Adder Implementation

Addition of binary digits

$x_i + y_i + c_i$			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

$s_i = x_i' \cdot y_i' \cdot c_i + x_i' \cdot y_i \cdot c_i' + x_i \cdot y_i' \cdot c_i' + x_i \cdot y_i \cdot c_i =$

=… (Algebraic Manipulation)…

$= x_i \oplus y_i \oplus c_i$

(Intuitively, when one input bit is 1, or when all three are 1, then $s_i=1$)

$C_{i+1} = x_i' \cdot y_i \cdot c_i + x_i \cdot y_i' \cdot c_i + x_i \cdot y_i \cdot c_i' + x_i \cdot y_i \cdot c_i =$

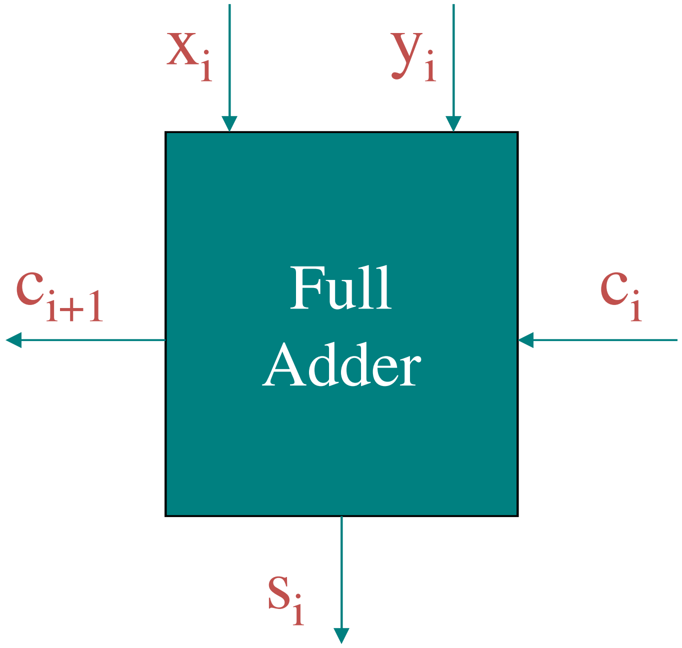
=… (Algebraic Manipulation)…

$= x_i \cdot y_i + c_i \cdot (x_i \oplus y_i)$

(Intuitively, when two input bits are 1, or when all three are 1, then $c_{i+1}=1$)

Full Adder

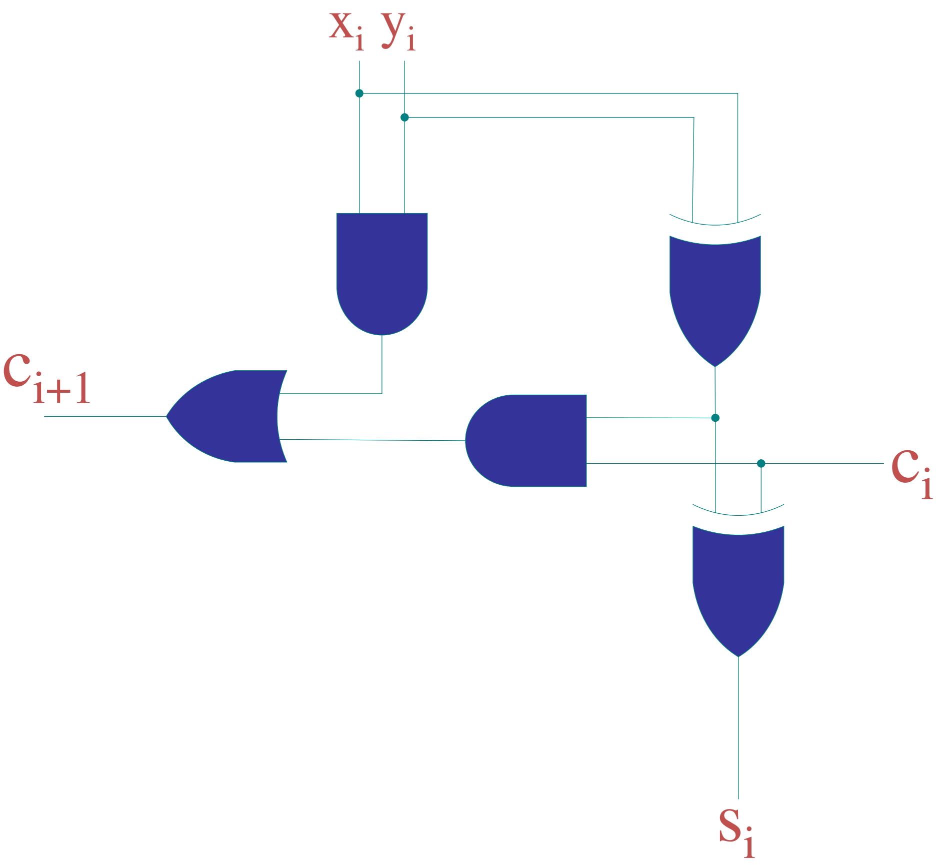
Symbol:



