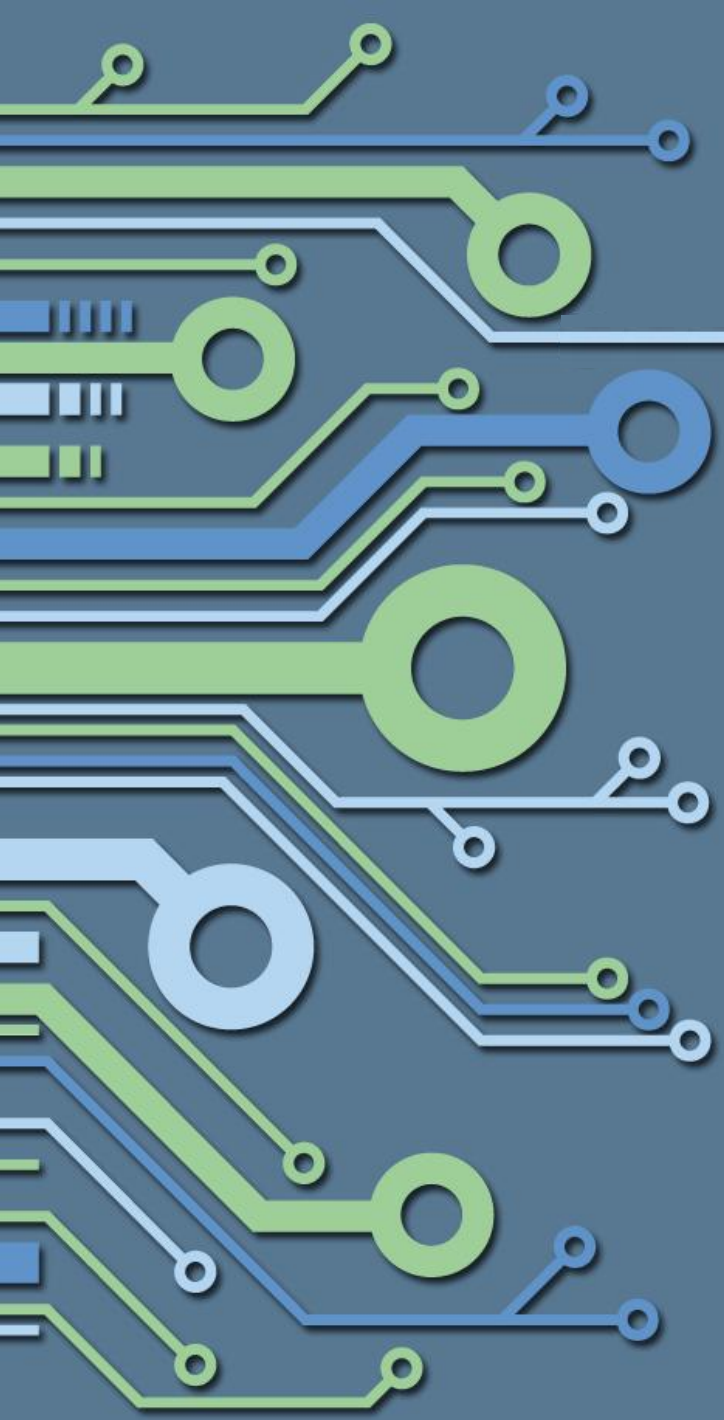


# Chapter 7

## COMBINATIONAL LOGIC BUILDING BLOCKS

# Introduction

- Number of gates in SSI is limited by the number of pins in the package.
- The cost of ICs is determined by the number and types of ICs employed and the number of interconnections needed to implement the given digital circuit.

# Cost Reduction Methods

- MSI Components
  - standard circuits
  - perform specific digital functions commonly needed in the design of digital systems
- PLD
  - an IC with internal logic gates that are connected through electric fuses
  - can be programmed to incorporate complex logic function with one LSI circuit

# Examples of Combinational Circuits

- Adders
  - Half-Adder
  - Full-Adder
- Subtracter
  - Half-Subtracter
  - Full-Subtracter
- Magnitude Comparators
- Decoders
- Demultiplexers
- Encoders
- Multiplexers
- Code Converters
- Parity Generators or Checkers

# Adders

- Half-Adder
  - a device which accepts two binary digits as inputs and produces two binary digits as outputs, a sum bit and a carry bit
- Full-Adder
  - a device which accepts three inputs including an input carry and generates a sum output and an output carry

# Adders

- Serial
  - One full-adder circuit and a storage device to hold the generated output carry
  - The stored output carry from one pair of bits is used as input carry for the next pair of bits.
- Parallel
  - Uses n full-adder circuits
  - Output carry from one full-adder is connected to the input carry of the full-adder one position to its left

# Addition of rightmost bit only


| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| ---- | ---- | ---- | ---- |

# Addition of rightmost bit only

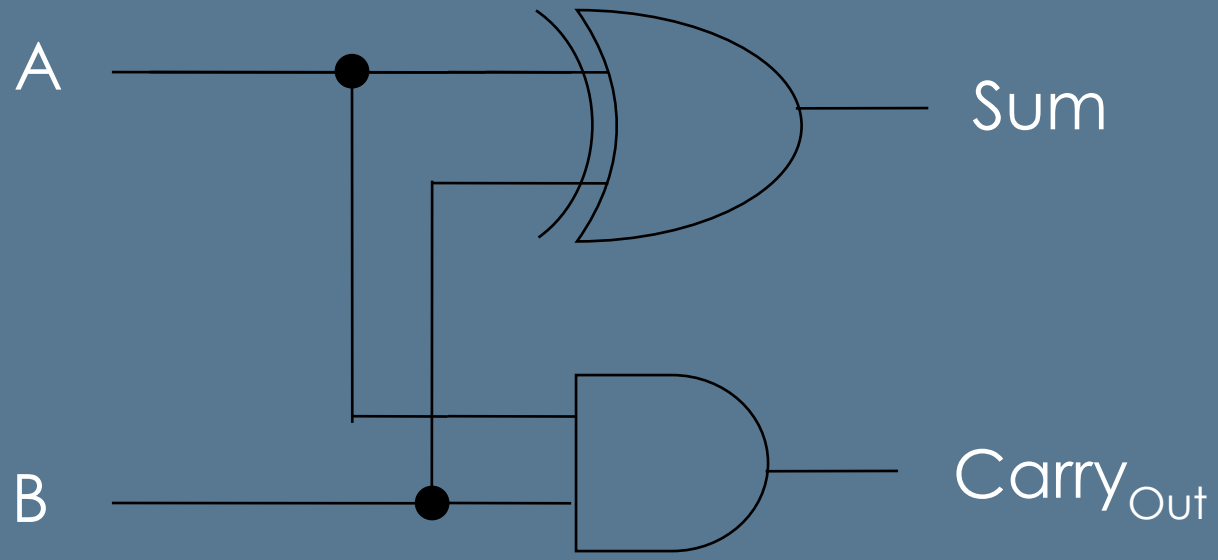
| | | | | |
|---|---|---|---|---|
| $Carry_{out}$ → | 0 | 1 | 1 | 1 | |
| A → | | 0 | 0 | 1 | 1 |
| B → | | 0 | 1 | 0 | 1 |
| | | ---- | ---- | ---- | ---- |
| Sum → | | 1 | 0 | 0 | 0 |

# Addition of rightmost bit only

| | | | | |
|---|---|---|---|---|
| $Carry_{out}$ → | 0 | 1 | 1 | 1 |
| A → | | 0 | 0 | 1 |
| B → | | 0 | 1 | 0 |
| | | --- | --- | --- |
| Sum → | | 1 | 0 | 0 |

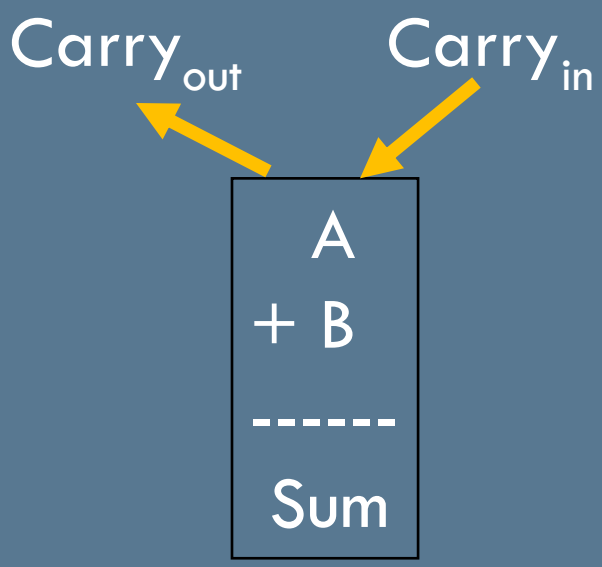
| A | B | $C_{out}$ | Sum |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Half-adder



