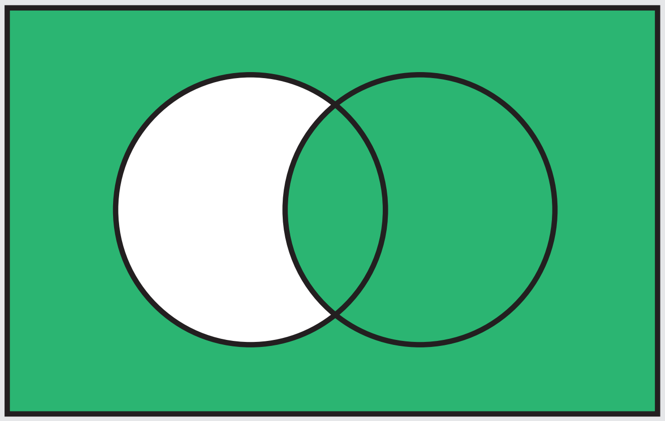
Computer Architecture

Paul Mellies

Lecture 9 : Boolean Logic & Karnaugh Maps

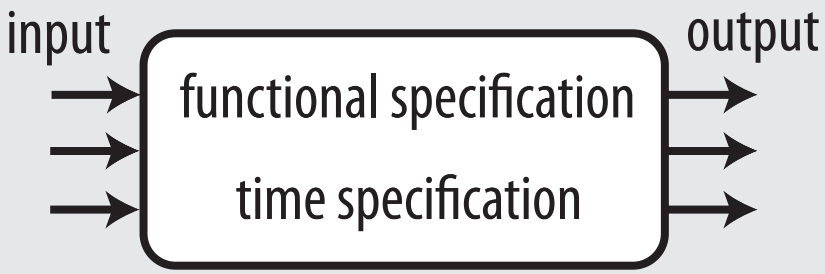


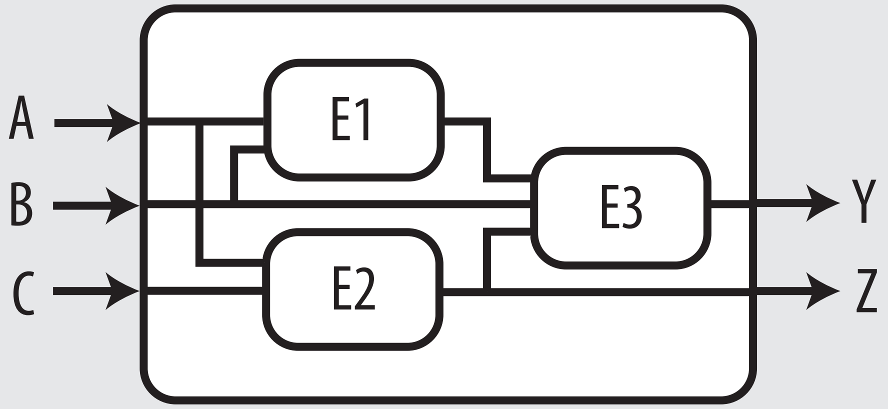
Boolean Logic

Circuit (in digital electronics)

A circuit can be viewed as a black box with

- one or more discrete-valued **input terminals**
- one or more discrete-valued **output terminals**
- a **functional specification** describing the relationship between input and output
- a **timing specification** describing the delay between inputs changing and outputs responding





Peering inside the black box, it is composed of

- nodes — wires classified as input, output and internal
- elements — themselves circuits with inputs, outputs and specifications

Boolean algebra

Boolean algebra is based on three operations :

• conjunction	:	$x \wedge y$
• disjunction	:	$x \vee y$
• negation	:	$\neg x$

and two constants :

• true	:	$\mathbf{1}$
• false	:	$\mathbf{0}$

These operations and constants are required
to satisfy a series of equations.

Boolean algebra

Associativity :

$$(x \wedge y) \wedge z = x \wedge (y \wedge z)$$
$$(x \vee y) \vee z = x \vee (y \vee z)$$

Commutativity :

$$x \wedge y = y \wedge x$$
$$x \vee y = y \vee x$$

Distributivity :

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

Identities :

$$x \wedge 1 = x$$
$$x \vee 0 = x$$

Annihilation :

$$x \wedge 0 = 0$$

Boolean algebra

Idempotence :

$$x \wedge x = x$$
$$x \vee x = x$$

Absorption :

$$x \wedge (x \vee y) = x$$
$$x \vee (x \wedge y) = x$$

Distributivity :

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

Annihilation :

$$x \vee \mathbf{1} = \mathbf{1}$$

Boolean algebra

Complementation :

$$\begin{aligned} x \wedge \neg x &= \mathbf{0} \\ x \vee \neg x &= \mathbf{1} \end{aligned}$$

Double negation :

$$\neg\neg x = x$$

De Morgan duality :

$$\begin{aligned} \neg x \wedge \neg y &= \neg (x \vee y) \\ \neg x \vee \neg y &= \neg (x \wedge y) \end{aligned}$$

$$\begin{aligned} \neg \mathbf{1} &= \mathbf{0} \\ \neg \mathbf{0} &= \mathbf{1} \end{aligned}$$

Boolean algebra

Definition
A boolean algebra is defined as a set **A** equipped with the operations and constants $\wedge \quad \vee \quad \neg \quad \mathtt{0} \quad \mathtt{1}$ satisfying the mentioned equations.

Observation : every boolean algebra is ordered by the order below :

$$\boxed{x \leq y \qquad \Leftrightarrow \qquad x = x \wedge y}$$

In this ordered set **A** :

- the constant $\mathtt{0}$ is the least element
- the constant $\mathtt{1}$ is the greatest element

Recall that an order is a transitive, reflexive and antisymmetric relation.

Boolean algebra

Moreover, the two operations $\wedge$ $\vee$ are monotone in the sense that :

$\forall x,\ x',\ y,\ y' \in$ **A** x **A** x **A** x **A**

$$x \leq x' \text{ and } y \leq y' \quad \Rightarrow \quad x \wedge y \leq x' \wedge y'$$

$$x \leq x' \text{ and } y \leq y' \quad \Rightarrow \quad x \vee y \leq x' \vee y'$$

Accordingly, the negation $\neg$ is anti-monotone in the sense that :

$\forall x,\ y \in$ **A** x **A**

$$x \leq y \quad \Rightarrow \quad \neg\, y \leq \neg\, x$$

Example of boolean algebra

The set $\mathbf{A} = \{ \textbf{true} , \textbf{false} \}$ defines a boolean algebra with :

$$\begin{aligned} \textbf{true} \wedge \textbf{true} &= \textbf{true} \\ \textbf{true} \wedge \textbf{false} &= \textbf{false} \\ \textbf{false} \wedge \textbf{false} &= \textbf{false} \end{aligned} \qquad \begin{aligned} \textbf{true} \vee \textbf{true} &= \textbf{true} \\ \textbf{true} \vee \textbf{false} &= \textbf{true} \\ \textbf{false} \vee \textbf{false} &= \textbf{false} \end{aligned}$$

$$\neg\ \textbf{true} = \textbf{false} \qquad \neg\ \textbf{false} = \textbf{true}$$

$$1 = \textbf{true} \qquad 0 = \textbf{false}$$

Observe that in this boolean algebra :

$$\textbf{false} = \textbf{false} \wedge \textbf{true}$$

and thus :

$$\textbf{false} \leq \textbf{true}$$

Example of boolean algebra

Here, we suppose given a set $\mathbf{U}$.

The set $\mathbf{A}$ whose elements are the subsets of $\mathbf{U}$ is a boolean algebra with conjunction, disjunction and negation defined as follows :

$$\mathbf{X} \wedge \mathbf{Y} = \mathbf{X} \cap \mathbf{Y} \qquad \mathbf{X} \vee \mathbf{Y} = \mathbf{X} \cup \mathbf{Y}$$

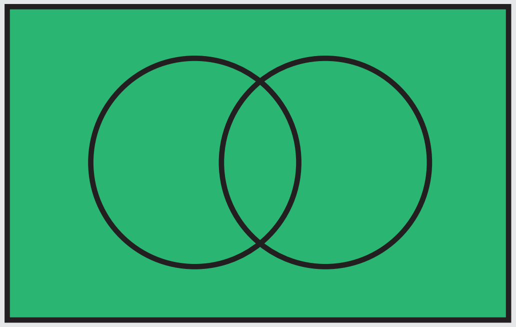
$$\neg\, \mathbf{X} = \mathbf{U} \setminus \mathbf{X}$$

$$1 = \mathbf{U} \qquad 0 = \emptyset$$

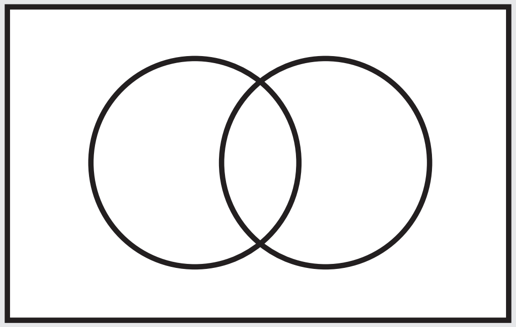
Observe that one recovers the boolean algebra { true , false } when $\mathbf{U}$ is defined as the singleton set $\mathbf{U} = \{*\}$.

$$\mathbf{true} = \{*\} \qquad \mathbf{false} = \emptyset$$

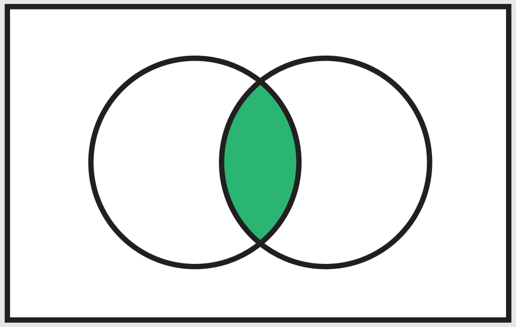
Venn diagrams



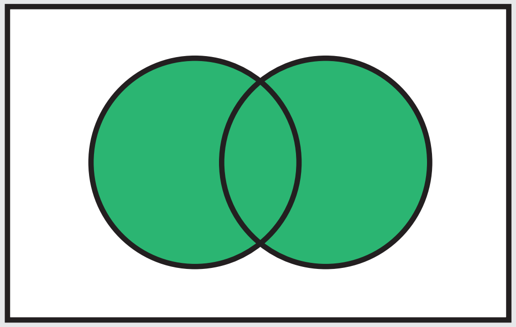
$1 = U$



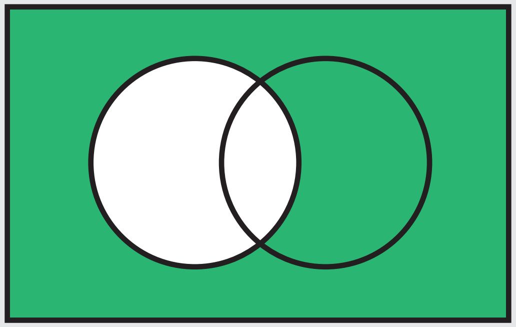
$0 = \emptyset$



$X \wedge Y = X \cap Y$



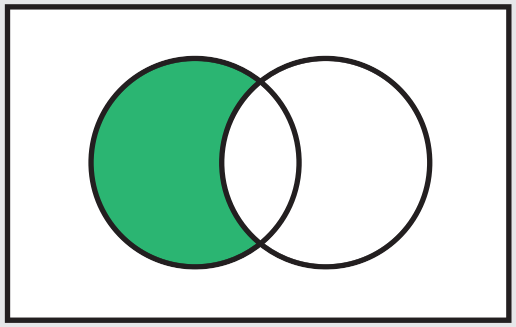
$X \vee Y = X \cup Y$



$\neg X = U \setminus X$

Venn diagrams
(derived constructions)

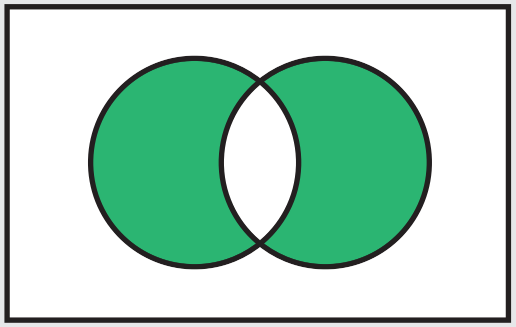
Difference



$$X \setminus Y = X \wedge \neg Y$$

Symmetric difference

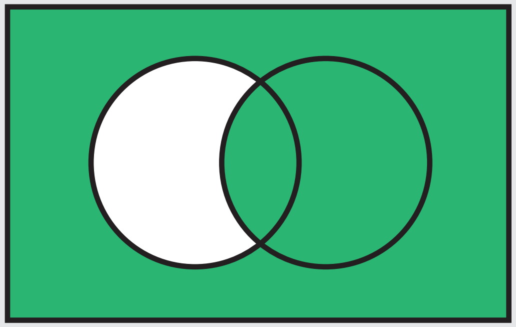
= exclusive or



$$X \triangle Y = (X \vee Y) \setminus (X \wedge Y)$$
$$= (X \vee Y) \wedge \neg (X \wedge Y)$$

Venn diagrams
(derived constructions)

Implication



$$X \rightarrow Y \;=\; \neg\,(X \wedge \neg Y)$$

$$=\; \neg X \vee Y$$

Key property of implication :

$$\boxed{X \leq Y \rightarrow Z \quad \Leftrightarrow \quad X \wedge Y \leq Z}$$

Product-of-sums form

Every boolean expression may be transformed modulo the equations of boolean algebras into a conjunction of disjunctions :

$$\begin{aligned} \textbf{prodofsum} &= 1 \mid \textbf{sum} \mid \textbf{prodofsum} \wedge \textbf{prodofsum} \\ \textbf{sum} &= 0 \mid \textbf{literal} \mid \textbf{sum} \vee \textbf{sum} \\ \textbf{literal} &= \textbf{atom} \mid \neg\, \textbf{atom} \end{aligned}$$

Same grammar formulated this time with the arithmetic notation :

$$\begin{aligned} \textbf{prodofsum} &= 1 \mid \textbf{sum} \mid \textbf{prodofsum} \times \textbf{prodofsum} \\ \textbf{sum} &= 0 \mid \textbf{literal} \mid \textbf{sum} + \textbf{sum} \\ \textbf{literal} &= \textbf{atom} \mid \overline{\textbf{atom}} \end{aligned}$$

Sum-of-products form

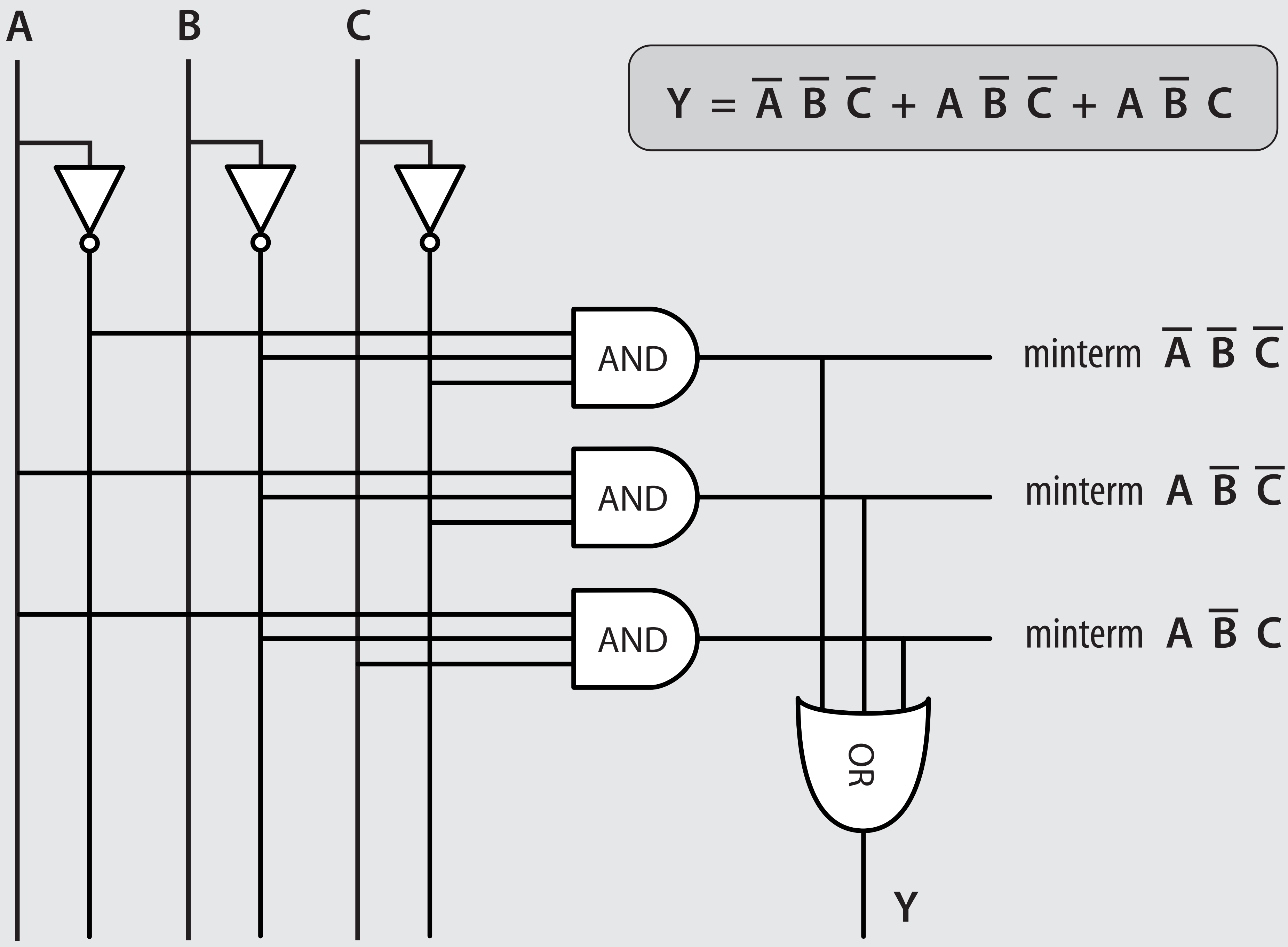
Every boolean expression may be transformed modulo the equations of boolean algebras into a disjunction of conjunctions :

$$
\begin{array}{rclllll}
\textbf{sumofprod} & = & 0 & | & \textbf{prod} & | & \textbf{sumofprod} \vee \textbf{sumofprod} \\
\textbf{prod} & = & 1 & | & \textbf{literal} & | & \textbf{prod} \wedge \textbf{prod} \\
\textbf{literal} & = & \textbf{atom} & | & \neg\, \textbf{atom} & &
\end{array}
$$

Same grammar formulated this time with the arithmetic notation :

$$
\begin{array}{rclllll}
\textbf{sumofprod} & = & 0 & | & \textbf{prod} & | & \textbf{sumofprod} + \textbf{sumofprod} \\
\textbf{prod} & = & 1 & | & \textbf{literal} & | & \textbf{prod}\ \textbf{x}\ \textbf{prod} \\
\textbf{literal} & = & \textbf{atom} & | & \overline{\textbf{atom}} & &
\end{array}
$$

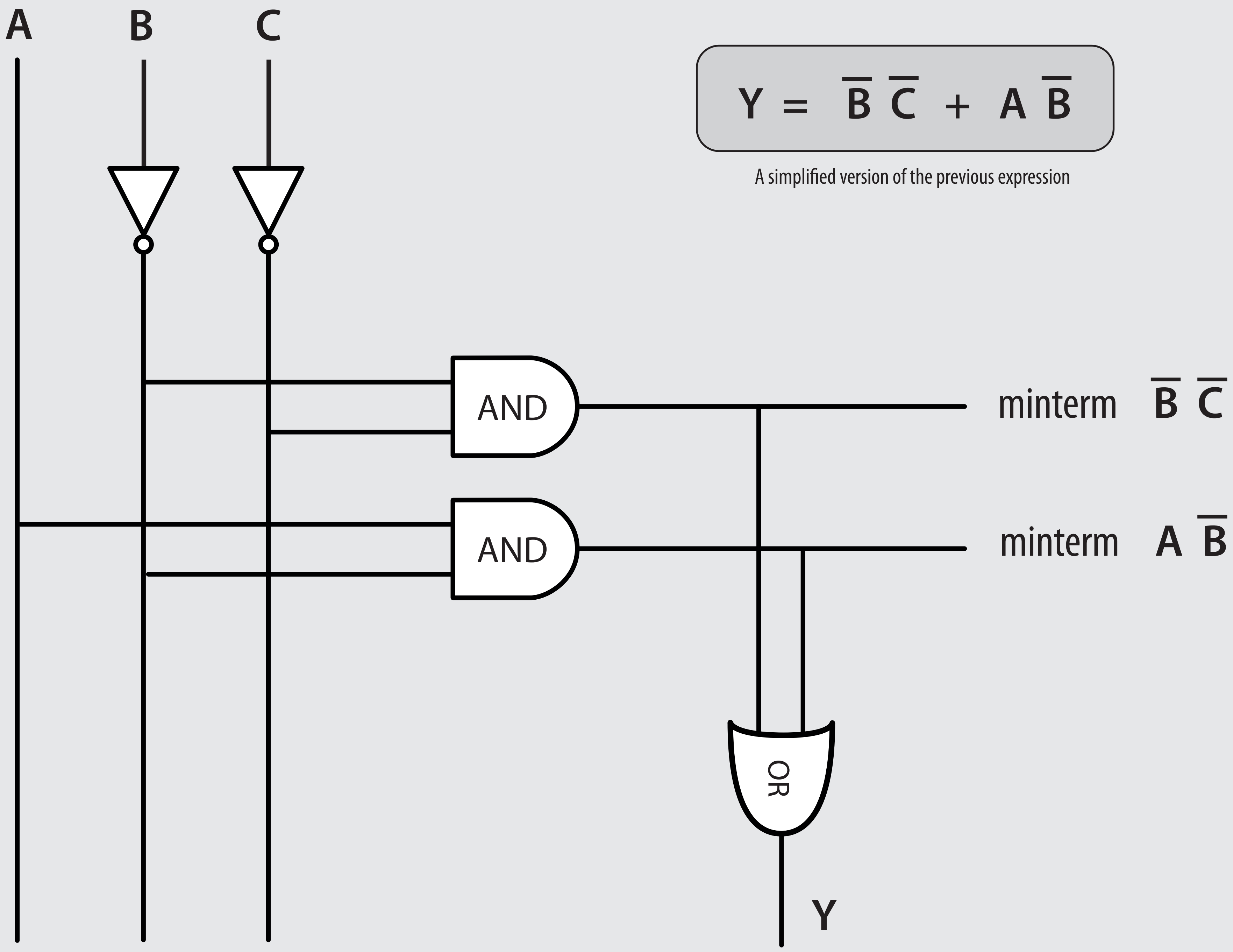
Sum of products in schematics


A
B
C
$Y = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C$
AND
AND
AND
minterm $\overline{A}\,\overline{B}\,\overline{C}$
minterm $A\,\overline{B}\,\overline{C}$
minterm $A\,\overline{B}\,C$
OR
Y

Sum of products in schematics

$$Y = \overline{B}\,\overline{C} + A\,\overline{B}$$

A simplified version of the previous expression



A
B
C
AND
AND
minterm $\overline{B}\,\overline{C}$
minterm $A\,\overline{B}$
OR
Y

Exercise

Show that the equality

$$\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C = \overline{B}\,\overline{C} + A\,\overline{B}$$

follows from the equations of boolean algebra.

