ITI1100 Section Z
Digital Systems I

Chapter 4:

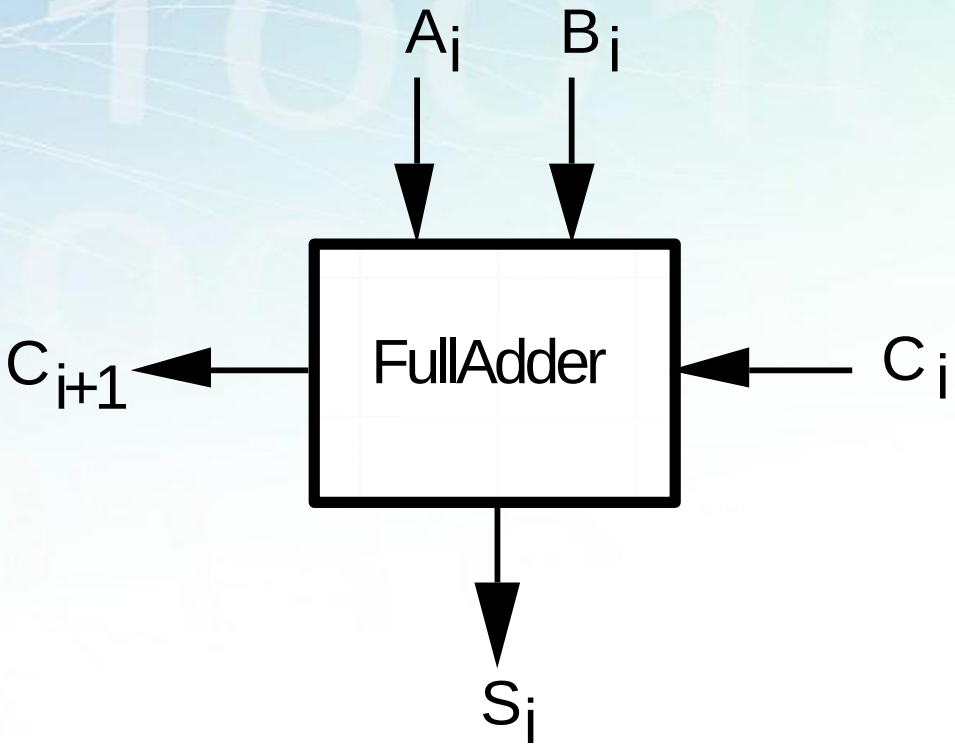
Combinational Logic (2)

Prof. Mohammad Alja'Afreh

Winter 2019

Full Adder (cont'd)

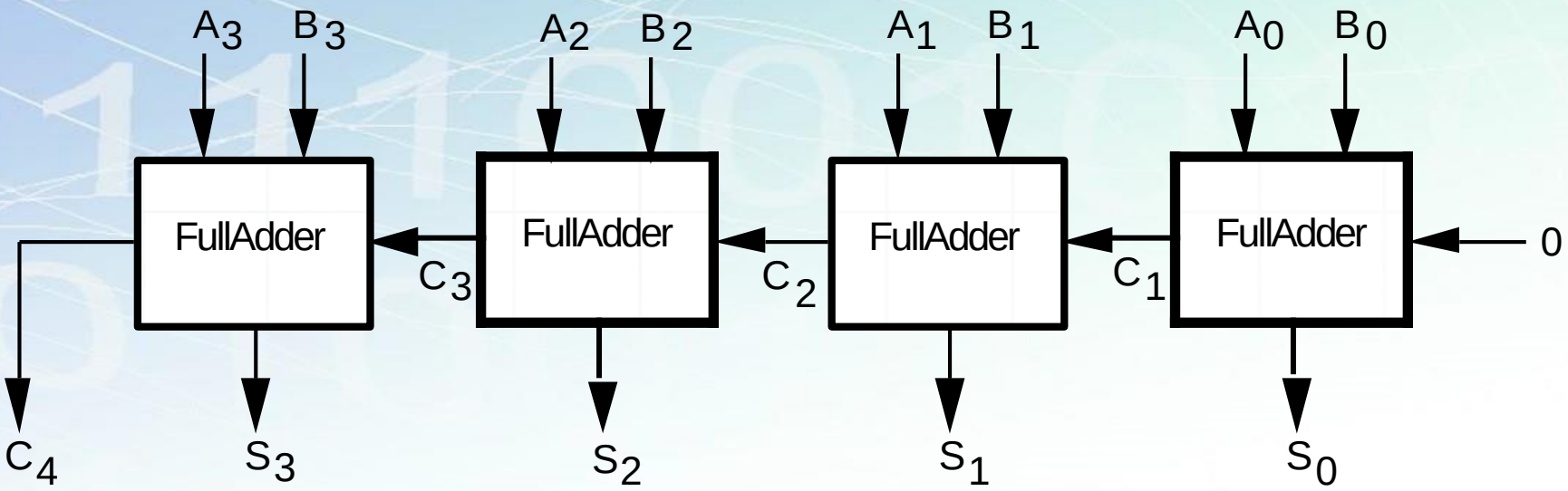
- **Putting it all together**
  - **Single-bit full adder**
  - **Common piece of computer hardware**



Block Diagram

Full Adder (cont'd)

