ESC201: Lecture 13

Dr. Imon Mondal

Assistant Professor,
Electrical Engineering, IIT Kanpur

Boolean Expressions & Truth Tables

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$$\overline{x_1}\,.\,\overline{x_2}$$

$$x_1\,.\,x_2$$

$$y = \overline{x_1}\,.\,\overline{x_2} + x_1 . x_2$$

Sum of Products (SOP) form

Boolean Expressions & Truth Tables

Instead of writing expressions as sum of terms that make y equal to 1, we can also write expressions using terms that make y equal to 0

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

$$y=\overline{x_1}.\,\overline{x_2}+\overline{x_1}.\,x_2+x_1.\overline{x_2}$$

Here we are telling when y will be true

$$y=\overline{x_1.x_2}$$

FALSE when both are true

Here we are telling when y will be false

$$y=\overline{x_1}+\overline{x_2}$$

Recall

$$\overline{x_1}+\overline{x_2}=\overline{x_1\,.\,x_2}$$

Boolean Expressions & Truth Tables

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = \overline{x_1} \, . \, x_2 + x_1 . \overline{x_2}$$

Sum of Products (SOP) form

Boolean Expressions & Truth Tables

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$x_1 + x_2$$

$$\overline{x_1} + \overline{x_2}$$

y_1	y_2
0	1
1	1
1	1
1	0

$$(x_1+x_2)\cdot(\overline{x_1}+\overline{x_2})$$

$$y = (x_1 + x_2).(\overline{x_1} + \overline{x_2})$$

Product of Sum (POS) form

Digital Design

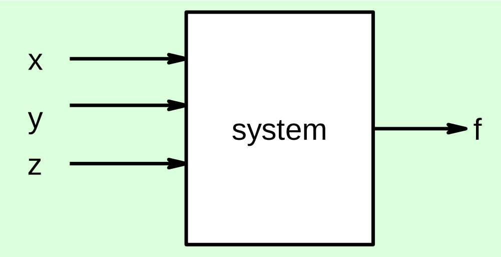
System Description

Truth Table

Boolean Expression

Minimized Boolean Expression

Gate Netlist

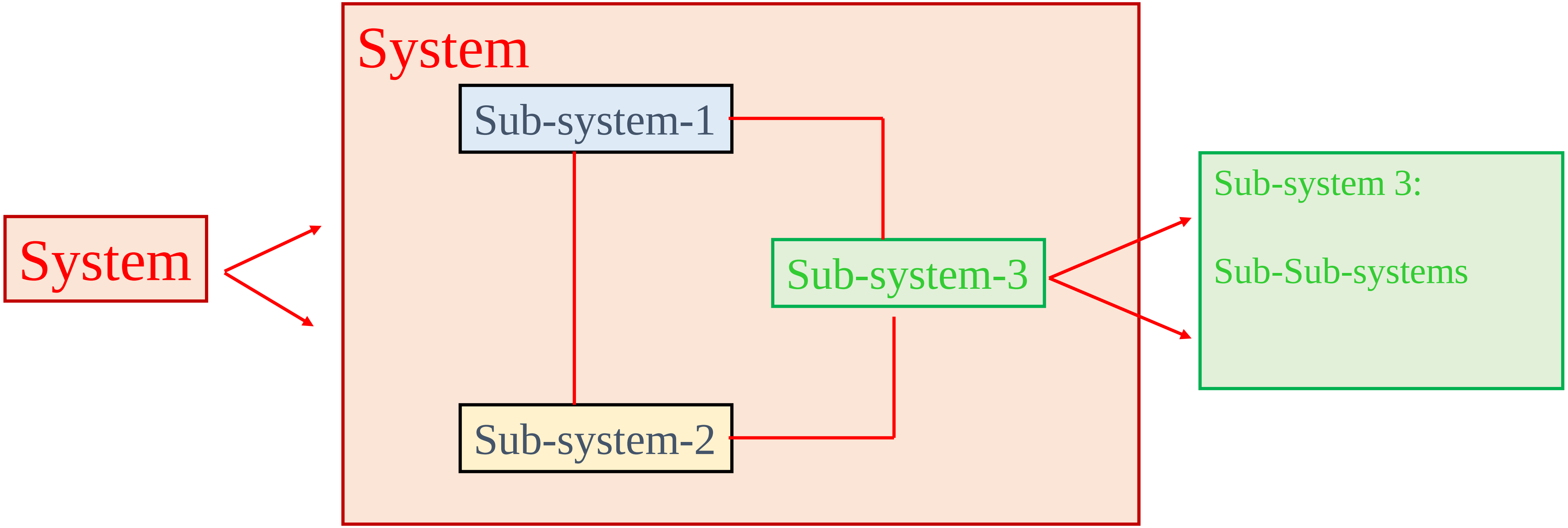


x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$f = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.z + x.y.z$$

$$\Rightarrow \; f = z$$

Modular Approach



There are certain sub-systems or blocks that are used quite often such as :

1. Adder/Subtractors, Multipliers
2. Decoders, Encoders
3. Multiplexers, Demultiplexers
4. Comparators
5. Parity Generators

Binary Addition

$$\begin{array}{r} 0 \\ 0 \\ \hline 0 \\ \hline \end{array}$$

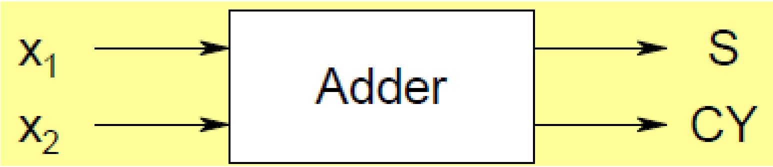
$$\begin{array}{r} 1 \\ 0 \\ \hline 1 \\ \hline \end{array} \qquad \begin{array}{r} 0 \\ 1 \\ \hline 1 \\ \hline \end{array}$$

$$\begin{array}{r} 1 \\ 1 \\ \hline 1\;0 \\ \hline \end{array}$$

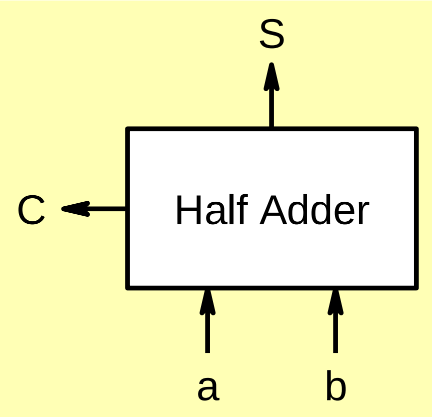
$$\begin{array}{r} 1 \\ 1 \\ 1 \\ \hline 1\;1 \\ \hline \end{array}$$

$$\begin{array}{r} 1\;0\;1 \\ 1\;1\;0 \\ \hline 1\;0\;1\;1 \\ \hline \end{array}$$

$$\begin{array}{r} 1\;1\;0\;1 \\ +\;\;1\;1\;1\;0 \\ \hline 1\;1\;0\;1\;1 \\ \hline \end{array}$$



Addition



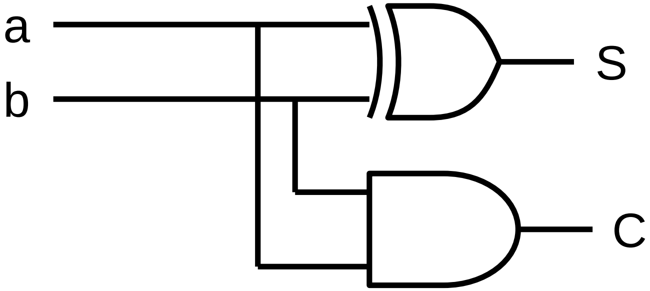
a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

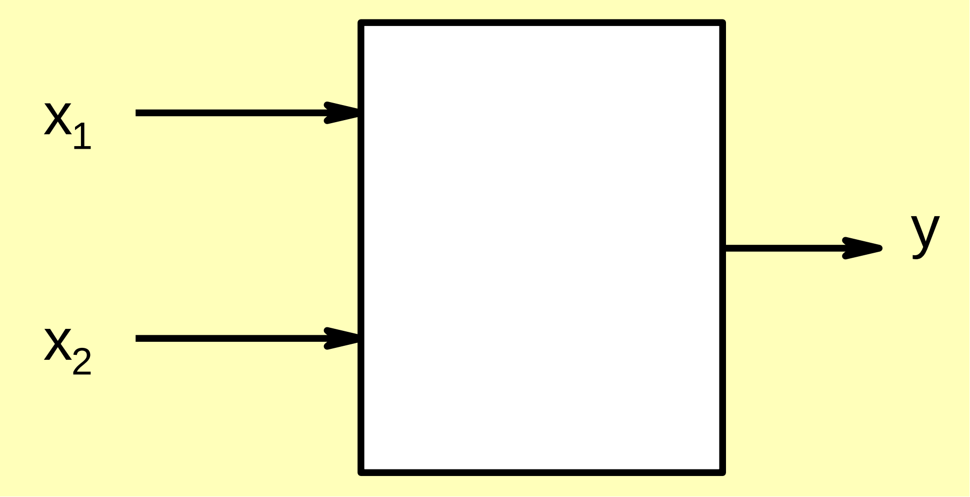
$$S = \bar{a}.b + a.\bar{b}; C = a.b$$

How to get this expression?

How to get this gate implementation?



How to get an expression from truth table?



x_1	x_2	y
0	0	0
0	1	1
1	0	0
1	1	0

y = 1 when x_1 is 0 and x_2 is 1

Boolean expression

$$y = \overline{x_1} \, . \, x_2$$

(NOT x_1) AND x_2

How to get an expression from truth table?

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0

$$y = \overline{x_1} \, . \, \overline{x_2}$$

x_1	x_2	y
0	0	0
0	1	0
1	0	1
1	1	0

$$y = x_1 \, . \, \overline{x_2}$$

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

y_1 $\overline{x_1} \, . \, \overline{x_2}$ y_2

y_1	y_2
1	0
0	0
0	0
0	1

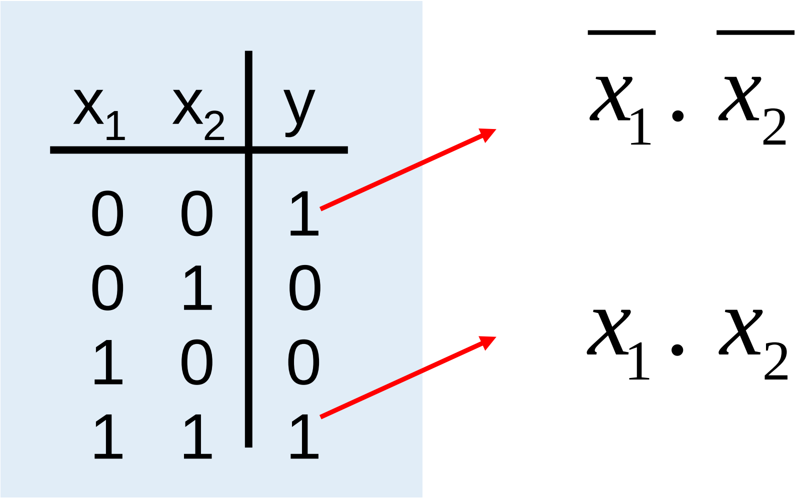
$x_1 \, . \, x_2$

$y = y_1 + y_2$

$$y = \overline{x_1} \, . \, \overline{x_2} + x_1 . x_2$$

(NOT x_1) AND (NOT x_2) OR x_1 AND x_2

Boolean Expressions & Truth Tables



x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

$$\overline{x_1}\,.\,\overline{x_2}$$

$$x_1\,.\,x_2$$

$$y = \overline{x_1}\,.\,\overline{x_2} + x_1 . x_2$$

Sum of Products (SOP) form

Boolean Expressions & Truth Tables

Instead of writing expressions as sum of terms that make y equal to 1, we can also write expressions using terms that make y equal to 0

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

$$y=\overline{x_1}.\,\overline{x_2}+\overline{x_1}.\,x_2+x_1.\overline{x_2}$$

Here we are telling when y will be true

$$y = \overline{x_1 . x_2}$$

FALSE when both are true

Here we are telling when y will be false

$$y = \overline{x_1} + \overline{x_2}$$

Recall

$$\overline{x_1} + \overline{x_2} = \overline{x_1\,.\,x_2}$$

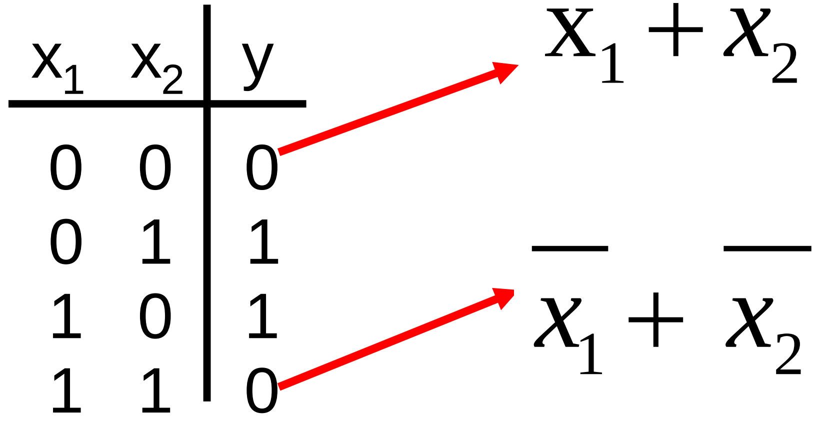
Boolean Expressions & Truth Tables

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$y = \overline{x_1} . \, x_2 + x_1 . \overline{x_2}$$