$s_i = x_i \oplus y_i \oplus c_i$

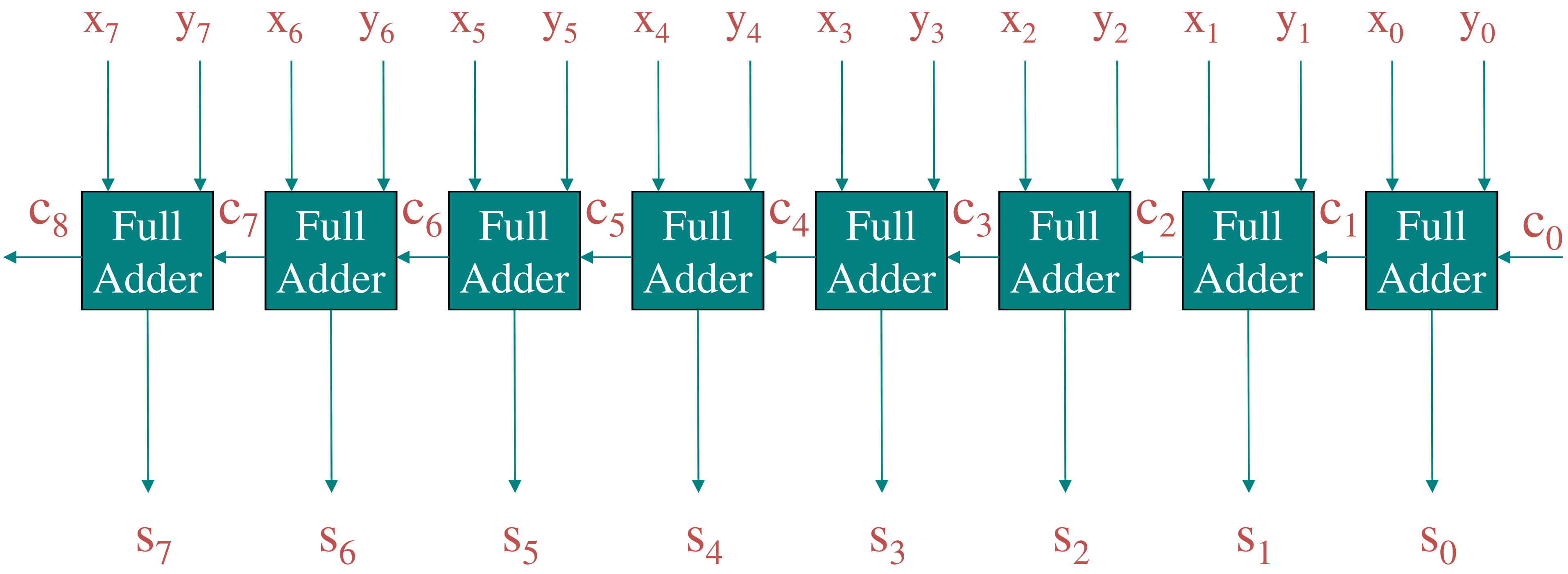
$c_{i+1} = x_i \cdot y_i + c_i \cdot (x_i \oplus y_i)$

Gate-Level Implementation:

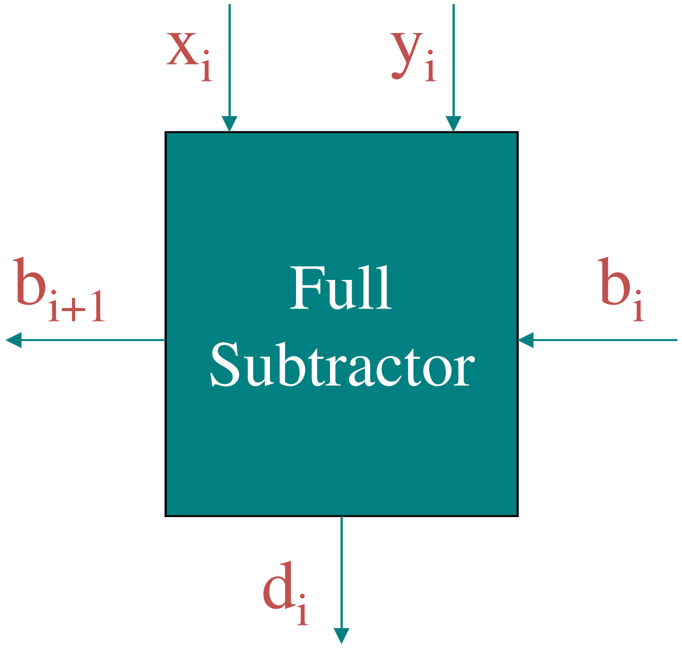


8-Bit Ripple Carry Adder

Hierarchical Implementation:



Full Subtractor



Subtraction of binary digits

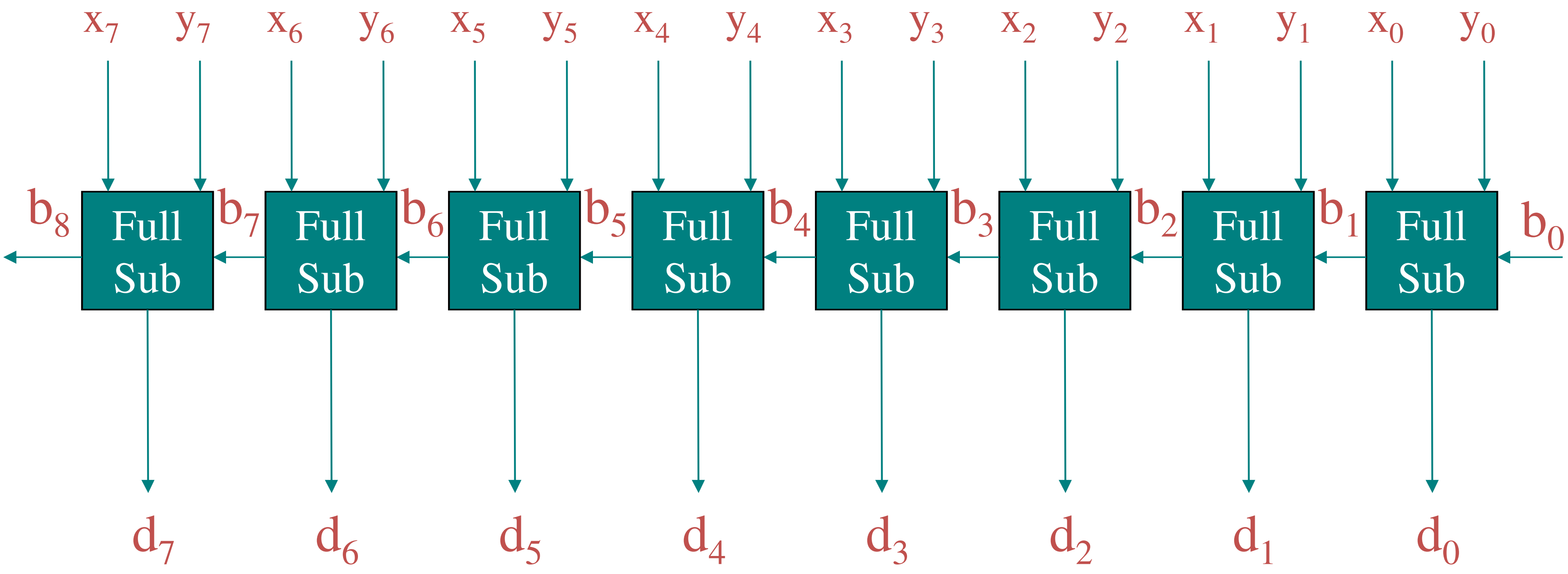
x_i - y_i - b_i			b_{i+1}	d_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$$d_i = (x_i \odot y_i) \odot b_i$$

$$b_{i+1} = (x_i \odot y_i) \cdot b_i + x_i' \cdot y_i$$

8-Bit Ripple Borrow Subtractor

Hierarchical Implementation:



2’s Complement Representation

Obtaining 2s Complement of a Binary Number:

Invert All Bits, Add 1 to the Result, Ignore MSB Carry

Example:

A=01001010

Invert all Bits: A’=10110101

Add 1: 10110101 + 00000001 = 10110110

The 2’s Complement of 0 is itself:

B=00000000

Invert all Bits: B’=11111111

Add 1: 11111111 + 00000001 = 1 00000000

Ignore the MSB Carry: 00000000

Note: The 2’s complement of the 2’s complement of a number is the number itself

Subtraction by 2’s Complement Addition

Alternative Subtraction Method:

(A-B: A is the “Minuend”, B is the “Subtrahend”)

The difference A-B can be obtained by:

i) taking the 2’s complement of the subtrahend, and

ii) adding it to the minuend, ignoring any MSB carry

Example:

A = 11001010 = $(202)_{10}$, B = 00110110 = $(54)_{10}$

i) 2’s Complement of B is :

B’+1 = 11001001 + 00000001 = 11001010

ii) Adding it to A gives:

11001010 + 11001010 = 1 10010100 = $(148)_{10}$

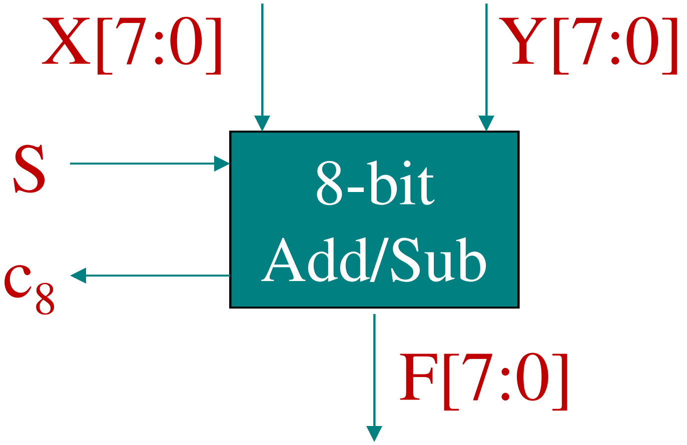
(Ignoring the MSB carry of the addition)

Note: We restrict ourselves to positive integers with A>B

8-bit Adder/Subtractor

Operation:

S	Function	Comment
0	F=X+Y	Addition
1	F=X+Y'+1	Subtraction

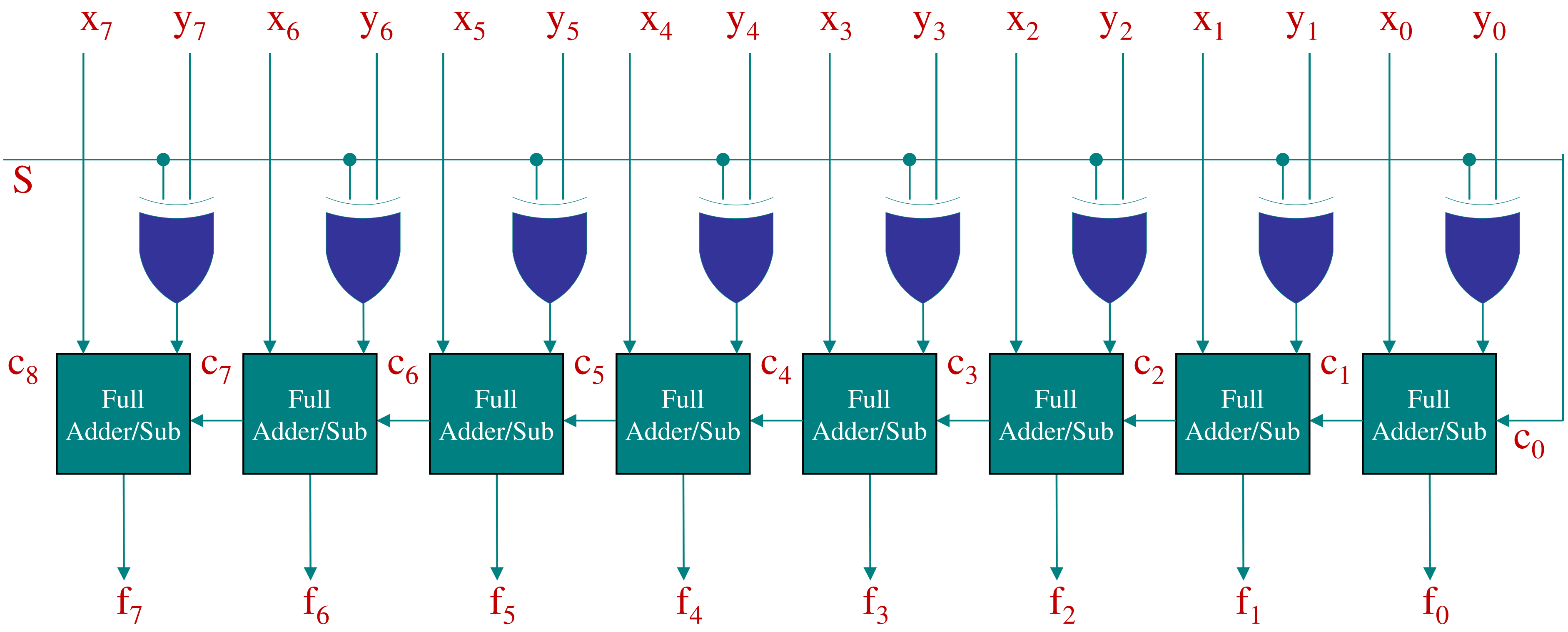


Explanation:

When S=0, the circuit behaves as an 8-bit adder, while when S=1, it behaves as an 8-bit subtractor. Subtraction is performed by adding the 2's complement of Y. The 2's complement of Y is obtained by inverting each bit of Y, which gives us Y', and then adding 1 to it

8-bit Adder/Subtractor

Hierarchical Implementation:



Note: The XOR gates behave as inverters when S=1, and the 1 for obtaining the 2's complement is added by connecting S to the carry c_0

2 to 1 Multiplexer (Selector)

Operation:

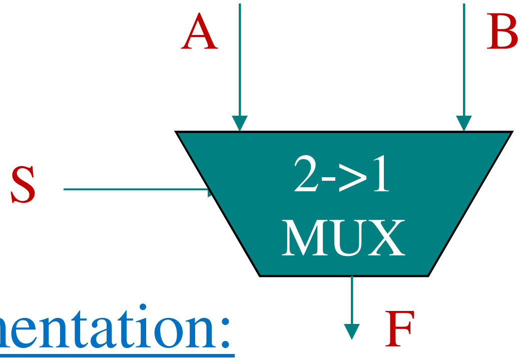
S	Function	Comment
0	F=A	Select A
1	F=B	Select B

S	A	B	S’	S’·A	S·B	F
0	0	0	1	0	0	0
0	0	1	1	0	0	0
0	1	0	1	1	0	1
0	1	1	1	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	1	1
1	1	0	0	0	0	0
1	1	1	0	0	1	1

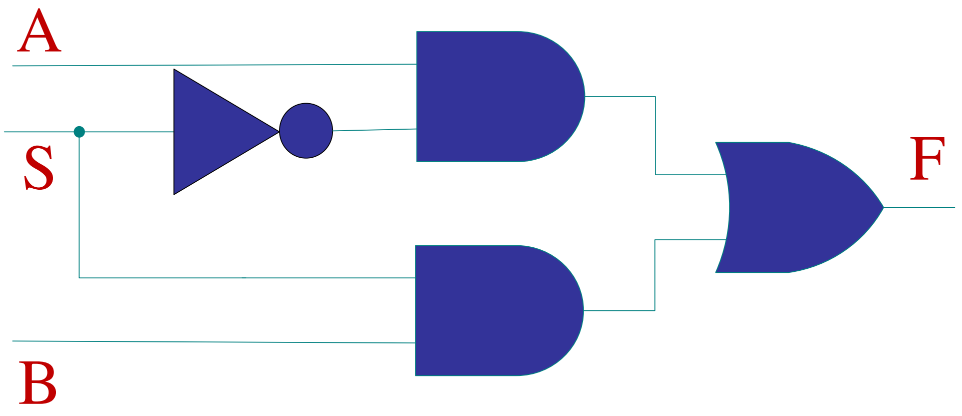
Logic Expression:

$F=(S' \cdot A)+(S \cdot B)$

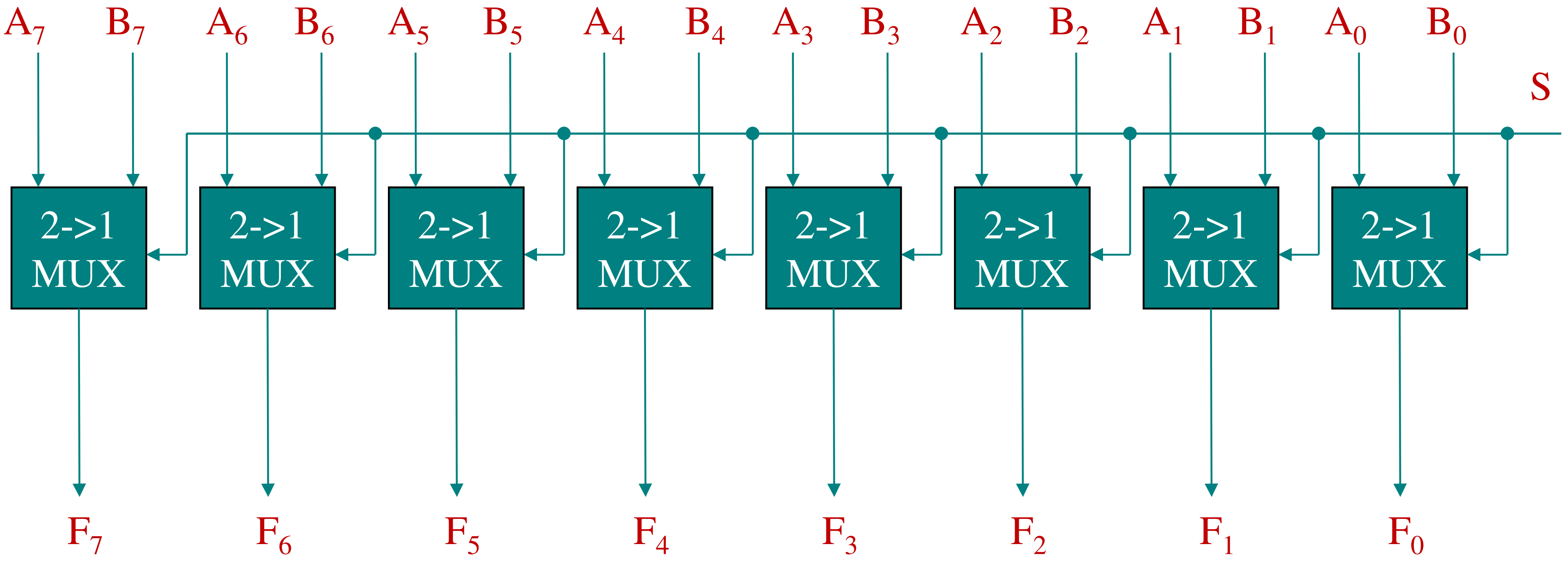
Symbol:



Gate-Level Implementation:



8-Bit 2 to 1 Mutliplexer

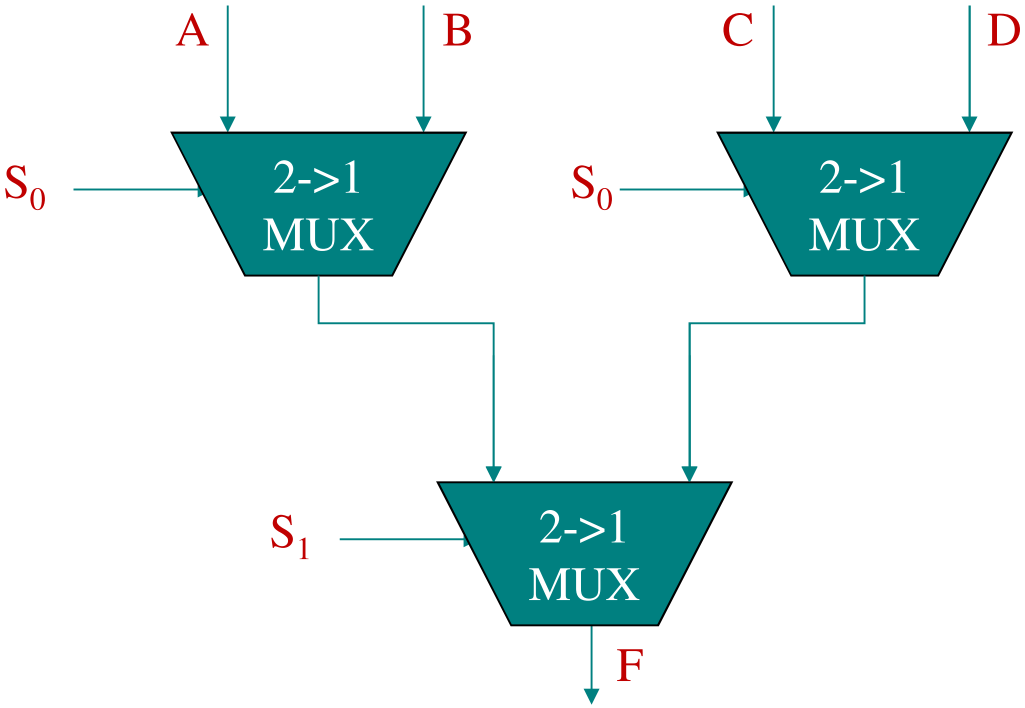


Note: The same select signal S is used to drive all MUXs so that ALL the bits of either A or B pass through