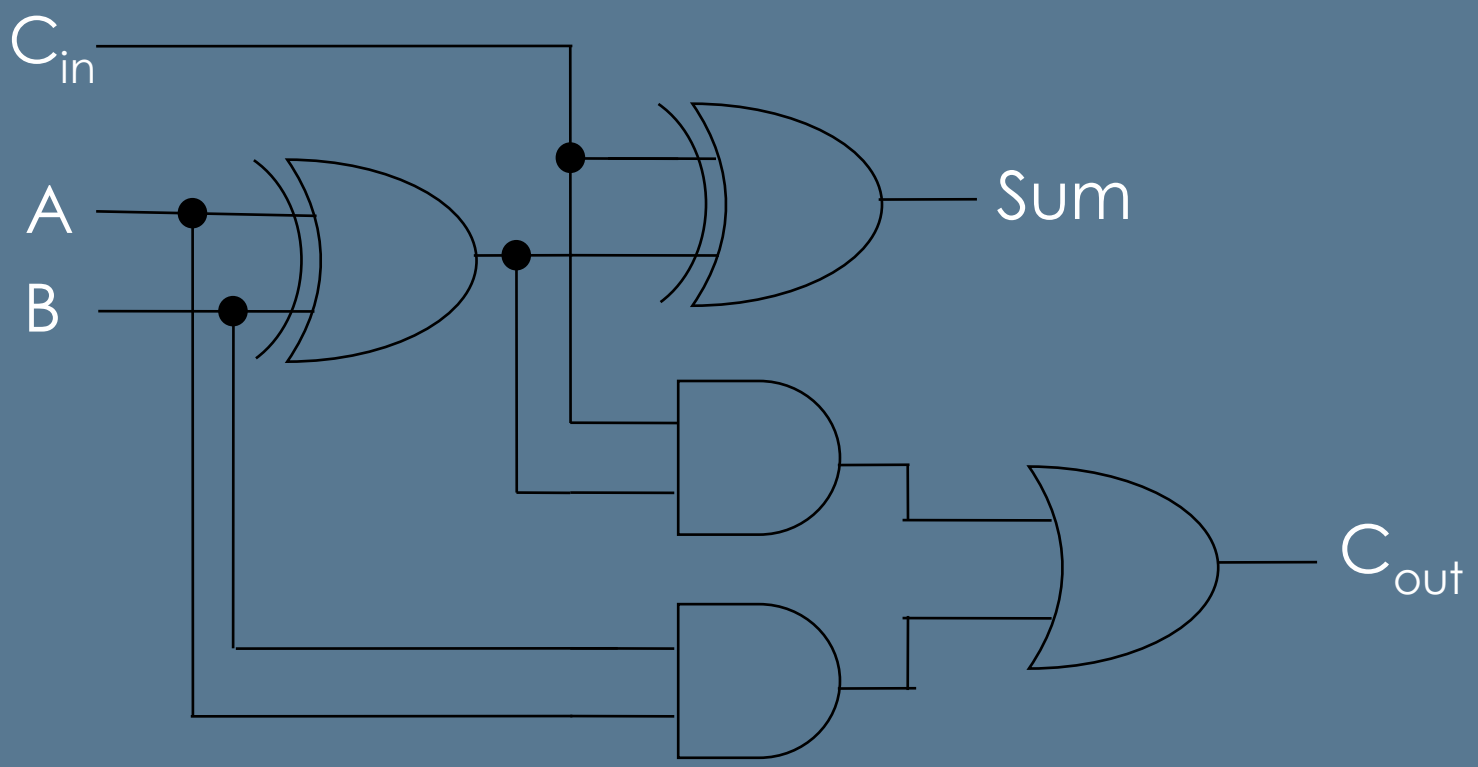

$$Sum = A \oplus B$$
$$C_{out} = A\,B$$

# One-bit full-adder




| $C_{in}$ | A | B | $C_{out}$ | Sum |
|---|---|---|---|---|
| 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 |