Solution

Show that the equality

$$\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C = \overline{B}\,\overline{C} + A\,\overline{B}$$

follows from the equations of boolean algebra.

$$\begin{aligned}\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} &= (A + \overline{A})\,\overline{B}\,\overline{C} && \text{distributivity}\\ &= 1\,\overline{B}\,\overline{C} && \text{complementation}\\ &= \overline{B}\,\overline{C} && \text{identity}\end{aligned}$$

$$\begin{aligned}A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C &= A\,\overline{B}\,(C + \overline{C}) && \text{distributivity}\\ &= A\,\overline{B}\,1 && \text{complementation}\\ &= A\,\overline{B} && \text{identity}\end{aligned}$$

Solution

Show that the equality

$$\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C = \overline{B}\,\overline{C} + A\,\overline{B}$$

follows from the equations of boolean algebra.

$$\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,\overline{B}\,C$$

by idempotence and associativity

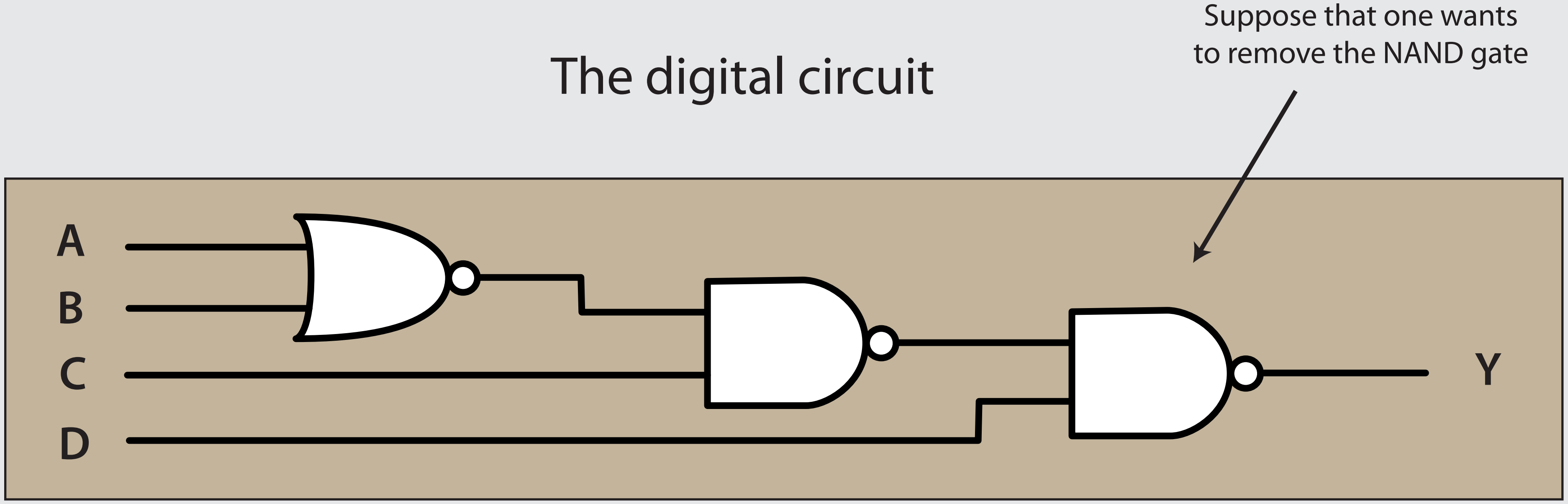
$$= \overline{B}\,\overline{C} + A\,\overline{B}$$

by the two equations of the previous page

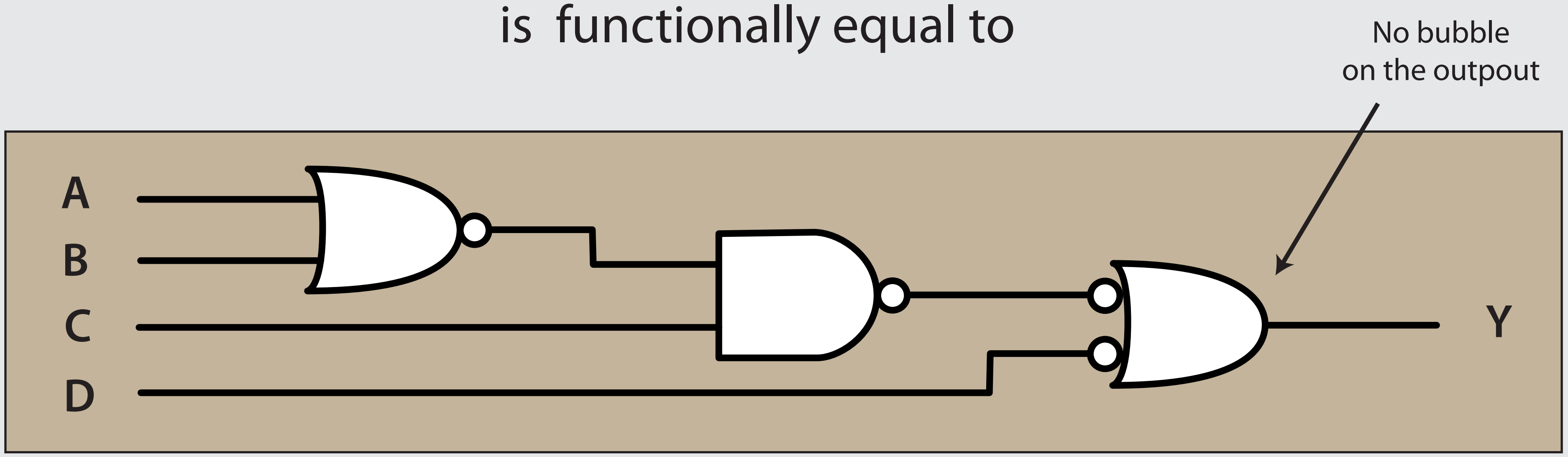
This establishes the expected equation.

Bubble Pushing

The digital circuit

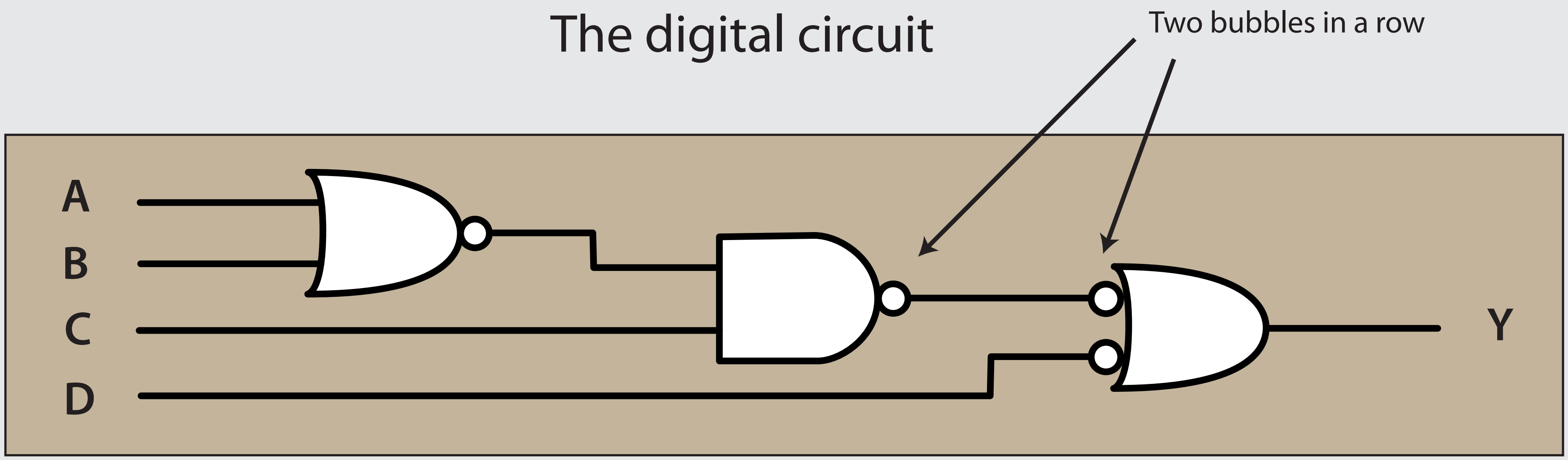


is functionally equal to

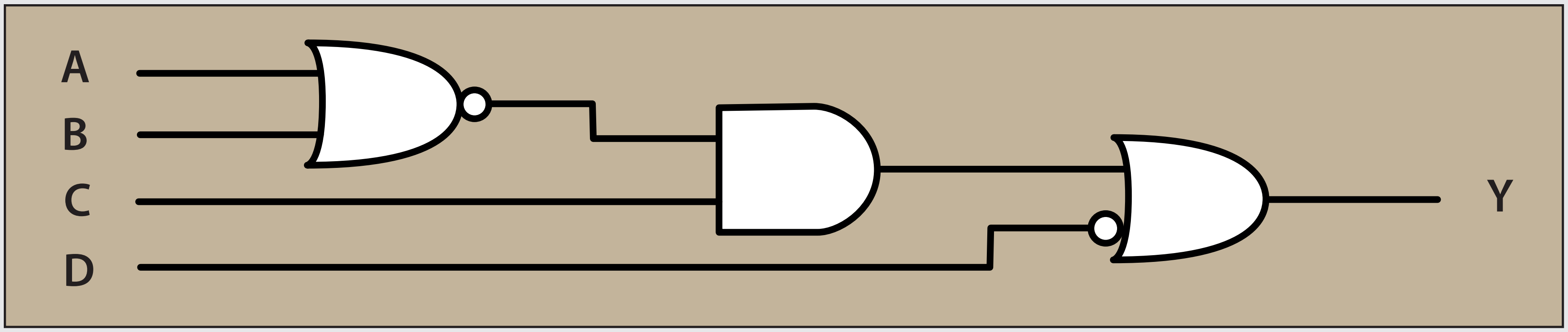


Bubble Pushing

The digital circuit

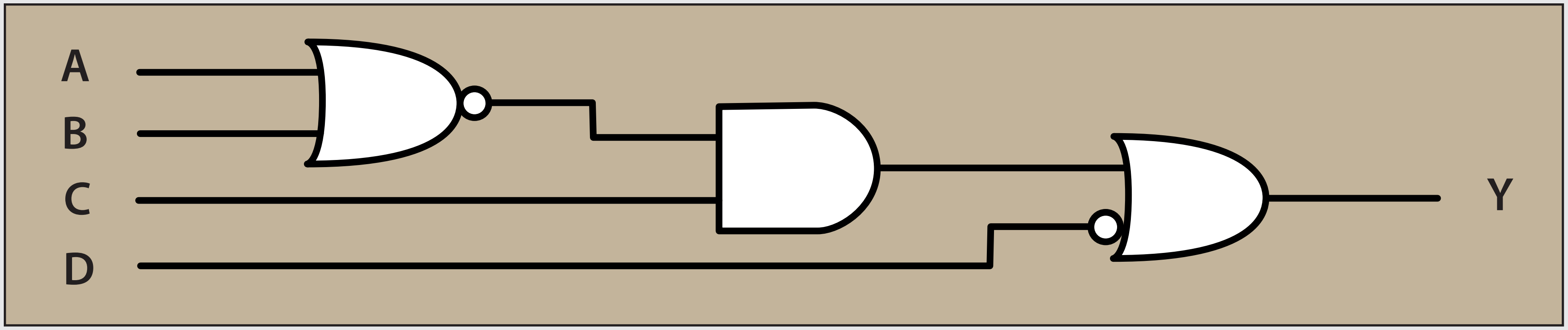


is functionally equal to



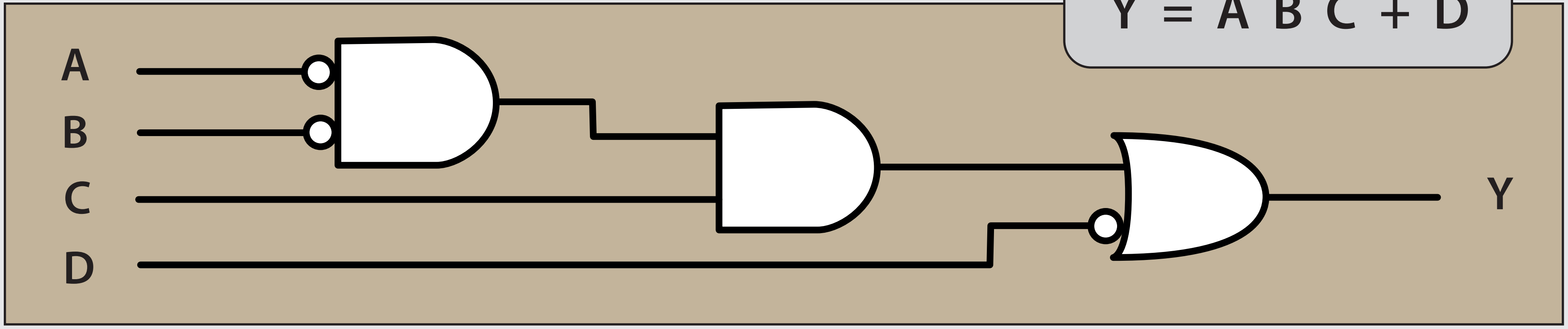
Bubble Pushing

The digital circuit



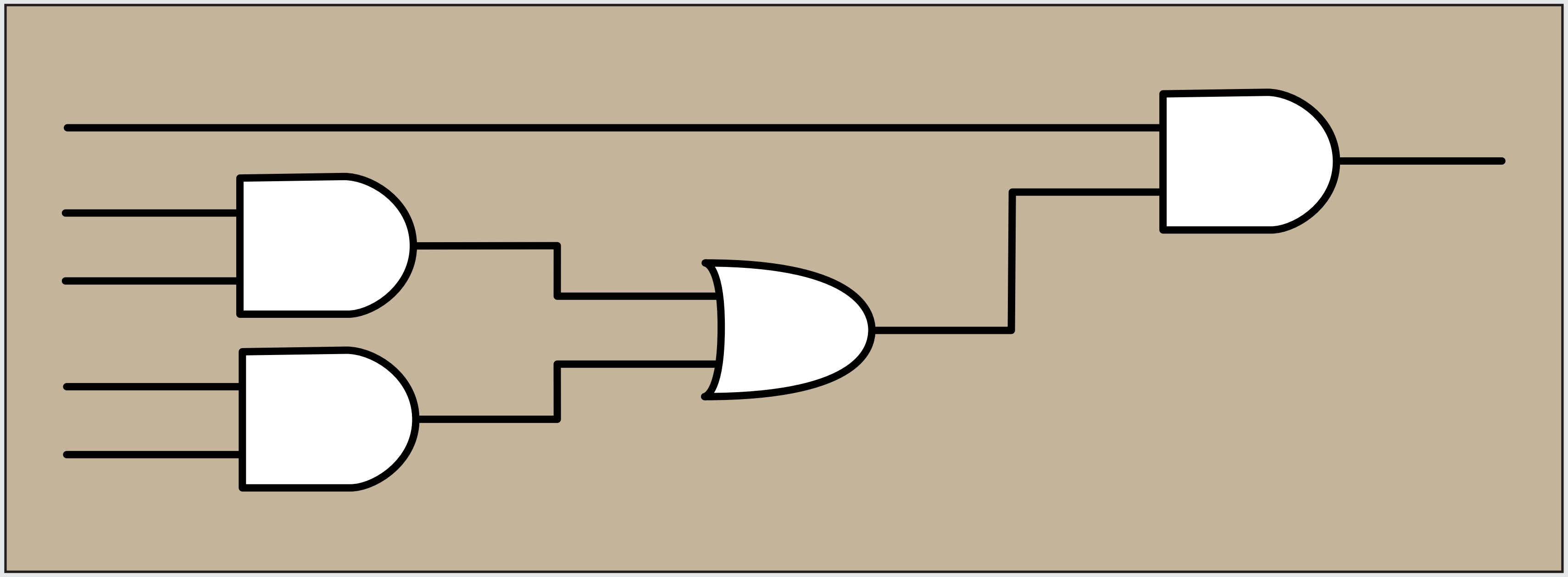
is functionally equal to

$$Y = \overline{A}\,\overline{B}\,C + \overline{D}$$



Bubble Pushing for CMOS logic

In some situations, one would like to transform a logical circuit expressed with AND and OR gates into an equivalent logical circuit constructed with NAND gates, NOR gates and inverters.



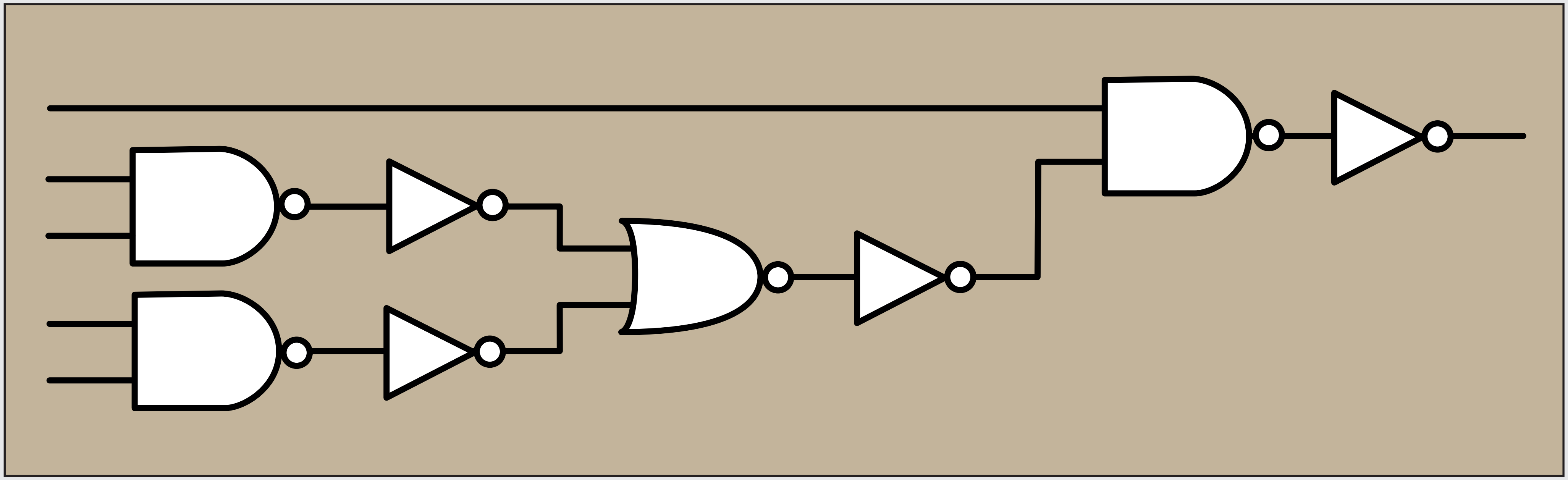
One typical reason is that NAND gates, NOR gates and inverters are easier to construct in CMOS technology.

Bubble Pushing for CMOS logic

One brutal way to achieve the translation is to replace

- every AND gate by a NAND gate followed by an inverter
- every OR gate by a NOR gate followed by an inverter

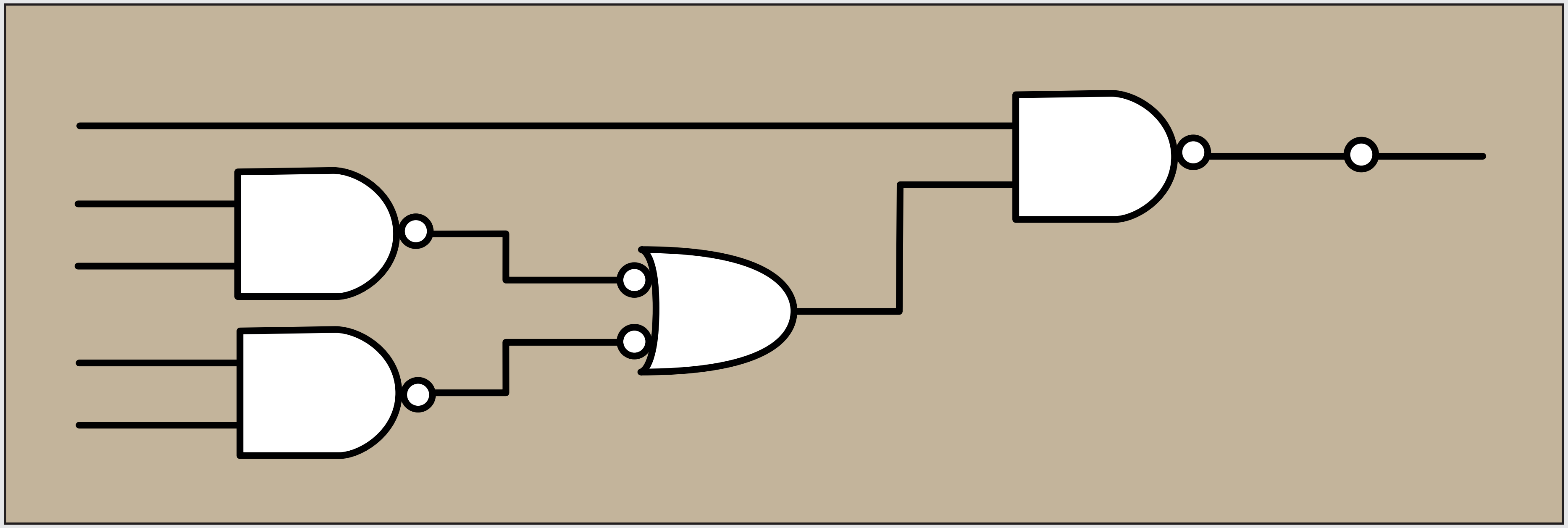
in the way done below :



This procedure works but is far from optimal in general in the number of logical gates in the final CMOS circuit.