4 to 1 Multiplexer (Selector)

Operation:

S_1S_0	Function	Comment
00	F=A	Select A
01	F=B	Select B
10	F=C	Select C
11	F=D	Select D



Logic Expression:

$$F=(S_1'\cdot S_0'\cdot A)+(S_1'\cdot S_0\cdot B)+(S_1\cdot S_0'\cdot C)+(S_1\cdot S_0\cdot D)$$

Decision Making

Generating Decision Signals:

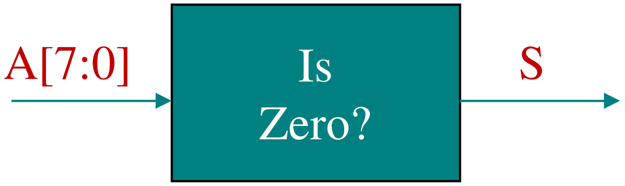
So far we have built components that manipulate binary numbers of a given width and produce a result of the same width, such as addition, subtraction, & propagation.

How do we generate single-bit signals, such as the S signal of the MUX ?

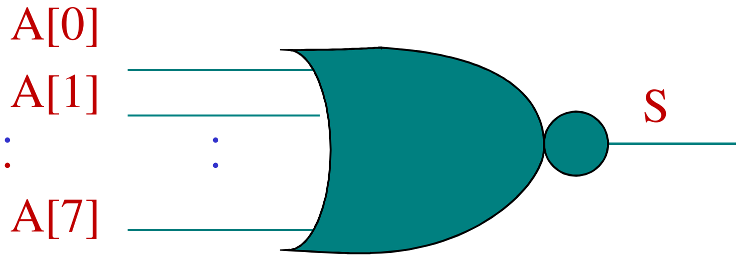
Example:

S=1 if and only if A[7:0] =00000000

Symbol:



Implementation:



Signal Propagation Through Gates

Physical Signal Implementation:

- We learned how to design circuits based on the abstract notation of "logic 1" and "logic 0"
- These two values are physically implemented using two distinct levels of voltage, e.g. 0 V for "0" and 5 V for "1" (also called "low" and "high" voltage)

Signal Transitions:

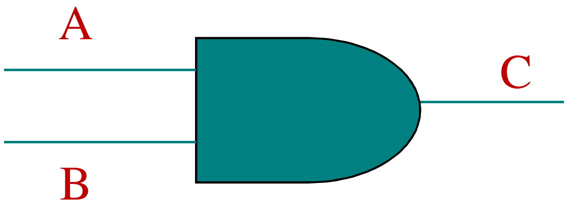
- A signal remains at a voltage level to represent 0 or 1
- Transition from high to low or from low to high signifies a change in the logic value from 1 to 0 or 0 to 1

Propagation Delay:

- The time that it takes for a logic gate to generate the output value based on the input values

Timing Diagrams

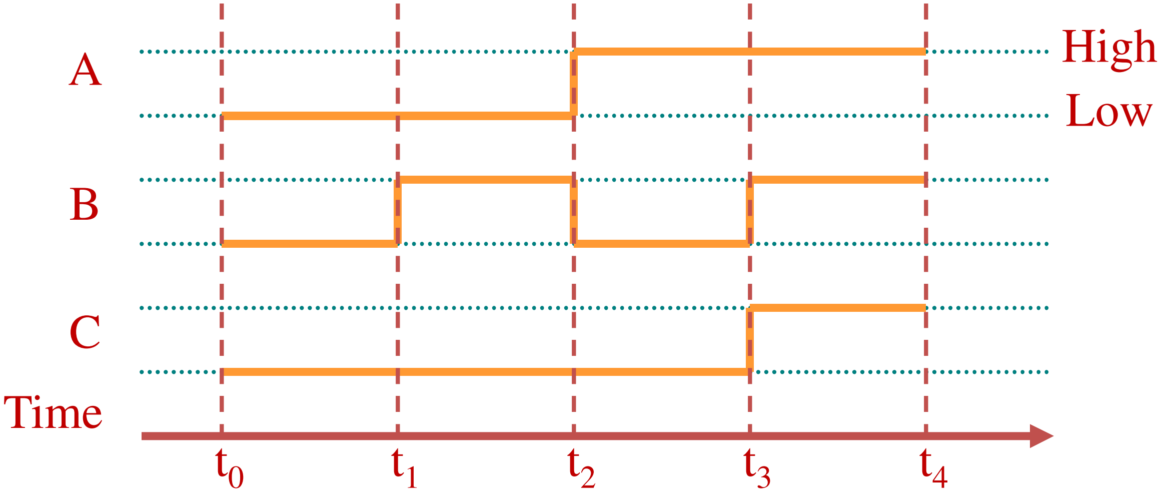
Symbolic Representation:



AND

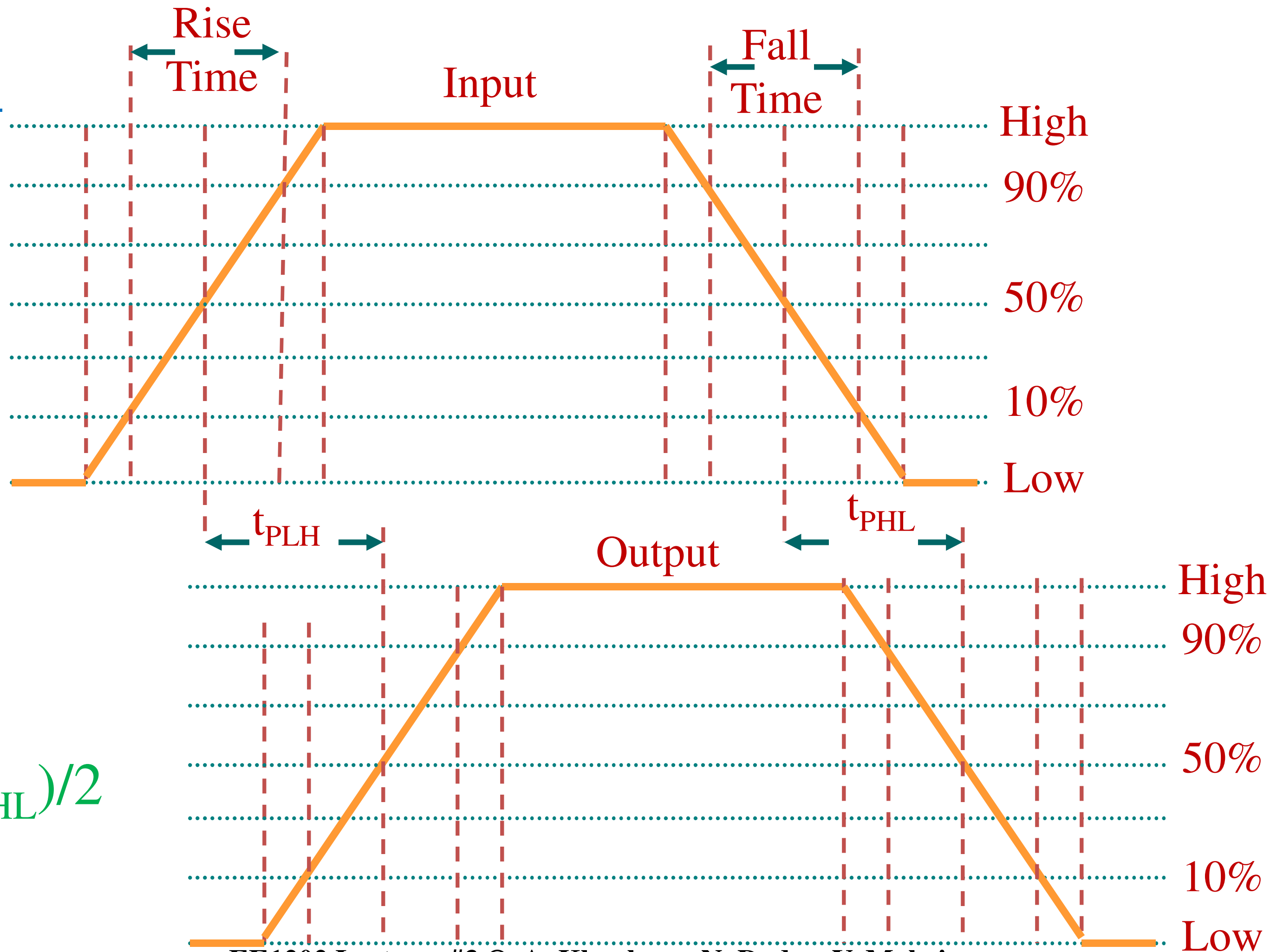
C=A·B

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



Timing Diagrams (cont'd)

Actual Timing:



$t_P=$
$(t_{PLH}+t_{PHL})/2$