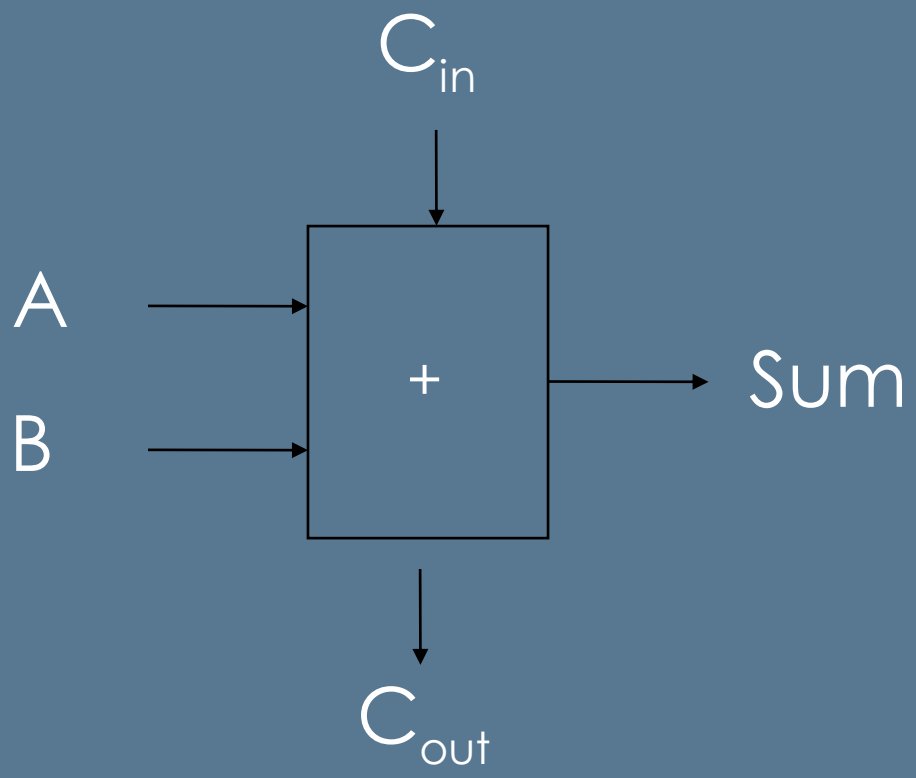
# Full-adder logic




$\text{Sum} = (A \oplus B) \oplus C_{in}$

$C_{out} = AB + C_{in}(A \oplus B)$

# Block diagram of one-bit full-adder



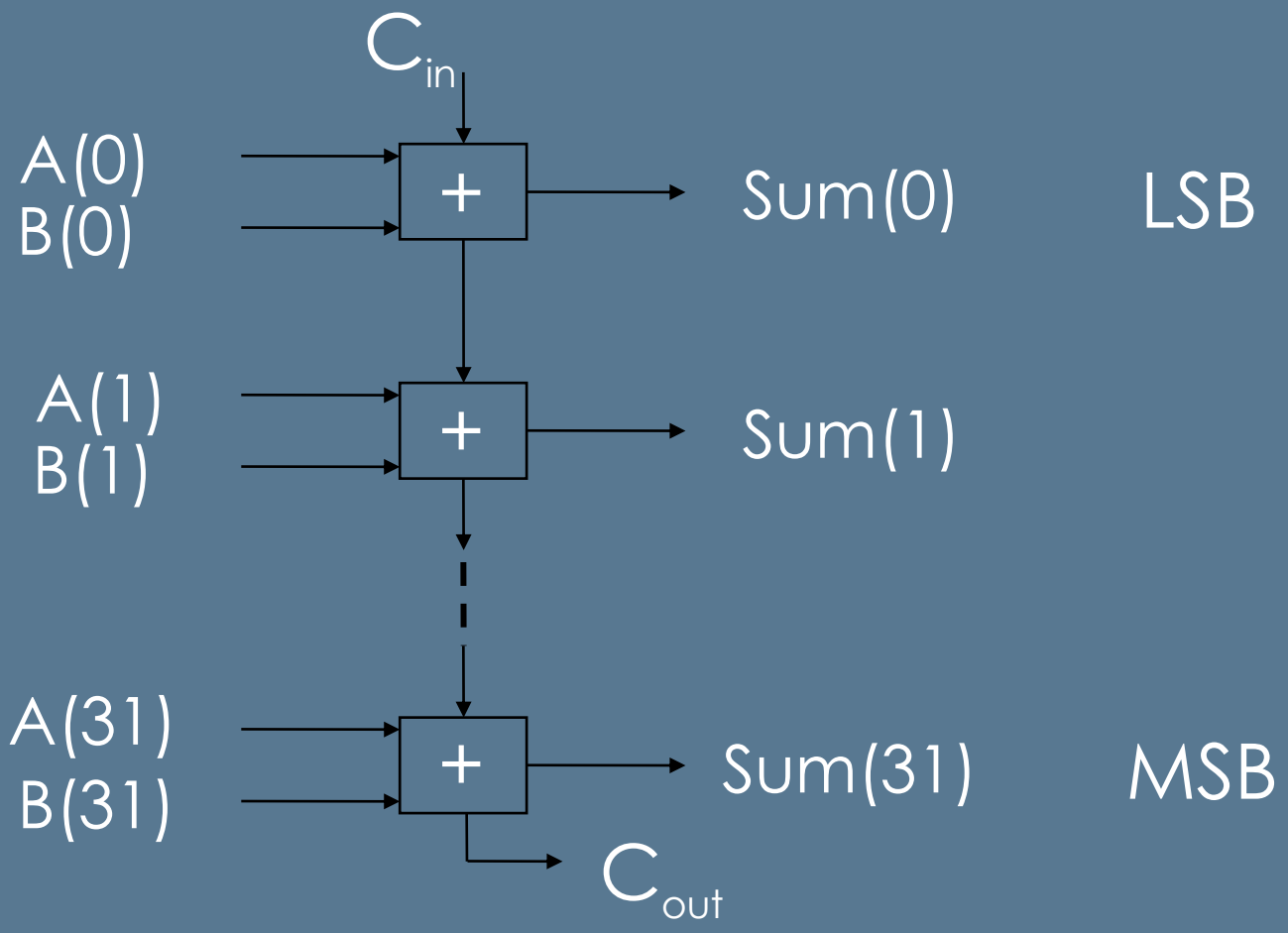
# Cascading n full-adder circuits

- The full-adder circuit can easily be cascaded to implement addition of up to n-bit inputs:

$$\begin{array}{cccccccc} & A_n & \ldots & A_3 & A_2 & A_1 & A_0 \\ + & B_n & \ldots & B_3 & B_2 & B_1 & B_0 \\ \hline C & S_n & \ldots & S_3 & S_2 & S_1 & S_0 \end{array}$$

Carry out from $S_n$

# Example: add two 32-bit numbers

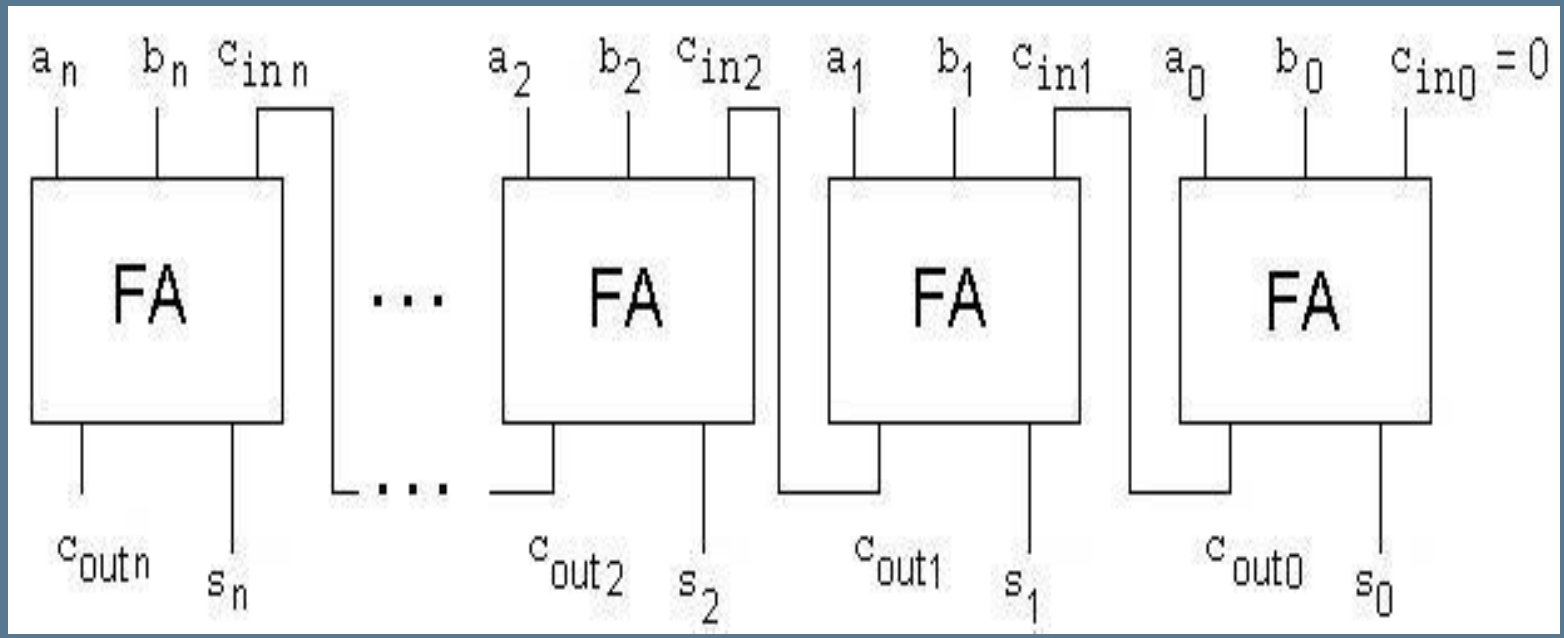


# Cascading n-bit full-adder circuits

- To perform n-bit addition, you'll need to cascade n full-adder circuits, with each circuit waiting for the output ($C_{out}$) from a previous full-adder unit before it can compute the sum. This results into a very slow adder with a lot of data hazards.

# Cascading n-bit full-adder circuits

- Solution: Implement "carry look-ahead"



# Carry Propagation

- It is a method used to determine the $N^{th}$ carry-in without waiting for the $(N-1)^{th}$ full-adder to finish its operation.

  $Sum = (A \oplus B) \oplus C_{in}$ $\quad$ $C_{out} = AB + C_{in}(A \oplus B)$

# Carry Propagation

- It is a method used to determine the $N^{th}$ carry-in without waiting for the $(N-1)^{th}$ full-adder to finish its operation.

$$Sum = (A \oplus B) \oplus C_{in} \qquad C_{out} = AB + C_{in}(A \oplus B)$$

$$\text{Let } P = A \oplus B \quad \text{and} \quad G = AB$$

# Carry Propagation

- It is a method used to determine the $N^{th}$ carry-in without waiting for the $(N-1)^{th}$ full-adder to finish its operation.

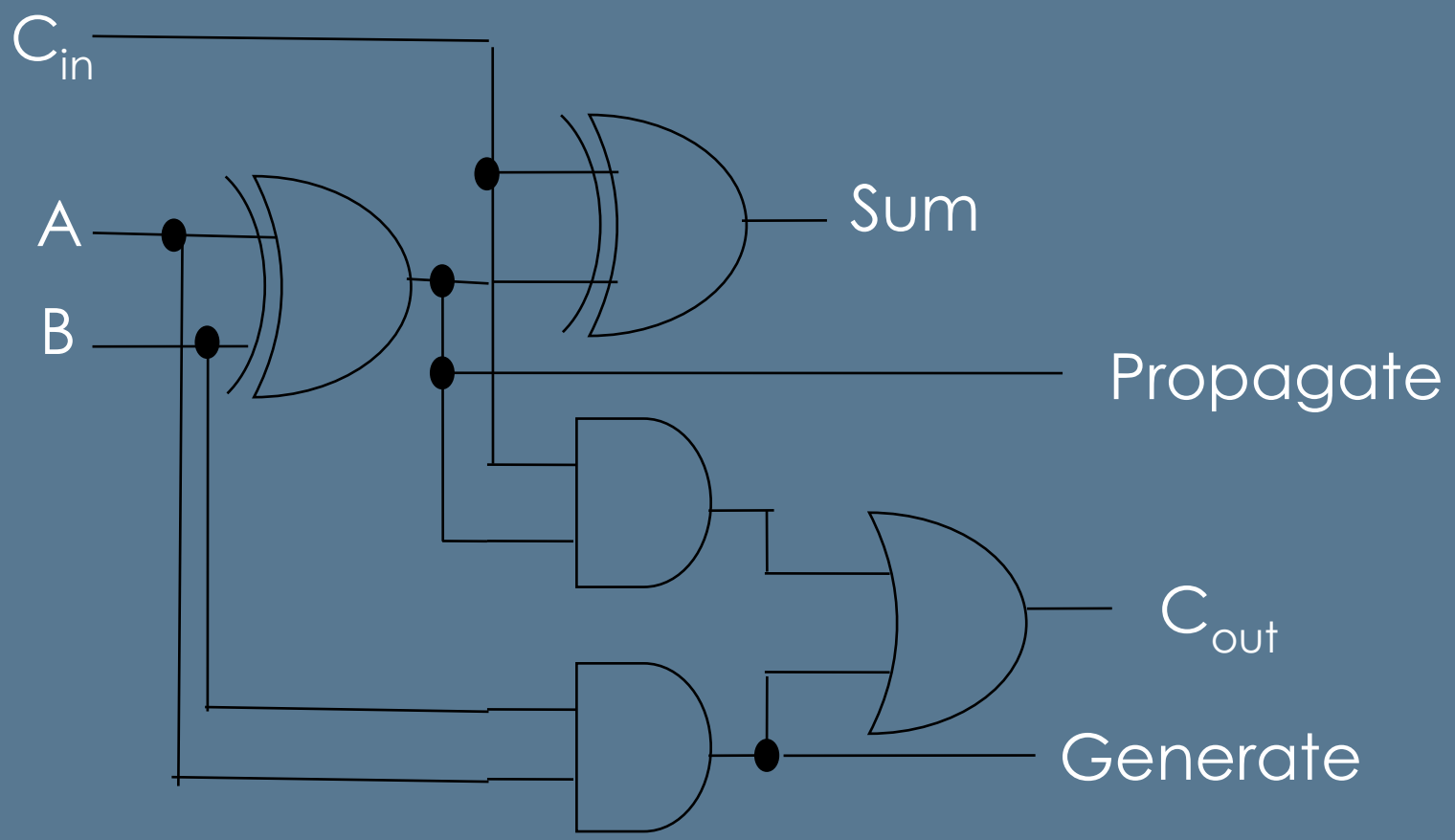
$Sum = (A \oplus B) \oplus C_{in}$ $\quad C_{out} = AB + C_{in}(A \oplus B)$