Bubble Pushing for CMOS logic

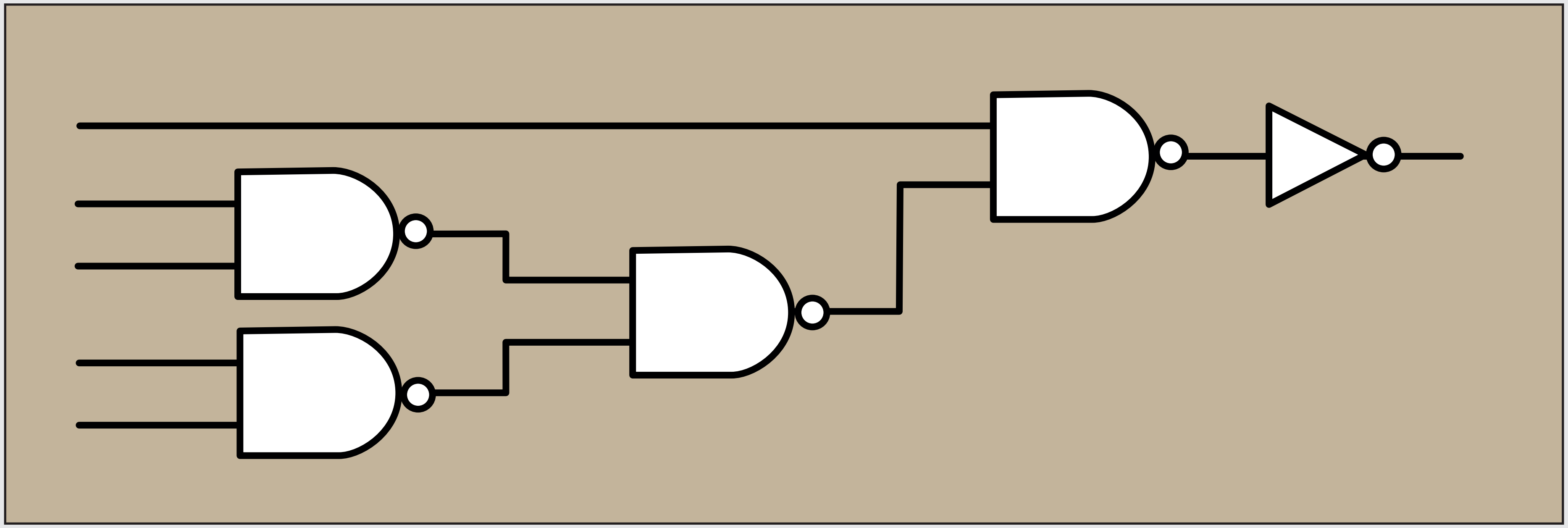
A more sophisticated way to achieve the translation is to add two negations on some of the wires of the original logical circuit :



Note that each negation is represented here by a bubble.

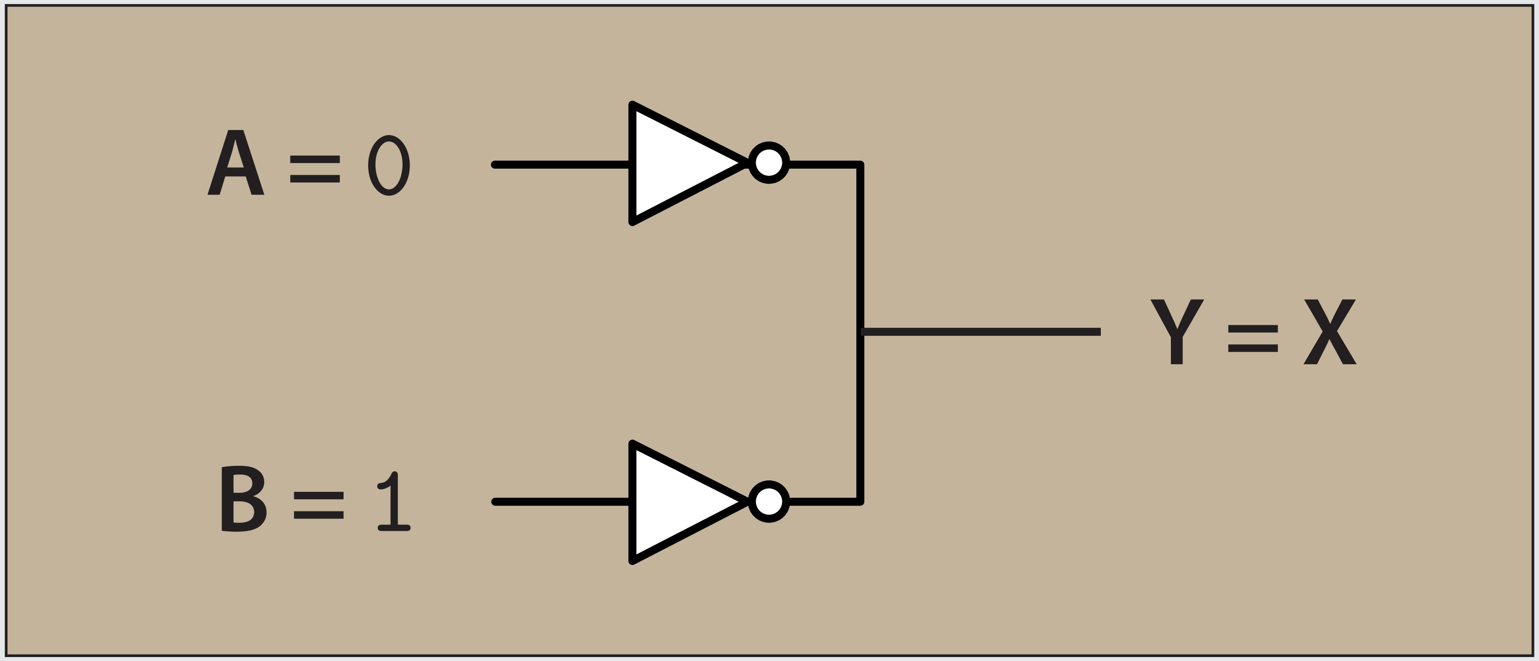
Bubble Pushing for CMOS logic

A more sophisticated way to achieve the translation is to add two negations on some of the wires of the original logical circuit :



One obtains in this way a CMOS circuit with five logical gates (instead of six gates as in the previous translation)

The illegal value X

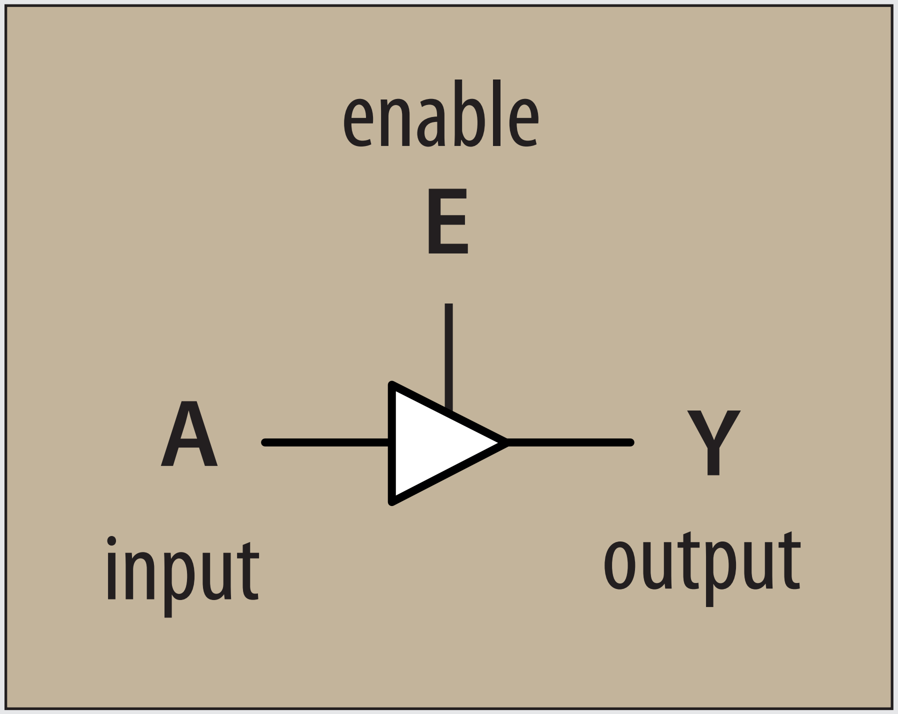


Here, the output Y has an unknown or illegal value.

The situation is called a « contention » and is considered as an error which should be avoided.

The reason is that contention may cause large amounts of power to be dissipated between the logical gates, resulting in the circuit getting hot and possibly damaged.

The floating value Z



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

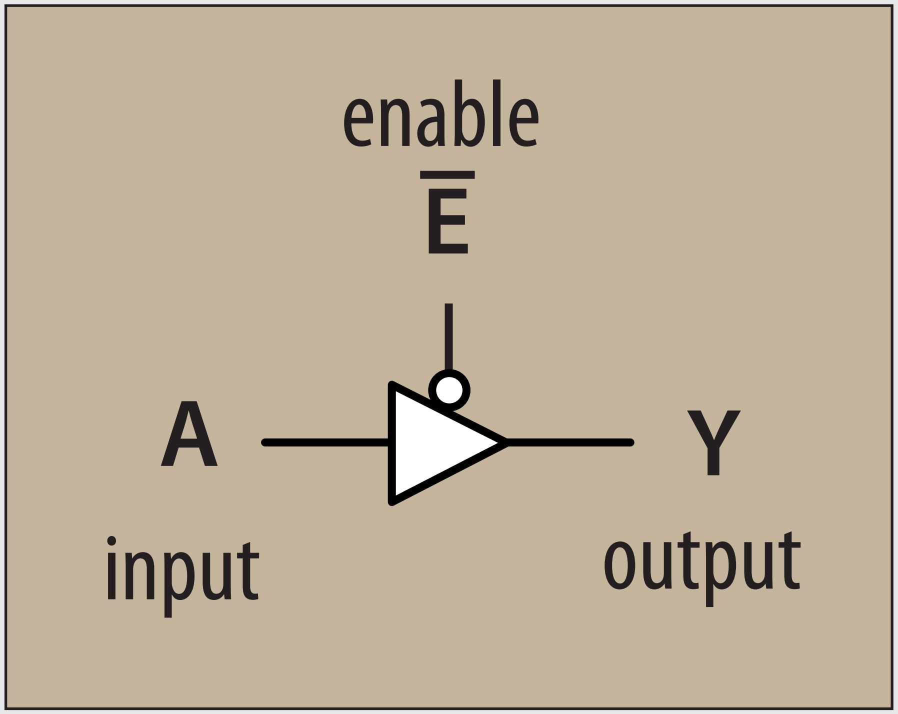
active high enable

The tristate buffer has three possible output values :

- HIGH (1)
- LOW (0)
- FLOATING (Z)

The output has a floating value means that the wire **Y** may receive any voltage depending on the context.

The floating value Z



$\overline{E}$	A	Y
0	0	0
0	1	1
1	0	Z
1	1	Z

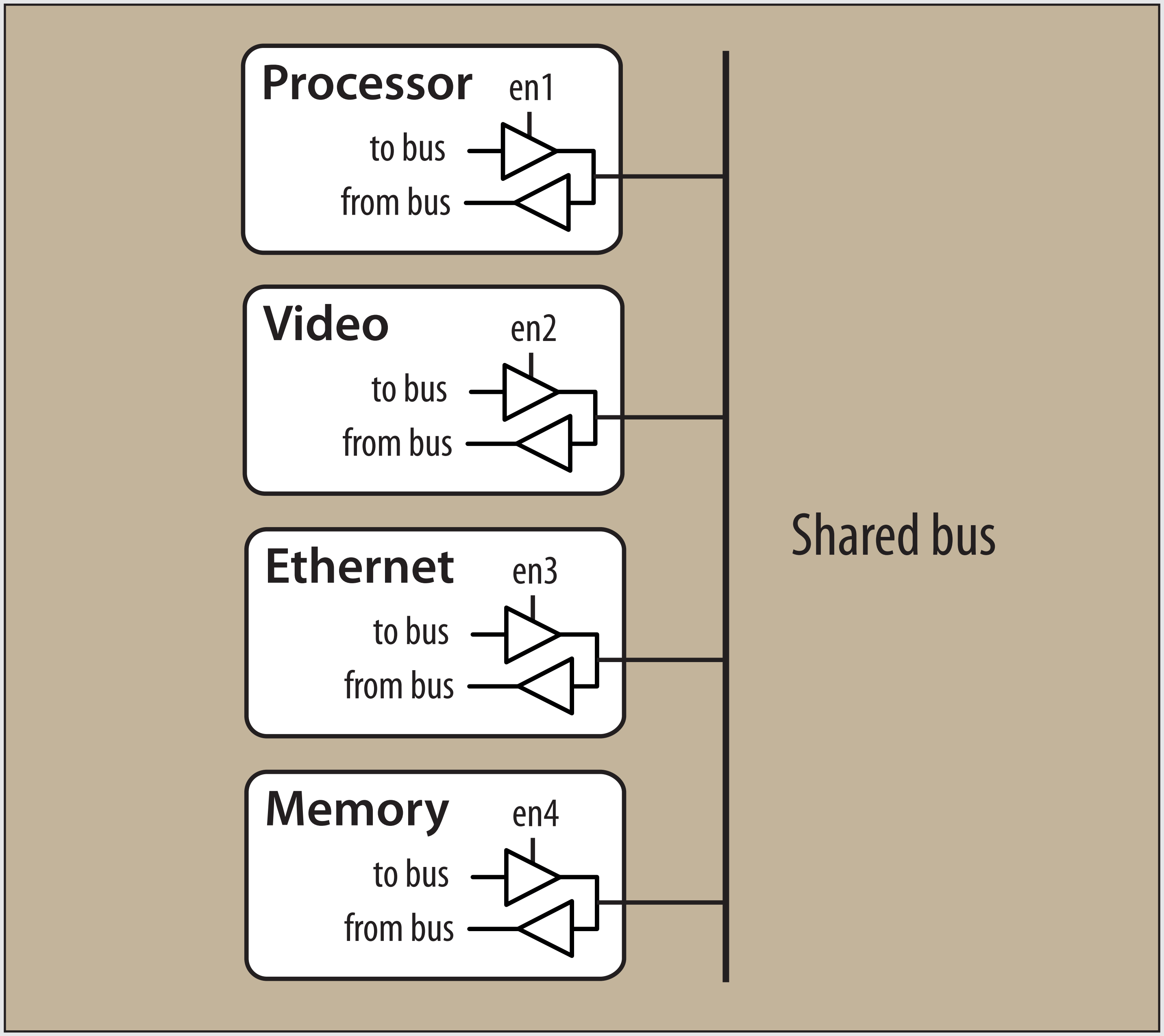
active low enable

The tristate buffer has three possible output values :

• HIGH	(1)
• LOW	(0)
• FLOATING	(Z)

The output has a floating value means that the wire **Y** may receive any voltage depending on the context.

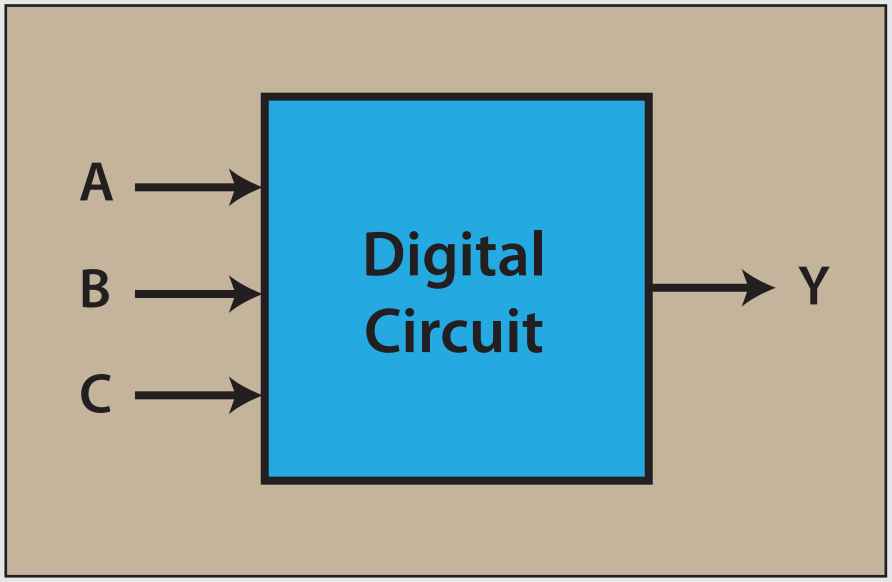
Tristate bus connecting multiple chips



Today, higher speeds are achieved by point-to-point links between chips

Karnaugh Maps

A typical truth value table



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

From the truth table to the K-map

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

Y C \ AB	00	01	11	10
0	1	0	0	0
1	1	0	0	0

Note the clever use of Gray codes here

Same K-map with associated minterms

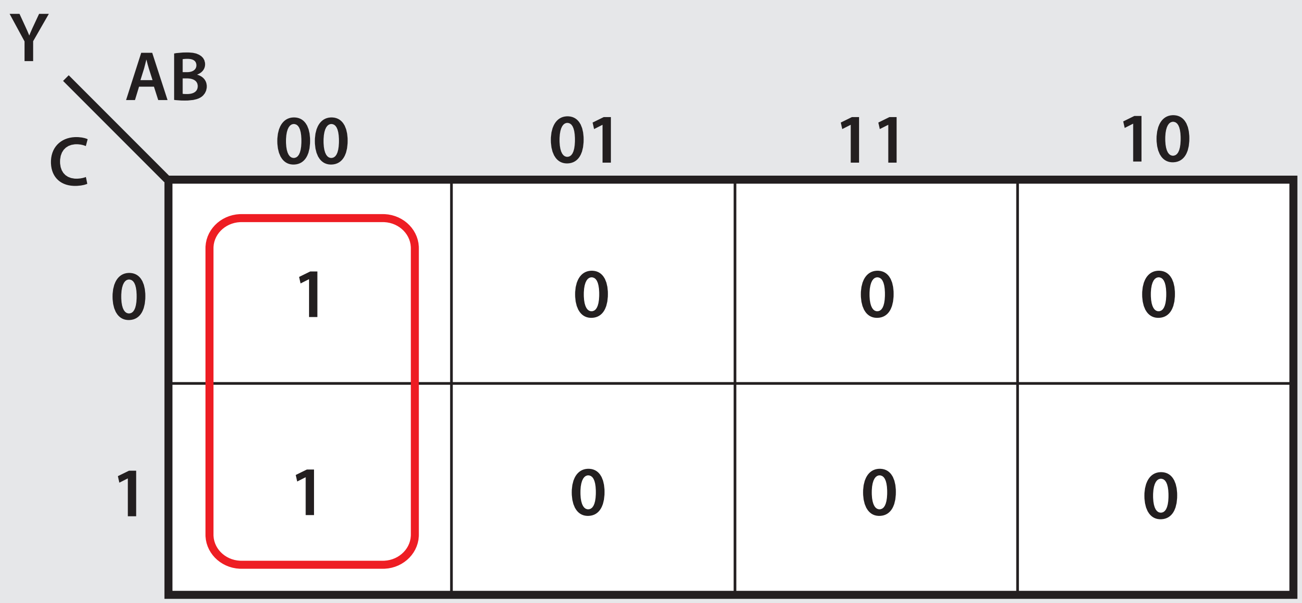
Y: C \ AB	00	01	11	10
0	1	0	0	0
1	1	0	0	0

Y: C \ AB	00	01	11	10
0	$\overline{A}\,\overline{B}\,\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\,\overline{C}$
1	$\overline{A}\,\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$

From this, one deduces that :

$$\boxed{Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C}$$

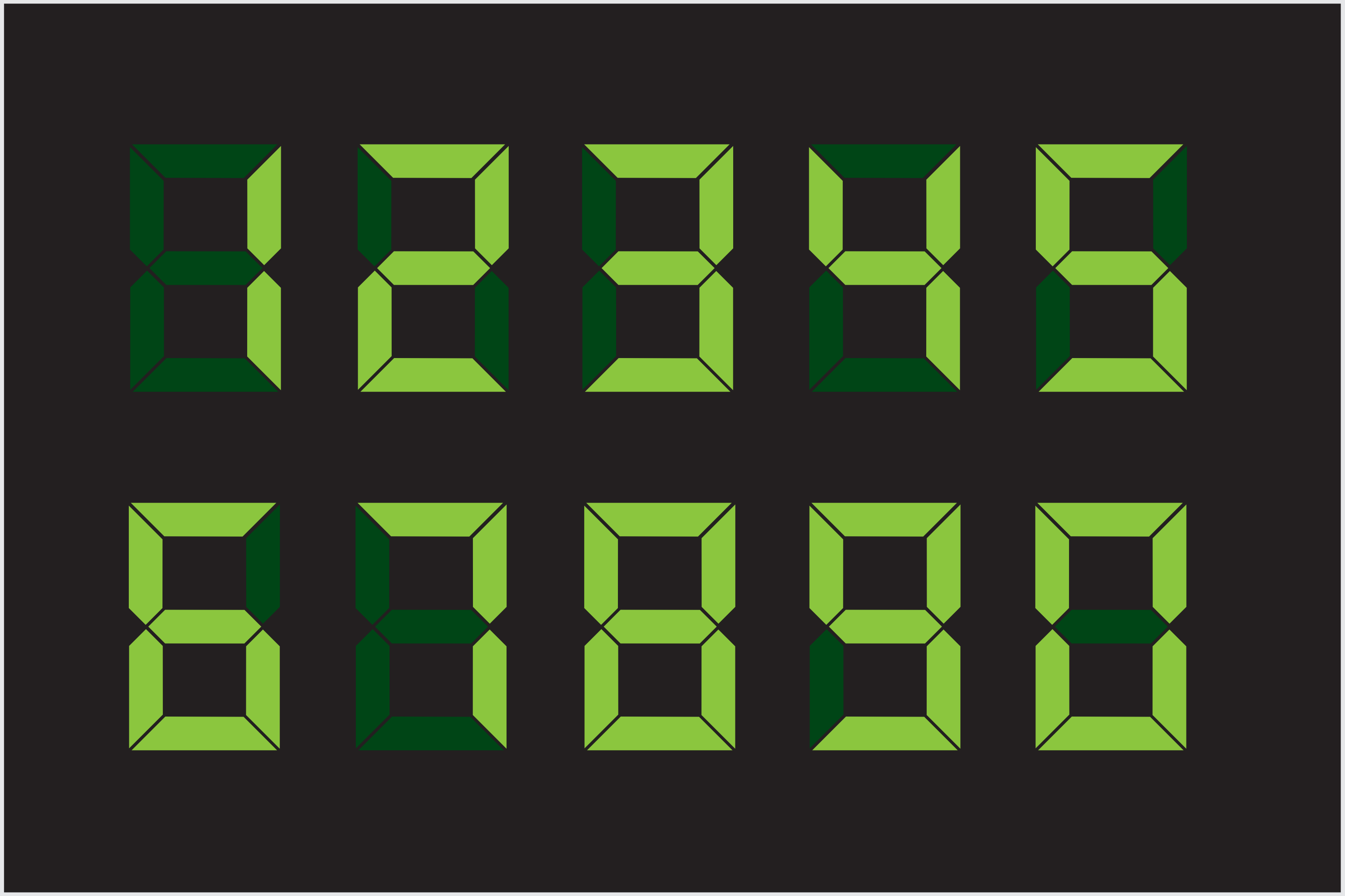
K-map minimisation



By circling the 1's in adjacent squares, one deduces that :