Sum of Products (SOP) form

Boolean Expressions & Truth Tables



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$$x_1 + x_2$$

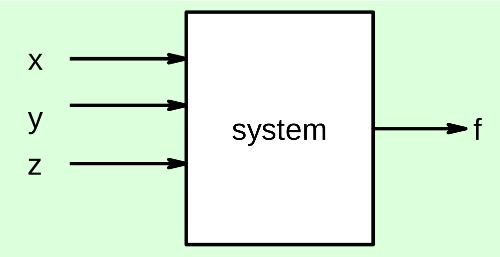
$$\overline{x_1} + \overline{x_2}$$

$$y = (x_1 + x_2).(\overline{x_1} + \overline{x_2})$$

Product of Sum (POS) form

Digital Design

System Description



Truth Table

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Boolean Expression

$$\mathrm{f} = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}.z + x.y.z$$

Minimized Boolean Expression

$$\Rightarrow \ \mathrm{f} = z$$

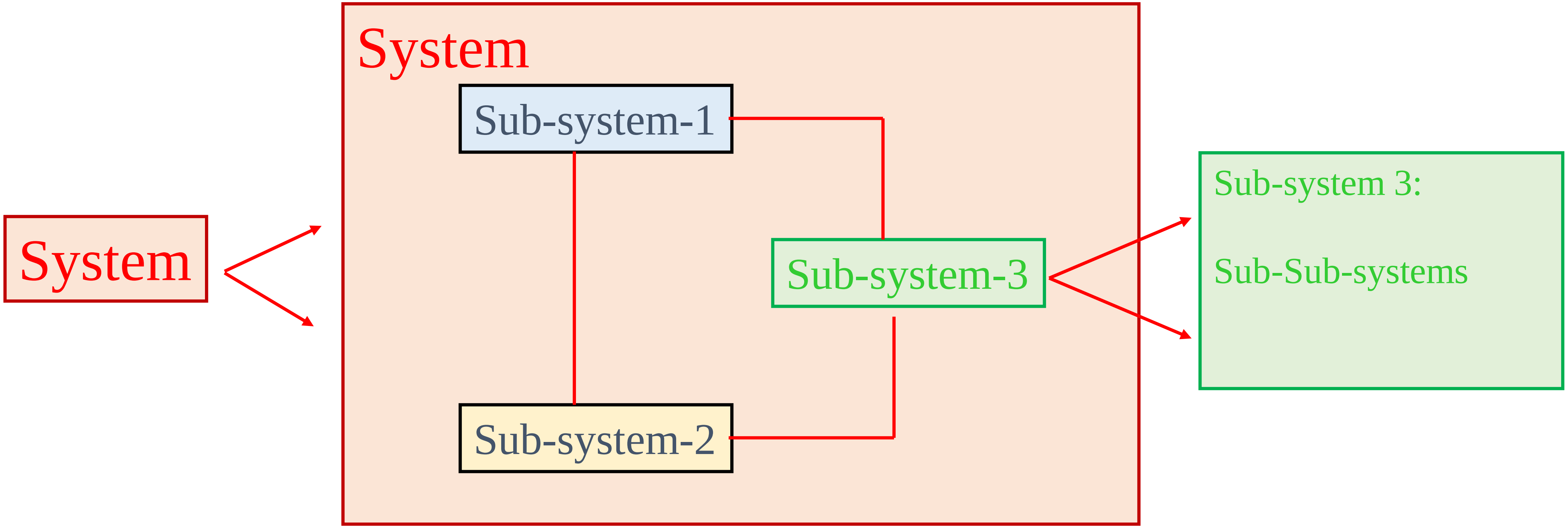
Gate Netlist

Calculation using Digital System

- Binary signals represent logic states
  - can implement any logic
  - logic consists of AND, OR & NOT condition
  - Boolean Algebra
- Binary signals can represent any numbers
  - We can do all arithmetic over it
  - Enables any calculations
  - Addition, Subtraction, Multiplication, Division, etc.
- Computers can do calculations
  - evaluate logic states and make decision over that

How to do such calculations?

Modular Approach



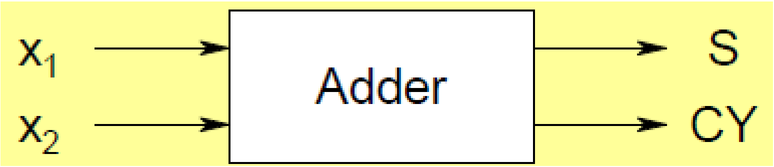
There are certain sub-systems or blocks that are used quite often such as :

1. Adder/Subtractors, Multipliers
2. Decoders, Encoders
3. Multiplexers, Demultiplexers
4. Comparators
5. Parity Generators

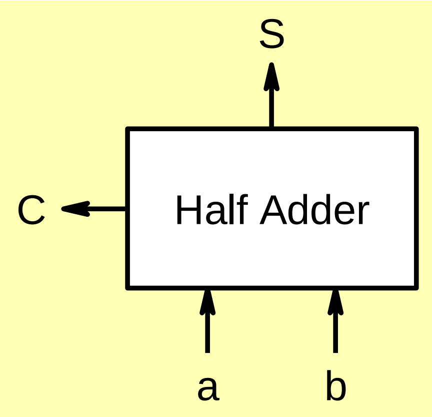
Binary Addition

$$\begin{array}{r} 0 \\ \underline{0} \\ \underline{0} \end{array} \qquad \begin{array}{r} 1 \\ \underline{0} \\ \underline{1} \end{array} \qquad \begin{array}{r} 0 \\ \underline{1} \\ \underline{1} \end{array} \qquad \begin{array}{r} 1 \\ \underline{1} \\ \underline{1\ 0} \end{array} \qquad \begin{array}{r} 1 \\ 1 \\ \underline{1} \\ \underline{1\ 1} \end{array}$$

$$\begin{array}{r} 1\ 0\ 1 \\ \underline{1\ 1\ 0} \\ \underline{1\ 0\ 1\ 1} \end{array} \qquad \begin{array}{r} 1\ 1\ 0\ 1 \\ \underline{+\ \ 1\ 1\ 1\ 0} \\ 1\ \underline{1\ 0\ 1\ 1} \end{array}$$



1 bit Addition: Half Adder

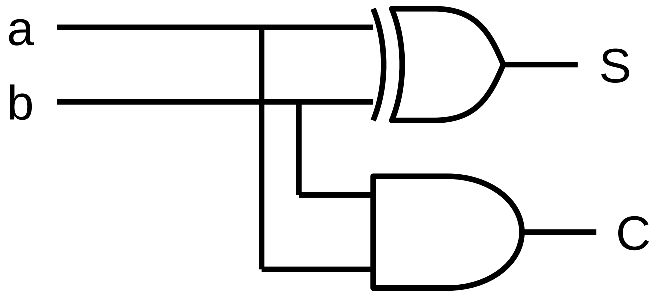


a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

$$S = \bar{a}.b + a.\bar{b}; C = a.b$$

Boolean Expression



Gate level

implementation

Why a Modular Approach?

- Let us make a 2 bit adder circuit which can add two 2-bit numbers

$$\begin{array}{ccc} & x_1 & x_0 \\ + & y_1 & y_0 \\ \cline{2-3} z_2 & z_1 & z_0 \end{array}$$

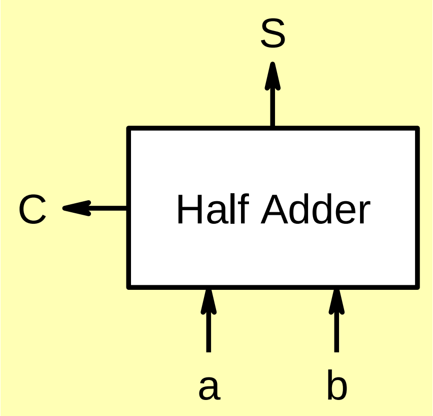
- There are 4 inputs and 3 outputs
- Let us write down all possible combinations!
  - 2^4 = 16 rows in the truth table

➢ Write down Boolean expressions and design implementation?

➢ What about 3 bits?

❑ Let us take the modular approach

Adder: First bit



$$\begin{array}{c}0\\ \underline{0}\\ \underline{0}\end{array}$$

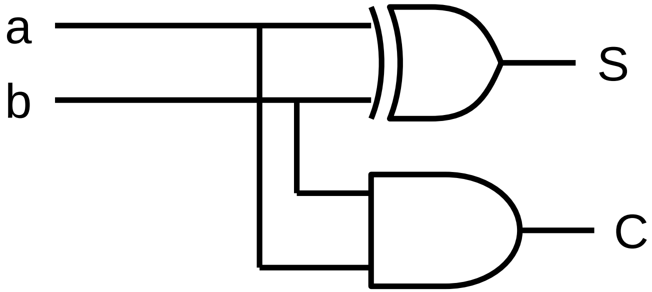
$$\begin{array}{c}1\\ \underline{0}\\ \underline{1}\end{array} \qquad \begin{array}{c}0\\ \underline{1}\\ \underline{1}\end{array}$$

$$\begin{array}{c}1\\ \underline{1}\\ \underline{1\ 0}\end{array}$$

Truth Table

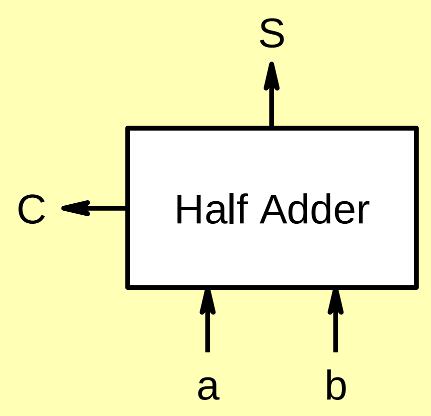
a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

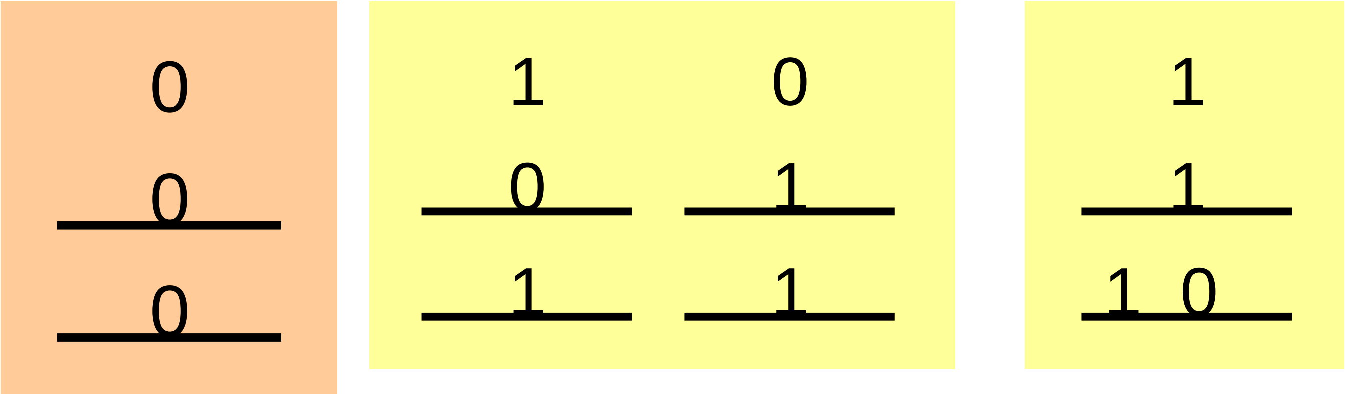
$$S = \bar{a}.b + a.\bar{b} ; C = a.b$$



Implementation

Adder: Second bit

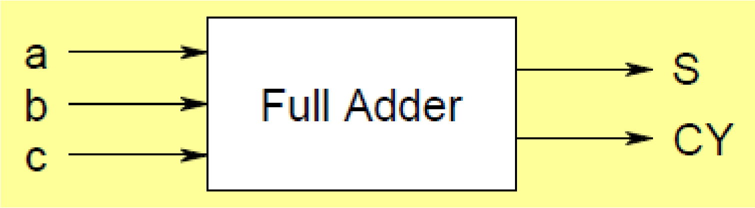




But there can be carry from previous bits.

