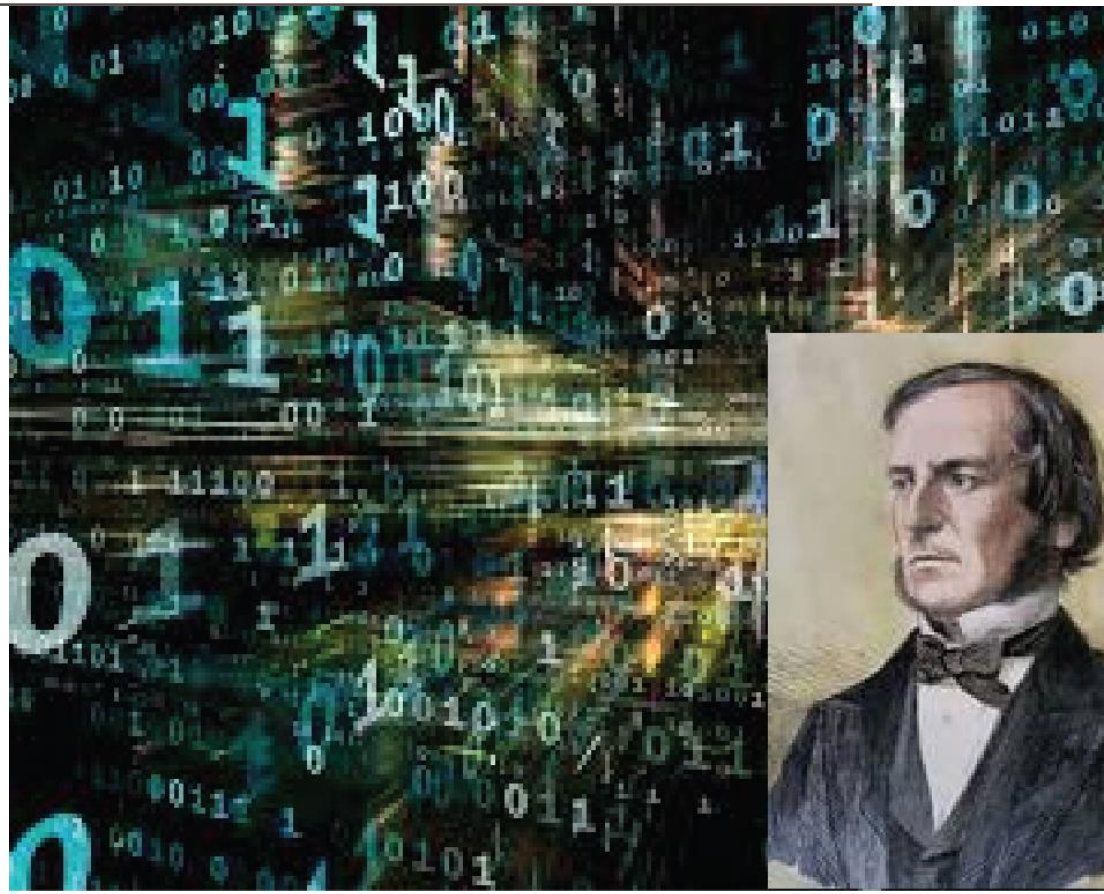
DIGITAL CIRCUITS

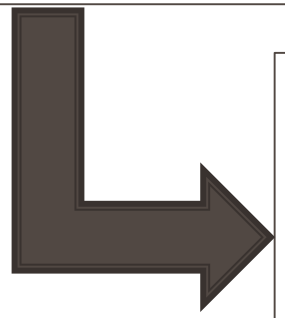


Week-9, Lecture-1
Combinational Circuits

Sneh Saurabh
3rd October, 2018

Digital Circuits: Announcements/Revision

Digital Circuits



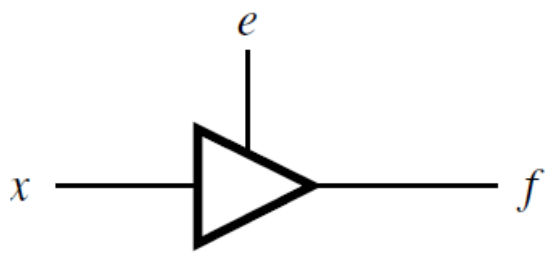
Three-state gates

Three-states in a digital circuit

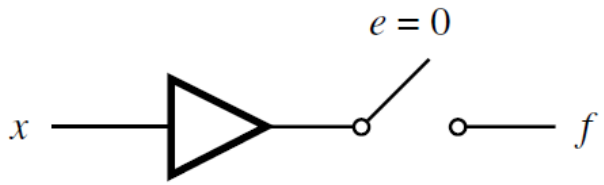
- **Two states**: conventional logic “0” and “1”

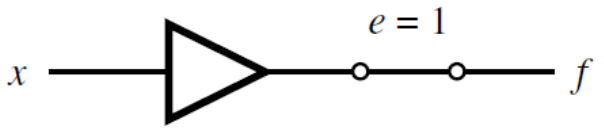
- **Third state**: *high-impedance* state
  - the logic behaves like an **open circuit**, which means that the output appears to be disconnected
