Rutgers University
School of Engineering

Fall 2022

332:231 – Digital Logic Design

Sophocles J. Orfanidis
ECE Department
orfanidi@rutgers.edu

Unit 3 – Combinational Circuits

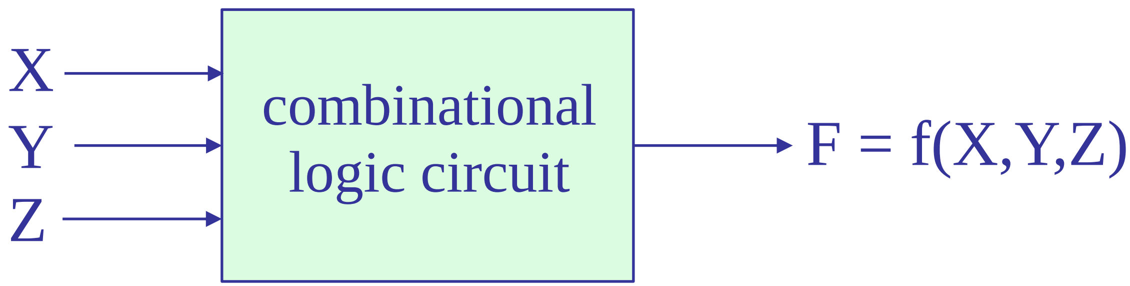
Course Topics

1. Introduction to DLD, Verilog HDL, MATLAB/Simulink
2. Number systems
3. Analysis and synthesis of combinational circuits
4. Decoders/encoders, multiplexers/demultiplexers
5. Arithmetic systems, comparators, adders, multipliers
6. Sequential circuits, latches, flip-flops
7. Registers, shift registers, counters, LFSRs
8. Finite state machines, analysis and synthesis

Text: J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018
additional references on Canvas Files > References

Analysis and Synthesis of Combinational Circuits

Analysis Problem: Given a combinational circuit made up of logic gates, determine the output F as a function of the input variables, X,Y,Z…



Synthesis/Design Problem: Given a combinational circuit defined by its I/O mapping, F = f(X,Y,Z,..), typically stated as a truth table, synthesize the circuit with logic gates, preferably using the minimum number of gates, as well as trying to minimize propagation delays.

Unit-3 Contents: (current reading:, Wakerly Chapter 3)

1. Boolean algebra, axioms, theorems, properties
2. Standard logic gates, AND, OR, NOT, NAND, NOR, XOR, XNOR
3. Operator precedence (· has higher precedence than +)
4. Duality
5. One-, two-, and three-variable theorems and their duals
6. De Morgan’s theorems, De Morgan duality
7. De Morgan’s theorems for NAND/NOR and AND/OR gates
8. NAND and NOR universal gates
9. NAND-NAND and NOR-NOR implementations
10. Bubble-to-bubble transformations, bubble-pushing operations
11. Proofs using truth tables, or using the basic theorems
12. Algebraic simplification of logic expressions

Contents, continued:

1. Boolean Algebra, Axioms, Theorems, Properties

Boolean algebra, or switching algebra, is a branch of mathematical logic in which variables take only two values:

TRUE, FALSE, or, alternatively, 1, 0, or, HIGH, LOW

It was invented by George Boole and its relevance to electrical engineering originated with Claude Shannon. See the Wikipedia resources below.

George Boole

Claude Shannon

The algebraic system is defined by certain axioms involving the AND, OR, and NOT operations and the constants 0,1.

but see also, Fuzzy Logic, in which TRUE is relative in the range (0,1]

The AND, OR, and NOT operations between any two Boolean variables X,Y are denoted by the dot $\cdot$ and plus $+$ operations, and by prime $'$ for the complement or inverse,

$$\text{AND}(X,Y) = X \cdot Y \quad = \text{logical AND}$$

$$\text{OR}(X,Y) = X + Y \quad = \text{logical OR}$$

$$\text{NOT}(X) = X' \quad = \text{logical NOT, or complement, or inverse}$$

Alternative notations are those used in computer languages, such as MATLAB and Verilog, using the symbols, **& | ~**

AND(X,Y) = X & Y = logical AND

OR(X,Y) = X | Y = logical OR

NOT(X) = ~X = logical NOT, or complement, or inverse

NOT is also denoted by an overbar, $\text{NOT}(X) = \overline{X}$

Also, the dot, $\cdot$, is often omitted in AND operations, provided it would not cause an ambiguity in the name of the variable,

for example, if A,B,C are three distinct variables, then we can write,

$$A \cdot B \cdot C = ABC$$

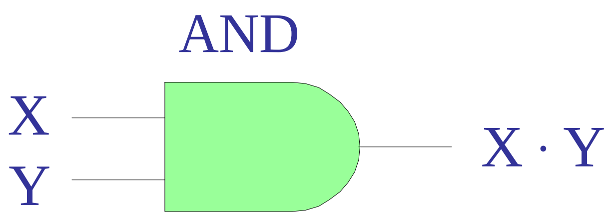
On the other hand, if AB and C are two distinct variables, then, it would be better to use the dot to avoid ambiguity, that is, write,

$$AB \cdot C$$

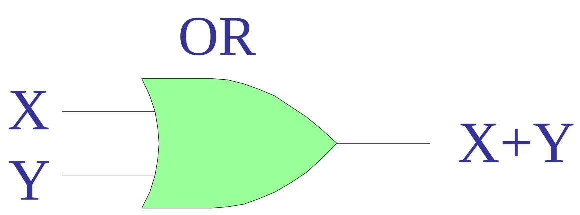
or, use parentheses, (AB)C, instead of the more ambiguous ABC

2. Standard logic gates

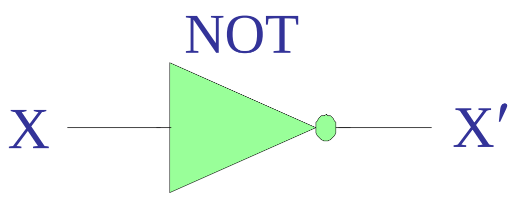
AND, OR, and NOT operations



X	Y	X · Y
0	0	0
0	1	0
1	0	0
1	1	1



X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

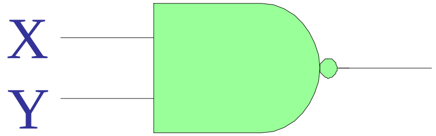


X	X′
0	1
1	0

2. Standard logic gates

NAND and NOR operations

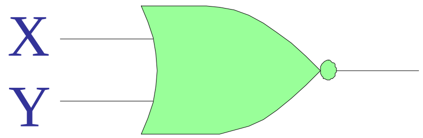
NAND



$(X \cdot Y)'$

		AND	NAND
X	Y	$X \cdot Y$	$(X \cdot Y)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

NOR



$(X+Y)'$

		OR	NOR
X	Y	$X+Y$	$(X+Y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

2. Standard logic gates

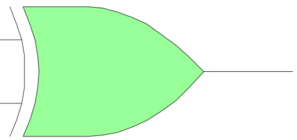
XOR and XNOR operations

exclusive OR

XOR

X

Y



$$XY' + X'Y = (X+Y)(X'+Y')$$

because, $XX' = Y'Y = 0$

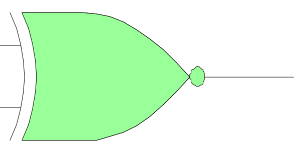
XOR

X	Y	$XY' + X'Y$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

X

Y



$$XY + X'Y' = (X+Y')(X'+Y)$$

XNOR

X	Y	$XY + X'Y'$
0	0	1
0	1	0
1	0	0
1	1	1

$(XY' + X'Y)' = XY + X'Y'$ from De Morgan

useful for testing equality

3. Operator precedence

Note: As in programming languages, “multiplication”, or AND operation, $\cdot$, has higher precedence than “addition”, or OR operation, +

e.g., $A \cdot B + C \cdot D = (A \cdot B) + (C \cdot D)$

but, use of parentheses is always recommended.

More generally, the NOT operation has higher precedence than AND and OR.

MATLAB has similar order of precedence, from higher to lower:

`~, &, |`

4. Duality

Every relationship, axiom, equation, or theorem among Boolean variables X,Y,…, and constants 0 and 1, has a dual obtained by interchanging the roles of the AND and the OR operations (i.e., interchanging, $\cdot$ and $+$), while also interchanging 0 and 1.

The dual and original relationships are not necessarily equivalent to each other, but they are both separately valid.

For example, the basic axioms relating 0 and 1 (TRUE and FALSE) come in dual pairs:

AND	OR
$0 \cdot 0 = 0$	$1 + 1 = 1$
$1 \cdot 1 = 1$	$0 + 0 = 0$
$0 \cdot 1 = 0$	$1 + 0 = 1$
$1 \cdot 0 = 0$	$0 + 1 = 1$

5. Boolean algebra theorems and their duals

<u>one-variable theorems and their duals:</u>

$X + 0 = X$	$X \cdot 1 = X$	(identities)
$X + 1 = 1$	$X \cdot 0 = 0$	(null elements)
$X + X = X$	$X \cdot X = X$	(idempotency)
$X + X' = 1$	$X \cdot X' = 0$	(complements)

$$(X')' = X$$ (involution)

two-variable and three-variable theorems and their duals

$X + Y = Y + X$	(commutative)
$(X + Y) + Z = X + (Y + Z)$	(associative)
$X \cdot A + X \cdot B = X \cdot (A + B)$	(distributive)
$X + X \cdot A = X$	(covering)
$X \cdot A + X \cdot A' = X$	(combining)
$X \cdot A + X' \cdot B = X \cdot A + X' \cdot B + \underbrace{A \cdot B}_{\text{consensus term}}$	(consensus theorem)

duals

$X \cdot Y = Y \cdot X$	(commutative)
$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	(associative)
$(X + A) \cdot (X + B) = X + (A \cdot B)$	(distributive)
$X \cdot (X + A) = X$	(covering)
$(X + A) \cdot (X + A') = X$	(combining)
$(X+A) \cdot (X' + B) = (X+A) \cdot (X' + B) \cdot \underbrace{(A+B)}_{\text{consensus term}}$	(consensus theorem)

see Wakerly/Table 3-3 for *n*-variable theorems and their duals

Boole/Shannon expansion theorem:

$$F(X,Y,Z) = X \cdot F(1,Y,Z) + X' \cdot F(0,Y,Z)$$

dual $\downarrow$

$$F(X,Y,Z) = [\, X + F(0,Y,Z)\,] \cdot [\, X' + F(1,Y,Z)\,]$$

Example:

$$F(X,Y,Z) = X \cdot Y + Y \cdot Z + Z \cdot X$$

$$F(1,Y,Z) \;= Y + Y \cdot Z + Z = Y + Z$$

$$F(0,Y,Z) \;= Y \cdot Z$$

$$F(X,Y,Z) = X \cdot F(1,Y,Z) + X' \cdot F(0,Y,Z) = X \cdot (Y+Z) + X' \cdot (Y \cdot Z)$$

$$F(X,Y,Z) = [X + F(0,Y,Z)] \cdot [X' + F(1,Y,Z)] =$$
$$= [X + (Y \cdot Z)] \cdot [X' + Y + Z]$$

Wikipedia - Boole/Shannon expansion theorem additional clarification & direct derivation:

note: $X + X' = 1$

$$
\begin{aligned}
F(X,Y,Z) &= XY + YZ + ZX \\
&= XY + (X + X')YZ + ZX \\
&= X(Y + YZ + Z) + X'YZ
\end{aligned}
$$

$$F(1,Y,Z) = Y + YZ + Z = Y + Z$$

$$F(0,Y,Z) = YZ$$

$$
\begin{aligned}
Y + YZ + Z &= Y + YZ + YZ + Z = \\
&= Y(1 + Z) + (Y + 1)Z = \\
&= Y + Z
\end{aligned}
$$

6. De Morgan’s theorems, De Morgan duality

De Morgan's laws - Wikipedia

$$(X \cdot Y)' = X' + Y' \qquad \text{(NAND)}$$

$$(X + Y)' = X' \cdot Y' \qquad \text{(NOR)}$$

$$X \cdot Y = (X' + Y')' \qquad \text{(AND)}$$

$$X + Y = (X' \cdot Y')' \qquad \text{(OR)}$$

Generalized De Morgan theorems – De Morgan duality:

The complement of an expression is the dual of the expression with all variables replaced by their complements, or, equivalently, the expression is equal to the complement of its dual with all variables complemented,

$$F(X,Y,Z,\ldots)' = F_{dual}(X',Y',Z',\ldots)$$

$$F(X,Y,Z,\ldots) = F_{dual}(X',Y',Z',\ldots)'$$

De Morgan examples

$$[X + (Y \cdot Z)]' = X' \cdot (Y' + Z')$$

$$[X \cdot (Y + Z)]' = X' + (Y' \cdot Z')$$

$$(X + Y + Z)' = X' \cdot Y' \cdot Z'$$

$$(X \cdot Y \cdot Z)' = X' + Y' + Z'$$

De Morgan duality:

$$F(X,Y,Z) = X + (Y \cdot Z), \qquad F_{dual}(X,Y,Z) = X \cdot (Y + Z)$$

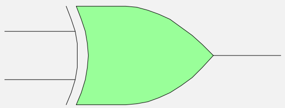
$$F(X,Y,Z)' = [X + (Y \cdot Z)]' = X' \cdot (Y' + Z') = F_{dual}(X',Y',Z')$$

$$F(X,Y,Z) = X \cdot (Y + Z), \qquad F_{dual}(X,Y,Z) = X + (Y \cdot Z)$$

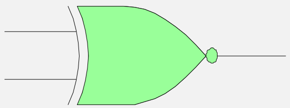
$$F(X,Y,Z)' = [X \cdot (Y + Z)]' = X' + (Y' \cdot Z') = F_{dual}(X',Y',Z')$$

Using De Morgan's theorems, show that, XOR$'$ = XNOR, defined by,

$$\mathrm{XOR}(X,Y) = X \cdot Y' + X' \cdot Y = (X + Y) \cdot (X' + Y')$$



$$\mathrm{XNOR}(X,Y) = X \cdot Y + X' \cdot Y' = (X + Y') \cdot (X' + Y)$$



$$\begin{aligned} \mathrm{XOR}' &= (X \cdot Y' + X' \cdot Y)' \\ &= (X \cdot Y')' \cdot (X' \cdot Y)' \\ &= (X' + Y) \cdot (X + Y') \\ &= X \cdot Y + X' \cdot Y' \\ &= \mathrm{XNOR} \end{aligned}$$

note, XOR, XNOR are invariant under the substitutions:

$$X \rightarrow X'$$
$$Y \rightarrow Y'$$

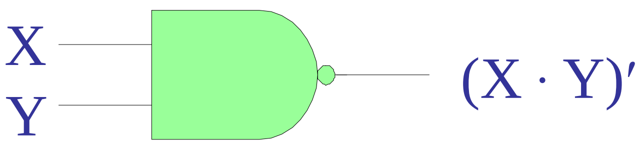
or more simply, using De Morgan duality:

$$\mathrm{XOR}_{\mathrm{dual}}(X,Y) = (X + Y') \cdot (X' + Y)$$

$$\mathrm{XOR}(X,Y)' = \mathrm{XOR}_{\mathrm{dual}}(X',Y') = (X' + Y) \cdot (X + Y') = \mathrm{XNOR}(X,Y)$$

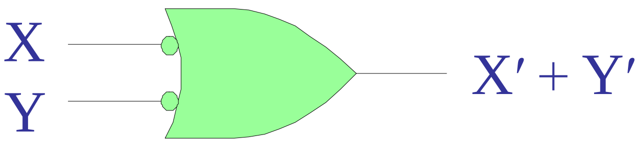
7. De Morgan's theorems for NAND and NOR gates

$$(X \cdot Y)' = X' + Y'$$
$$\text{NAND}(X,Y) = \text{OR}(X',Y')$$



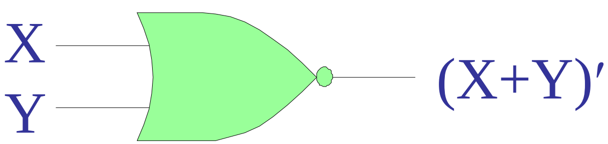
NAND

≡



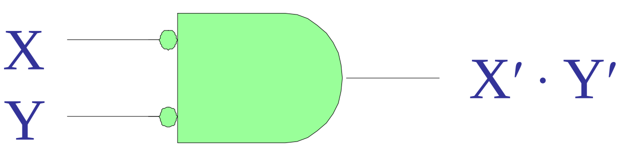
NAND equivalent using OR

$$(X+Y)' = X' \cdot Y'$$
$$\text{NOR}(X,Y) = \text{AND}(X',Y')$$



NOR

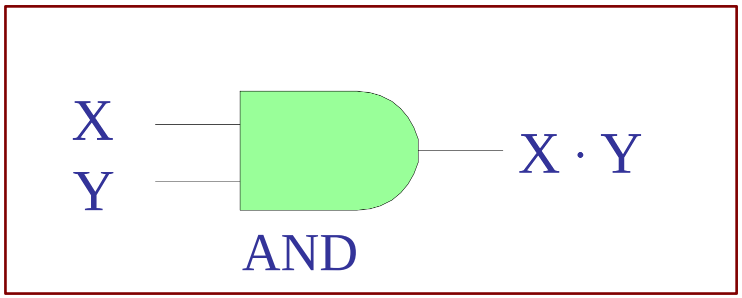
≡



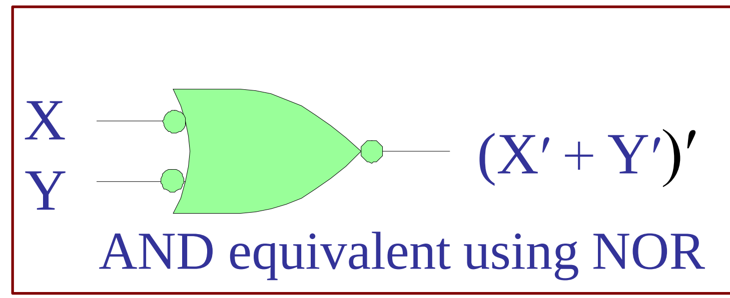
NOR equivalent using AND

7. De Morgan's theorems for AND and OR gates

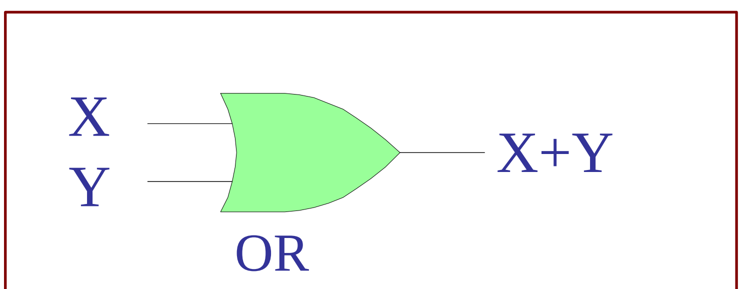
$$X \cdot Y = (X' + Y')'$$
$$\text{AND}(X,Y) = \text{NOR}(X',Y')$$



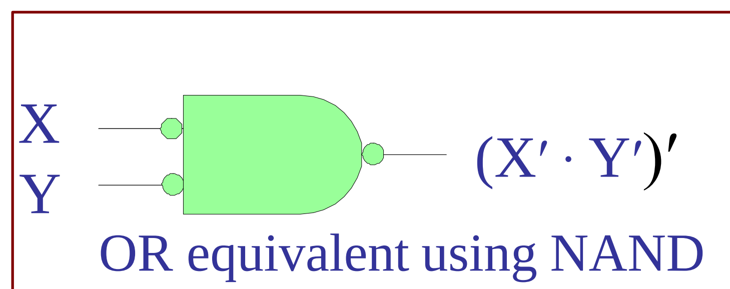
≡



$$X+Y = (X' \cdot Y')'$$
$$\text{OR}(X,Y) = \text{NAND}(X',Y')$$



≡

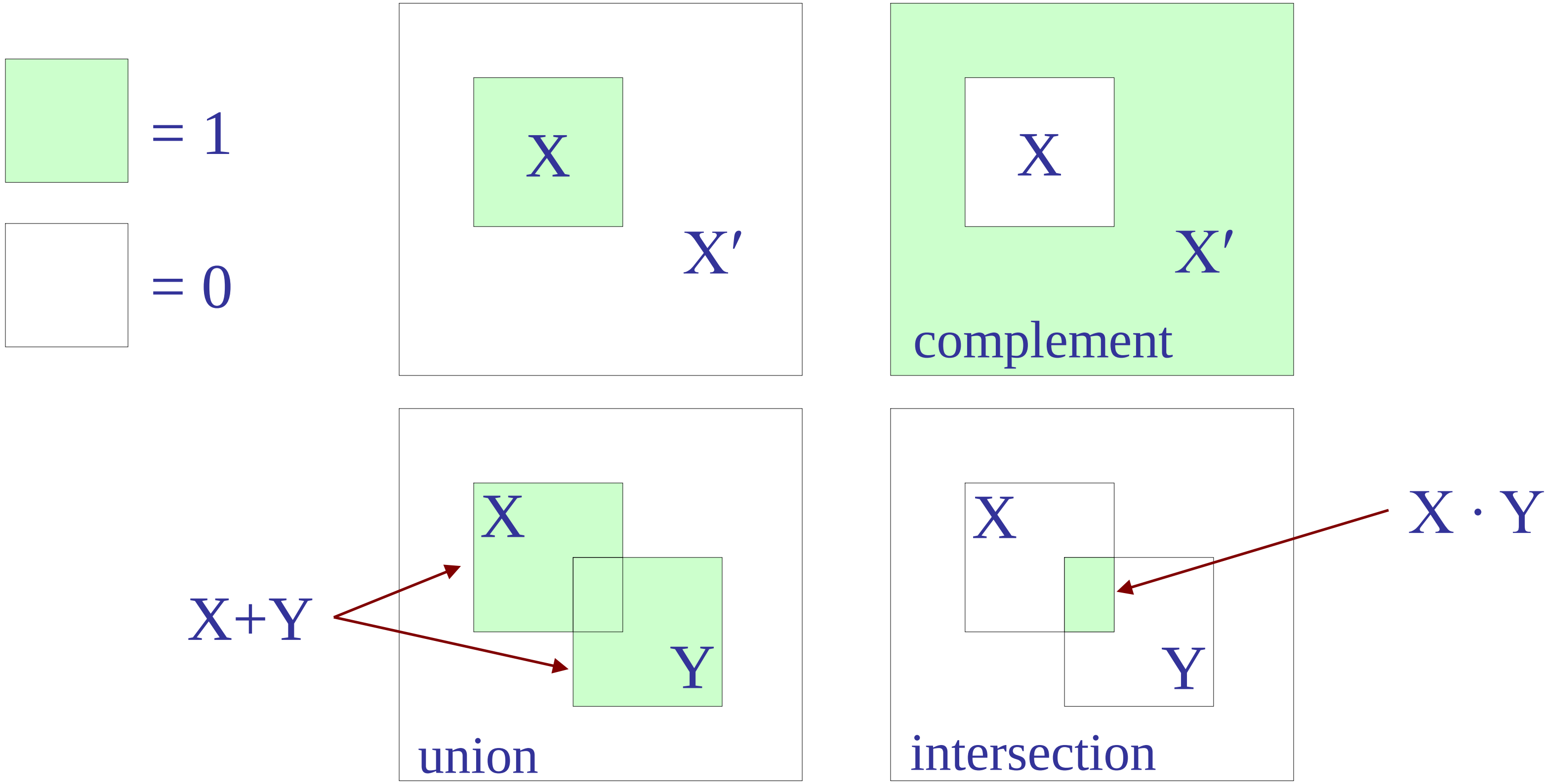


De Morgan's laws for sets

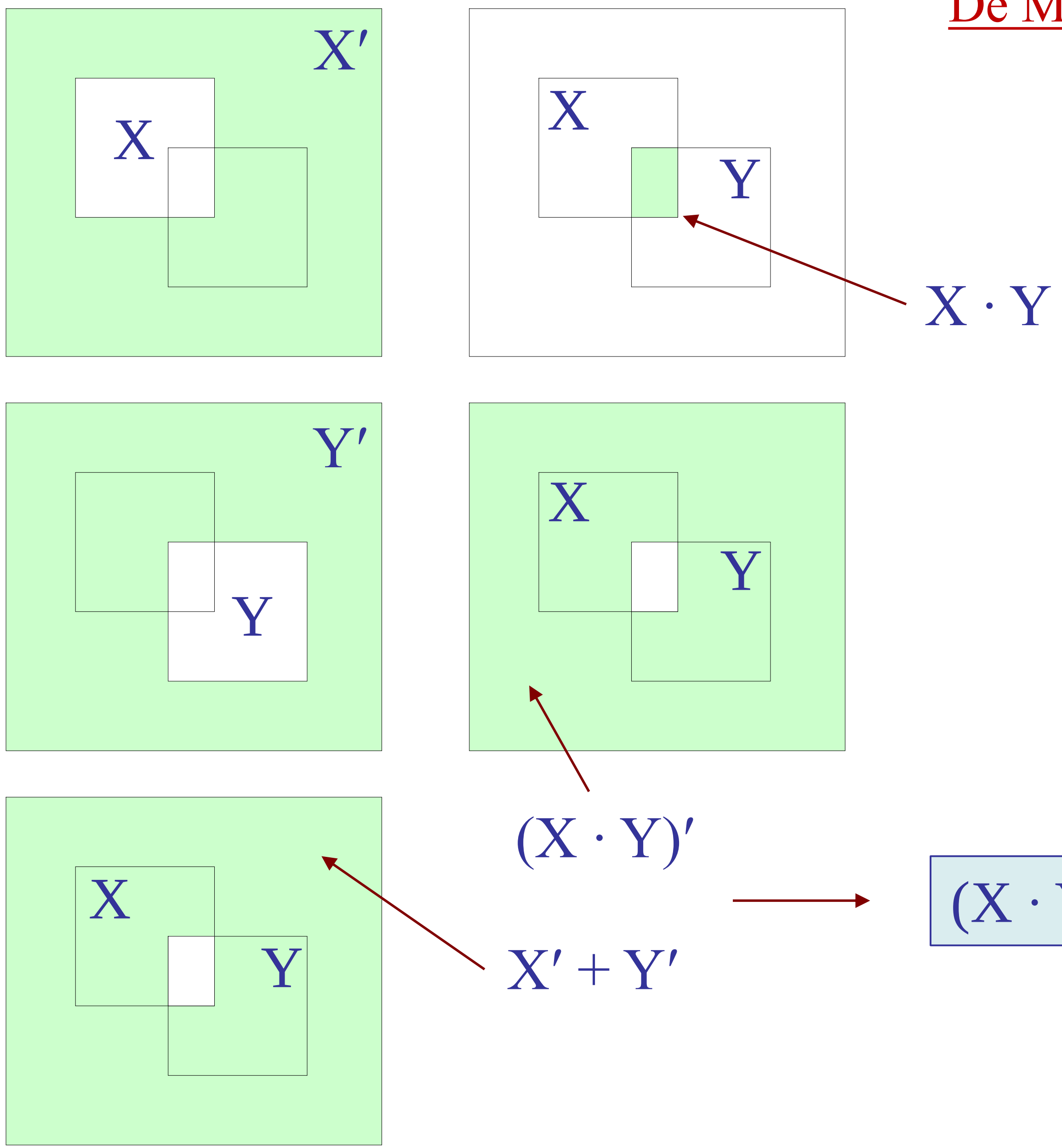
De Morgan's laws - Wikipedia

The Boolean algebra equivalents of the AND, OR, and NOT operations in the theory of sets are the Union and Intersection of two sets, and the Complement of a set.

They can be visualized with Venn diagrams.

