Bahria University
Discovering Knowledge

Computer Architecture and Logic Design (CALD)

Lecture 12

Dr. Sorath Hansrajani

Assistant Professor

Department of Software Engineering

Bahria University Karachi Campus

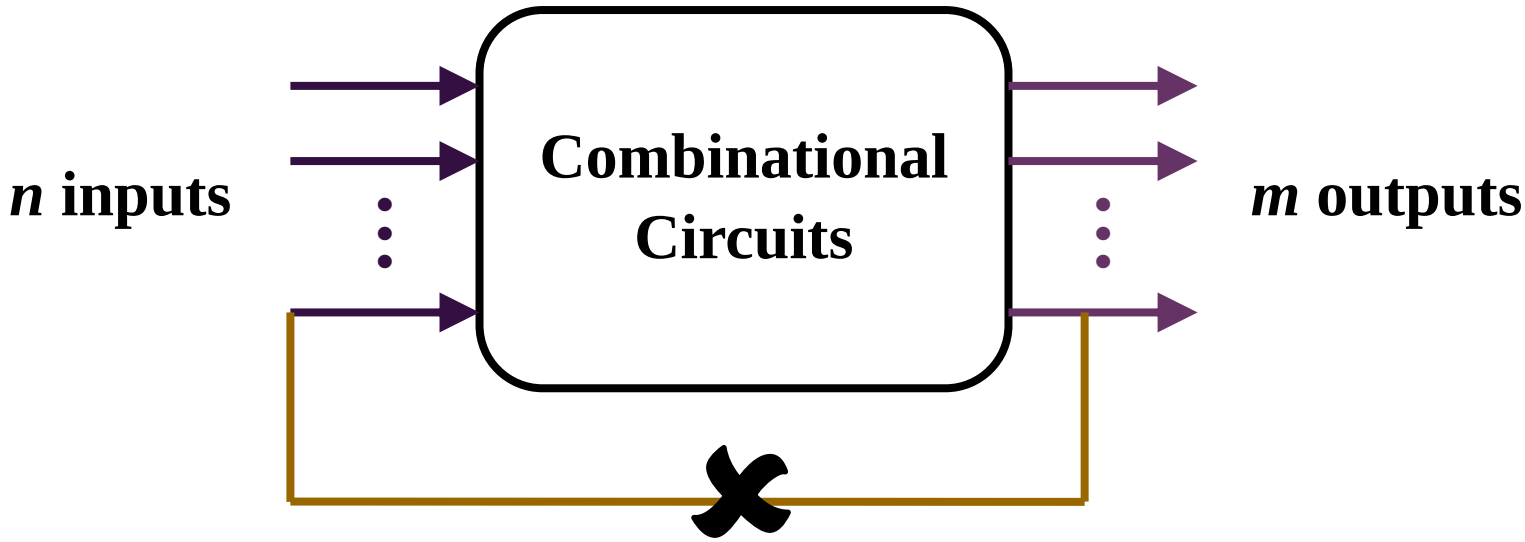
Email: sorathhansrajani.bukc@bahria.edu.pk

Combinational Logic

Combinational Circuits

- Output is function of input only

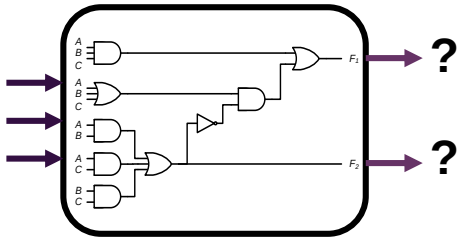
  i.e. no feedback



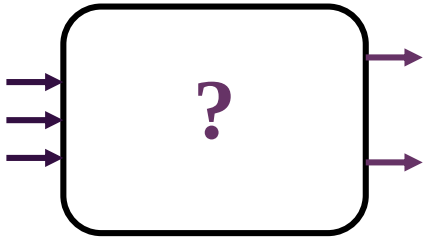
When input changes, output may change (after a delay)

Combinational Circuits

- Analysis
  - Given a circuit, find out its *function*
  - Function may be expressed as:
    - Boolean function
    - Truth table

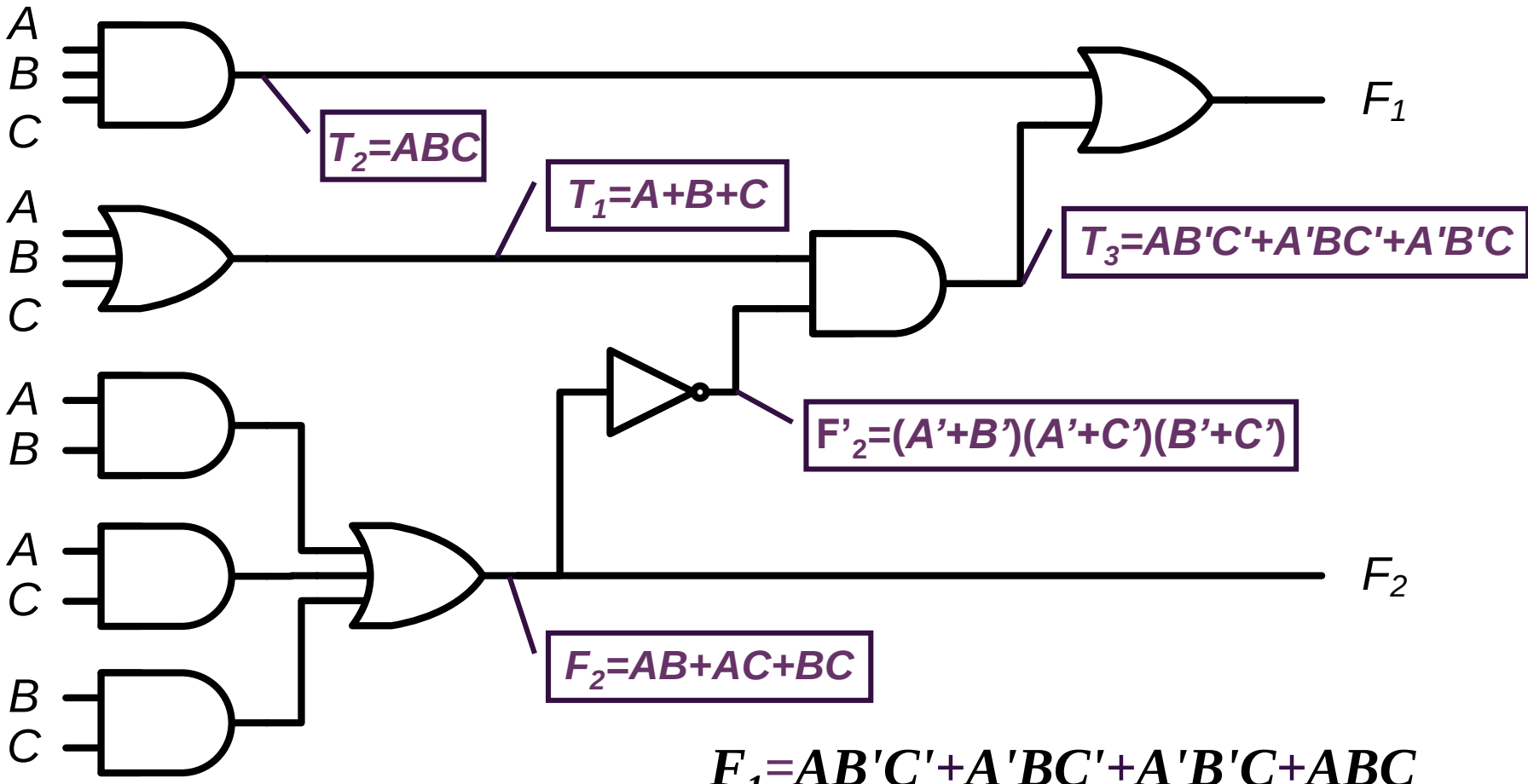


- Design
  - Given a desired function, determine its *circuit*
  - Function may be expressed as:
    - Boolean function
    - Truth table



Analysis Procedure

- Boolean Expression Approach

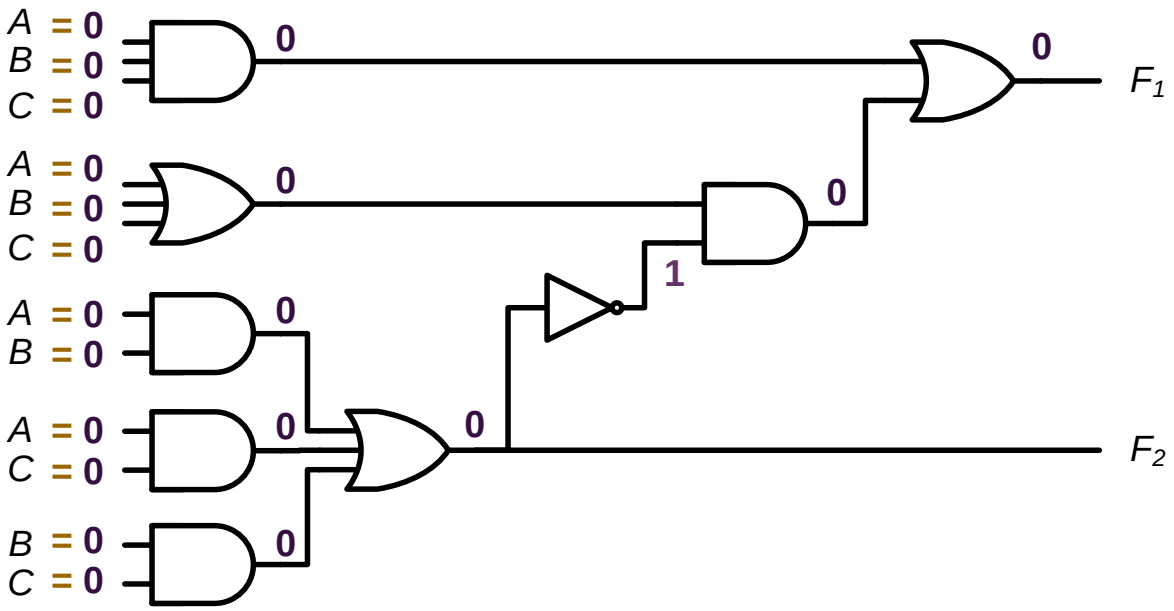


$$F_1 = AB'C' + A'BC' + A'B'C + ABC$$

$$F_2 = AB + AC + BC$$

Analysis Procedure

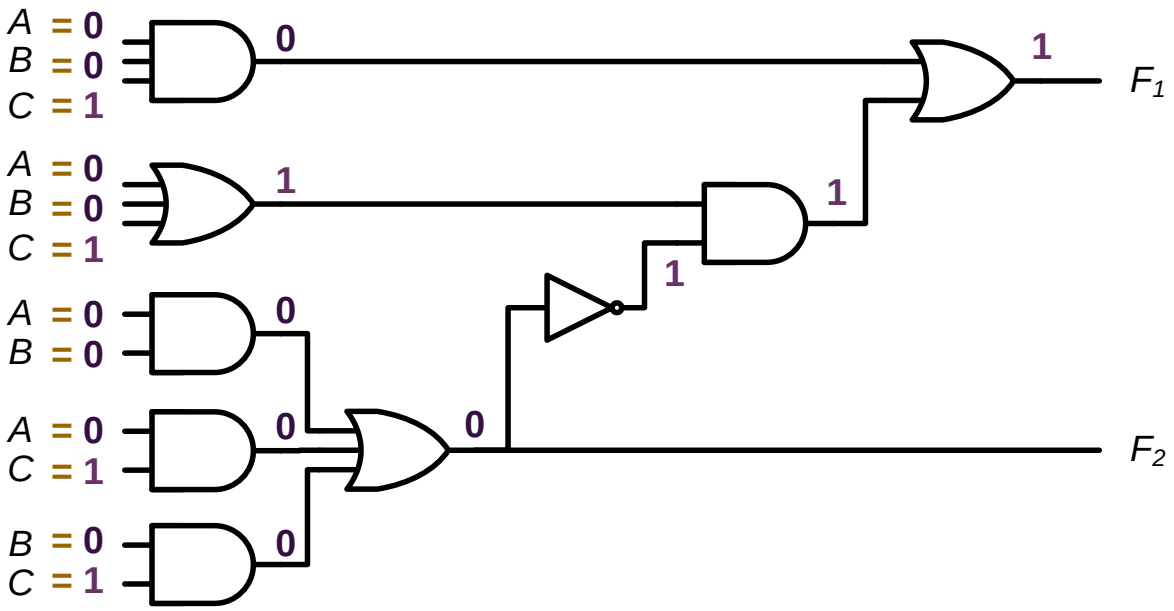
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0