  - the circuit connected to the output of the three-state gate is not affected by the inputs to the gate
  - represented as “Z”

Three-state (Tri-state) buffer



e	x	f
0	0	Z
0	1	Z
1	0	0
1	1	1





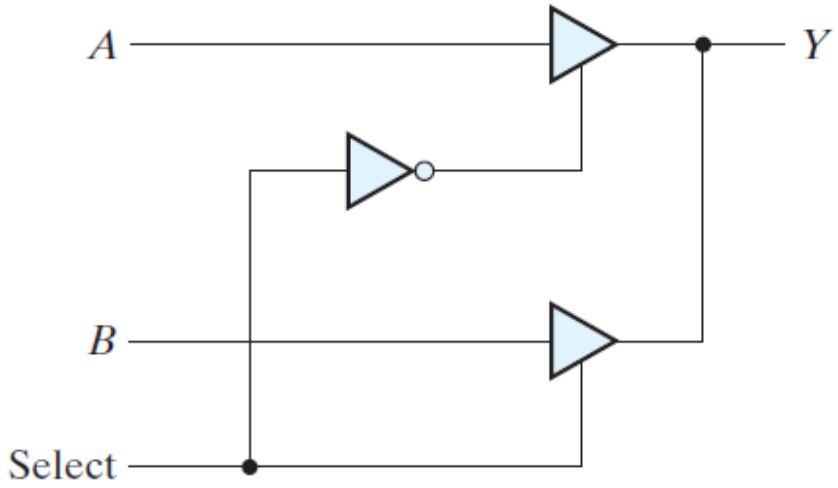
- Realizing logic gates is possible by connecting together the ***outputs*** of tristate gates (this is known as wired connection)

- This is not possible with ordinary logic gates, because their outputs are always ***active***; hence a ***short circuit*** would occur.

Priority Encoder: Implementation

Problem:

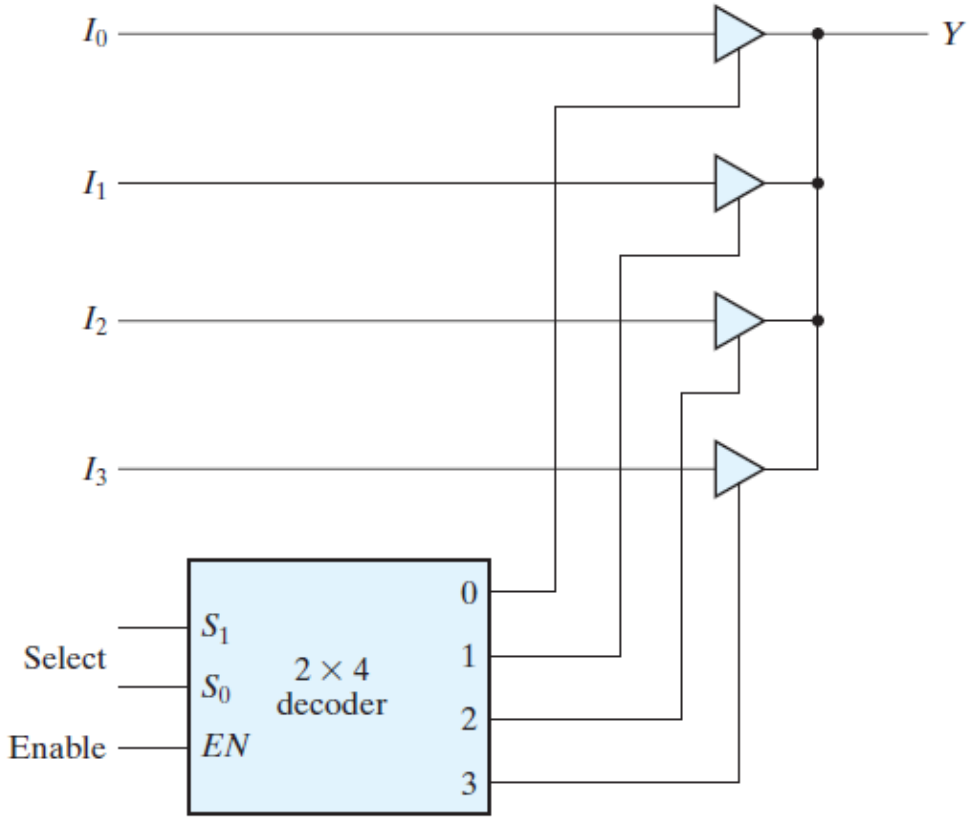
Implement a 2-to-1 MUX using two tri-state buffers and an inverter.



Priority Encoder: Implementation

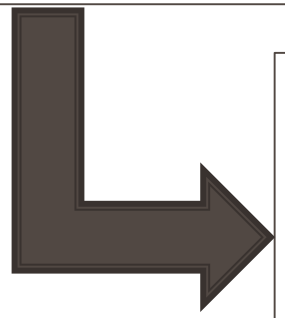
Problem:

Implement a 4-to-1 MUX using four tri-state buffers and one 2-to-4 decoder.



(b) 4-to-1-line mux

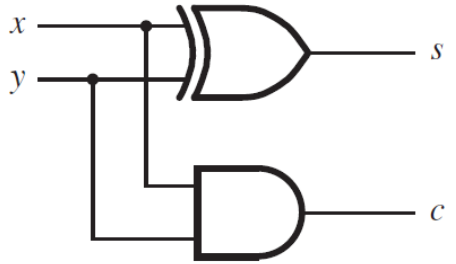
Digital Circuits



Arithmetic Operations

Adders: Half Adders and Full Adders (Recap…)

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



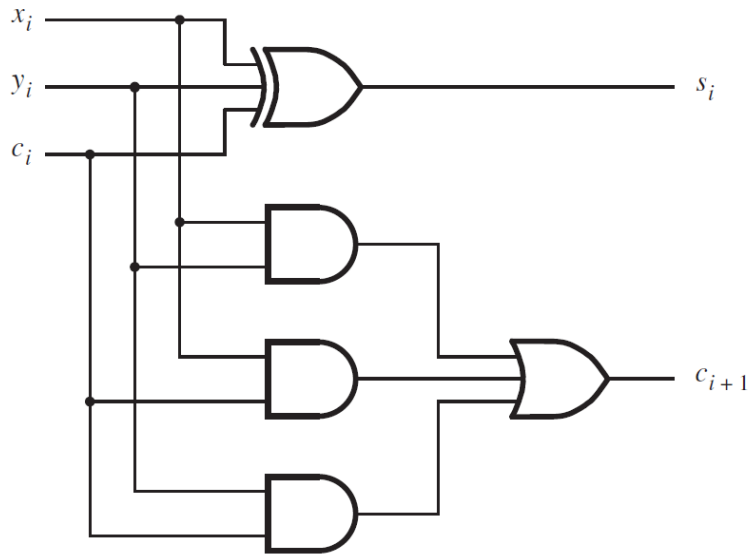
A circuit which implements the addition of only two bits is known as ***Half Adder (HA)***