$$\begin{array}{ccc} & 1 & \\ & x_1 & 1 \\ + & y_1 & 1 \\ \hline z_2 & z_1 & 0 \end{array}$$

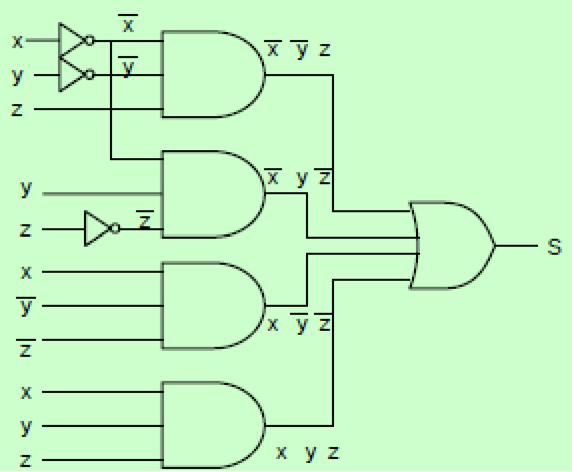
Single Bit Full Adder



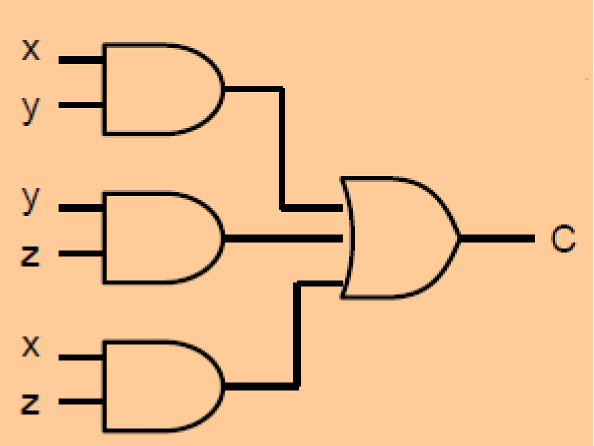
a	b	c	S	CY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \overline{x}.\overline{y}.z + \overline{x}.y.\overline{z} + x.\overline{y}.\overline{z} + x.y.z$$

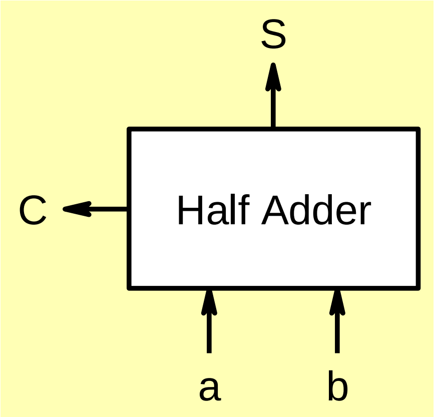
$$C = x.y + x.z + y.z$$



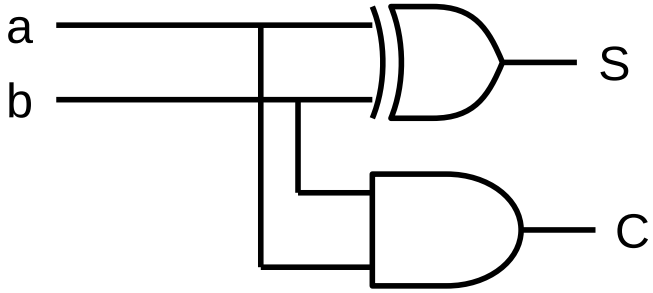
x
y
z
$\overline{x}$
$\overline{y}$
$\overline{x}\ \overline{y}\ z$
y
z
$\overline{z}$
$\overline{x}\ y\ \overline{z}$
x
$\overline{y}$
$\overline{z}$
$x\ \overline{y}\ \overline{z}$
x
y
z
x y z
s



Adder: Half Adder vs Full adder

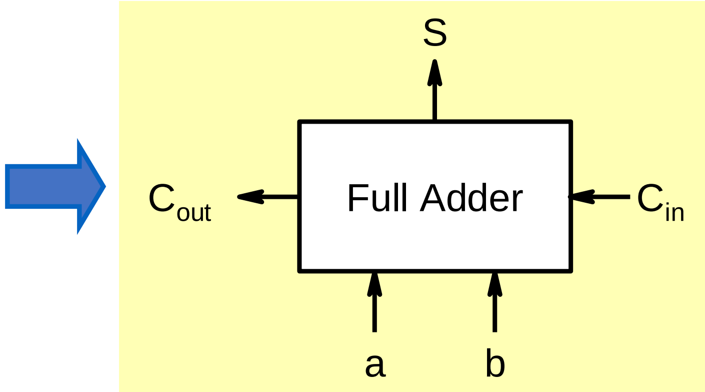


a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = \bar{a}.b + a.\bar{b}; C = a.b$$

```
 1
  1 1 1
  1 1 0
---------
1 1 0 1
```



a	b	C_{in}	S	C_{out}
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

$$S = \bar{a}.\bar{b}.c_{in} + \bar{a}.b.\overline{c_{in}} + a.\bar{b}.\overline{c_{in}} + a.b.c_{in};$$

$$C_{out} = \bar{a}.b.c_{in} + a.\bar{b}.c_{in} + a.b.\overline{c_{in}} + a.b.c_{in}$$

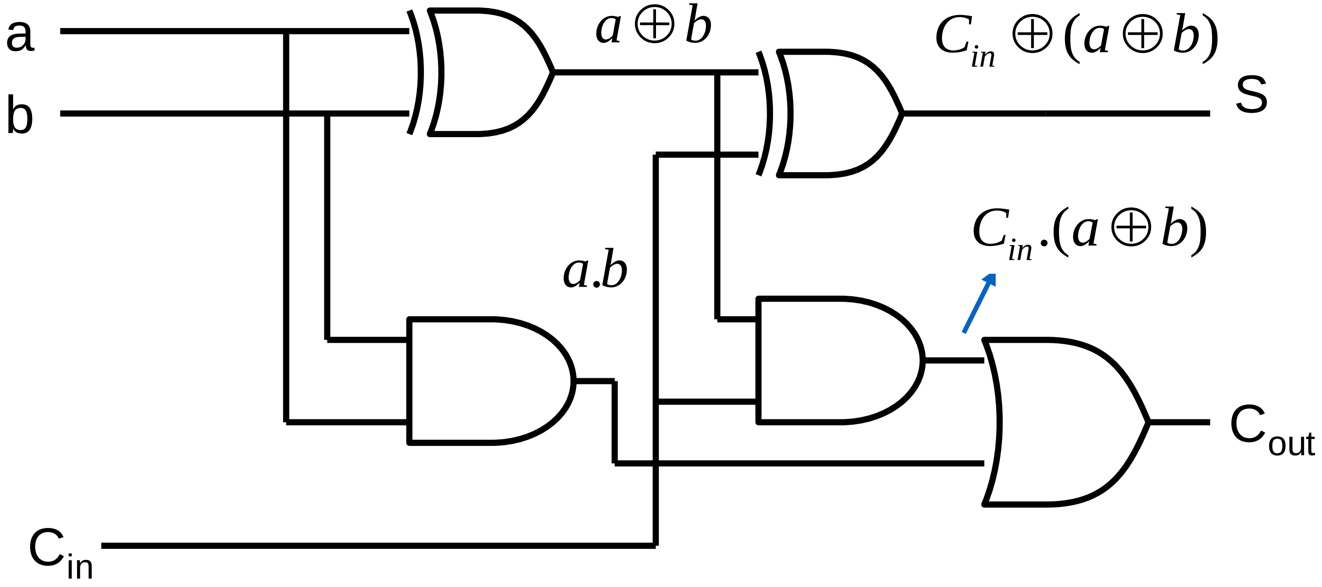
Full Adder Circuit using Half Adders

$$S = \bar{a}.\bar{b}.c_{in} + \bar{a}.b.\overline{c_{in}} + a.\bar{b}.\overline{c_{in}} + a.b.c_{in}$$

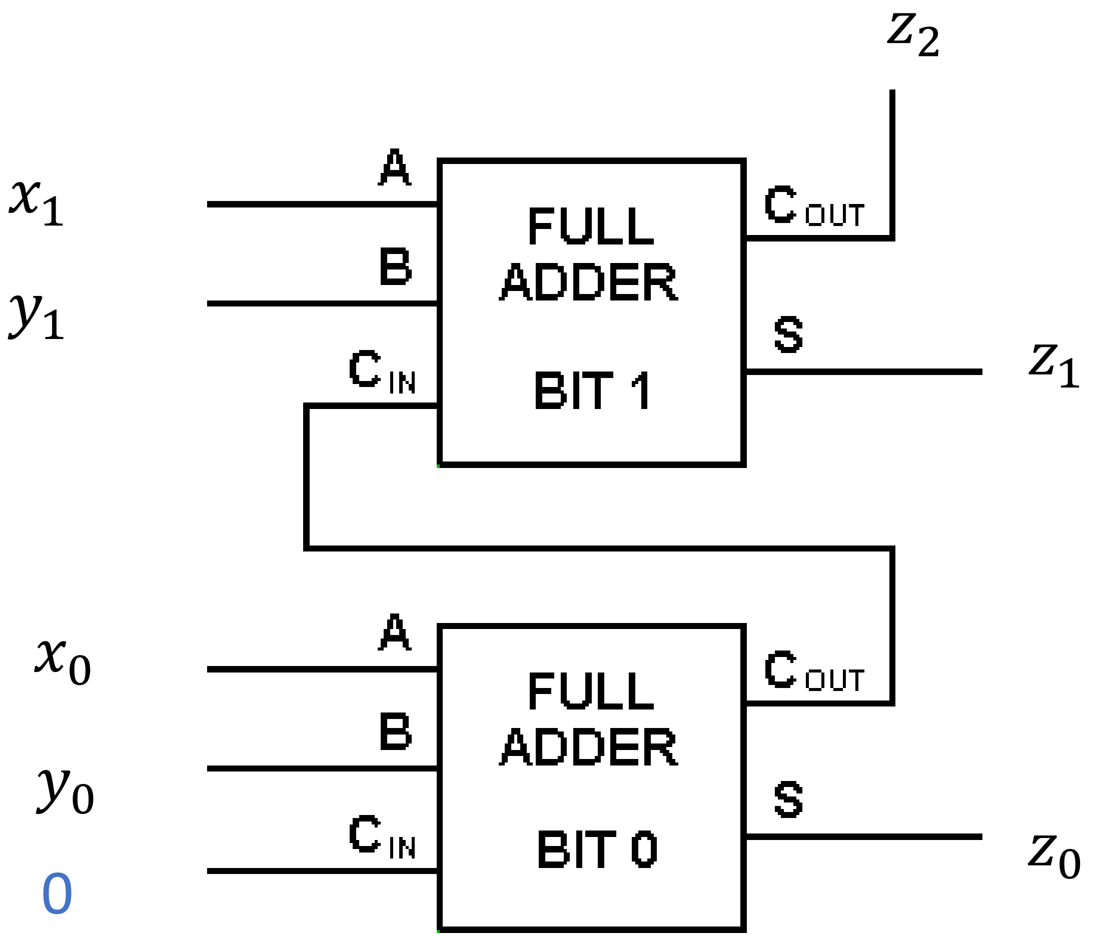
$$S = C_{in} \oplus (a \oplus b)$$

$$C_{out} = \bar{a}.b.C_{in} + a.\bar{b}.C_{in} + a.b.\overline{C_{in}} + a.b.C_{in}$$

$$C_{out} = C_{in}(a.\bar{b} + \bar{a}.b) + a.b = C_{in}.(a \oplus b) + a.b$$



Multi-bit Adder

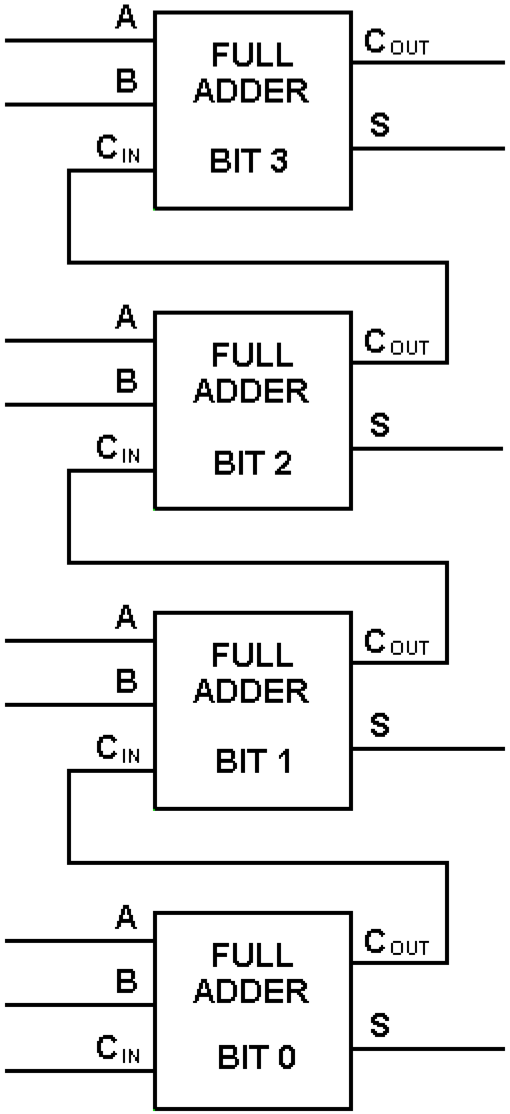


$$\begin{array}{cccc} & & x_1 & x_0 \\ + & & y_1 & y_0 \\ \hline & z_2 & z_1 & z_0 \end{array}$$