Analysis Procedure

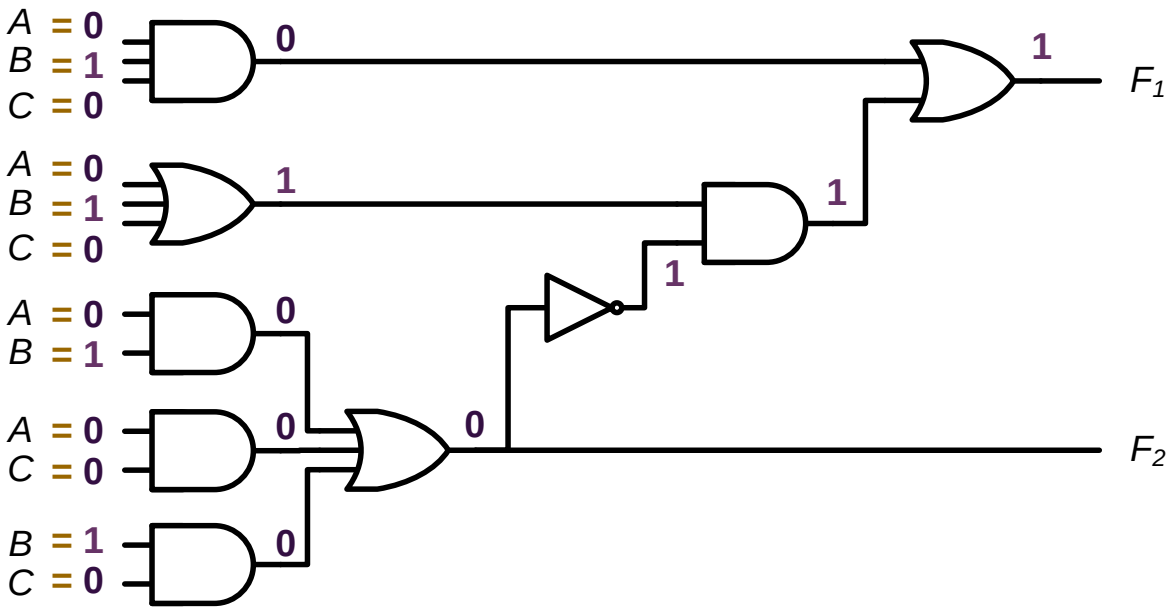
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0

Analysis Procedure

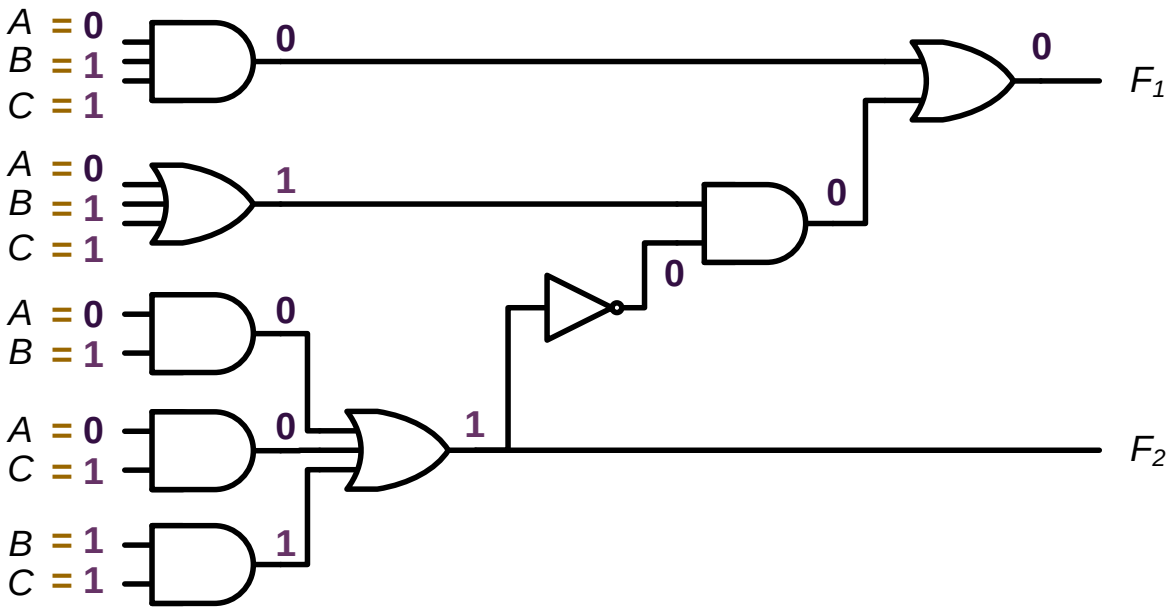
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0

Analysis Procedure

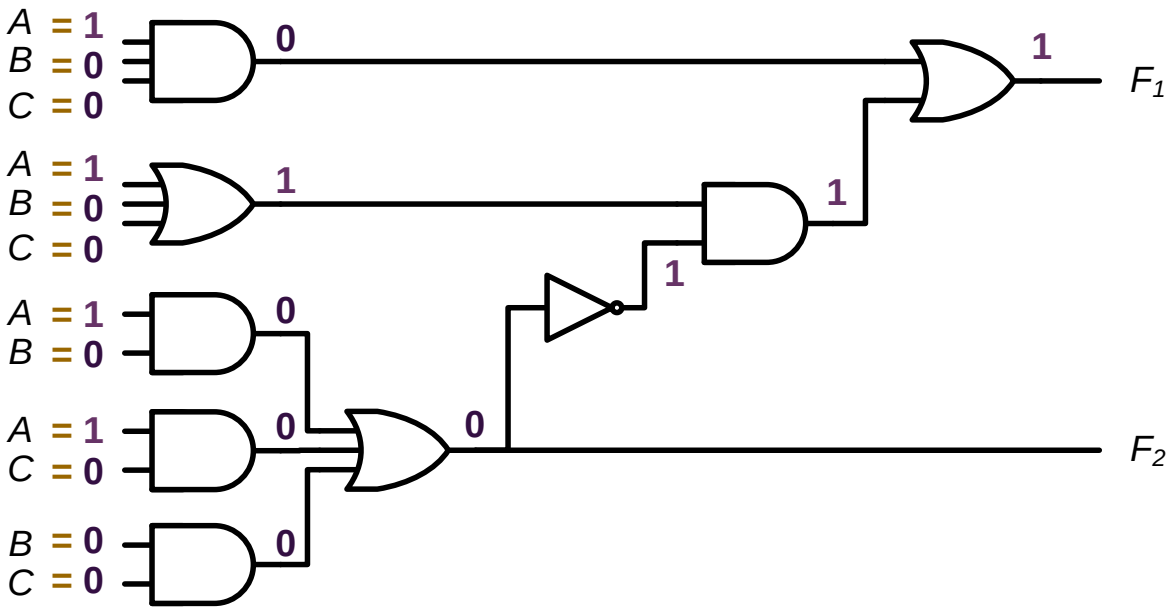
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0
0 1 0	1	0
0 1 1	0	1

