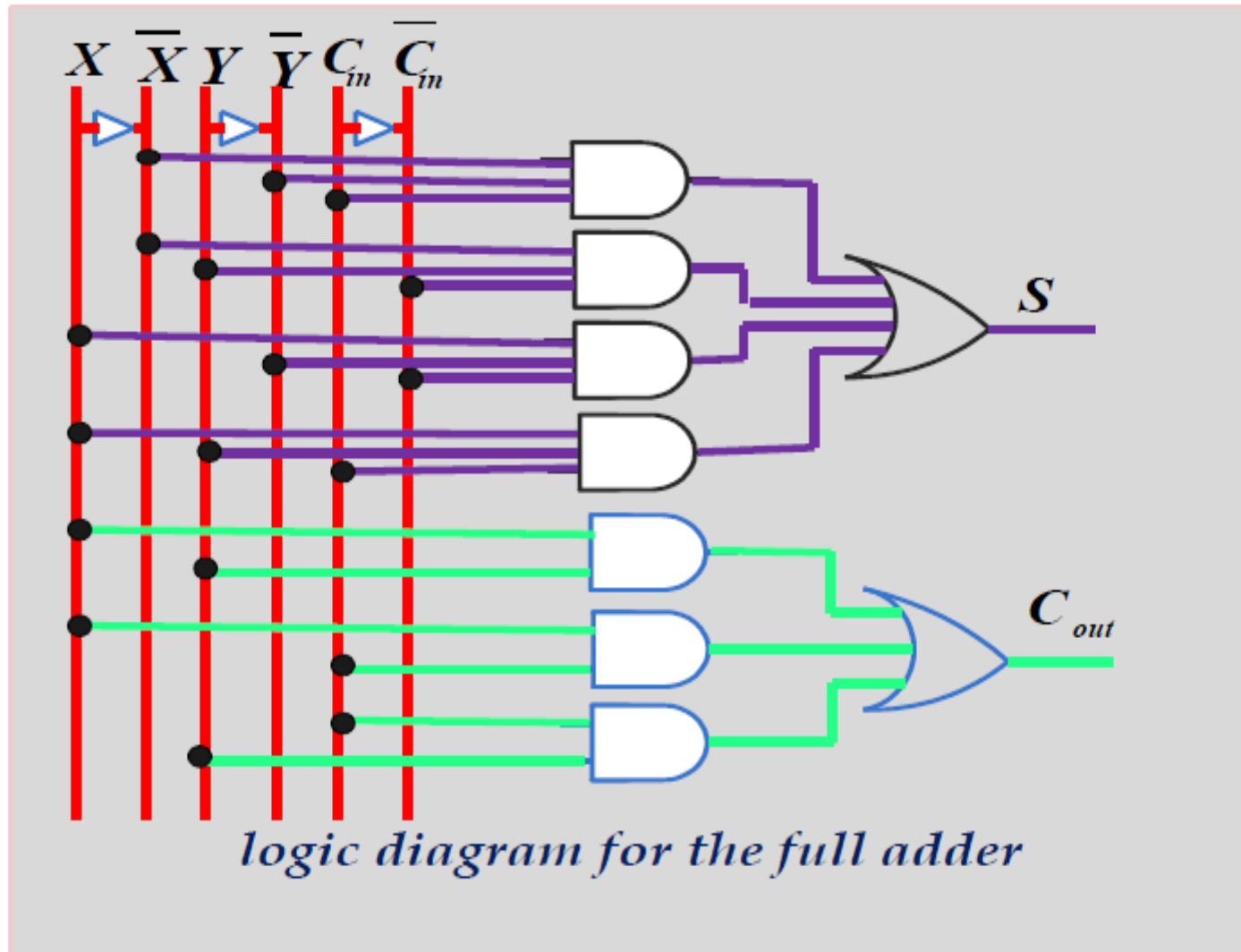
$$S = \overline{X}\overline{Y}C_{in} + \overline{X}Y\overline{C_{in}} + X\overline{Y}\overline{C_{in}} + XYC_{in}$$

$X \backslash YC_{in}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_{out} = XY + XC_{in} + YC_{in}$$

$$\left. \begin{array}{l} S = \overline{X}\overline{Y}C_{in} + \overline{X}Y\overline{C_{in}} + X\overline{Y}\overline{C_{in}} + XYC_{in} \\ C_{out} = XY + XC_{in} + YC_{in} \end{array} \right\} \text{(Sum of products)}$$

- The *logic diagrams* for the full adder implemented in *sum-of-products* form are the following:



- It can also be implemented using *two half adders* and *one OR gate* (using **XOR** gates).

$$\left\{ \begin{array}{l} S = C_{in} \oplus (X \oplus Y) \\ C_{out} = C_{in} \cdot (X \oplus Y) + XY \end{array} \right\}$$

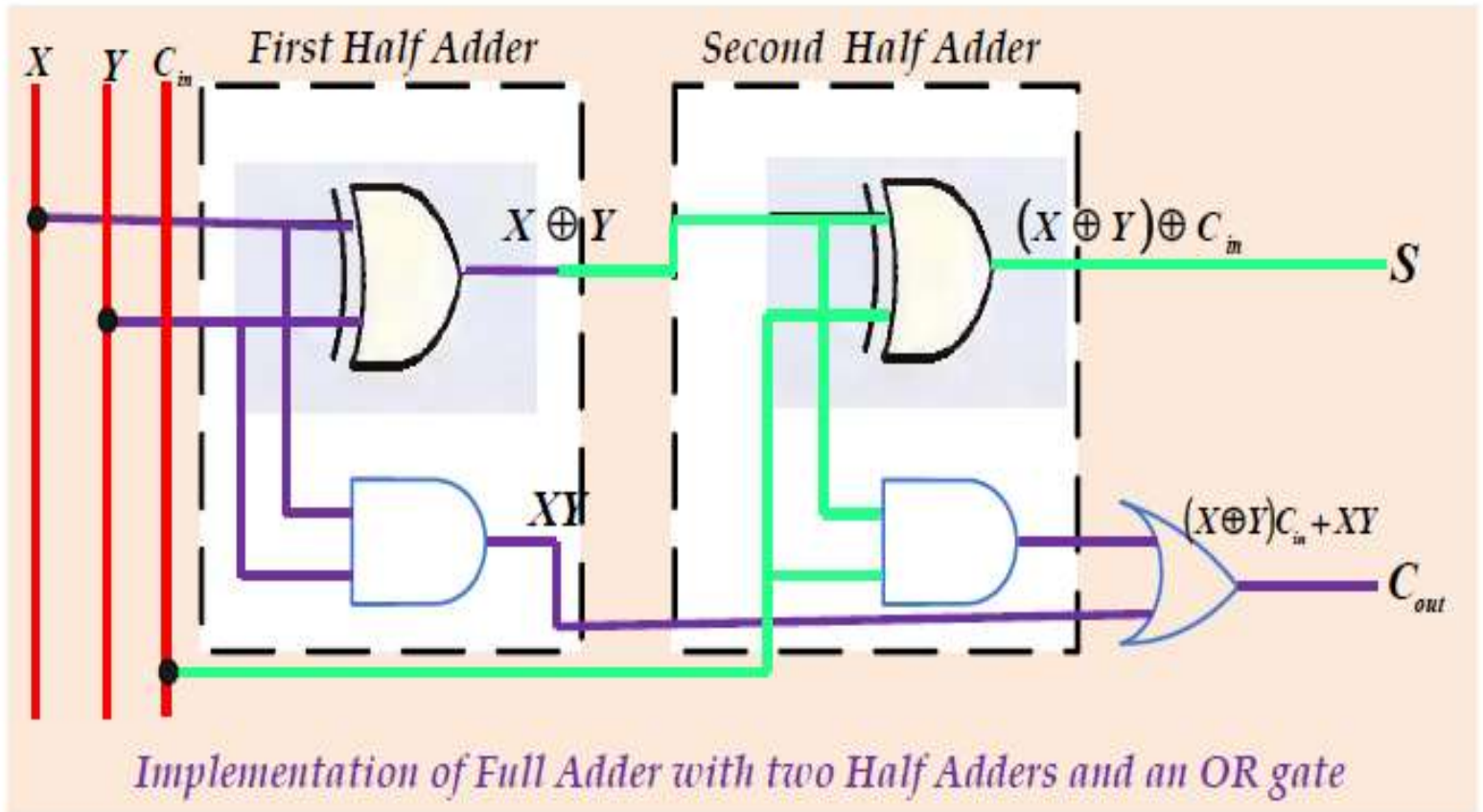
Proof:

The sum:

$$\begin{aligned} S &= \bar{X}\bar{Y}C_{in} + \bar{X}Y\bar{C}_{in} + X\bar{Y}\bar{C}_{in} + XYC_{in} \\ &= \bar{C}_{in}(\bar{X}Y + X\bar{Y}) + C_{in}(\bar{X}\bar{Y} + XY) \\ &= \bar{C}_{in}(\bar{X}Y + X\bar{Y}) + C_{in}(\overline{\bar{X}Y + X\bar{Y}}) \\ S &= C_{in} \oplus (X \oplus Y) \end{aligned}$$

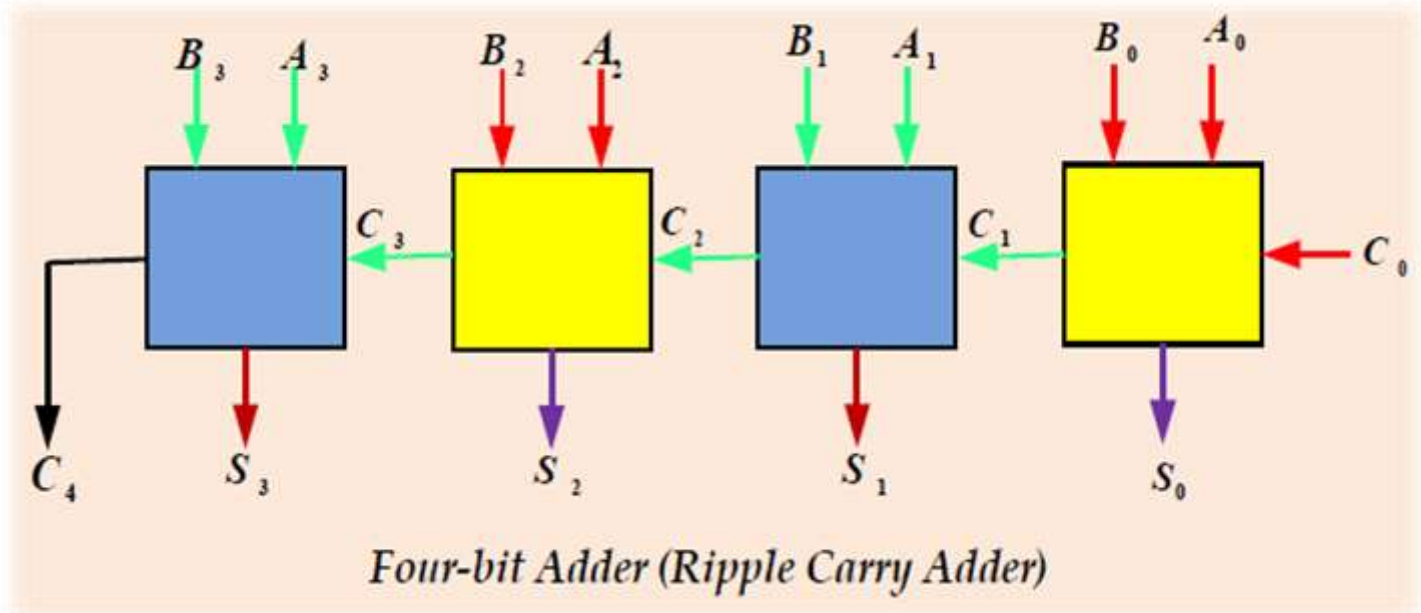
The carry output:

$$\begin{aligned} C_{out} &= \bar{X}Y C_{in} + X\bar{Y} C_{in} + XY C_{in} + XY \bar{C}_{in} \\ &= C_{in}(\bar{X}Y + X\bar{Y}) + XY(C_{in} + \bar{C}_{in}) \end{aligned}$$



3. Binary Adder (Asynchronous Ripple-Carry Adder)

- A binary adder is a digital circuit that produces the *arithmetic sum of two binary numbers*.
- A binary adder can be constructed with *full adders connected in cascade with the output carry from each full adder connected to the input carry of the next full adder in the chain*.
- The *four-bit adder is a typical example of a standard component. It can be used in many application involving arithmetic operations*.



- The input carry to the adder is C_0 and it ripples through the full adders to the output carry C_4 .
- n -bit binary adder requires n full adders.