c_i	x_i	y_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

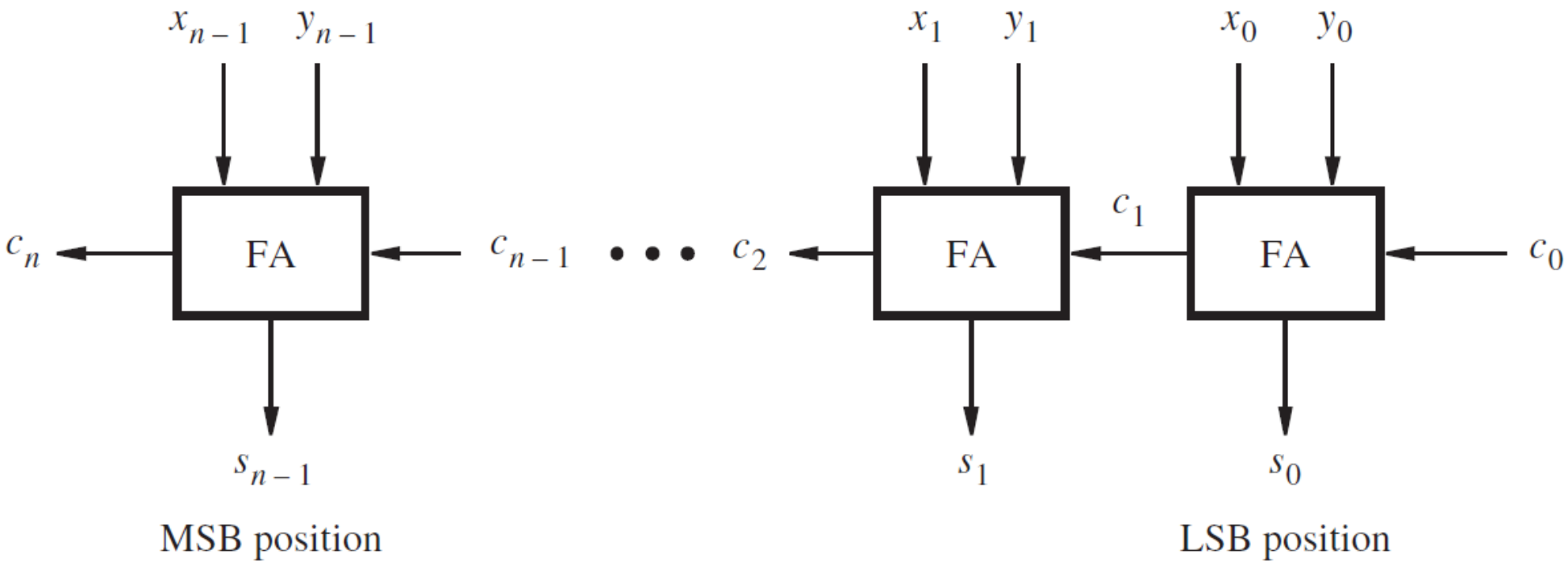
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$s_i = x_i \oplus y_i \oplus c_i$$