Analysis Procedure

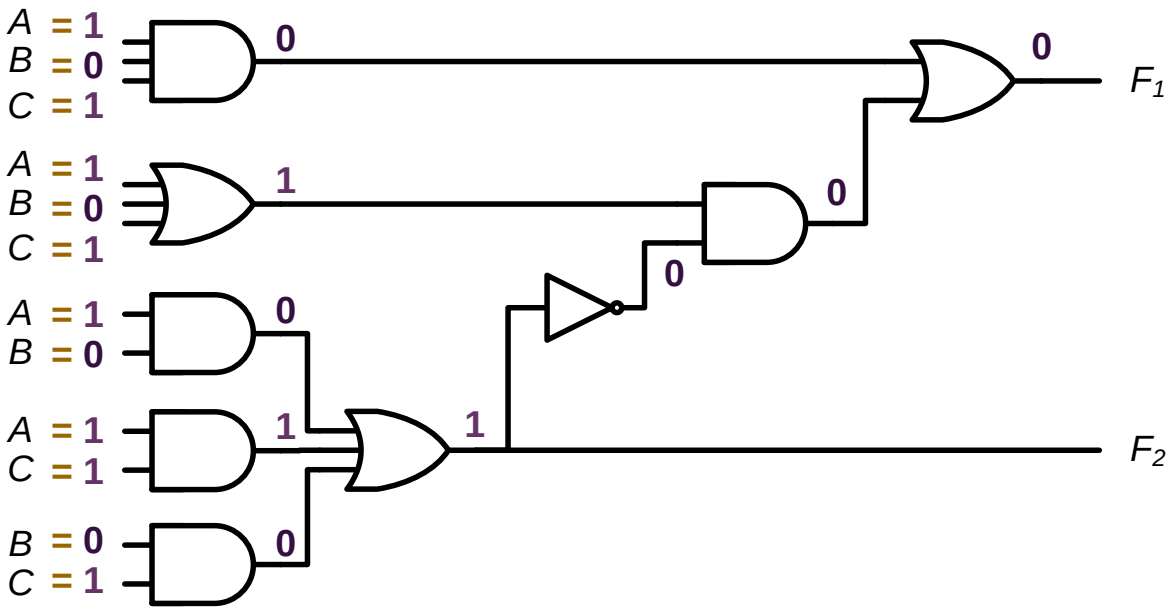
- Truth Table Approach



A	B	C	F_1	F_2
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

Analysis Procedure

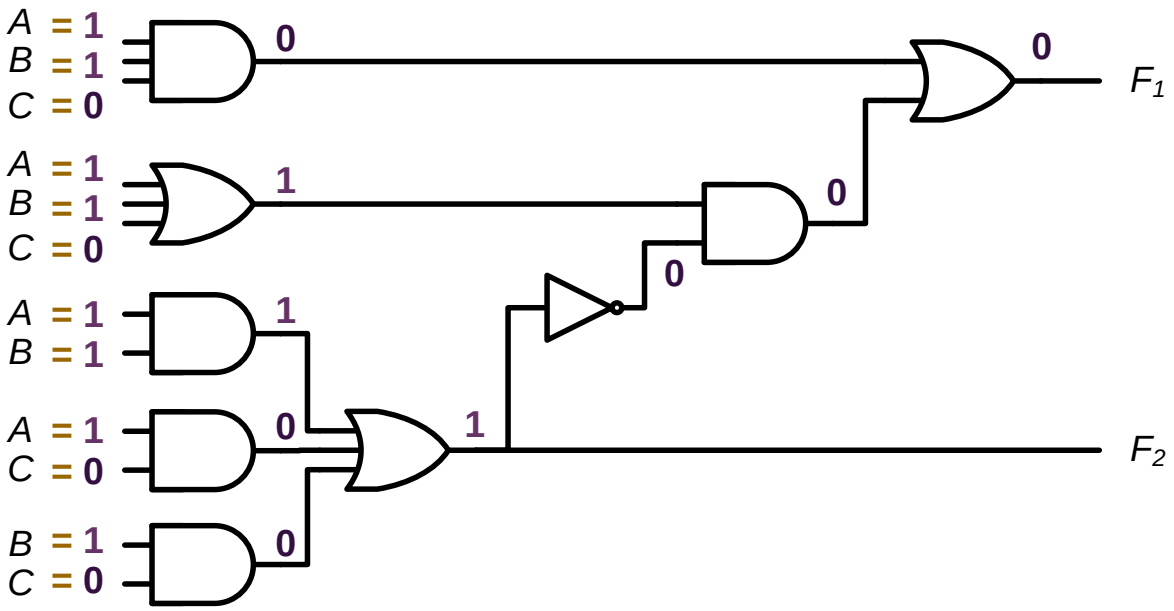
- Truth Table Approach



A	B	C	F_1	F_2
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

Analysis Procedure

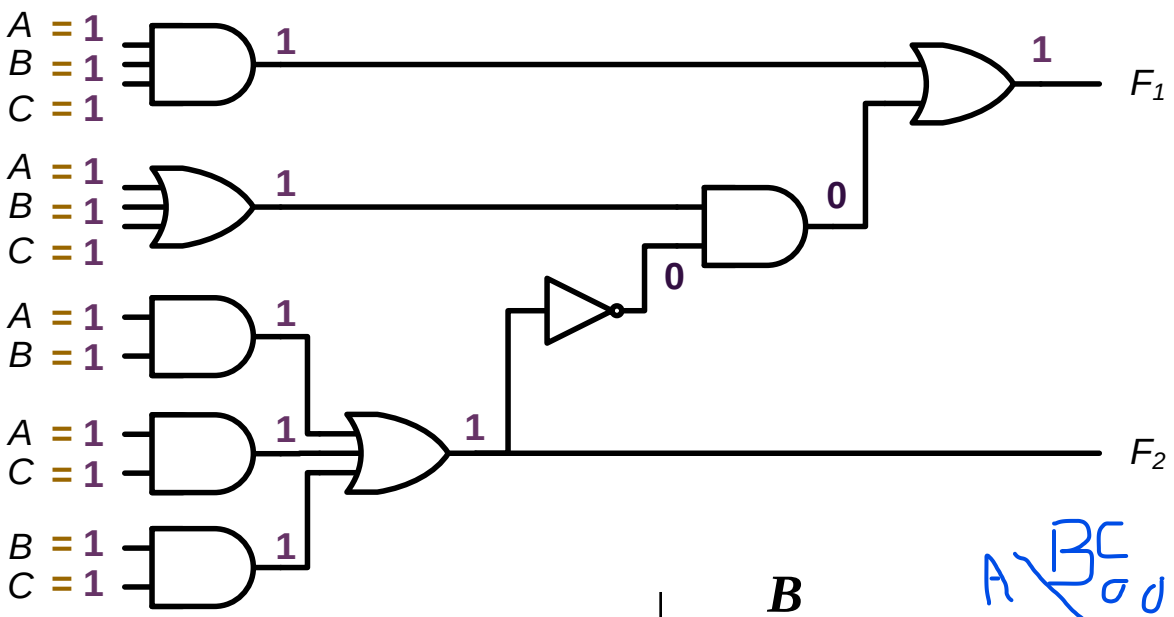
- Truth Table Approach



A	B	C	F_1	F_2
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

Analysis Procedure

- Truth Table Approach



A	B	C	F_1	F_2
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

A \ BC	00	01	11	10
0	0	0	1	0
1	1	0	1	0

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

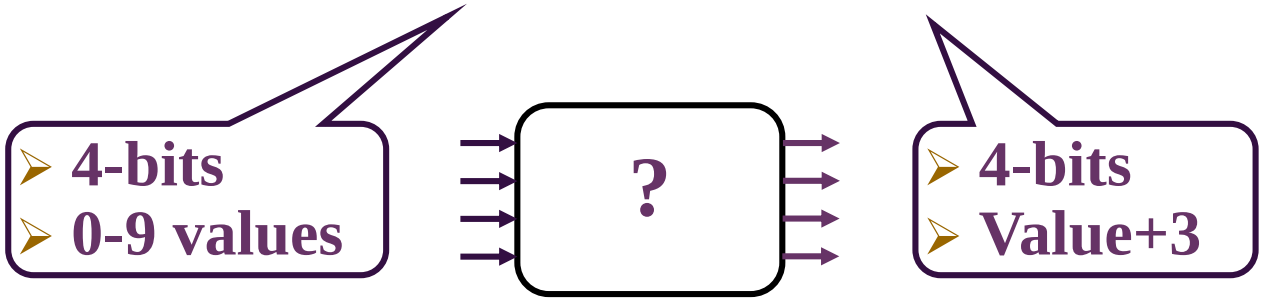
$$F_1 = AB'C' + A'BC' + A'B'C + ABC \qquad F_2 = AB + AC + BC$$

Design Procedure

- Given a problem statement:
  - Determine the number of *inputs* and *outputs*
  - Derive the truth table
  - Simplify the Boolean expression for each output
  - Produce the required circuit

Example:

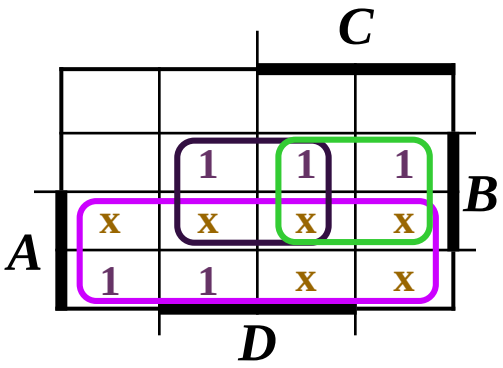
Design a circuit to convert a "BCD" code to "Excess 3" code



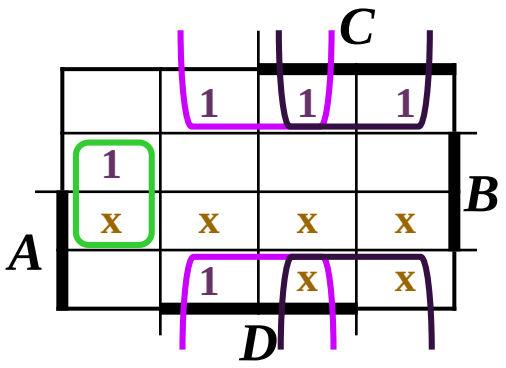
Design Procedure

- BCD-to-Excess 3 Converter

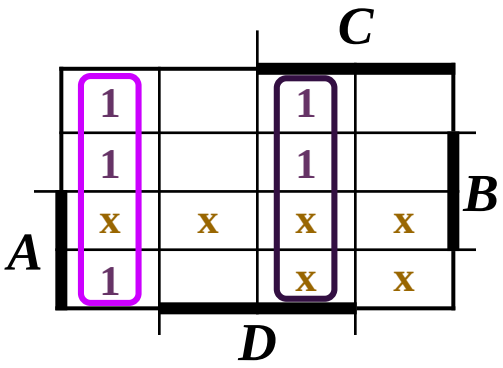
A B C D	w x y z
0 0 0 0	0 0 1 1
0 0 0 1	0 1 0 0
0 0 1 0	0 1 0 1
0 0 1 1	0 1 1 0
0 1 0 0	0 1 1 1
0 1 0 1	1 0 0 0
0 1 1 0	1 0 0 1
0 1 1 1	1 0 1 0
1 0 0 0	1 0 1 1
1 0 0 1	1 1 0 0
1 0 1 0	x x x x
1 0 1 1	x x x x
1 1 0 0	x x x x
1 1 0 1	x x x x
1 1 1 0	x x x x
1 1 1 1	x x x x



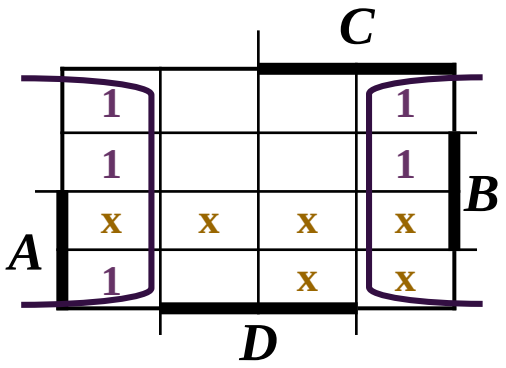
$w = A+BC+BD$



$x = B'C+B'D+BC'D'$



$y = C'D'+CD$

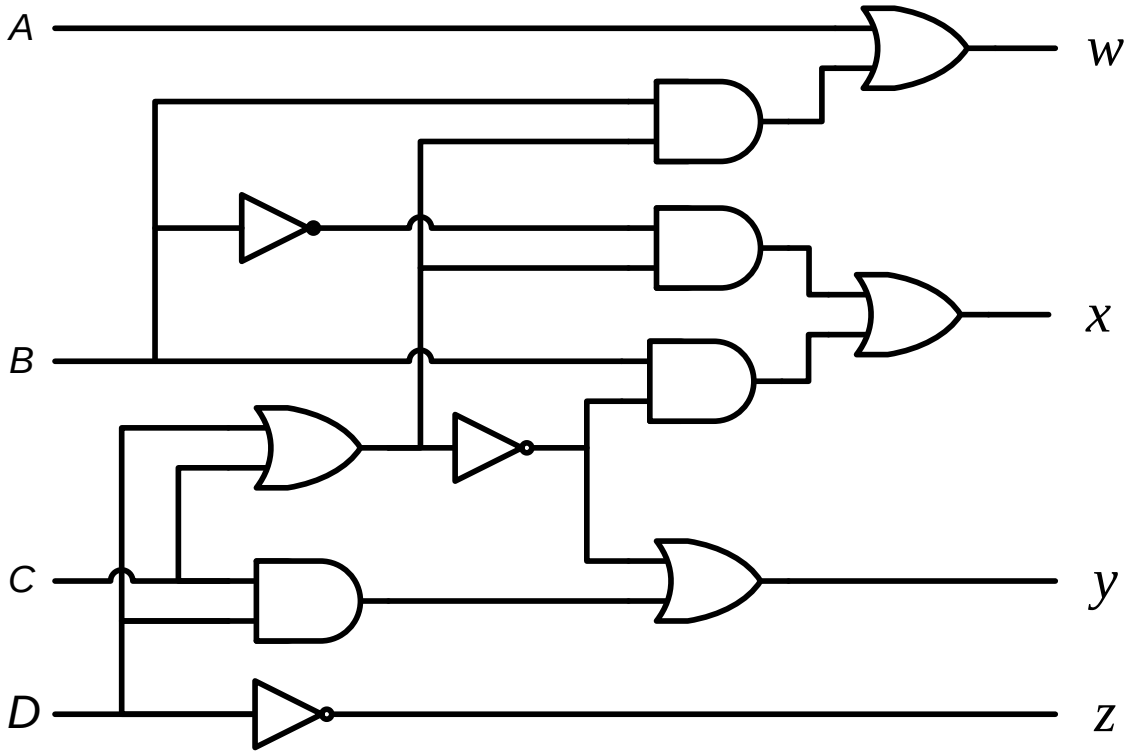


$z = D'$

Design Procedure

- BCD-to-Excess 3 Converter

A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

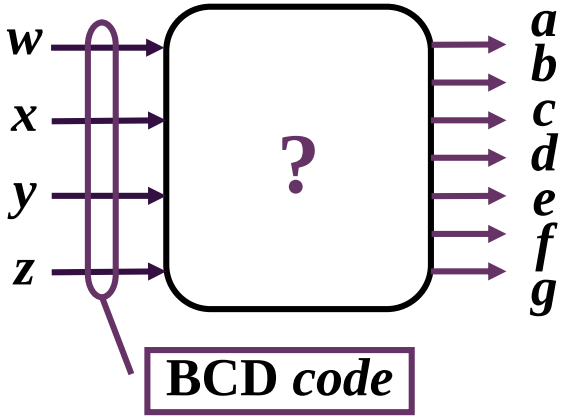


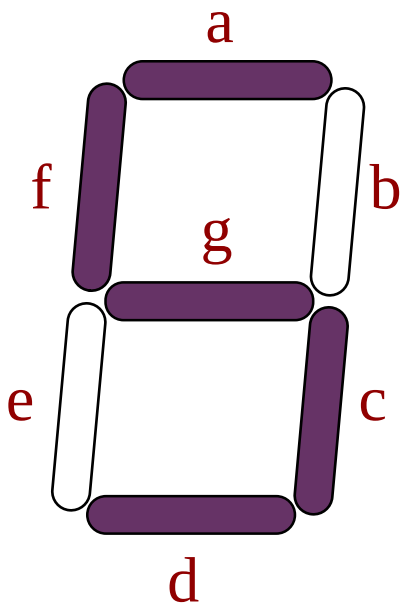
$w = A + B(C+D)$ $\quad y = (C+D)' + CD$

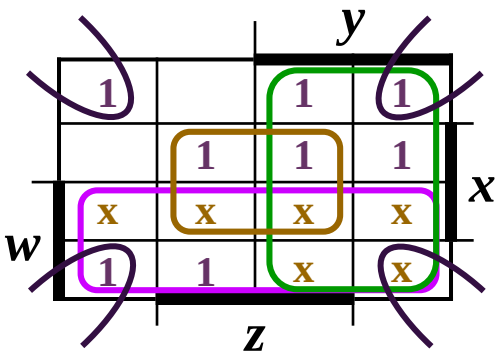
$x = B'(C+D) + B(C+D)'$ $\quad z = D'$

Seven-Segment Decoder

w x y z	a b c d e f g
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 1 0 1 1
1 0 1 0	x x x x x x x
1 0 1 1	x x x x x x x
1 1 0 0	x x x x x x x
1 1 0 1	x x x x x x x
1 1 1 0	x x x x x x x
1 1 1 1	x x x x x x x