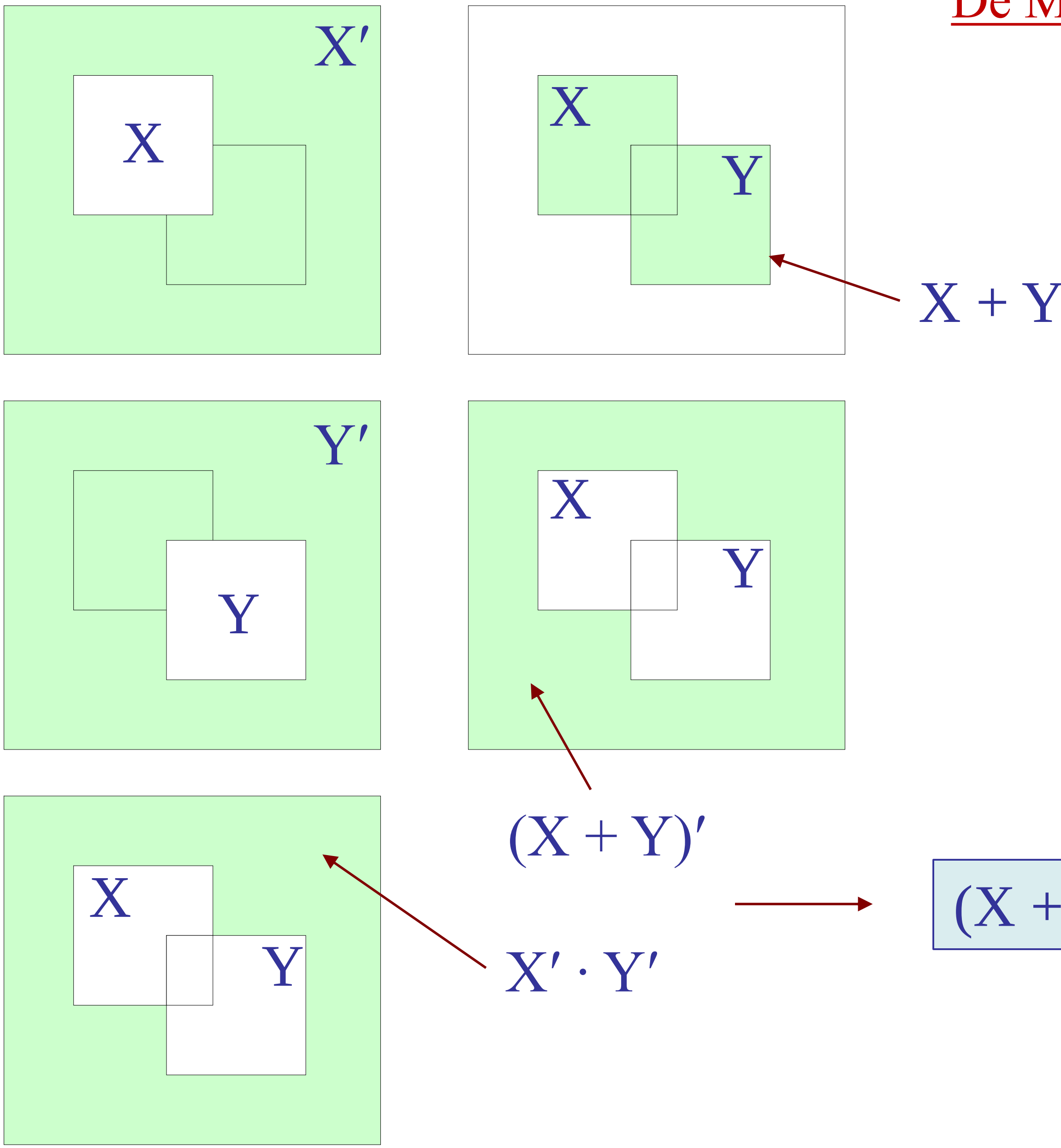


De Morgan's laws for sets



$$\boxed{(X \cdot Y)' = X' + Y'}$$

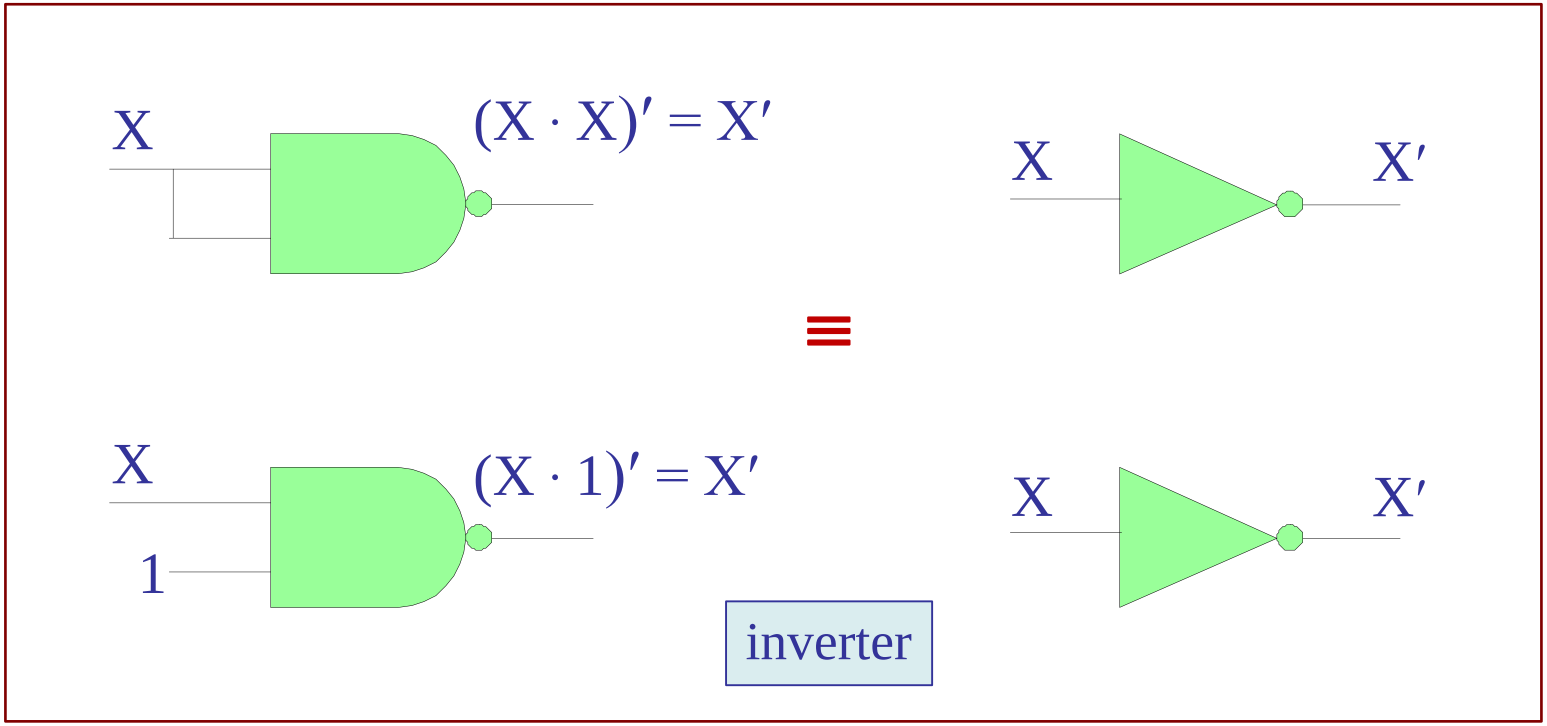
De Morgan’s laws for sets



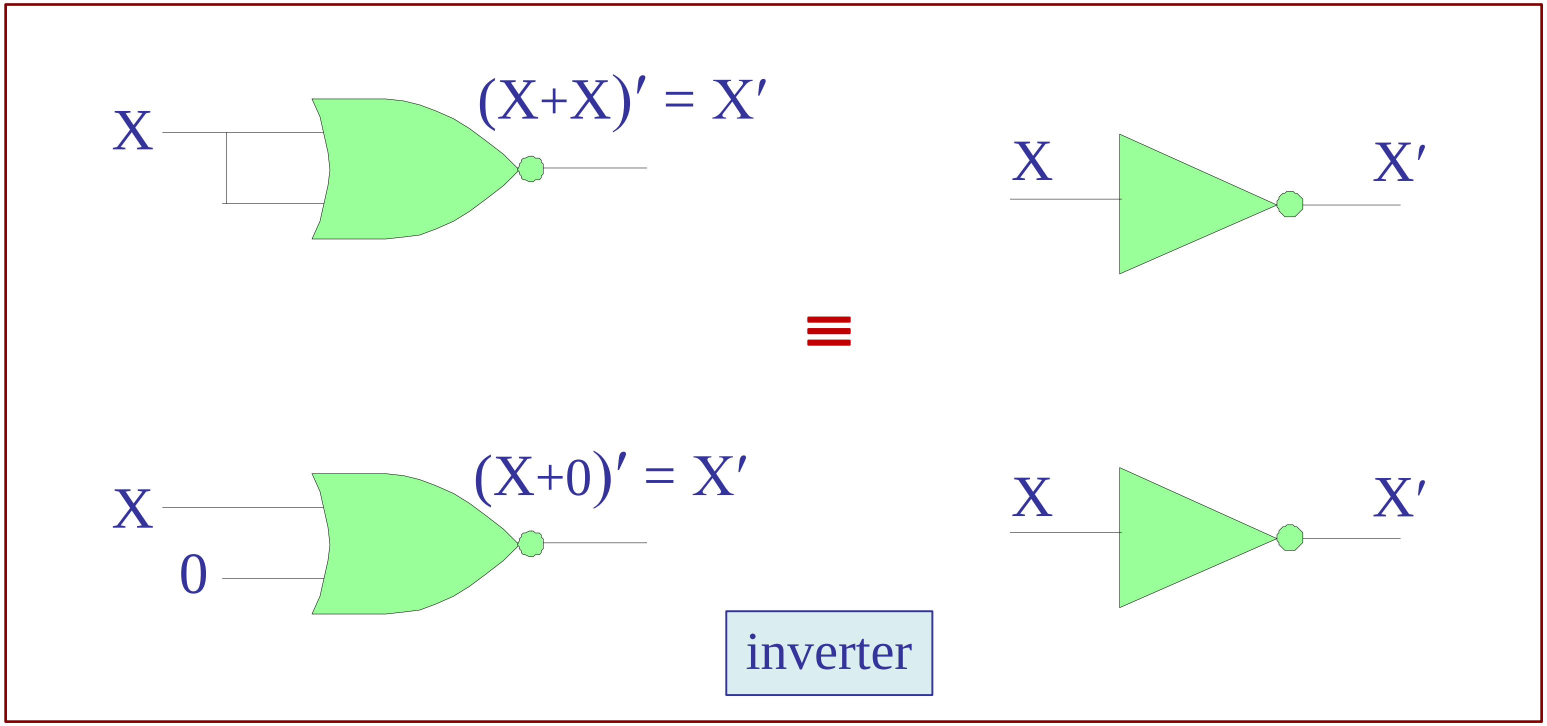
$$\boxed{(X + Y)' = X' \cdot Y'}$$

8. NAND and NOR universal gates

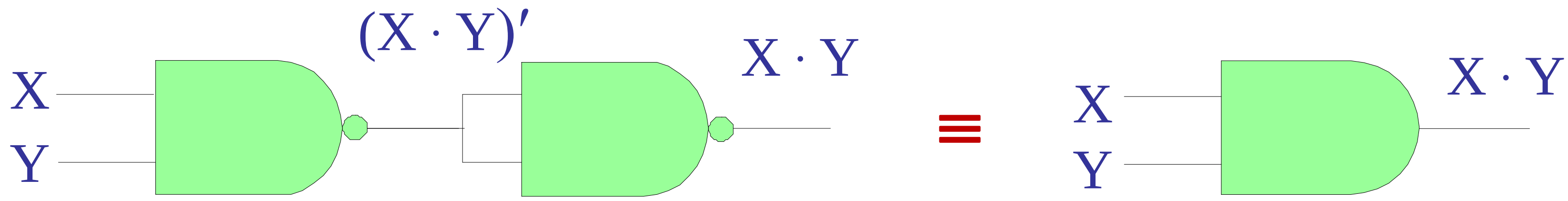
NAND and NOR gates are universal gates in the sense they can be used to build any other type of gate – they are preferred in all IC logic families because they are fast, economical, and easy to fabricate.



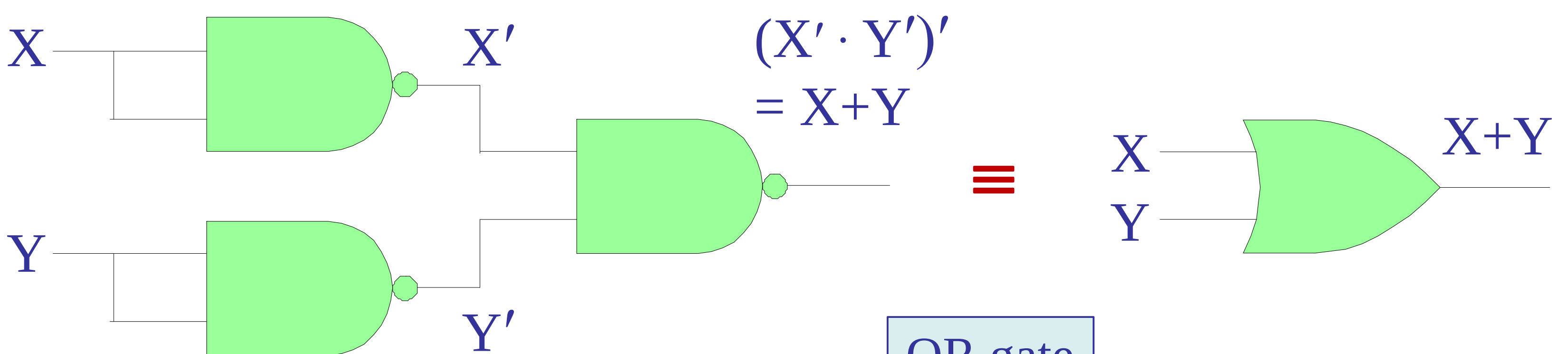
8. NAND and NOR universal gates



8. NAND and NOR universal gates

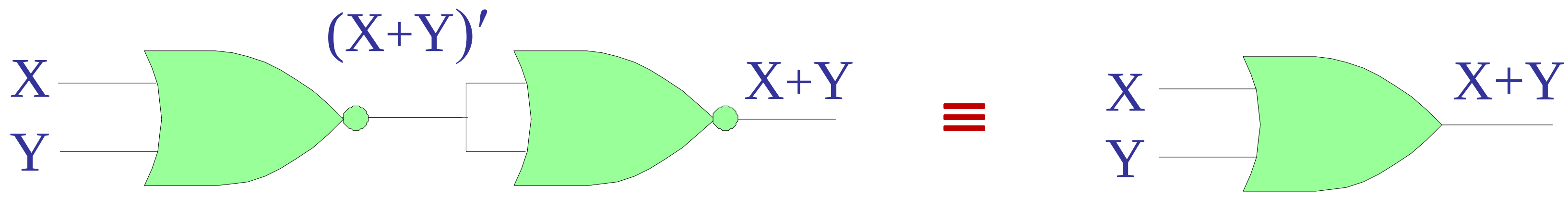


AND gate

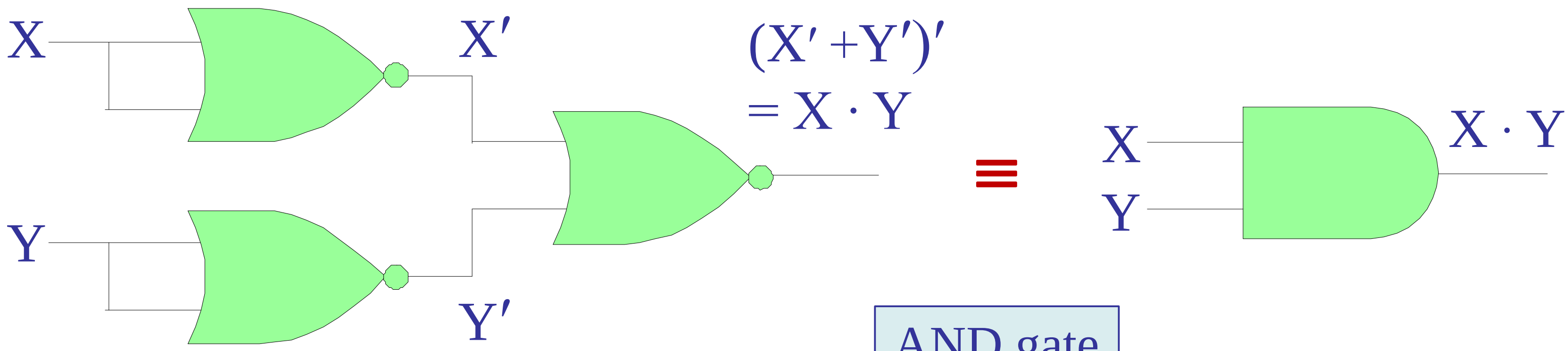


OR gate

8. NAND and NOR universal gates



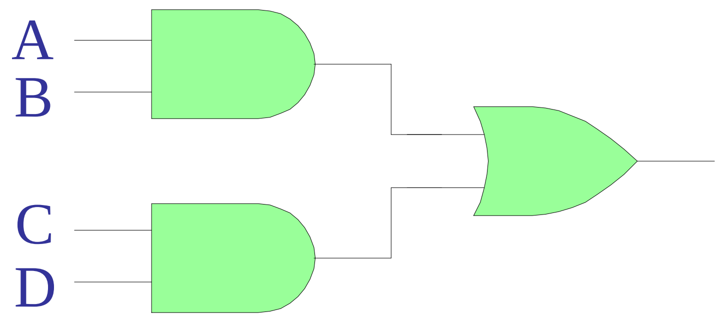
OR gate



AND gate

9. NAND–NAND and NOR–NOR implementations

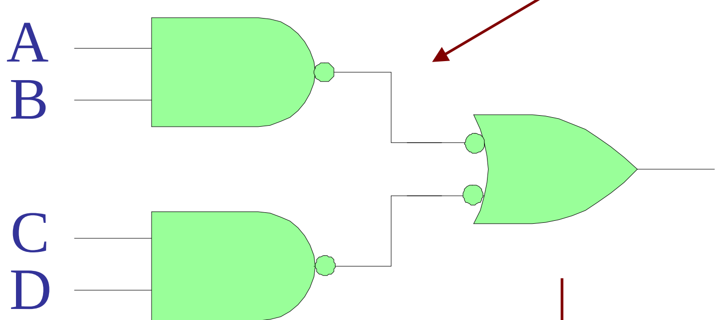
10. Bubble-to-bubble transformations, bubble pushing operations



$F = AB + CD$

AND – OR implementation

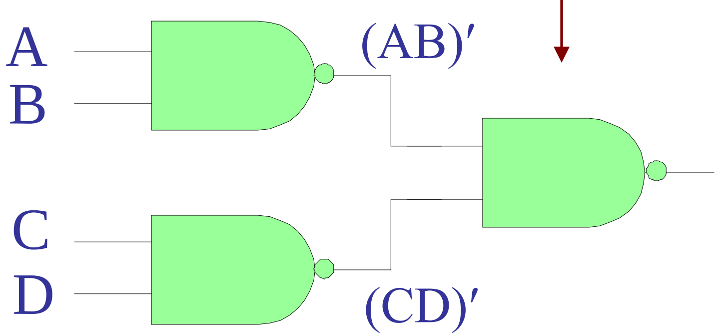
bubbles cancel on same line, double-NOT



$F = (AB)'' + (CD)''$

De Morgan's theorem

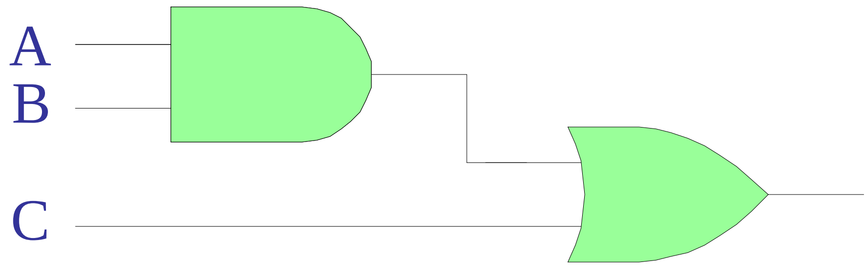
$(X \cdot Y)' = X' + Y'$



$F = \big((AB)' \cdot (CD)'\big)'$

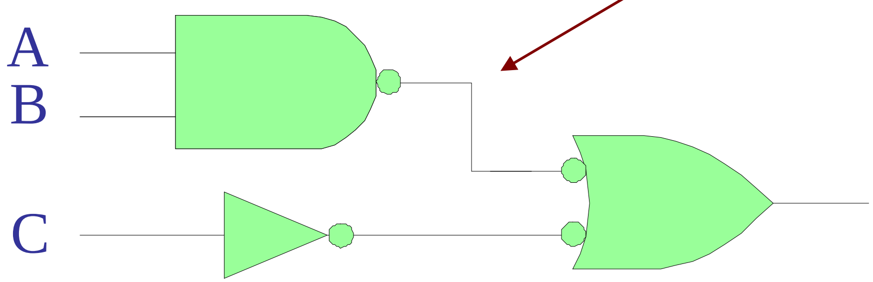
NAND – NAND implementation

NAND–NAND realizations
bubble-to-bubble transformations
bubble pushing operations



$F = AB + C$

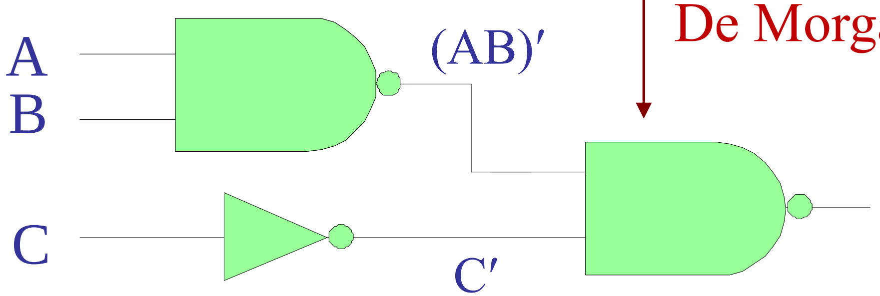
bubbles cancel on same line,
double-NOT



$F = (AB)'' + (C)''$

De Morgan's theorem

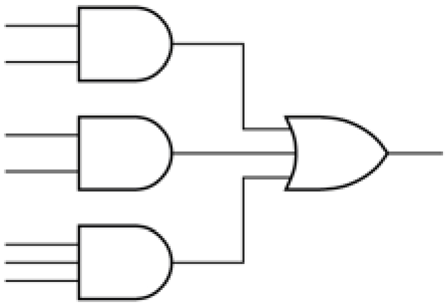
$(X \cdot Y)' = X' + Y'$



$F = [(AB)' \cdot (C)']'$

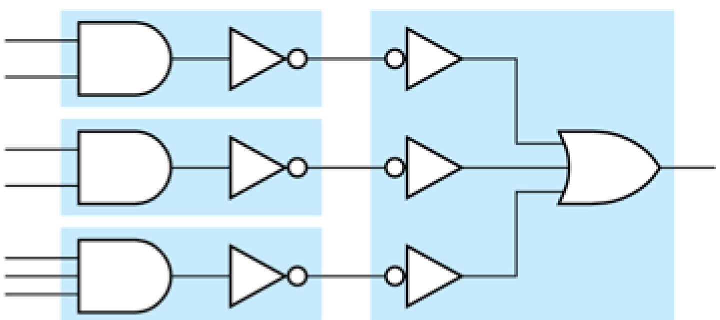
more examples from Wakerly

NAND–NAND realizations
bubble-to-bubble transformations
bubble pushing operations

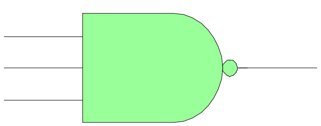


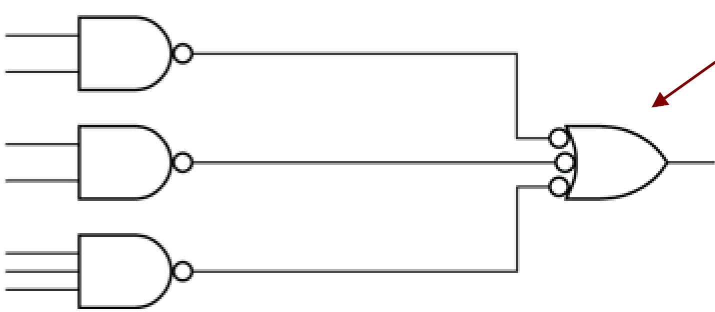
sum-of-products (SOP) form

bubbles cancel on same line



NAND gate

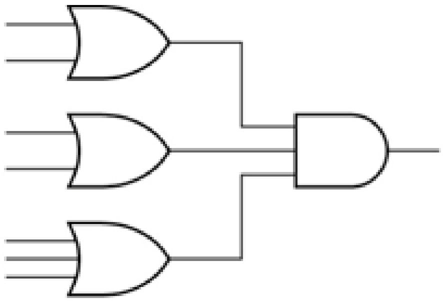




De Morgan equivalents

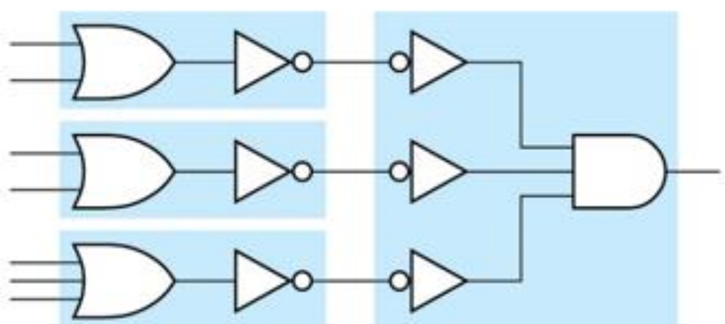
more examples from Wakerly

NOR–NOR realizations
bubble-to-bubble transformations
bubble pushing operations

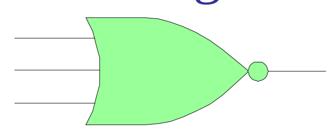


product-of-sums (POS) form

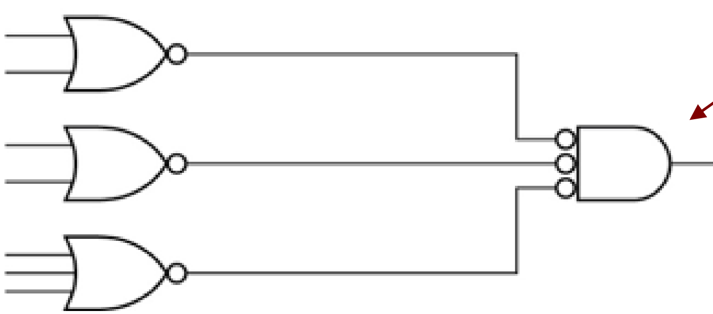
bubbles cancel on same line



NOR gate



De Morgan equivalents



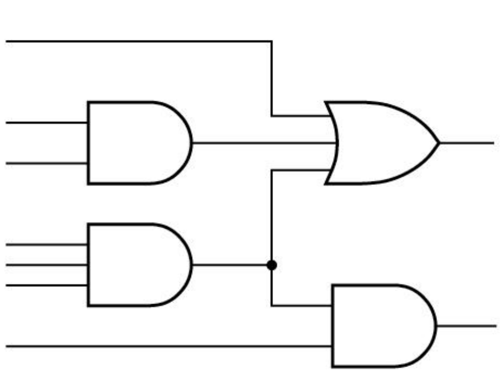
NOR gates

more examples from Wakerly

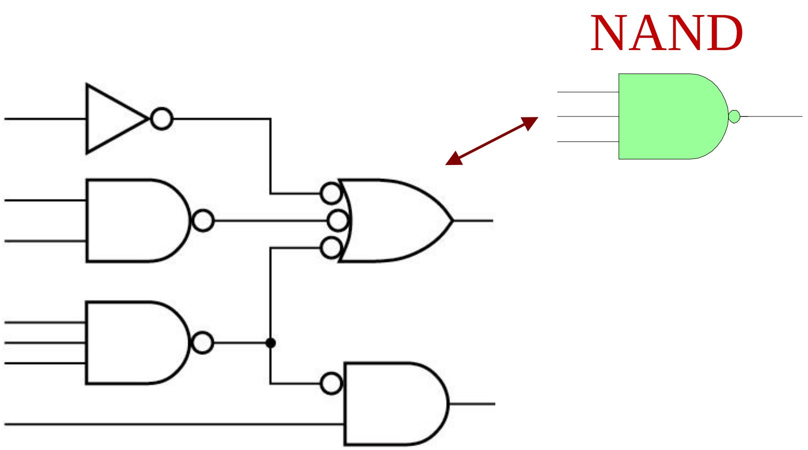
(a) original
(b) non-standard gate
(c) eliminate non-standard gate
(d) preferred inverter placement

NAND–NOR realizations
bubble-to-bubble transformations
bubble pushing operations

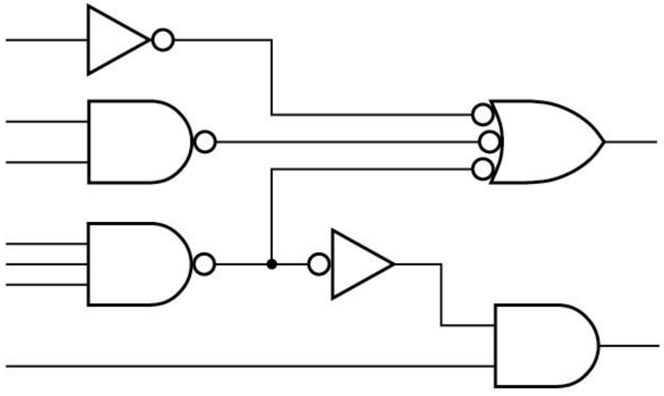
(a)



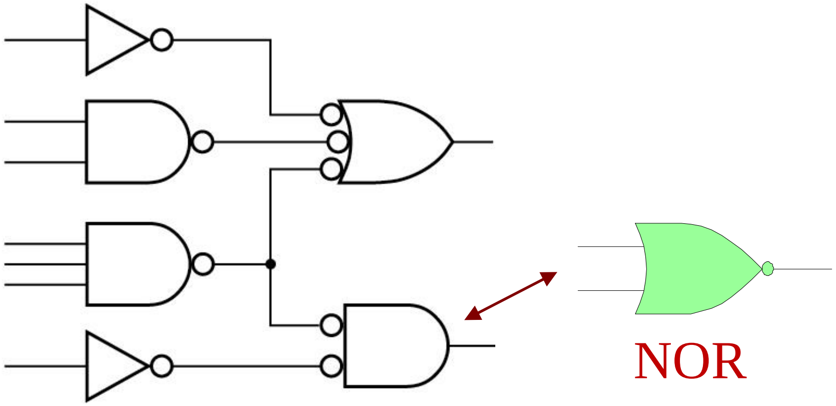
(b)



(c)



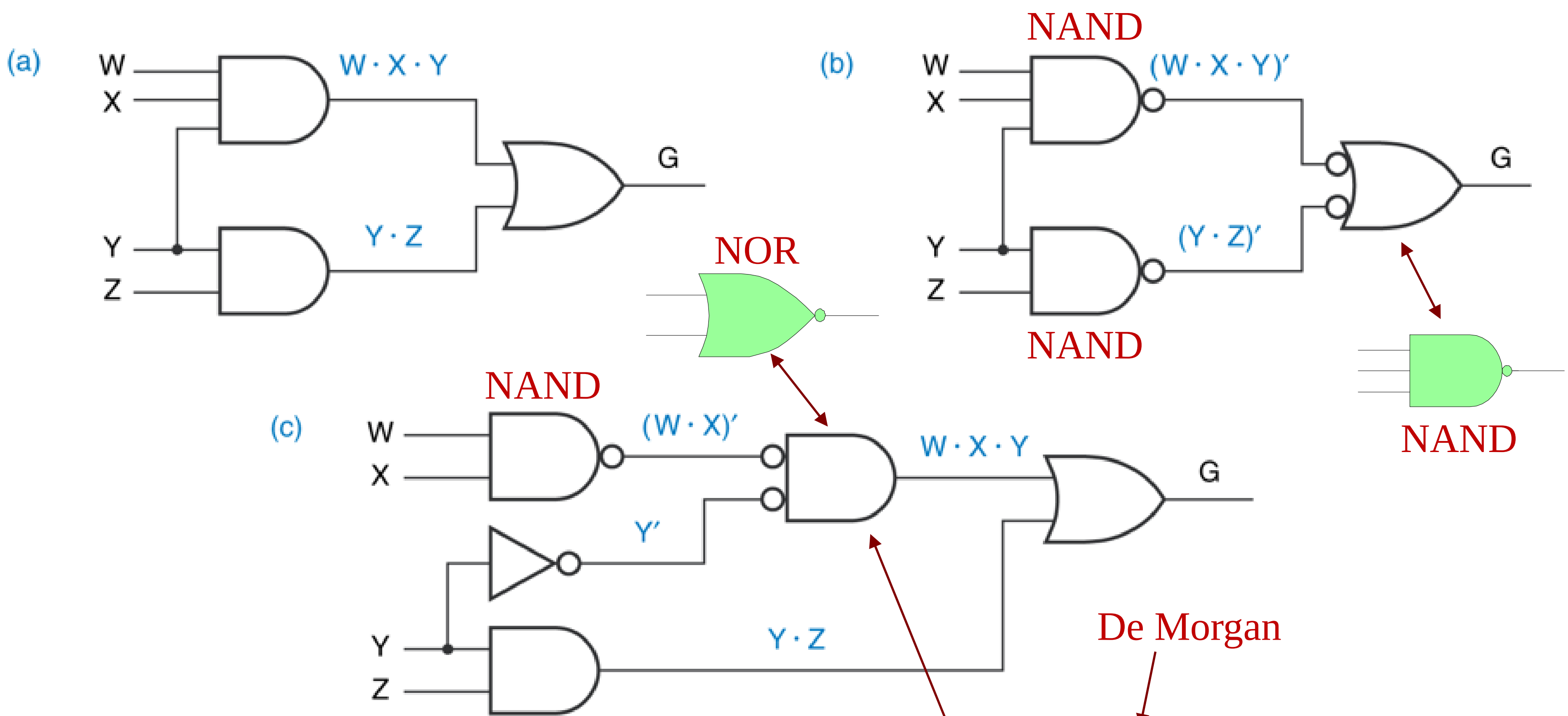
(d)



more examples from Wakerly

(a) two-level AND-OR
(b) two-level NAND-NAND
(c) 2-input gates only

NAND–NAND realizations
bubble-to-bubble transformations
bubble pushing operations



$G = W \cdot X \cdot Y + Y \cdot Z$

$W \cdot X \cdot Y = (W \cdot X)'' \cdot (Y)'' = ((WX)' + Y')'$

11. Boolean theorem proofs

Proofs of the various Boolean theorems can be given by simply verifying the truth tables of the two sides of the expressions.

For example, to prove the covering theorem, $X + X \cdot Y = X$, we may evaluate both sides of the expression for all possible values of the Boolean variables X,Y, making a truth table for each side:

X	Y	$X \cdot Y$	$X + X \cdot Y$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal

Alternative analytical proof:

$$
\begin{aligned}
X + X \cdot Y &= X \cdot (1+Y) \\
&= X \cdot 1 \\
&= X
\end{aligned}
$$

note, $1+Y = 1$

As another example, in order to prove the distributive theorem,

$$X + Y \cdot Z = (X + Y) \cdot (X + Z)$$

we construct the truth tables for each side:

X Y Z	$Y \cdot Z$	$X + Y \cdot Z$	$X + Y$	$X + Z$	$(X + Y) \cdot (X + Z)$
0 0 0	0	0	0	0	0
0 0 1	0	0	0	1	0
0 1 0	0	0	1	0	0
0 1 1	1	1	1	1	1
1 0 0	0	1	1	1	1
1 0 1	0	1	1	1	1
1 1 0	0	1	1	1	1
1 1 1	1	1	1	1	1

3-bit binary pattern

Analytical proof of the distributive theorem,

$$X + Y \cdot Z = (X + Y) \cdot (X + Z)$$

using the covering theorem, $A + A \cdot B = A$, and the idempotent theorem, $A \cdot A = A$,

$$(X + Y) \cdot (X + Z) = X \cdot X + X \cdot Y + X \cdot Z + Y \cdot Z$$

$$= X + X \cdot Y + X \cdot Z + Y \cdot Z$$

$$= X + X \cdot Y + X \cdot Z + Y \cdot Z$$

$$= X + X \cdot Z + Y \cdot Z$$

$$= X + Y \cdot Z$$

MATLAB proof of the distributive theorem,

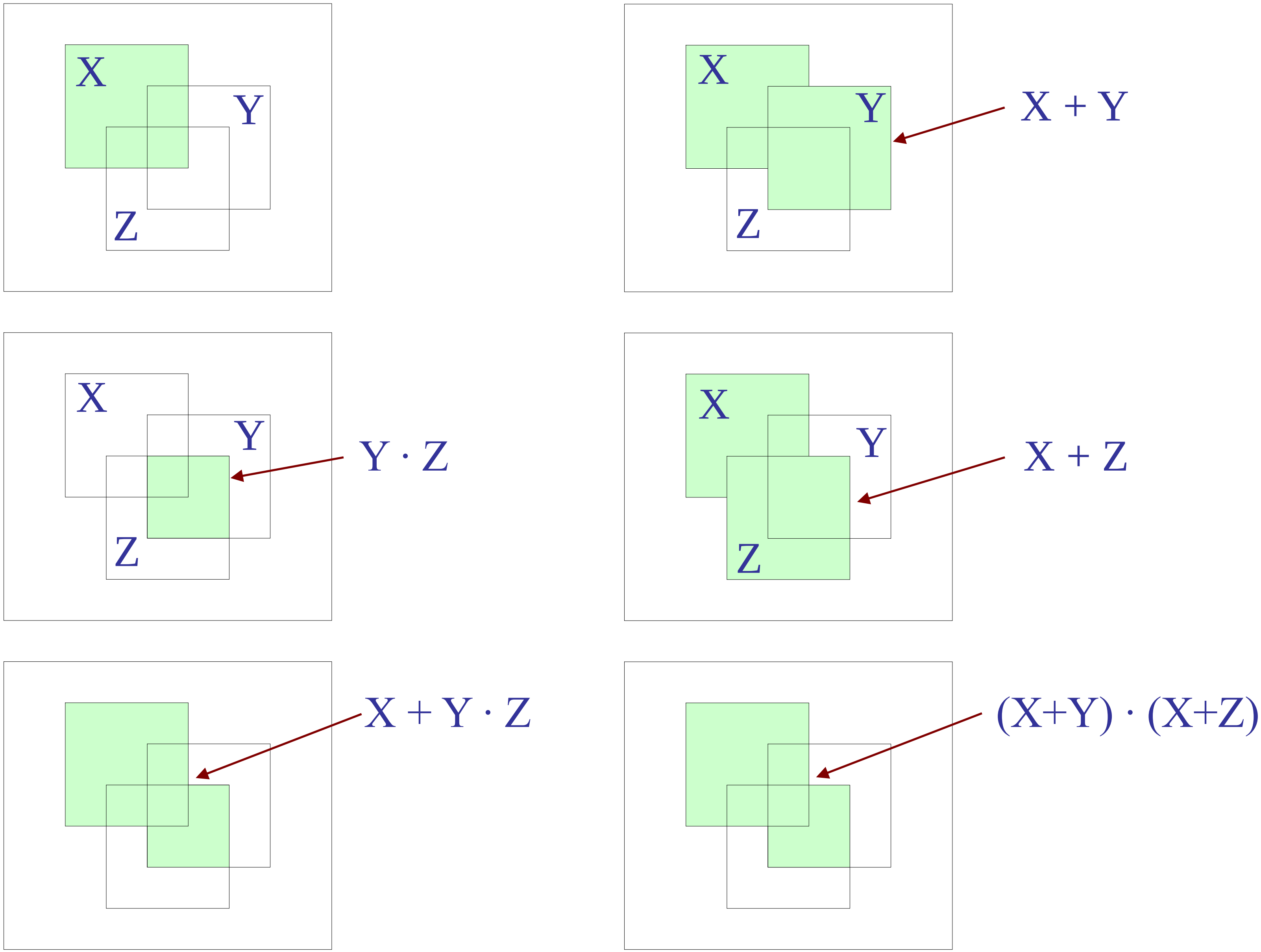
$$X + Y \cdot Z = (X + Y) \cdot (X + Z)$$

MATLAB code:

```
[X,Y,Z] = a2d(0:7,3);      % generate inputs
F1 = X | (Y&Z);            % left-hand side
F2 = (X|Y) & (X|Z);        % right-hand side
[X,Y,Z,F1,F2]              % print columns
```

X	Y	Z	F1	F2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Set-theoretic proof of the distributive theorem, $X + Y \cdot Z = (X + Y) \cdot (X + Z)$



Proofs of the consensus theorems:

note: $X + X' = 1$

$$
\begin{aligned}
X \cdot A + X' \cdot B + A \cdot B &= X \cdot A + X' \cdot B + (X + X') \cdot A \cdot B \\
&= X \cdot A + X \cdot A \cdot B + X' \cdot B + X' \cdot A \cdot B \\
&= X \cdot A \cdot (1+B) + X' \cdot B \cdot (1+A) \\
&= X \cdot A \cdot 1 + X' \cdot B \cdot 1 \\
&= X \cdot A + X' \cdot B
\end{aligned}
$$

dual

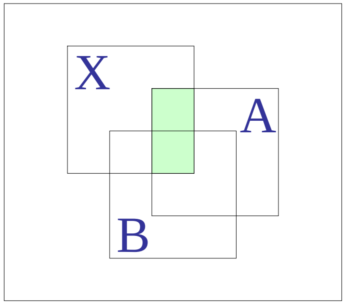
$$
\begin{aligned}
(X + A)(X' + B)(A+B) &= (X X' + XB + X'A + AB)(A+B) \\
&= (XB + X'A + AB)(A+B) \\
&= XB(A+B) + X'A(A+B) + AB(A+B) \\
&= X(BA+BB) + X'(AA+AB) + AAB+BBA \\
&= X(BA+B) + X'(A+AB) + AB+BA \\
&= XB(A+1) + X'A(1+B) + AB \\
&= XB + X'A + AB \\
&= X X' + XB + X'A + AB \\
&= (X+A) \cdot (X' +B)
\end{aligned}
$$

note: $AA = A$ $\longrightarrow$

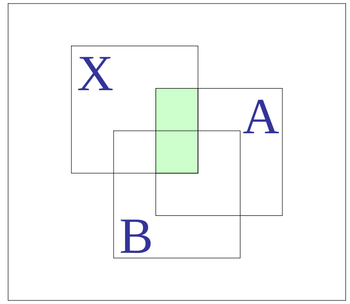
note: $A + 1 = 1$ $\longrightarrow$

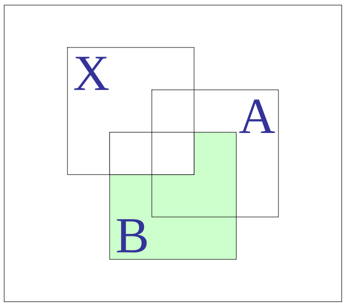
note: $XX' = 0$ $\longrightarrow$

Set-theoretic proof
of the consensus theorem

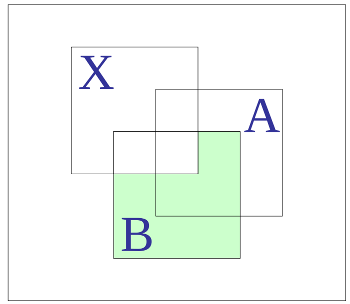


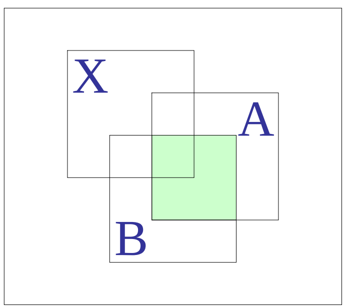
$X \cdot A$



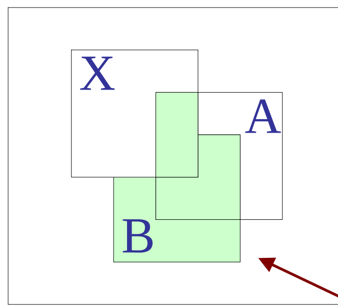


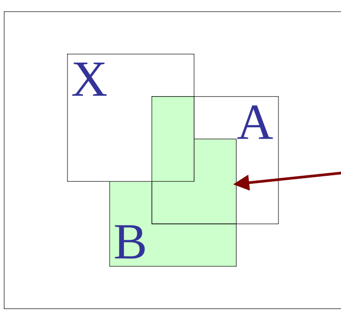
$X' \cdot B$





$A \cdot B$





$$X \cdot A + X' \cdot B + A \cdot B = X \cdot A + X' \cdot B$$

12. Algebraic simplification of combinational logic expressions

Next, we discuss a few examples of using the Boolean properties and theorems to simplify logic expressions.