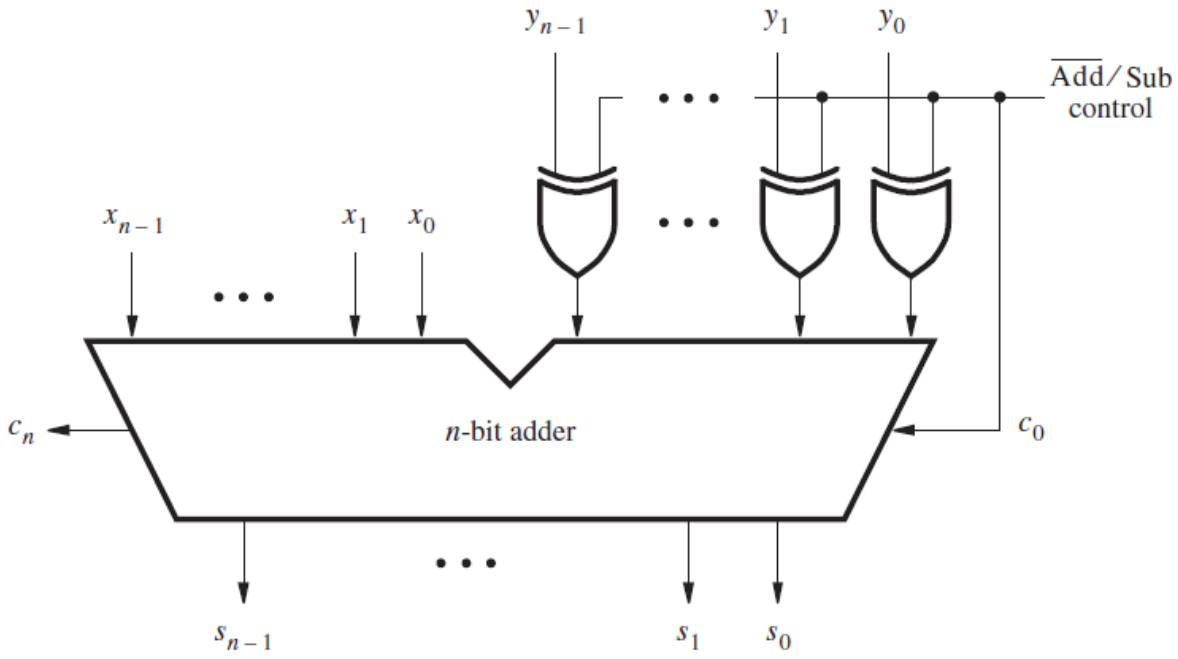
Full Adder Circuit



Adders: Ripple Carry Adder (Recap…)



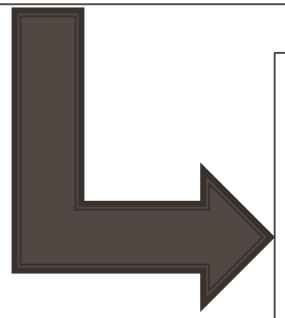
Adders: Adder Subtractor Unit (Recap…)



We will look at two problems of this implementation

1. Arithmetic Overflow
2. Speed

Adder/Subtractor



Arithmetic Overflow

Arithmetic Overflow: Definition

- The result of addition/subtraction is supposed to fit within the significant bits used to represent the numbers
  - For example, if the numbers are represented in 4-bits, then sum/difference of two numbers should also fit in 4-bits

- If N bits are used to represent signed numbers, then the result must be in the range -2^{N-1} and $2^{N-1}-1$
  - For example, if $N = 4$, then the result of addition/subtraction should be within -8 to $+7$.
- If the result does not fit in this range, then we say that ***arithmetic overflow*** has occurred.