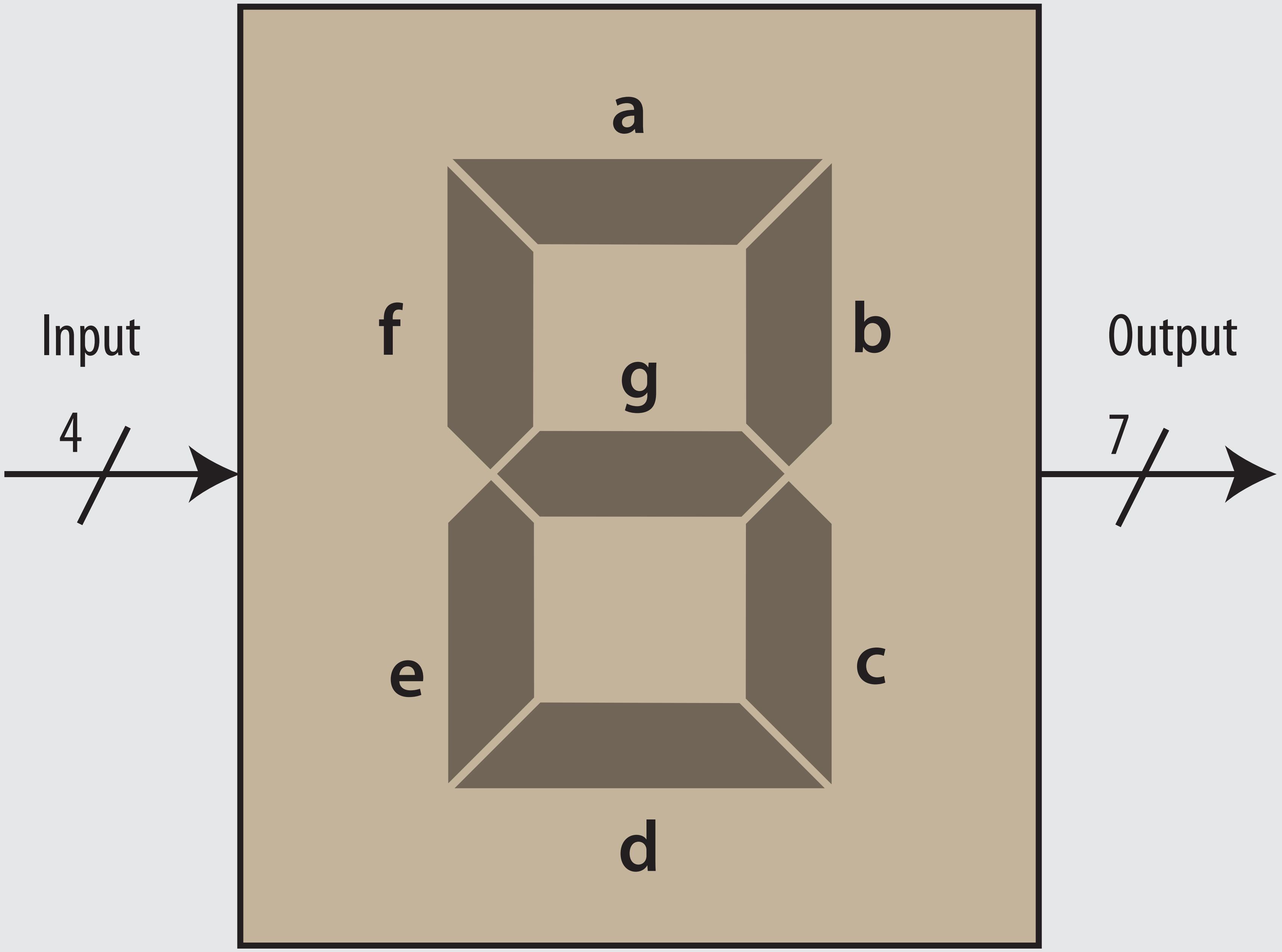
$$\boxed{Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C = \bar{A}\bar{B}}$$

Seven-segment display digits



Seven-segment display digits were invented at the beginning of the 20th century. They became very popular in the 1970’s with the advent of LED.

Seven-segment display decoder



Seven-segment display decoder truth table

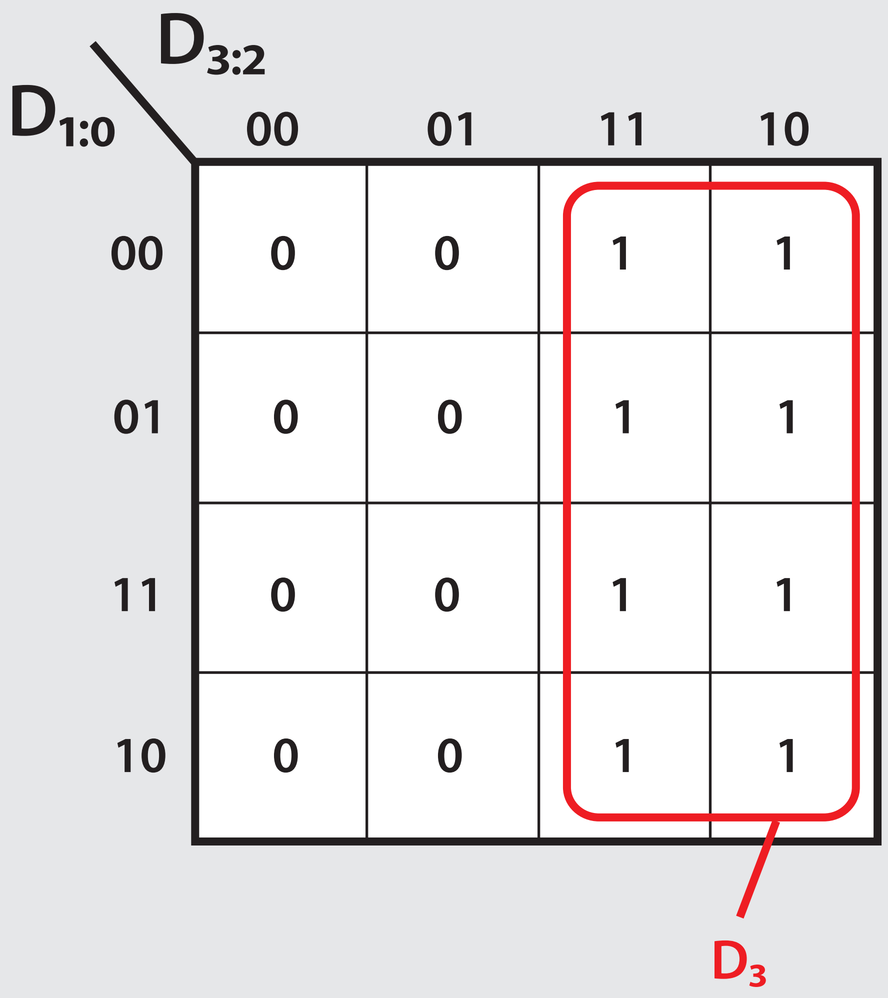
$D_{3:0}$	Sa	Sb	Sc	Sd	Se	Sf	Sg
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1
others	0	0	0	0	0	0	0

Decoder K-maps minimisation

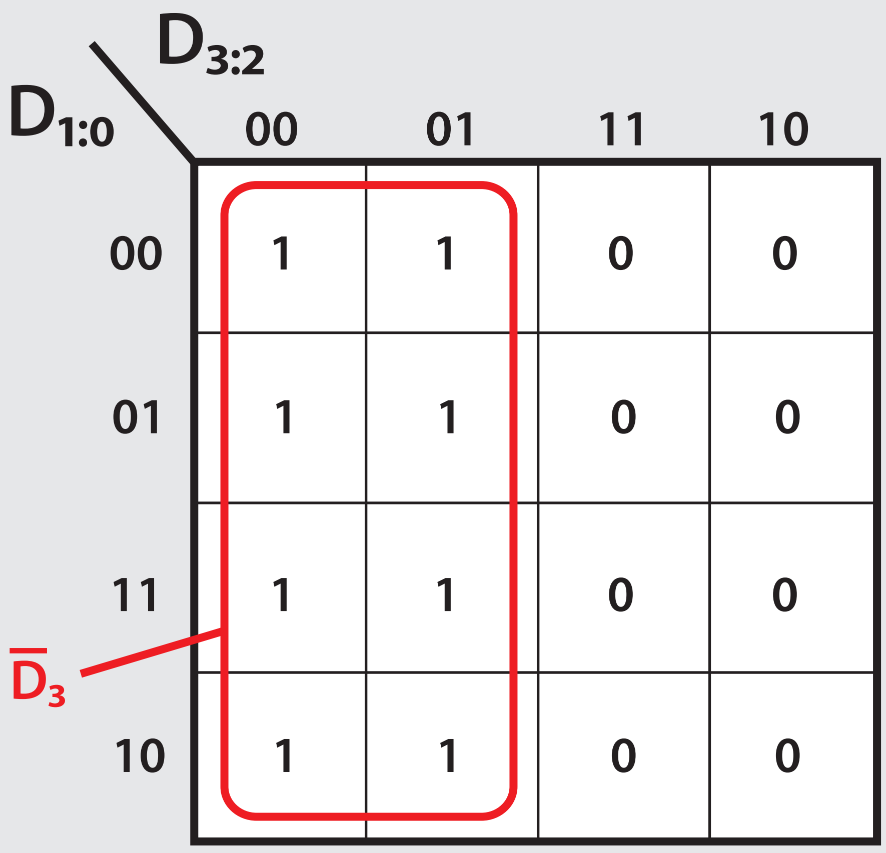
Sa $D_{1:0}$ $D_{3:2}$	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	0

Sb $D_{1:0}$ $D_{3:2}$	00	01	11	10
00	1	1	0	1
01	1	0	0	1
11	1	1	0	0
10	1	0	0	0

Decoder K-maps



$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	1	1
10	0	0	1	1

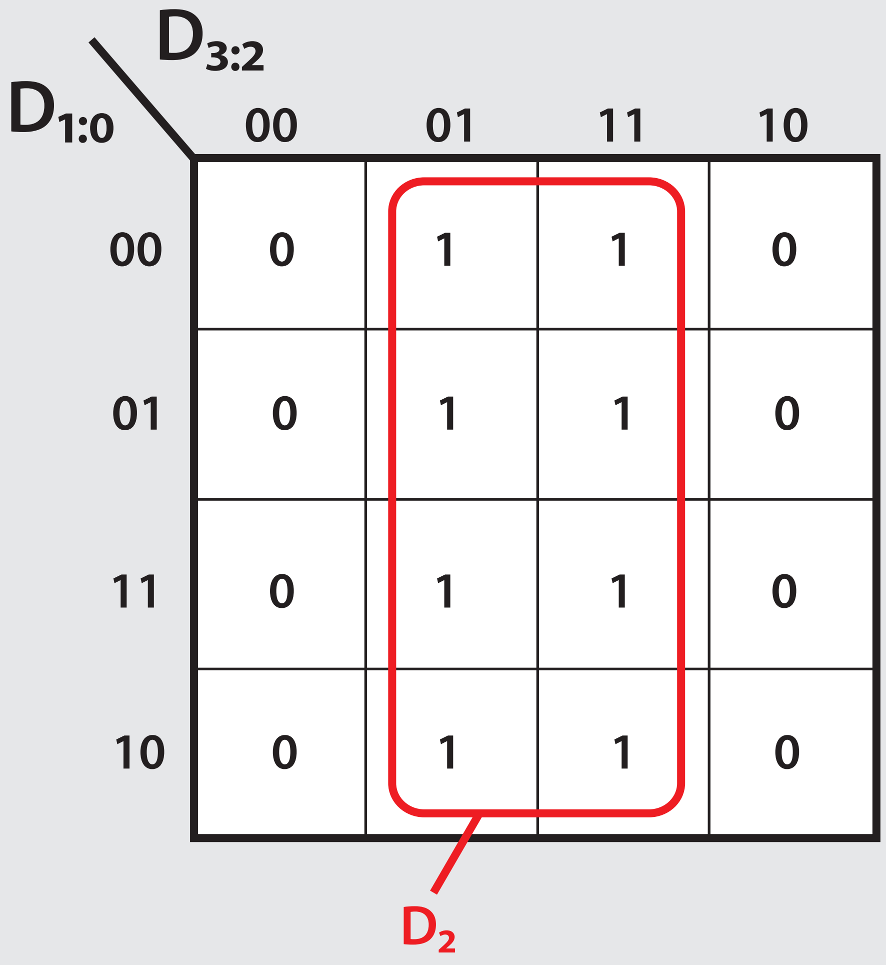


$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	1	1	0	0
10	1	1	0	0

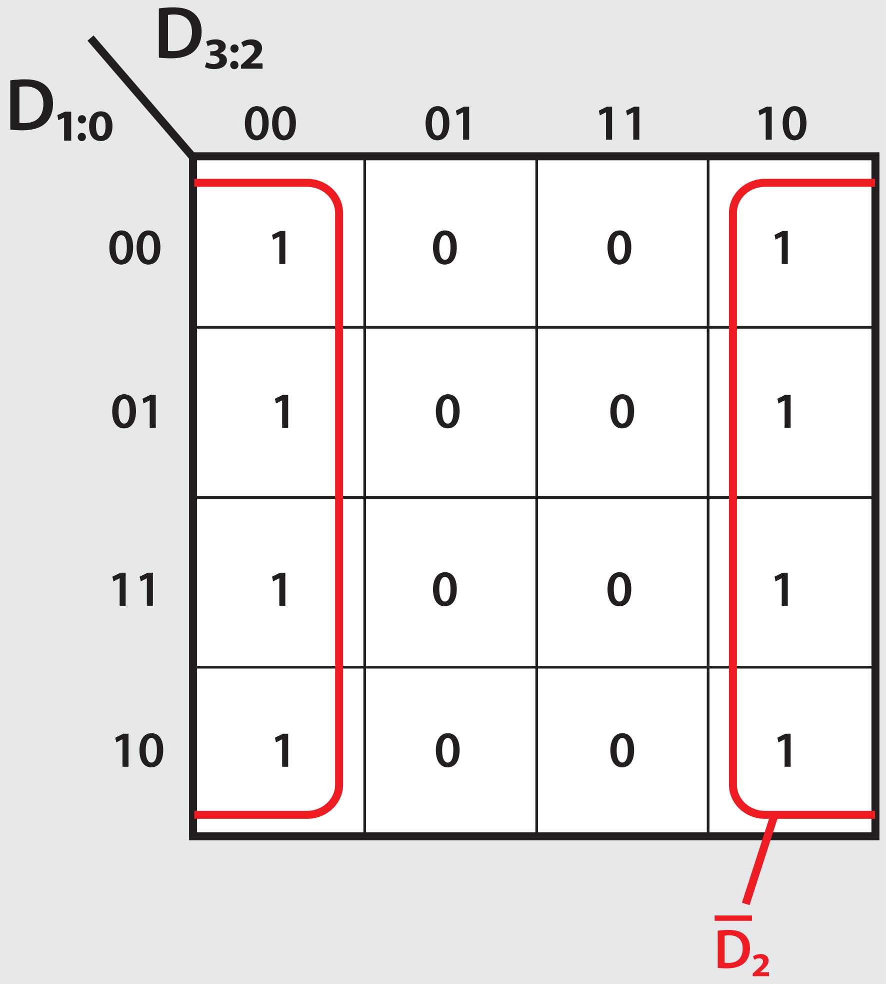
D_3 denotes the set of inputs with input bit D_3 equal to 1

$\overline{D_3}$ denotes the set of inputs with input bit D_3 equal to 0

Decoder K-maps



$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0



$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

D_2 denotes the set of inputs with input bit D_2 equal to 1

$\overline{D}_2$ denotes the set of inputs with input bit D_2 equal to 0

Decoder K-maps

$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

D_1

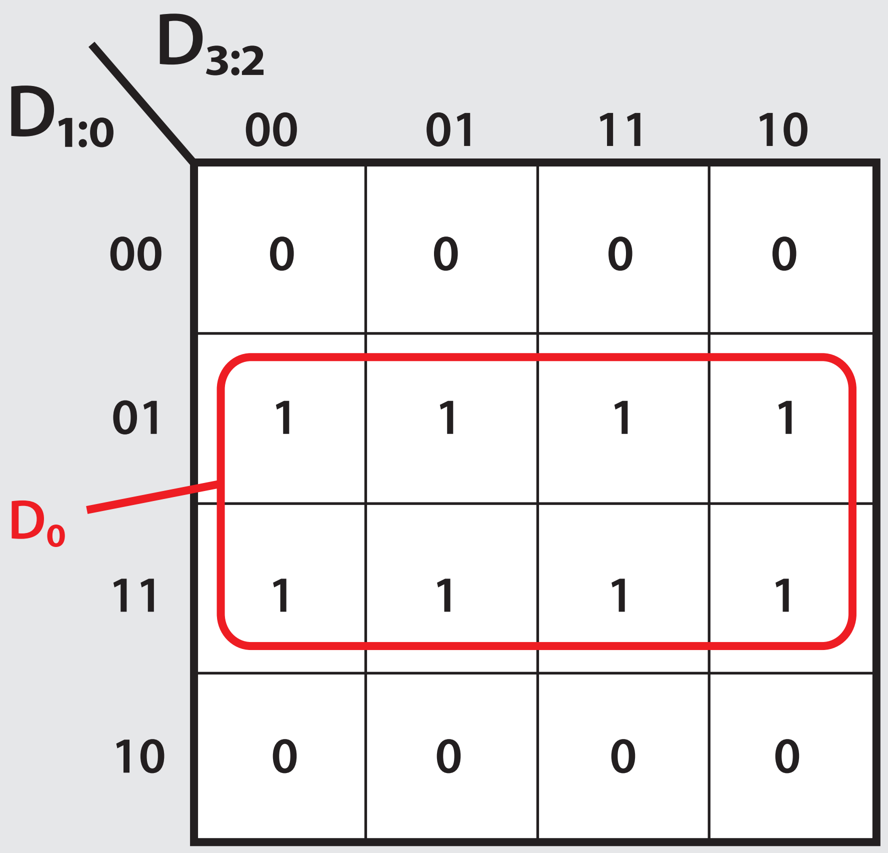
$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

$\overline{D_1}$

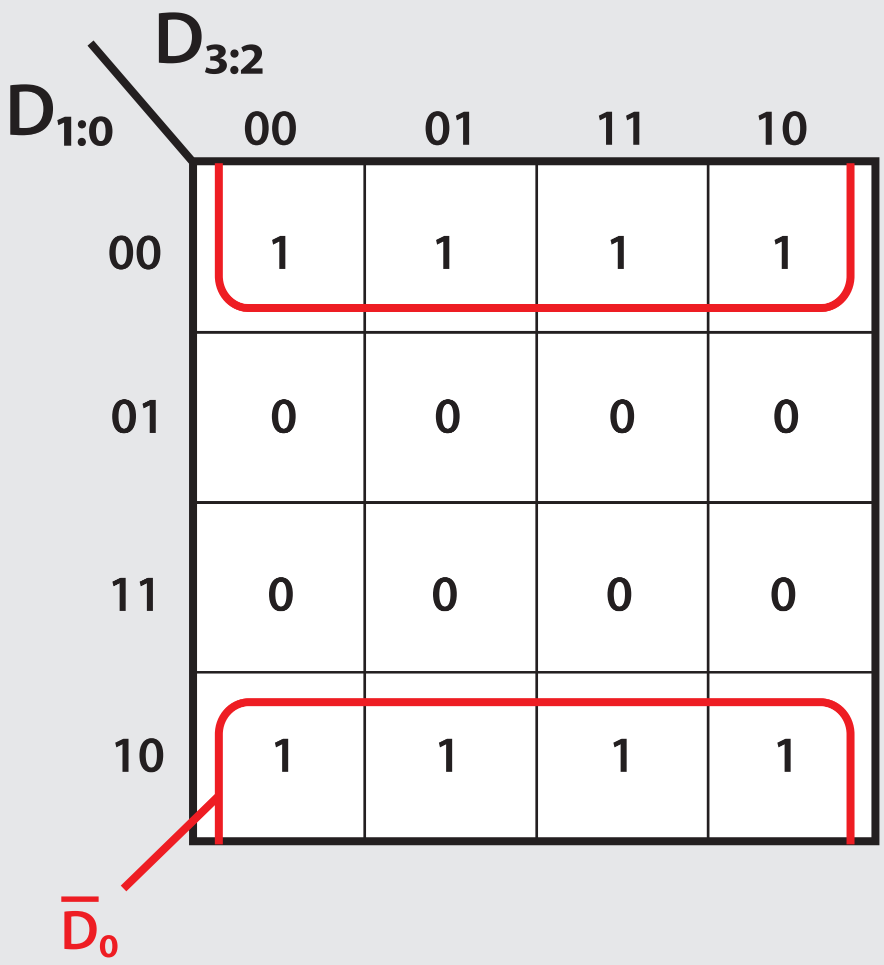
D_1 denotes the set of inputs with input bit D_1 equal to 1

$\overline{D_1}$ denotes the set of inputs with input bit D_1 equal to 0

Decoder K-maps



$$\begin{array}{c|cccc}
D_{1:0} \backslash D_{3:2} & 00 & 01 & 11 & 10 \\
\hline
00 & 0 & 0 & 0 & 0 \\
01 & 1 & 1 & 1 & 1 \\
11 & 1 & 1 & 1 & 1 \\
10 & 0 & 0 & 0 & 0
\end{array}$$



$$\begin{array}{c|cccc}
D_{1:0} \backslash D_{3:2} & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 1 & 1 & 1 \\
01 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 \\
10 & 1 & 1 & 1 & 1
\end{array}$$