$a = w + y + xz + x'z'$

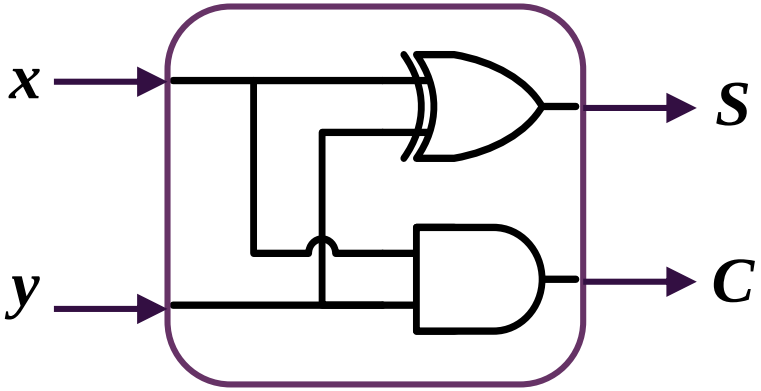
$b = \ldots$
$c = \ldots$
$d = \ldots$

Binary Adder

- Half Adder
  - Adds 1-bit plus 1-bit
  - Produces Sum and Carry

$$\begin{array}{rr} & x \\ + & y \\ \hline C & S \end{array}$$

x y	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

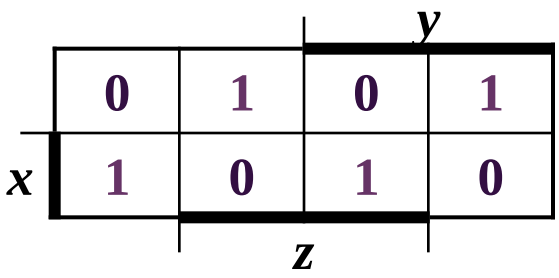


Binary Adder

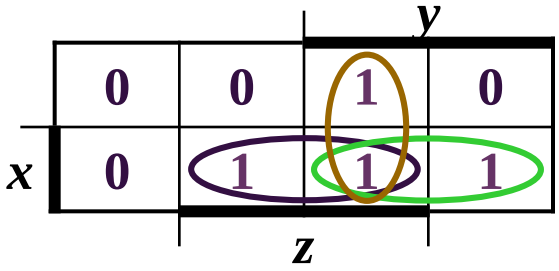
- Full Adder
  - Adds 1-bit plus 1-bit plus 1-bit
  - Produces Sum and Carry