Simplification leads to more efficient implementations requiring fewer logic gates.

Although, the theorems can always be used to simplify an expression, a much better and easier approach is through the use of Karnaugh maps (K-maps) – they will be discussed in detail later on.

An additional requirement in designing logic circuits is the minimization of propagation delays through the various stages (or, levels) of the realization.

It should be noted, however, that minimizing the number of logic gates does not necessarily guarantee shorter propagation delays.

Nomenclature

literal: a single Boolean variable, e.g., X

product term: a product of variables, e.g., XZ

sum term: a sum of variables, e.g., Y+Z

minterm: a product of input variables corresponding to a row in truth table
e.g., $XY'Z$, corresponding to row 101 = 5

canonical SOP: canonical minterm sum-of-products, e.g., $\Sigma_{X,Y,Z}(0,3,4,6,7)$

minterm list : list of row numbers that appear in a canonical SOP

maxterm: a sum of input variables, e.g., $X+Y'+Z$,
corresponding to the complement of a row in the truth table

canonical POS: canonical maxterm product-of-sums, e.g., $\Pi_{X,Y,Z}(1,2,5)$

maxterm list : list of row numbers that appear in a canonical POS

to be explained further later on

Nomenclature

implicant: a minterm or sum of minterms appearing in a function F, if an implicant evaluates to 1, then so does F as a whole, i.e., if, implicant=1, then it implies, F=1

prime implicant: a simplified implicant that cannot be combined into another implicant that has fewer number of literals.

covers: all implicants that account for all possible evaluations of the function into F=1 (i.e., all the 1's in a Karnaugh map).

essential prime implicant: a prime implicant that contains an F=1 minterm that is not included in any other prime implicant,

all essential prime implicants must be included in the cover of the function.

In addition to the essential PIs, it may be necessary to include possible non-essential PIs in order to achieve a complete cover, (if there are several such possibilities, one could choose the one that has the smallest number of literals.

Nomenclature

Sum-of-Products (SOP) rule:

F is the sum of those minterms that correspond
to the values F=1 in the truth table

Notation: $F = \Sigma$(of the F=1 minterms) = canonical sum-of-products

as indicated by the row numbers in the truth table

Product-of-Sums (POS) rule:

F is the product of those maxterms that correspond
to the values F=0 (or, $F' = 1$) in the truth table

Notation: $F = \Pi$(of the F=0 maxterms) = canonical product-of-sums

as indicated by the row numbers in the truth table

Truth-table representations with minterms or maxterms

3-variable logic function F(X,Y,Z)						
row	X	Y	Z	F	minterms	maxterms
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

complements of each other
by De Morgan

12. Algebraic simplification of combinational logic expressions

Example 1: Prove the dual results,

$$X + X' \cdot Y = X + Y$$

$$X \cdot (X' + Y) = X \cdot Y$$

Proof: Using, $A + A' = 1$, and the distributive property:

$$\boxed{A + B \cdot C = (A+B) \cdot (A+C)}$$

with, $A = X$, $B = X'$, $C = Y$, we have,

$$X + X' \cdot Y = (X+X') \cdot (X+Y) = 1 \cdot (X+Y) = X+Y$$

For the dual, we simply multiply the terms out,

$$X \cdot (X' + Y) = X \cdot X' + X \cdot Y = 0 + X \cdot Y = X \cdot Y$$

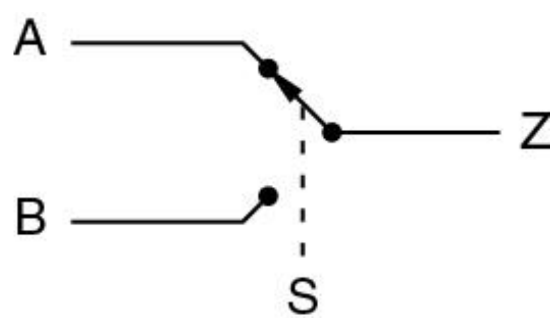
Example 2: Truth table of multiplexer function from Unit-1

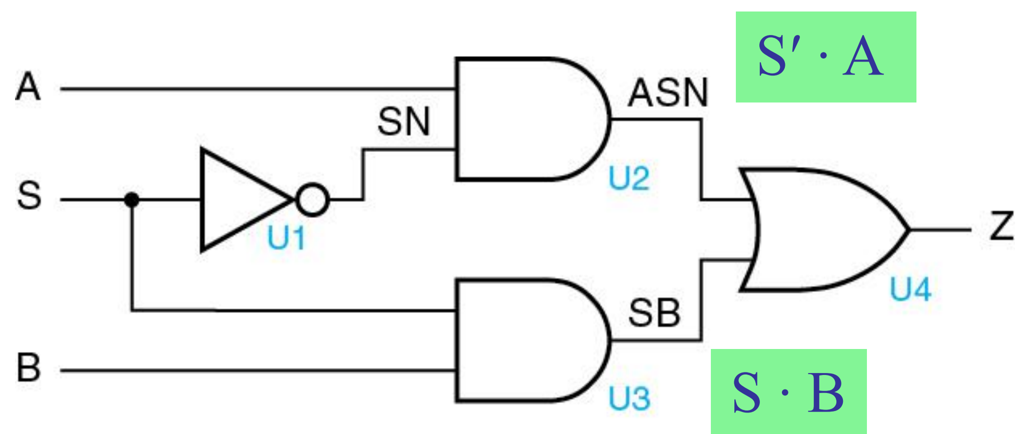
$$Z = S' \cdot A \cdot B' + S' \cdot A \cdot B + S \cdot A' \cdot B + S \cdot A \cdot B$$
$$= S' \cdot A \cdot (B' + B) + S \cdot (A' + A) \cdot B$$
$$= S' \cdot A \cdot 1 + S \cdot 1 \cdot B$$
$$= S' \cdot A + S \cdot B$$

minterm SOP

$$Z = \Sigma_{S,A,B}(2,3,5,7)$$

	rows	S	A	B	Z
	0	0	0	0	0
	1	0	0	1	0
$S' \cdot A \cdot B'$	2	0	1	0	1
$S' \cdot A \cdot B$	3	0	1	1	1
	4	1	0	0	0
$S \cdot A' \cdot B$	5	1	0	1	1
	6	1	1	0	0
$S \cdot A \cdot B$	7	1	1	1	1





Example 2: Truth table of multiplexer function from Unit-1

$$Z' = S' \cdot A' \cdot B' + S' \cdot A' \cdot B + S \cdot A' \cdot B' + S \cdot A \cdot B'$$

De Morgan

$$Z = (S+A+B) \cdot (S+A+B') \cdot (S'+A+B) \cdot (S'+A'+B)$$

combining

$$= (S+A) \cdot (S'+B) = S' \cdot A + S \cdot B + A \cdot B$$

consensus

$$= S' \cdot A + S \cdot B$$

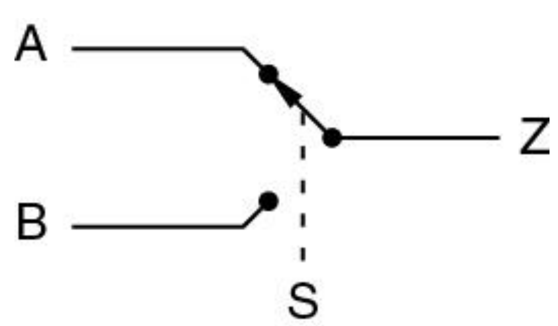
	S	A	B	Z	Z′	rows
$S' \cdot A' \cdot B'$	0	0	0	0	**1**	0
$S' \cdot A' \cdot B$	0	0	1	0	**1**	1
	0	1	0	1	**0**	2
	0	1	1	1	**0**	3
$S \cdot A' \cdot B'$	1	0	0	0	**1**	4
	1	0	1	1	**0**	5
$S \cdot A \cdot B'$	1	1	0	0	**1**	6
	1	1	1	1	**0**	7

maxterm POS

$$Z = \Pi_{S,A,B}(0,1,4,6)$$

minterm SOP

$$Z = \Sigma_{S,A,B}(2,3,5,7)$$

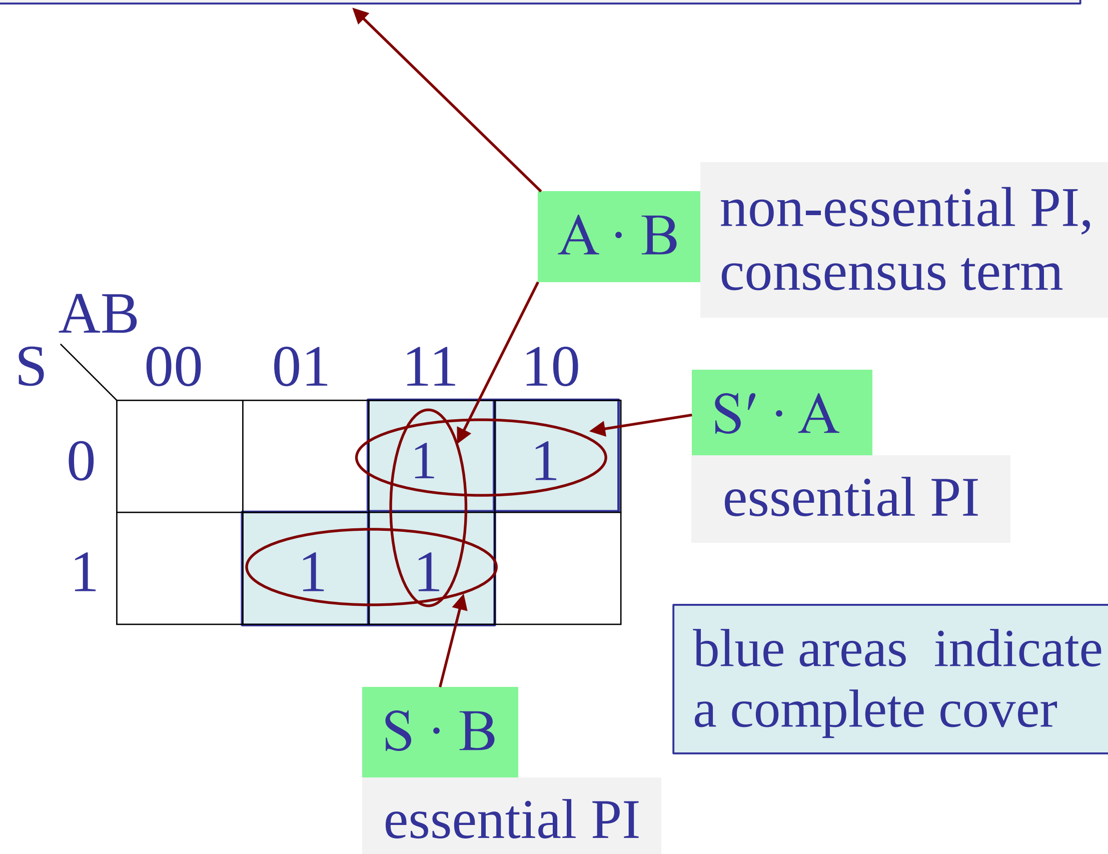


Example 2: Truth table of multiplexer function from Unit-1

Karnaugh map (K-map) simplification (to be discussed in detail later on)

$$Z = S' \cdot A \cdot B' + S' \cdot A \cdot B + S \cdot A' \cdot B + S \cdot A \cdot B$$
$$= S' \cdot A + S \cdot B = S' \cdot A + S \cdot B + A \cdot B$$

	S	A	B	Z
	0	0	0	0
	0	0	1	0
$S' \cdot A \cdot B'$	0	1	0	1
$S' \cdot A \cdot B$	0	1	1	1
	1	0	0	0
$S \cdot A' \cdot B$	1	0	1	1
	1	1	0	0
$S \cdot A \cdot B$	1	1	1	1



Example 3: For the truth table given in Table 3-5 of the Wakerly text, demonstrate the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(1) $F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$

(2) $F = Y'Z' + YZ + XYZ'$

(3) $F = Y'Z' + YZ + XZ'$

(4) $F = Y'Z' + YZ + XY$

(5) $F = Y'Z' + YZ + XZ' + XY$

(6) $F = (Y + Z')(X + Y' + Z)$

(7) $F = (X + Y + Z')(X + Y' + Z)(X' + Y + Z')$

minterm SOP (sum-of-products)

literals, product terms

maxterm POS (product-of-sums)

In particular, given (1) use the theorems to demonstrate the equivalence of (2)-(7) to (1).

$(1) \longrightarrow (2)$

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$$

$$= (X' + X)Y'Z' + (X' + X)YZ + XYZ' = Y'Z' + YZ + XYZ'$$

$(1) \longrightarrow (4)$

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$$

duplicated

$$= X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ + XYZ'$$

$$= (X' + X)Y'Z' + (X' + X)YZ + XY(Z + Z')$$

$$= Y'Z' + YZ + XY$$

$(1) \longrightarrow (2)$

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$$
$$= (X' + X)Y'Z' + (X' + X)YZ + XYZ' = Y'Z' + YZ + XYZ'$$

$(2) \longrightarrow (3)$

distributive

$$\boxed{X + A \cdot B = (X+A) \cdot (X+B)}$$

$$F = Y'Z' + YZ + XYZ'$$
$$= (Y' + XY)Z' + YZ = (Y' + X)\,(Y' + Y)\,Z' + YZ$$
$$= (Y' + X)\,Z' + YZ = Y'Z' + YZ + XZ'$$

$(2) \longrightarrow (4)$

distributive

$$F = Y'Z' + YZ + XYZ'$$
$$= Y'Z' + Y(Z + XZ') = Y'Z' + Y(Z + X)\,(Z + Z')$$
$$= Y'Z' + Y(Z + X) = Y'Z' + YZ + XY$$

$(3) \longrightarrow (5)$

consensus

$$F = Y'Z' + (YZ + XZ') = Y'Z' + (YZ + XZ' + XY)$$

truth table

X	Y	Z	F	F′
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

justification of Eq.(7) from the truth table

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$$

minterm SOP for F (sum-of-products)

minterm SOP for F′ (sum-of-products)

maxterm POS for F (product-of-sums)

$$F' = X'Y'Z + X'YZ' + XY'Z$$

De Morgan

$$F = (X + Y + Z')\,(X + Y' + Z)\,(X' + Y + Z') \qquad \text{Eq.(7)}$$

Karnaugh map examples – 1

(to be fully explained later on)

Previously (in Example 3) we considered the simplification of the truth table function given in Table 3-5 of the Wakerly text, and demonstrated the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

(1) $F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$

(2) $F = Y'Z' + YZ + XZ'$ → p.64, gate-level realizations

(3) $F = Y'Z' + YZ + XY$

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Z \ XY	00	01	11	10
0	1		1	1
1		1	1	

blue areas indicate a complete cover

Y′Z′ essential PI

XZ′

essential PI YZ

XY

essential PIs, must include either for a complete cover one is essential, the other not

12. Algebraic simplification of combinational logic expressions

Example 4: For the truth table given below [ref. A. F. Kana, on Canvas], show the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

truth table

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(1) $F = X'YZ + XY'Z + XYZ' + XYZ$

(2) $F = X'YZ + XY'Z + XY$

(3) $F = X'YZ + XZ + XYZ'$

(4) $F = YZ + XY'Z + XYZ'$

(5) $F = XY + YZ + XZ$

In particular, given Eq.(1) use the theorems to demonstrate the equivalence of Eqs.(2)-(5) to Eq.(1).

$(1) \longrightarrow (2)$ $\quad F = X'YZ + XY'Z + XYZ' + XYZ$

$$= X'YZ + XY'Z + XY(Z' + Z) = X'YZ + XY'Z + XY$$

$(1) \longrightarrow (3)$ $\quad F = X'YZ + XY'Z + XYZ' + XYZ$

$$= X'YZ + X(Y' + Y)Z + XYZ' = X'YZ + XZ + XYZ'$$

$(1) \longrightarrow (4)$ $\quad F = X'YZ + XY'Z + XYZ' + XYZ$

$$= (X' + X)YZ + XY'Z + XYZ' = YZ + XY'Z + XYZ'$$

$(2) \longrightarrow (5)$ $\quad F = X'YZ + XY'Z + XY = X'YZ + X(Y + Y'Z)$

$$= X'YZ + X(Y + Y')(Y + Z) = X'YZ + X(Y + Z)$$

$$= X'YZ + XY + XZ = Y(X + X'Z) + XZ$$

$$= Y(X + X')(X + Z) + XZ = Y(X + Z) + XZ$$

$$= XY + YZ + XZ$$

(1) $\longrightarrow$ (5) , alternative method,

duplicate XYZ and use the property $A + A + A = A$

$$F = X'YZ + XY'Z + XYZ' + XYZ$$

$$= X'YZ + XYZ + XY'Z + XYZ + XYZ' + XYZ$$

$$= (X' + X)YZ + X(Y' + Y)Z + XY(Z' + Z)$$

$$= YZ + XZ + XY$$

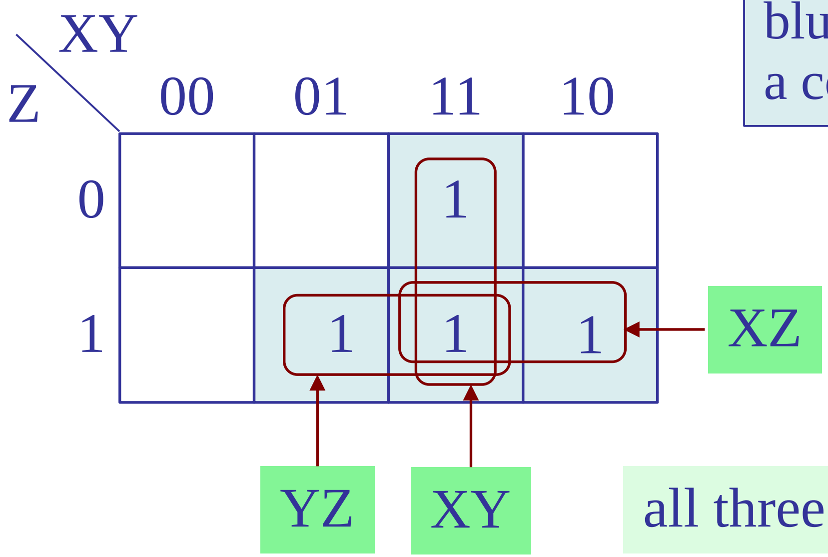
Example 4 – K-map method (to be fully explained later on): For the truth table given below [ref. A. F. Kana], show the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

truth table

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(1) $F = X'YZ + XY'Z + XYZ' + XYZ$

(5) $F = XY + YZ + XZ$



blue areas indicate a complete cover

all three are essential PIs

12. Algebraic simplification of combinational logic expressions

Example 5: Provide two proofs of the equivalence of the following two expressions:

$$F = X'Y + XY' + XY$$
$$F = X + Y$$

Method 1: $F = X'Y + XY' + XY = X'Y + X(Y' + Y) =$

$= X'Y + X = X + Y$ (from Example 1)

Method 2: $F = X'Y + XY' + XY$

$= X'Y + XY' + XY + XY =$

$= (X' + X)Y + X(Y' + Y)$

$= X + Y$

replicate XY using the property $A + A = A$

Example 6: Show the equivalence of the following two 4-variable expressions [ref. A. F. Kana],

$$F = A'BC'D + A'BCD + ABC'D' + ABC'D + ABCD + ABCD' + \ldots + AB'CD + AB'CD'$$

$$F = BD + AB + AC$$

Proof:

$$\begin{aligned}
F &= A'BC'D + A'BCD + ABC'D' + ABC'D + ABCD + ABCD' + \ldots + AB'CD + AB'CD' \\
&= A'B(C' + C)D + ABC'D' + AB(C' + C)D + ABCD' + \ldots + AB'CD + AB'CD' \\
&= A'BD + ABC'D' + ABD + ABCD' + AB'CD + AB'CD' \\
&= (A' + A)BD + AB(C' + C)D' + AB'C(D + D') \\
&= BD + ABD' + AB'C = B(D + AD') + AB'C \\
&= B(D + A)(D + D') + AB'C = B(D+A) + AB'C = BD + AB + AB'C \\
&= BD + A(B + B'C) = BD + A(B + B')(B + C) = BD + A(B + C) \\
&= BD + AB + AC
\end{aligned}$$

see next page for simpler K-map method

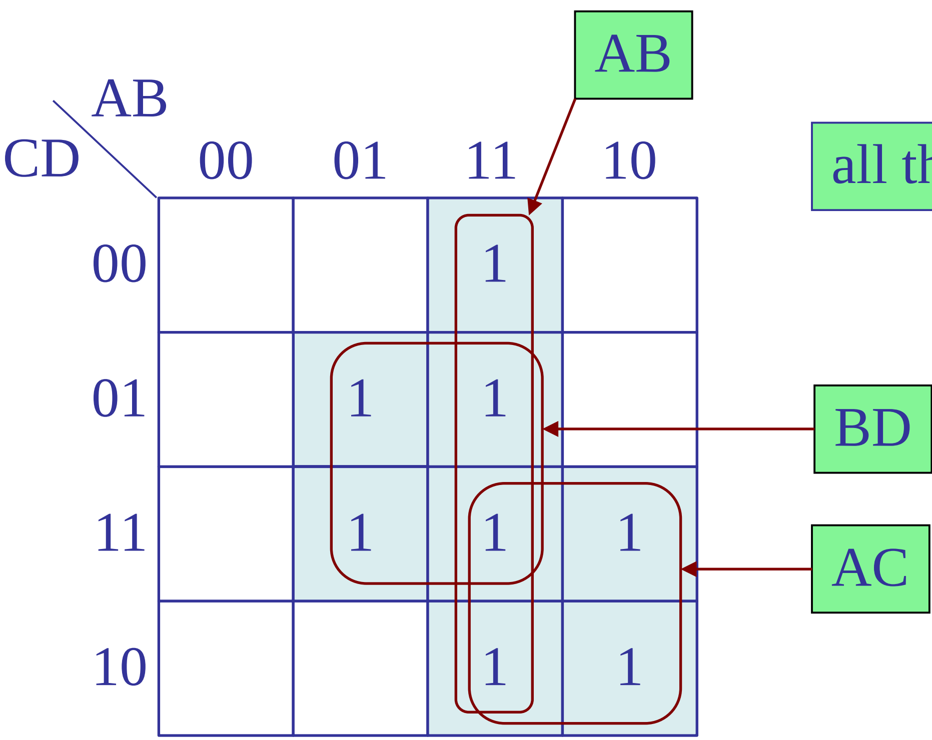
Karnaugh map examples – 5

(to be fully explained later on)

Previously (in Example 6), we showed the equivalence of the following two 4-variable expressions [ref. A. F. Kana],

$$F = A'BC'D + A'BCD + ABC'D' + ABC'D + ABCD + ABCD' + AB'CD + AB'CD'$$

$$F = BD + AB + AC$$



all three are essential PIs

blue areas indicate a complete cover

implicant example

if $AB = 1$, then, $A=1$, $B=1$, $A'=0$, $B'=0$

$F = C'D' + C'D + CD + CD'$

$= (C+C')(D+D') = 1 \cdot 1 = 1$

13. Combinational circuit truth-table synthesis example

Example 3, continued: Previously (p.52), we demonstrated the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$$

$$F = Y'Z' + YZ + XZ'$$

As a first synthesis example, we wish to realize the second relationship by means of logic gates, and on the Emona board, and simulate it in MATLAB and Simulink, generate the given truth table and a timing diagram, and observe the input and output signals on a scope, and plot them in MATLAB.

The following files are on Canvas Resources, and may be used as templates for future examples:

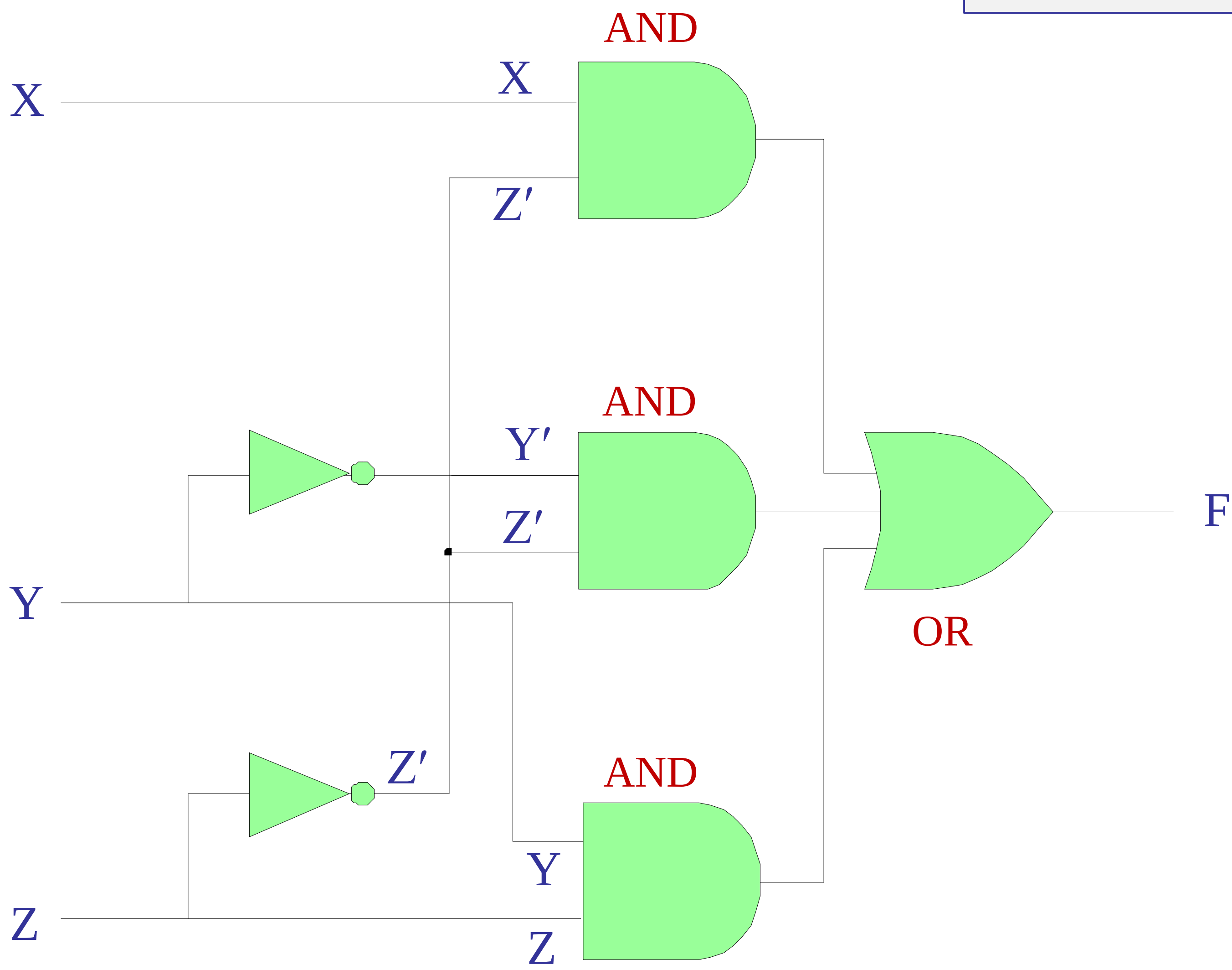
truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