Let $P = A \oplus B$ and $G = AB$

$Sum = P \oplus C_{in}$ $\quad P =$ propagate

$C_{out} = G + PC_{in}$ $\quad G =$ generate

# Carry Propagation



# Recall: Full-adder circuit

$S = (A \oplus B) \oplus C_{in}$

$C_{out} = AB + C_{in}(A \oplus B)$

Let $P = A \oplus B$ and $G = AB$

$Sum = P \oplus C_{in}$

$C_{out} = G + PC_{in}$

# Recall: Full-adder circuit

$S = (A \oplus B) \oplus C_{in}$ $\quad C_{out} = AB + C_{in}(A \oplus B)$

Let $P = A \oplus B$ and $G = AB$

$Sum = P \oplus C_{in}$ $\quad C_{out} = G + PC_{in}$

For n-bit addition:

$S_0 = P_0 \oplus C_{in0}$ and $C_{out0} = G_0 + P_0 C_{in0}$

$S_1 = P_1 \oplus C_{in1}$ and $C_{out1} = G_1 + P_1 C_{in1}$

# Recall: Full-adder circuit

$S = (A \oplus B) \oplus C_{in}$ $\quad C_{out} = AB + C_{in}(A \oplus B)$

Let $P = A \oplus B$ and $G = AB$

$Sum = P \oplus C_{in}$ $\quad C_{out} = G + PC_{in}$

For n-bit addition:

$S_0 = P_0 \oplus C_{in0}$ and $C_{out0} = G_0 + P_0C_{in0}$

$S_1 = P_1 \oplus C_{in1}$ and $C_{out1} = G_1 + P_1C_{in1}$

$= G_1 + P_1(G_0 + P_0C_{in0})$

# Recall: Full-adder circuit

$S_0 = P_0 \oplus C_{in0}$ and $C_{out0} = G_0 + P_0 C_{in0}$

$S_1 = P_1 \oplus C_{in1}$ and $C_{out1} = G_1 + P_1 C_{in1}$

$= G_1 + P_1(G_0 + P_0 C_{in0})$

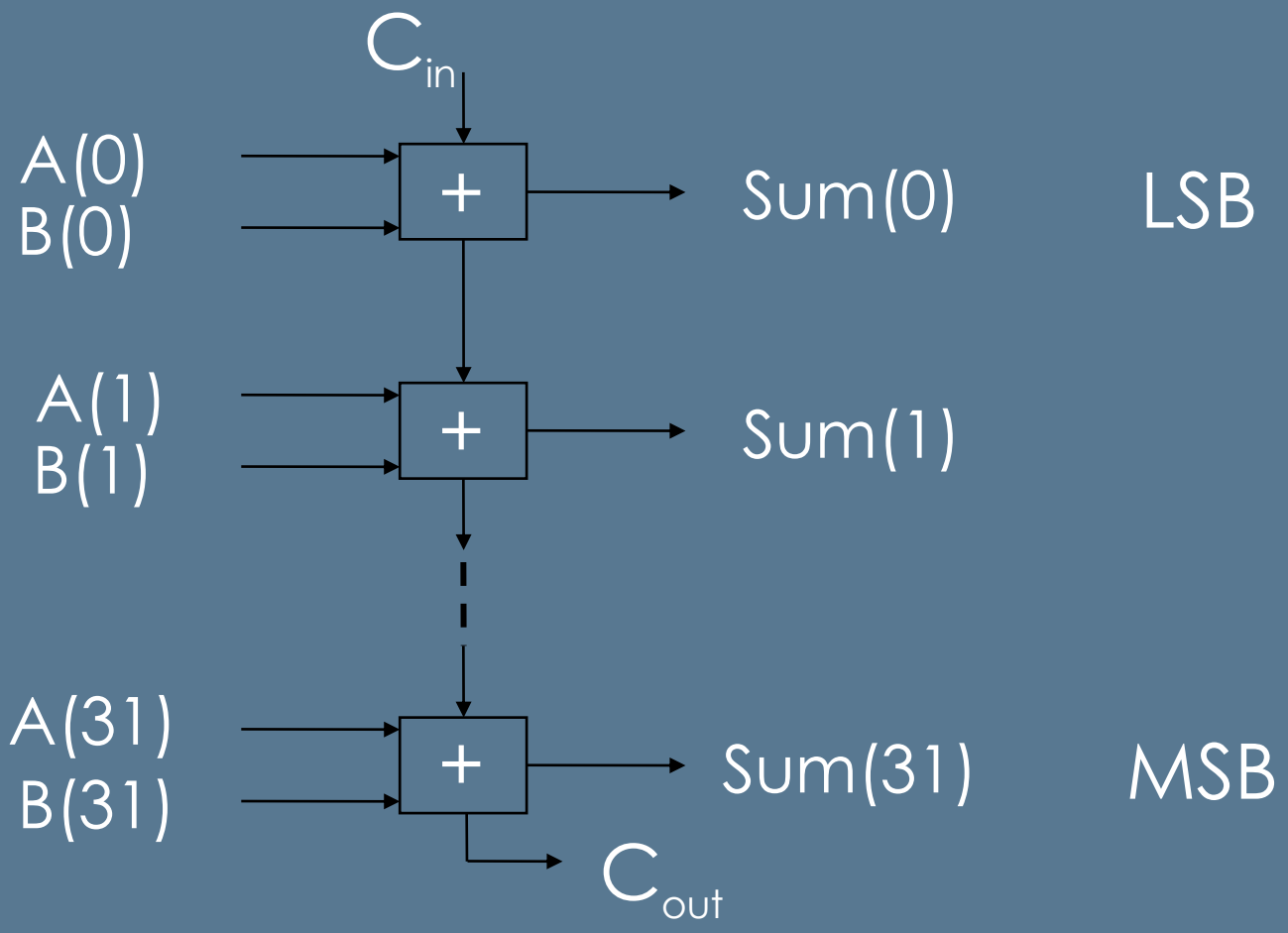
$S_2 = P_2 \oplus C_{in2}$ and $C_{out2} = G_2 + P_2 C_{in2}$