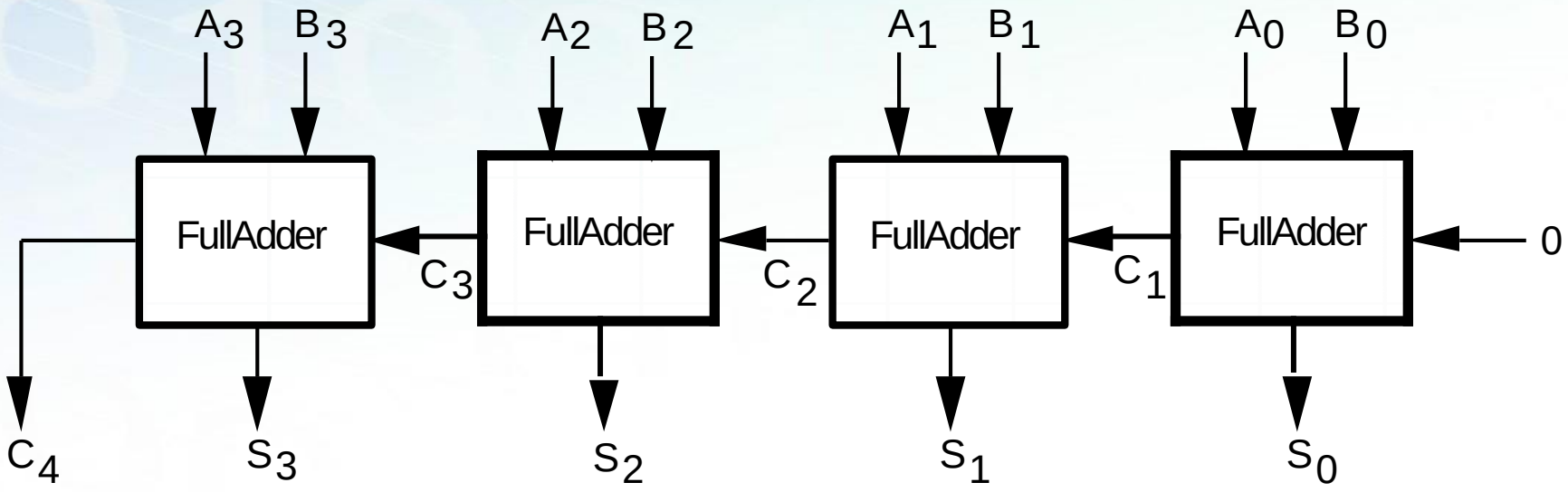
- **4-bit adder: Chain single-bit adders together**



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

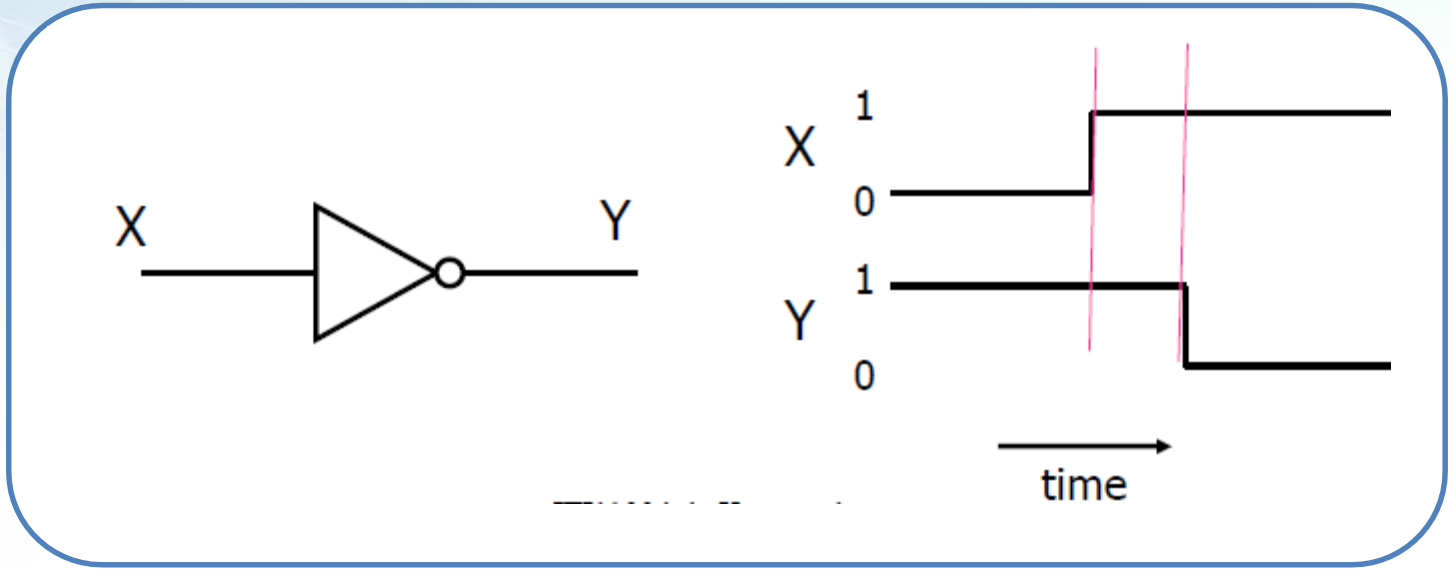
Parallel Adder

- **Add two n-bit numbers** together, **n full adders** should be **cascaded**
- **Each full-adder** represents a **column** in the long addition.
- The **carry signals** '**ripple**' through the adders from **right to left**



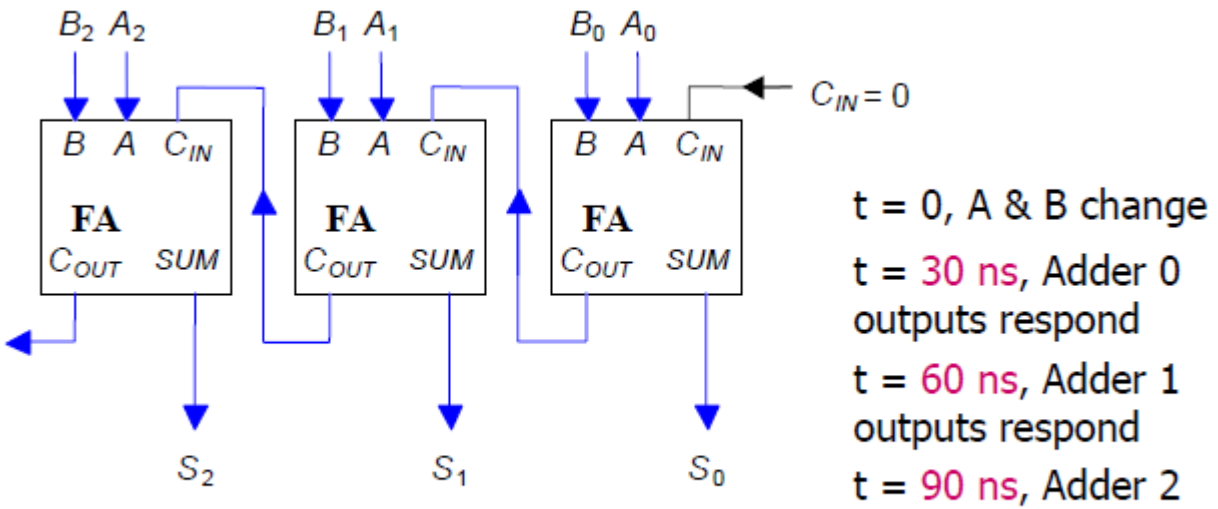
Parallel Adder (cont'd)