D_0 denotes the set of inputs with input bit D_0 equal to 1

$\overline{D}_0$ denotes the set of inputs with input bit D_0 equal to 0

Decoder K-maps minimisation

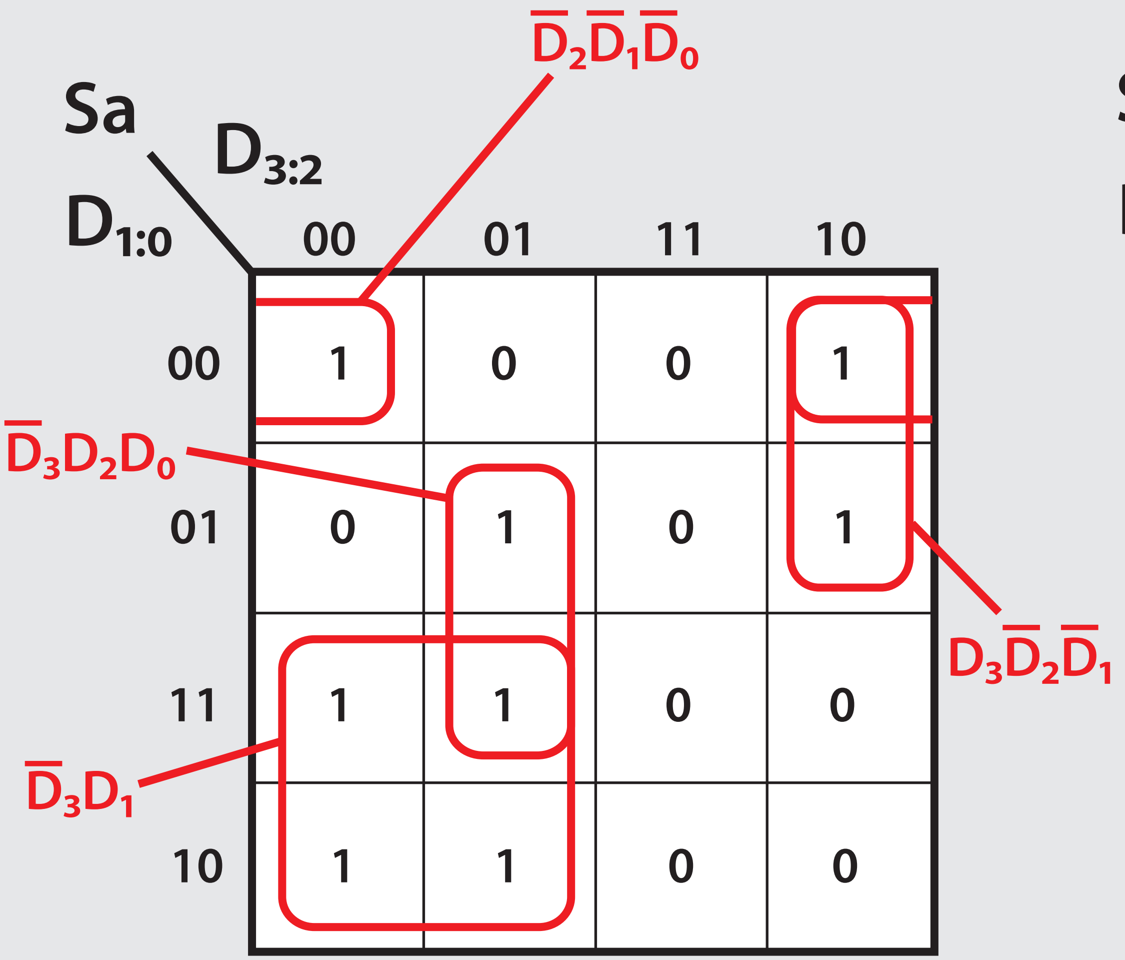
Sa

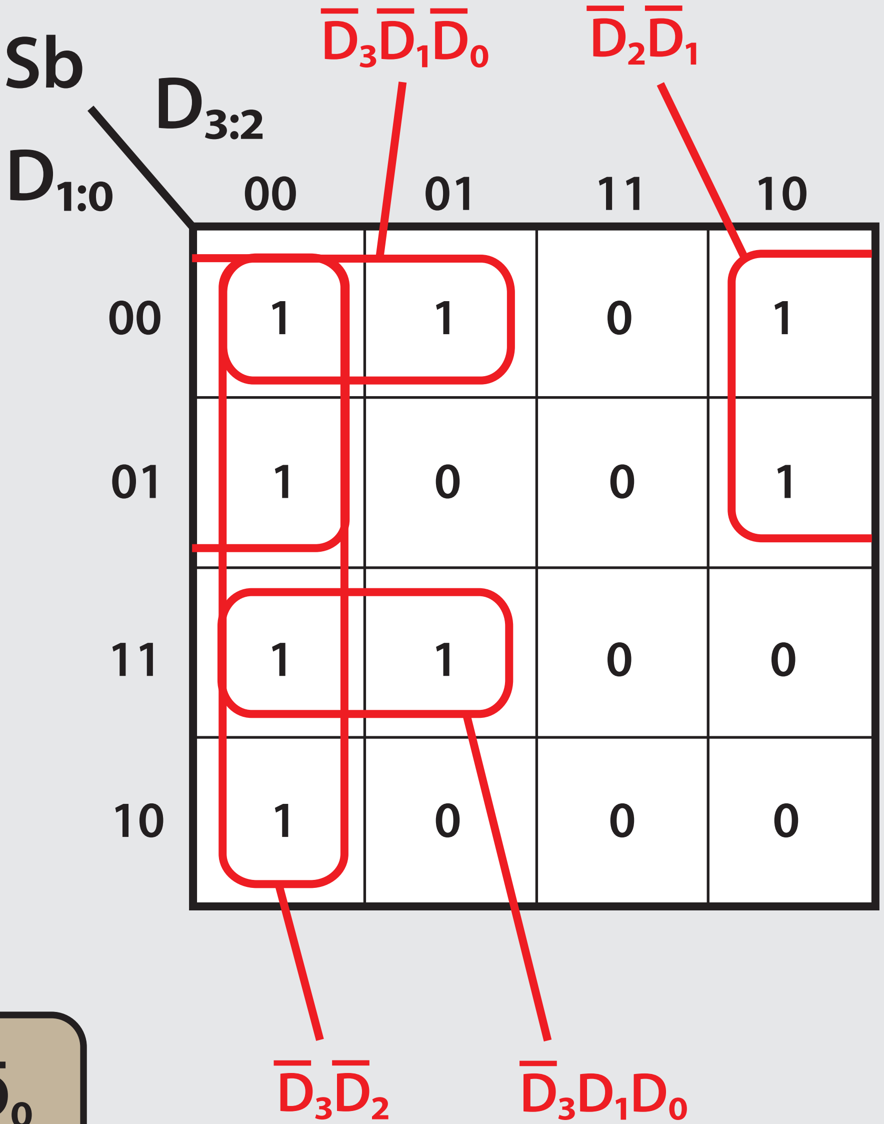
$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	0

Sb

$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	1	0	1
01	1	0	0	1
11	1	1	0	0
10	1	0	0	0

Decoder K-maps minimisation



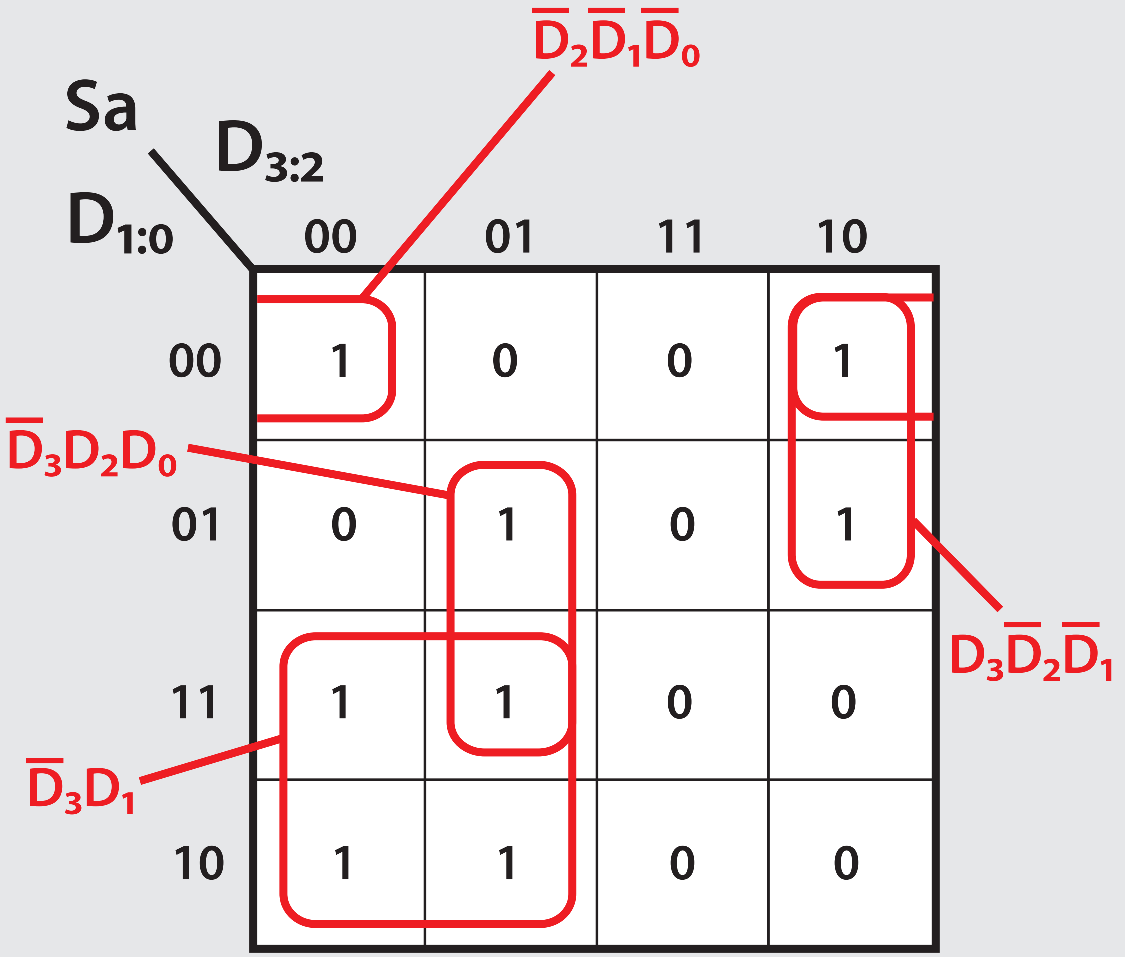


Sb
$D_{3:2}$
$D_{1:0}$
00 01 11 10
00 01 11 10
1 1 0 1
1 0 0 1
1 1 0 0
1 0 0 0
$\overline{D}_3\overline{D}_1\overline{D}_0$
$\overline{D}_2\overline{D}_1$
$\overline{D}_3\overline{D}_2$
$\overline{D}_3D_1D_0$

$$Sa = \overline{D}_3D_1 + \overline{D}_3D_2D_0 + D_3\overline{D}_2\overline{D}_1 + \overline{D}_2\overline{D}_1\overline{D}_0$$

$$Sb = \overline{D}_3\overline{D}_2 + \overline{D}_2\overline{D}_1 + \overline{D}_3D_1D_0 + \overline{D}_3\overline{D}_1\overline{D}_0$$

Decoder K-maps minimisation

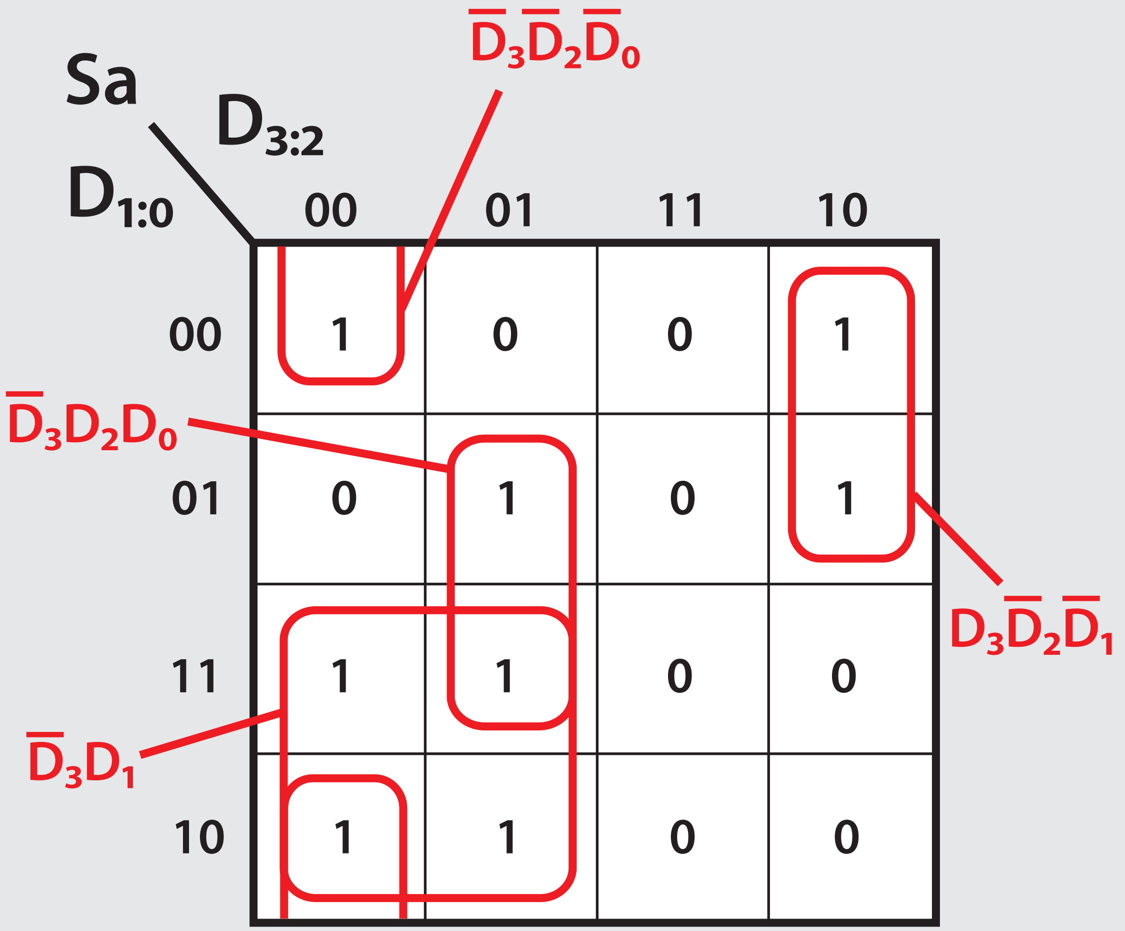


Sa
$D_{3:2}$
$D_{1:0}$
00
01
11
10
$\overline{D}_2\overline{D}_1\overline{D}_0$
$\overline{D}_3D_2D_0$
$D_3\overline{D}_2\overline{D}_1$
$\overline{D}_3D_1$

$$
\begin{array}{c|cccc}
D_{1:0} \backslash D_{3:2} & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 0 & 0 & 1 \\
01 & 0 & 1 & 0 & 1 \\
11 & 1 & 1 & 0 & 0 \\
10 & 1 & 1 & 0 & 0
\end{array}
$$

$$\boxed{Sa = \overline{D}_3D_1 + \overline{D}_3D_2D_0 + D_3\overline{D}_2\overline{D}_1 + \overline{D}_2\overline{D}_1\overline{D}_0}$$

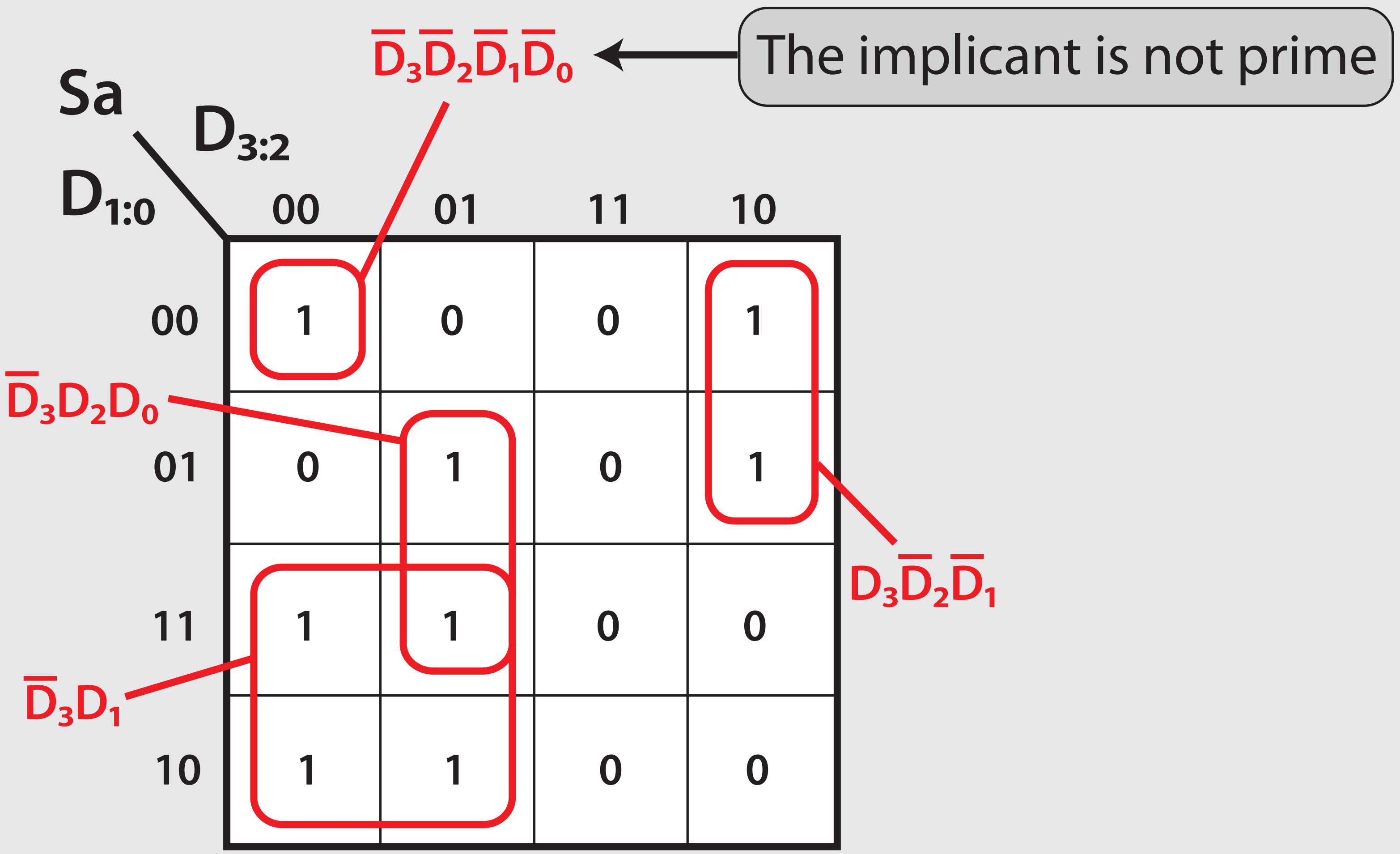
Decoder K-maps minimisation


Sa
$D_{3:2}$
$D_{1:0}$
$\overline{D}_3\overline{D}_2\overline{D}_0$
$\overline{D}_3D_2D_0$
$D_3\overline{D}_2\overline{D}_1$
$\overline{D}_3D_1$

$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	0

$$Sa = \overline{D}_3D_1 + \overline{D}_3D_2D_0 + D_3\overline{D}_2\overline{D}_1 + \overline{D}_3\overline{D}_2\overline{D}_0$$

A common mistake



The implicant is not prime
$\overline{D_3}\,\overline{D_2}\,\overline{D_1}\,\overline{D_0}$
Sa
$D_{3:2}$
$D_{1:0}$
00
01
11
10
$\overline{D_3}D_2D_0$
$D_3\overline{D_2}\,\overline{D_1}$
$\overline{D_3}D_1$

$$
\begin{array}{c|cccc}
D_{1:0} \backslash D_{3:2} & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 0 & 0 & 1 \\
01 & 0 & 1 & 0 & 1 \\
11 & 1 & 1 & 0 & 0 \\
10 & 1 & 1 & 0 & 0
\end{array}
$$

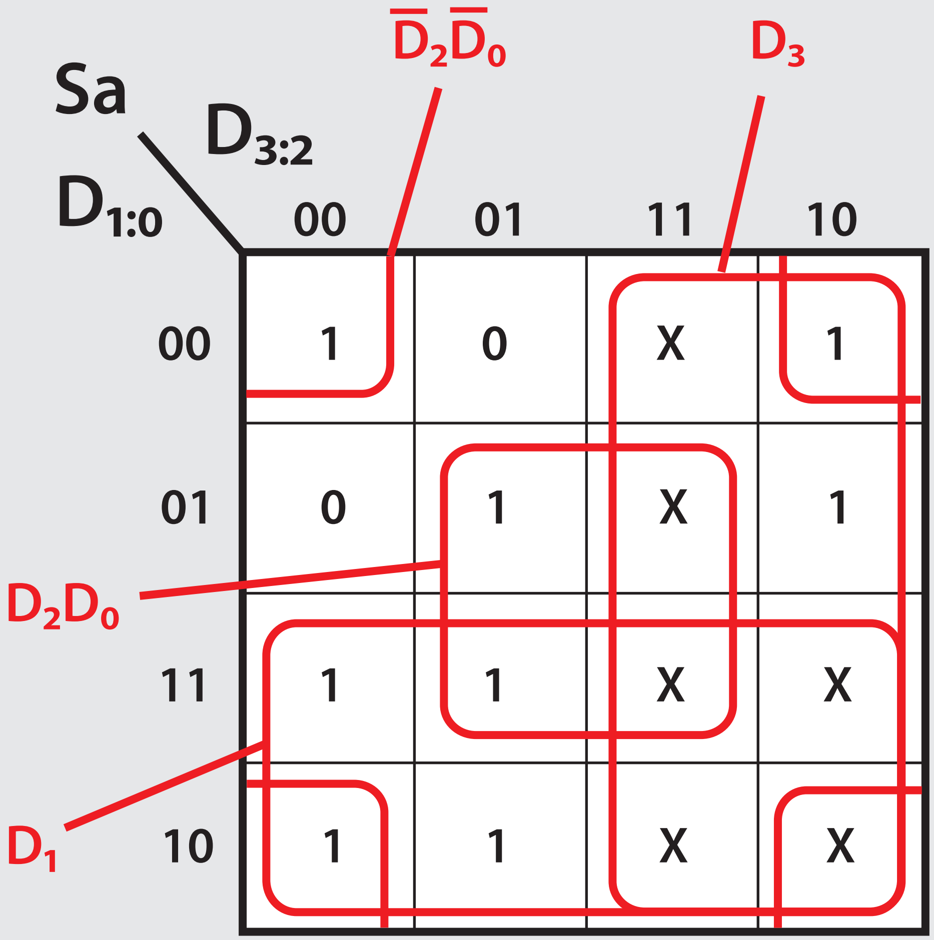
$$Sa = \overline{D_3}D_1 + \overline{D_3}D_2D_0 + D_3\overline{D_2}\,\overline{D_1} + \overline{D_3}\,\overline{D_2}\,\overline{D_1}\,\overline{D_0}$$

Seven-segment display decoder truth table with X's

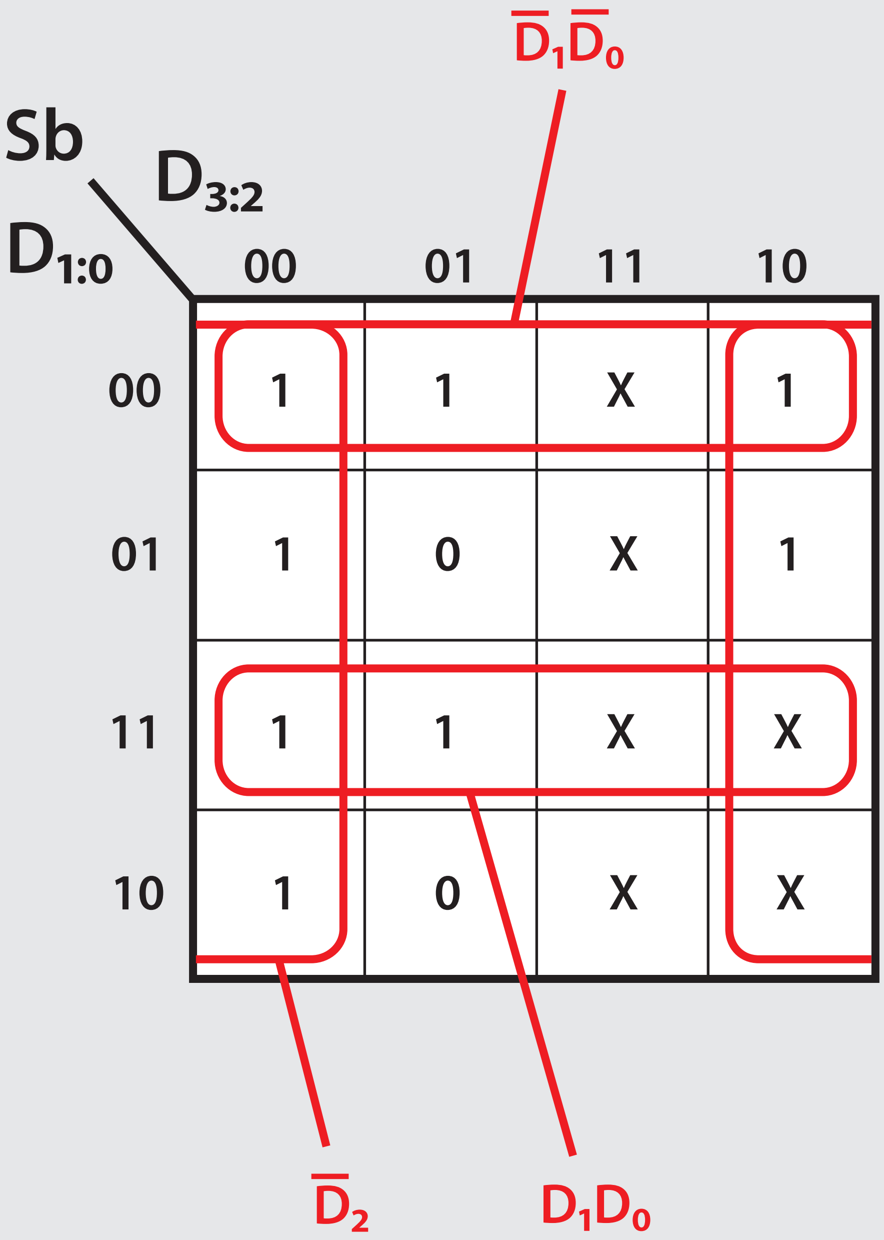
$D_{3:0}$	Sa	Sb	Sc	Sd	Se	Sf	Sg
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1
others	X	X	X	X	X	X	X

Here, the value **X** means that the value is undetermined : it can be 0 or 1.