$= G_2 + P_2(G_1 + P_1(G_0 + P_0 C_{in0}))$

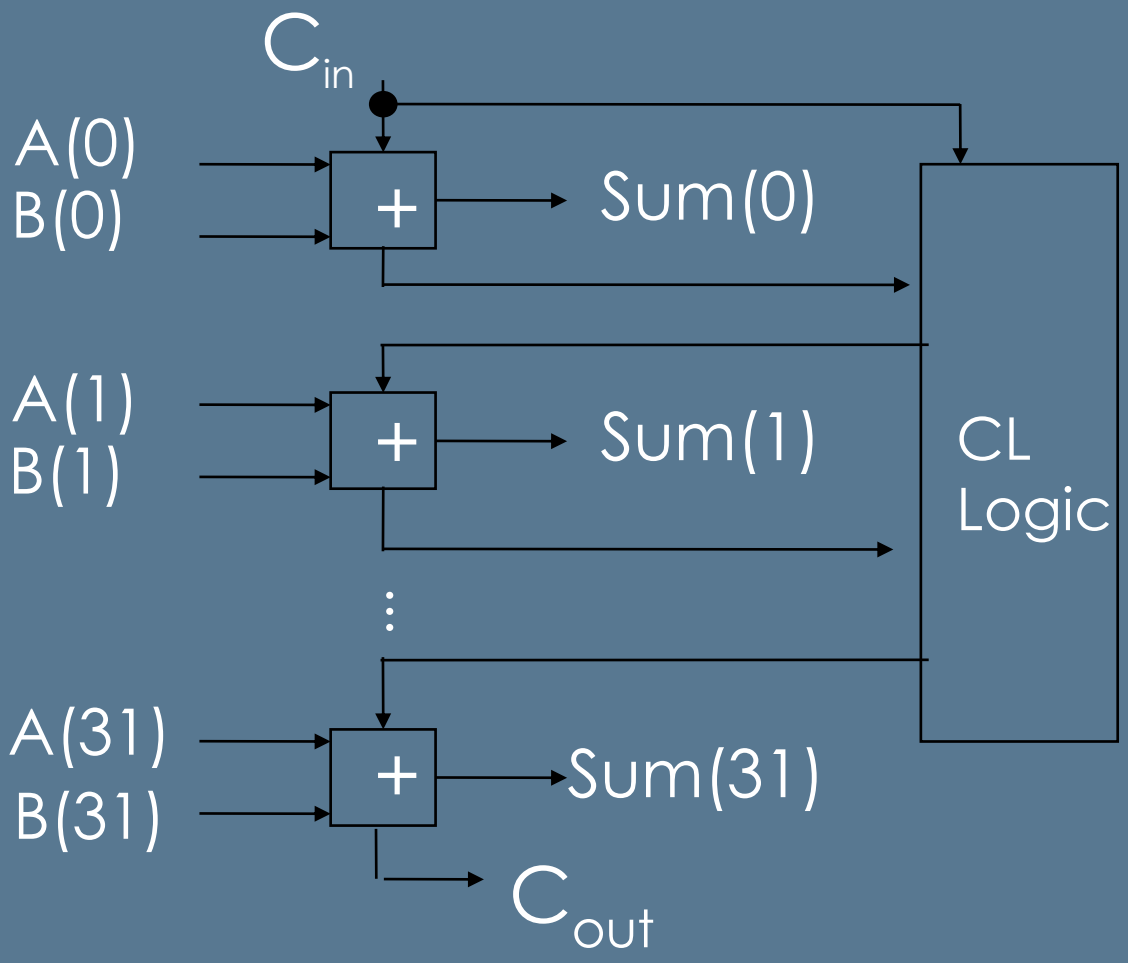
... ...

$S_n = P_n \oplus C_{inn}$ and $C_{outn} = G_n + P_n C_{inn}$

# Example: add two 32-bit numbers



# 32-bit Look-ahead Adder

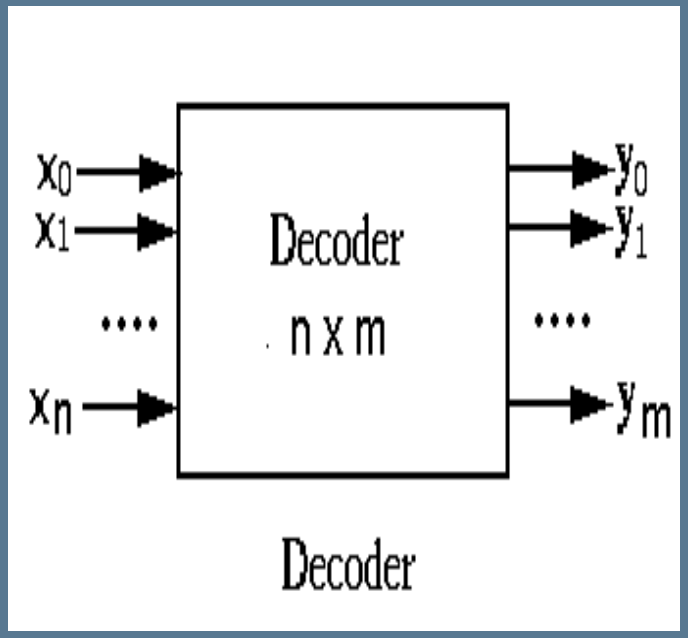


# Compromise on carry look-ahead

- Use ripple carry for a number of bits (e.g., 4)
- Do carry look-ahead for each 4-bit ripple carry adder.
- Result: 10-fold increase in speed of addition depending on number of bits.

# Decoder

- A combinational circuit that converts binary information from n input lines to a maximum of $2^n$ unique output lines.
- They are called n-to-m line decoders where $m \leq 2^n$.

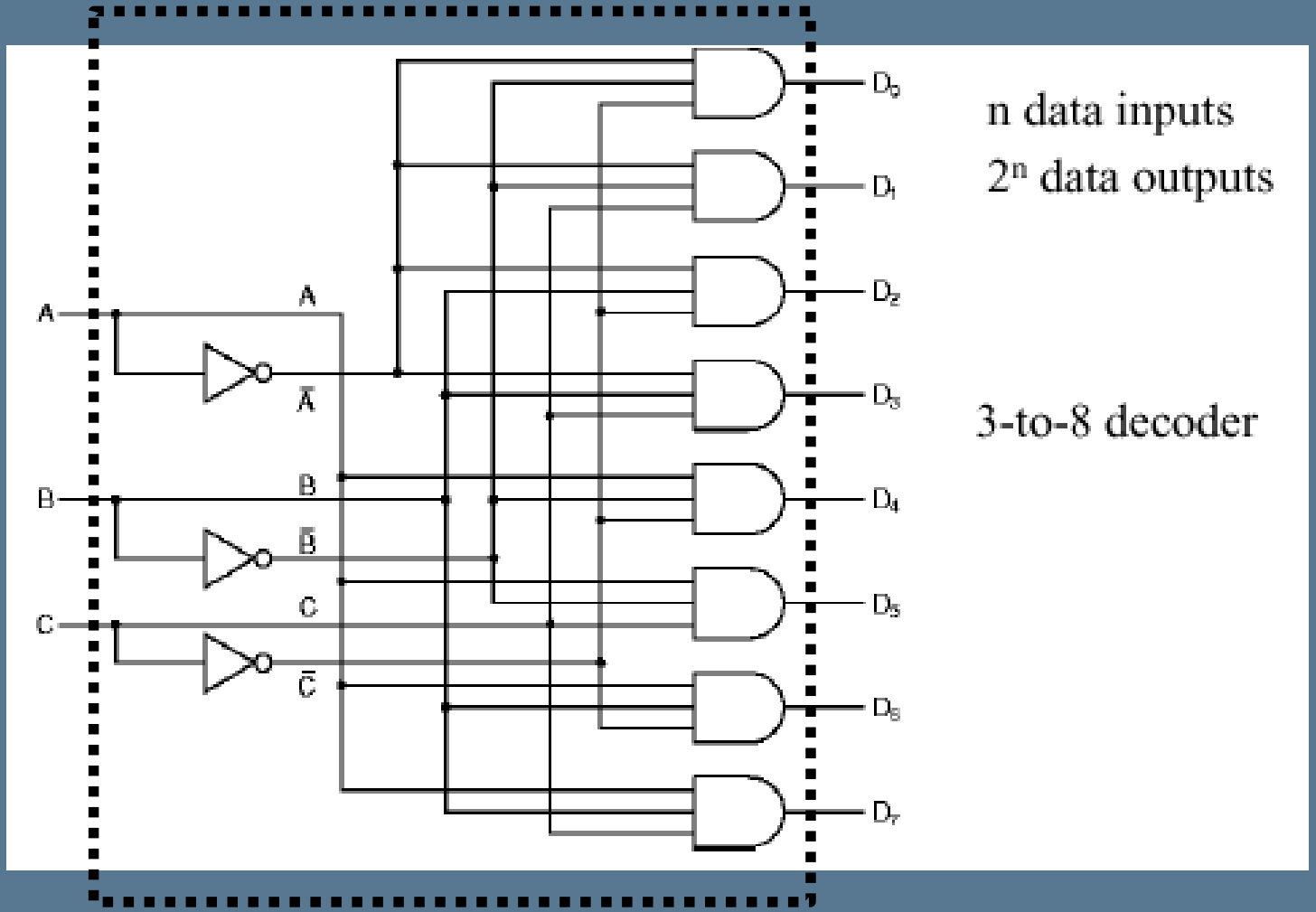


# Truth table for a 3-to-8 line decoder

| A | B | C | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

Only one output is HIGH for each input code ($2^n$ codes)