- All logic gates take a **non-zero time delay** to **respond** to a **change in input**
- This is the propagation delay of the gate, typically measured in **tens of nanoseconds**



Parallel Adder (cont'd)

- **A and B inputs change**, corresponding changes to **C_{IN} inputs 'ripple' through the circuit**

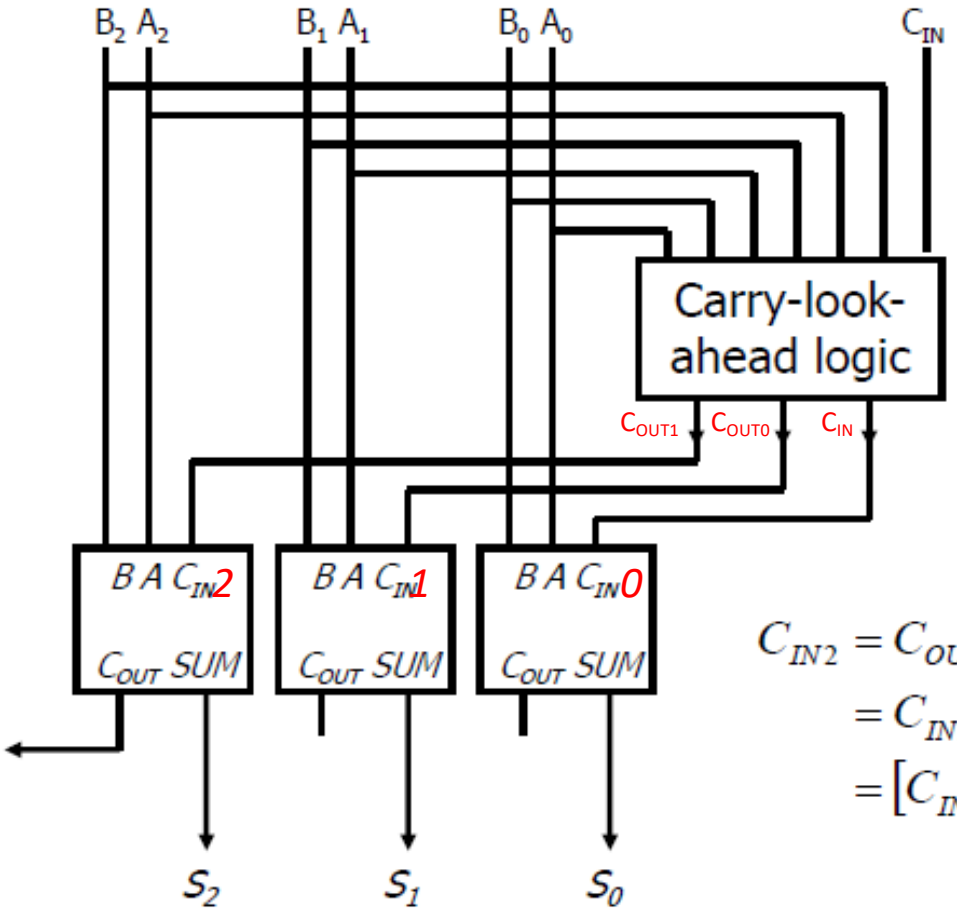


Parallel Adder (cont'd)

- The accumulated delay in large parallel adders can be very large
- **Example:** 64 bits using 30 ns full-adders:
  - 64 x 30 ns = 1.920 μs
- **Solution:** Generate the carry-input signals directly from the A and B inputs rather than using the ripple arrangement

Parallel Adder (cont'd)