Decoder K-maps minimisation with X's



$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	0	X	1
01	0	1	X	1
11	1	1	X	X
10	1	1	X	X



$D_{1:0}$ \ $D_{3:2}$	00	01	11	10
00	1	1	X	1
01	1	0	X	1
11	1	1	X	X
10	1	0	X	X

$$Sa = D_3 + D_2D_0 + \overline{D}_2\overline{D}_0 + D_1$$

$$Sb = \overline{D}_2 + D_1D_0 + \overline{D}_1\overline{D}_0$$

Exercise 1

Complete the design of the seven-segment decoder
by designing boolean equations for the segments Sc and Sd :

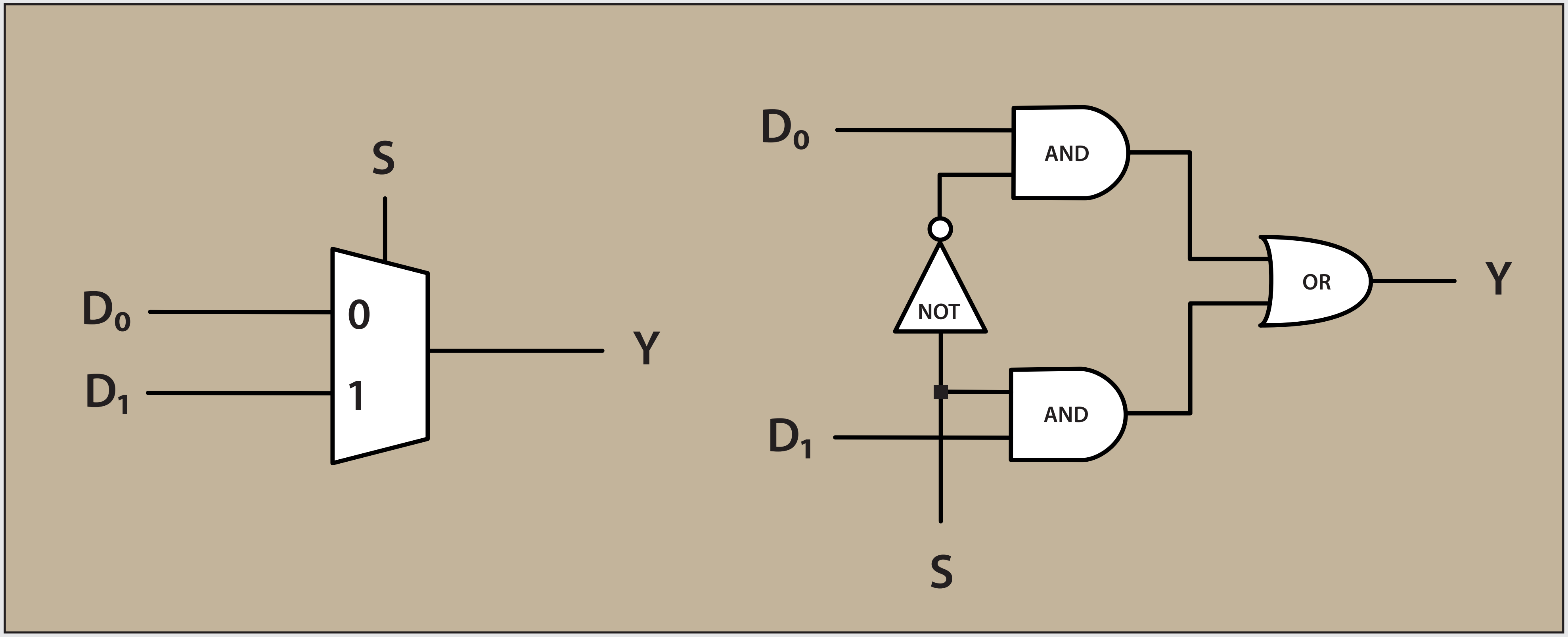
a. assuming that inputs greater than 9
must produce blank (0) outputs

b. assuming that inputs greater than 9
are don’t cares

Then, sketch a reasonably simple gate-level implementation
in the case b. and simulate the resulting circuits on CircuitLab.

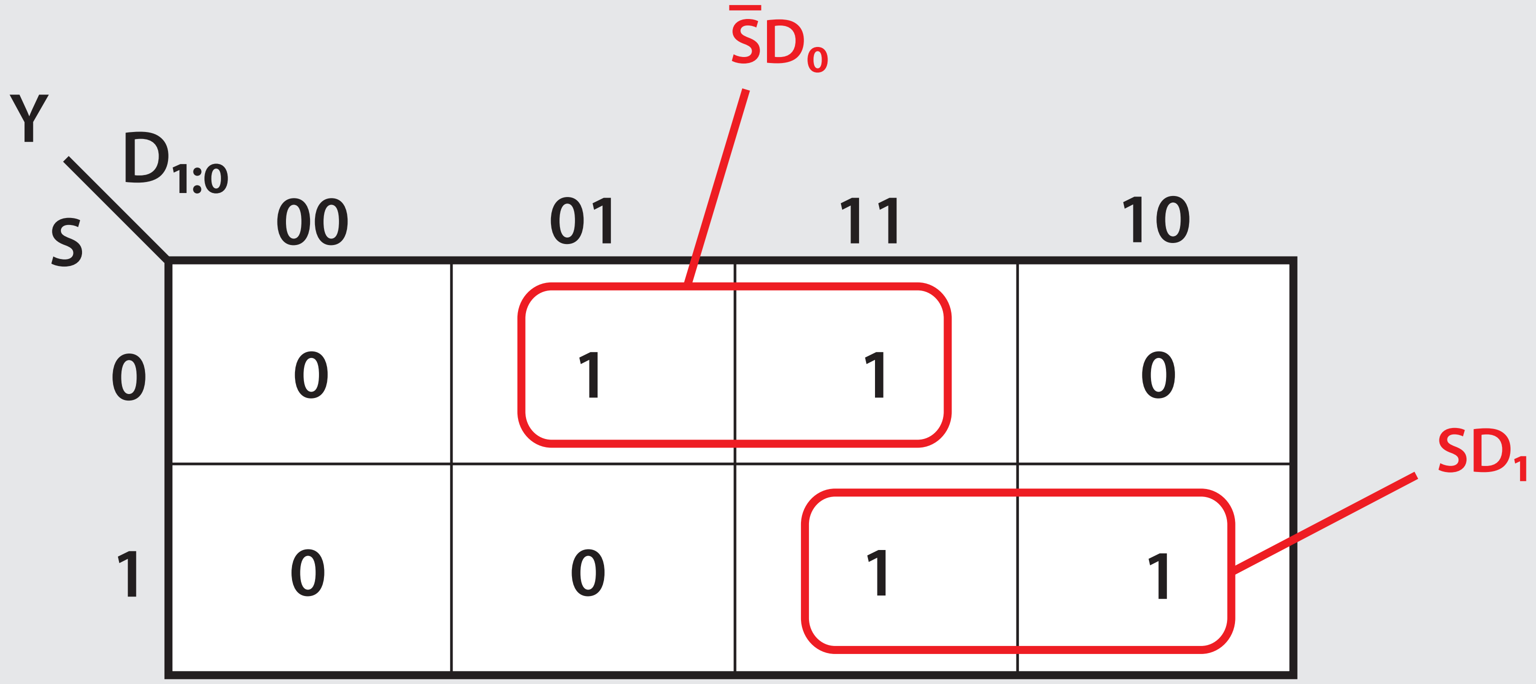
Combinational Building Blocks

Multiplexor (mux)



S	Y
0	D_0
1	D_1

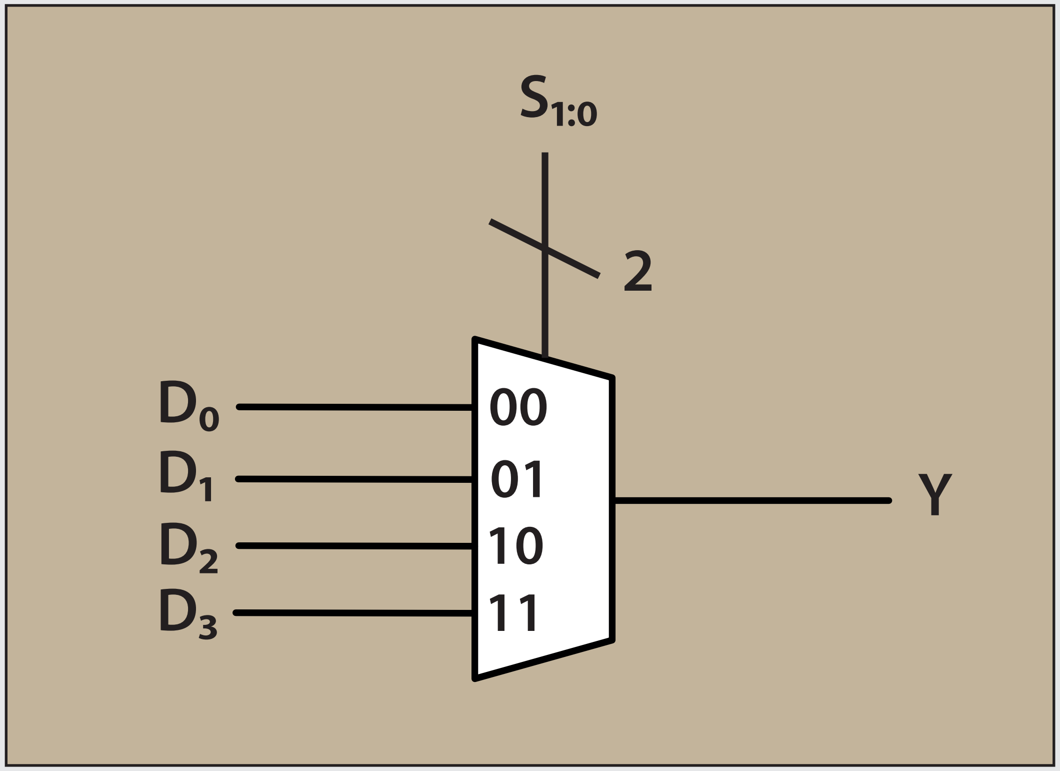
K-map minimisation



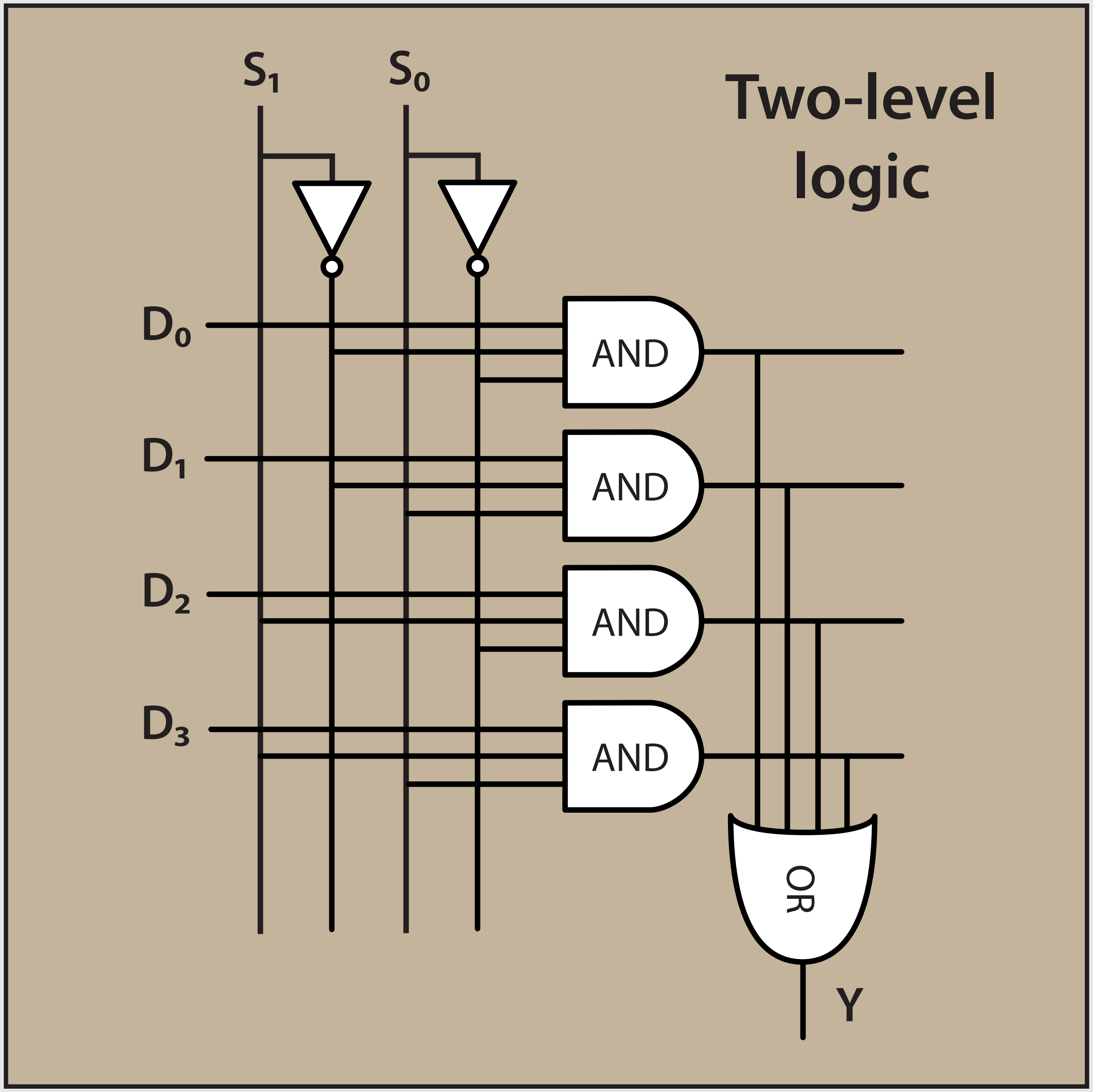
By circling the 1's in adjacent squares, one deduces that :

$$\boxed{Y = SD_1 + \bar{S}D_0}$$

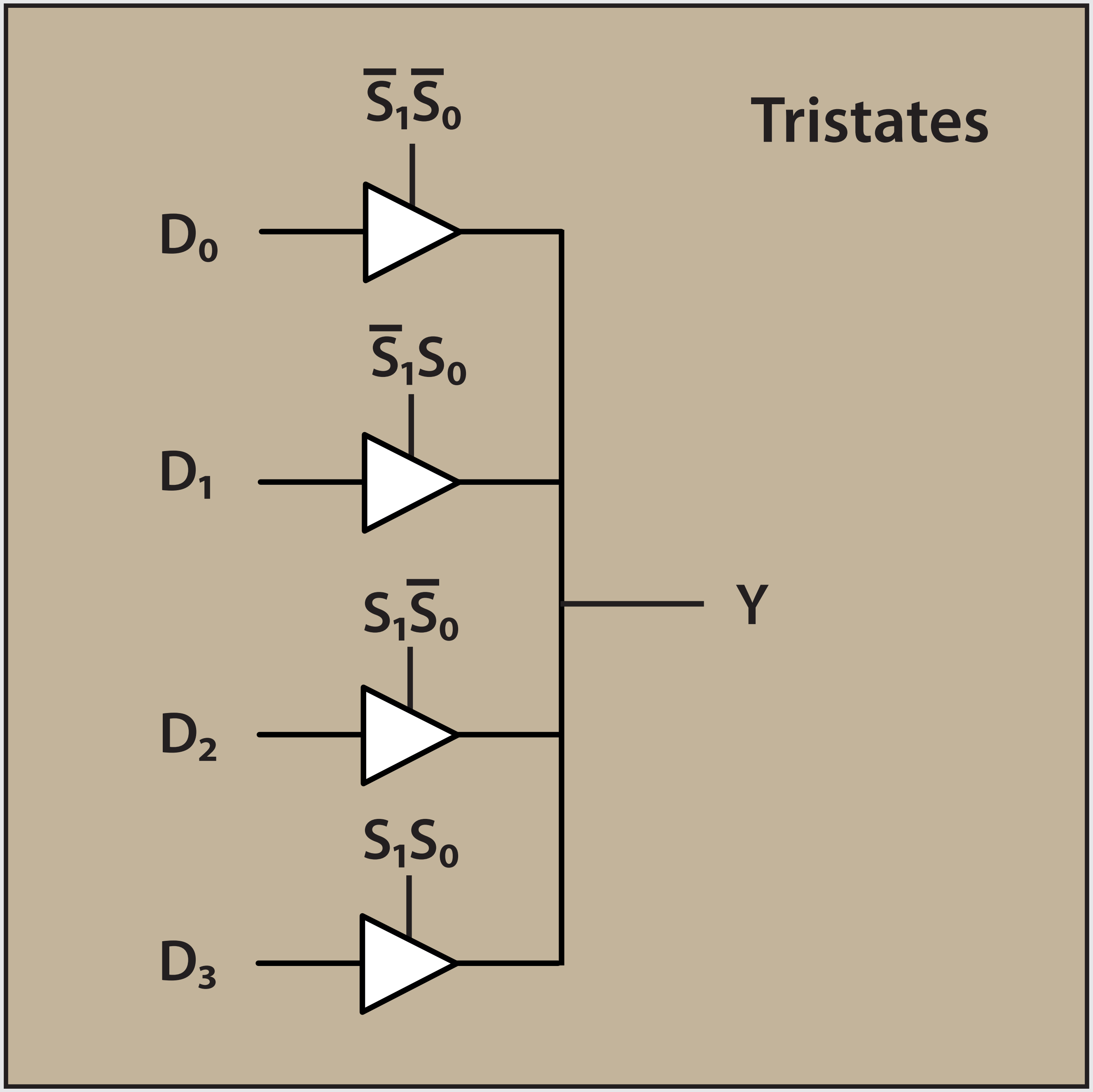
4 : 1 multiplexers



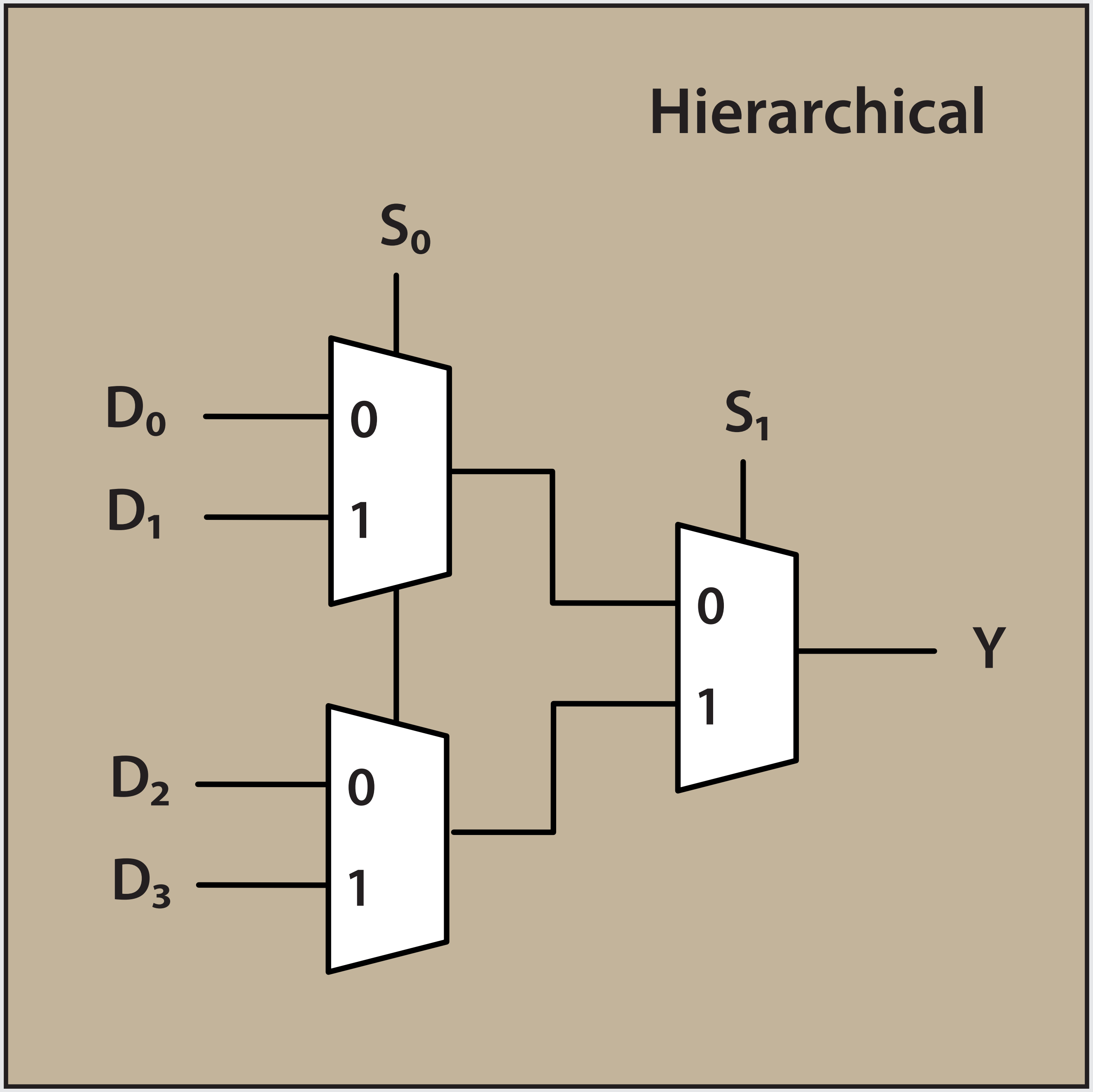
Implementation of the 4:1 multiplexer



Implementation of the 4:1 multiplexer



Implementation of the 4:1 multiplexer



Exercise

(Example 2.12 in Harris and Harris)

Alyssa P. Hacker needs to implement the function

$$Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$$

to finish her senior project, but when she looks in her lab kit, the only part she has left is an 8:1 multiplexer.