# Logic circuit of a 3-to-8 line decoder



# Example: Decoder

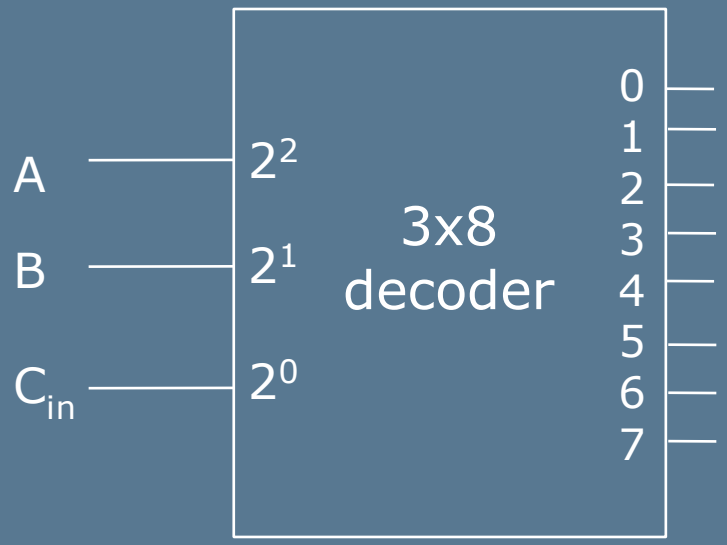
- Implement a full-adder using a decoder and external gates.

  Recall: $S = \sum(1,2,4,7)$

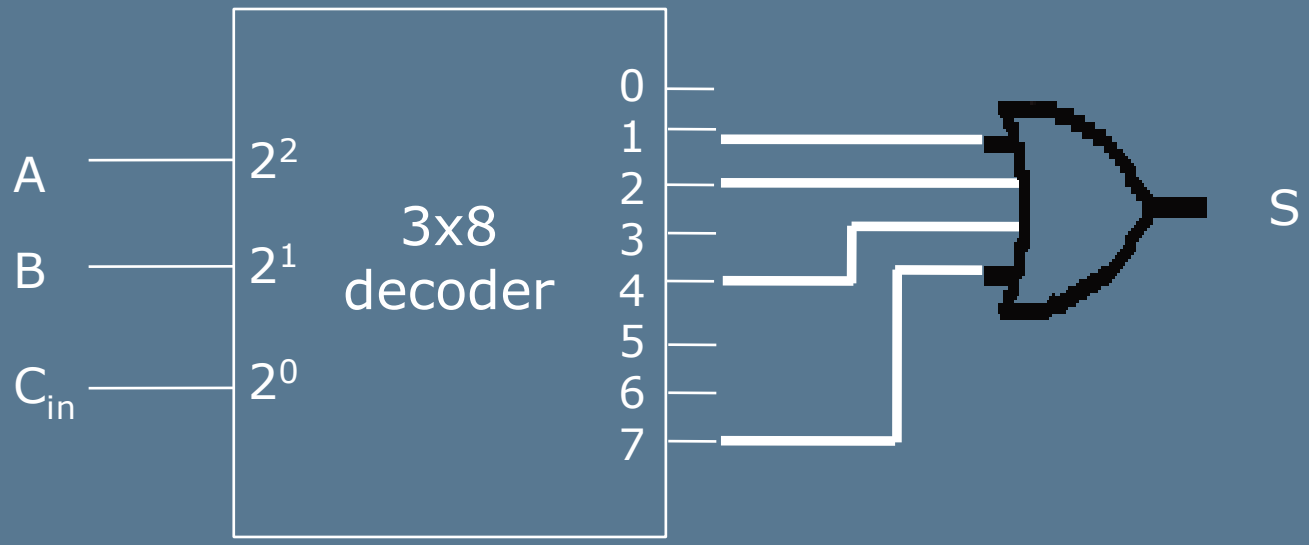
  and $C_{out} = \sum(3,5,6,7)$

- Since there are three inputs, n = 3, use a 3-to-8 line decoder
- Since there are two output functions, two external OR gates are needed.