Multi-bit Adder

- How to add two 4-bit numbers?
- Truth table would have 2^8=256 entries
- Instead, use already designed logic circuits as subsystems

$$\begin{array}{r} 1\,1\,0\,1 \\ +\;\underline{1\,1\,1\,0} \\ 1\,\underline{1\,0\,1\,1} \end{array}$$



Addition/Subtraction Computation

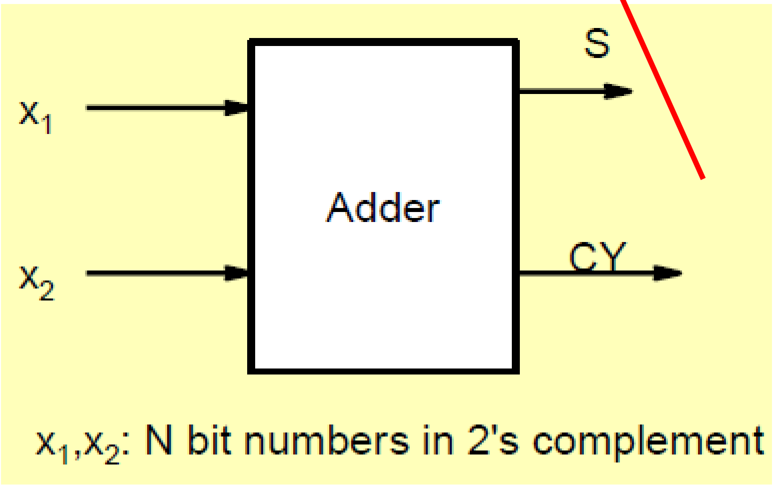
$$\begin{array}{r} +5 \\ +2 \\ \hline +7 \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\,0\,0\,1\,0 \\ \hline 0\,1\,1\,1 \\ \hline \end{array} \qquad \begin{array}{r} +5 \\ -2 \\ \hline +3 \\ \hline \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\,1\,1\,1\,0 \\ \hline 0\,0\,1\,1 \\ \hline \end{array}$$

$$\begin{array}{r} -5 \\ +2 \\ \hline -3 \\ \hline \end{array} \qquad \begin{array}{r} 1\,0\,1\,1 \\ +\,0\,0\,1\,0 \\ \hline 1\,1\,0\,1 \\ \hline \end{array} \qquad \begin{array}{r} -5 \\ -2 \\ \hline -7 \\ \hline \end{array} \qquad \begin{array}{r} 1\,0\,1\,1 \\ +\,1\,1\,1\,0 \\ \hline 1\,0\,0\,1 \\ \hline \end{array}$$

2's complement is 0011 = 3

2's complement is 0111 = 7

Answer is in 2's complement form



x_1,x_2: N bit numbers in 2's complement

Overflow

- Take care to detect overflow when adding

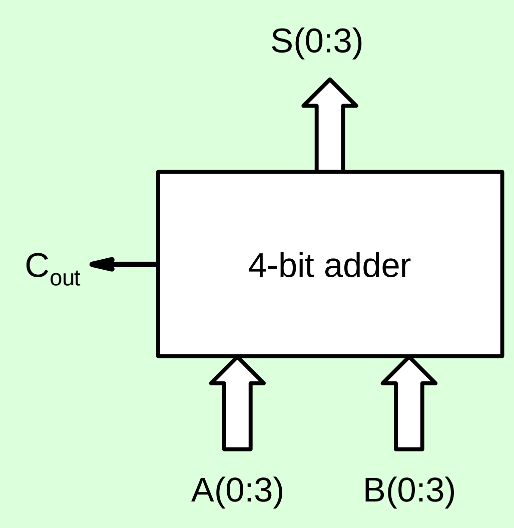
+ 5	0 0 1 0 1
+ 13	0 1 1 0 1
+ 18	0 1 0 0 1 0

After discarding the final carry 0,
2’s complement of 10010 is 01110 = $(14)_{10}$
We get a wrong answer!

- Sum of positive numbers = negative
- Sum of negative numbers = positive

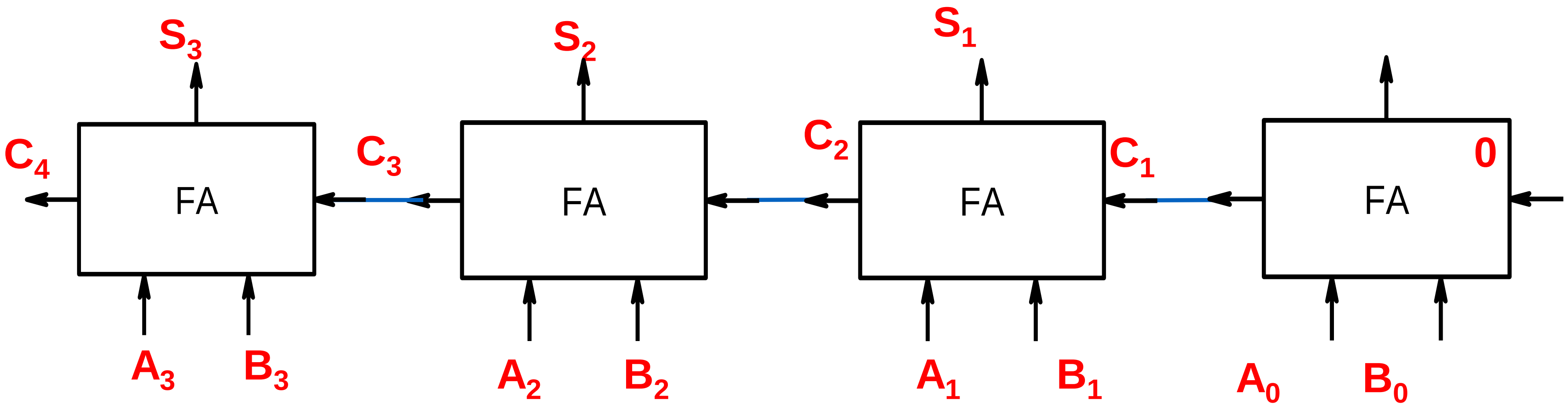
overflow

4-bit Adder



$A_3A_2A_1A_0$	$B_3B_2B_1B_0$	$S_3S_2S_1S_0$	C_{out}
0000	0000	0000	0
0000	0001	0001	0
0001	0000	0001	0
⋮	⋮	⋮	⋮

$$\begin{array}{ccccc} & C_3 & C_2 & C_1 & \\ & A_3 & A_2 & A_1 & A_0 \\ & B_3 & B_2 & B_1 & B_0 \\ \hline C_4 & S_3 & S_2 & S_1 & S_0 \\ \hline \end{array}$$