- To ensure the correct operation of an arithmetic circuit, it is important to be able to detect the occurrence of an arithmetic overflow

Arithmetic Overflow: Detection

- When sign of numbers are of opposite types, then no overflow can occur
- Take examples of **two *4-bit numbers***, and check under what condition overflow can occur
  - ➢ Consider the carry-out from **MSB position (c_3)** and carry out from **sign-bit position (c_4)**

$$\begin{array}{r} (+7) \\ +\ (+2) \\ \hline (+9) \end{array} \qquad \begin{array}{r} 0\ 1\ 1\ 1 \\ +\ 0\ 0\ 1\ 0 \\ \hline 1\ 0\ 0\ 1 \end{array}$$

$c_4 = 0$
$c_3 = 1$

$$\begin{array}{r} (-7) \\ +\ (+2) \\ \hline (-5) \end{array} \qquad \begin{array}{r} 1\ 0\ 0\ 1 \\ +\ 0\ 0\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

$c_4 = 0$
$c_3 = 0$

- Examples suggest that overflow occurs when these carry-outs have different values, and a correct sum is produced when they have the same value
  - ➢ $Overflow = c_3 \oplus c_4$

$$\begin{array}{r} (+7) \\ +\ (-2) \\ \hline (+5) \end{array} \qquad \begin{array}{r} 0\ 1\ 1\ 1 \\ +\ 1\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 0\ 1 \end{array}$$