`table35m.m`, MATLAB m-file for the truth table and plotting
`table35s.slx`, Simulink file,
`table35v.v`, Verilog code generated by Simulink

$$F = Y'Z' + YZ + XZ'$$

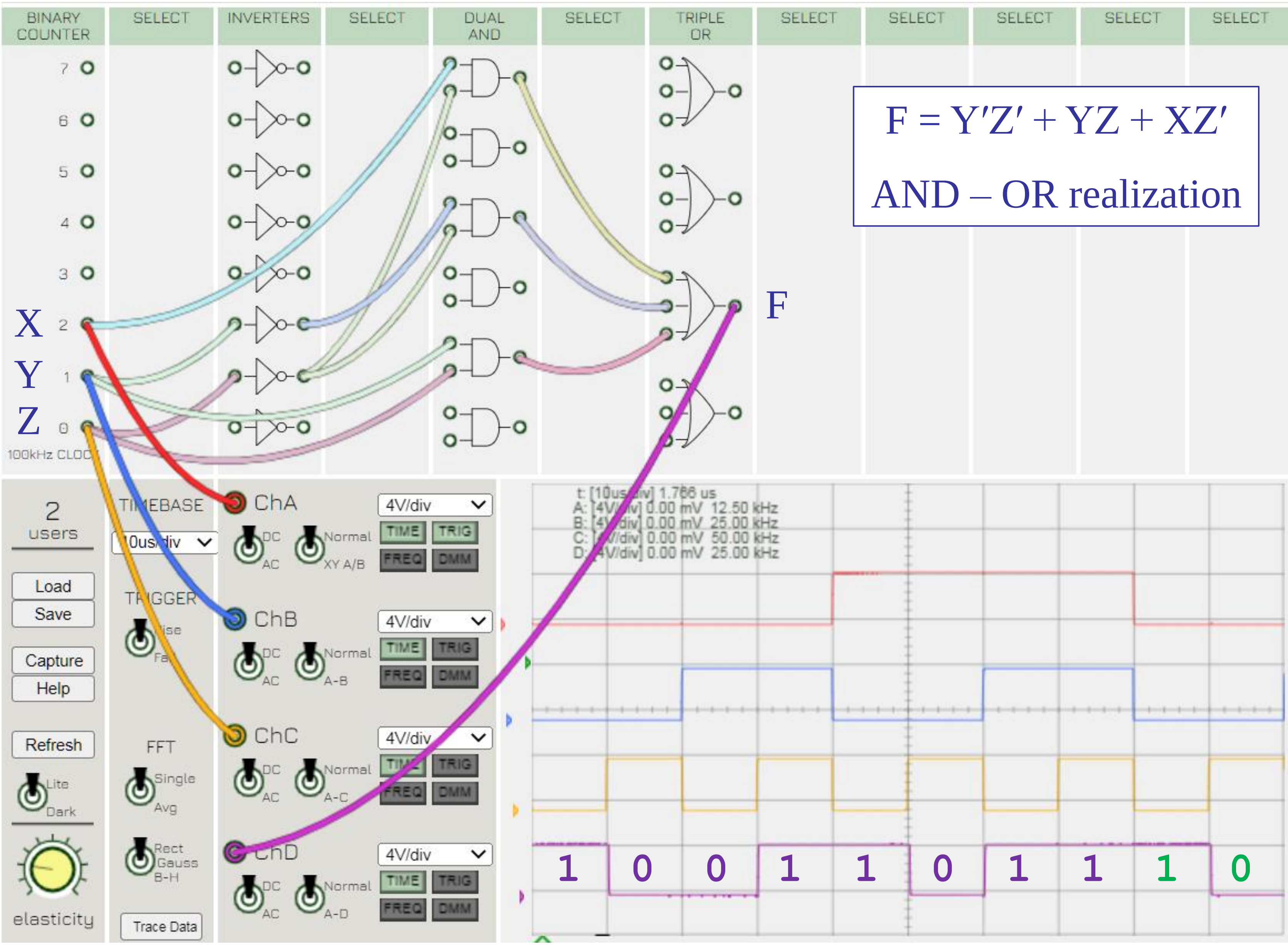
AND – OR realization



Emona board realization

$F = Y'Z' + YZ + XZ'$

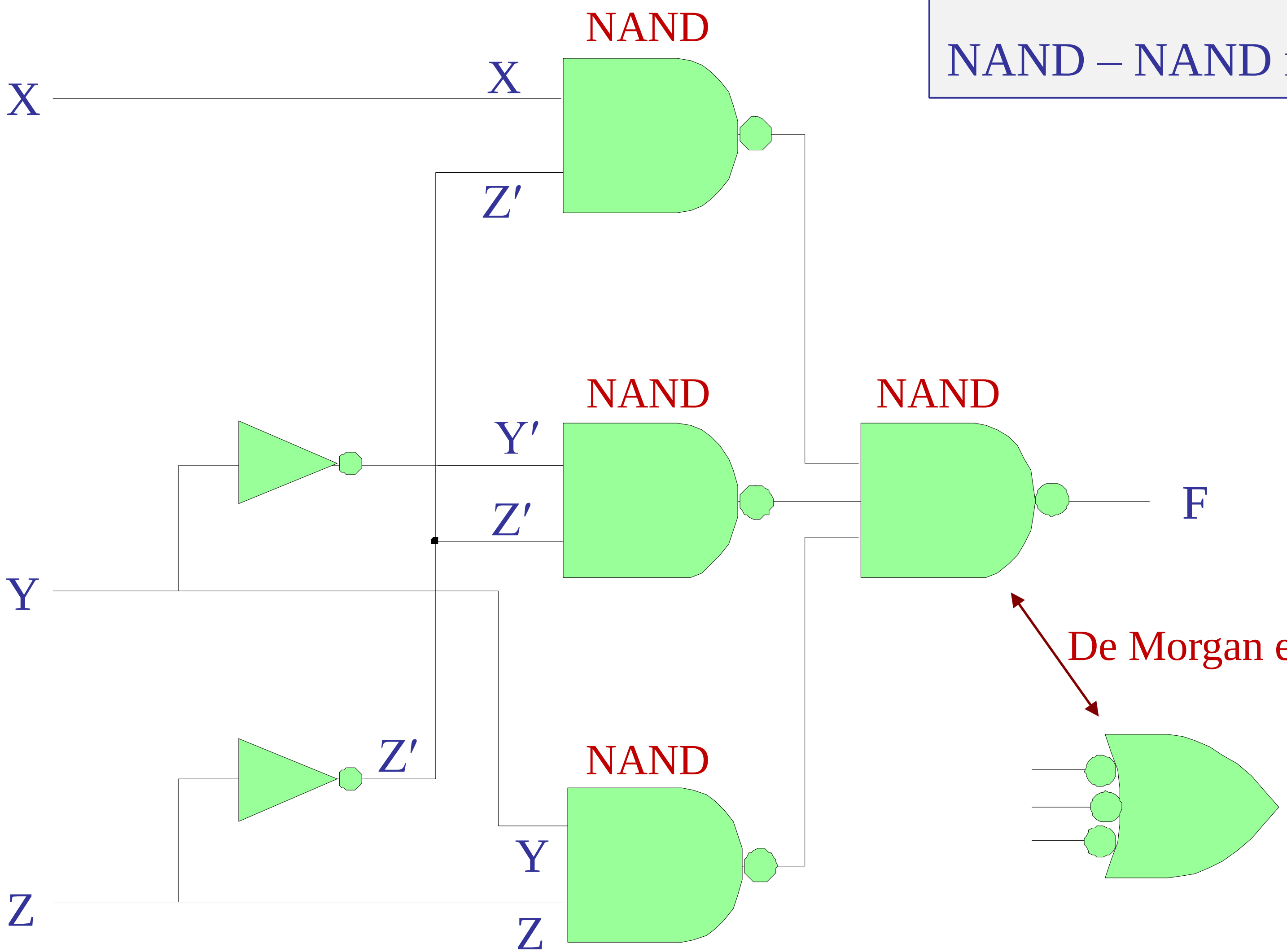
AND – OR realization



$F = Y'Z' + YZ + XZ'$

$F = (\ (Y'Z')'\,(YZ)'\,(XZ')'\ )'$

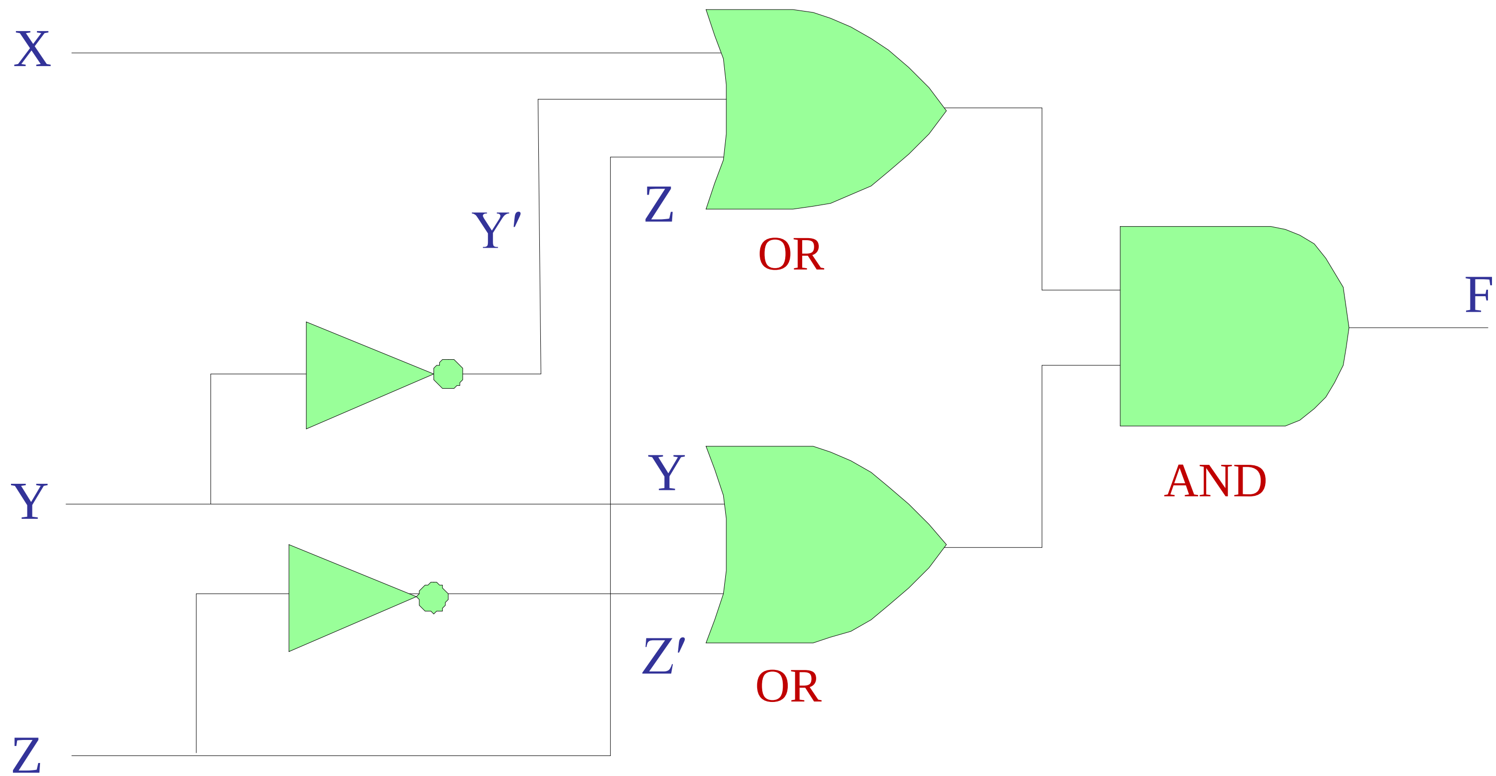
NAND – NAND realization



$F = (Y + Z')(X + Y' + Z)$

simplified POS realization
OR – AND realization

see next page for explanation



truth table

X	Y	Z	F	F′
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

minterm SOP form for F, from the truth table

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$$

minterm SOP form for F′, from the truth table

$$F' = X'Y'Z + X'YZ' + XY'Z = (X' + X)Y'Z + X'YZ' = Y'Z + X'YZ'$$

De Morgan

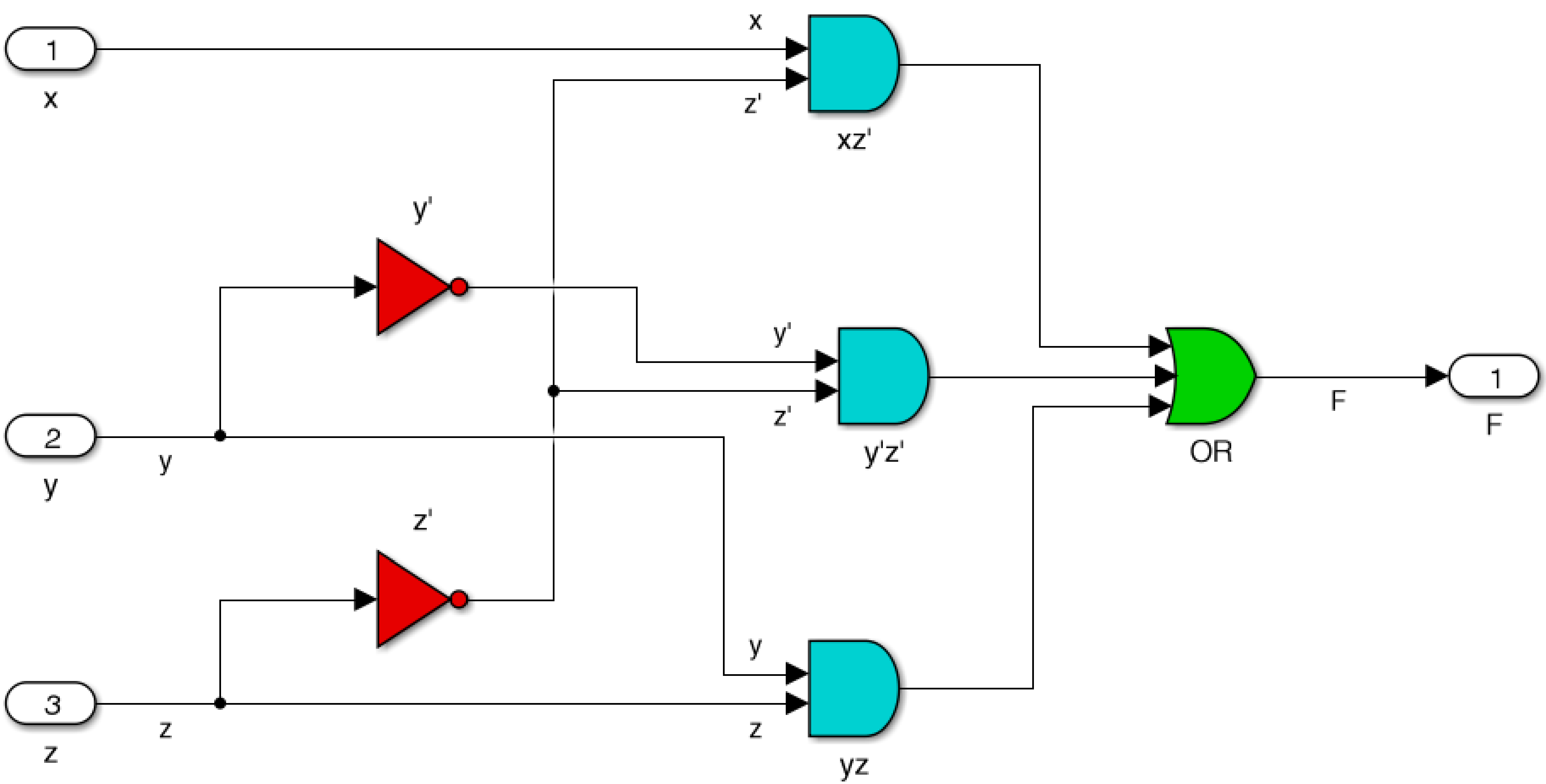
$$F = (Y + Z')\,(X + Y' + Z)$$

simplified POS realization

14. MATLAB/Simulink implementations, and exporting to Verilog

$$F = Y'Z' + YZ + XZ'$$

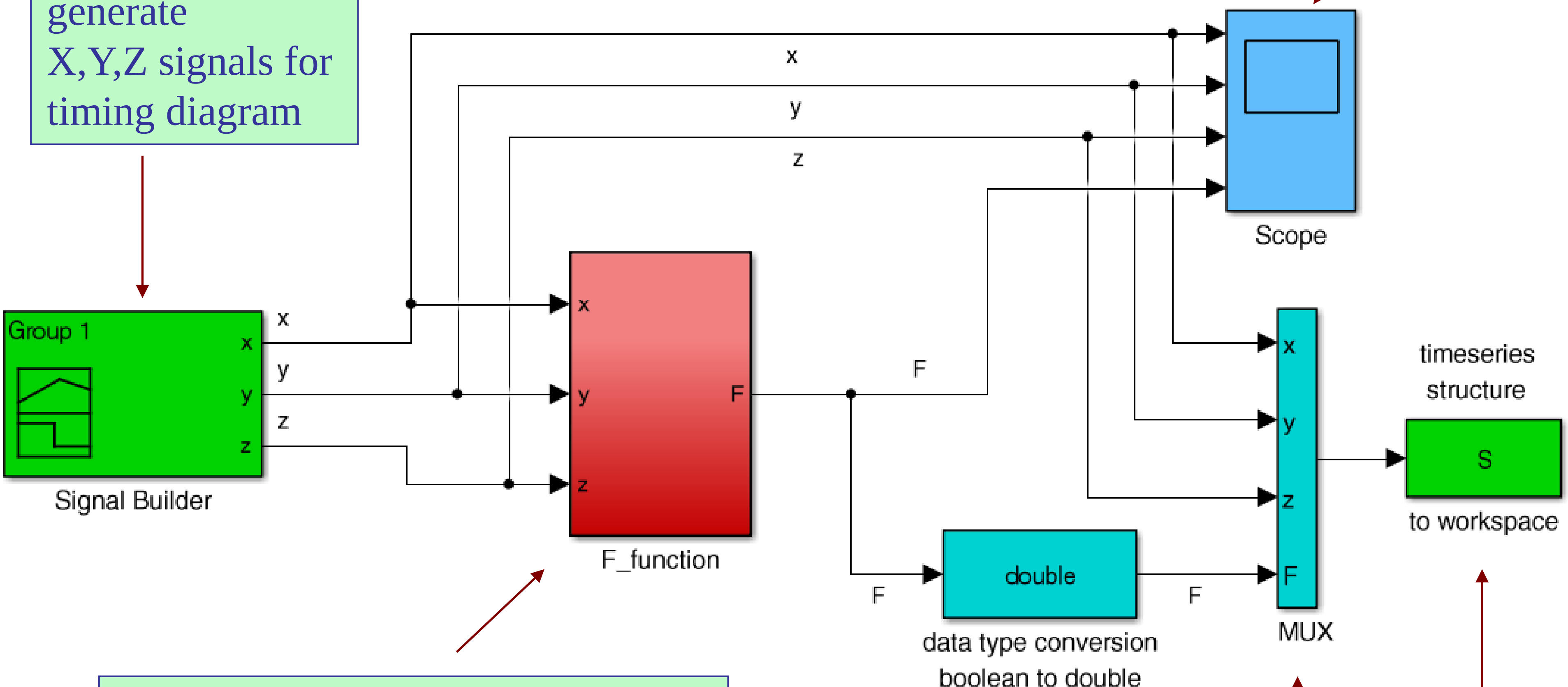
AND–OR realization, F-function re-drawn with Simulink



$F = Y'Z' + YZ + XZ'$
F-function embedded in executable Simulink realization

generate X,Y,Z signals for timing diagram

plot X,Y,Z,F signals

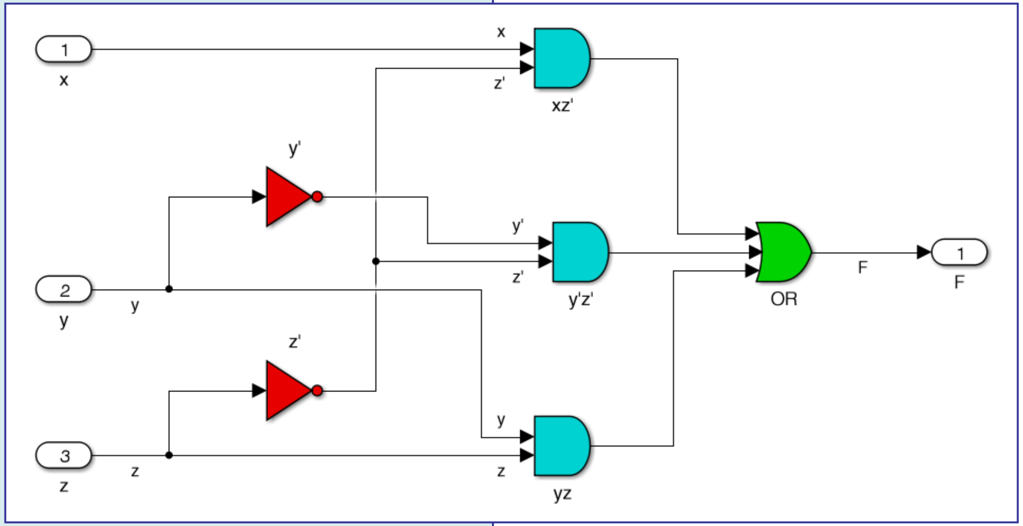


F-function resides within this block and can be exported to Verilog

needed for exporting X,Y,Z,F data to MATLAB’s workspace

Verilog code generated by Simulink – file table35v.v

```
module table35v (x, y, z, F);

  input   x, y, z, F;

  wire z_2; wire z_3; wire x_1;
  wire y_2; wire y_3; wire y_4; wire y_5;
  wire xz_out1;

  assign z_2 =  ~ z;
  assign z_3 = z_2;
  assign x_1 = x & z_3;
  assign y_2 =  ~ y;
  assign y_3 = y_2;
  assign y_4 = y_3 & z_3;
  assign y_5 = y & z;
  assign xz_out1 = y_5 | (x_1 | y_4);
  assign F = xz_out1;

endmodule
```

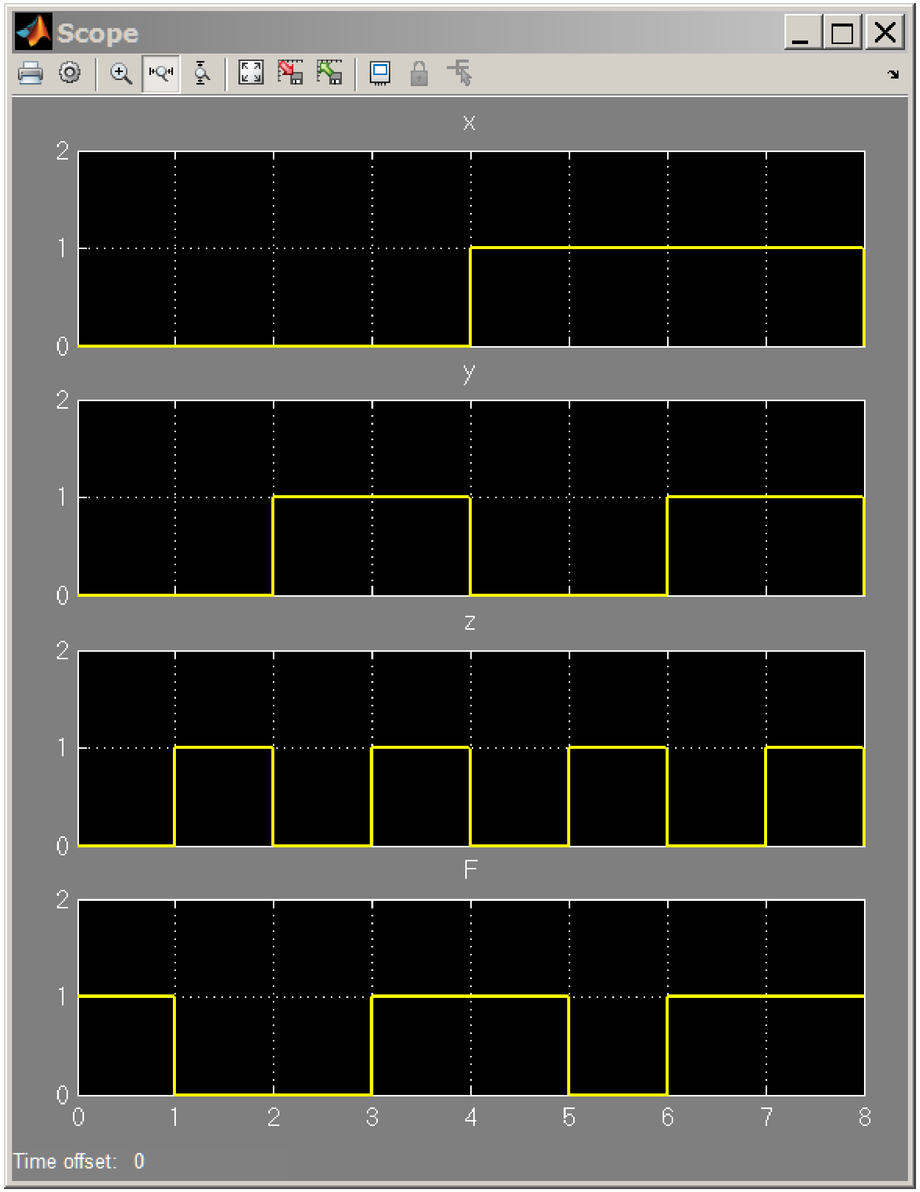


F = Y′Z′ + YZ + XZ′
verify truth table

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

viewing timing diagram on scope



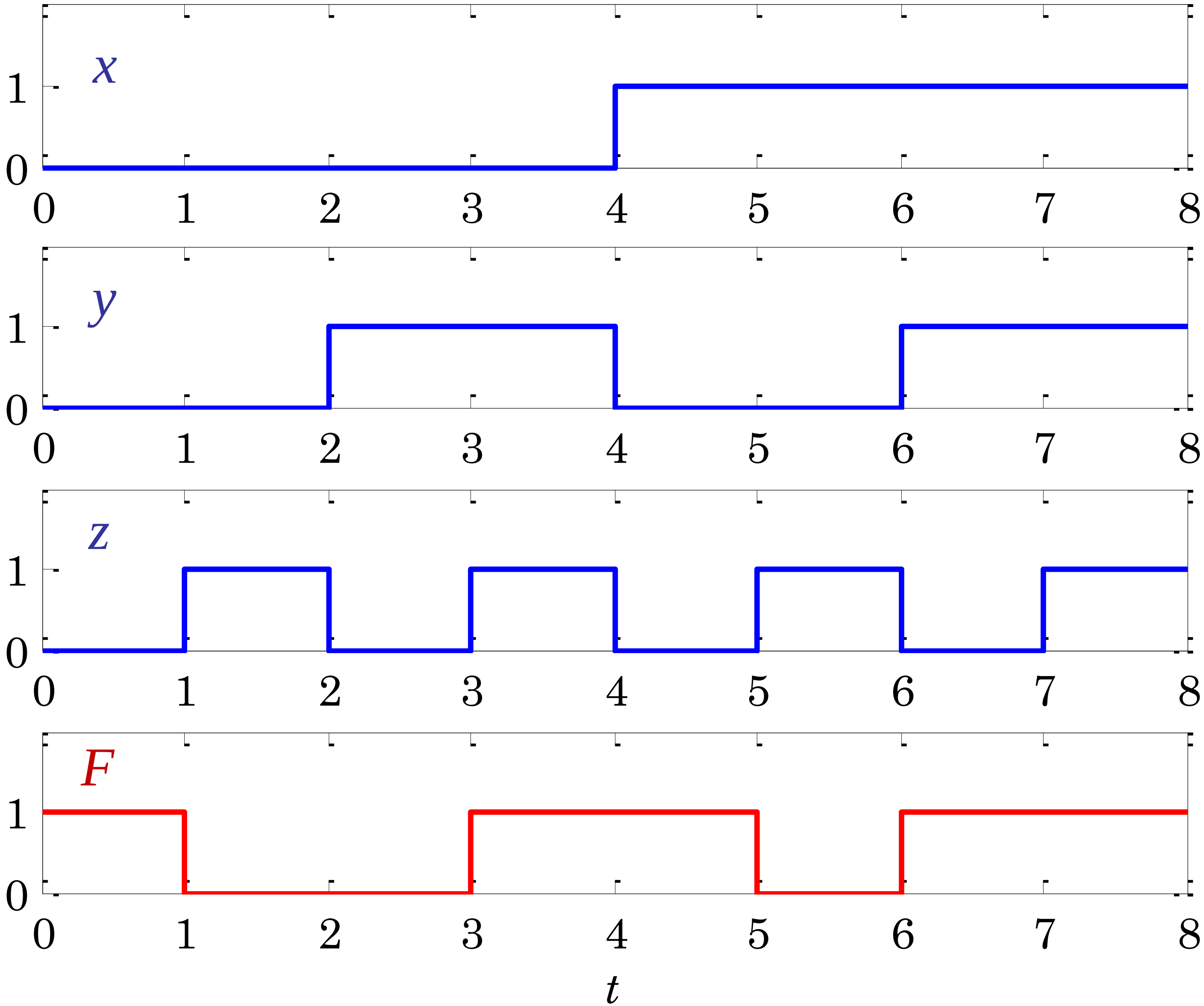
```
%% table35m.m - generating the truth table

[X,Y,Z] = a2d(0:7,3);    % 3-bit binary pattern

F = (~Y & ~Z)|(Y & Z)|(X & ~Z);    % construct output F

[X,Y,Z,F]  % print truth table

%  X  Y  Z  F
%  ----------
%  0  0  0  1
%  0  0  1  0
%  0  1  0  0
%  0  1  1  1
%  1  0  0  1
%  1  0  1  0
%  1  1  0  1
%  1  1  1  1
```

MATLAB version of
$F = Y'Z' + YZ + XZ'$

the operations **&** and **|** are vectorized

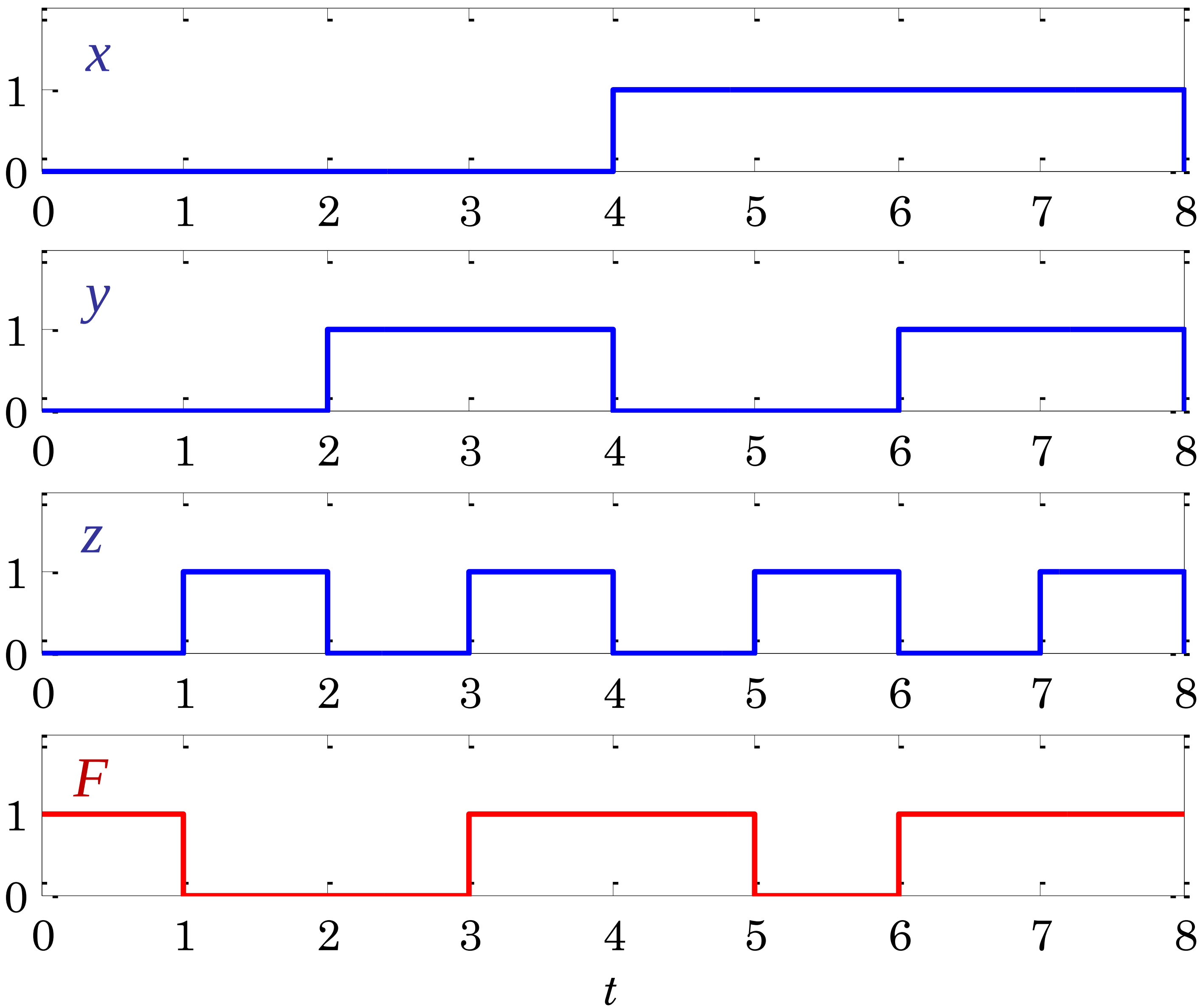
```
% plot timing diagram

t = (0:8);      % last bit has duration from t=7 to t=8

x = [X; X(end)];            % extend duration of last bit
y = [Y; Y(end)];
z = [Z; Z(end)];
f = [F; F(end)];

set(0,'DefaultAxesFontSize',10);

figure;              % xaxis, yaxis are on Canvas M-files
subplot(4,1,1);
    stairs(t,x,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,2);
    stairs(t,y,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,3);
    stairs(t,z,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,4);
    stairs(t,f,'r-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
xlabel('\itt');
```

timing diagram



```
% plot timing diagram from saved simulation data
% extracted from the timeseries structure S

t = S.time;
x = S.data(:,1);
y = S.data(:,2);
z = S.data(:,3);
F = S.data(:,4);

figure;               % xaxis,yaxis are on Canvas M-files
subplot(4,1,1);
   stairs(t,x,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,2);
   stairs(t,y,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,3);
   stairs(t,z,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,4);
   stairs(t,F,'r-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
xlabel('\itt')
```

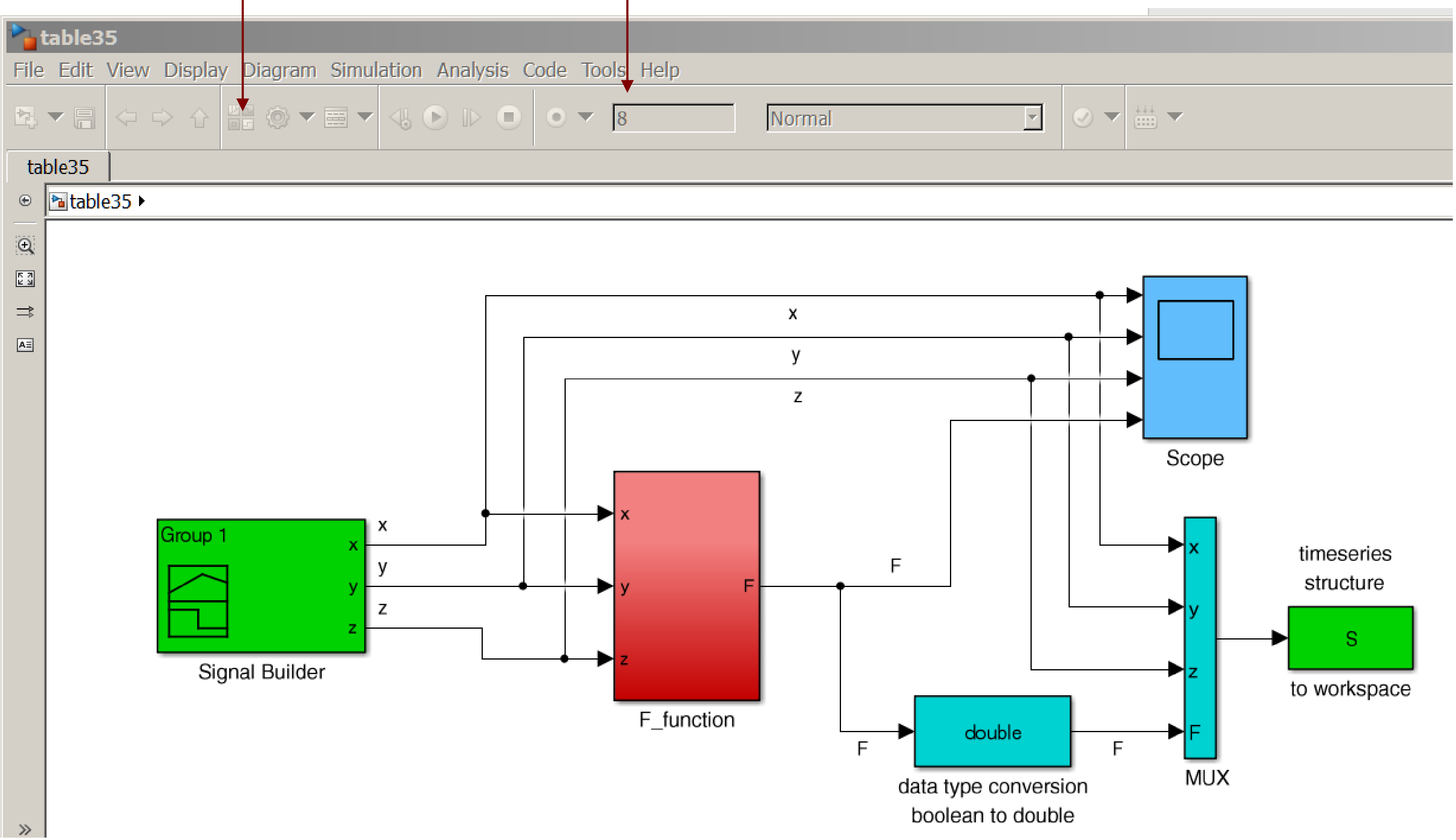
generating a better plot of the timing diagram, than from the scope plot timing diagram from Simulink data exported to workspace