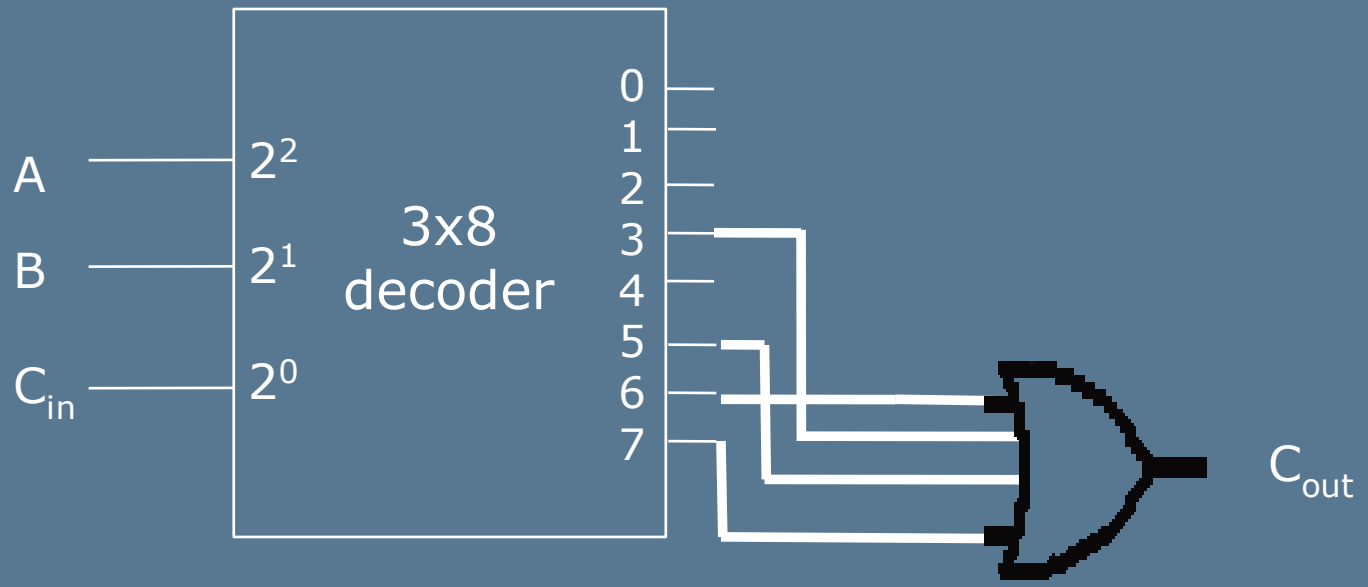
# Example: Decoder



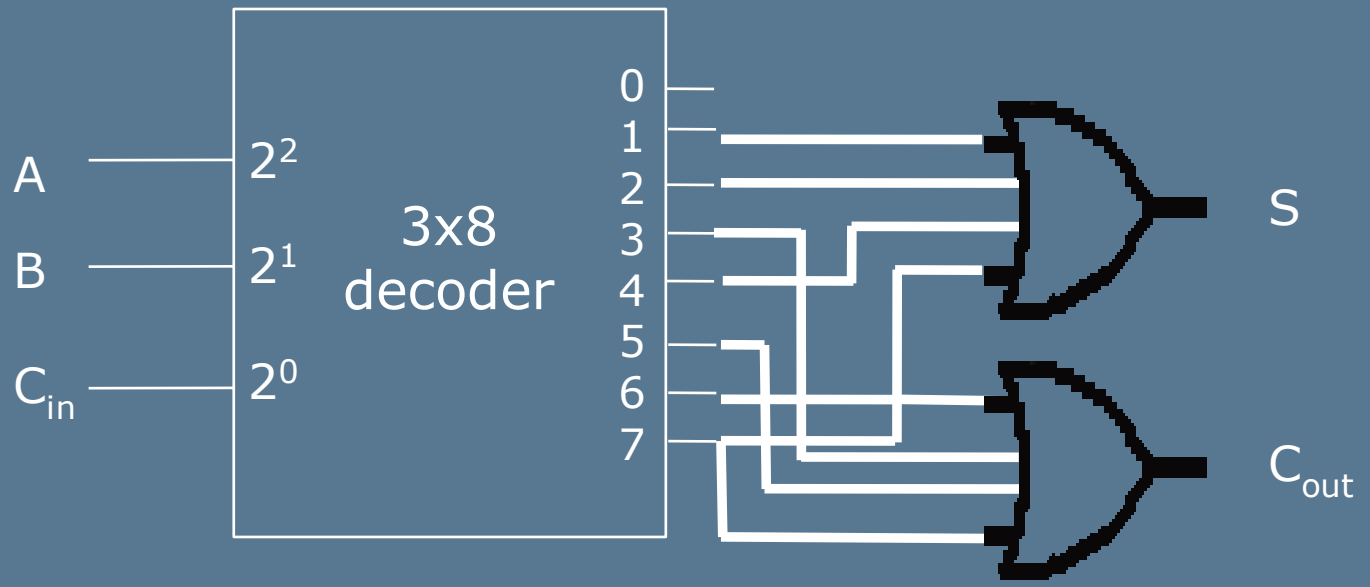
# Example: Decoder



# Example: Decoder



# Example: Decoder

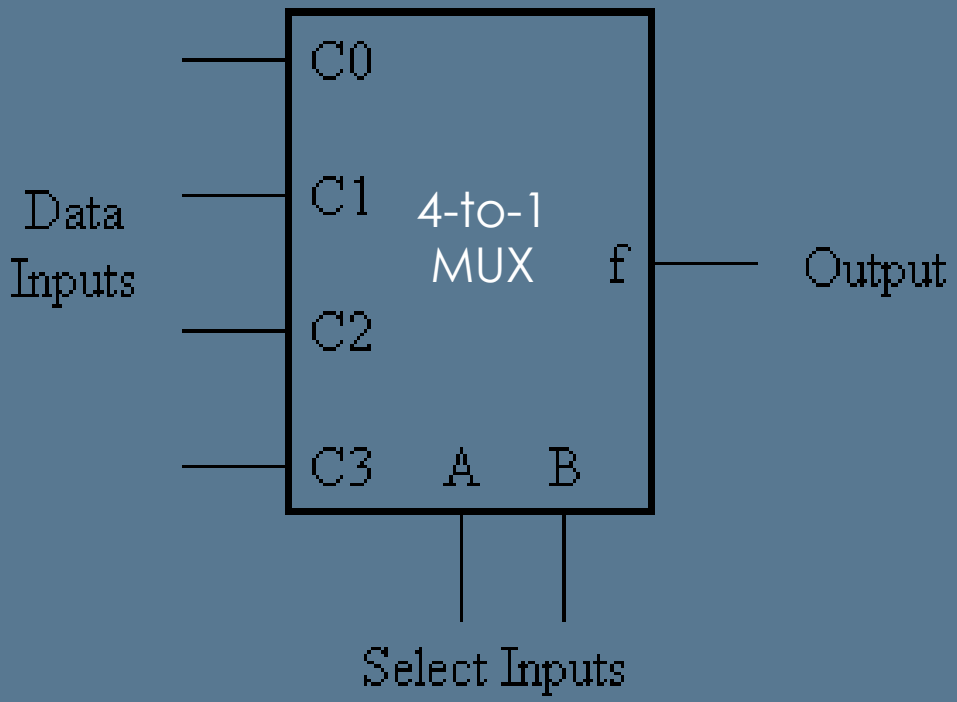



In general, any n inputs with m outputs can be implemented using n-to-$2^n$ line decoders and m OR gates.

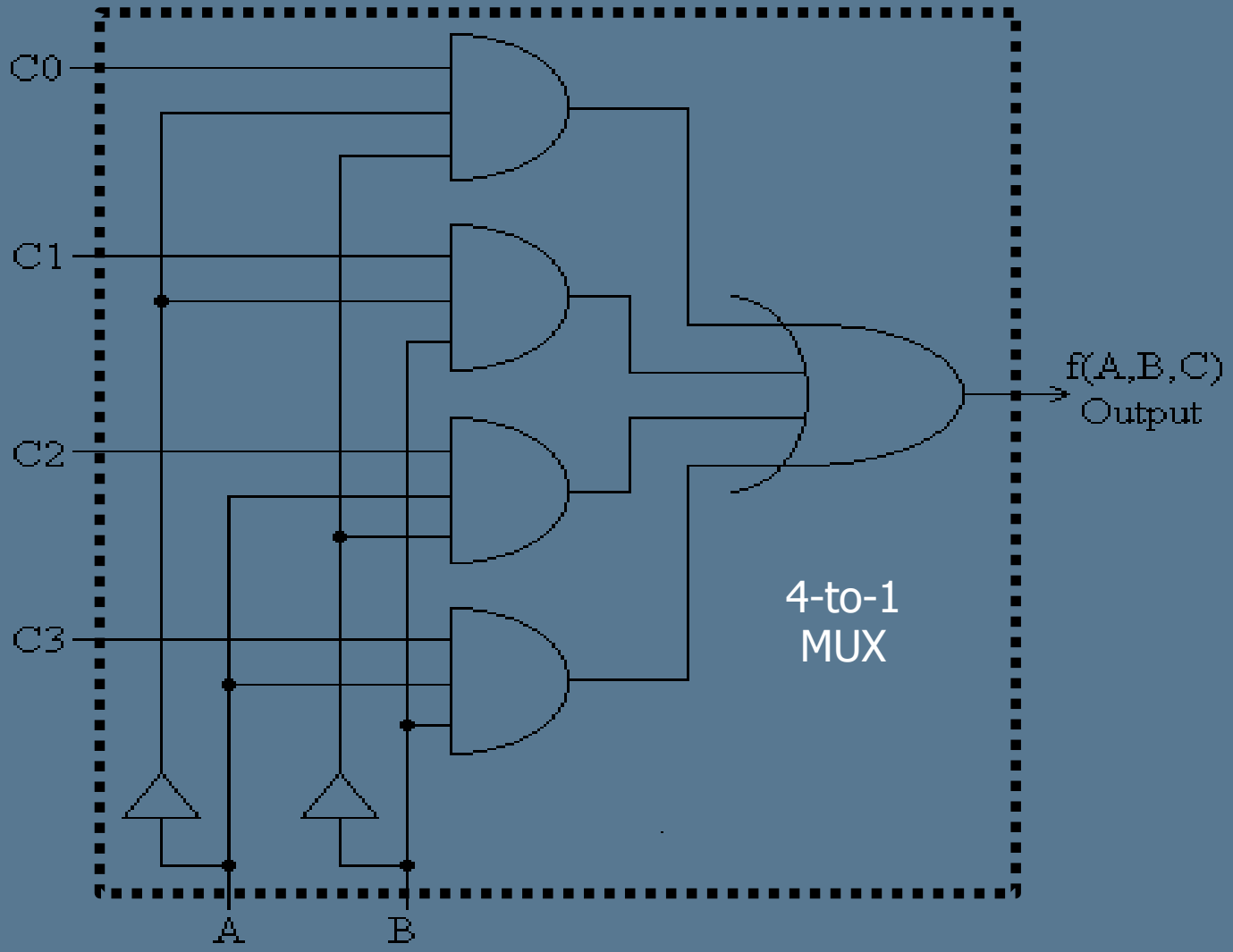
# Multiplexer (Data selectors)

- A device that selects binary information from one of many input lines and directs it to a single output line.
- Control signal pattern forms binary index of input connected to output
- Two forms:
  - Logical form (with n selection lines)
  - Functional form (with n-1 selection lines)

# Example: MUX



# Example: MUX

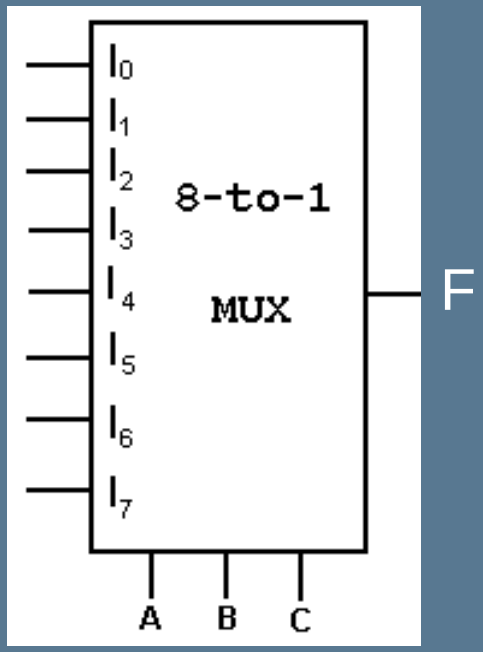


# Example: MUX

- Implement the function $F = \sum(1,3,5,6)$ using a MUX in (a) Logical form and (b) Functional form