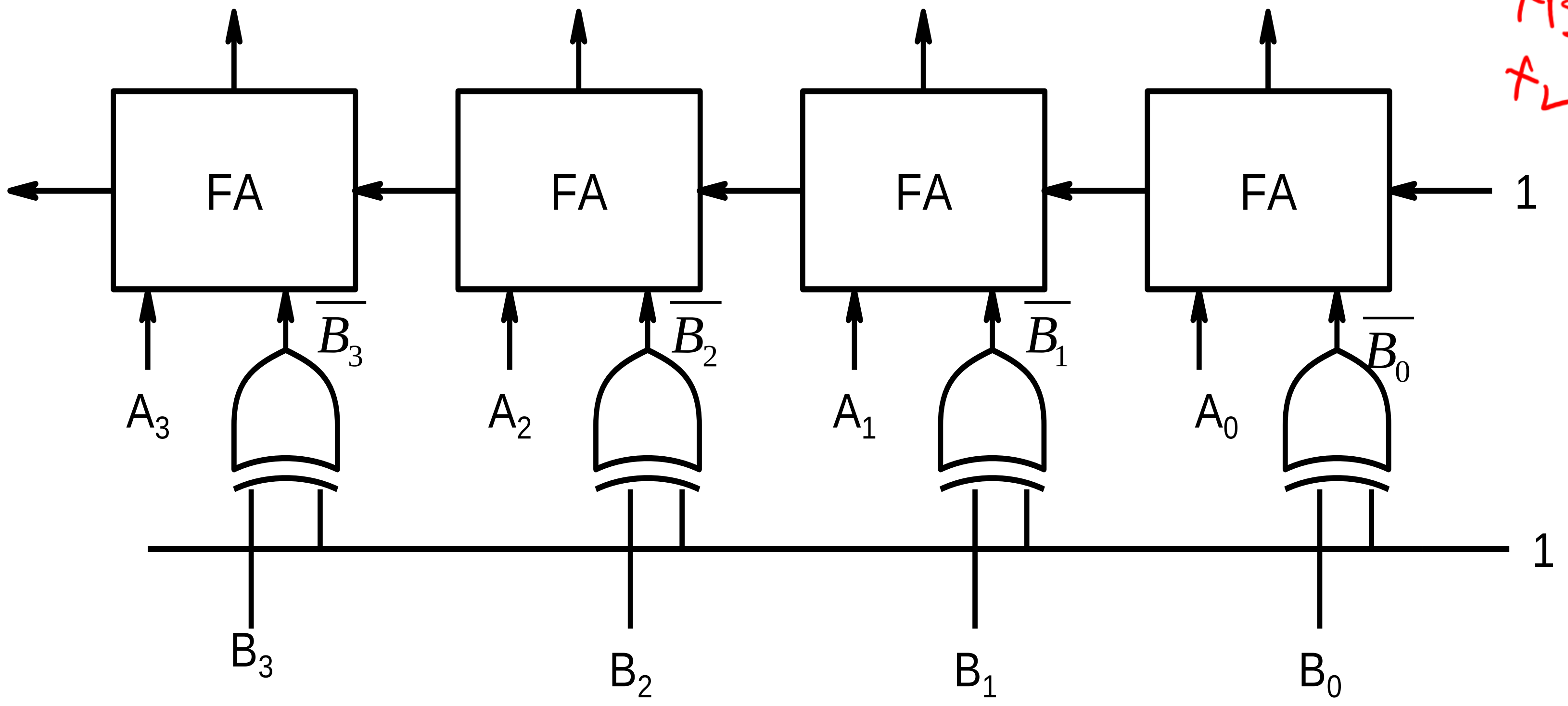


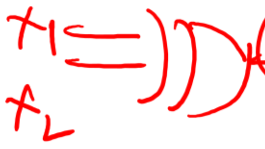
4-bit Subtractor

A – B = A + 2's complement of B

A – B = A + 1's complement of B + 1

$$A - B = A + \overline{B} + 1$$





$x_1\overline{x_2}$

$+\ \overline{x_1}\,x_2$

$x_2 = 1$

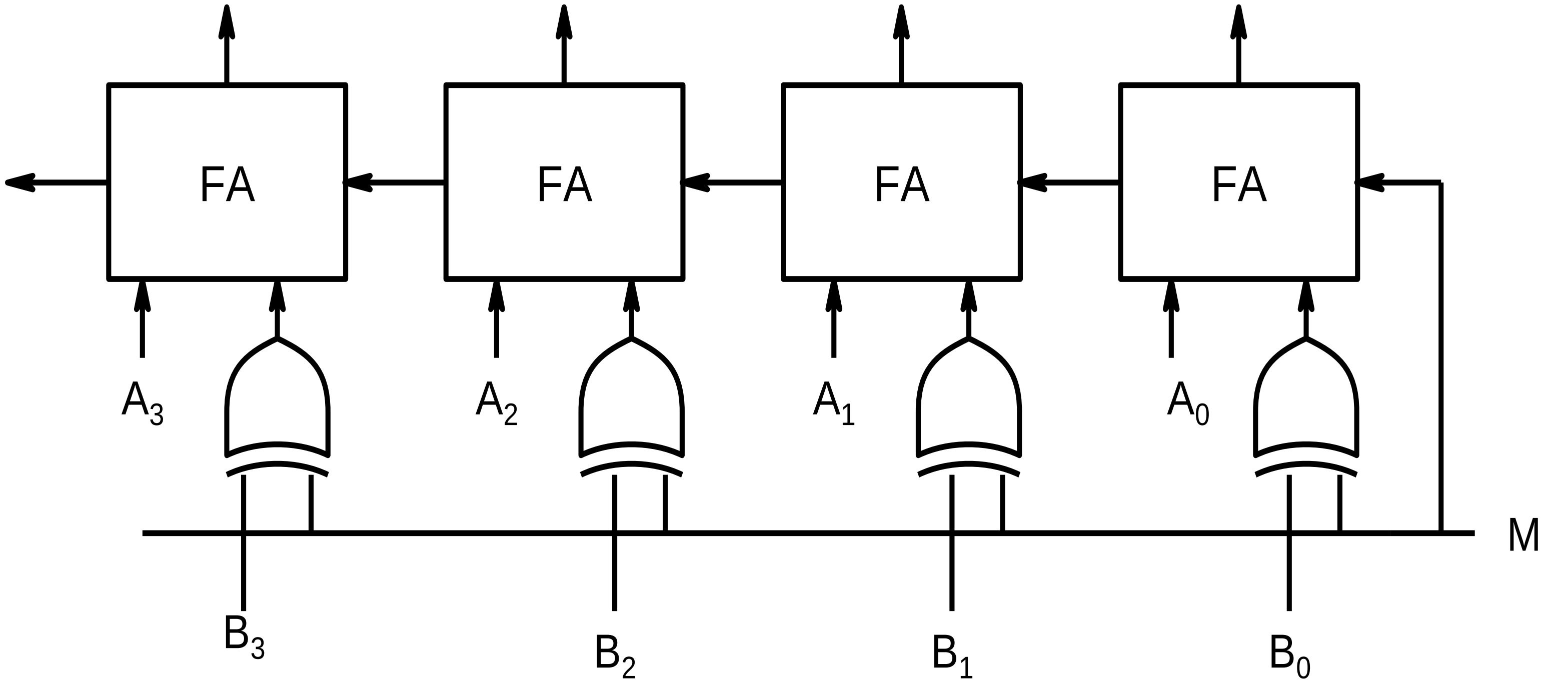
$\overline{x_1}$

$x_2 = 0$

x_1

$$B_0 \oplus 1 = B_0.\bar{1} + \overline{B_0}.1 = \overline{B_0}$$

4-bit Adder and Subtractor



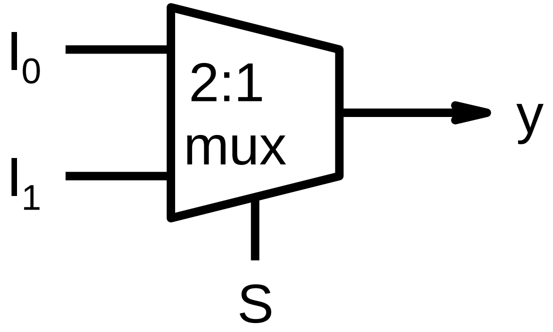
$$B_0 \oplus 0 = B_0.\bar{0} + \overline{B_0}.0 = B_0$$

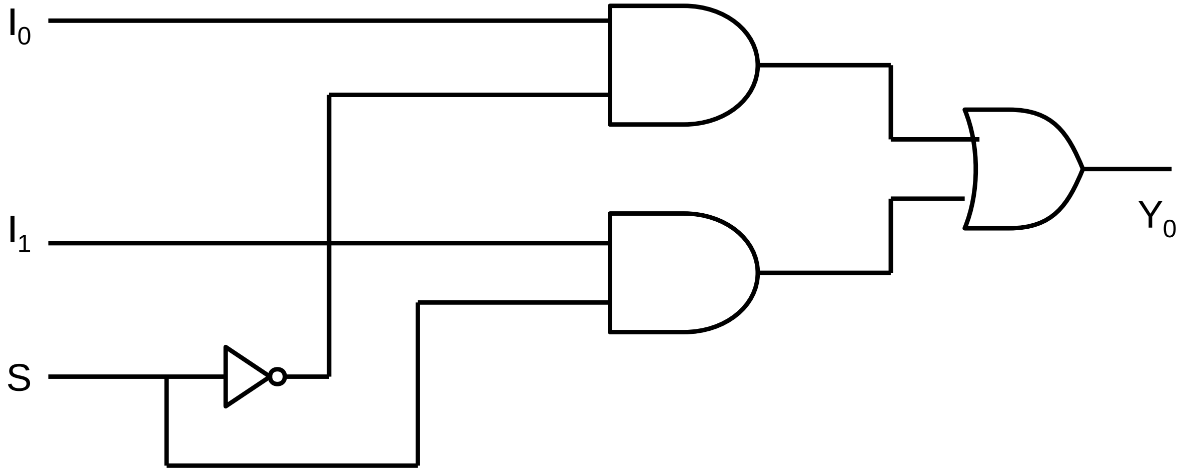
$$B_0 \oplus 1 = B_0.\bar{1} + \overline{B_0}.1 = \overline{B_0}$$

$M = 0$ for Adder

M=1 for Subtractor

Multiplexers (MUX)





S	y
0	I_0
1	I_1

is a shortcut to say

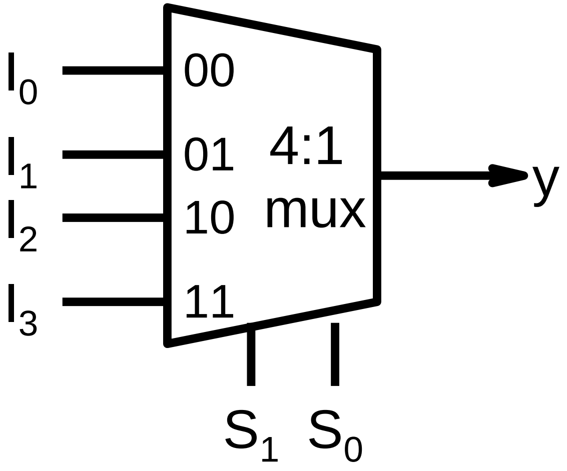
S	I_0	I_1	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$y = \bar{S}\, I_0 + S\, I_1$$

$$\begin{aligned} y &= \bar{S} I_0 \overline{I_1} + \bar{S} I_0 I_1 + S \overline{I_0} I_1 + S I_0 I_1 \\ &= \bar{S} I_0 (\overline{I_1} + I_1) + S I_1 (\overline{I_0} + I_0) \\ &= \bar{S}\, I_0 + S\, I_1 \end{aligned}$$

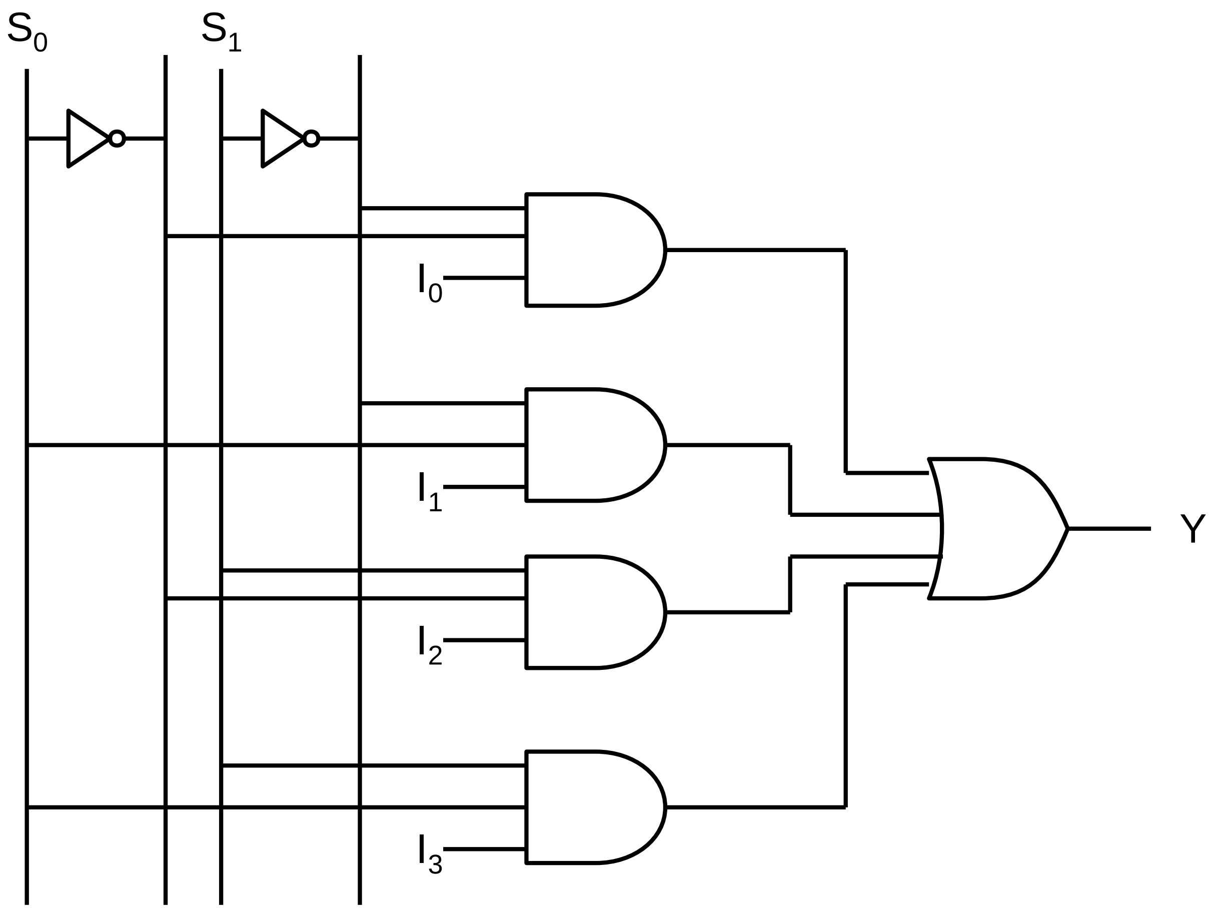
- The shortcut version of truth table is more useful
- => Minimization is more natural

Bigger Multiplexers



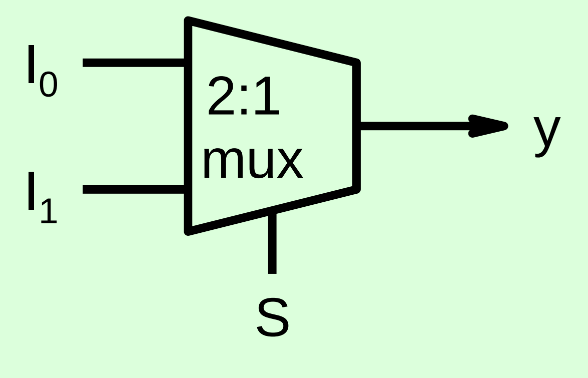
$$y = \overline{S_1}\,\overline{S_0} I_0 + \overline{S_1}\, S_0\, I_1 + S_1 \overline{S_0}\, I_2 + S_1 S_0 I_3$$