- Designing a **carry-look-ahead logic circuit** for **3-bit parallel adder**



$$C_{IN0} = C_{IN}$$

$$C_{IN1} = C_{OUT0} = C_{IN}(A_0 + B_0) + A_0B_0$$

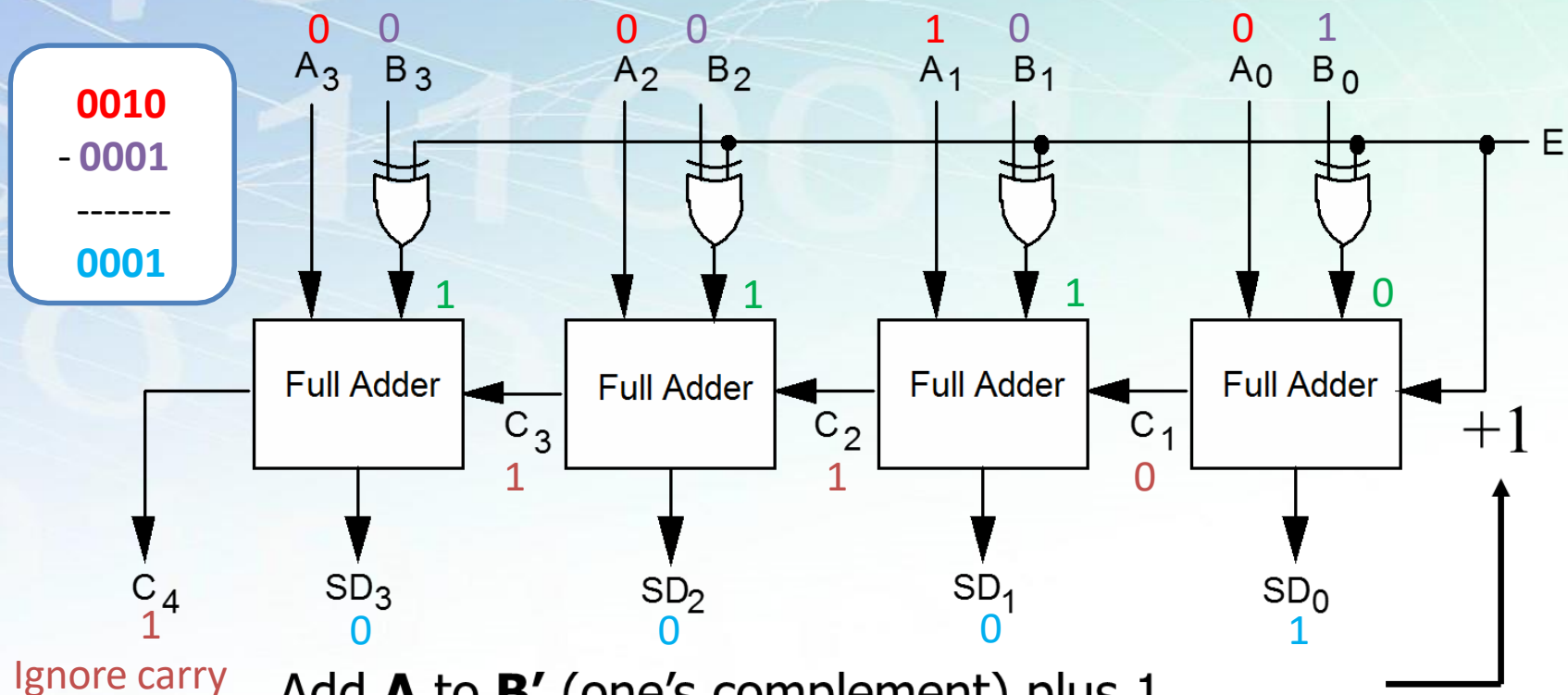
$$C_{IN2} = C_{OUT1} = C_{IN1}(A_1 + B_1) + A_1B_1 = [C_{IN}(A_0 + B_0) + A_0B_0](A_1 + B_1) + A_1B_1$$

Subtractor

- **Subtracting a number is the same as:**
  - **Perform 2's complement (2's complement = 1's complement +1)**
  - **Perform addition**
- **We can augment adder with second's complement hardware**

Subtractor (cont'd)

- **4-bit subtractor: 4 Full adders plus 4 XORs**



Add **A** to **B'** (one's complement) plus 1

That is, add **A** to two's complement of **B**

D = A - B

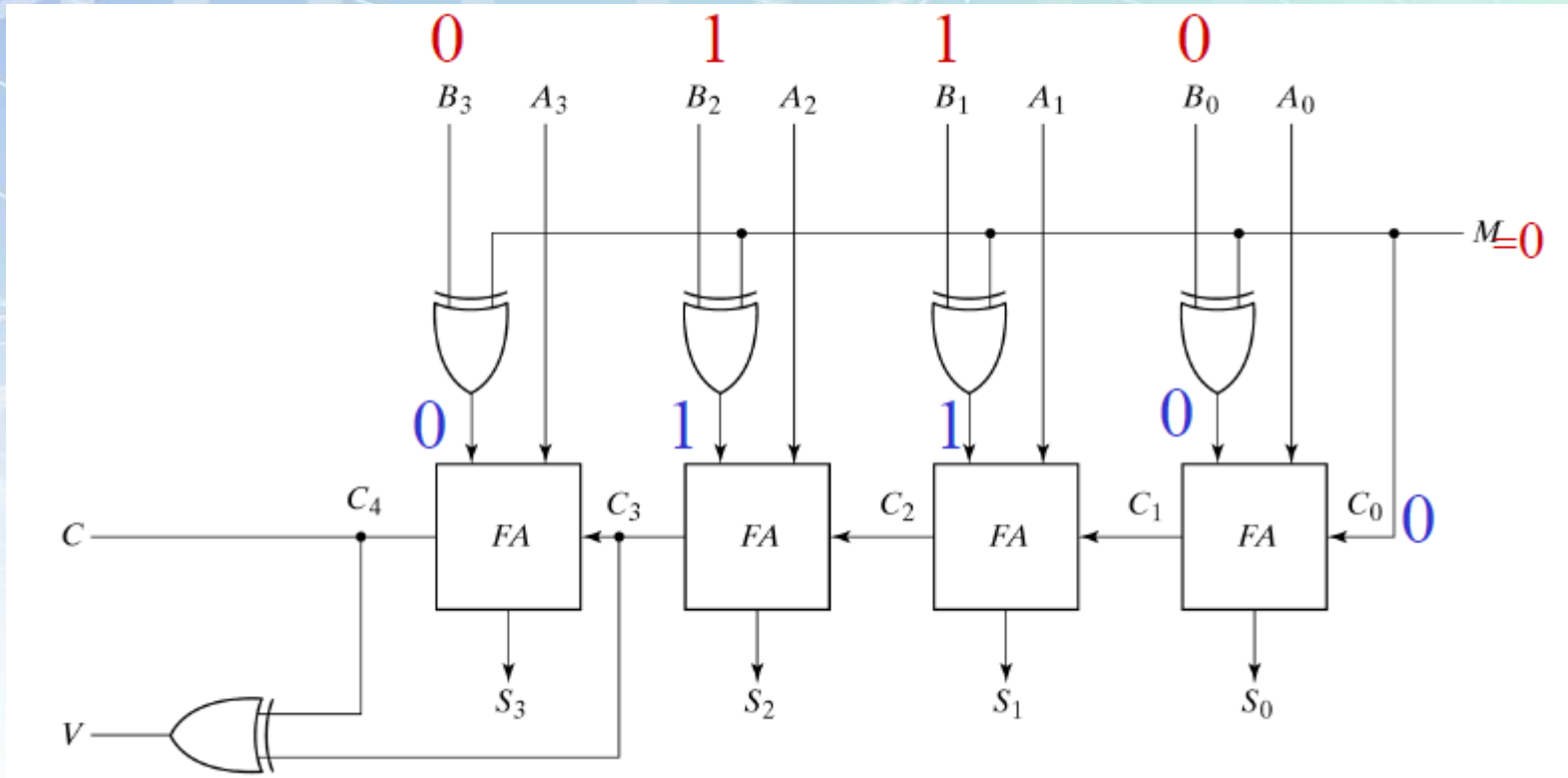
Overflow

- **Overflow (OFL)** is an important issue for computers
  - **Processors often have hardware to detect OFL**
- **OFL occurs** when two values of the **same signs are added**:
  - **Result won't fit in the number of bits provided**
  - **Result has the opposite sign**