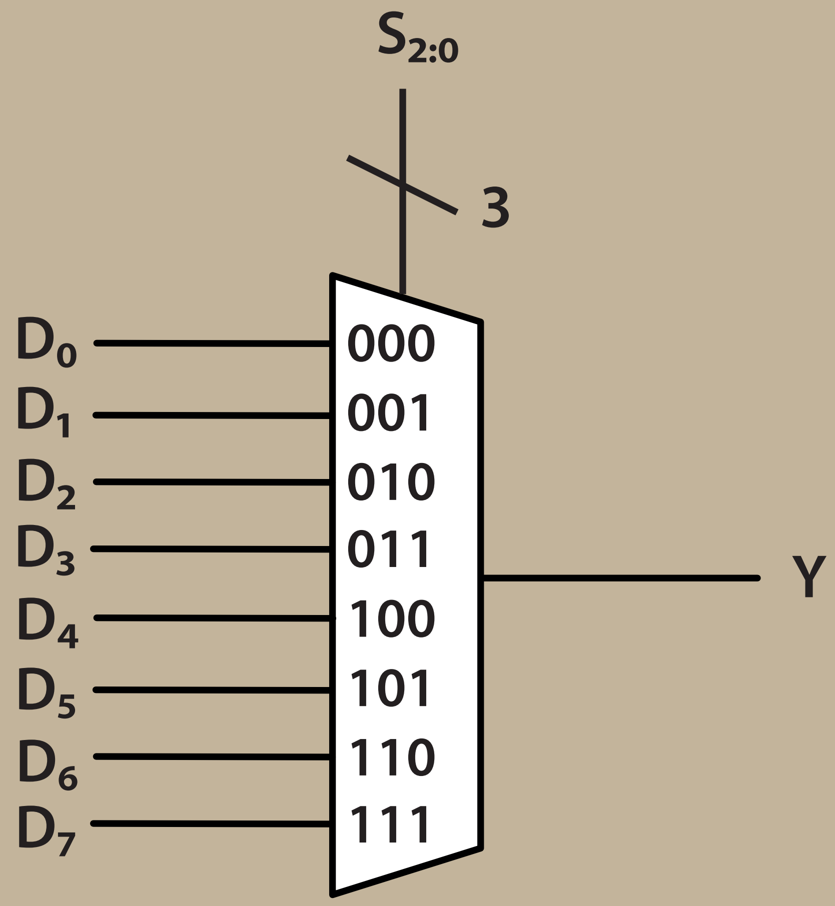
How does she implement the function?

Solution

$$Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$$

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

8:1 multiplexer (mux)

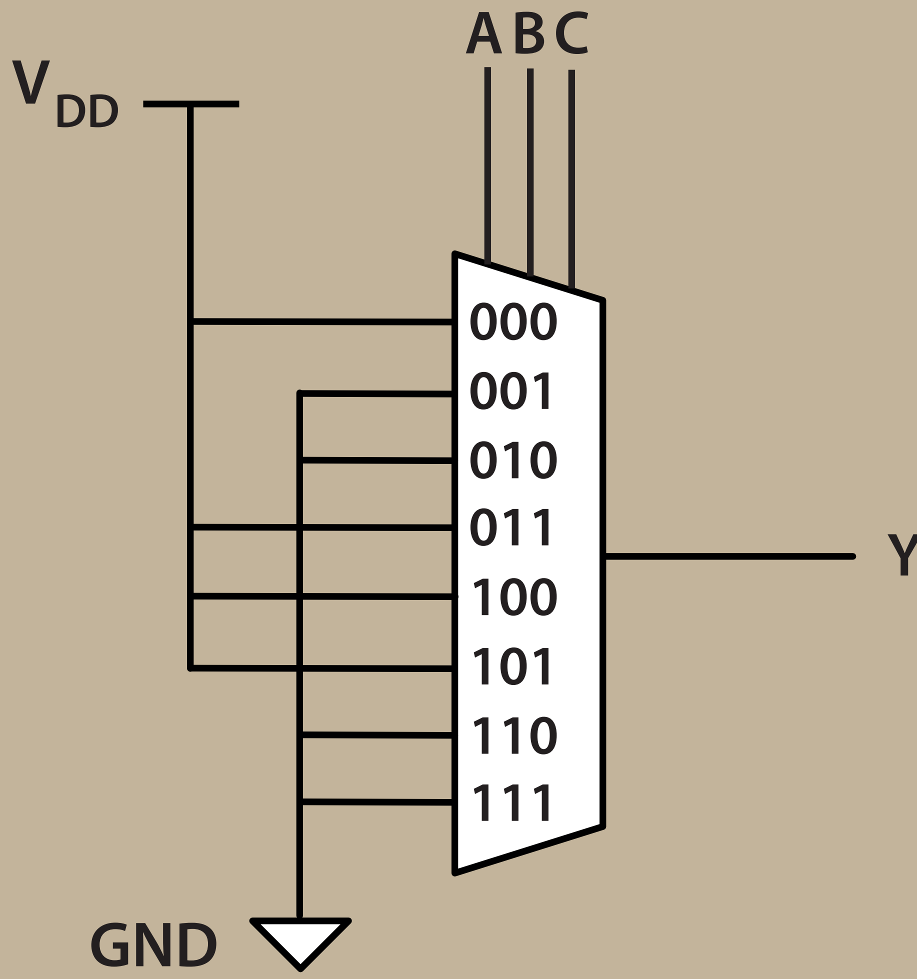


Solution

$$Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$$

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

8:1 multiplexer (mux)

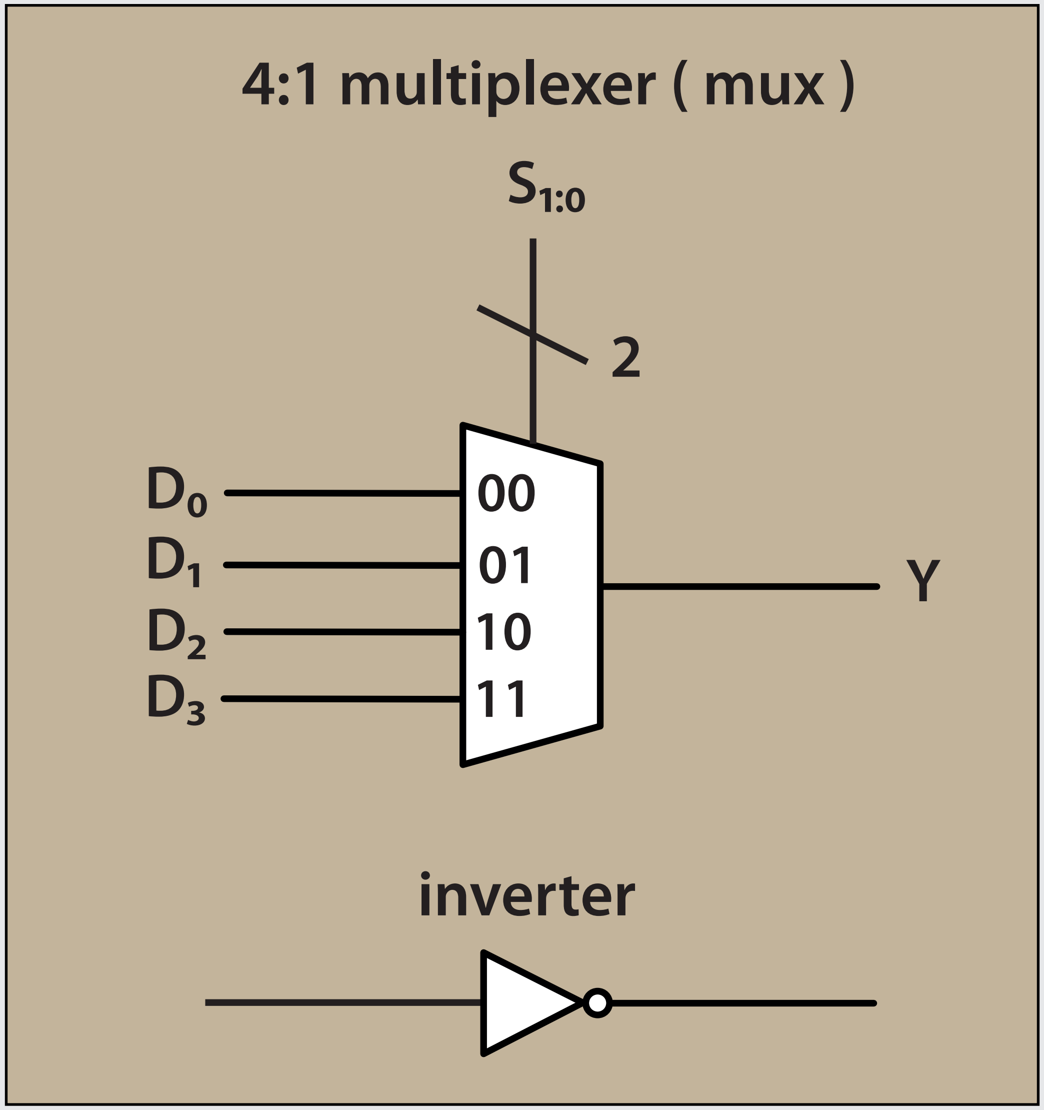


Exercise

Can you construct the same boolean function with a 4:1 multiplexer and an inverter ?

$$Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$$

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

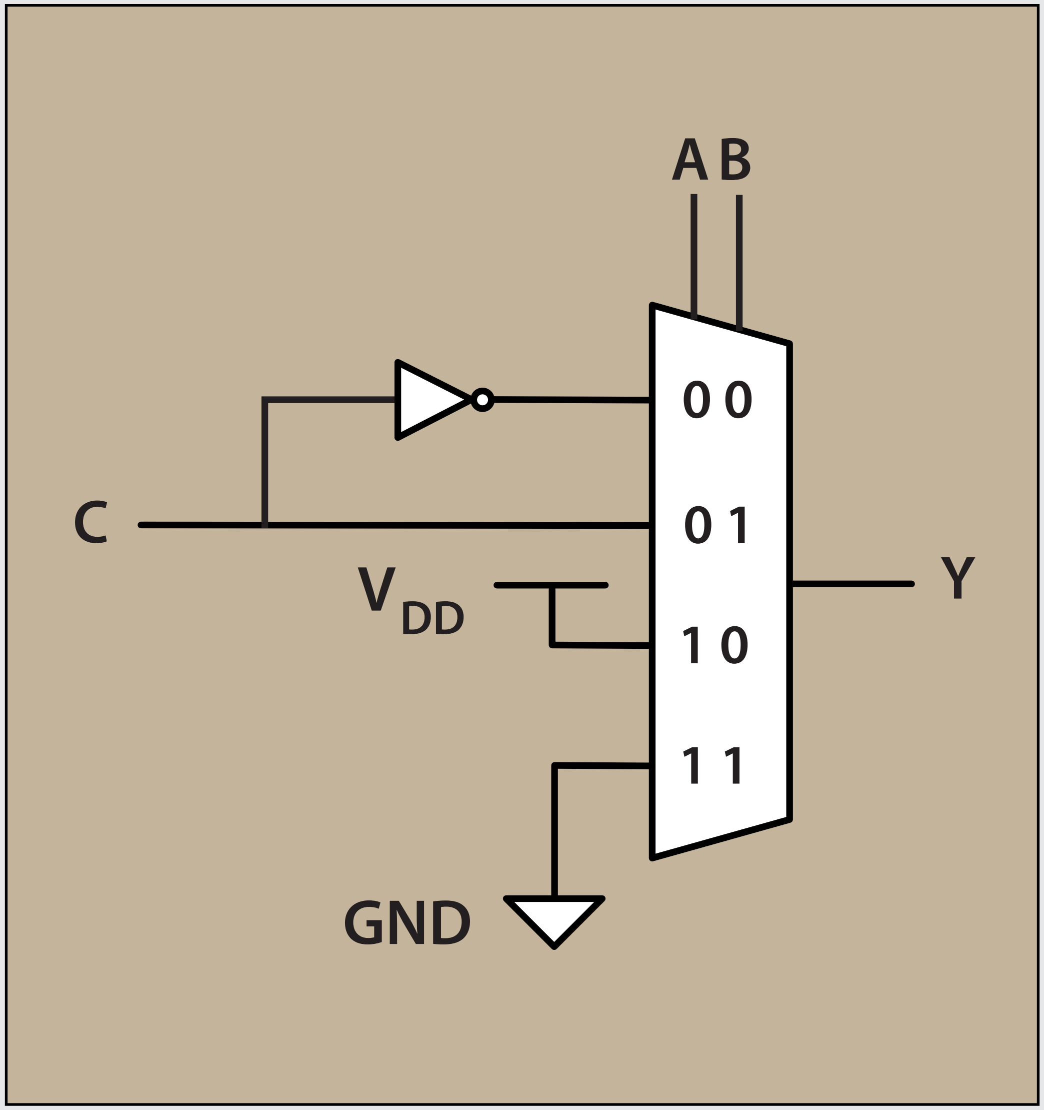


Solution

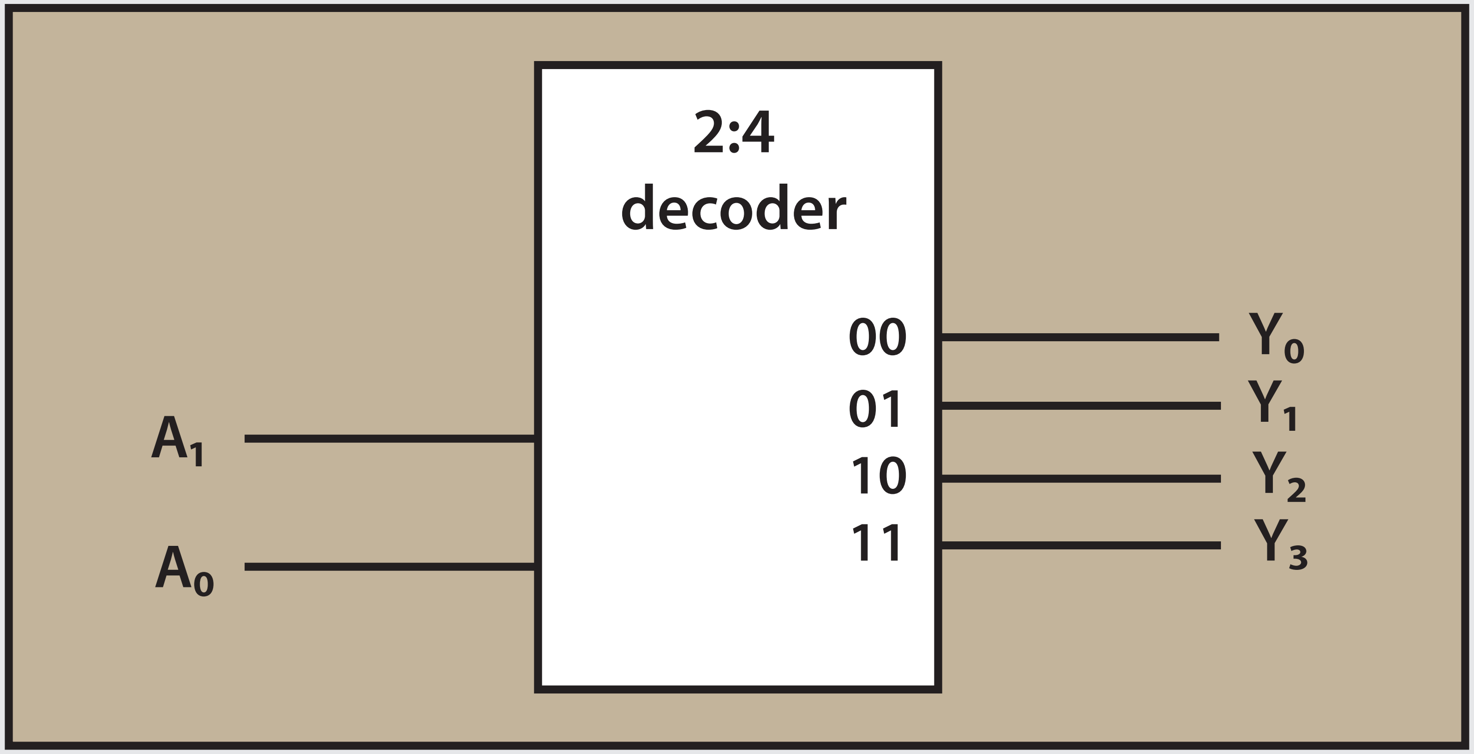
Can you construct the same boolean function with a 4:1 multiplexer and an inverter ?

$Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$

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

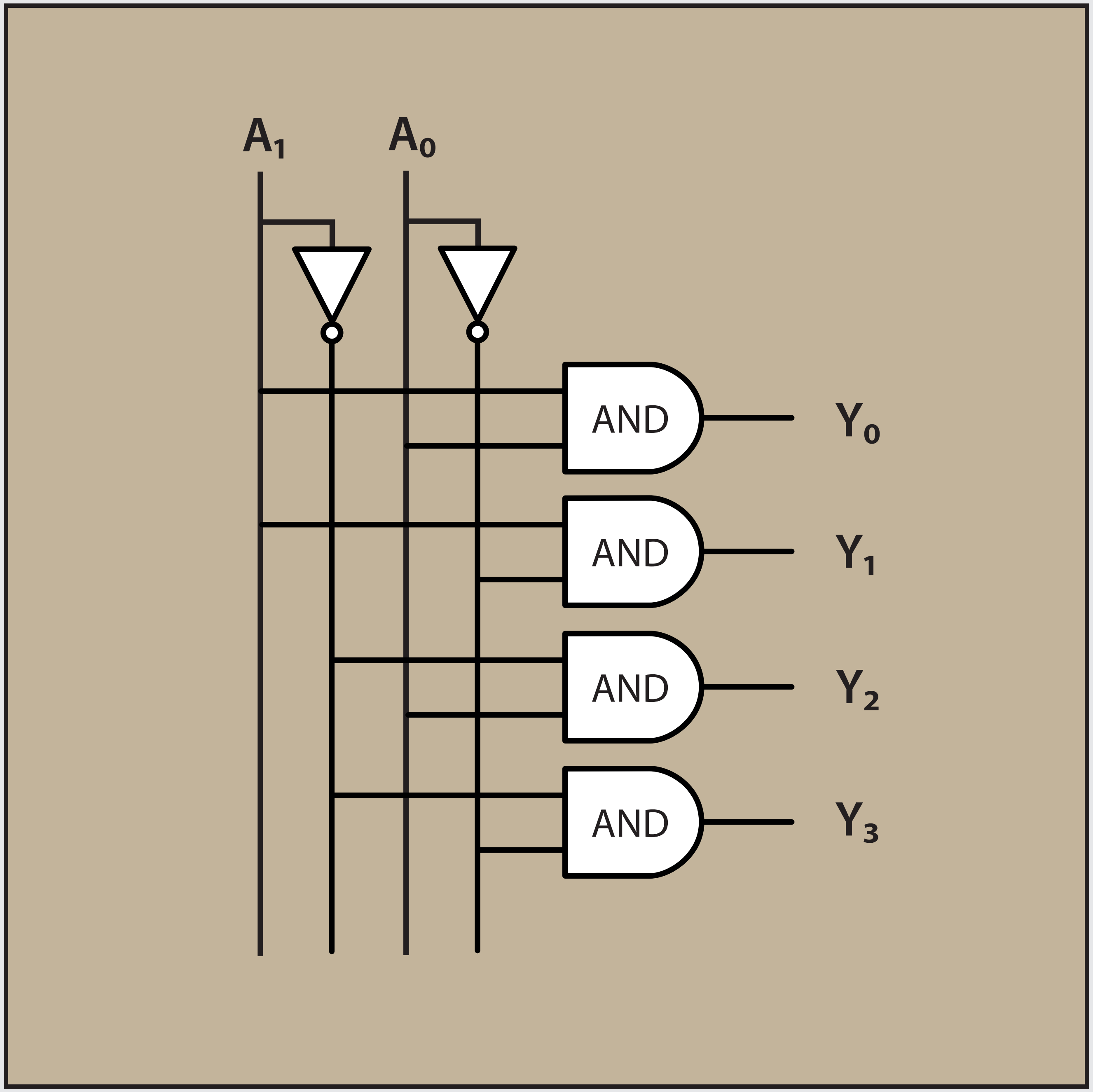


Decoders



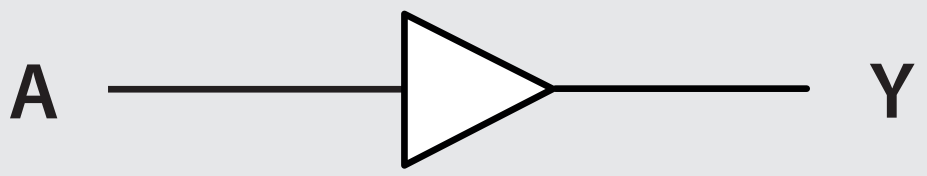
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