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

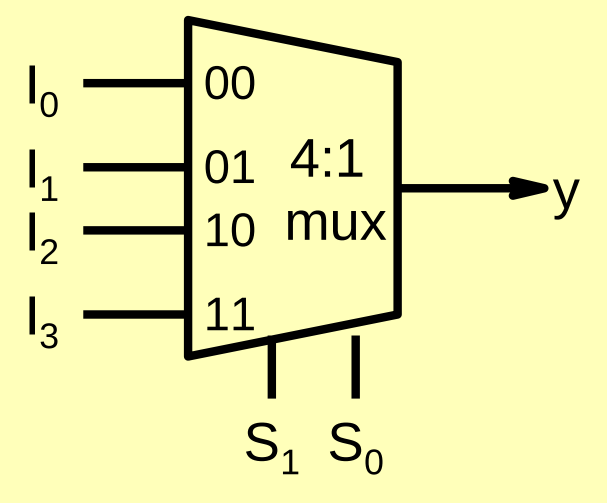


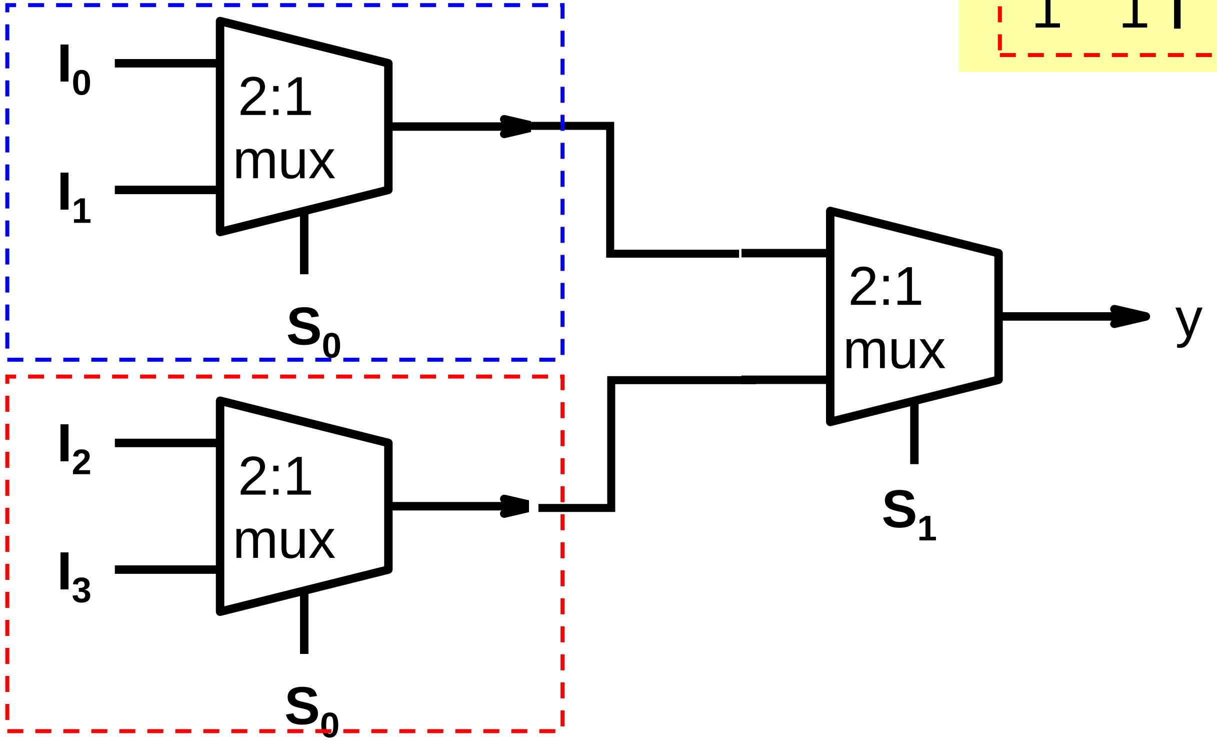
Bigger MUX from Smaller MUX

S	y
0	I_0
1	I_1

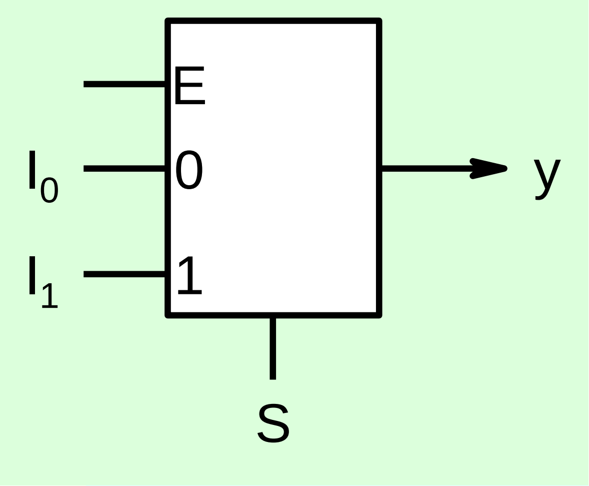


S_1	S_0	y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



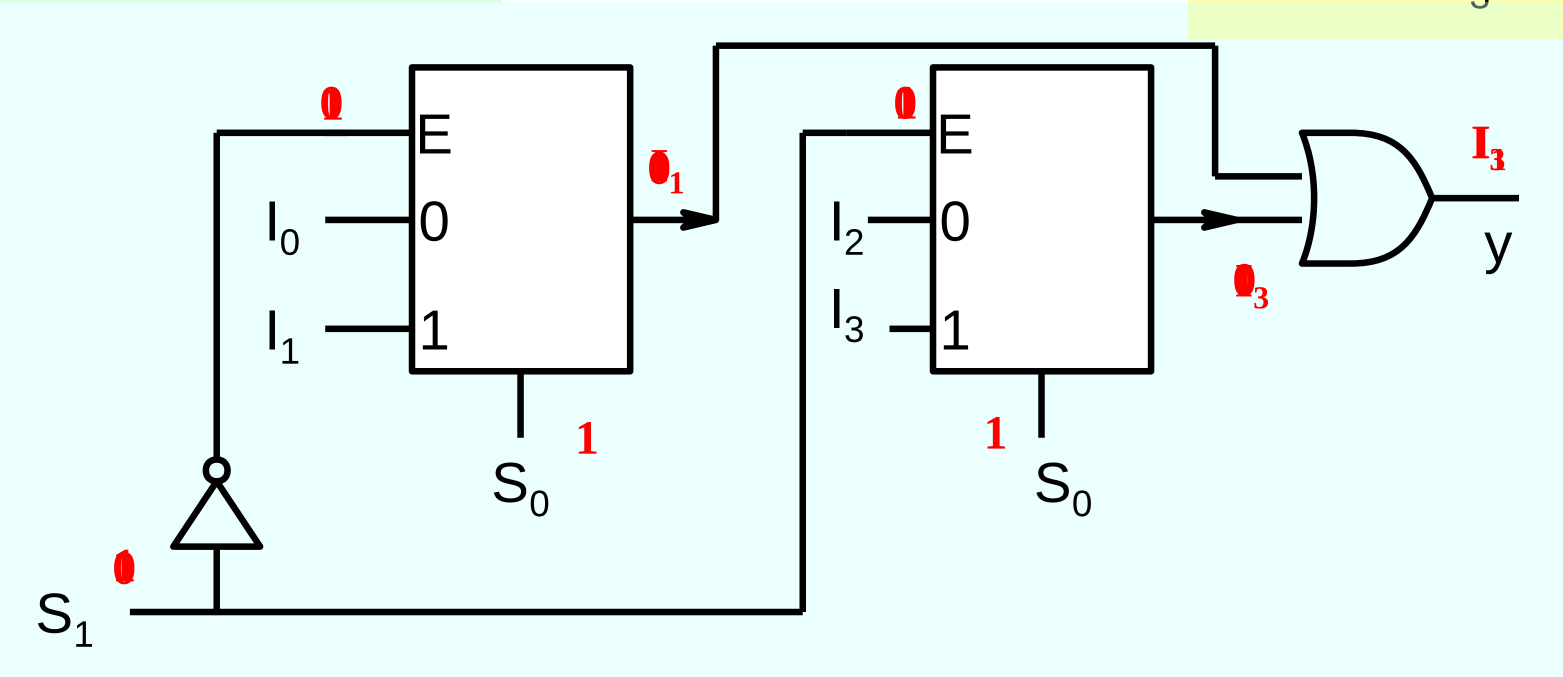


Bigger MUX from Smaller MUX with Enable



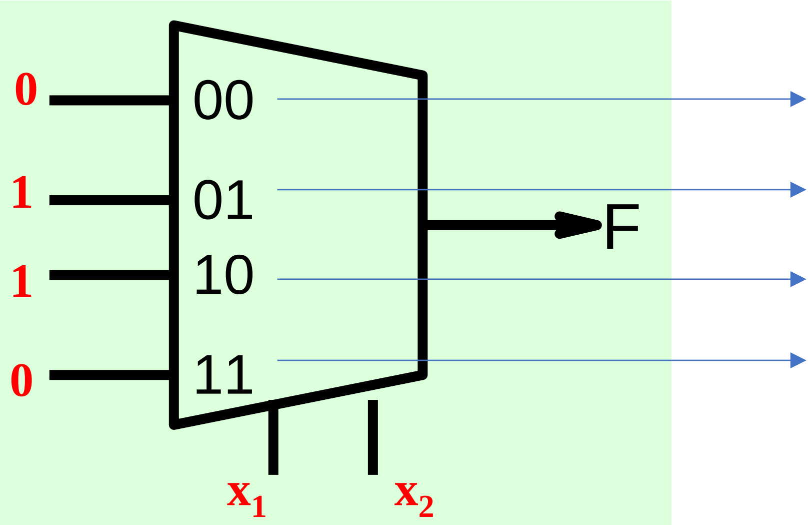
E	S	y
0	x	0
1	0	I_0
1	1	I_1

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



Implementation of a Function using Mux

A 2 variable function can be implemented with a 4:1 mux with 2 select lines: **one-to-one correspondence**



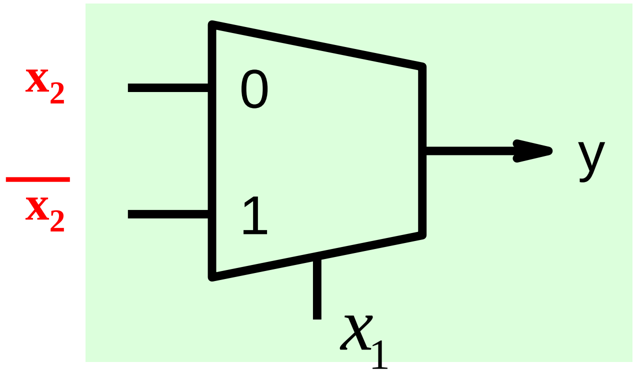
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Can we do better?

Implementation of a Function using Mux

A 2 variable function can be implemented with a 2:1 mux with 1 select line

$$y = x_1\overline{x_2} + \overline{x_1}x_2$$



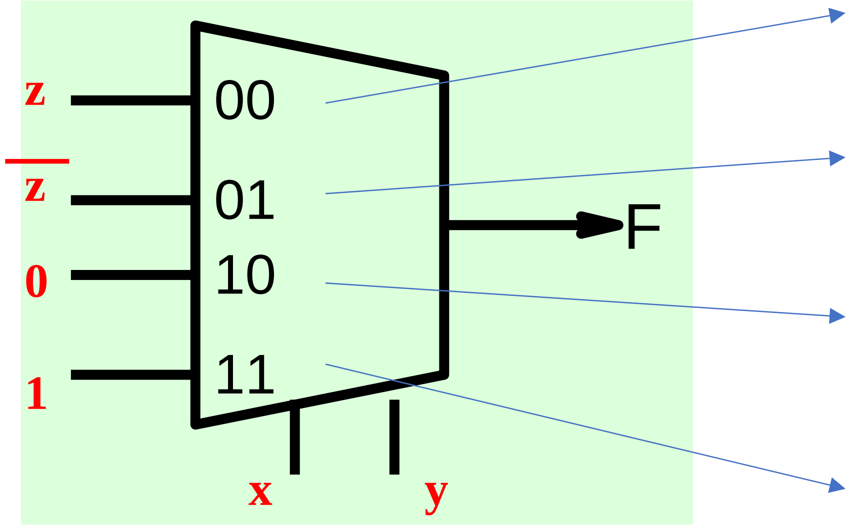
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

$y = x_2$ when $x_1 = 0$

$y = \overline{x_2}$ when $x_1 = 1$

Implementation of a Function using Mux

A 3 variable function can be implemented with a 4:1 mux with 2 select lines



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

F = z when xy = 00

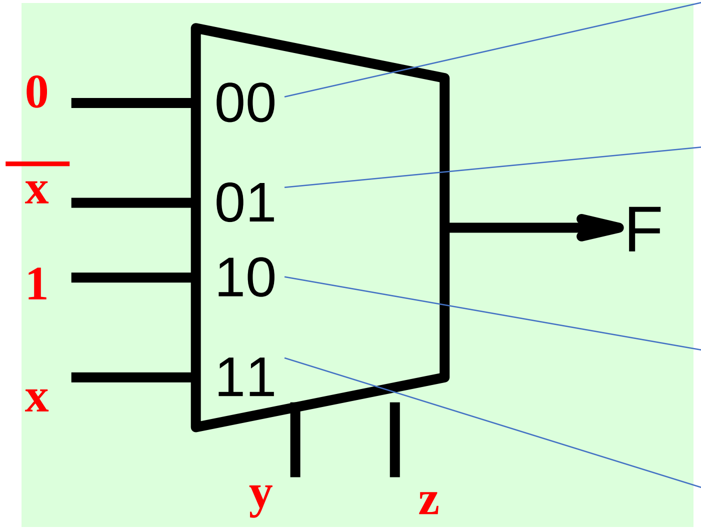
F = $\bar{z}$ when xy =01

F = 0 when xy = 10

F = 1 when xy = 11

Implementation of a Function using Mux

A 3 variable function can be implemented with a 4:1 mux with 2 select lines: **not an unique implementation**