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

$$\begin{array}{rr} & x \\ + & y \\ + & z \\ \hline C & S \end{array}$$



$$S = xy'z' + x'yz' + x'y'z + xyz = x \oplus y \oplus z$$



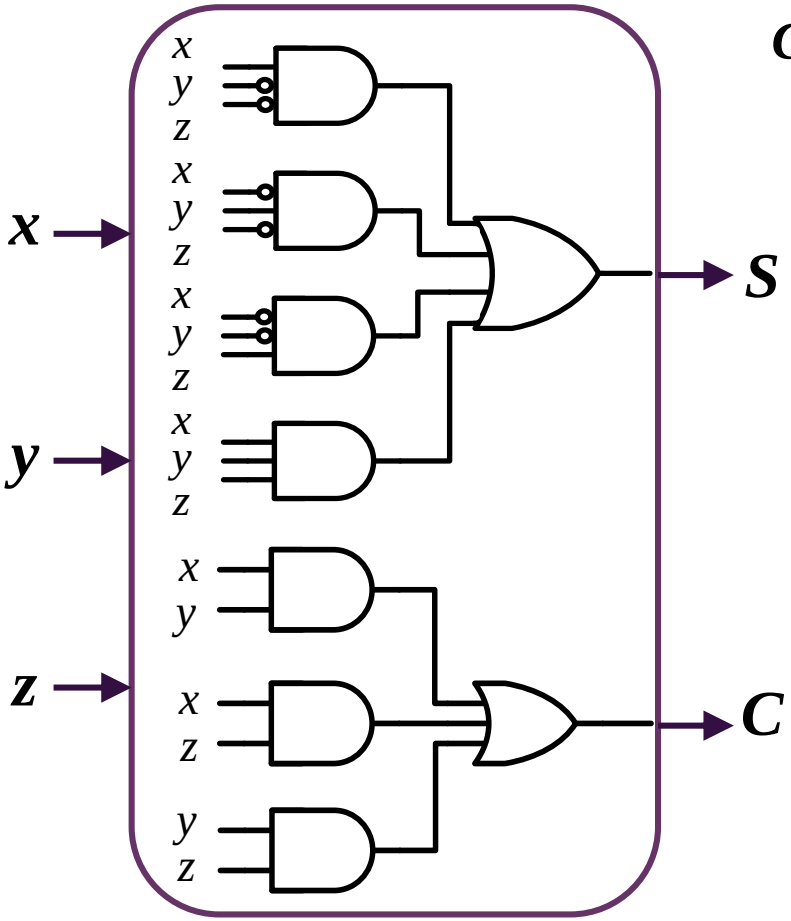
$$C = xy + xz + yz$$

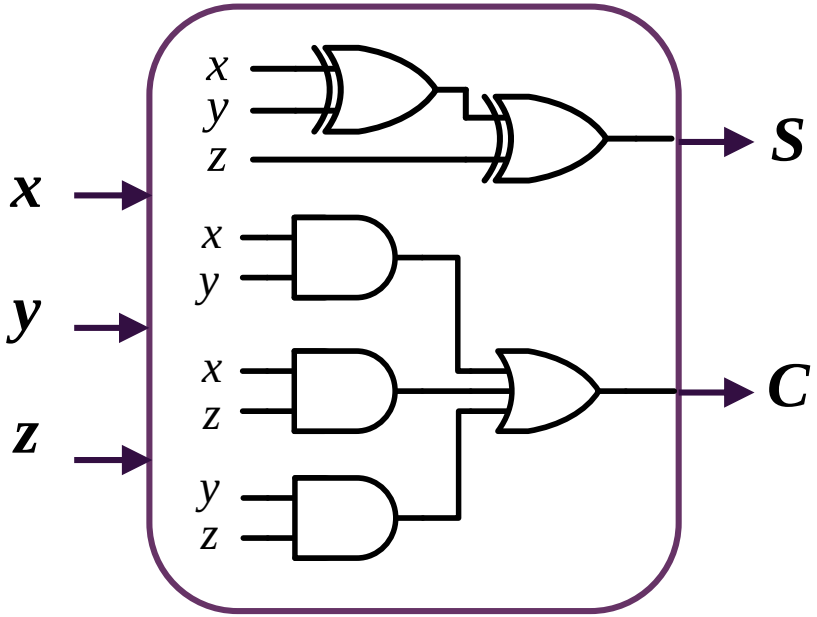
Binary Adder

- Full Adder

$$S = xy'z'+x'yz'+x'y'z+xyz = x \oplus y \oplus z$$

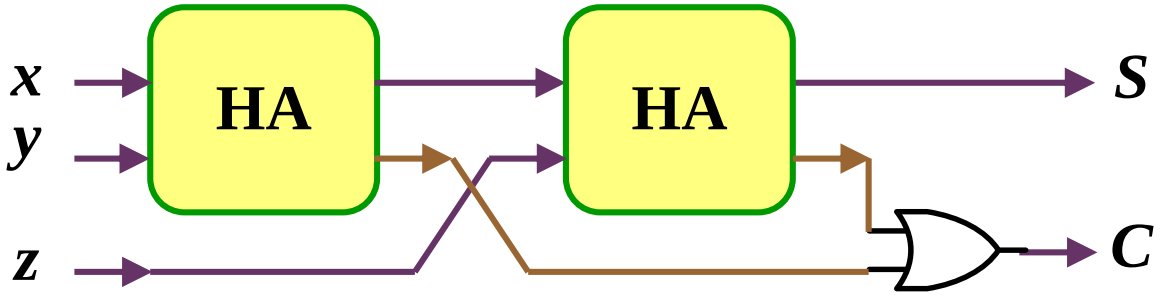
$$C = xy + xz + yz$$

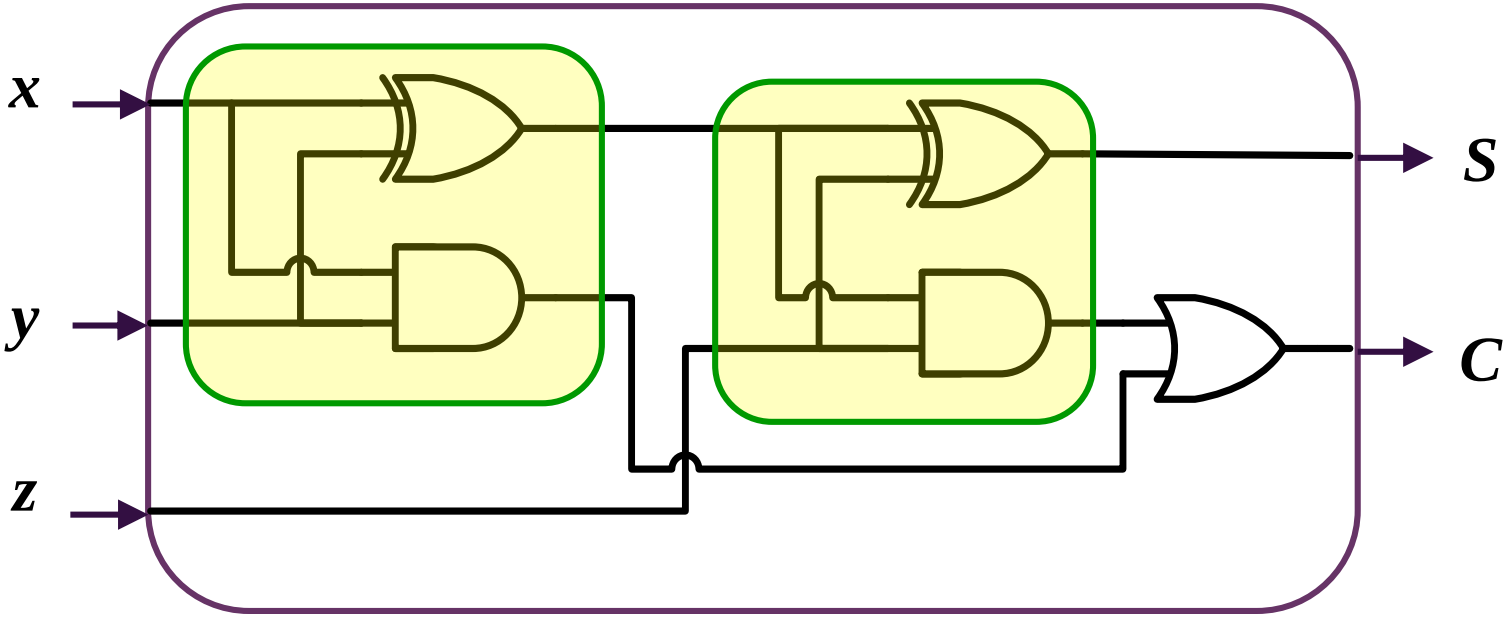




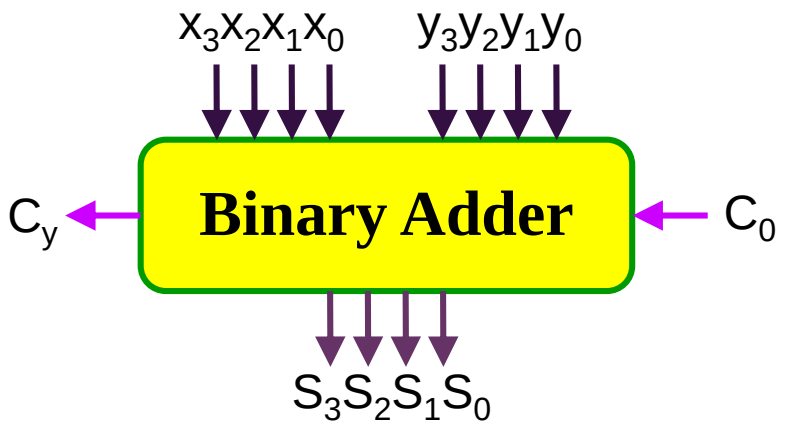
Binary Adder

- Full Adder



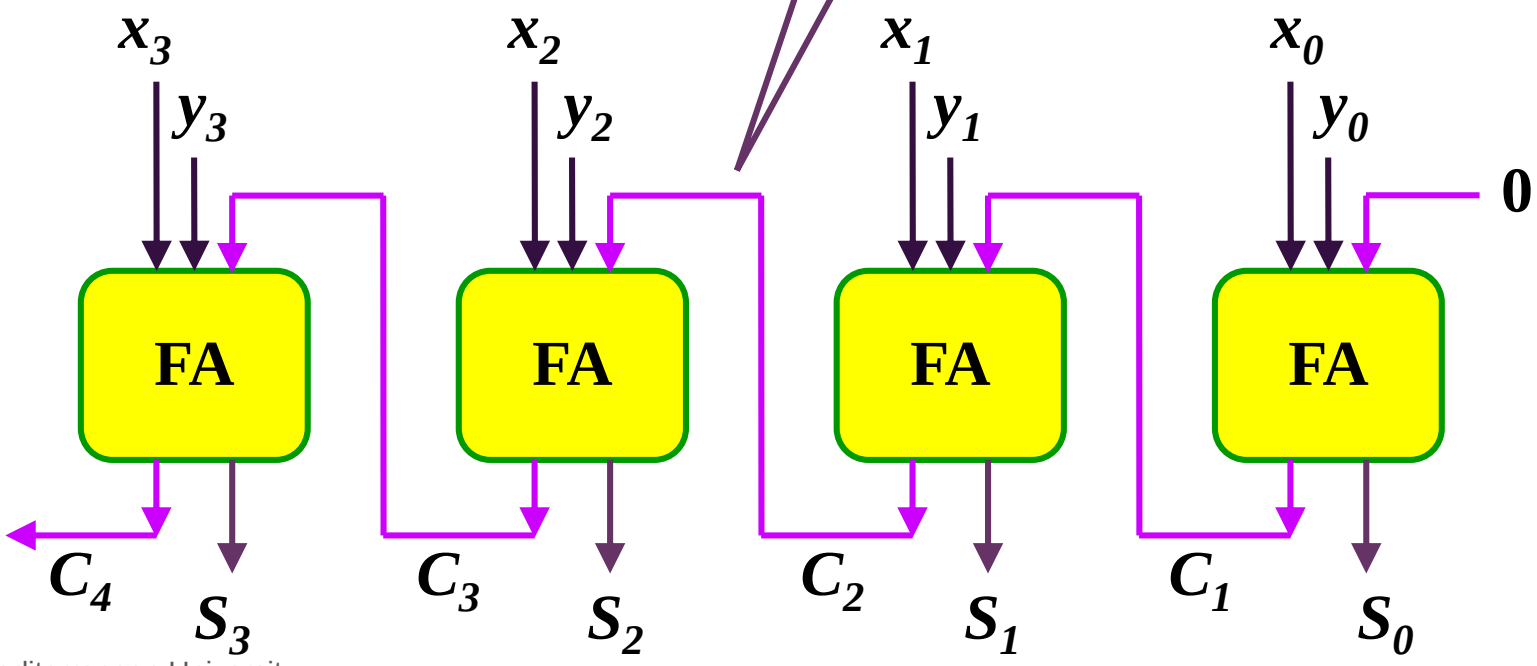


Binary Adder



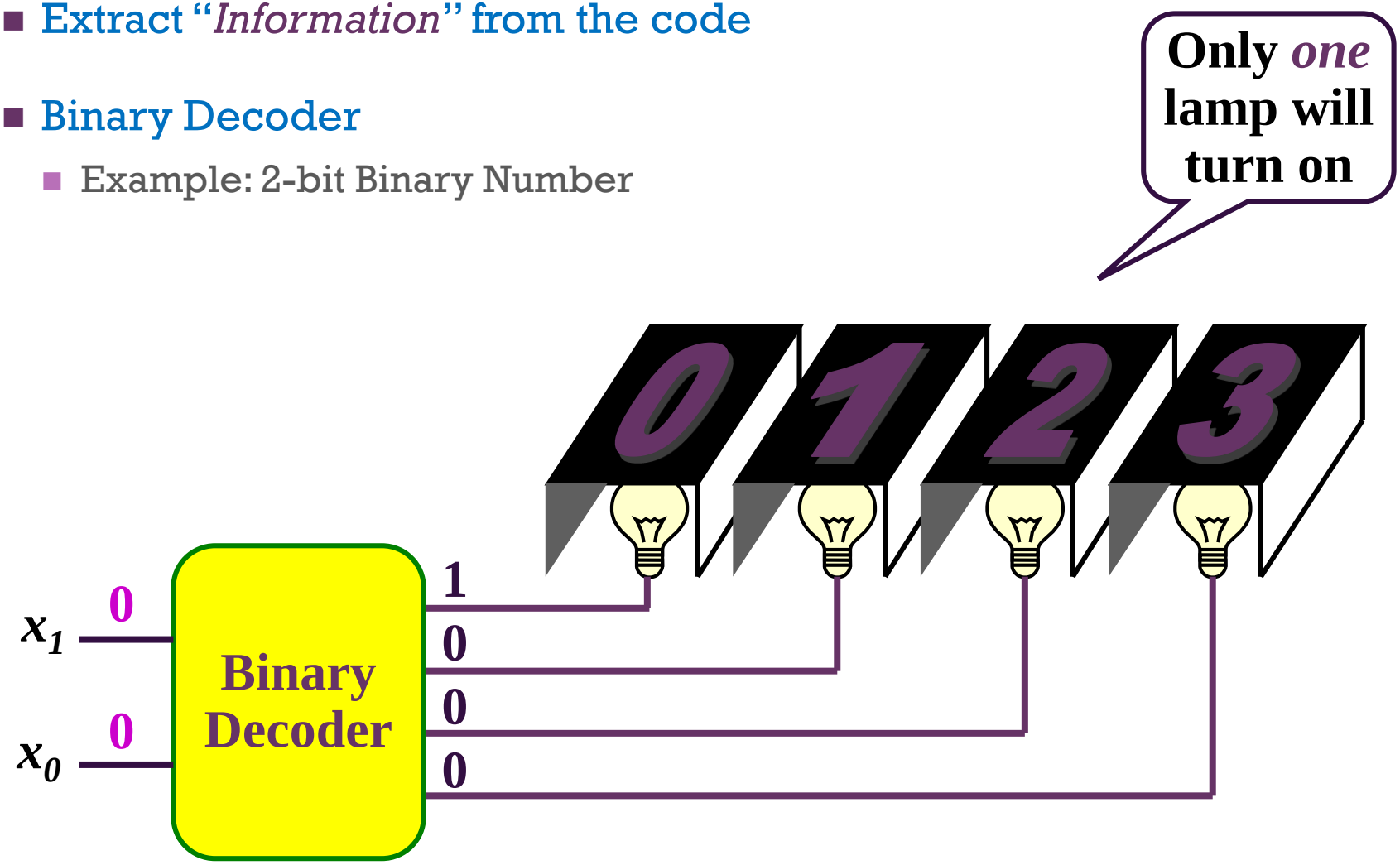
Carry Propagate Addition

$$\begin{array}{llllll} & c_3 & c_2 & c_1 & & \\ + & x_3 & x_2 & x_1 & x_0 & \\ + & y_3 & y_2 & y_1 & y_0 & \\ \hline Cy & S_3 & S_2 & S_1 & S_0 & \end{array}$$



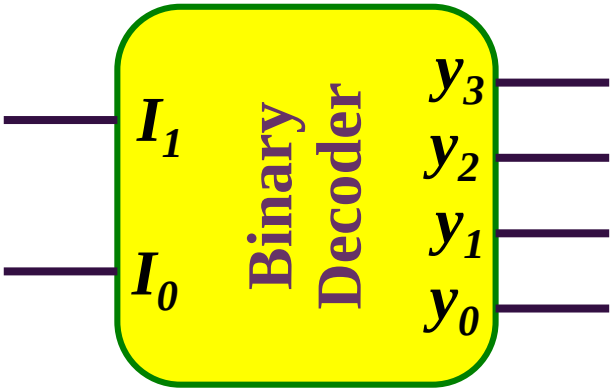
Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

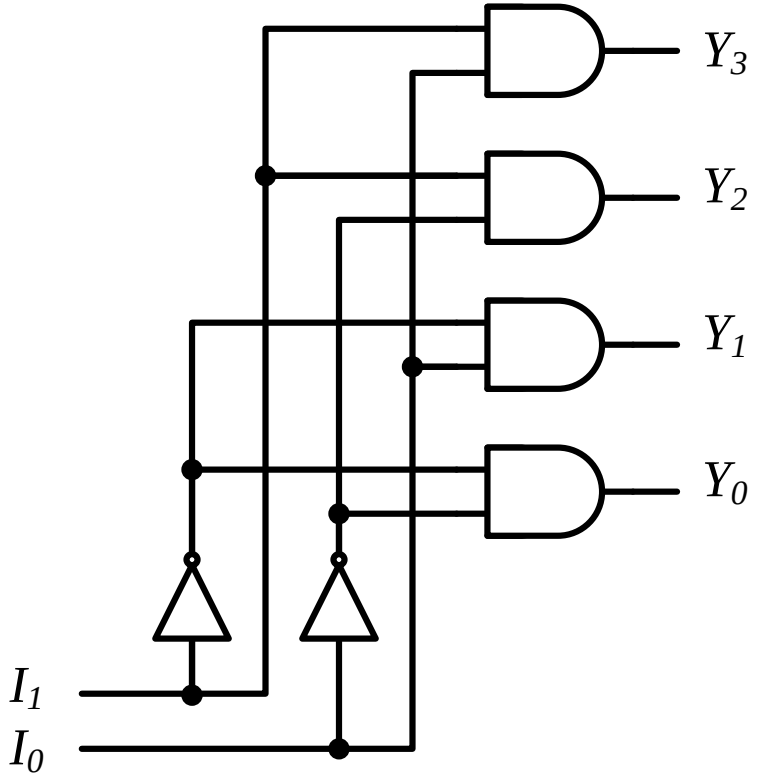


Decoders

- 2-to-4 Line Decoder



I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

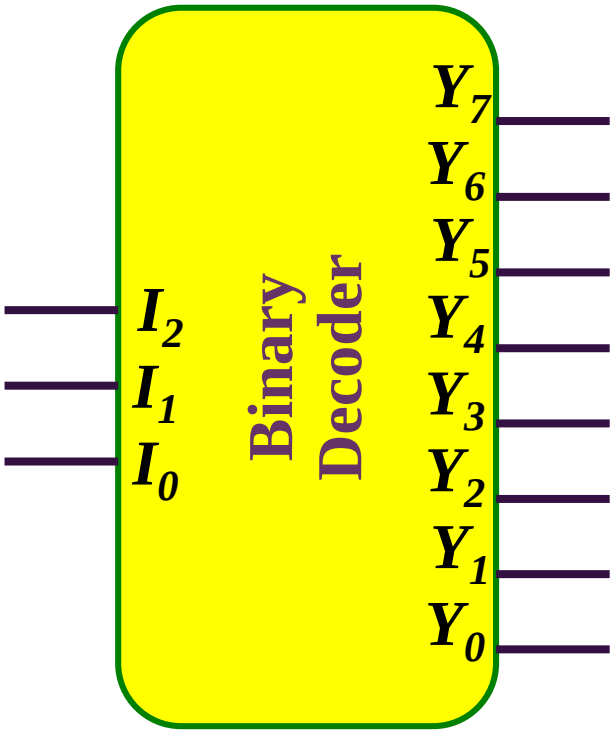


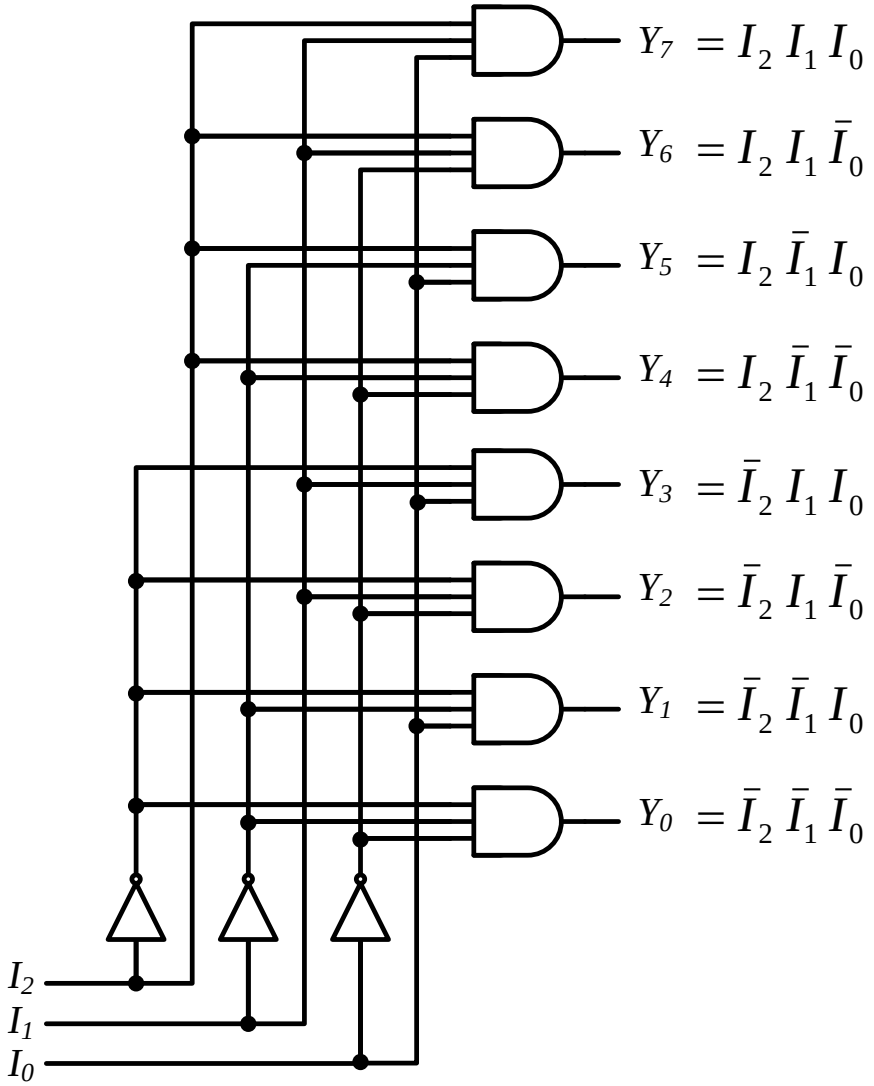
$$Y_3 = I_1 I_0 \qquad Y_2 = I_1 \bar{I}_0$$