00	01	11	10	00	11
0010	0011	1110	1101	0010	1110
0011	0110	1101	1010	1100	0100
0101	1001	1011	0111	1110	0010
2	3	-2	-3	2	-2
3	6	-3	-6	-4	4
5	-7	-5	7	-2	2
	OFL		OFL		

X: sign bit

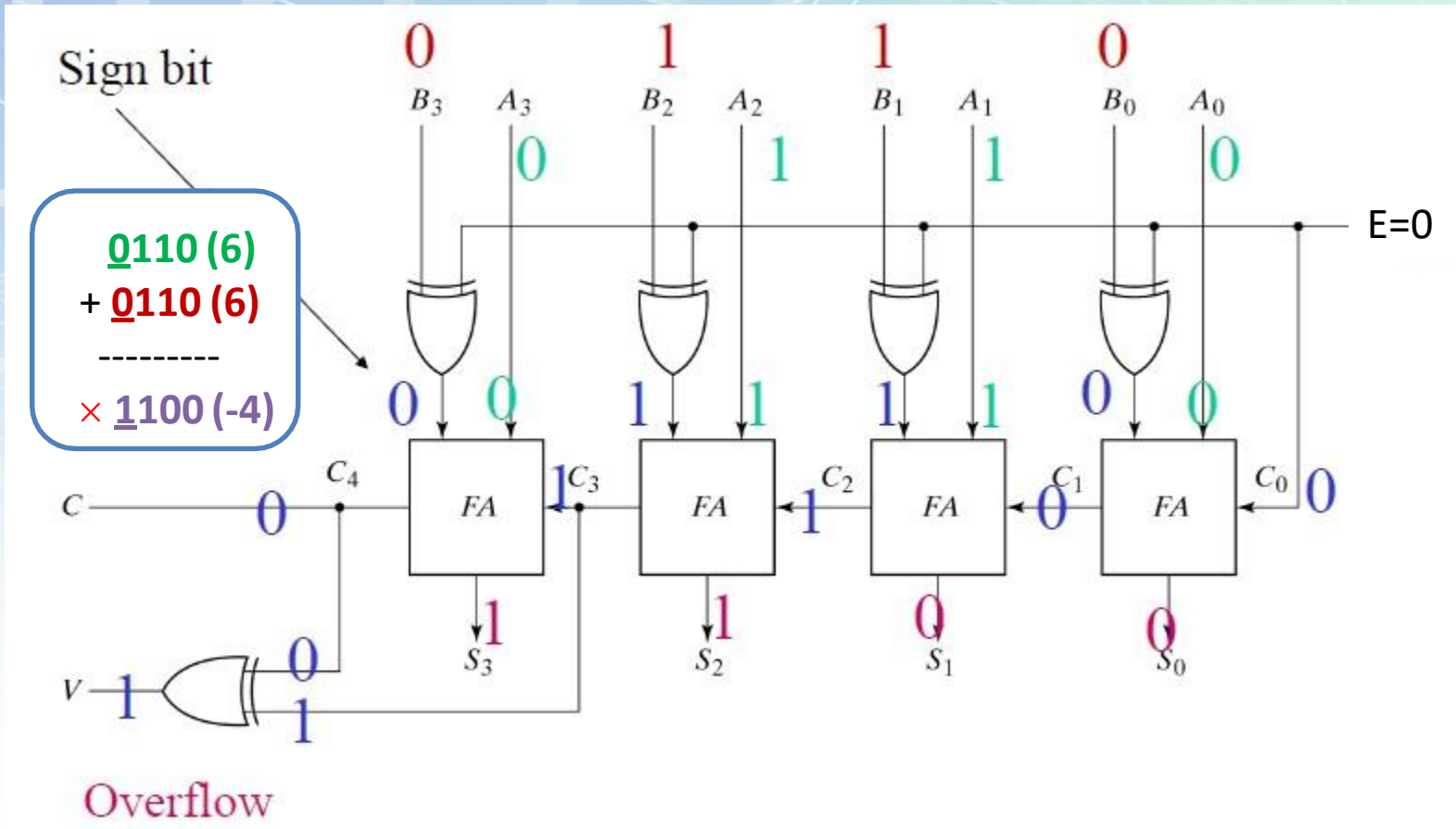
Overflow (cont'd)