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

F = 0 when yz = 00

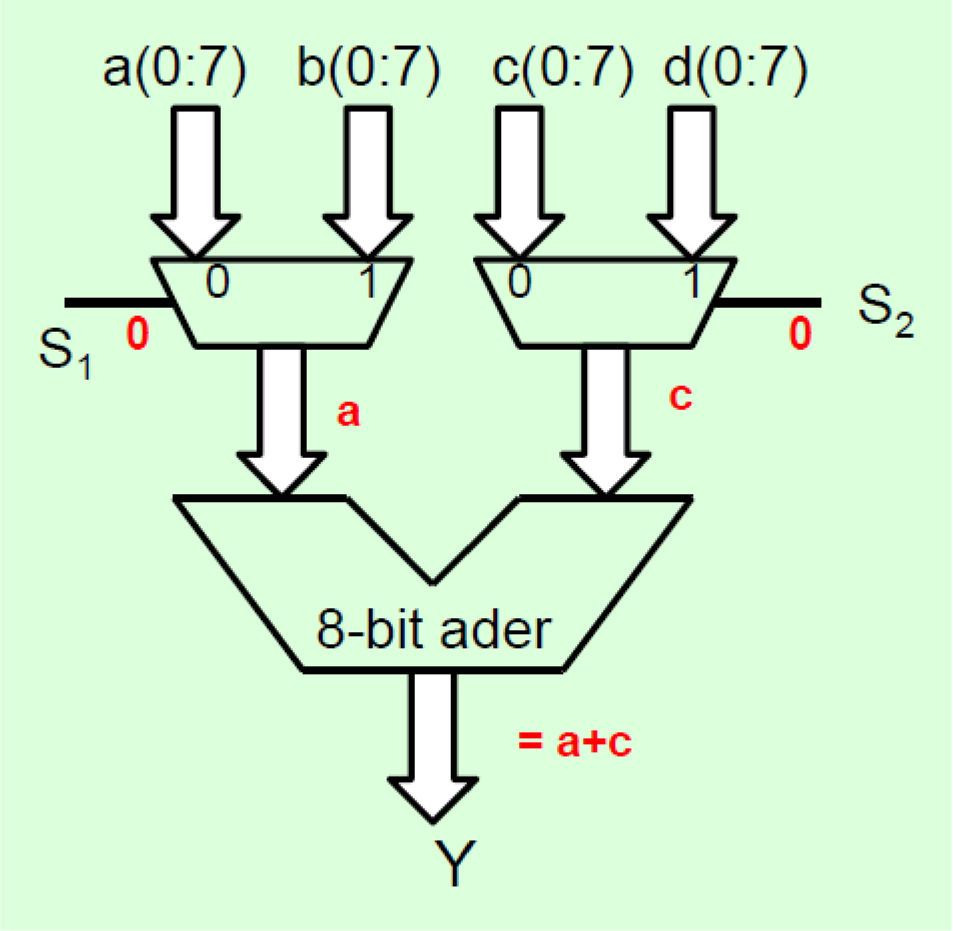
F = $\overline{x}$ when yz =01

F = 1 when yz = 10

F = x when yz = 11

Mux Applications

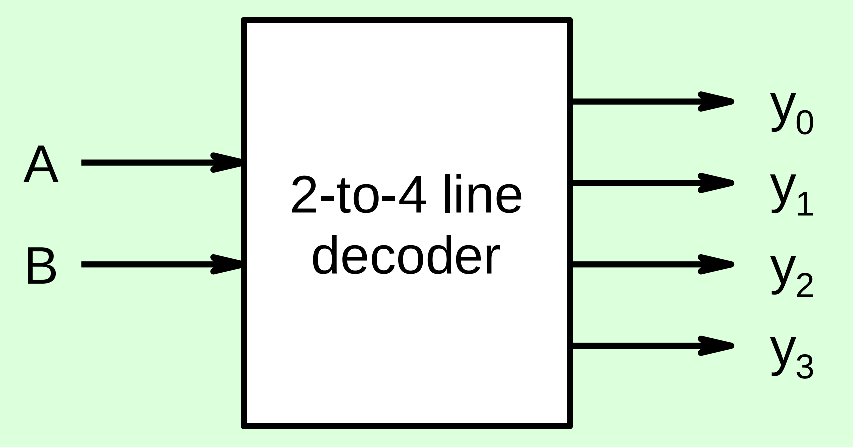
Resource Sharing



S_1	S_0	y =
0	0	a+c
0	1	a+d
1	0	b+c
1	1	b+d

Decoders

- Decodes an encoded information
  - maps a smaller number of inputs to a larger set of outputs



B	A	Y_0	Y_1	Y_2	Y_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

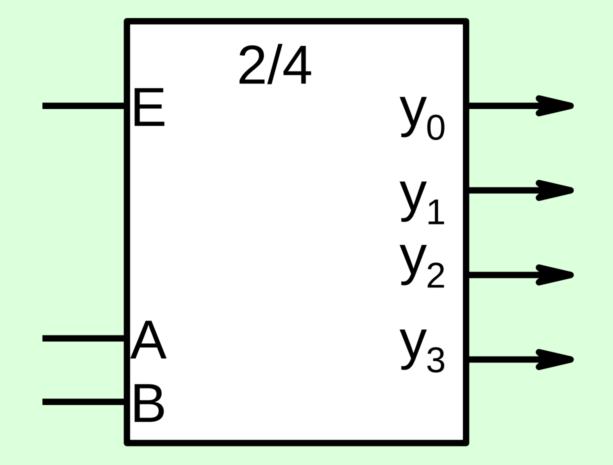
$Y_0 = \overline{B}.\overline{A};$

$Y_1 = \overline{B}.A;$

$Y_2 = B.\overline{A};$

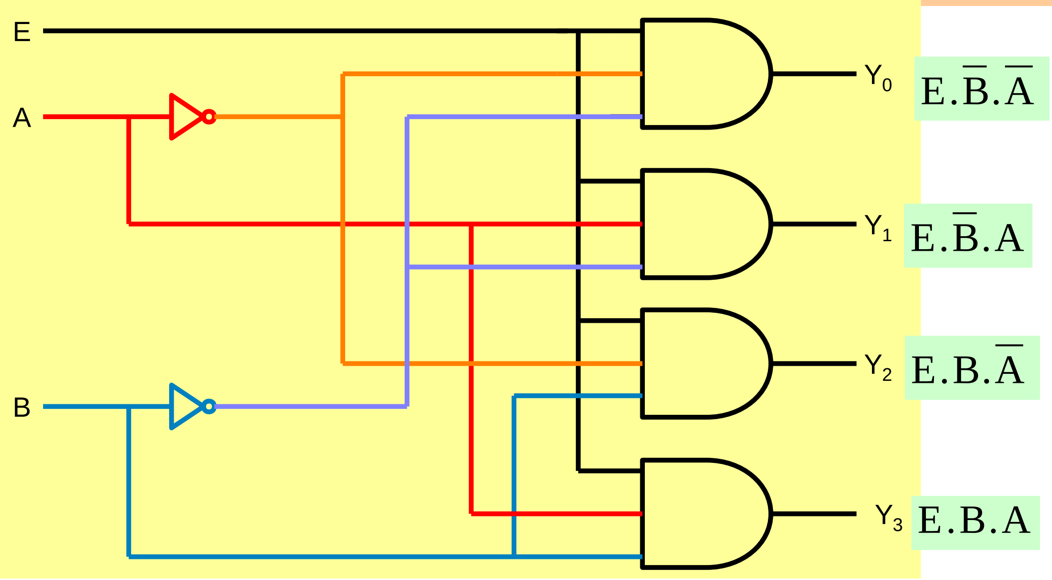
$Y_3 = B.A$

Decoders with ‘Enable’ input

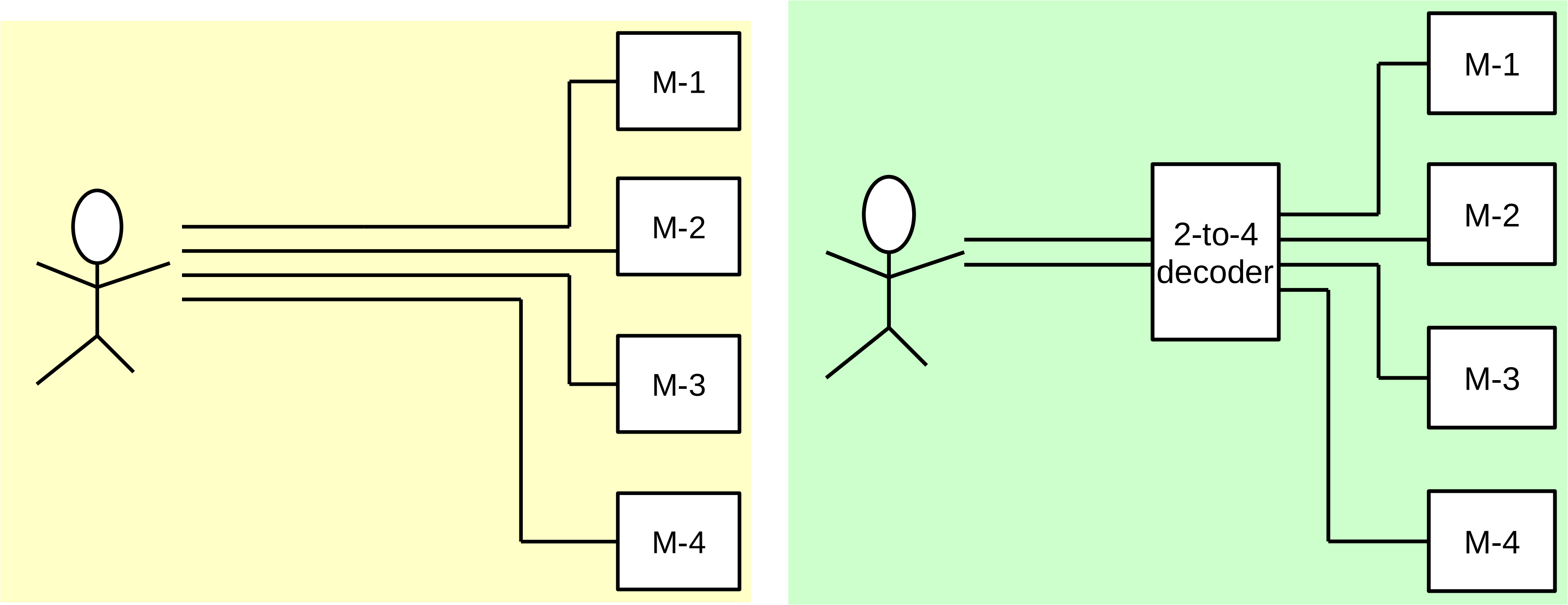


E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$$Y_0 = E.\overline{B}.\overline{A}\ ; Y_1 = E.\overline{B}.A\ ; Y_2 = E.B.\overline{A}\ ; Y_3 = E.B.A$$