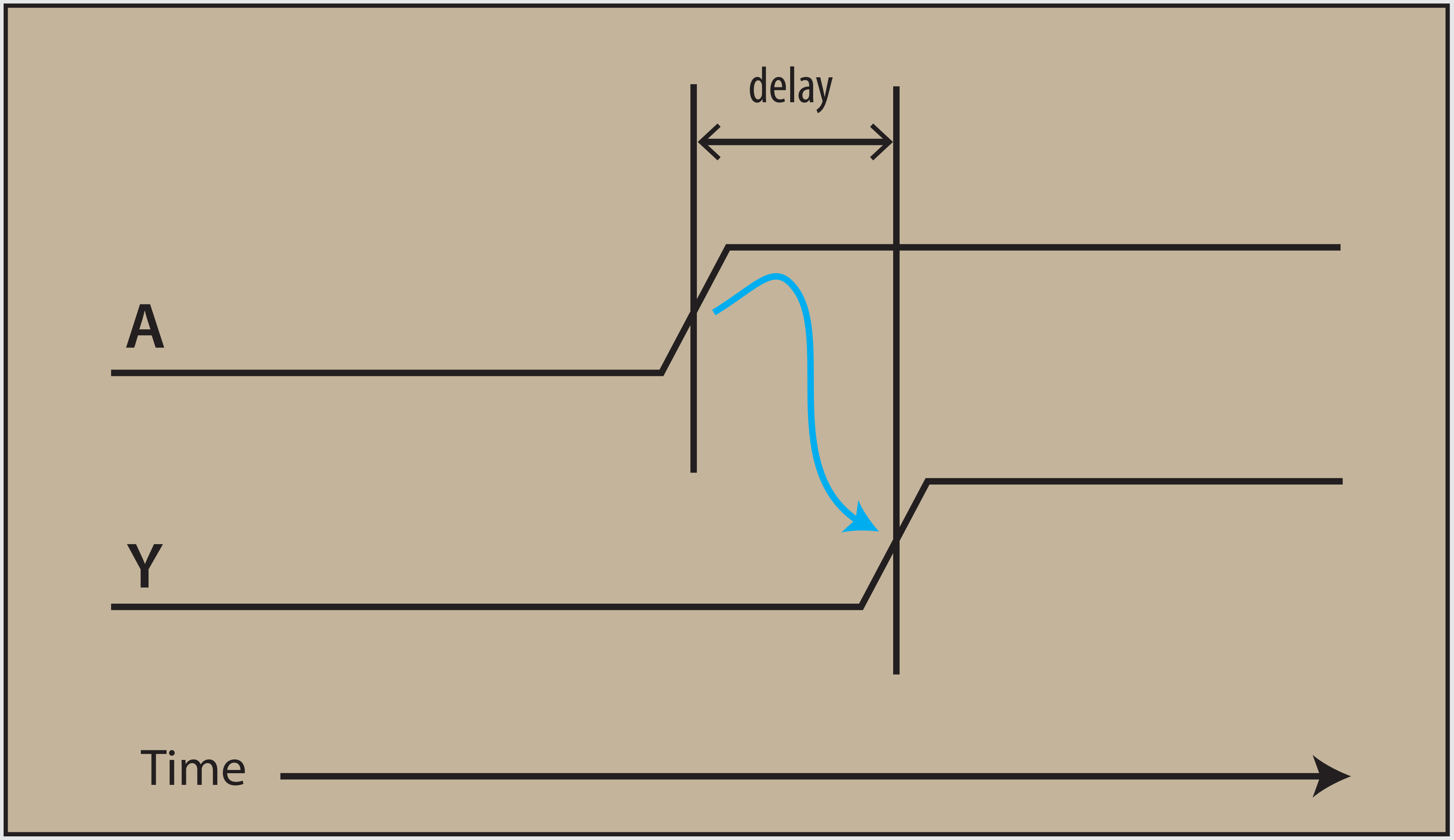
Implementation of the 2:4 decoder



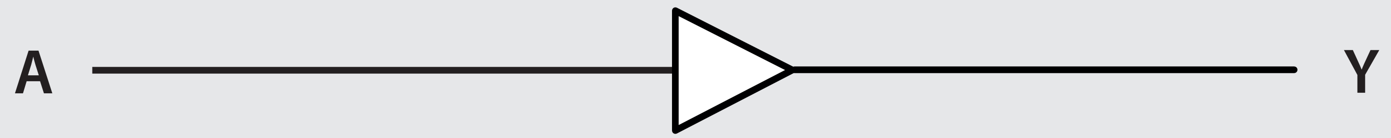
Timing

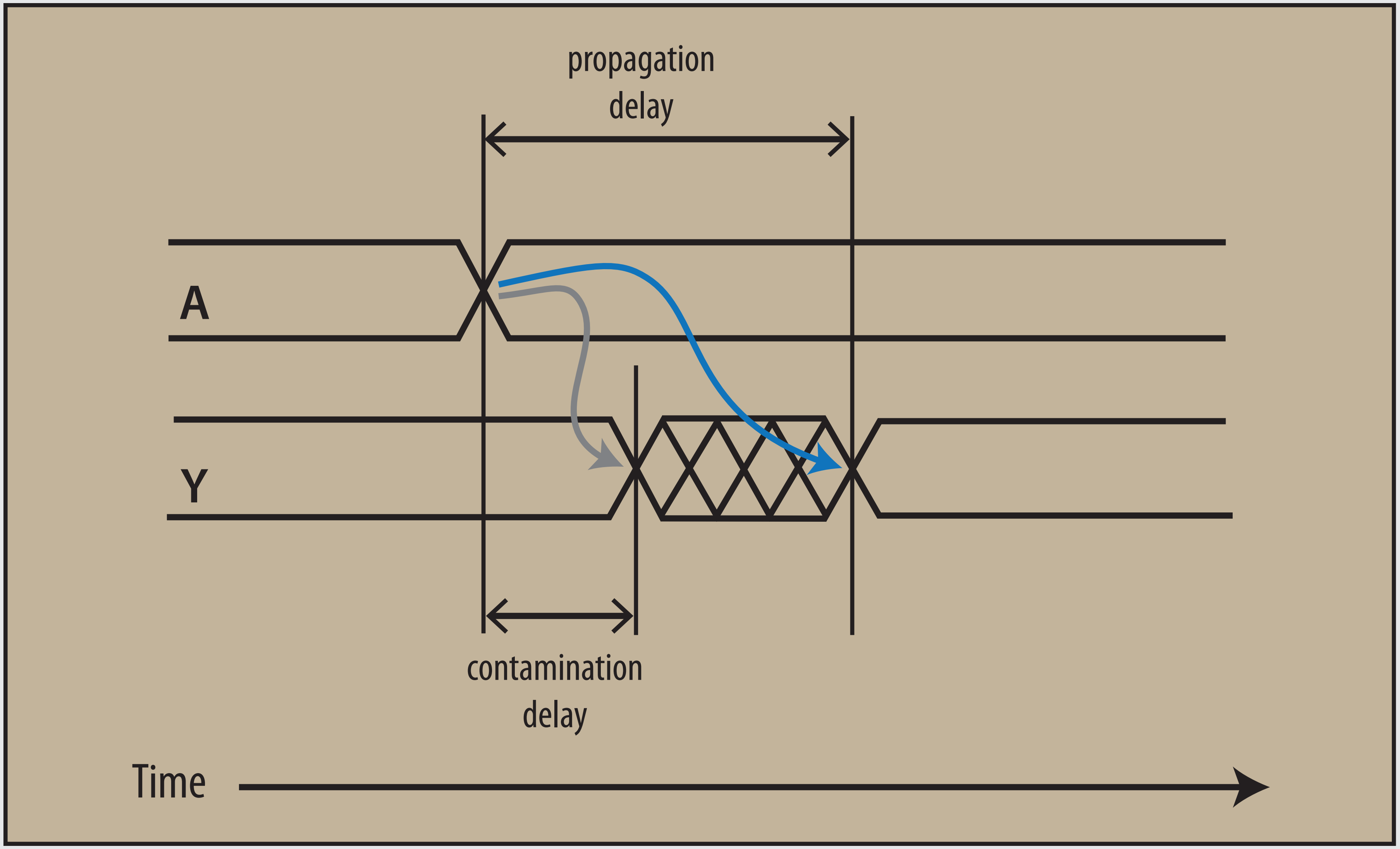
Time delay



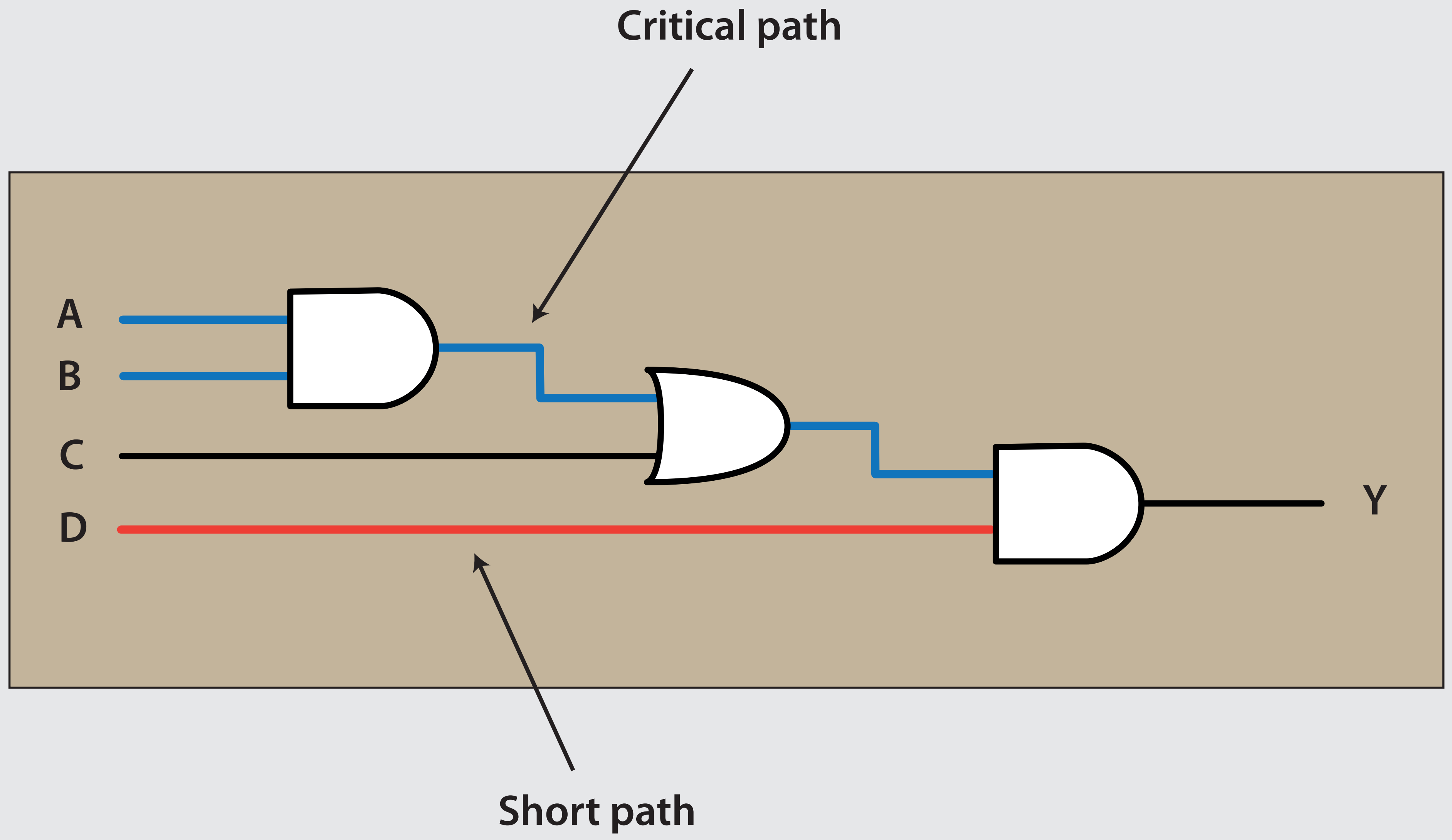


Propagation and contamination delay

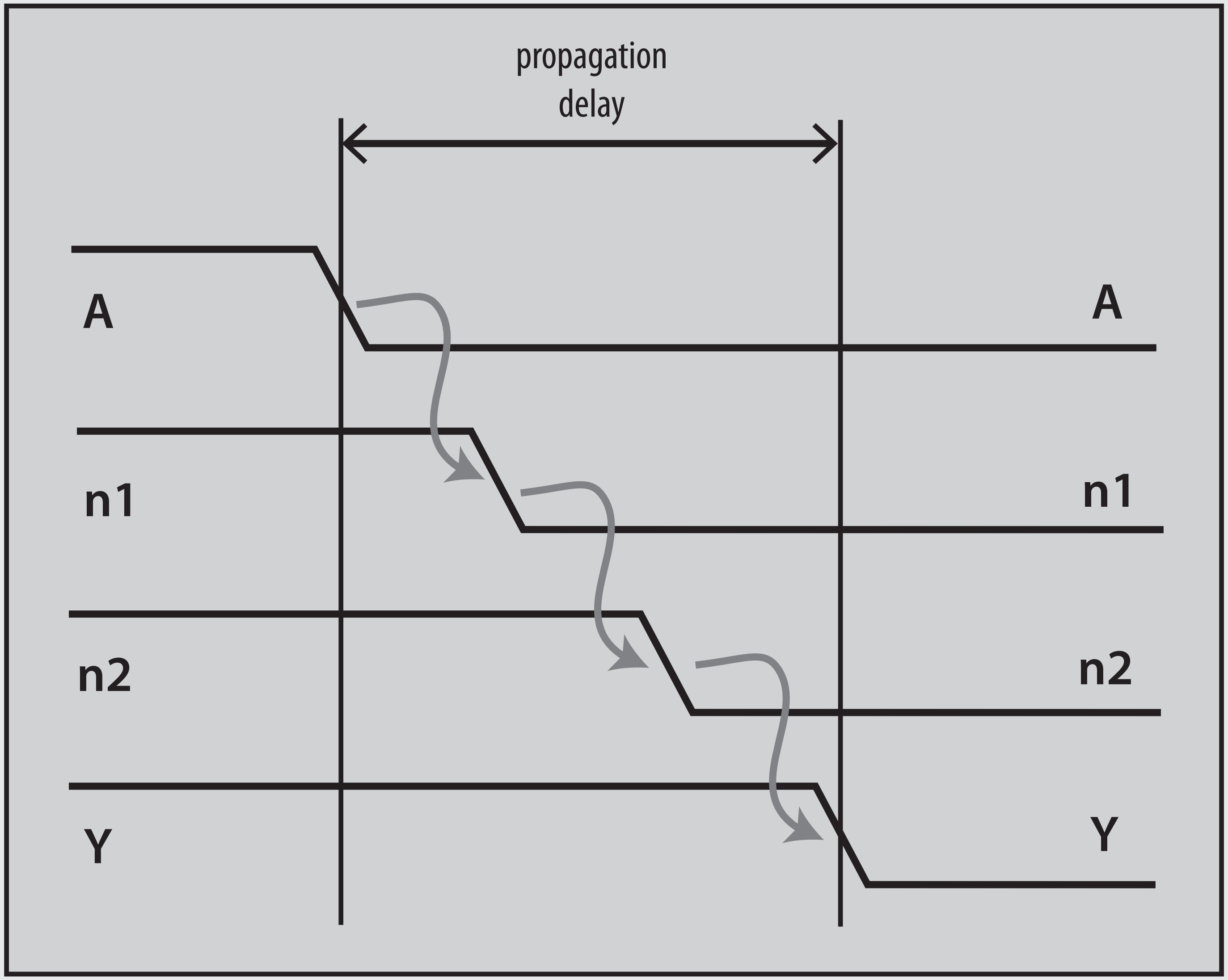


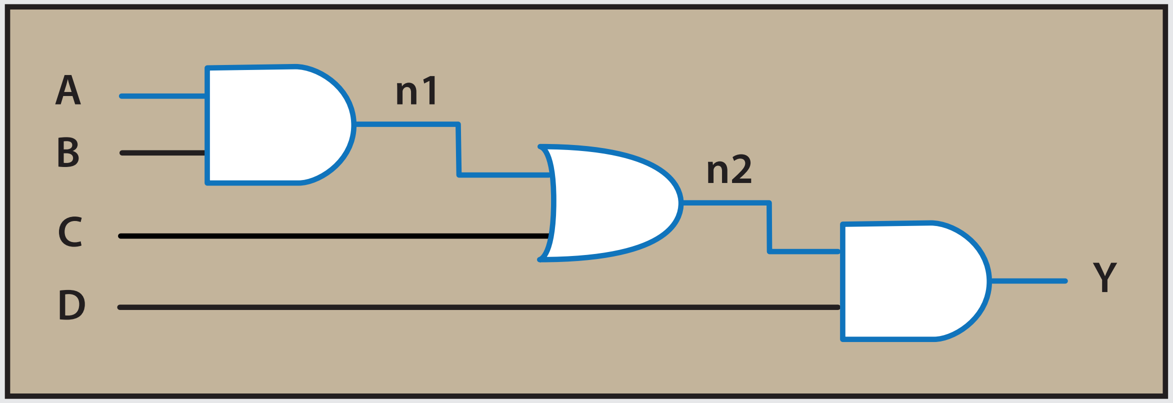


Critical vs. short path



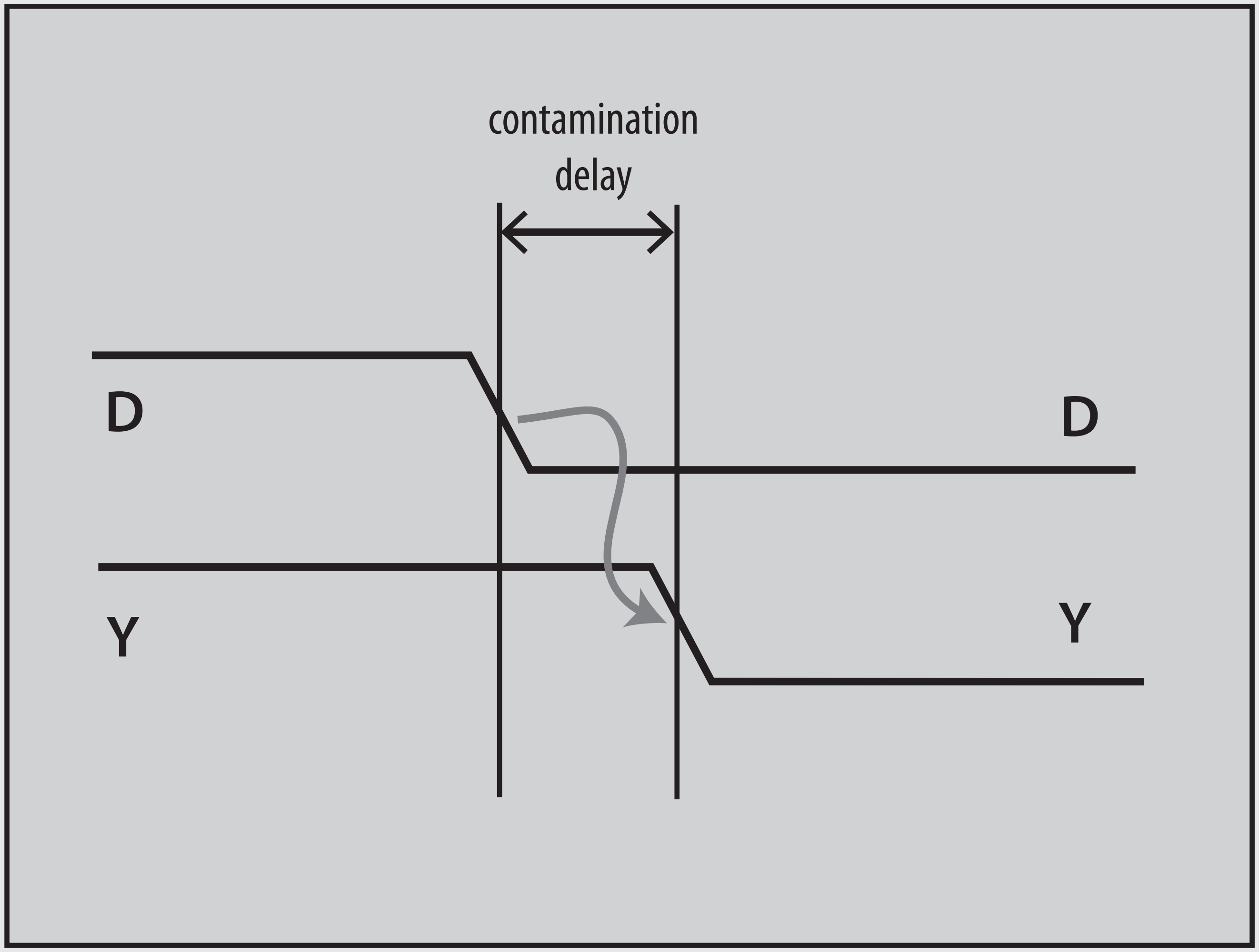
Critical path waveform

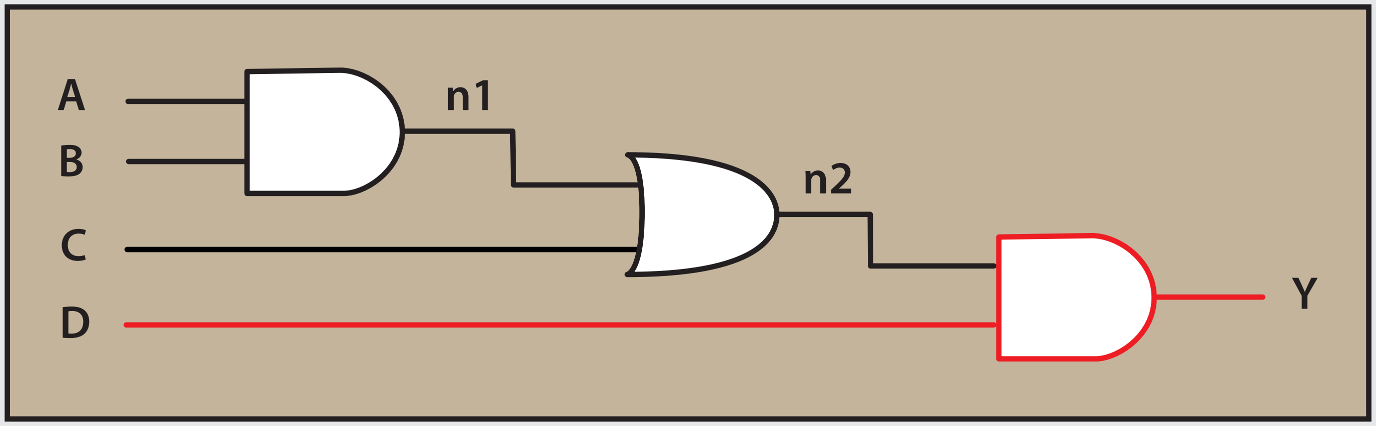




$$t_{pd} = 2\ t_{pd\_and} + t_{pd\_or}$$

Short path waveform





$t_{cd} = t_{cd\_and}$

Exercise 2

H & H Example 2.16

Given the propagation delays for the components given below,
compare the worst-case timing of the three four-input multiplexer designs.

What is the critical path for each design?

Given your timing analysis, why might you choose one design over the other ?

Gate	t_{pd} (ps)
NOT	30
2-input AND	60
3-input AND	80
4-input OR	90
tristate (A to Y)	50
tristate (enable to Y)	35