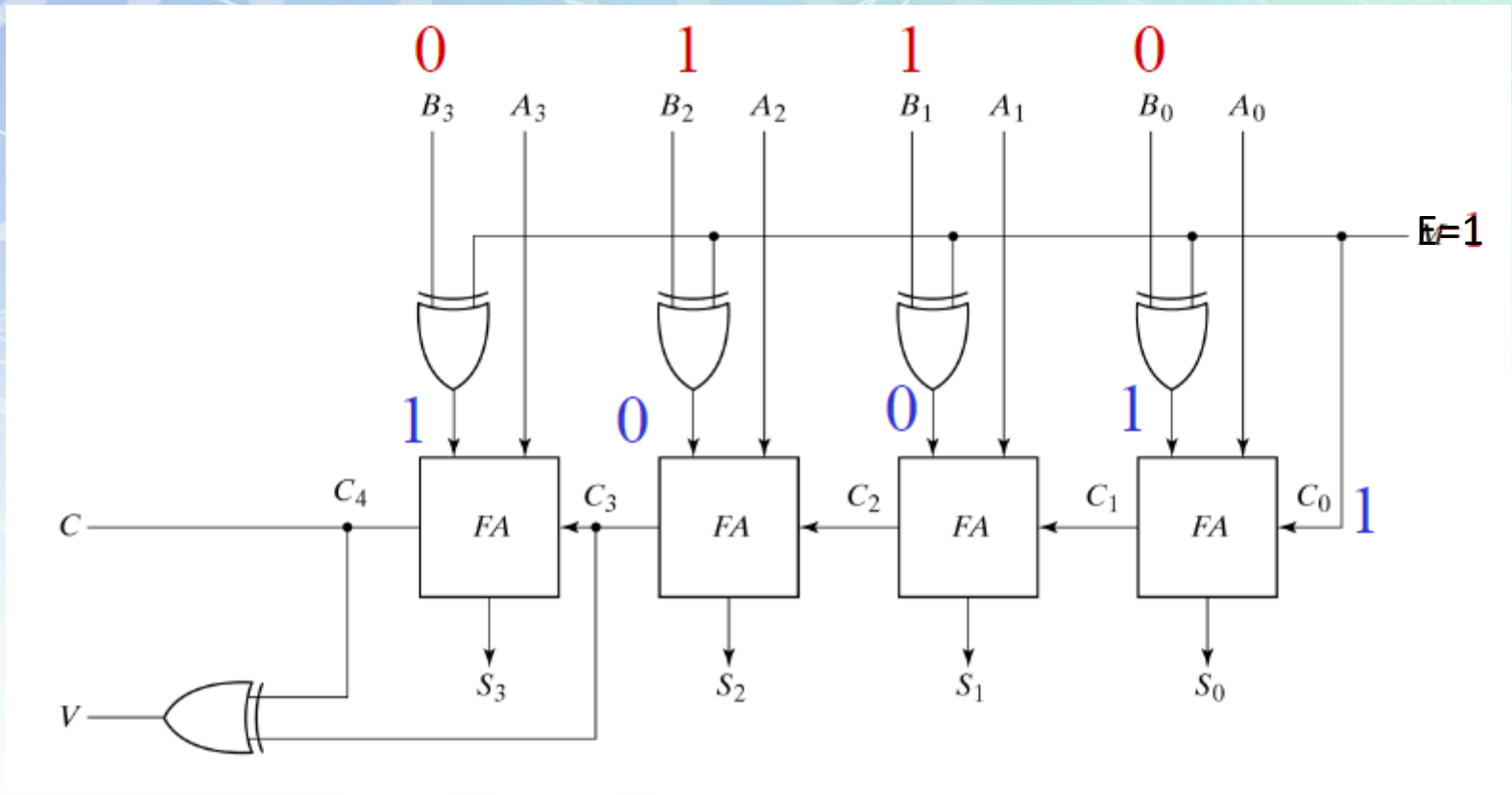
Overflow detector: V=1 means there is OFL

M=0 means 4-bit parallel adder

Overflow (cont'd)