Example

$A + B$

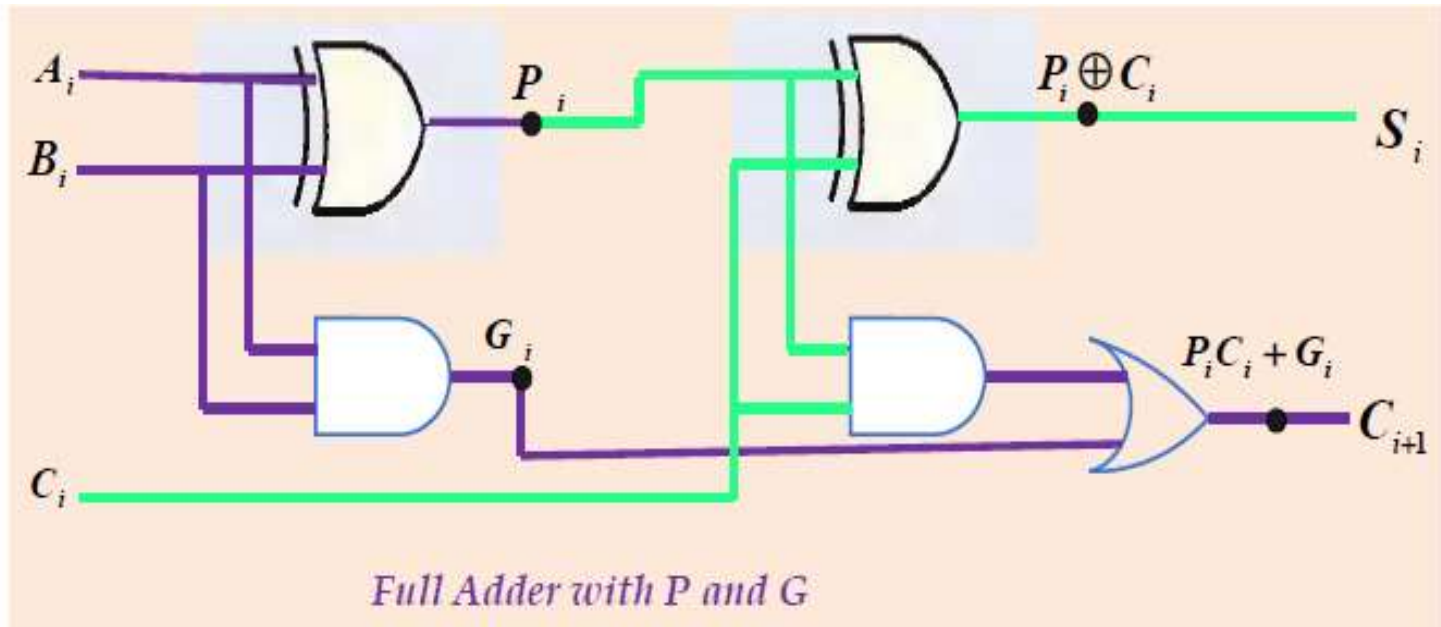
($A = 1011$) and ($B = 0011$)

<i>Subscript i</i>	3	2	1	0	
<i>Input Carry</i>	0	1	1	0	C_i
<i>A</i>	1	0	1	1	A_i
+					
<i>B</i>	0	0	1	1	B_i
<i>Sum</i>	1	1	1	0	S_i
<i>Output Carry</i>	0	0	1	1	C_{i+1}

$C_0 = 0$

Carry Propagation

- The addition of $A + B$ binary numbers in *parallel* implies that all the bits of A and B are available for computation at the same time.
- As in any combinational circuit, the signal must **propagate** through the gates before the correct output sum is available.
- The output will not be correct unless the signals are given enough time to propagate through the gates connected from the input to the output.
- The longest **propagation delay time** in an adder is the time it takes the carry to propagate through the full adders.



- The signal from the carry input C_i to the output carry C_{i+1} propagates through an **AND** gate and an **OR** gate, which equals **2 gate levels**.
 - If there are **4** full adders in the binary adder, the output carry C_4 would have **$2 \times 4 = 8$ gate levels**, from C_0 to C_4
 - For an **n -bit** adder, **$2n$** gate levels for the carry to propagate from input to output are required.

- The **carry propagation time** is an important attribute of the adder because it limits the speed with which two numbers are added.
- To reduce the carry propagation delay time:
 - 1) Employ faster gates with reduced delays.
 - 2) Employ the principle of **Carry Lookahead Logic**.

Proof: (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 $i + 1$.
- ✓ The **Boolean function** for the carry outputs of each stage and substitute the value of each C_i from the previous equations.

$$\left. \begin{aligned} C_0 &= \text{input carry} \\ C_1 &= G_0 + P_0 C_0 \\ C_2 &= G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) \\ &= G_1 + P_1 G_0 + P_1 P_0 C_0 \\ C_3 &= G_2 + P_2 C_2 = G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 C_0) \\ &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \end{aligned} \right\}$$

- 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 .