$$Y_1 = \bar{I}_1 I_0 \qquad Y_0 = \bar{I}_1 \bar{I}_0$$

Decoders

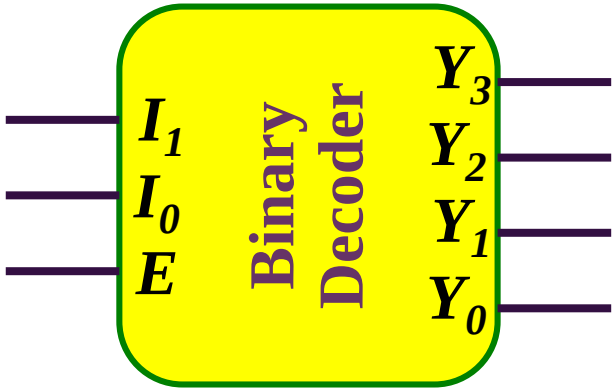
- 3-to-8 Line Decoder



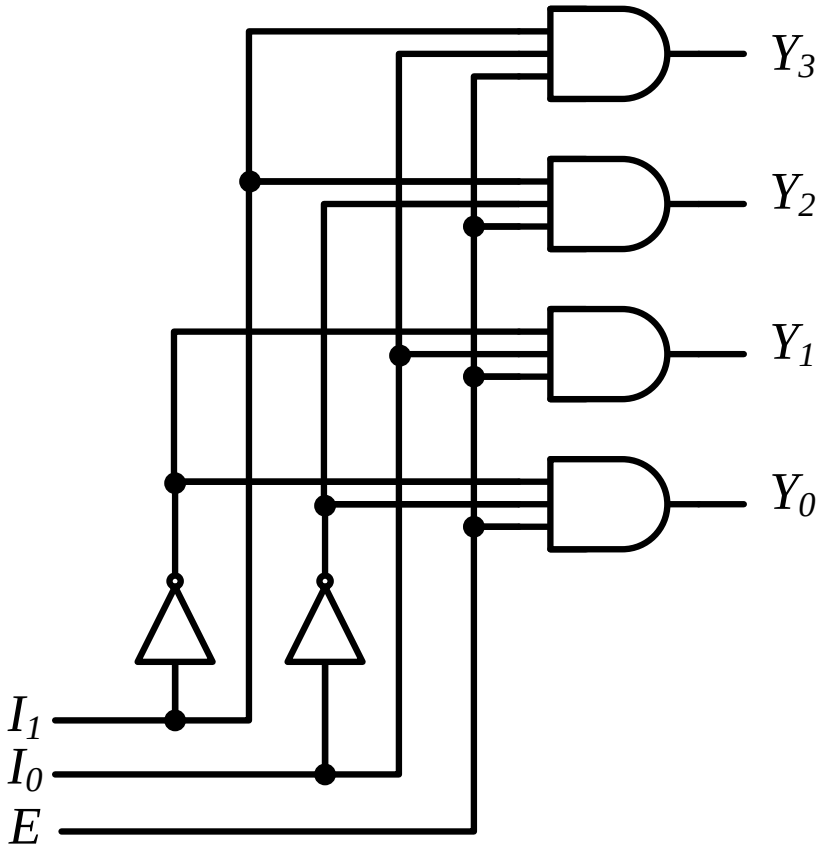


Decoders

- "*Enable*" Control



E	I_1 I_0	Y_3 Y_2 Y_1 Y_0
0	x x	0 0 0 0
1	0 0	0 0 0 1
1	0 1	0 0 1 0
1	1 0	0 1 0 0
1	1 1	1 0 0 0



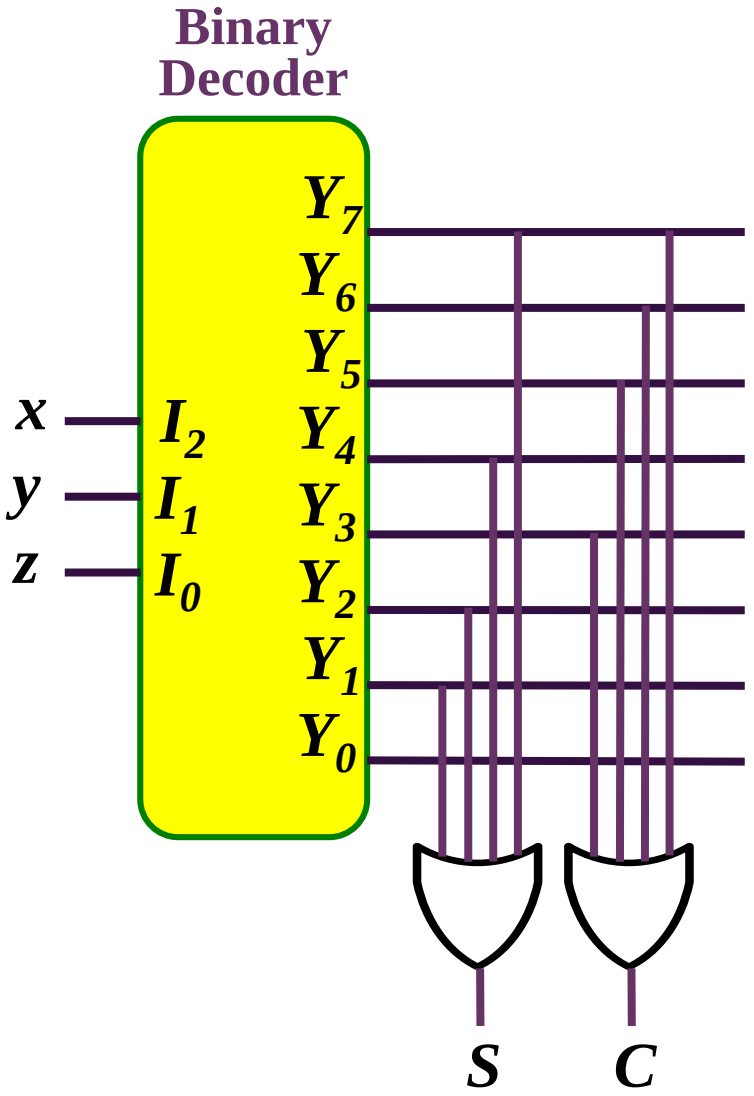
Implementation Using Decoders

- Each output is a minterm
- All minterms are produced
- Sum the required minterms

Example: Full Adder

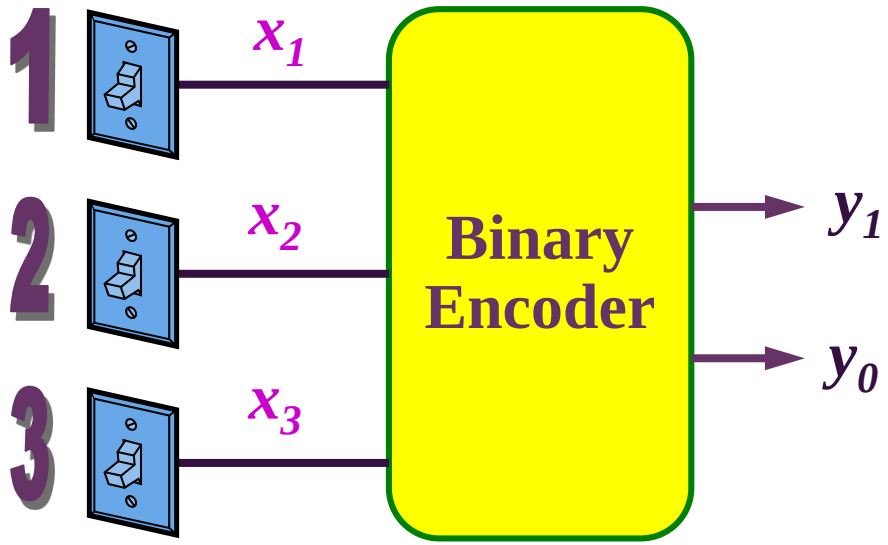
$S(x, y, z) = \sum(1, 2, 4, 7)$

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



Encoders

- Put "*Information*" into code
- Binary Encoder
  - Example: 4-to-2 Binary Encoder



Only *one* switch should be activated at a time

x_3	x_2	x_1	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1

Encoders

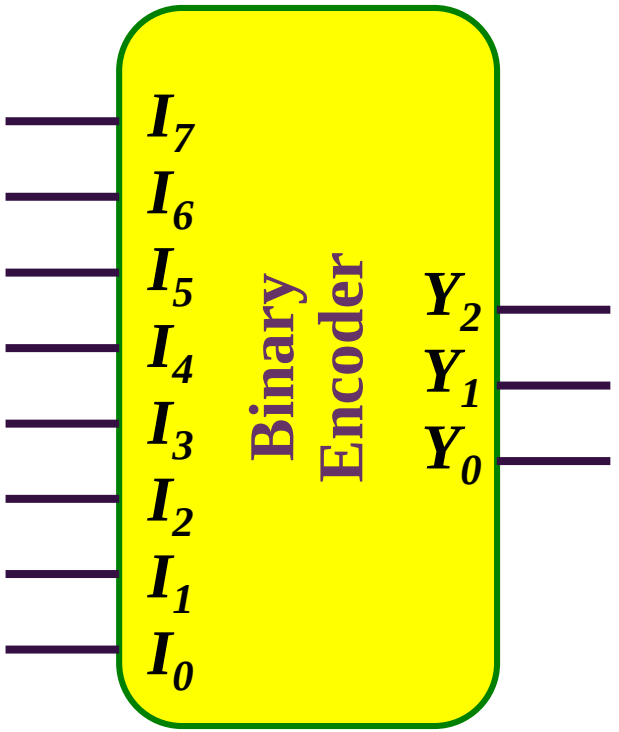
- Octal-to-Binary Encoder (8-to-3)

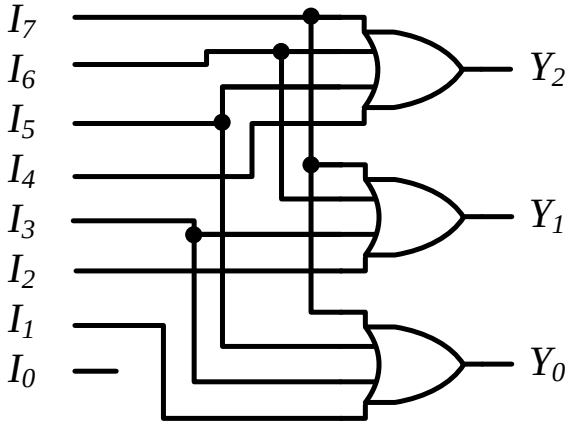
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y_2 = I_7 + I_6 + I_5 + I_4$$