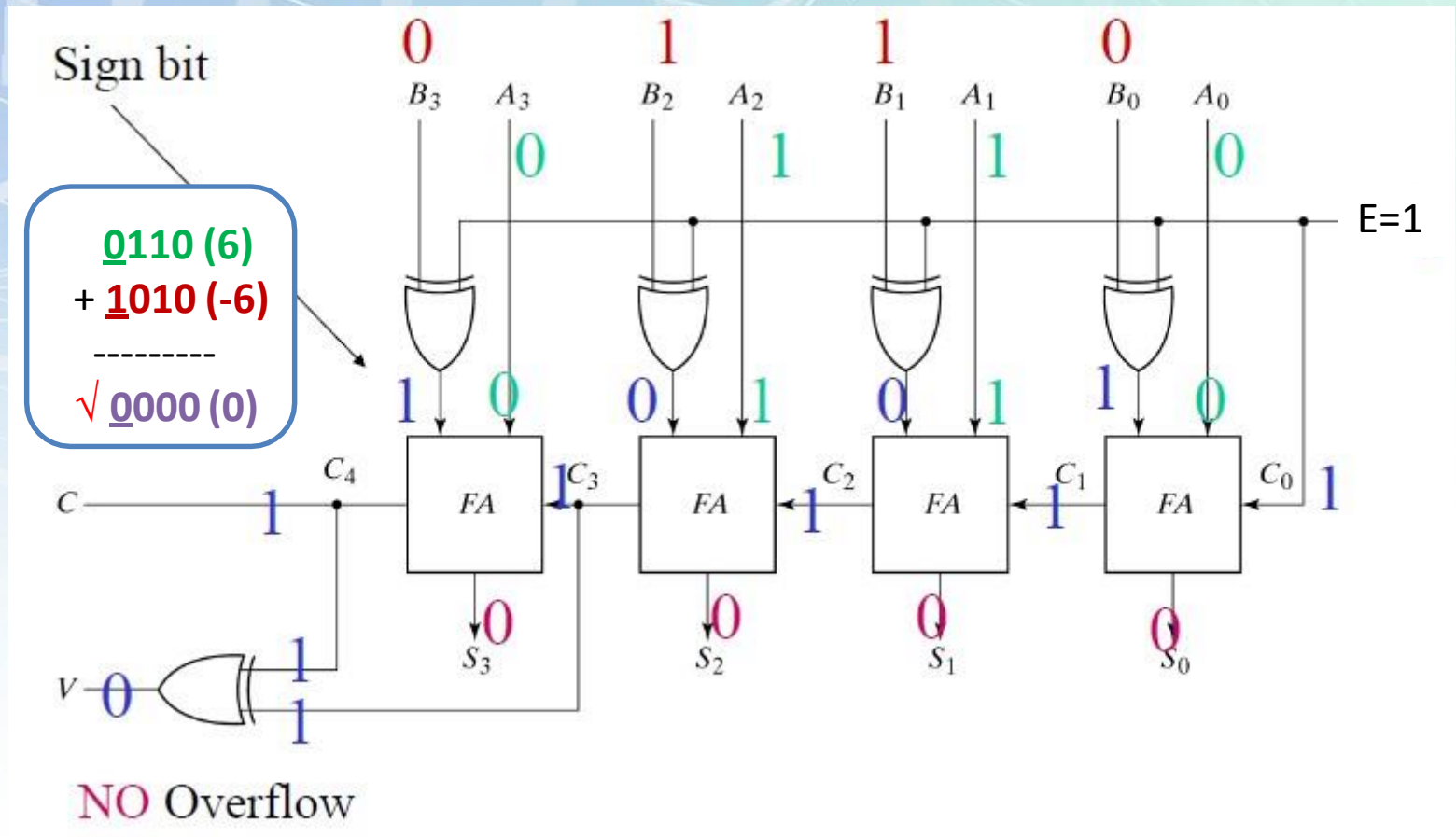


Overflow (cont'd)



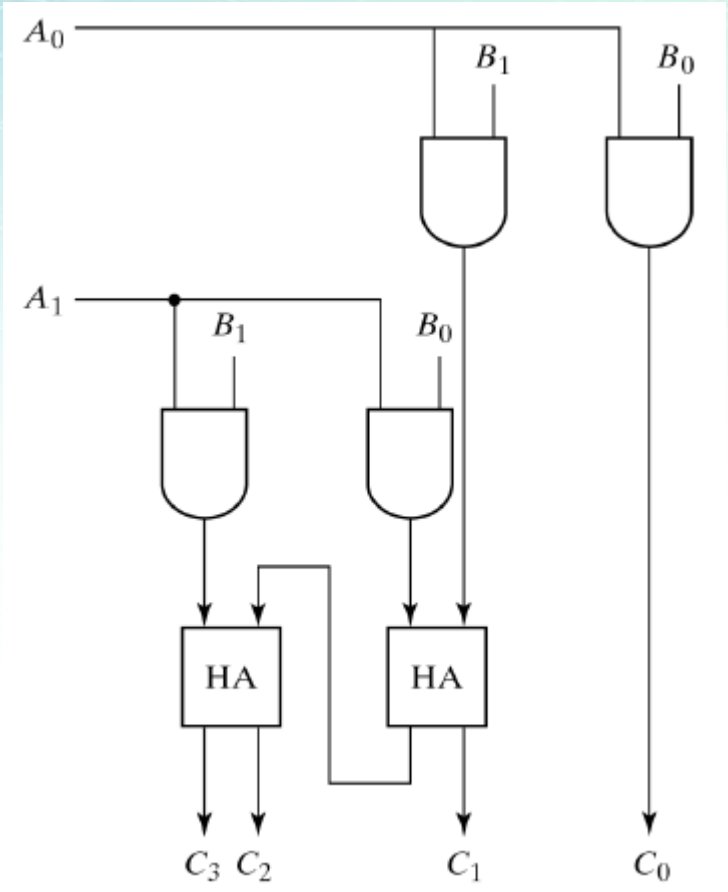
E=1 means 4-bit parallel subtractor

Overflow (cont'd)



Binary Multiplier

$$
\begin{array}{cccc}
 & & B_1 & B_0 \\
 & \times & A_1 & A_0 \\
\hline
 & & A_0B_1 & A_0B_0 \\
+ & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}
$$



- The **AND gates** produce the **partial products**
- For a **2-bit by 2-bit multiplier**, we can just use **two half adders to sum the partial products**
- C_3-C_0 are the product

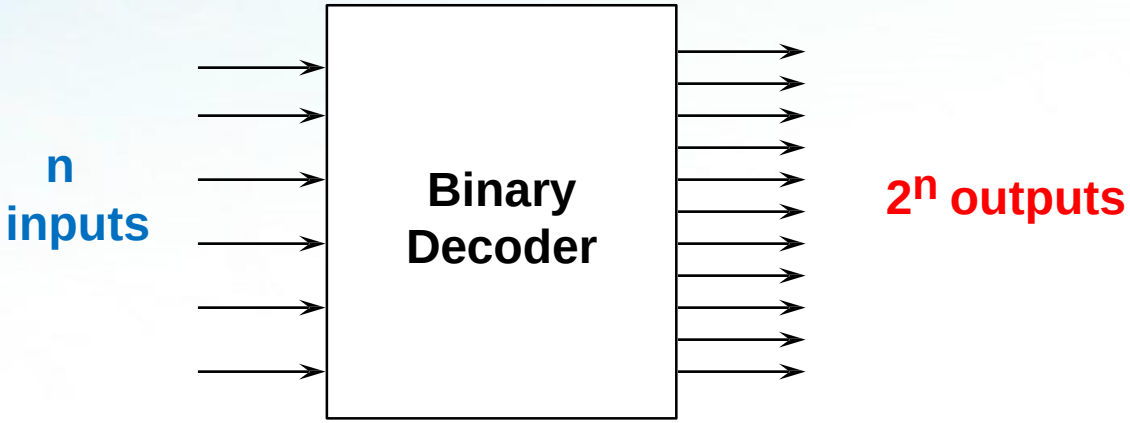
Encoders and Decoders

Overview

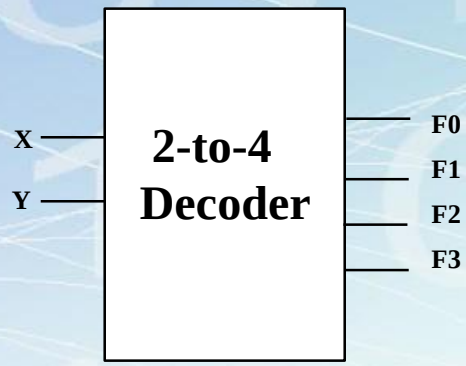
- Binary decoders
  - Converts an n-bit code to a single active output (one of the 2^n unique output lines)
  - Can be developed using AND/OR gates
  - Can be used to implement logic circuits
- Binary encoders
  - Converts one of 2^n inputs to an n-bit output
  - Can be developed using AND/OR gates
- Both encoders and decoders are extensively used in digital systems