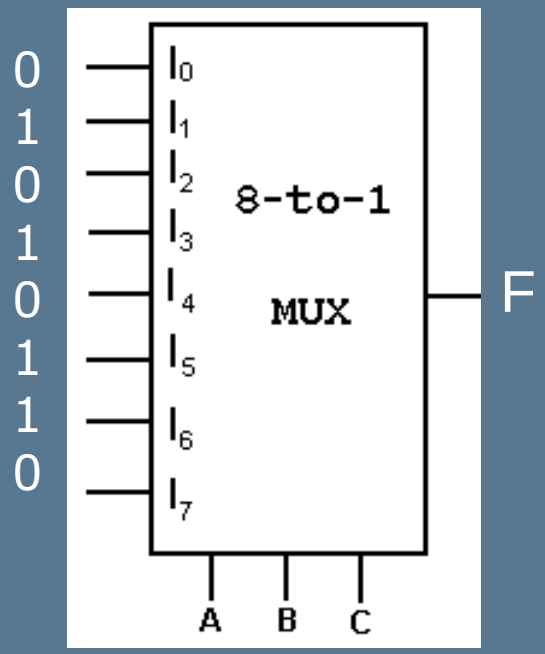
# Example: MUX

(a) F = $\sum(1,3,5,6)$ in Logical form (Use an 8-to-1 MUX)



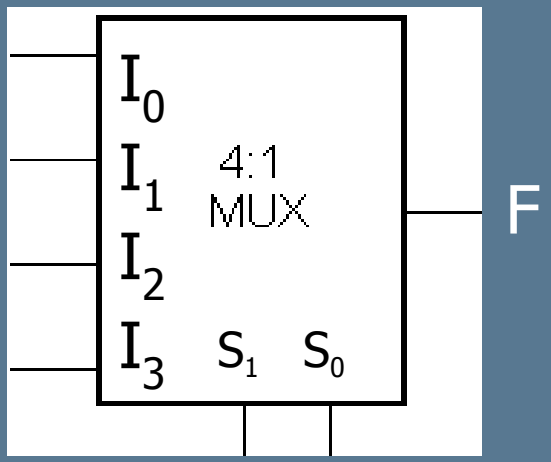
# Example: MUX

(a) F = $\sum(1,3,5,6)$ in Logical form (Use an 8-to-1 MUX)



# Example: MUX

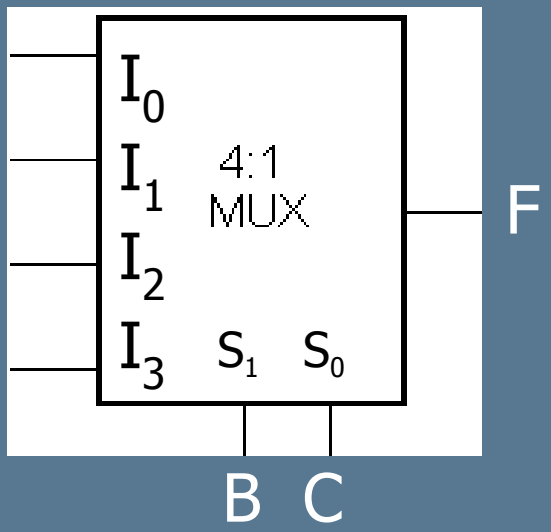
(b) F = ∑(1,3,5,6) in Functional form (Use a 4-to-1 MUX)



# Example: MUX

(b) F = $\sum(1,3,5,6)$ in Functional form (Use a 4-to-1 MUX)

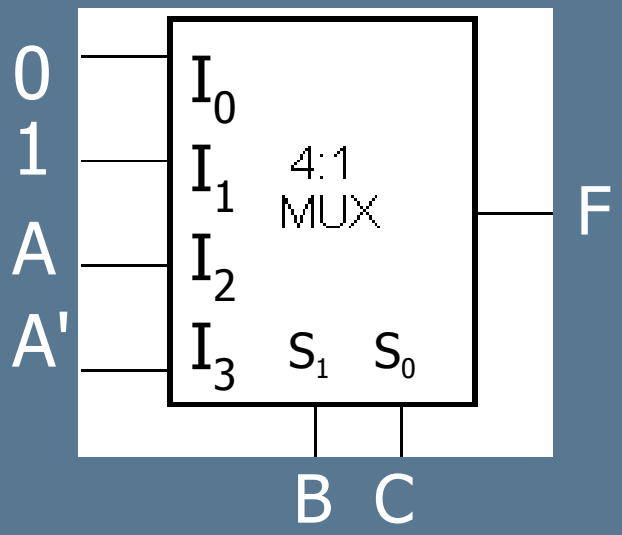
| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| A’ | 0 | 1 | 2 | 3 |
| A | 4 | 5 | 6 | 7 |
| | | | | |



# Example: MUX

(b) F = ∑(1,3,5,6) in Functional form (Use a 4-to-1 MUX)

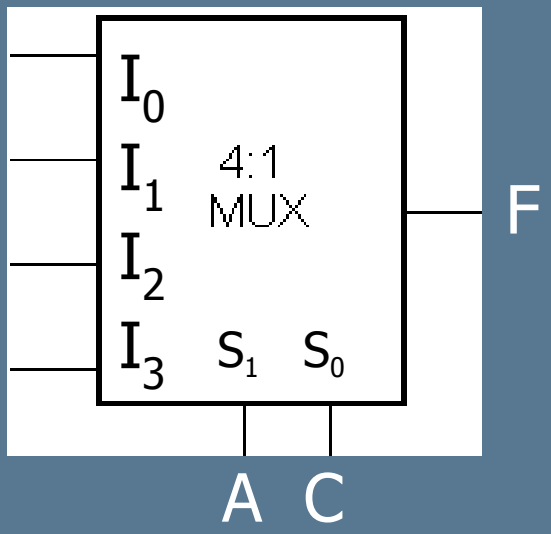
| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| A' | 0 | 1 | 2 | 3 |
| A | 4 | 5 | 6 | 7 |
| | 0 | 1 | A | A' |



# Example: MUX

(b) F = $\sum(1,3,5,6)$ in Functional form (Use a 4-to-1 MUX)

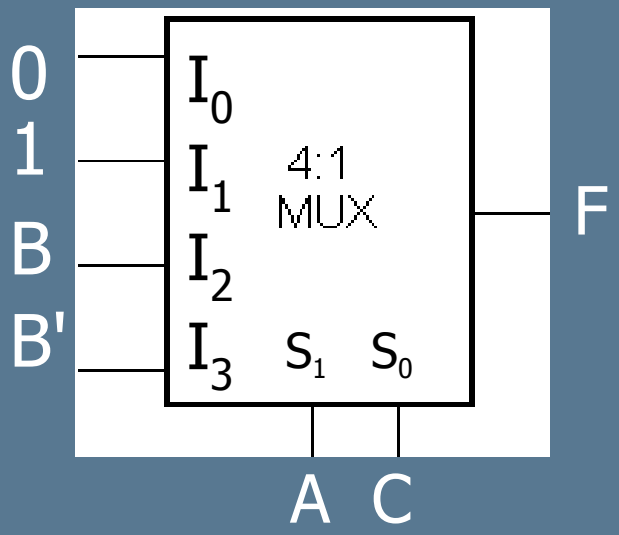
| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| B' | 0 | 1 | 4 | 5 |
| B | 2 | 3 | 6 | 7 |
| | | | | |



# Example: MUX

(b) F = ∑(1,3,5,6) in Functional form (Use a 4-to-1 MUX)

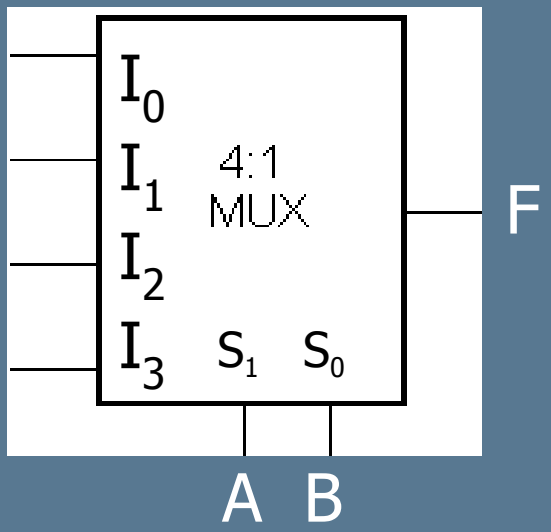
| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| B’ | 0 | 1 | 4 | 5 |
| B | 2 | 3 | 6 | 7 |
| | 0 | 1 | B | B’ |



# Example: MUX

(b) F = ∑(1,3,5,6) in Functional form (Use a 4-to-1 MUX)

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| C' | 0 | 2 | 4 | 6 |
| C | 1 | 3 | 5 | 7 |
| | | | | |



# Example: MUX

(b) F = ∑(1,3,5,6) in Functional form (Use a 4-to-1 MUX)

| | $I_0$ | $I_1$ | $I_2$ | $I_3$ |
|---|---|---|---|---|
| C' | 0 | 2 | 4 | 6 |
| C | 1 | 3 | 5 | 7 |
| | C | C | C | C' |