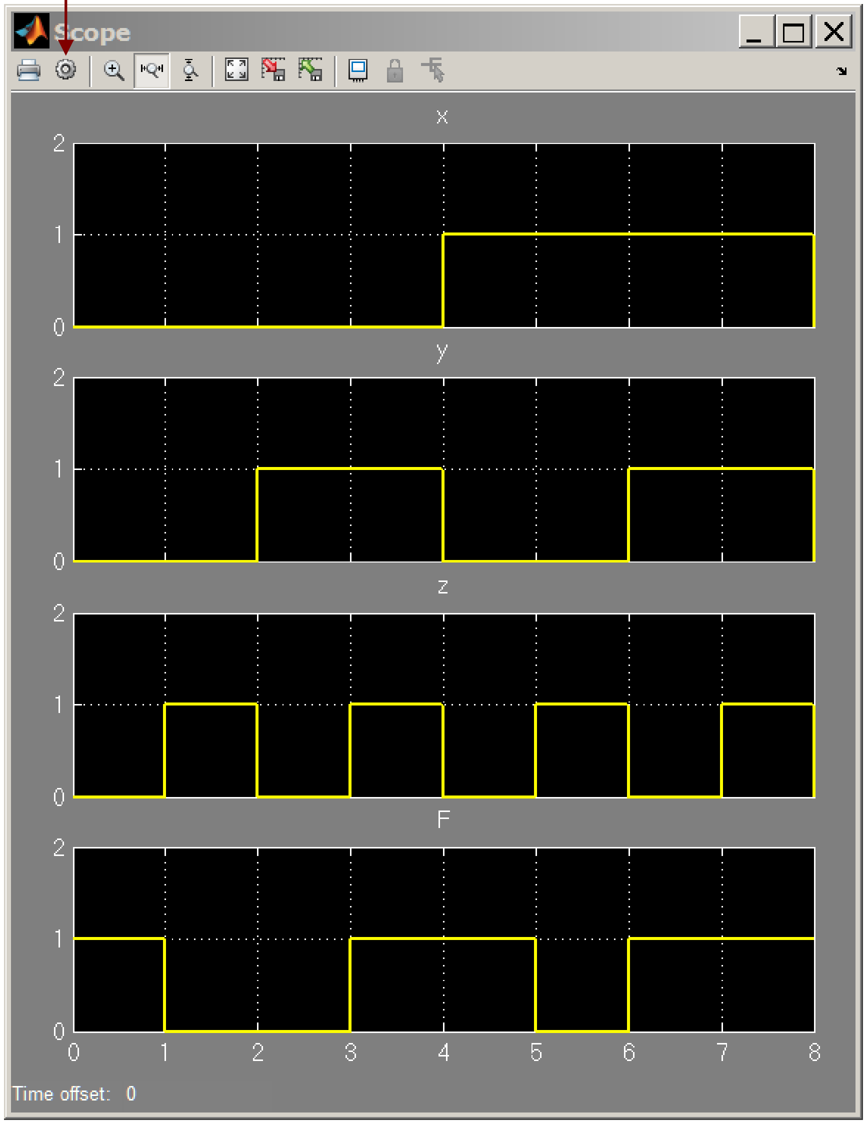
Additional notes on Simulink

click here to open Simulink Library

set simulation duration to 8 time units

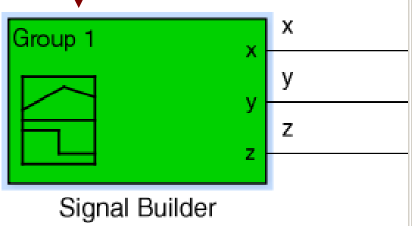


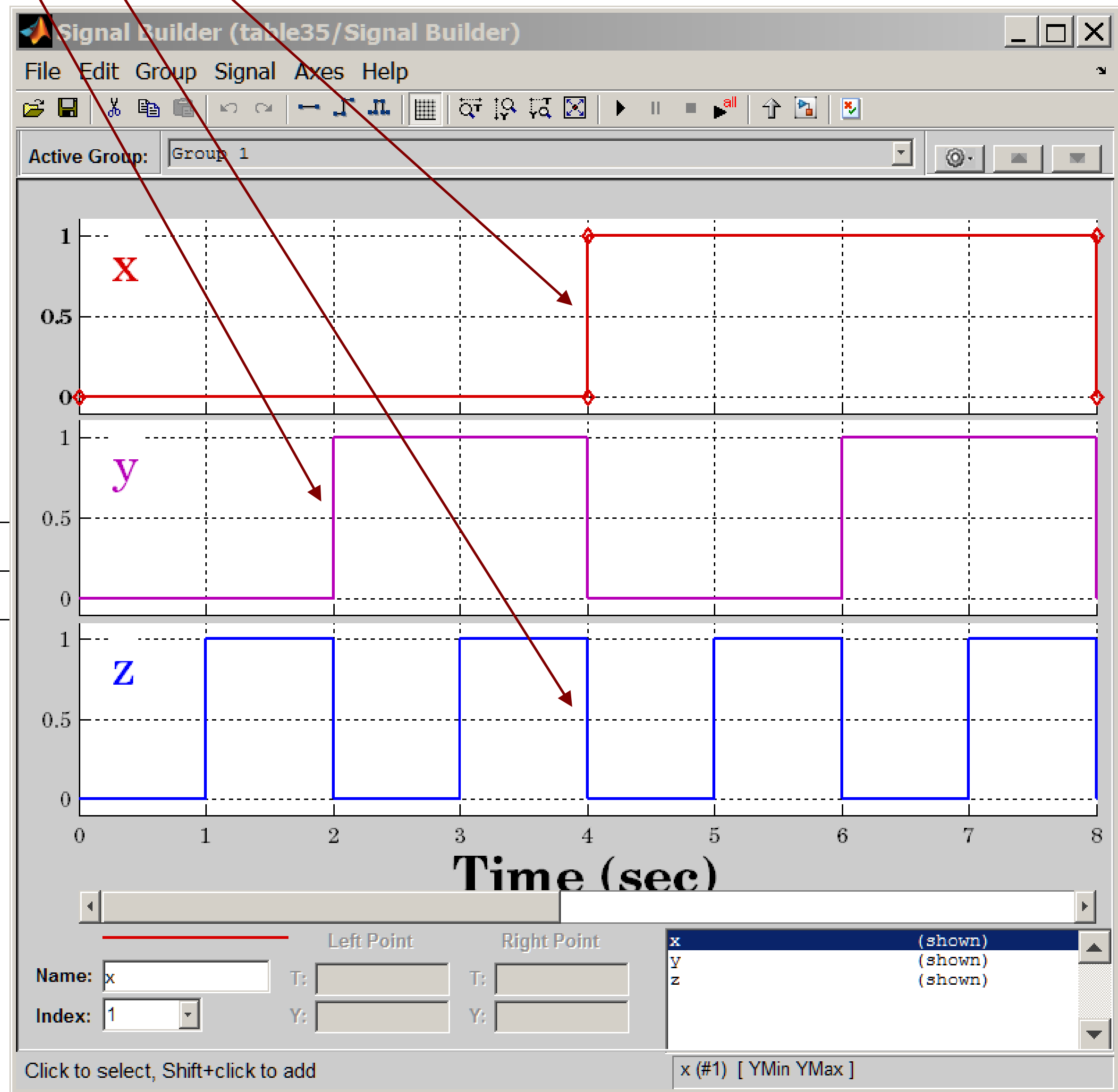
click here to open scope parameters and select 4 axes (for X,Y,X,F) and time range of 8 units



drag the signal edges to change the shape of the signals

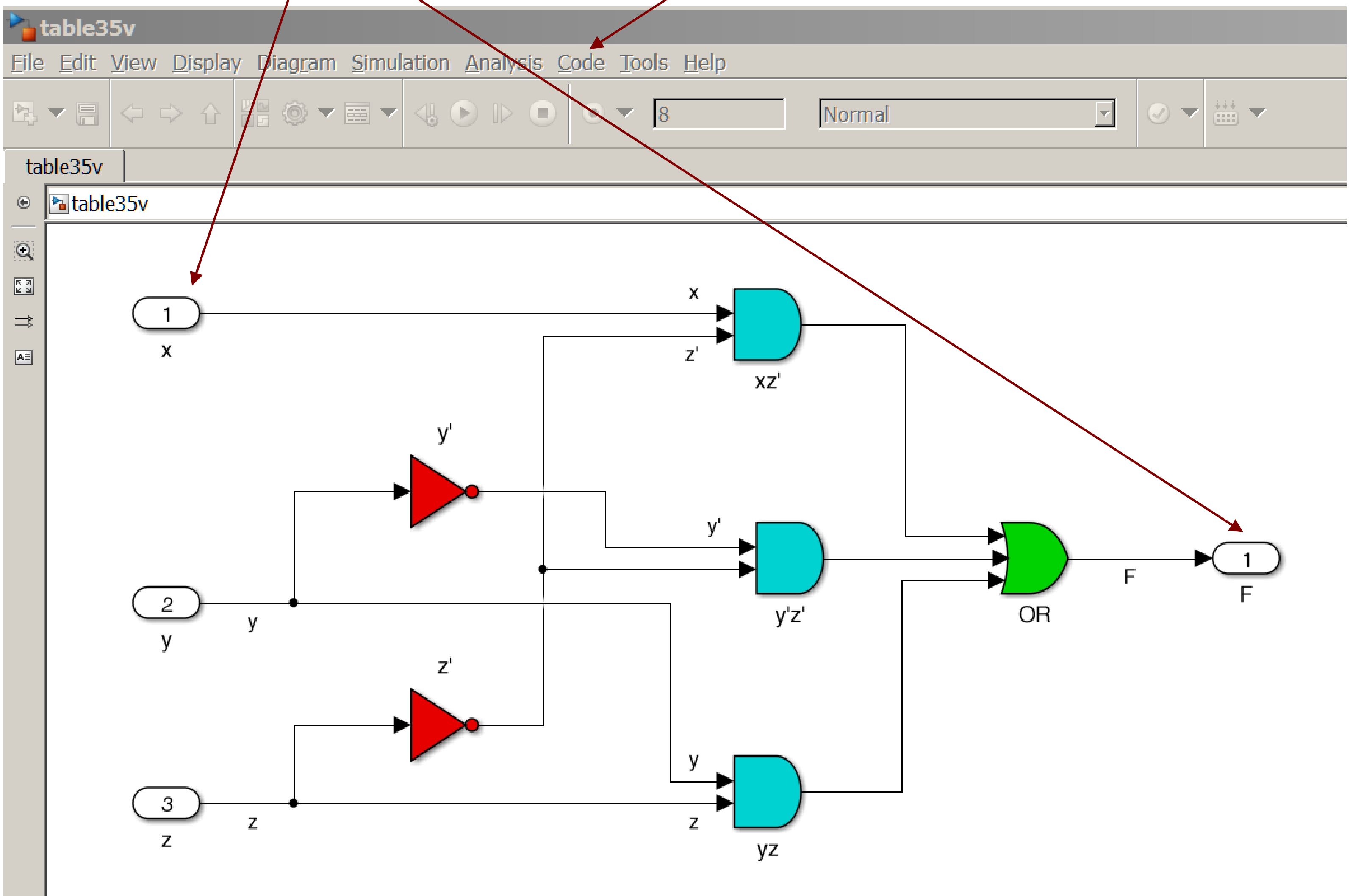
double-click on the Signal Builder block to view the signals X,Y,Z over the time range of 8 units





can also be preloaded by copy/pasting from the file signals234.slx on Canvas before exporting into Verilog, save the subfunction into a separate SLX Simulink file, then set the data types of the input/output ports to Boolean

select Code, then HDL code, Options, and Verilog



15. Standard representations of combinational circuits

truth table representations

2-variable F(X,Y)

row	X	Y	F
0	0	0	F(0,0)
1	0	1	F(0,1)
2	1	0	F(1,0)
3	1	1	F(1,1)

3-variable F(X,Y,Z)

row	X	Y	Z	F
0	0	0	0	F(0,0,0)
1	0	0	1	F(0,0,1)
2	0	1	0	F(0,1,0)
3	0	1	1	F(0,1,1)
4	1	0	0	F(1,0,0)
5	1	0	1	F(1,0,1)
6	1	1	0	F(1,1,0)
7	1	1	1	F(1,1,1)

$\downarrow$ binary order

row	A	B	C	D	F
0	0	0	0	0	F(0,0,0,0)
1	0	0	0	1	F(0,0,0,1)
2	0	0	1	0	F(0,0,1,0)
3	0	0	1	1	F(0,0,1,1)
4	0	1	0	0	F(0,1,0,0)
5	0	1	0	1	F(0,1,0,1)
6	0	1	1	0	F(0,1,1,0)
7	0	1	1	1	F(0,1,1,1)
8	1	0	0	0	F(1,0,0,0)
9	1	0	0	1	F(1,0,0,1)
10	1	0	1	0	F(1,0,1,0)
11	1	0	1	1	F(1,0,1,1)
12	1	1	0	0	F(1,1,0,0)
13	1	1	0	1	F(1,1,0,1)
14	1	1	1	0	F(1,1,1,0)
15	1	1	1	1	F(1,1,1,1)

4-variable function F(A,B,C,D)

Truth-table representations with minterms or maxterms

3-variable F(X,Y,Z)						
row	X	Y	Z	F	minterms	maxterms
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

complements of each other
by De Morgan

Truth-table representations with minterms or maxterms

2-variable F(X,Y)					
row	X	Y	F	minterms	maxterms
0	0	0	F(0,0)	$X' \cdot Y'$	$X + Y$
1	0	1	F(0,1)	$X' \cdot Y$	$X + Y'$
2	1	0	F(1,0)	$X \cdot Y'$	$X' + Y$
3	1	1	F(1,1)	$X \cdot Y$	$X' + Y'$

complements of each other
by De Morgan

Example 3, continued: Previously, we demonstrated the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

$$F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ' \quad = \text{(sum of minterms, SOP)}$$

$$F = (X + Y + Z')(X + Y' + Z)(X' + Y + Z') \quad = \text{(product of maxterms, POS)}$$

They were based on the truth table shown on the right (Table 3-5 of the Wakerly text)

To understand these expressions, we expand the truth table to also include the complement of F, that is, F′, obtained by interchanging 0s and 1s in the F column.

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

$F = X'Y'Z' + X'YZ + XY'Z' + XYZ' + XYZ$ (minterm SOP, sum-of-products)

$F' = X'Y'Z + X'YZ' + XY'Z$ (minterms of F′ from truth table)

$F = F'' = (X'Y'Z + X'YZ' + XY'Z)' = (X'Y'Z)'\,(X'YZ')'\,(XY'Z)'$ ← De Morgan

$F = (X + Y + Z')\,(X + Y' + Z)\,(X' + Y + Z')$ (maxterm POS, product-of-sums)

truth table

X Y Z	F	F′	F minterms	F′ minterms	F maxterms
0 0 0	1	0	X′Y′Z′		
0 0 1	0	1		X′Y′Z	X+Y+Z′
0 1 0	0	1		X′Y Z′	X+Y′+Z
0 1 1	1	0	X′Y Z		
1 0 0	1	0	X Y′Z′		
1 0 1	0	1		X Y′Z	X′+Y+Z′
1 1 0	1	0	X Y Z′		
1 1 1	1	0	X Y Z		

De Morgan →

(maxterms of F = (minterms of F′) ′

16. Canonical minterm/SOP and maxterm/POS representations

Sum-of-Products (SOP) rule:

F is the sum of those minterms that correspond
to the values F=1 in the truth table

Notation: $F = \Sigma$(of the F=1 minterms) = canonical sum-of-products

as indicated by the row numbers in the truth table

Product-of-Sums (POS) rule:

F is the product of those maxterms that correspond
to the values F=0 (or, $F' = 1$) in the truth table

Notation: $F = \Pi$(of the F=0 maxterms) = canonical product-of-sums

as indicated by the row numbers in the truth table

Example:

$$F = X'Y'Z' + X'YZ + XY'Z' + XYZ' + XYZ = \Sigma_{X,Y,Z}(0,3,4,6,7)$$

equivalent

$$F = (X + Y + Z')\,(X + Y' + Z)\,(X' + Y + Z') = \Pi_{X,Y,Z}(1,2,5)$$

truth table

row	X Y Z	F	F′
0	0 0 0	1	0
1	0 0 1	0	1
2	0 1 0	0	1
3	0 1 1	1	0
4	1 0 0	1	0
5	1 0 1	0	1
6	1 1 0	1	0
7	1 1 1	1	0

how does it work? →

X Y Z	F	minterms	X′Y′Z′	X′Y Z	X Y′Z′	X Y Z′	X Y Z
0 0 0	1	X′Y′Z′	1	0	0	0	0
0 0 1	0		0	0	0	0	0
0 1 0	0		0	0	0	0	0
0 1 1	1	X′Y Z	0	1	0	0	0
1 0 0	1	X Y′Z′	0	0	1	0	0
1 0 1	0		0	0	0	0	0
1 1 0	1	X Y Z′	0	0	0	1	0
1 1 1	1	X Y Z	0	0	0	0	1

F = OR'ing these five columns together

$$F = X'Y'Z' + X'YZ + XY'Z' + XYZ' + XYZ$$

```
[x,y,z] = a2d(0:7,3);

[x, y, z, ~x&~y&~z, ~x&y&z, x&~y&~z, x&y&~z, x&y&z]
```

X Y Z	F	maxterms	X+Y+Z′	X+Y′+Z	X′+Y+Z′
0 0 0	1		1	1	1
0 0 1	0	X+Y+Z′	0	1	1
0 1 0	0	X+Y′+Z	1	0	1
0 1 1	1		1	1	1
1 0 0	1		1	1	1
1 0 1	0	X′+Y+Z′	1	1	0
1 1 0	1		1	1	1
1 1 1	1		1	1	1

F = AND’ing these three columns together

$F = (X + Y + Z')\,(X + Y' + Z)\,(X' + Y + Z')$

```
[x,y,z] = a2d(0:7,3);

[x, y, z, x|y|~z, x|~y|z, ~x|y|~z]
```

Nomenclature

literal:	a single Boolean variable, e.g., X
product term:	a product of variables, e.g., XZ
sum term:	a sum of variables, e.g., Y+Z
minterm:	a product of input variables corresponding to a row in truth table e.g., XY′Z, corresponding to row 101 = 5
canonical SOP:	canonical minterm sum-of-products, e.g., $\Sigma_{X,Y,Z}(0,3,4,6,7)$
minterm list :	list of row numbers that appear in a canonical SOP
maxterm:	a sum of input variables, e.g., X+Y′+Z, corresponding to the complement of a row in the truth table
canonical POS:	canonical maxterm product-of-sums, e.g., $\Pi_{X,Y,Z}(1,2,5)$
maxterm list :	list of row numbers that appear in a canonical POS

Nomenclature

implicant: a minterm or sum of minterms appearing in a function F, if an implicant evaluates to 1, then so does F as a whole, i.e., if, implicant=1, then it implies, F=1

prime implicant: a simplified implicant that cannot be combined into another implicant that has fewer number of literals.

covers: all implicants that account for all possible evaluations of the function into F=1 (i.e., all the 1's in a Karnaugh map).

essential prime implicant: a prime implicant that contains an F=1 minterm that is not included in any other prime implicant,

all essential prime implicants must be included in the cover of the function.

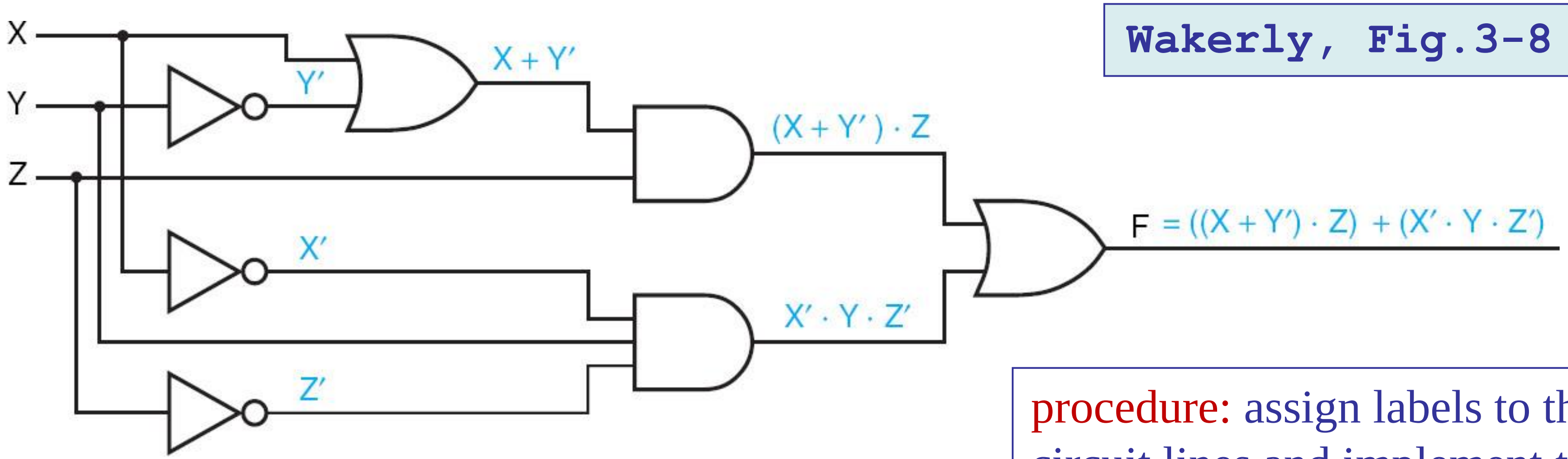
In addition to the essential PIs, it may be necessary to include possible non-essential PIs in order to achieve a complete cover, (if there are several such possibilities, one could choose the one that has the smallest number of literals.

17. Combinational circuit analysis – Example 1

Previously (in Example 3) we looked at a combinational circuit synthesis problem, in which the circuit was defined by a truth table, and we synthesized it in several equivalent ways using logic gates.

Here, we look at an analysis example of a circuit constructed in terms of logic gates, as shown below, the objective being to determine its input/output function, F=f(X,Y,Z), and its truth table, construct its timing diagram, and additionally, realize it in alternative ways.

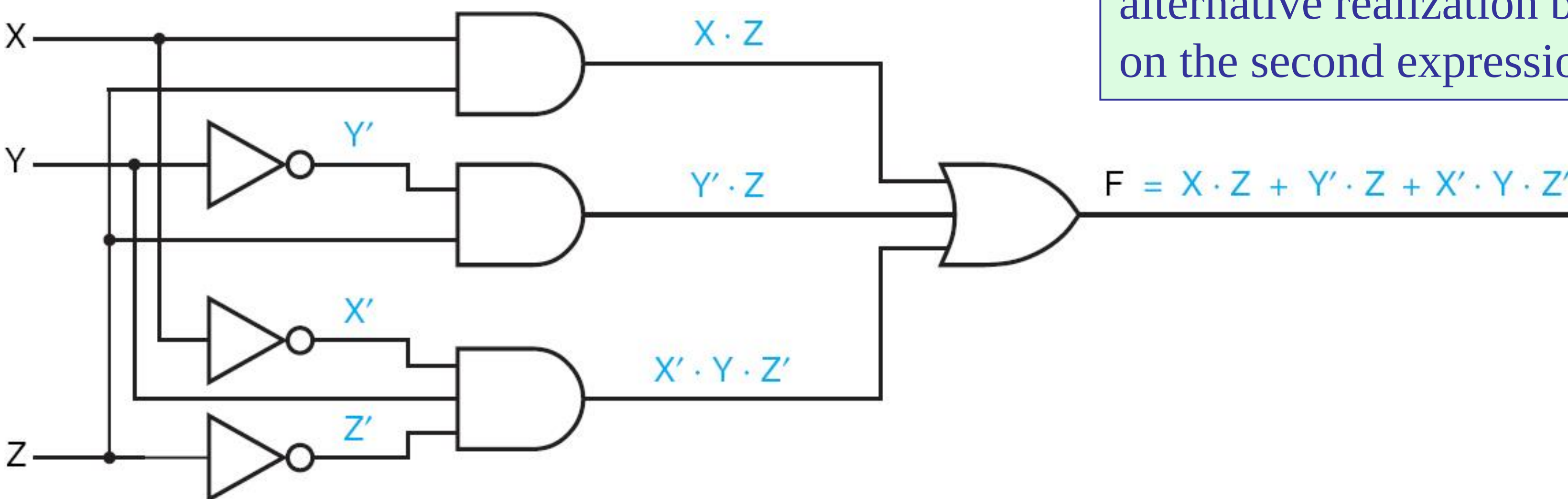
Wakerly, Fig.3-8



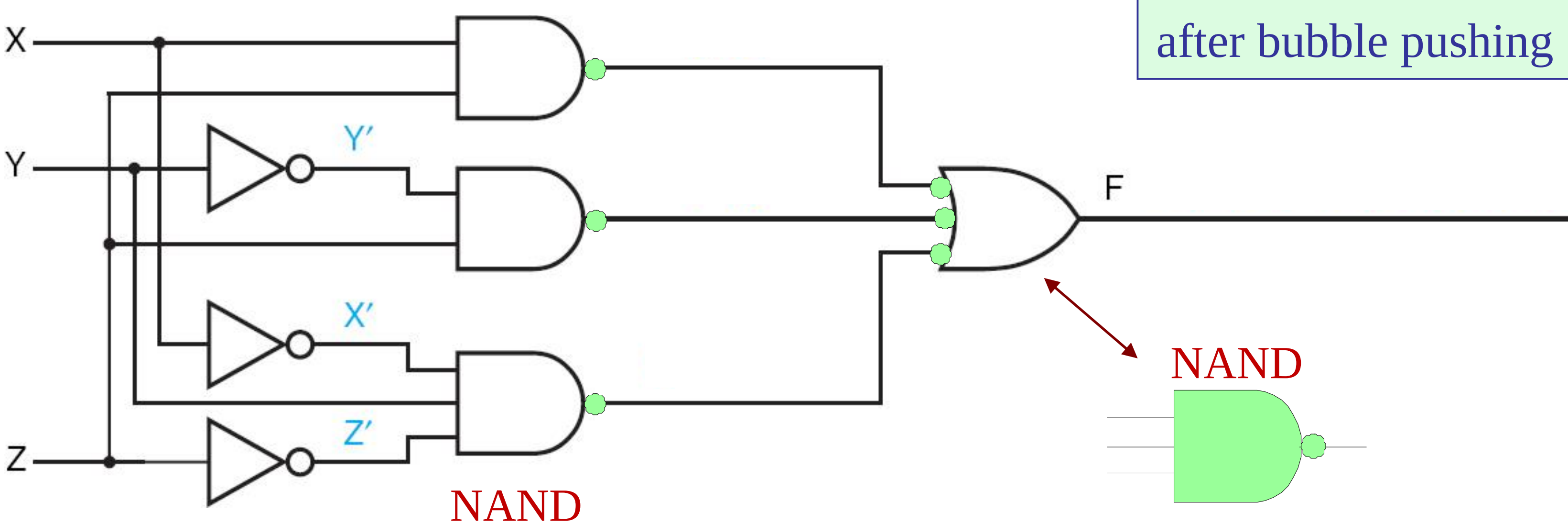
$F = (X+Y')Z + X'YZ' = XZ + Y'Z + X'YZ'$

procedure: assign labels to the circuit lines and implement the indicated logic gate operations from the inputs to the output.

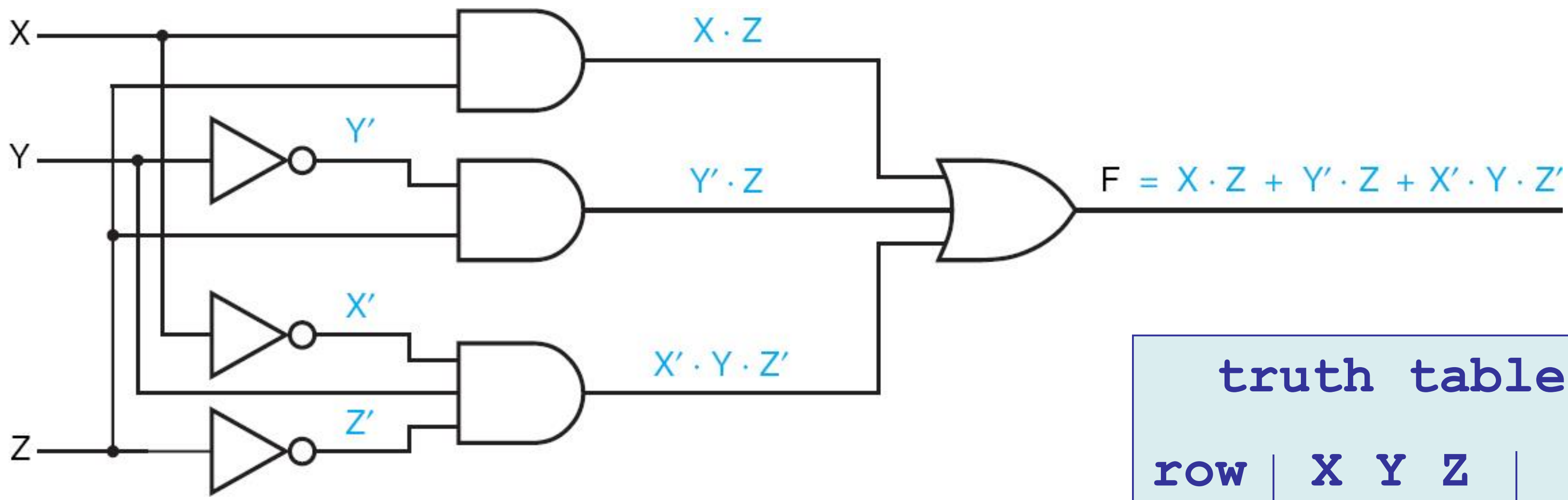
F = (X+Y′)Z + X′YZ′
F = XZ+Y′Z + X′YZ′
alternative realization based on the second expression



F = XZ+Y′Z + X′YZ′
NAND-NAND realization after bubble pushing



$F = XZ + Y'Z + X'YZ'$ truth table



truth table

row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Wakerly, Table 3-7

MATLAB implementation

```
[X,Y,Z] = a2d(0:7,3);                    % 3-bit binary pattern

F = (X & Z) | (~Y & Z) | (~X & Y & ~Z);              % output F

[X,Y,Z,F]                         % collect X,Y,Z,F for printing
                                  % print truth table
% 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 1
% 1 1 0 0
% 1 1 1 1
```

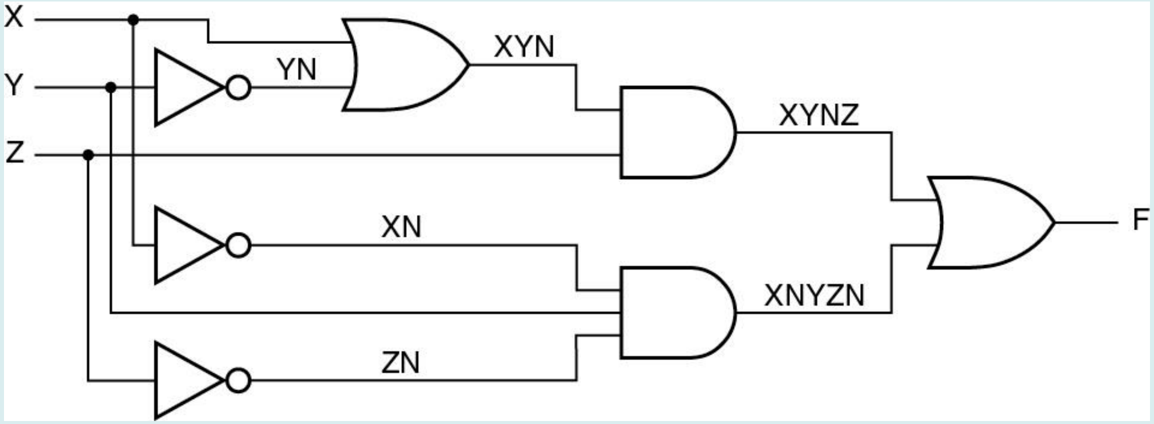
MATLAB code for
$F = XZ+Y'Z + X'YZ'$

the operations **&** and **|** are vectorized

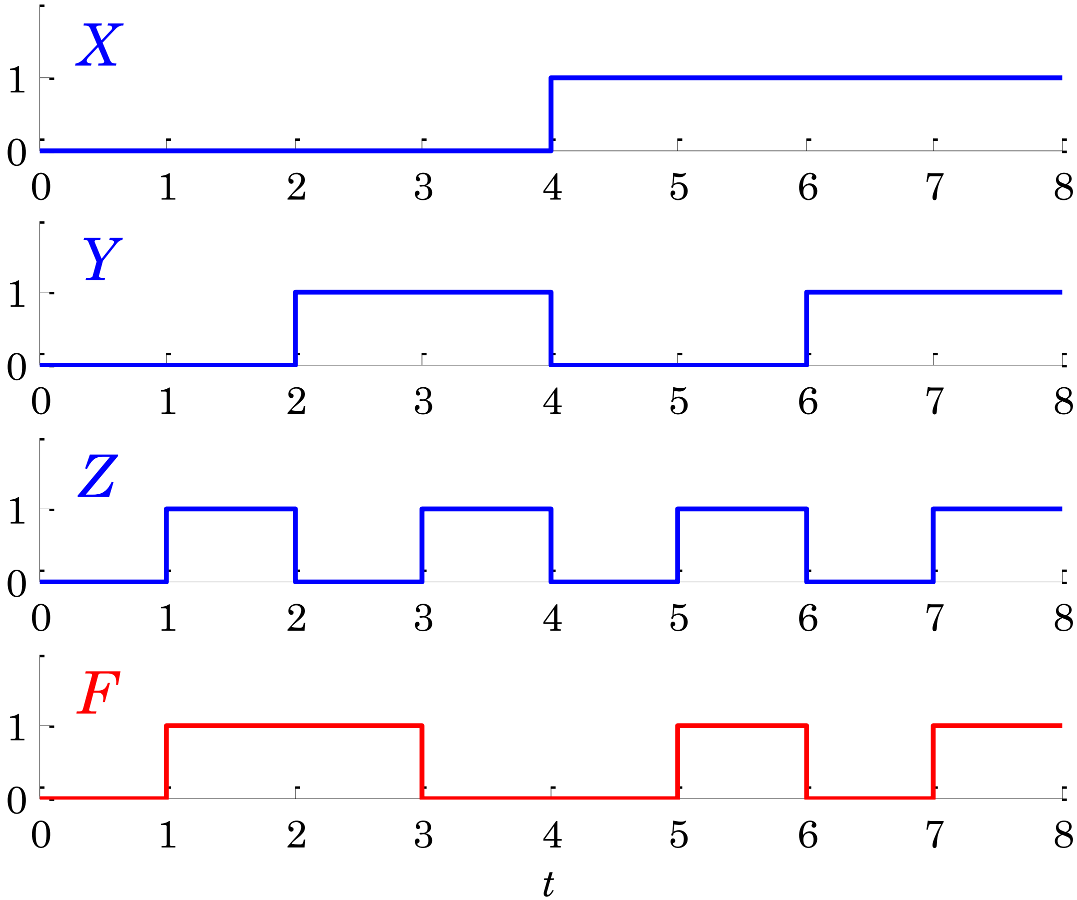
Verilog implementation

```
module Acirc1f(
  input X,Y,Z,
  wire XN, YN, ZN, XYN, XYNZ, XNYZN,
  output wire F
  );

  assign XN = ~X; assign YN = ~Y; assign ZN = ~Z;
  assign XYN = X | YN; assign XYNZ = XYN & Z;
  assign XNYZN = XN & Y  & ZN; assign F = XYNZ | XNYZN;

endmodule
```