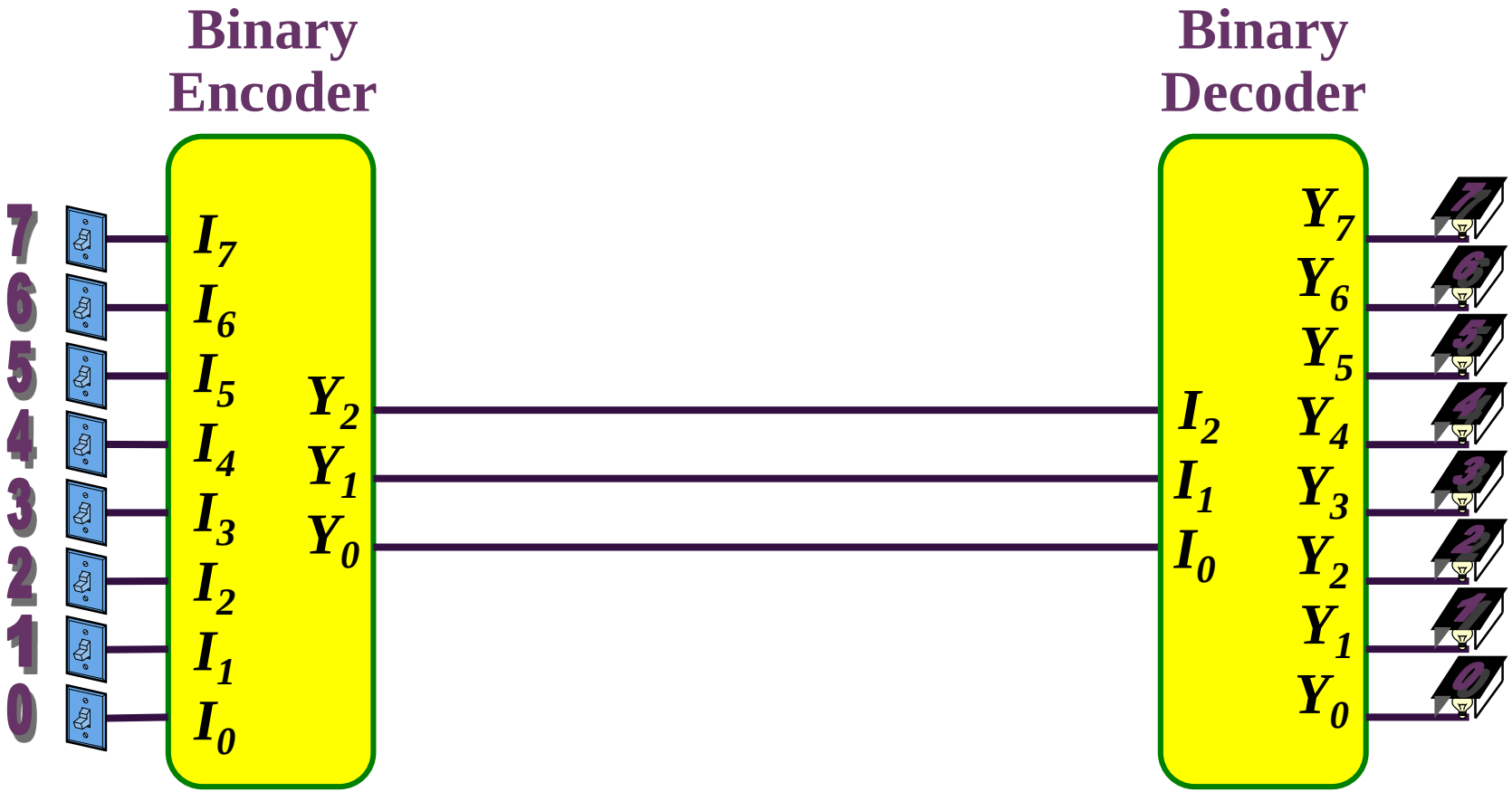
$$Y_1 = I_7 + I_6 + I_3 + I_2$$

$$Y_0 = I_7 + I_5 + I_3 + I_1$$

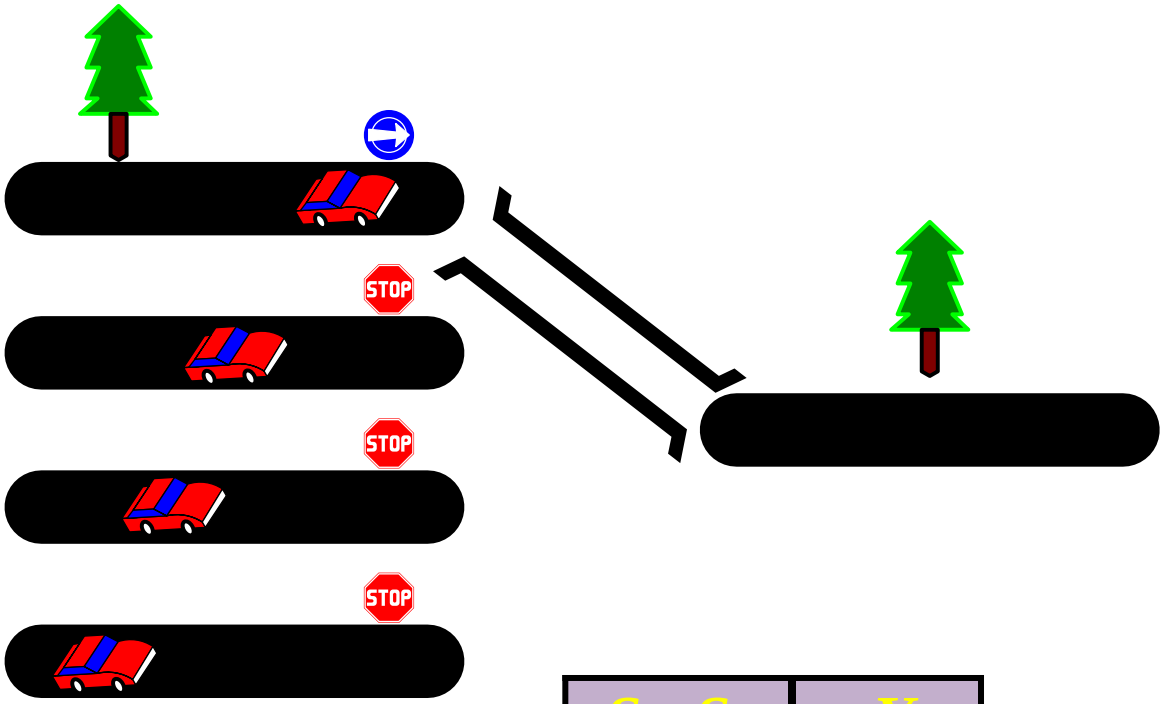




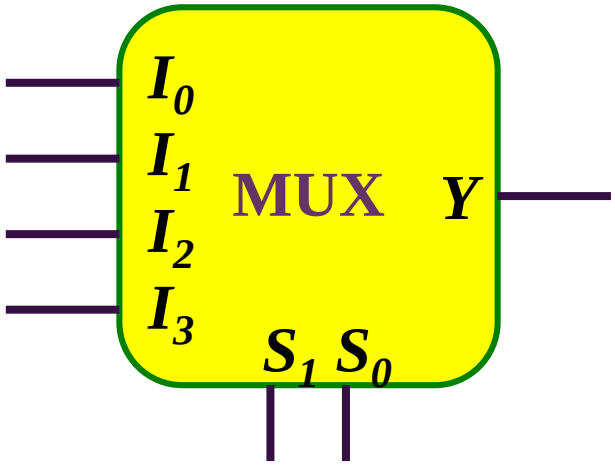
Encoder / Decoder Pairs



Multiplexers

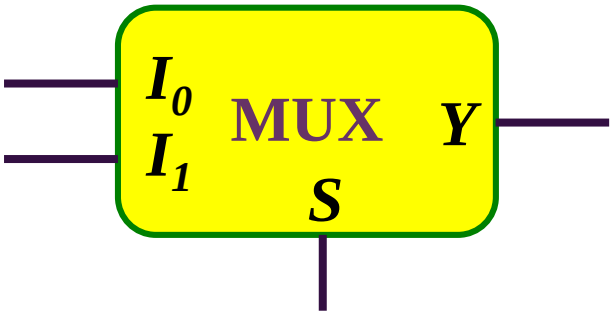


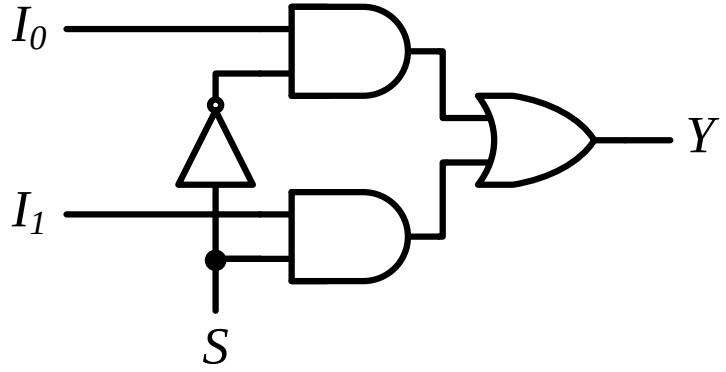
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