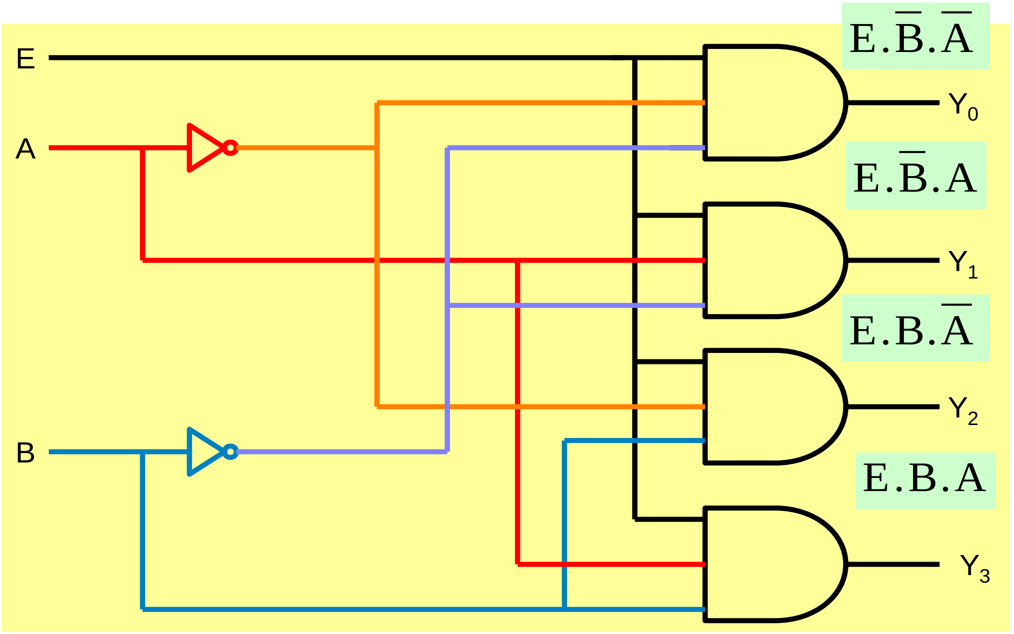


Decoders in Vending Machine

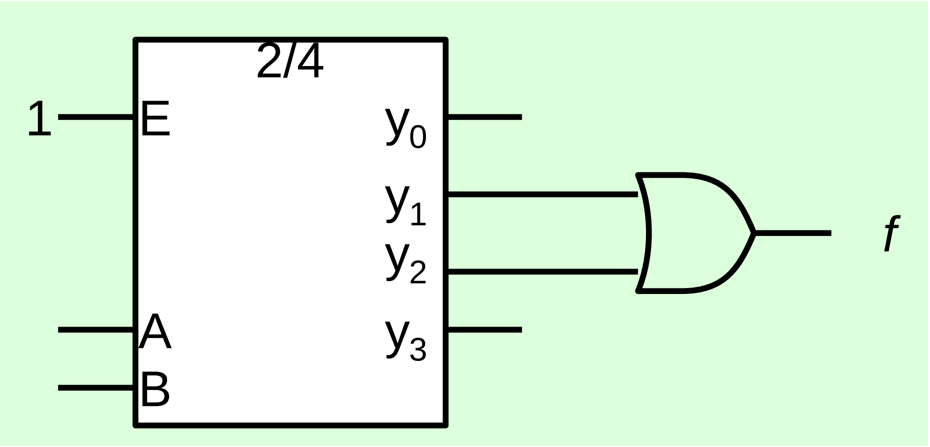


Term Generator for SOP Expression of a Function

x	y	term
0	0	$\bar{x} . \bar{y}$
0	1	$\bar{x} . y$
1	0	$x . \bar{y}$
1	1	$x . y$



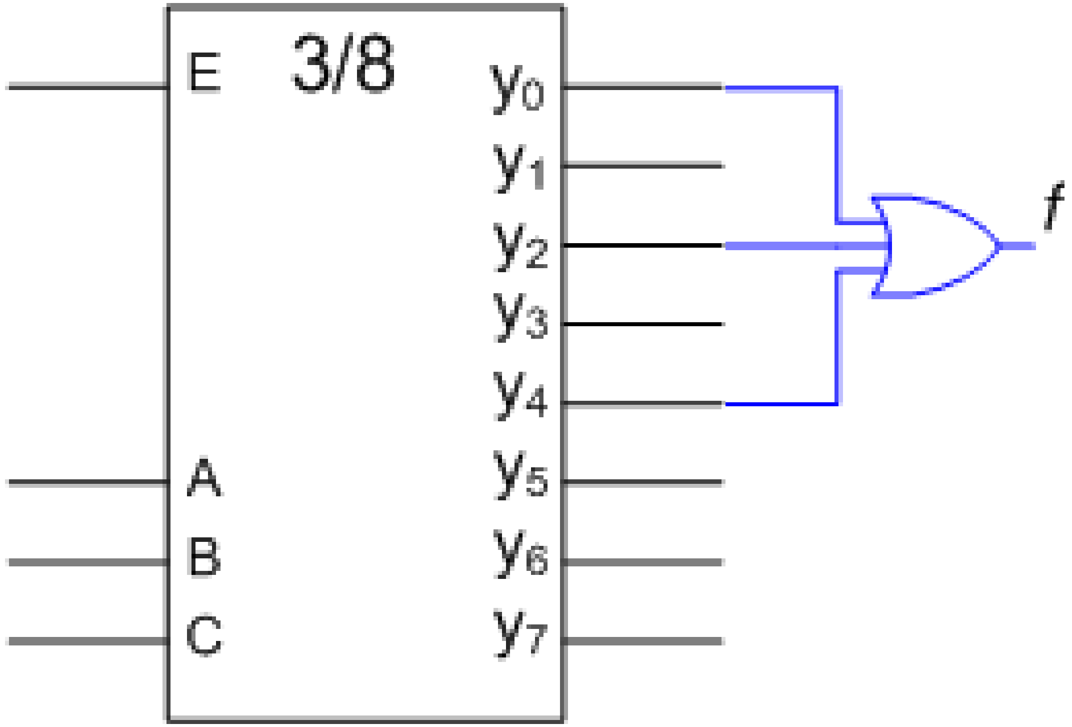
B	A	f_1
0	0	0
0	1	1
1	0	1
1	1	0



Implementation of a Function using Decoders

- Decoder allows a quick implementation
- No minimization, but lots of gates…

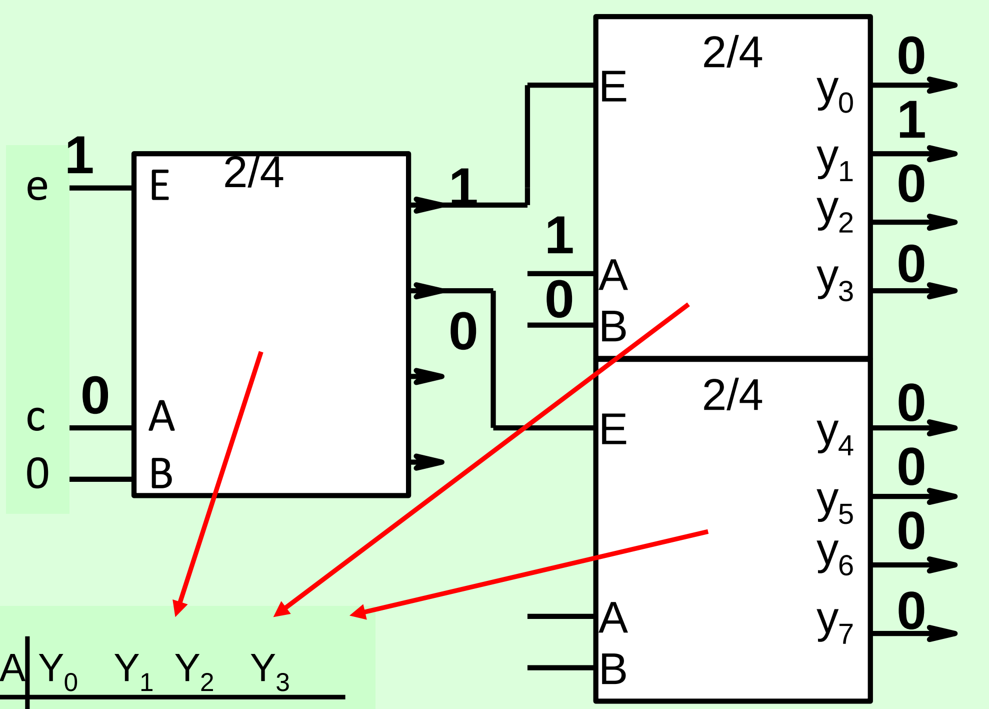
C	B	A	*f*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



Bigger Decoders

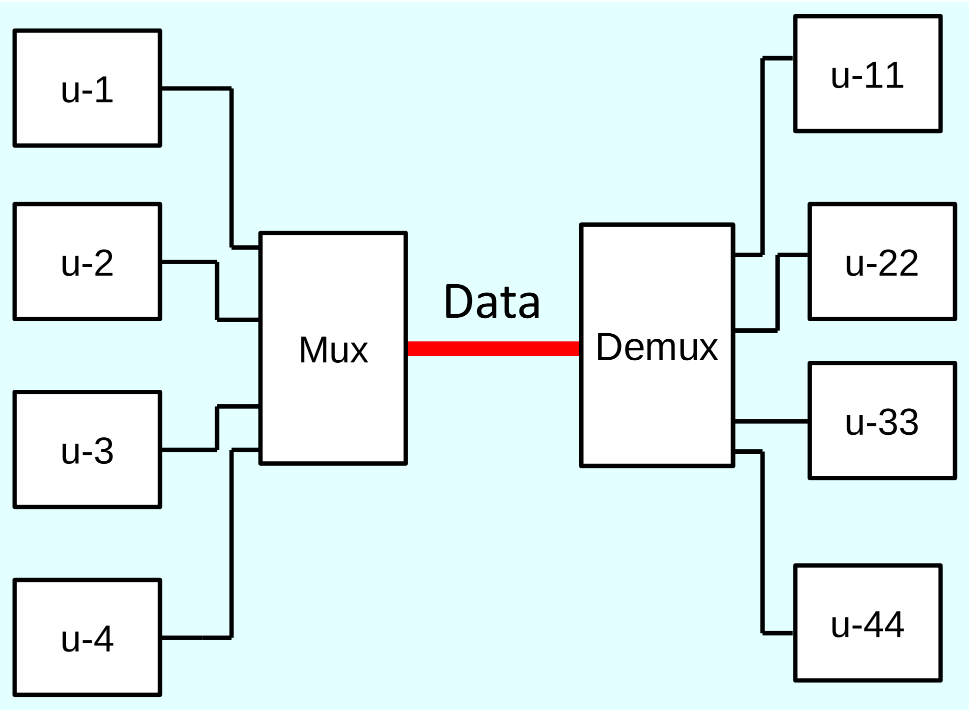
- 3 by 8 decoder using a 2 by 4 decoder

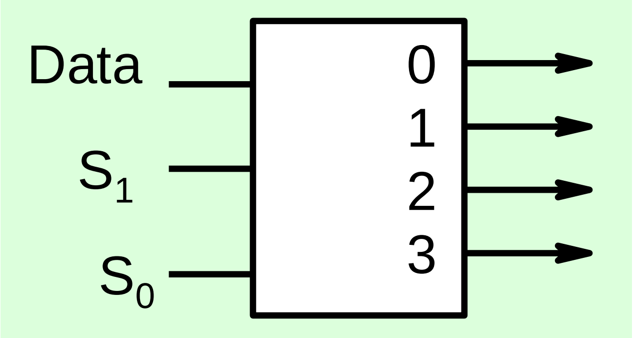
e	c	b	a	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1



E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

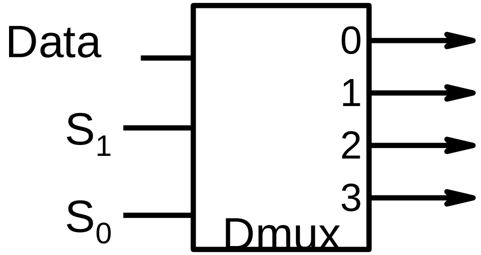
De-Multiplexer

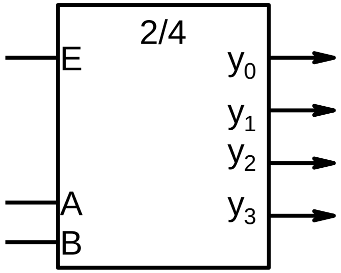




S_1	S_0	y_0	y_1	y_2	y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

De-Mux vs Decoder

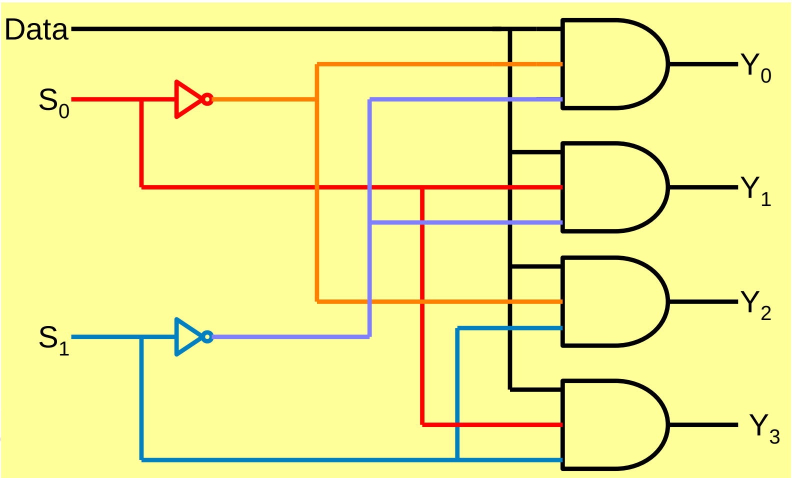


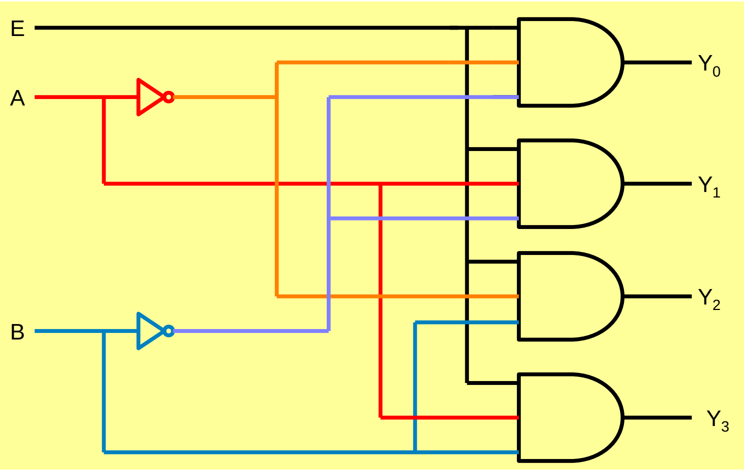


Only difference is in name and application

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

E	B	A	Y_0	Y_1	Y_2	Y_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1


Data
S_0
S_1
Y_0
Y_1
Y_2
Y_3


E
A
B
Y_0
Y_1
Y_2
Y_3