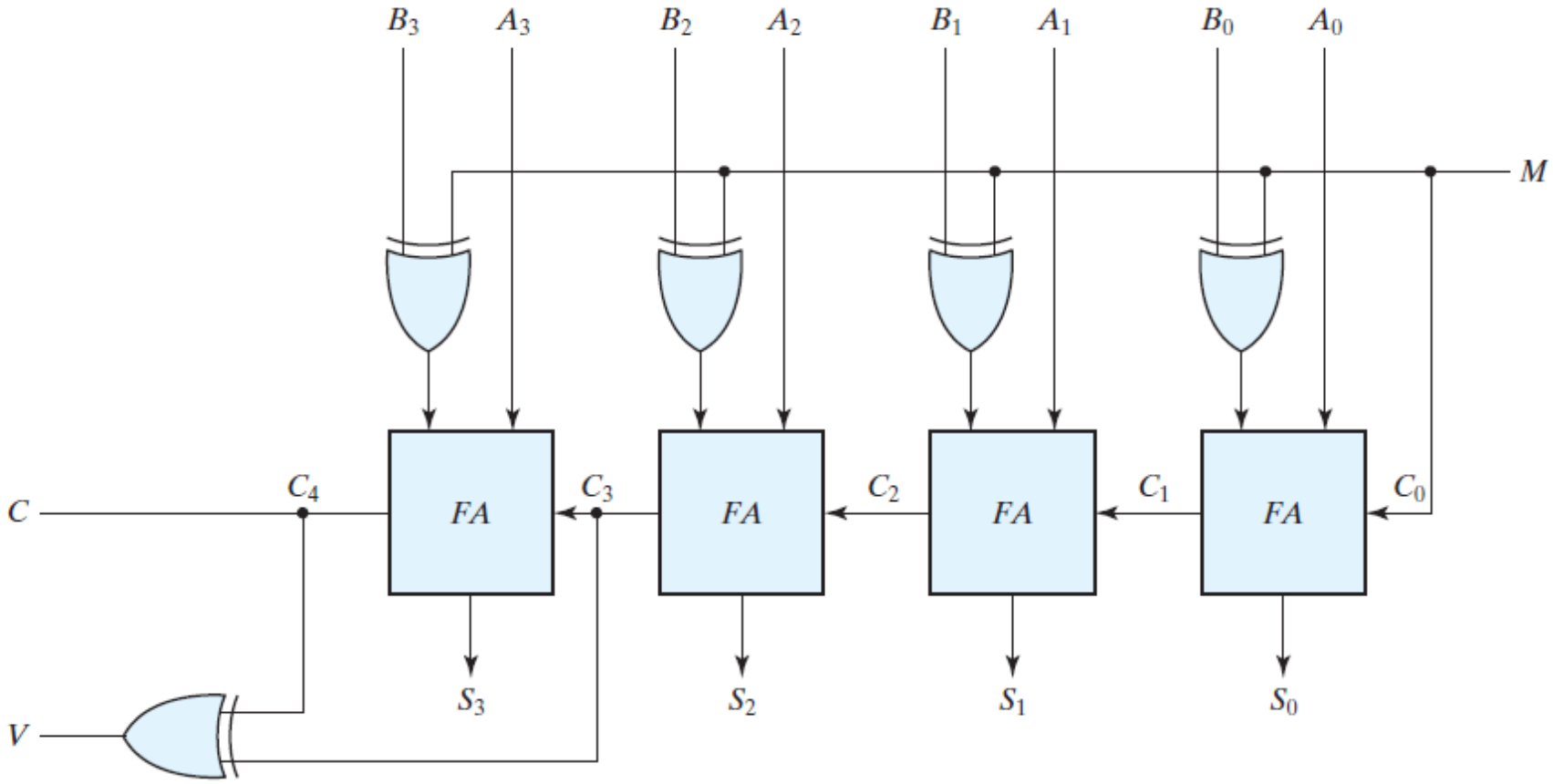
$c_4 = 1$
$c_3 = 1$

$$\begin{array}{r} (-7) \\ +\ (-2) \\ \hline (-9) \end{array} \qquad \begin{array}{r} 1\ 0\ 0\ 1 \\ +\ 1\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 1 \end{array}$$

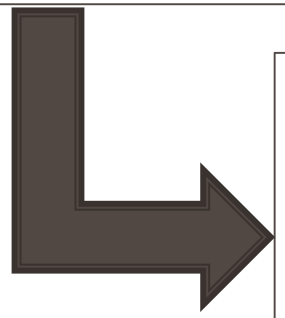
$c_4 = 1$
$c_3 = 0$

- Indeed, this is true in general for both addition and subtraction of 2's-complement numbers.
  - ➢ $Overflow = c_{N-1} \oplus c_N$