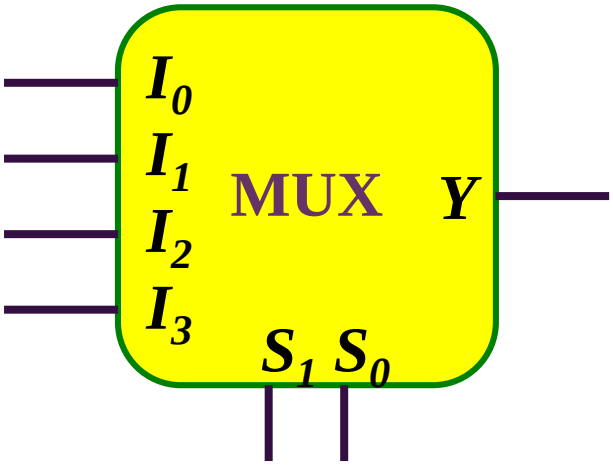
Multiplexers

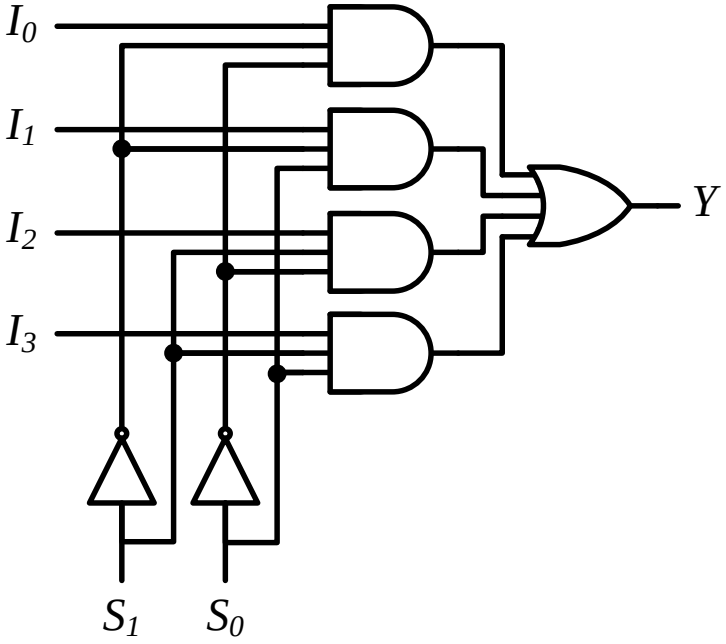
- 2-to-1 MUX





- 4-to-1 MUX



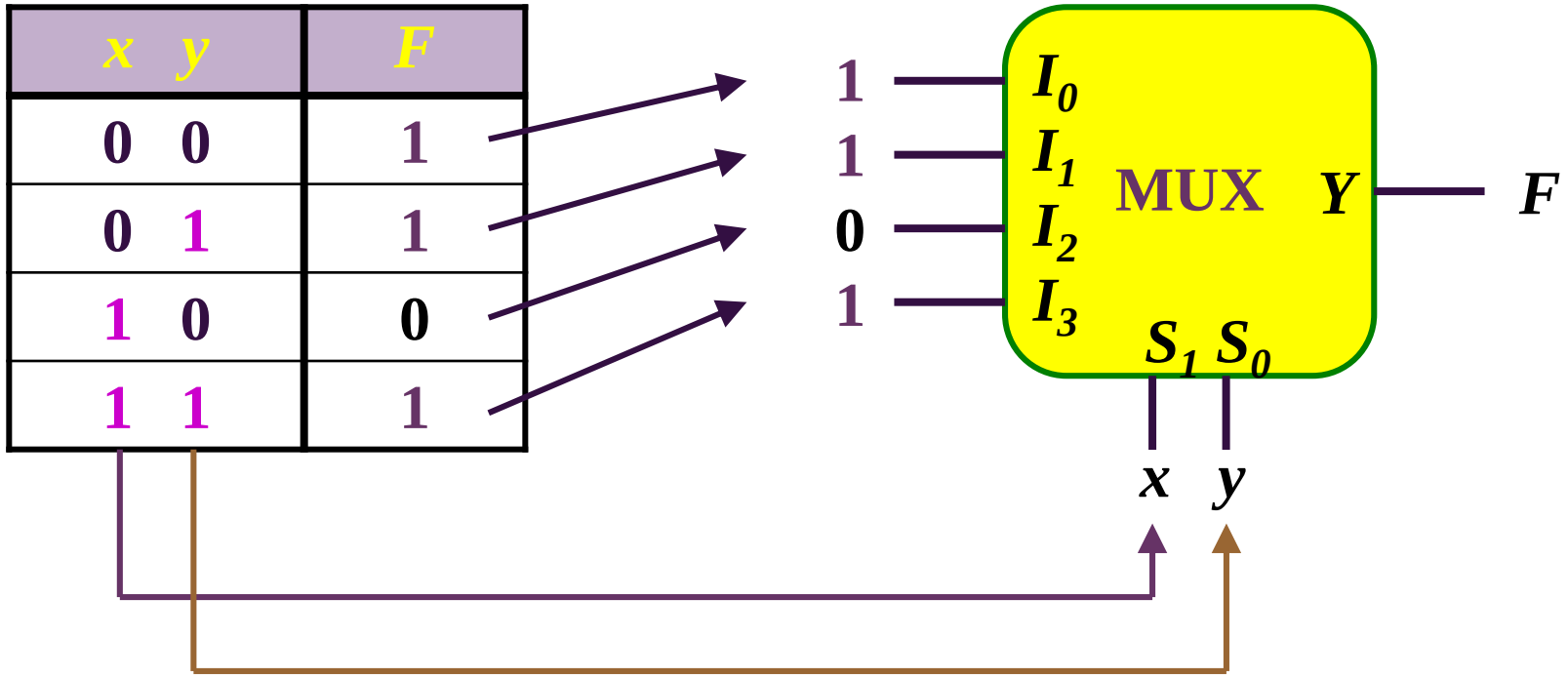


Implementation Using Multiplexers

- Example

  $F(x, y) = \sum(0, 1, 3)$

x	y	F
0	0	1
0	1	1
1	0	0
1	1	1

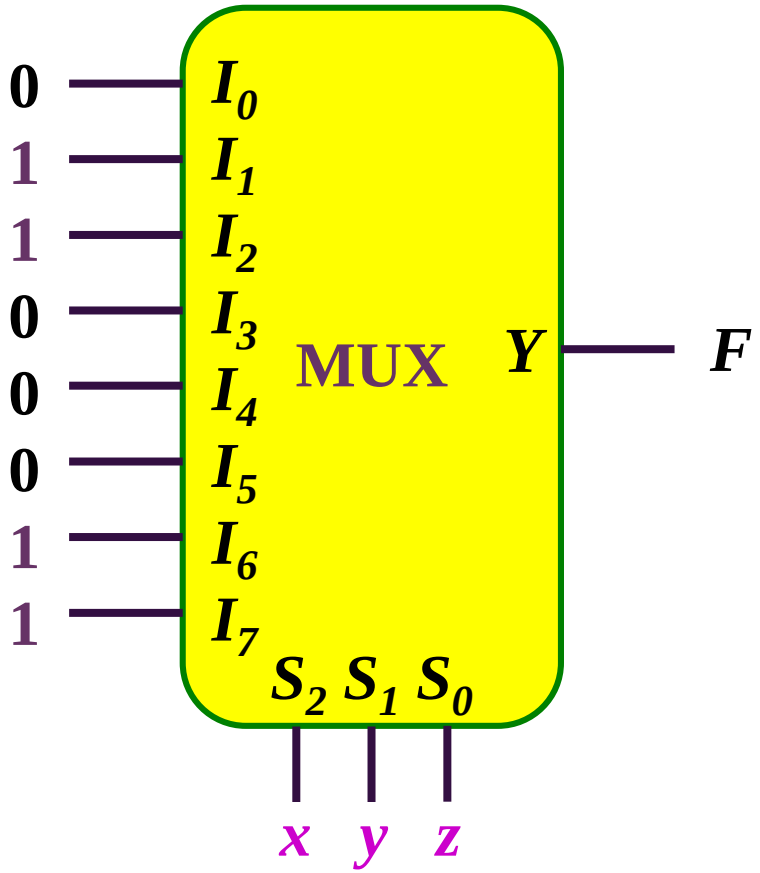


Implementation Using Multiplexers

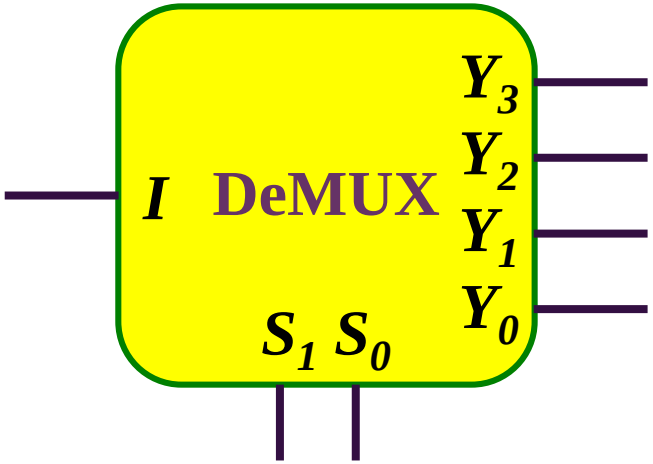
- Example

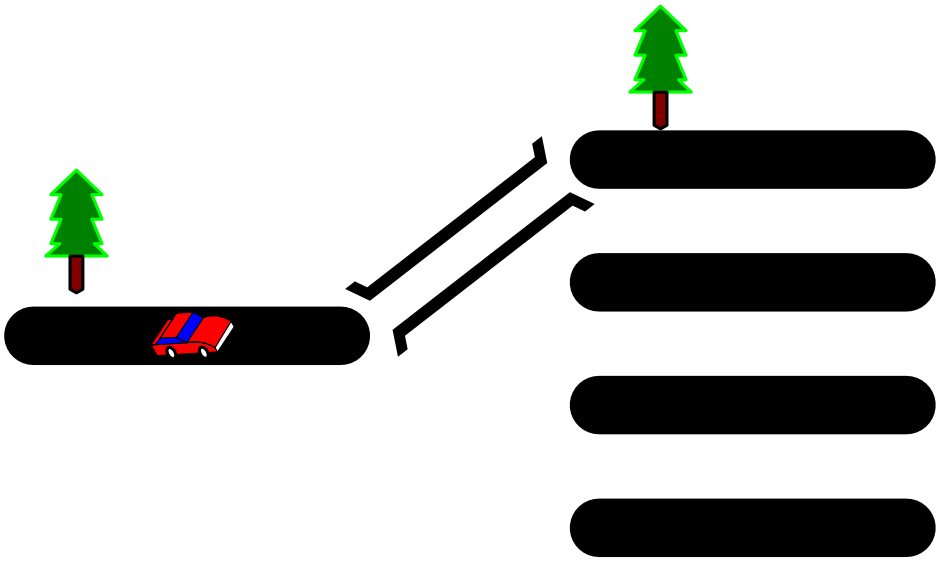
  $F(x, y, z) = \sum(1, 2, 6, 7)$

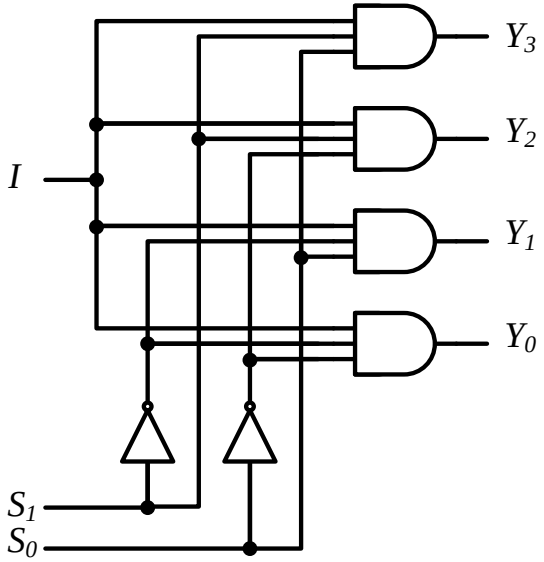
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



DeMultiplexers

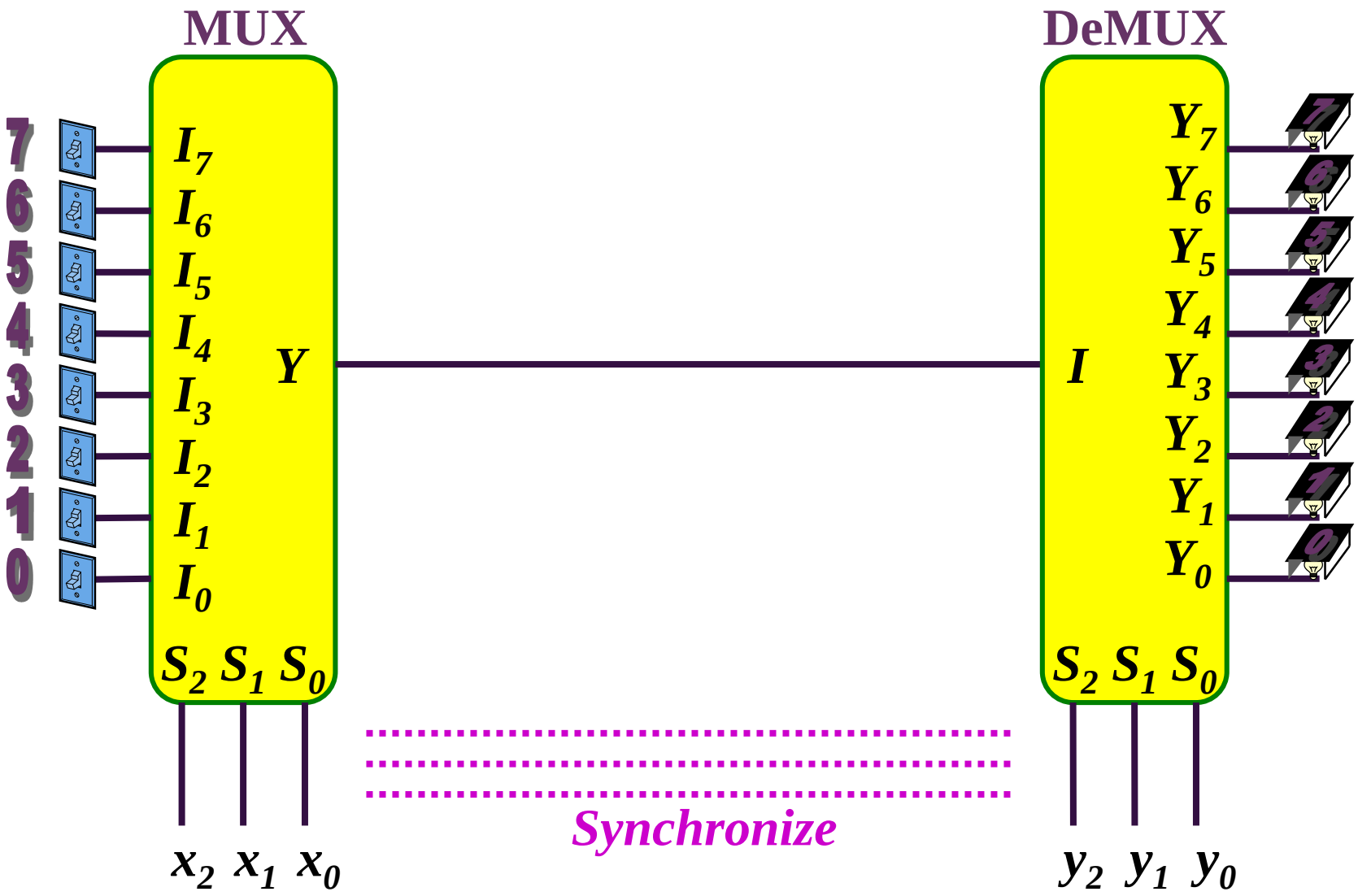




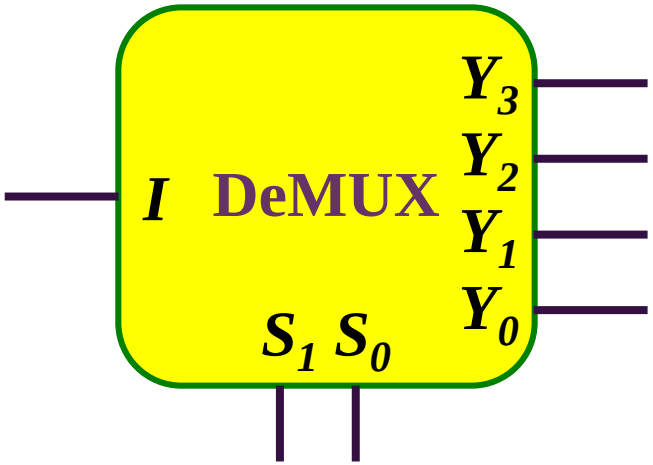


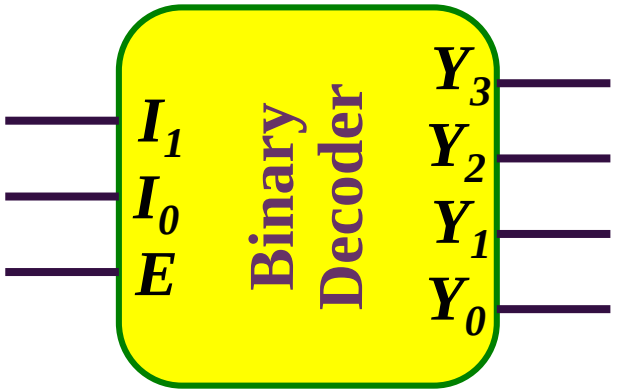
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Multiplexer / DeMultiplexer Pairs



DeMultiplexers / Decoders





S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

E	I_1	I_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0