Decoder

- A combinational circuit with **n input lines** and **2^n output lines**
- **Only one output is a 1 for any given inputs**
- BCD-to-7-segment decoder (**n=4** and **m=7**) is the case when the **n-bit coded information has unused combinations** $\Rightarrow$ **less than 2^n outputs**



Implementation of 2-to-4 Decoder

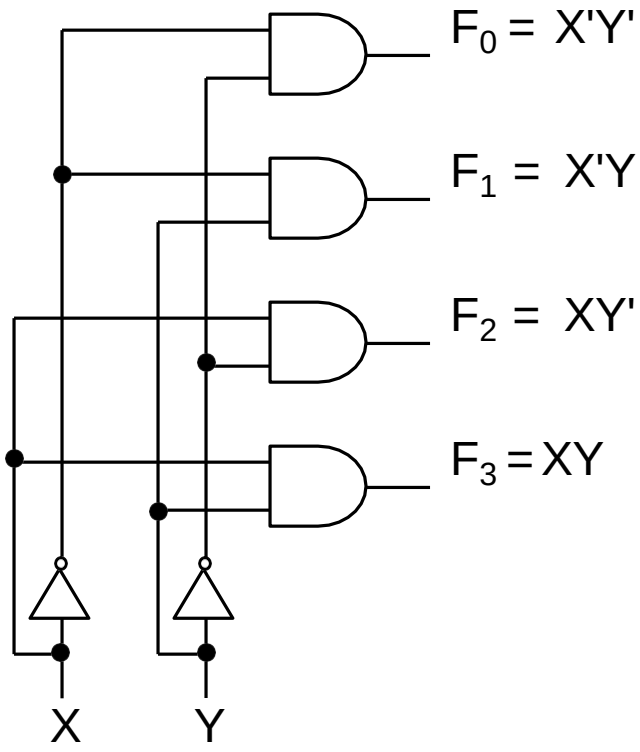


Truth Table:

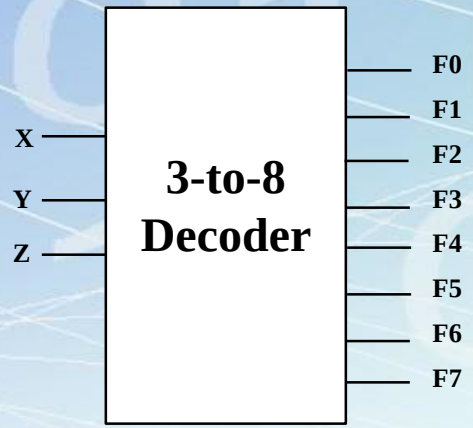
X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

- **From truth table, each output is a 2-variable minterm (X'Y', X'Y, XY',or XY)**

- **Circuit for 2x4 decoder is:**

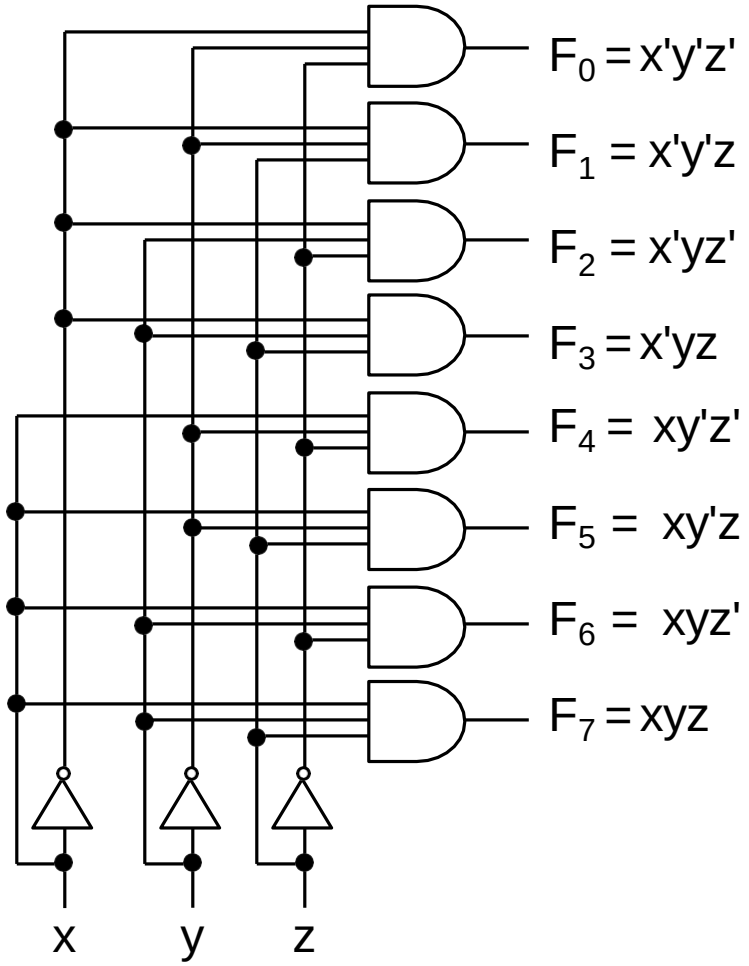


Implementation of 3-to-8 Decoder



Truth Table:

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Implementing Boolean Functions Using Decoders

- **Any combinational circuit** can be **implemented with decoders and OR gates**
- A **single n-to-2^n decoder** generates **2^n minterms.** Decoders are sometimes called **minterm generators**
- **OR gate** forms the **sum**
- The **output lines of the decoder** corresponding to the **minterms** of the function are **used as inputs** to the **OR gate** to form **SOP expression**

Implementing Boolean Functions Using Decoders (cont'd)

- **Example: Full adder**

  $S(x, y, z) = \Sigma(1,2,4,7)$

  $C(x, y, z) = \Sigma(3,5,6,7)$

Since there are 3 inputs and a total of 8 minterms, we need a 3-to-8 decoder

x	y	z	C	S
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