Arithmetic Overflow: Circuit

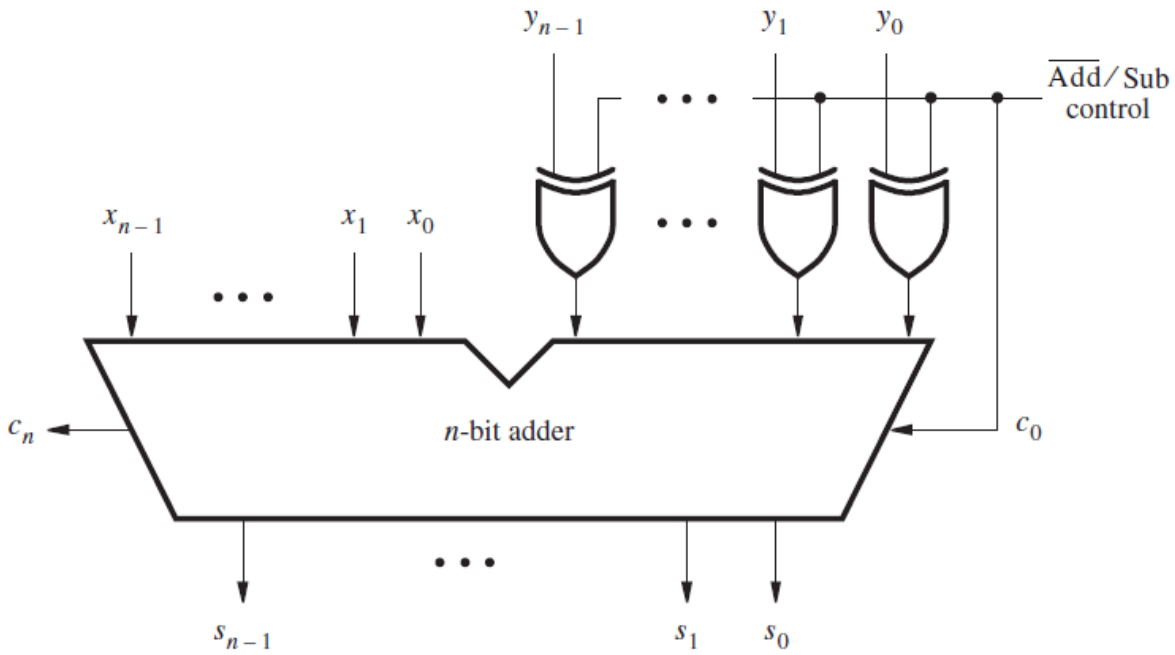


Adder/Subtractor



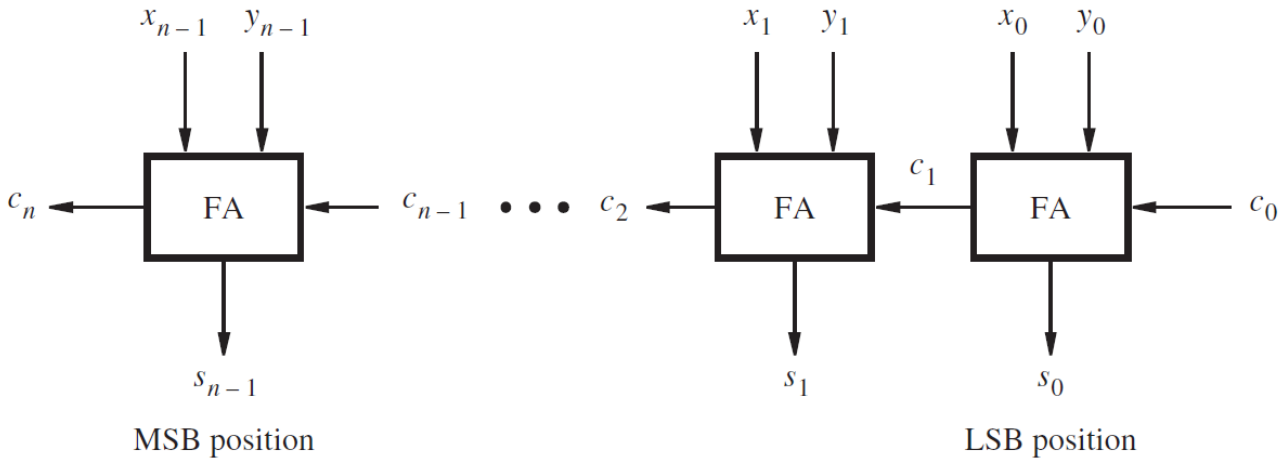
Improving Speed

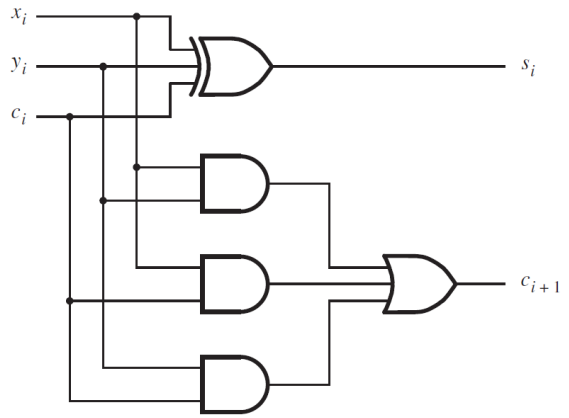
Adder Subtractor Unit: Performance Issues



- Speed is decided by the largest delay from the time the operands X and Y are presented as inputs, until the time all bits of the sum S and the final carry-out c_n are valid
- Most of delay is caused by the n-bit ripple-carry adder circuit

Adder Subtractor Unit: Performance Issues





- For a full-adder the delay for the carry-out signal is equal to two gate delays
- The longest delay is along the carry computation path
  - Computation of c_n requires that c_{n-1} is computed, c_{n-1} waits for c_{n-2} and so on

- For n-stage ripple carry adder, there is $(2n + 1)$ gate delays
- When $n = 32$ or $n = 64$, the delay becomes unacceptably high
- **Fast adder circuit is required: computation of carry must be speeded**