Verilog code for
$F = (X + Y')Z + X'YZ'$



F = XZ+Y′Z + X′YZ′ - timing diagram



truth table

row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

F = XZ+Y′Z + X′YZ′ - MATLAB code for timing diagram

```
t = 0:8;

% see p.98 for definitions of X,Y,Z,F

X = [X; X(end)];     % replicate last entries
Y = [Y; Y(end)];
Z = [Z; Z(end)];
F = [F; F(end)];     % t,X,Y,Z now have length 9

figure;
subplot(4,1,1);
    stairs(t,X,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,2);
    stairs(t,Y,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,3);
    stairs(t,Z,'b-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
subplot(4,1,4);
    stairs(t,F,'r-'); yaxis(0,2,0:1); xaxis(0,8,0:8);
xlabel('\itt')
```

$F = (X+Y')Z + X'YZ'$

alternative representation based on product-of-sums

$F = (X+Y'+Z')\,(X'+Z)\,(Y+Z) = (X+Y+Z)\,(X+Y'+Z')\,(X'+Y+Z)\,(X'+Y'+Z)$

it is actually a simplified version of the canonical maxterm expansion, but a direct proof is as follows:

Using the ordinary distributive law, $(X+A)(X+B) = (X+AB)$, we obtain the more general version of the distributive law,

$$(AB + CDE) = (A + C)\,(A + D)\,(A + E)\,(B + C)\,(B + D)\,(B + E)$$

and apply it with, $A=X+Y'$, $B=Z$, $C=X'$, $D=Y$, $E=Z'$, and note that, $X+1 = 1$, $X+X'=1$,

$$\begin{aligned} F &= (X + Y')Z + X'YZ' \\ &= (X + Y' + X')\,(X + Y' + Y)\,(X + Y' + Z')\,(Z + X')\,(Z + Y)\,(Z + Z') \\ &= (1 + Y')\,(X + 1)\,(X + Y' + Z')\,(Z + X')\,(Z + Y)\,1 = \\ &= (X + Y' + Z')\,(X' + Z)\,(Y + Z) \end{aligned}$$

Proof of the generalized distributive law:

$$(AB + CDE) = (A + C)(A + D)(A + E)(B + C)(B + D)(B + E)$$

apply the ordinary distributive law, $(X+A)(X+B) = (X+AB)$, in stages:

$$\begin{aligned} AB + CDE &= (A + CDE)(B + CDE) \\ &= (A + C)(A + DE)(B + C)(B + DE) \\ &= (A + C)(A + D)(A + E)(B + C)(B + D)(B + E) \end{aligned}$$

Next, we look at the minterm / maxterm canonical expansions of this example, and their simplifications

minterm / maxterm representations

truth table

X Y Z	F	F′	minterms	maxterms
0 0 0	0	1		X + Y + Z
0 0 1	1	0	X′Y′Z	
0 1 0	1	0	X′Y Z′	
0 1 1	0	1		X + Y′+ Z′
1 0 0	0	1		X′+ Y + Z
1 0 1	1	0	X Y′Z	
1 1 0	0	1		X′+ Y′+ Z
1 1 1	1	0	X Y Z	

$F = X'Y'Z + X'YZ' + XY'Z + XYZ$ = minterm SOP

$F = (X + Y + Z)\,(X + Y' + Z')\,(X' + Y + Z)\,(X' + Y' + Z)$ = maxterm POS

minterm / maxterm simplifications

minterms:

$$F = X'Y'Z + X'YZ' + XY'Z + XYZ = X'Y'Z + X'YZ' + XY'Z + XY'Z + XYZ$$
$$= (X' + X)Y'Z + X'YZ' + X(Y + Y')Z =$$
$$= Y'Z + X'YZ' + XZ$$

replicated

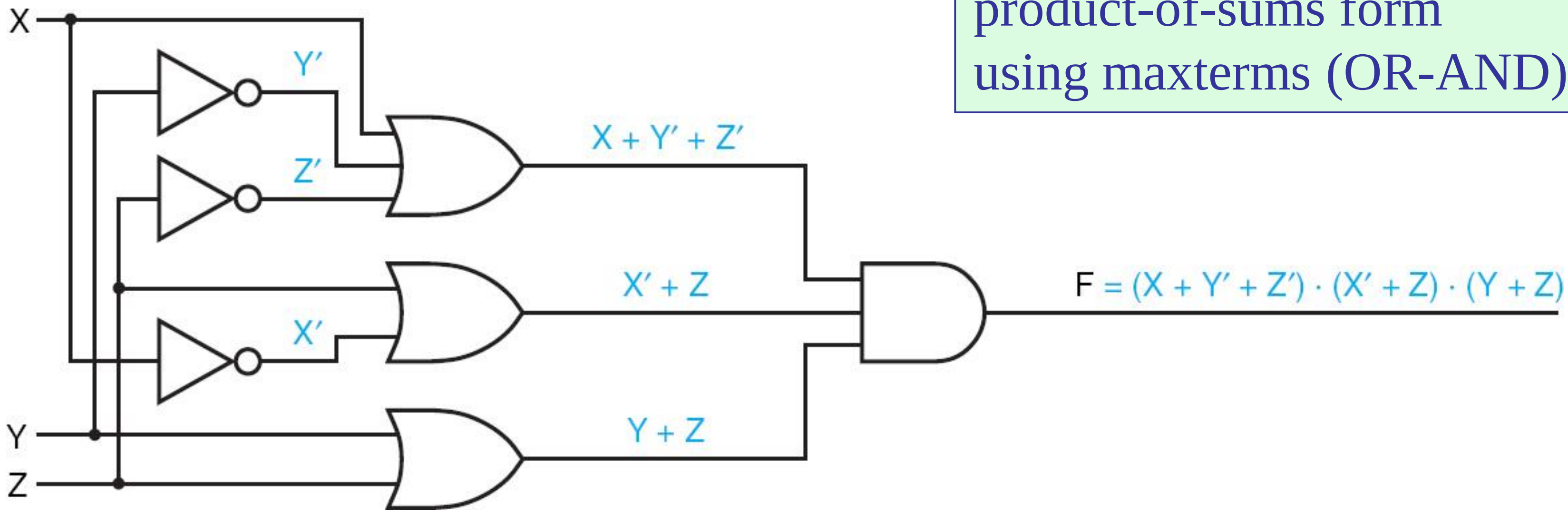
maxterms:

$$F = (X + Y + Z)(X + Y' + Z')(X' + Y + Z)(X' + Y' + Z) =$$
$$= (X + Y + Z)(X + Y' + Z')(X' + Y + Z)(X' + Y + Z)(X' + Y' + Z)$$
$$= (X + Y + Z)(X' + Y + Z)(X' + Y + Z)(X' + Y' + Z)(X + Y' + Z')$$

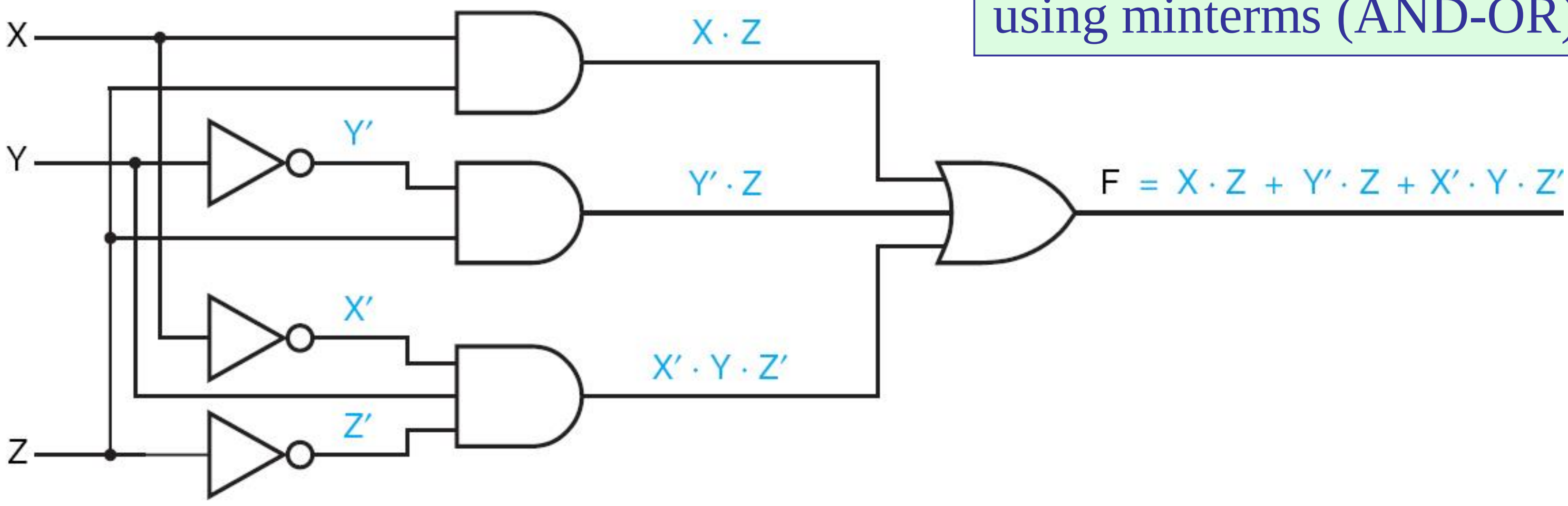
$$= (Y + Z)(X' + Z)(X + Y' + Z')$$

property: $(Y + A)(Y' + A) = A$

product-of-sums form
using maxterms (OR-AND)

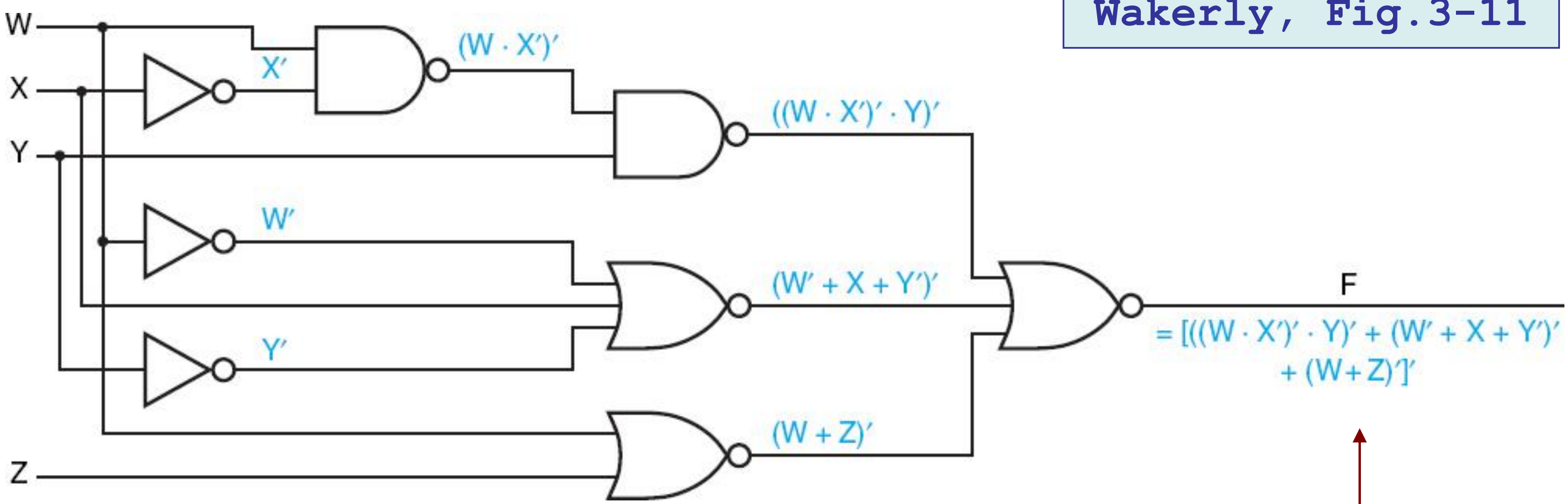


sum-of-products form
using minterms (AND-OR)



17. Combinational circuit analysis – Example 2

Wakerly, Fig.3-11

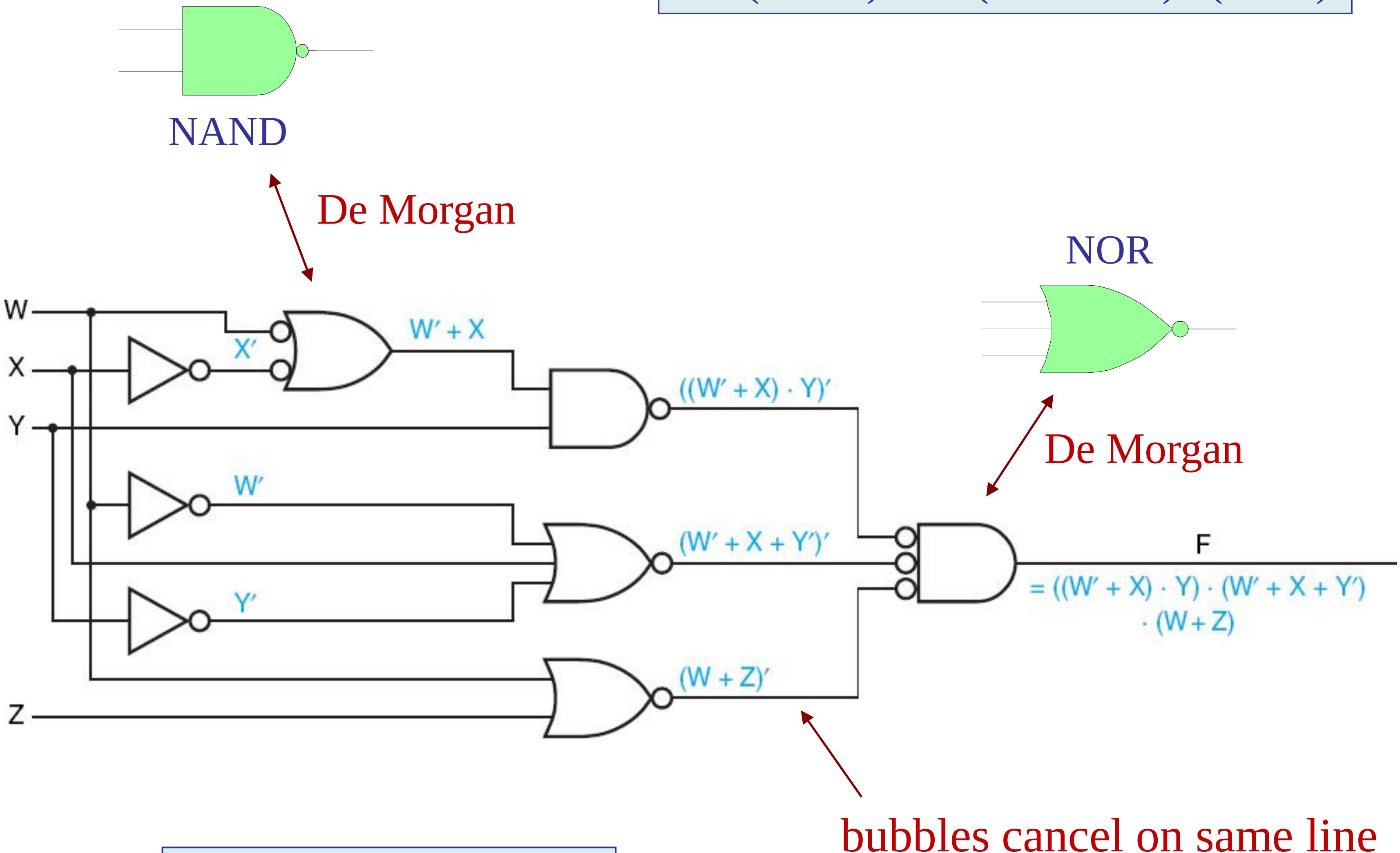


De Morgan

$$F = [((W \cdot X') \cdot 'Y)' + (W' + X + Y')' + (W+Z)']'$$
$$= ((W' + X)' + Y')' \cdot (W \cdot X' \cdot Y) \cdot '(W' \cdot Z')'$$
$$= ((W \cdot X')' \cdot Y \cdot (W' + X + Y') \cdot (W+Z)$$
$$= (W' + X) \cdot Y \cdot (W' + X + Y') \cdot (W+Z)$$

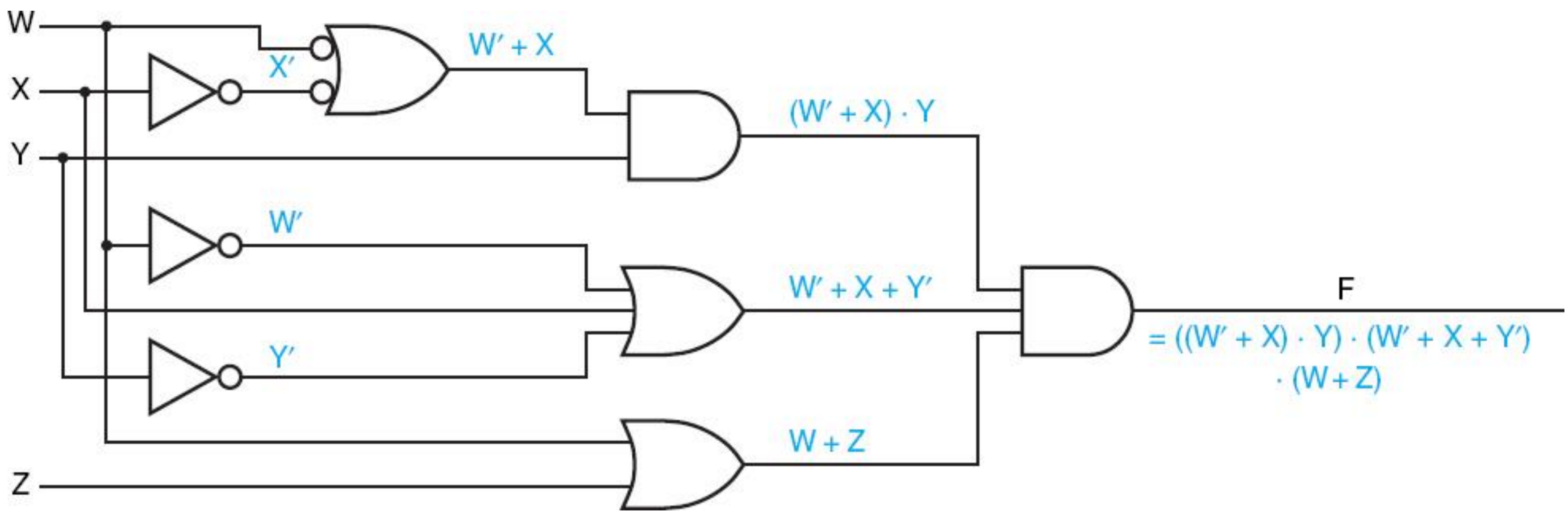
$$F = (W' + X) \cdot Y \cdot (W' + X + Y') \cdot (W+Z)$$

$$F = (W'+X) \cdot Y \cdot (W'+X+Y') \cdot (W+Z)$$



Wakerly, Fig.3-12

$$F = (W'+X) \cdot Y \cdot (W'+X+Y') \cdot (W+Z)$$



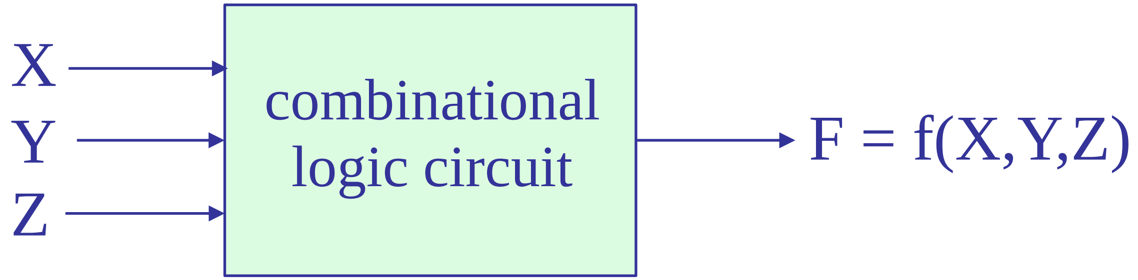
Wakerly, Fig.3-13

18. Combinational circuit synthesis

We recall that the analysis and synthesis problems are:

Analysis Problem: Given a combinational circuit made up of logic gates, determine the output F as a function of the input variables, X,Y,Z…

Synthesis/Design Problem: Given a combinational circuit defined by its I/O mapping, F = f(X,Y,Z,..), typically stated as a truth table, synthesize the circuit with logic gates, preferably using the minimum number of gates, as well as trying to minimize propagation delays.



Below, we present a few synthesis examples based on the circuit's I/O mapping, F = f(X,Y,Z,..), determined from a given (i) truth table, or, (ii) functional description of the circuit. Karnaugh maps provide a systematic synthesis method and will be discussed later on.

18. Combinational circuit synthesis

Example 1 – prime number detector

Wakerly, Sect.3.3.1

canonical minterm SOP form

$$F = \Sigma_{A,B,C,D}(1,2,3,5,7,11,13)$$

$$F = \Pi_{A,B,C,D}(0,4,6,8,9,10,12,14,15)$$

canonical maxterm POS form

row	A	B	C	D	F	minterms
0	0	0	0	0	0	
1	0	0	0	1	1	A′B′C′D
2	0	0	1	0	1	A′B′CD′
3	0	0	1	1	1	A′B′CD
4	0	1	0	0	0	
5	0	1	0	1	1	A′BC′D
6	0	1	1	0	0	
7	0	1	1	1	1	A′BCD
8	1	0	0	0	0	
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	1	AB′CD
12	1	1	0	0	0	
13	1	1	0	1	1	ABC′D
14	1	1	1	0	0	
15	1	1	1	1	0	

Example 1 – prime number detector

$$F = \Sigma_{A,B,C,D}(1,2,3,5,7,11,13) =$$

$$= A'B'C'D + A'B'CD'$$

$$+ A'B'CD + A'BC'D$$

$$+ A'BCD + AB'CD$$

$$+ ABC'D$$

row	A	B	C	D	F	minterms
0	0	0	0	0	0	
1	0	0	0	1	1	$A'B'C'D$
2	0	0	1	0	1	$A'B'CD'$
3	0	0	1	1	1	$A'B'CD$
4	0	1	0	0	0	
5	0	1	0	1	1	$A'BC'D$
6	0	1	1	0	0	
7	0	1	1	1	1	$A'BCD$
8	1	0	0	0	0	
9	1	0	0	1	0	
10	1	0	1	0	0	
11	1	0	1	1	1	$AB'CD$
12	1	1	0	0	0	
13	1	1	0	1	1	$ABC'D$
14	1	1	1	0	0	
15	1	1	1	1	0	

Example 1 – prime number detector

$= (A'B'C'D')'$

$F = \Pi_{A,B,C,D}(0,4,6,8,9,10,12,14,15)$

$= (A+B+C+D) \cdot (A+B'+C+D)$

$\cdot (A+B'+C'+D) \cdot (A'+B+C+D)$

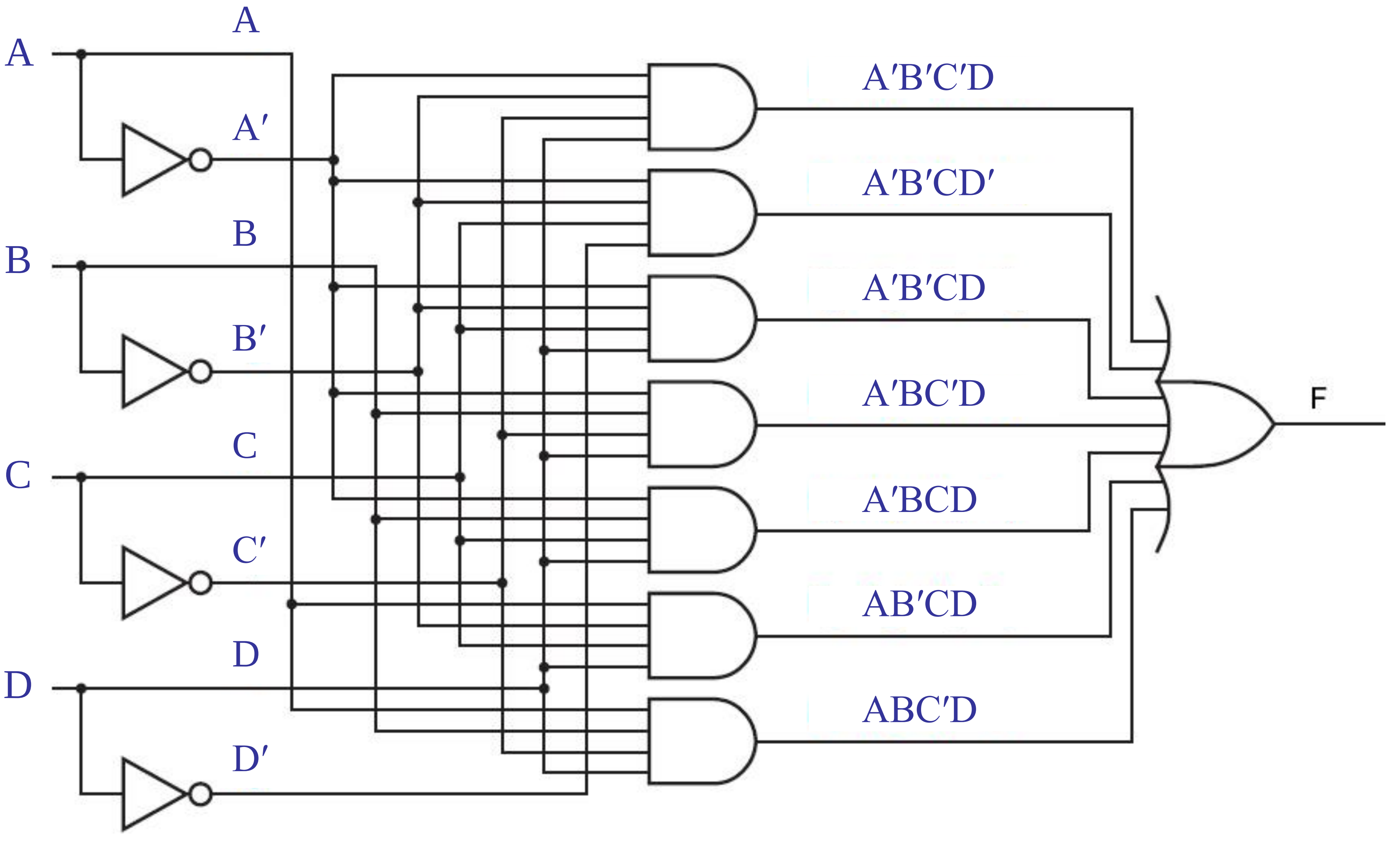
$\cdot (A'+B+C+D') \cdot (A'+B+C'+D)$

$\cdot (A'+B'+C+D) \cdot (A'+B'+C'+D)$

$\cdot (A'+B'+C'+D')$

row	A	B	C	D	F	maxterms
0	0	0	0	0	0	$A+B+C+D$
1	0	0	0	1	1	
2	0	0	1	0	1	
3	0	0	1	1	1	
4	0	1	0	0	0	$A+B'+C+D$
5	0	1	0	1	1	
6	0	1	1	0	0	$A+B'+C'+D$
7	0	1	1	1	1	
8	1	0	0	0	0	$A'+B+C+D$
9	1	0	0	1	0	$A'+B+C+D'$
10	1	0	1	0	0	$A'+B+C'+D$
11	1	0	1	1	1	
12	1	1	0	0	0	$A'+B'+C+D$
13	1	1	0	1	1	
14	1	1	1	0	0	$A'+B'+C'+D$
15	1	1	1	1	0	$A'+B'+C'+D'$

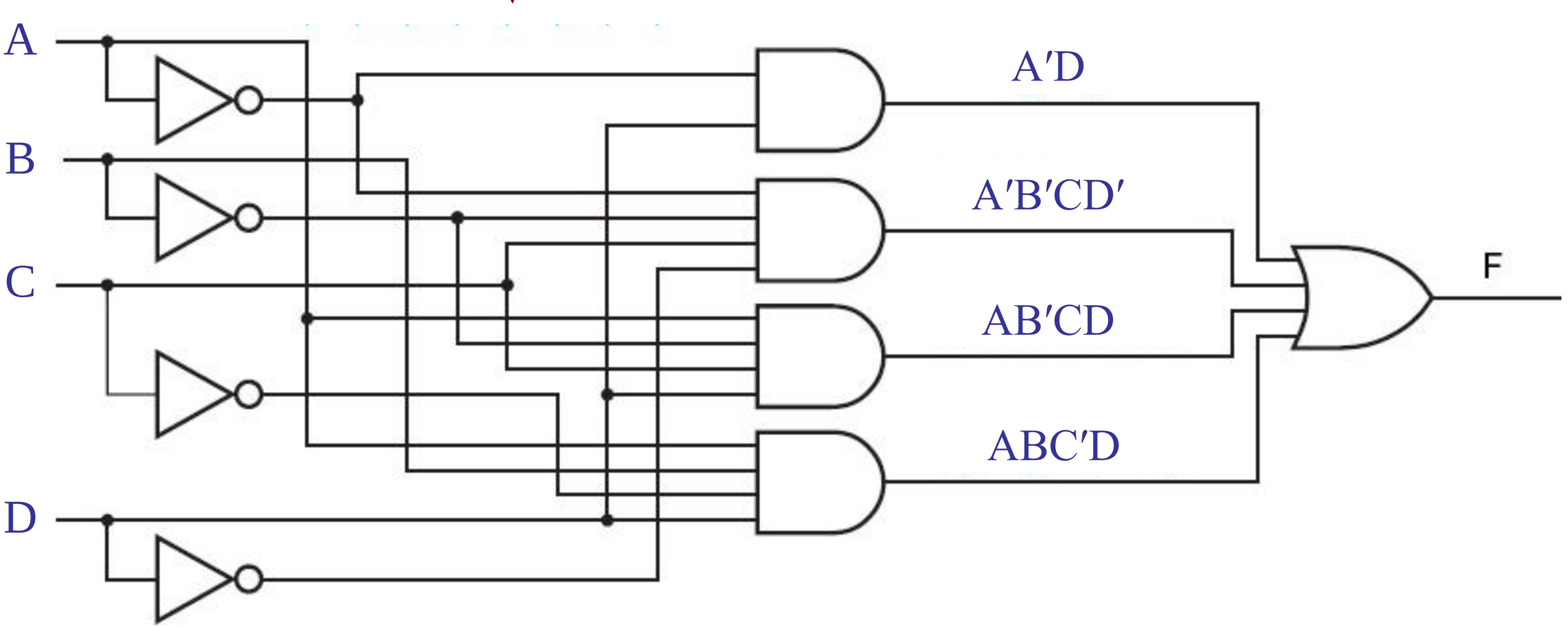
$$F = A'B'C'D + A'B'CD' + A'B'CD + A'BC'D + A'BCD + AB'CD + ABC'D$$



partial simplification:

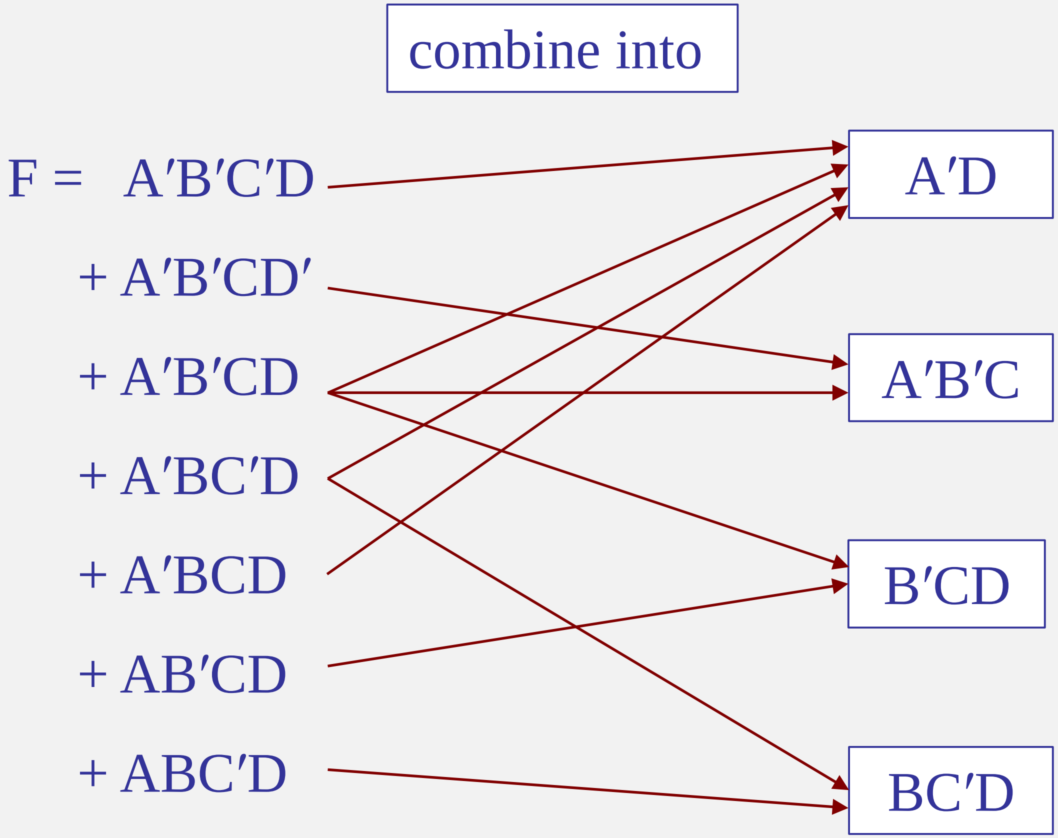
$$F = A'B'C'D + A'B'CD' + A'B'CD + A'BC'D + A'BCD + AB'CD + ABC'D$$
$$= A'C'D + A'B'CD' + A'CD + AB'CD + ABC'D$$

$$F = A'D + A'B'CD' + AB'CD + ABC'D$$



$F = A'B'C'D + A'B'CD' + A'B'CD + A'BC'D + A'BCD + AB'CD + ABC'D$

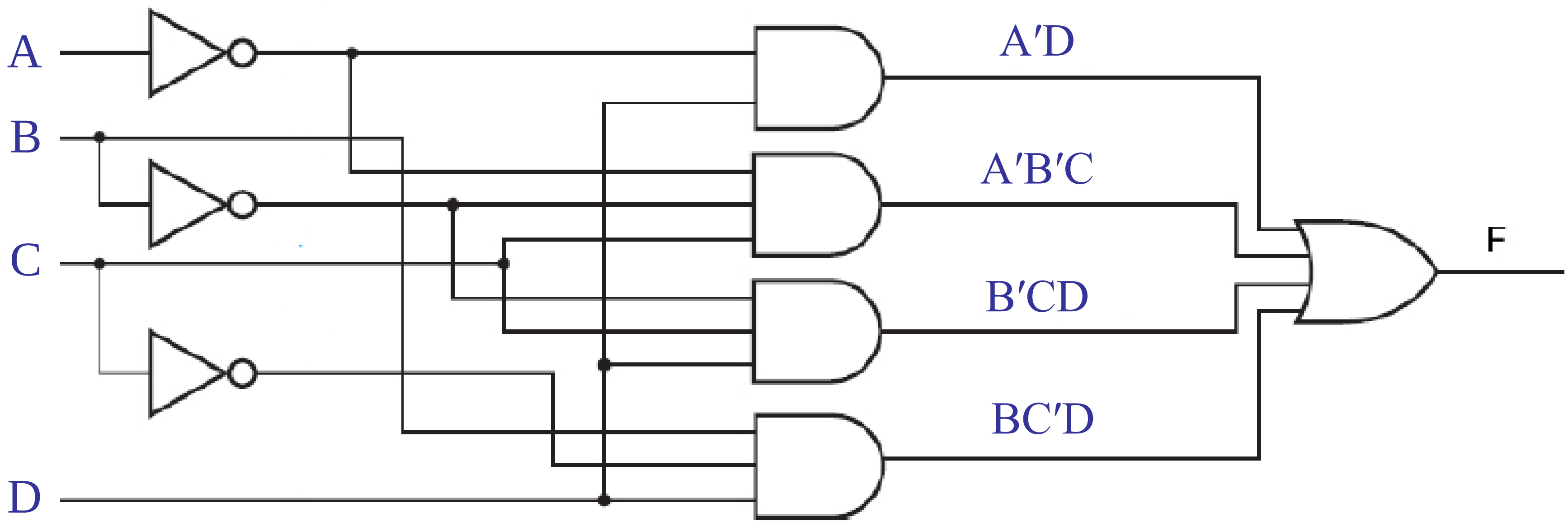
further simplification:



$F = A'D + A'B'C + B'CD + BC'D$

this is also the form obtained with K-maps

Example 1 – prime number detector



$$F = A'D + A'B'C + B'CD + BC'D$$

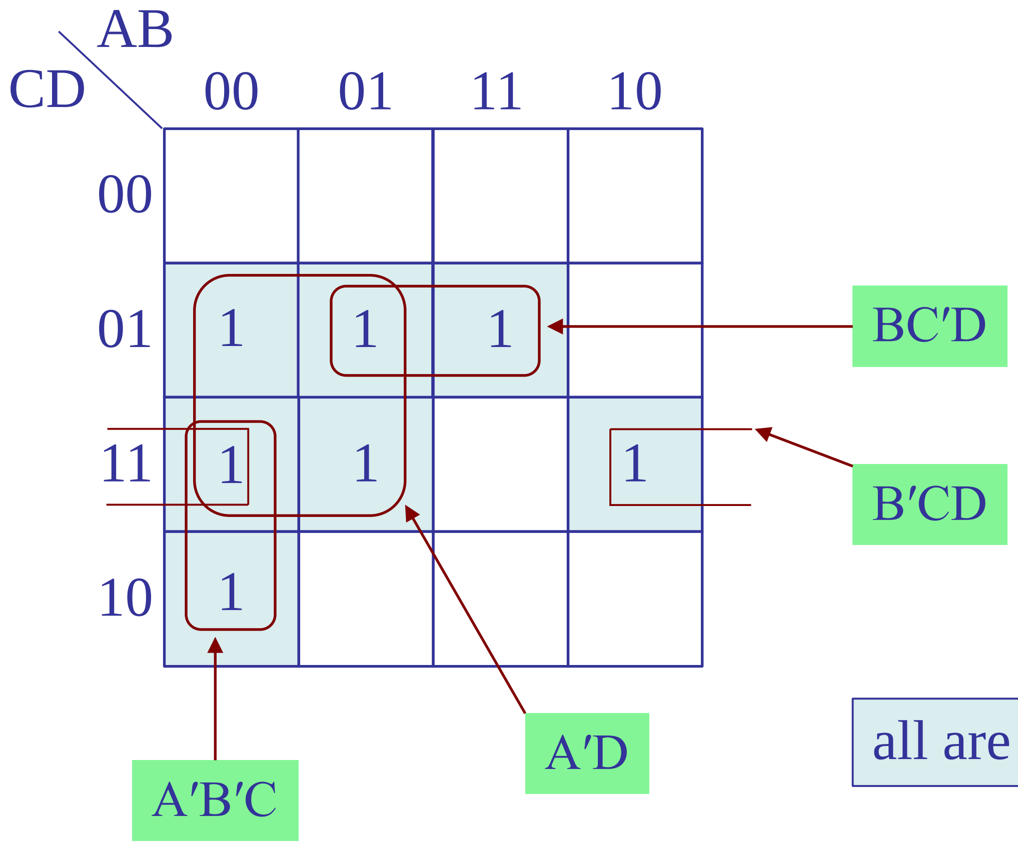
Matlab code:

```
[A,B,C,D] = a2d(0:15, 4);
F = (~A & D) | (~A & ~B & C) | (~B & C & D) | (B & ~C & D);
```

Karnaugh map for prime number detector

$F = A'B'C'D + A'B'CD' + A'B'CD + A'BC'D + A'BCD + AB'CD + ABC'D$

$F = A'D + A'B'C + B'CD + BC'D$



blue areas indicate a complete cover

all are essential PIs

18. Combinational circuit synthesis

Example 2 – alarm circuit

Wakerly, Sect.3.3.1

Design a home alarm circuit whose functional description is as follows:

The ALARM output is 1 if the PANIC-button input is 1, or if the ENABLE input is 1, the EXITING input is 0, and the house is not secure – the house is secure if the WINDOW, DOOR, and GARAGE inputs are all 1.

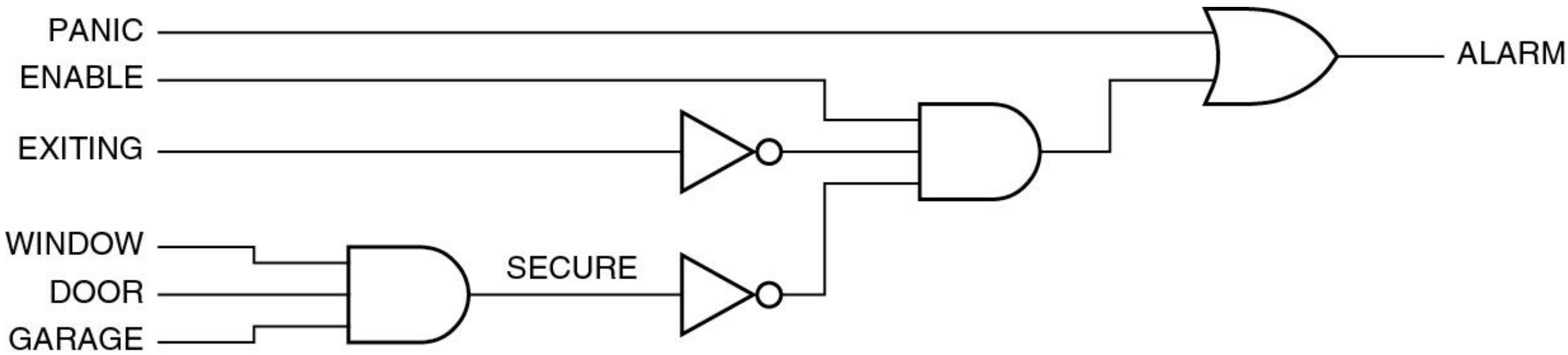
Translating this description into a Boolean algebraic expression, we have:

$$\text{SECURE} = \text{WINDOW} \cdot \text{DOOR} \cdot \text{GARAGE}$$

$$\begin{aligned}
\text{ALARM} &= \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{SECURE}' \\
&= \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot (\text{WINDOW} \cdot \text{DOOR} \cdot \text{GARAGE})' \\
&= \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot (\text{WINDOW}' + \text{DOOR}' + \text{GARAGE}') \\
&= \text{PANIC} + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{WINDOW}' \\
&\qquad + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{DOOR}' \\
&\qquad + \text{ENABLE} \cdot \text{EXITING}' \cdot \text{GARAGE}' = \text{sum-of-products form}
\end{aligned}$$

18. Combinational circuit synthesis

Example 2 – alarm circuit

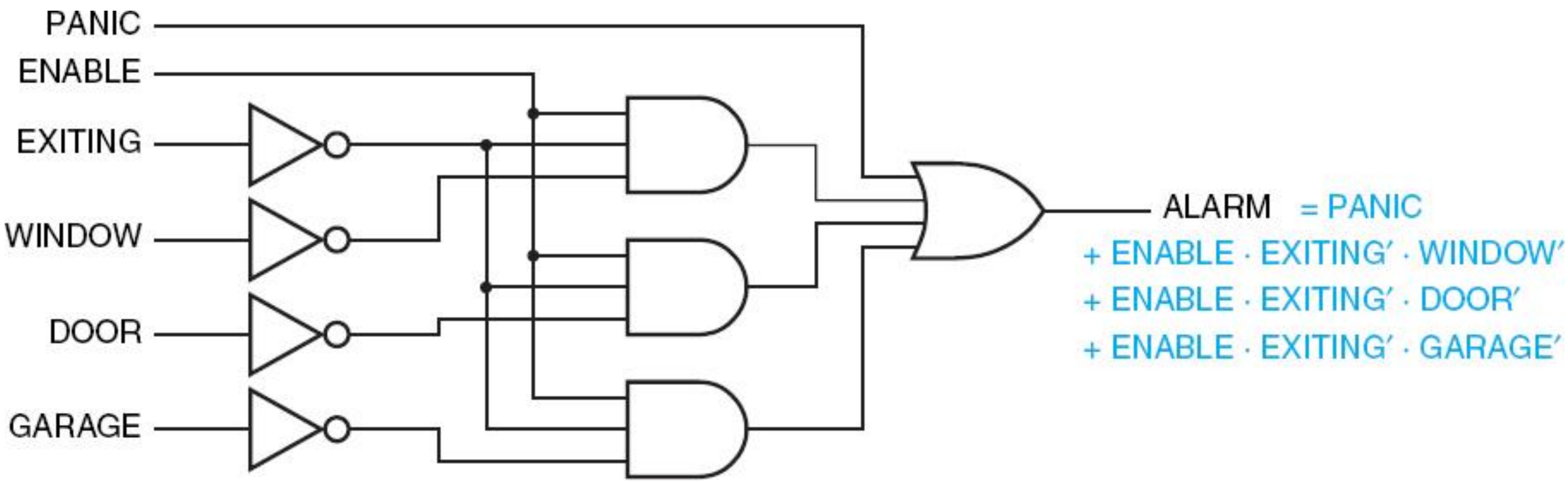


SECURE = WINDOW · DOOR · GARAGE

ALARM = PANIC + ENABLE · EXITING′ · SECURE′

This design has a serious limitation: If the alarm is set off because the house is not secure, then the alarm will turn off when the house becomes secure again, even though the alarm is still enabled (e.g., someone may break into the house through an alarmed door, and cause the alarm to turn off by simply closing the door.) See unit-8, Example-1 for a similar example and how to fix it by introducing memory into the system (with D flip-flops).

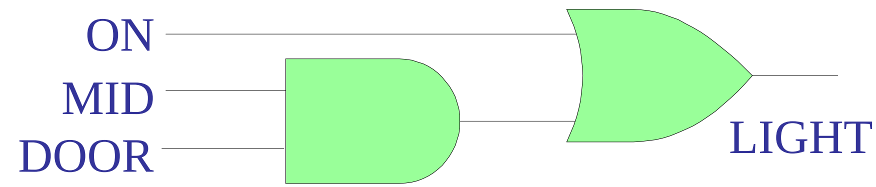
Example 2 – alarm circuit



alternative realization

ALARM = PANIC + ENABLE · EXITING′ · WINDOW′
+ ENABLE · EXITING′ · DOOR′
+ ENABLE · EXITING′ · GARAGE′ = sum-of-products form

18. Combinational circuit synthesis



Example 3 – car dome light

Determine the Boolean function for a car dome light based on the following description [cf. Wakerly]:

The light has a 3-position switch such that the light turns on if the switch is in the ON position or if the middle switch MID is on and the door signal DOOR is also on when any door is open, otherwise the light is off when the switch is in the OFF position.

Translating this description into a Boolean algebraic expression, we have:

$$\mathrm{LIGHT} = \mathrm{ON} + \mathrm{MID} \cdot \mathrm{DOOR}$$

The circuit diagram with three inputs, ON, MID, DOOR, and one output LIGHT, is easily drawn using one AND gate and one OR gate.

see p.163 for an alternative realization, and K-map derivations with don’t care entries

18. Combinational circuit synthesis

Example 4 – equality test

Given two 2-bit numbers, **a**, **b**, determine the Boolean function F that is equal to 1 when the two numbers are equal, **a** = **b**, and is equal to 0 otherwise.

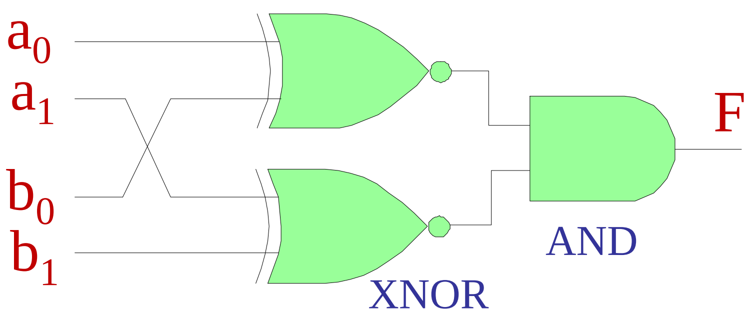
Let the two bits of each number be, **a** = $(a_1 a_0)$, and, **b** = $(b_1 b_0)$

$$F = (a_1 b_1 + a_1' b_1') \cdot (a_0 b_0 + a_0' b_0') = \text{XNOR}(a_1, b_1) \cdot \text{XNOR}(a_0, b_0)$$

The circuit diagram with four inputs, a_1 , a_0 , b_1 , b_0 , and one output F, is easily drawn using one AND gate and two XNOR gates

note, xnor(A,B) = 1 only if A=B (either both 0 or both 1) →

A	B	$AB + A'B'$
0	0	1
0	1	0
1	0	0
1	1	1



19. Combinational circuit minimization – Karnaugh maps

Karnaugh maps (K-maps) are two-dimensional representations of truth tables that provide an intuitive way to simplify a logic circuit and realize it with fewer logic gate operations.

See also,

Karnaugh maps - Wikipedia

Karnaugh maps are convenient for 1-5 input variables. For more variables see the following more advanced methods,

logic minimization methods - Wikipedia

Quine-McCluskey algorithm - Wikipedia

Petrick's method - Wikipedia

Consider a 3-variable function, F = f(A,B,C). In an ordinary truth table, the possible values of the Boolean variables A,B,C, and the corresponding values of F, are listed in linear arithmetic progression such that the row numbers are represented by their 3-bit binary-pattern, ABC, with each row corresponding to a particular minterm.

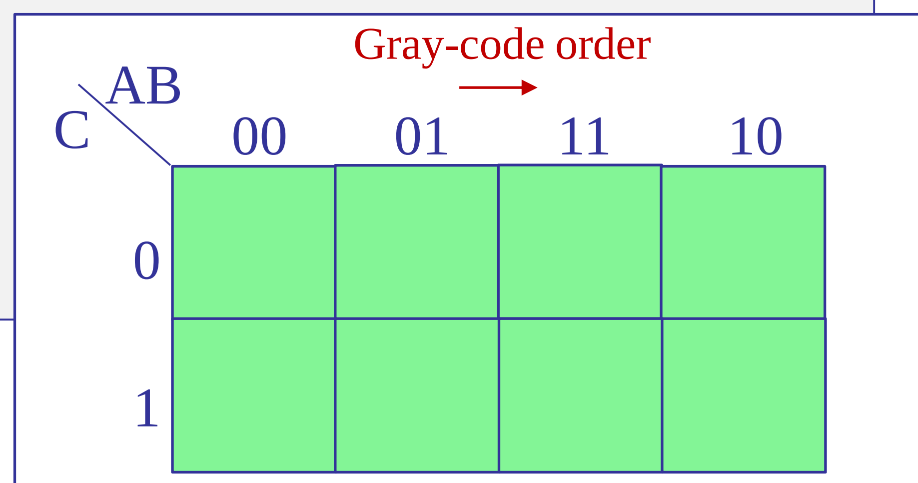
In each row, the function F is either 0 or 1, and the corresponding minterms for which F=1 are added to represent the function as a sum-of-products.

In the Karnaugh map, on the other hand, the AB values are listed horizontally and the C values, vertically. However, the AB values are not listed in ordinary binary-order, but rather in Gray-code order such that only one bit changes in moving across from column to column. We recall from unit-2 that the ordinary 2-bit binary-order for AB is:

AB = 00, 01, 10, 11

whereas the Gray-code order is:

AB = 00, 01, 11, 10



ordinary truth table

row	A B C	minterms
0	0 0 0	A′B′C′
1	0 0 1	A′B′C
2	0 1 0	A′BC′
3	0 1 1	A′BC
4	1 0 0	AB′C′
5	1 0 1	AB′C
6	1 1 0	ABC′
7	1 1 1	ABC

Karnaugh map

Gray-code order

C \ AB	00	01	11	10
0	A′B′C′	A′BC′	ABC′	AB′C′
1	A′B′C	A′BC	ABC	AB′C

For a particular function, F = f(A,B,C), some of the indicated minterms will be replaced by 0’s and some by 1’s.

Two adjacent minterms, either horizontally or vertically, combine to a simpler expression by eliminating that variable that has changed across the pair of minterms.

This is a consequence of the identity, $X + X' = 1$.

The number of adjacent minterms must always be a power of 2, that is, the number of grouped minterms must be 1, 2, 4, for a 3-variable function, or, 1,2,4,8, for a 4-variable function.

Moreover, grouped adjacent minterms can overlap either horizontally or vertically.

To understand the simplification mechanism, consider a few examples of groupings.

two adjacent terms, horizontally or vertically

C \ AB	00	01	11	10
0	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
1	$A'B'C$	$A'BC$	ABC	$AB'C$

$A'BC' + ABC' =$
$(A' + A)\,BC' = BC'$

here, A is changing
and was eliminated

C \ AB	00	01	11	10
0	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
1	$A'B'C$	$A'BC$	ABC	$AB'C$

$A'BC' + A'BC =$
$A'B(C' + C) = A'B$

here, C is changing
and was eliminated

two horizontally overlapping groups of 2

C \ AB	00	01	11	10
0	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
1	$A'B'C$	$A'BC$	ABC	$AB'C$

$A'BC' + ABC' = (A' + A)BC' = BC'$
$ABC' + AB'C' = A(B + B')C' = AC'$
total $= BC' + AC'$

Variable A varies across the first group, and B across the second. The ABC' term was replicated based on the property,

$X + X = X$

two vertically overlapping groups of 2

C \ AB	00	01	11	10
0	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
1	$A'B'C$	$A'BC$	ABC	$AB'C$

replicated term

$A'BC' + ABC' + A'BC' + A'BC =$
$= (A' + A)BC' + A'B(C' + C) =$
$= BC' + A'B$

group of 4 horizontal/vertical terms

C \ AB	00	01	11	10
0	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
1	$A'B'C$	$A'BC$	ABC	$AB'C$

$$A'BC' + ABC' + A'BC + ABC$$
$$= (A' + A)BC' + (A' + A)BC$$
$$= BC' + BC =$$
$$= B(C' + C) = B$$

The variable A varies horizontally, and C, vertically. Both A,C were eliminated.

group of 4 horizontally adjacent terms

C \ AB	00	01	11	10
0	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
1	$A'B'C$	$A'BC$	ABC	$AB'C$

$$A'B'C' + A'BC' + ABC' + AB'C'$$
$$= A'(B' + B)C' + A(B + B')C'$$
$$= A'C' + AC' = (A' + A)C'$$
$$= C'$$

Variables A,B vary horizontally across adjacent cells, and were eliminated.

left and right edges wrap around

C \ AB	00	01	11	10
0	A′B′C′	A′BC′	ABC′	AB′C′
1	A′B′C	A′BC	ABC	AB′C

$A'B'C' + AB'C' \ + \ A'B'C + AB'C$
$= \ (A' + A)B'C' \ + \ (A' + A)B'C$
$= \ B'C' \ + \ B'C =$
$= \ B'(C' \ + \ C) = B'$

Variable A varies horizontally, and C, vertically across the wrapped square, thus, both A,C were eliminated.

left and right edges wrap around

C \ AB	00	01	11	10
0	A′B′C′	A′BC′	ABC′	AB′C′
1	A′B′C	A′BC	ABC	AB′C

$A'B'C' + AB'C'$
$= (A' + A)\ B'C' = B'C'$

Variable A varies horizontally across the wrapped cells, and was eliminated.

row	A B C D	minterms
0	0 0 0 0	A′B′C′D′
1	0 0 0 1	A′B′C′D
2	0 0 1 0	A′B′CD′
3	0 0 1 1	A′B′CD
4	0 1 0 0	A′BC′D′
5	0 1 0 1	A′BC′D
6	0 1 1 0	A′BCD′
7	0 1 1 1	A′BCD
8	1 0 0 0	AB′C′D′
9	1 0 0 1	AB′C′D
10	1 0 1 0	AB′CD′
11	1 0 1 1	AB′CD
12	1 1 0 0	ABC′D′
13	1 1 0 1	ABC′D
14	1 1 1 0	ABCD′
15	1 1 1 1	ABCD

4-variable Karnaugh map

Gray-code order

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

Gray-code order

Karnaugh map

truth-table row-number ordering

B \ A	0	1
0	0	2
1	1	3

C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

simplifies into = A′B

variables C,D vary vertically across adjacent cells, and are eliminated.

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

simplifies into = CD

variables A,B vary horizontally across adjacent cells, and are eliminated.

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

simplifies into $= A'B + CD$

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	$A'B'C'D'$	$A'BC'D'$	$ABC'D'$	$AB'C'D'$
01	$A'B'C'D$	$A'BC'D$	$ABC'D$	$AB'C'D$
11	$A'B'CD$	$A'BCD$	$ABCD$	$AB'CD$
10	$A'B'CD'$	$A'BCD'$	$ABCD'$	$AB'CD'$

simplifies into $= B'C' + BC$

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

simplifies into = BC′ + AB

always choose the largest sub-areas of adjacent cells, containing 2, 4, or, 8 minterms

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

simplifies into = BD + AC

4-variable Karnaugh map

simplifies into = ABD′ + B′C′

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

top/bottom and left/right edges wrap around

4-variable Karnaugh map

simplifies into = A′D′ + B′C

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

top/bottom and left/right edges wrap around

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

top/bottom and left/right corners wrap around

simplifies into = B′D′

4-variable Karnaugh map

A′C′ BC′ AB

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

AC

always choose the largest sub-areas of adjacent cells, containing 2, 4, or, 8 minterms

simplifies into

$= A'C' + AC + BC'$

or, into

$= A'C' + AC + AB$

the BC′ and AB areas are partially covered by each other, and by the other areas A′C′ and AC, so only one of them, BC′ or AB, needs to be included – including both would be redundant

4-variable Karnaugh map

CD \ AB	00	01	11	10
00	A′B′C′D′	A′BC′D′	ABC′D′	AB′C′D′
01	A′B′C′D	A′BC′D	ABC′D	AB′C′D
11	A′B′CD	A′BCD	ABCD	AB′CD
10	A′B′CD′	A′BCD′	ABCD′	AB′CD′

B′D′ essential PI

A′C

BCD essential PI

simplifies into
= B′D′ + A′C + BCD

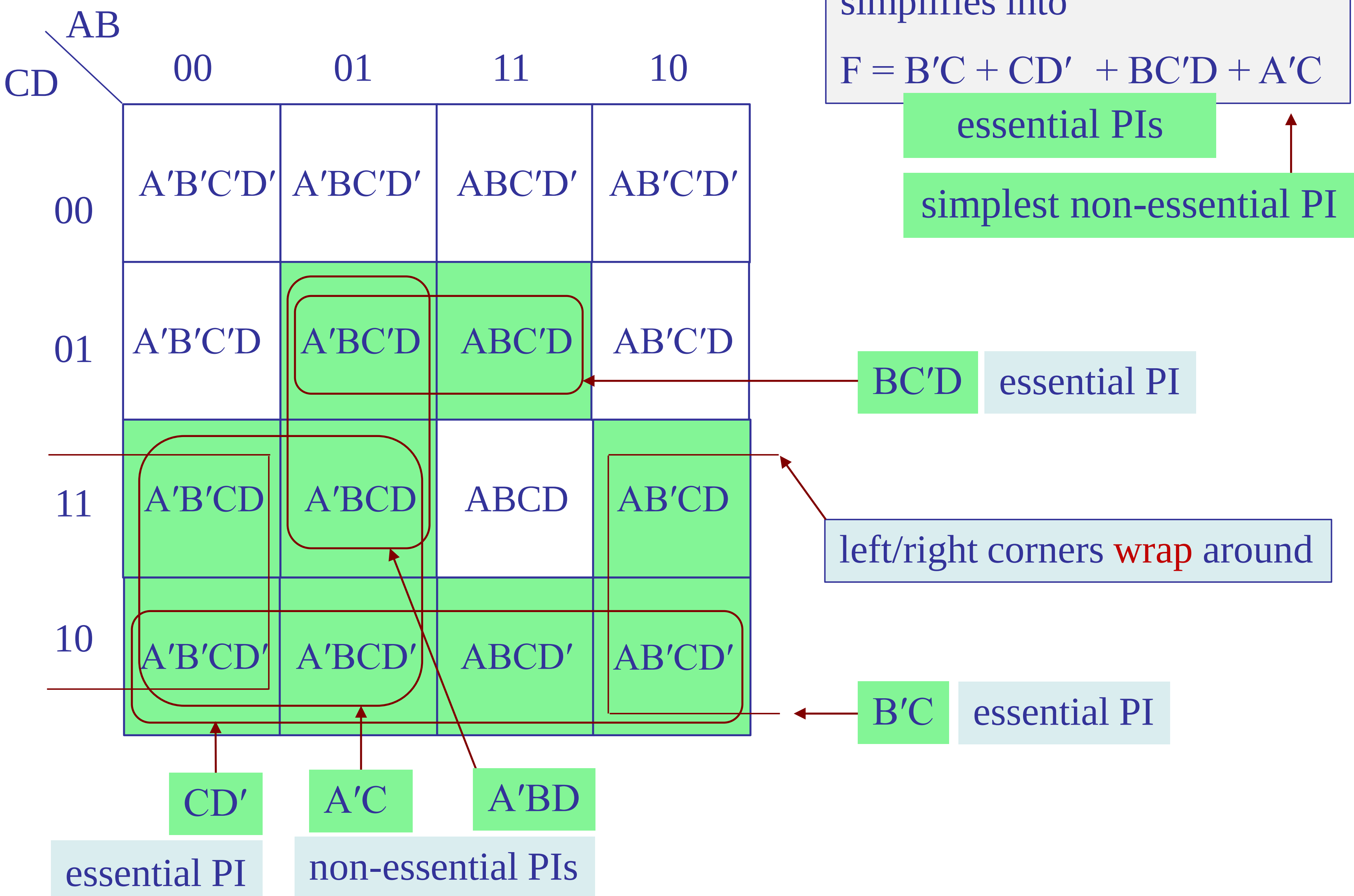
top/bottom and left/right corners wrap around

always choose the largest sub-areas of adjacent cells, containing 2, 4, or, 8 minterms

4-variable Karnaugh map

simplifies into

F = B′C + CD′ + BC′D + A′C



Nomenclature

implicant: a minterm or sum of minterms appearing in a function F, if an implicant evaluates to 1, then so does F as a whole, i.e., if, implicant=1, then it implies, F=1

prime implicant: a simplified implicant that cannot be combined into another implicant that has fewer number of literals.

covers: all implicants that account for all possible evaluations of the function into F=1 (i.e., all the 1's in a Karnaugh map).

essential prime implicant: a prime implicant that contains an F=1 minterm that is not included in any other prime implicant,

all essential prime implicants must be included in the cover of the function.

In addition to the essential PIs, it may be necessary to include possible non-essential PIs in order to achieve a complete cover, (if there are several such possibilities, one could choose the one that has the smallest number of literals.

blue areas indicate a complete cover

simplifies into = $A'D + ABD' + AC'D'$

CD \ AB	00	01	11	10
00			1	1
01	1	1		
11	1	1		
10			1	

(non-prime) implicant

essential prime implicants

essential prime implicants

CD \ AB	00	01	11	10
00	1			
01	1	1	1	
11	1	1	1	
10	1			

simplifies into = $A'B' + BD$

= $A'B' + BD + A'D$ = with all PIs + consensus

non-essential PI, already covered by other essential PIs, consensus term

Summary

Karnaugh map minimization steps for combinational functions of 2, 3, or, 4 Boolean variables:

1. Place 1's in the squares of the K-map for those minterms where F=1.
2. For each such minterm, find the largest sub-area containing that minterm – these are the prime implicants. The number of elements in such subareas must be a power of 2, e.g., 2,4,8,etc.
3. Identify the essential prime implicants covering those minterms that are not covered by any other prime implicant.
4. All of the essential prime implicants must be included in the final simplified expression, and in addition, include any other prime implicants so that all minterms of the function are covered.
5. Any particular minterm may be covered by more than one prime implicant, but all minterms must be covered.

Karnaugh map examples – 1

Previously (in Example 3) we considered the simplification of the truth table function given in Table 3-5 of the Wakerly text, and demonstrated the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

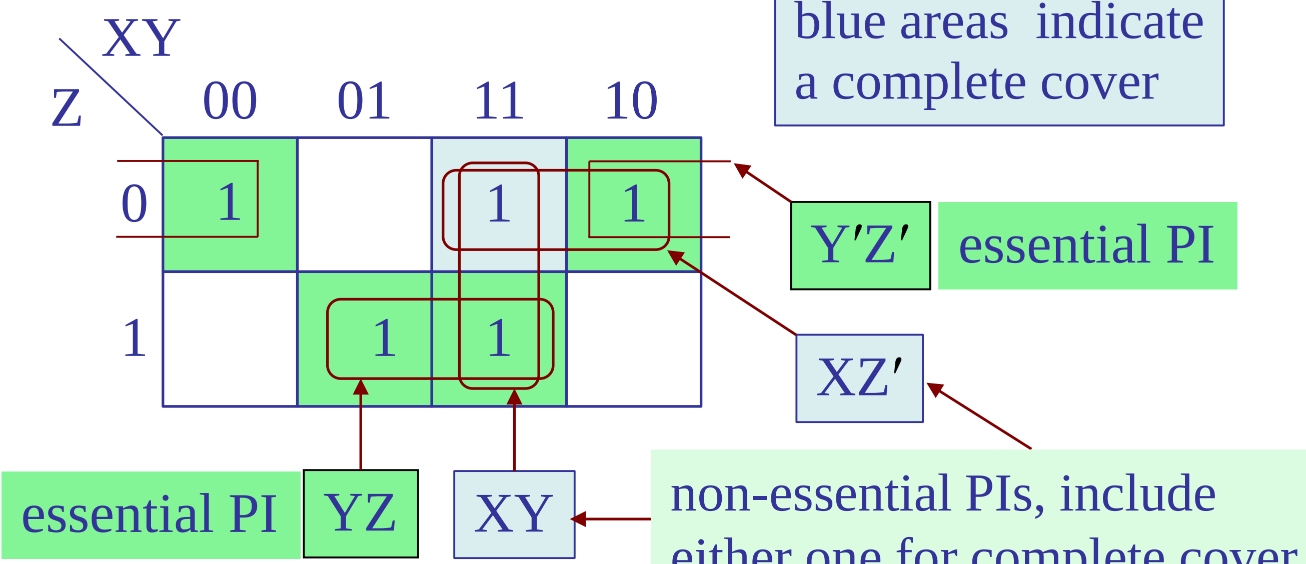
(1) $F = X'Y'Z' + XY'Z' + X'YZ + XYZ + XYZ'$

(2) $F = Y'Z' + YZ + XZ'$

(3) $F = Y'Z' + YZ + XY$

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



blue areas indicate a complete cover

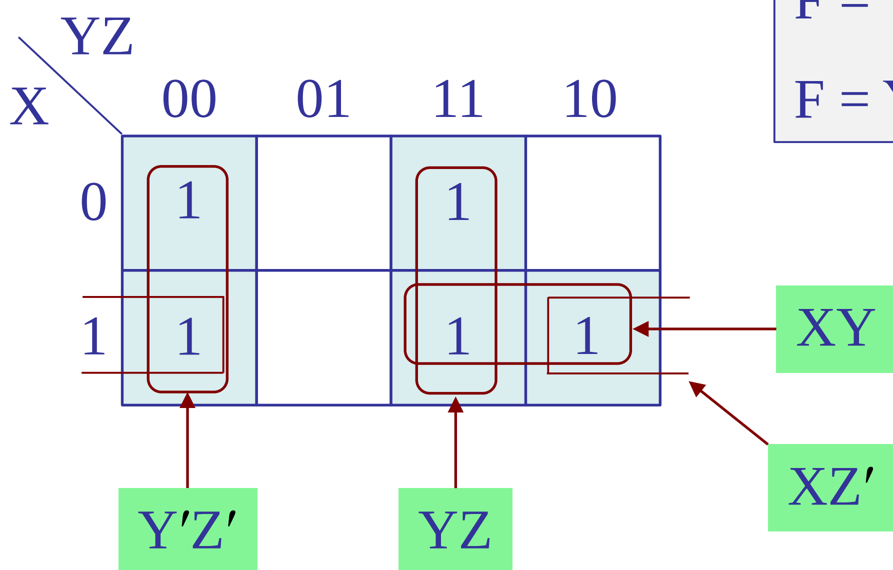
non-essential PIs, include either one for complete cover

Karnaugh map examples – 1

alternative way of arranging the variables in the K-map – the final answer is the same

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



essential PIs

include either one for complete cover

$F = Y'Z' + YZ + XZ'$

$F = Y'Z' + YZ + XY$

blue areas indicate a complete cover

Karnaugh map examples – 1

yet, another way of drawing the K-map

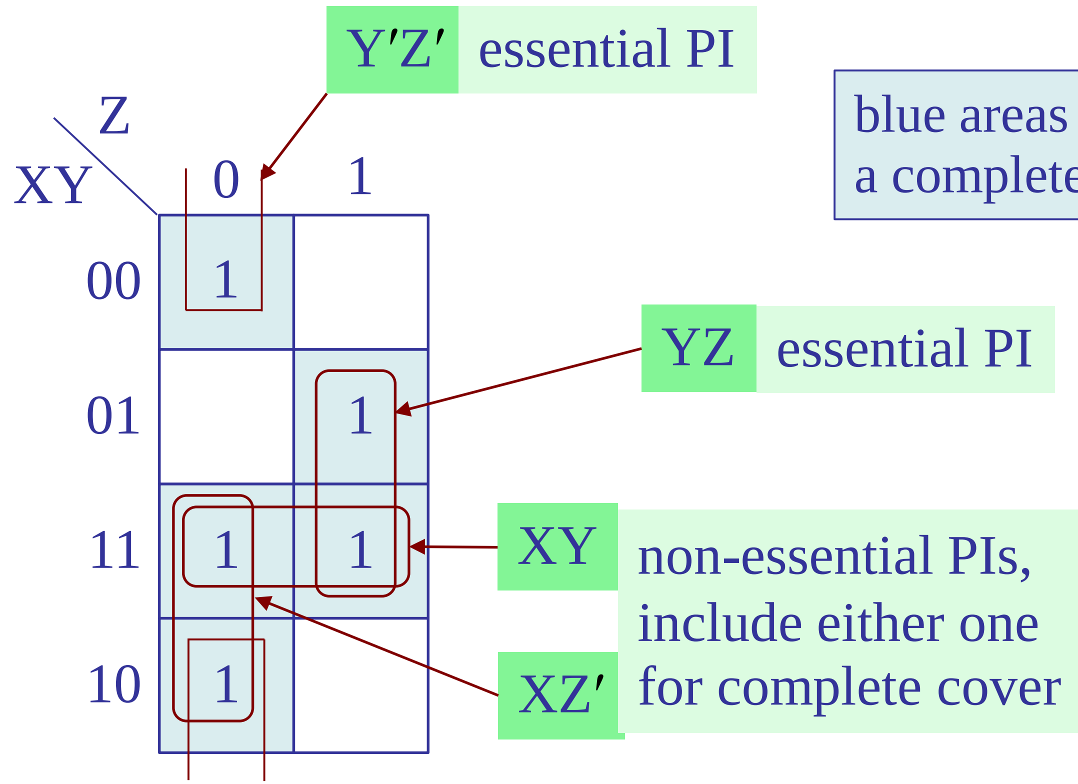
$F = Y'Z' + YZ + XZ'$

$F = Y'Z' + YZ + XY$

blue areas indicate a complete cover

truth table

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

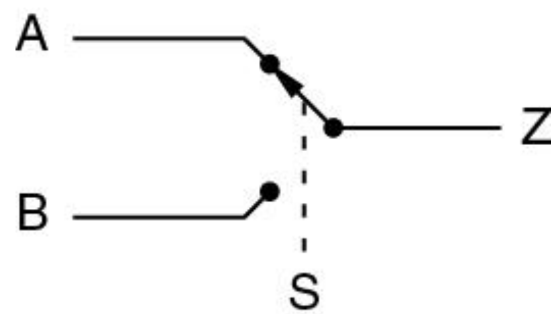


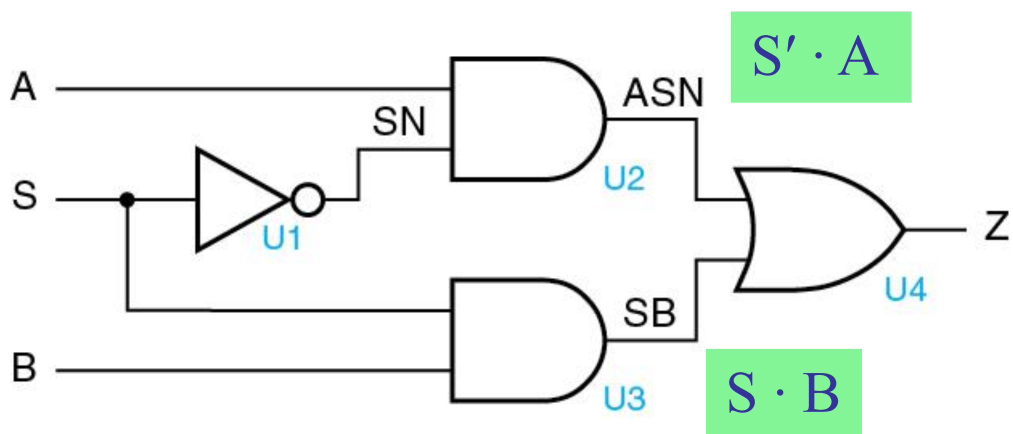
Karnaugh map examples – 2

Truth table of multiplexer function and its simplification:

$$Z = S' \cdot A \cdot B' + S' \cdot A \cdot B + S \cdot A' \cdot B + S \cdot A \cdot B$$
$$= S' \cdot A + S \cdot B$$

	S	A	B	Z
	0	0	0	0
	0	0	1	0
$S' \cdot A \cdot B'$	0	1	0	1
$S' \cdot A \cdot B$	0	1	1	1
	1	0	0	0
$S \cdot A' \cdot B$	1	0	1	1
	1	1	0	0
$S \cdot A \cdot B$	1	1	1	1



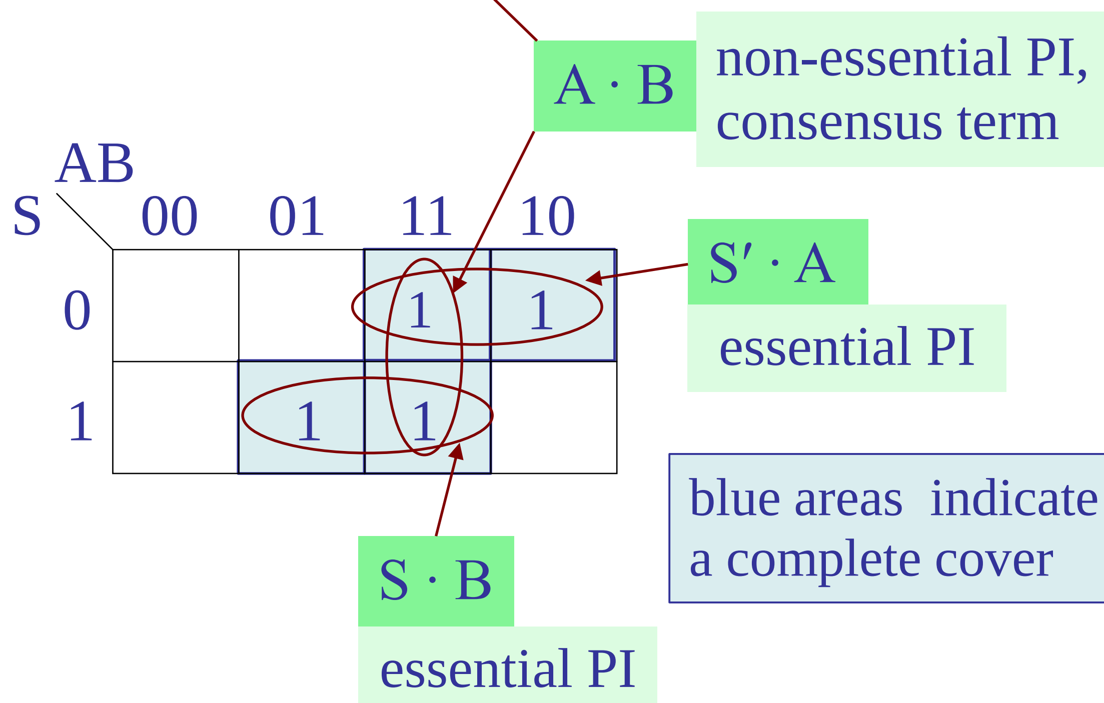


Karnaugh map examples – 2

Truth table of multiplexer function and its simplification:

$$Z = S' \cdot A \cdot B' + S' \cdot A \cdot B + S \cdot A' \cdot B + S \cdot A \cdot B$$
$$= S' \cdot A + S \cdot B = S' \cdot A + S \cdot B + A \cdot B$$

	S	A	B	Z
	0	0	0	0
	0	0	1	0
$S' \cdot A \cdot B'$	0	1	0	1
$S' \cdot A \cdot B$	0	1	1	1
	1	0	0	0
$S \cdot A' \cdot B$	1	0	1	1
	1	1	0	0
$S \cdot A \cdot B$	1	1	1	1



Karnaugh map examples – 3

Previously (in Example 4), we considered the truth table given below [ref. A. F. Kana], and showed the equivalence of the following expressions for F as a function of the Boolean variables X, Y, Z,

truth table

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(1) $F = X'YZ + XY'Z + XYZ' + XYZ$

(2) $F = XY + YZ + XZ$

F may be viewed as implementing a voting majority gate, that is, F=1, if two or more input variables are 1

Z \ XY	00	01	11	10
0	1		1	
1		1	1	1

XY

XZ

YZ

blue areas indicate a complete cover

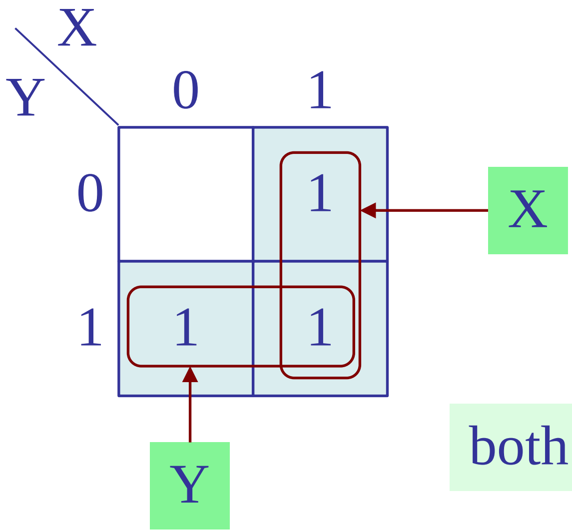
all three are essential PIs

Karnaugh map examples – 4

Previously (in Example 5) we showed the equivalence of the following two expressions:

$$F = X'Y + XY' + XY$$
$$F = X + Y$$

It can be understood simply by a 2-variable K-map.



blue areas indicate a complete cover

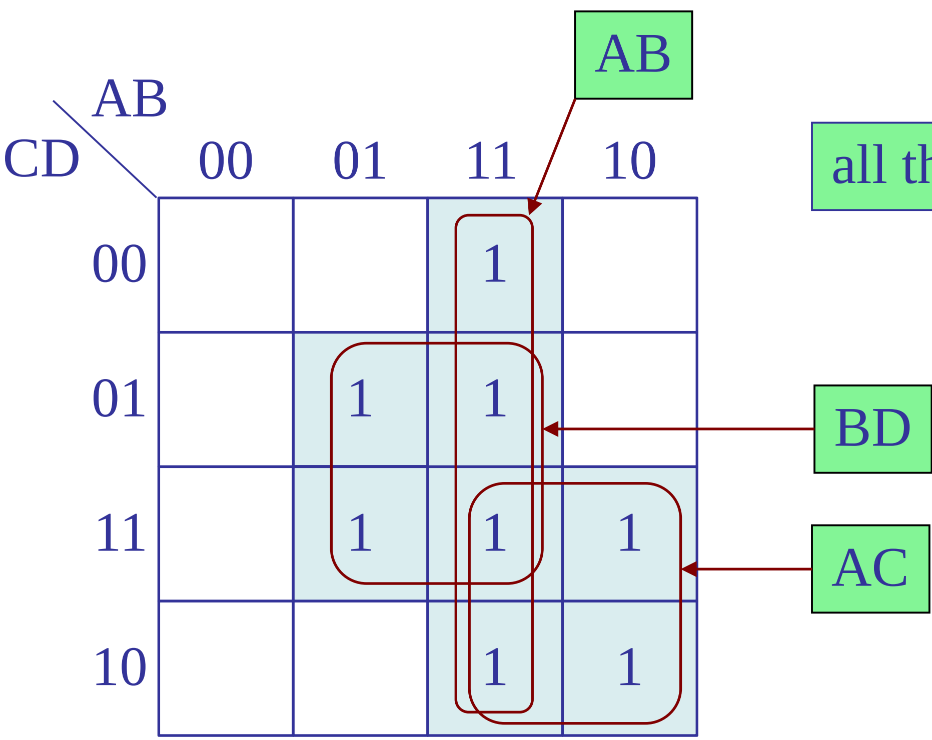
both are essential PIs

Karnaugh map examples – 5

Previously (in Example 6), we showed the equivalence of the following two 4-variable expressions [ref. A. F. Kana],

$$F = A'BC'D + A'BCD + ABC'D' + ABC'D + ABCD + ABCD' + AB'CD + AB'CD'$$

$$F = BD + AB + AC$$



all three are essential PIs

blue areas indicate a complete cover

example of implicant

if $AB = 1$, then, $A=1$, $B=1$, $A'=0$, $B'=0$

$F = C'D' + C'D + CD + CD'$

$\quad = (C + C')(D + D') = 1 \cdot 1 = 1$

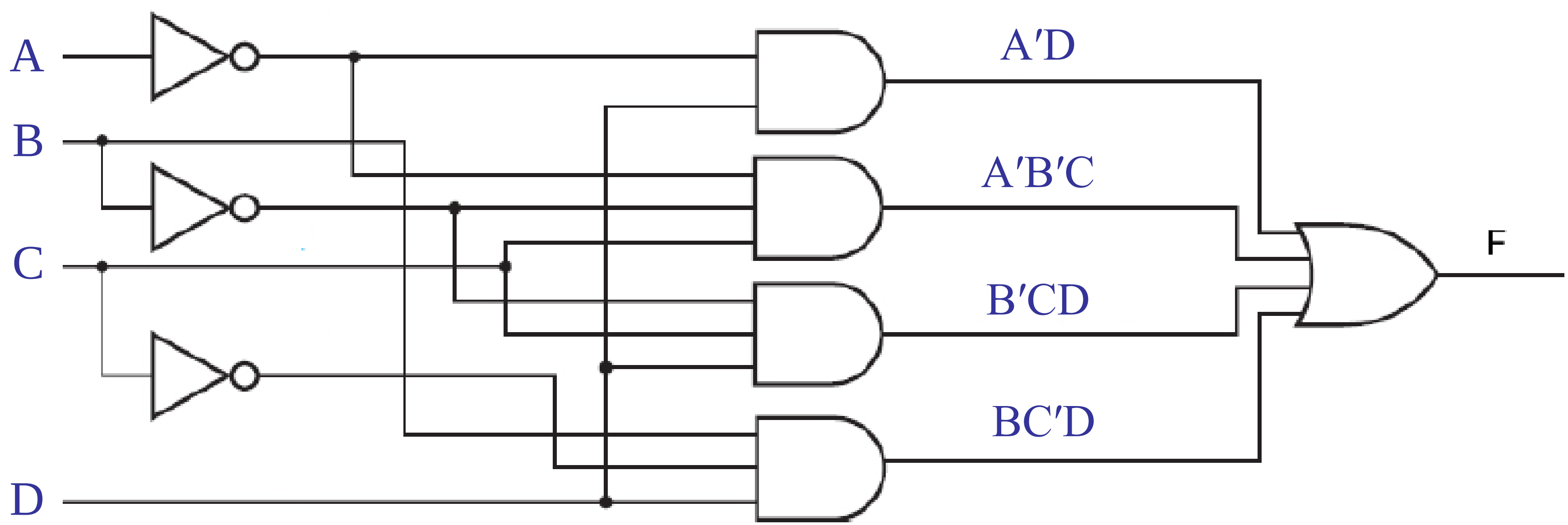
Karnaugh map examples – 6

Previously, we arrived at the following equivalent expressions for a prime number detector,

$F = \Sigma_{A,B,C,D}(1,2,3,5,7,11,13) =$

$F = A'B'C'D + A'B'CD' + A'B'CD + A'BC'D + A'BCD + AB'CD + ABC'D$

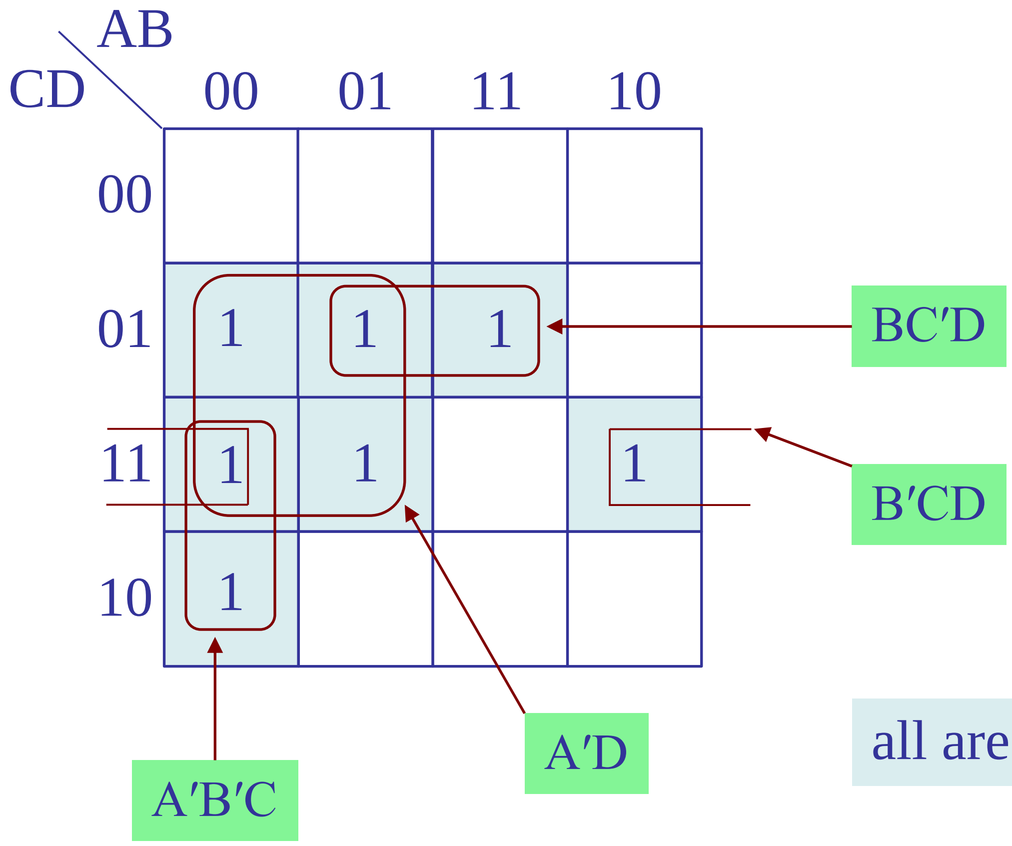
$F = A'D + A'B'C + B'CD + BC'D$



Karnaugh map examples – 6

$F = A'B'C'D + A'B'CD' + A'B'CD + A'BC'D + A'BCD + AB'CD + ABC'D$

$F = A'D + A'B'C + B'CD + BC'D$



blue areas indicate a complete cover

all are essential PIs

Karnaugh map examples – 7

Determine the minimum product-of-sums expression for the function F with the following truth table.

row	A B C D	F	F′
0	0 0 0 0	1	0
1	0 0 0 1	0	1
2	0 0 1 0	1	0
3	0 0 1 1	0	1
4	0 1 0 0	1	0
5	0 1 0 1	1	0
6	0 1 1 0	1	0
7	0 1 1 1	1	0
8	1 0 0 0	1	0
9	1 0 0 1	0	1
10	1 0 1 0	1	0
11	1 0 1 1	0	1
12	1 1 0 0	1	0
13	1 1 0 1	0	1
14	1 1 1 0	1	0
15	1 1 1 1	1	0

K-map for F′

CD \ AB	00	01	11	10
00				
01	1		1	1
11	1			1
10				

AC′D

B′D

F′ = B′D + AC′D

De Morgan

F = (B + D′)(A′ + C + D′)

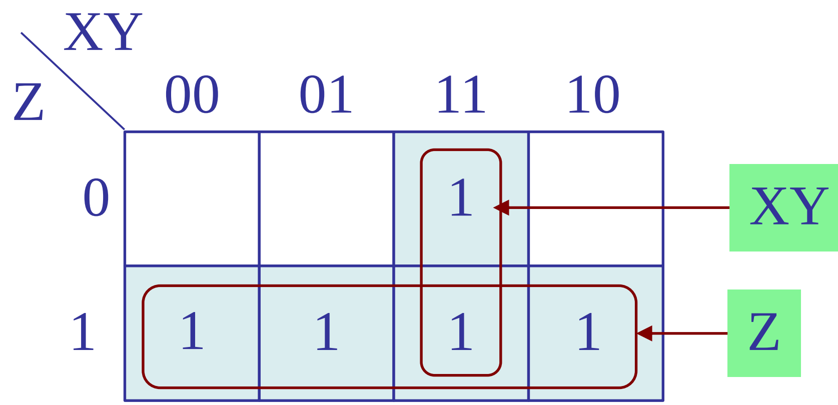
Karnaugh map examples – 8

Using K-maps, determine a simplified sum-of-products form for the function,

$$F = XY + ZX' + ZY'$$

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	1
1	1	1	1



$$F = XY + Z$$

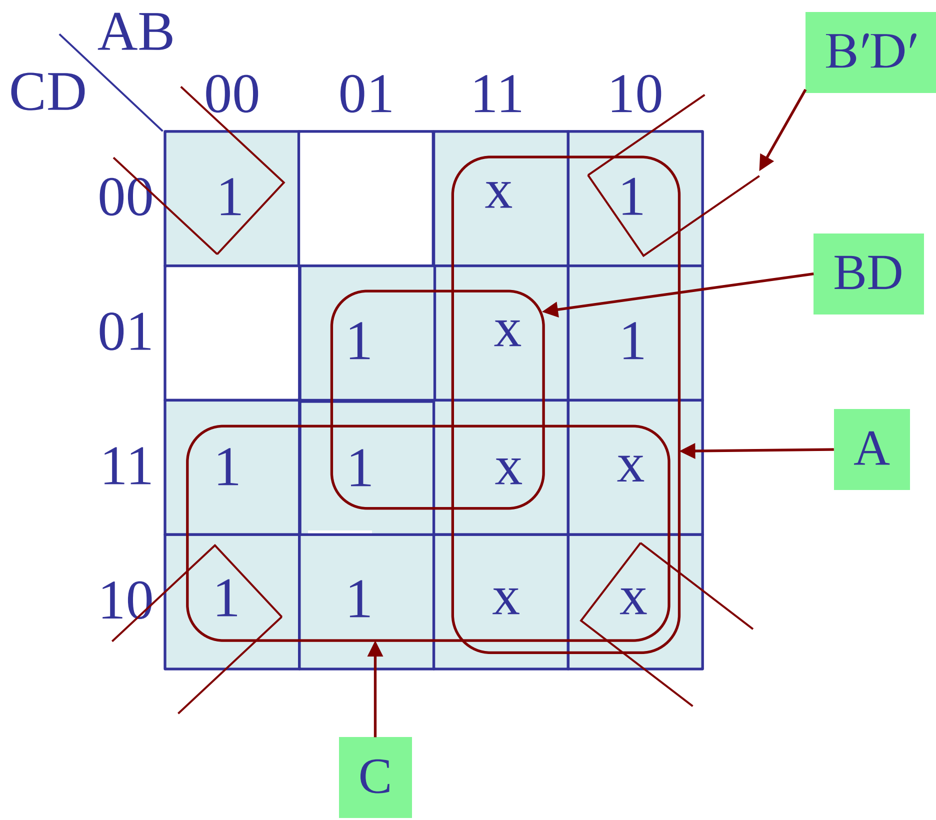
```
[x,y,z] = a2d(0:7,3);
f = (x&y) | (z&~x) | (z&~y);
[x,y,z,f]
```

Karnaugh map examples – 9

How to handle incompletely specified, or don’t care, entries in the truth table.

rule: treat them as 1’s in the minterm form

row	A B C D	F
0	0 0 0 0	1
1	0 0 0 1	0
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	0
5	0 1 0 1	1
6	0 1 1 0	1
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	1
10	1 0 1 0	x
11	1 0 1 1	x
12	1 1 0 0	x
13	1 1 0 1	x
14	1 1 1 0	x
15	1 1 1 1	x



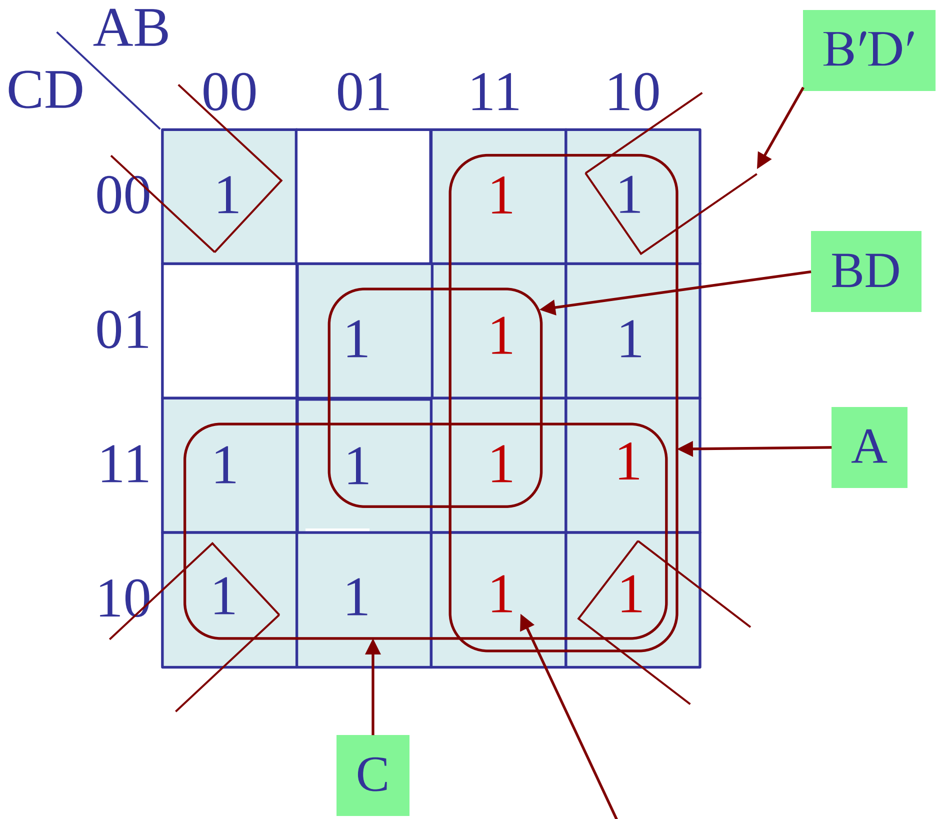
$F = B'D' + BD + A + C$

Karnaugh map examples – 9

```
[A,B,C,D] = a2d(0:15,4);
F = (~B & ~D) | (B & D) | A | C;
[A,B,C,D,F]
```

row	A B C D	F
0	0 0 0 0	1
1	0 0 0 1	0
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	0
5	0 1 0 1	1
6	0 1 1 0	1
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	1
10	1 0 1 0	1
11	1 0 1 1	1
12	1 1 0 0	1
13	1 1 0 1	1
14	1 1 1 0	1
15	1 1 1 1	1

$F = B'D' + BD + A + C$



don't cares are effectively replaced by 1's

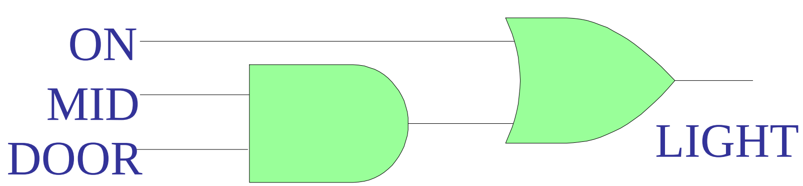
Karnaugh map examples – 10

Car dome light example (from p.122)

The light has a 3-position switch such that the light turns on if the switch is in the ON position or if the middle switch MID is on and the door signal DOOR is also on when any door is open, otherwise the light is off when the switch is in the OFF position.

Translating this description into a Boolean algebraic expression, we have:

$$\text{LIGHT} = \text{ON} + \text{MID} \cdot \text{DOOR}$$



OFF	MID	ON	DOOR	LIGHT
0	0	0	0	x
0	0	0	1	x
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	x
0	1	1	1	x
1	0	0	0	0
1	0	0	1	0
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

x's denote don't care or unrealizable entries because the switch can only be at one of 3 positions

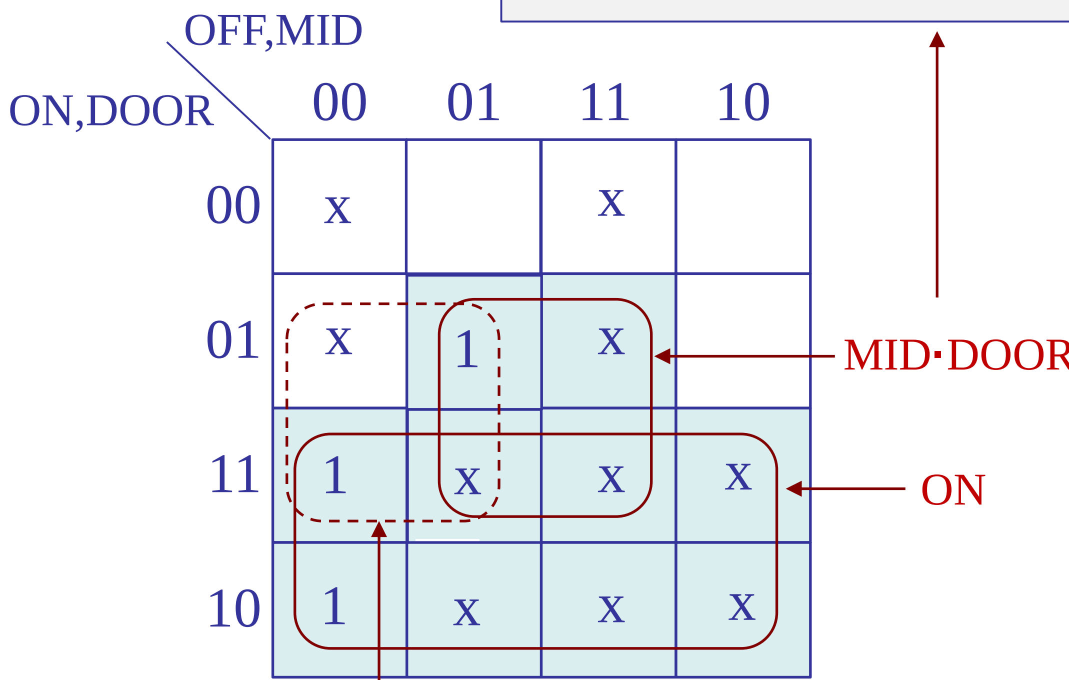
Karnaugh map examples – 10

OFF	MID	ON	DOOR	LIGHT
0	0	0	0	x
0	0	0	1	x
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	x
0	1	1	1	x
1	0	0	0	0
1	0	0	1	0
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

don't cares are treated as1's

not all don't cares were used here

LIGHT = ON + MID·DOOR



alternative grouping leads to

LIGHT = ON + (OFF)′·DOOR

not all don't cares were used here

Karnaugh map examples – 10

OFF	MID	ON	DOOR	LIGHT	L1	L2
0	0	0	0	x	0	0
0	0	0	1	x	0	1
0	0	1	0	1	1	1
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	1	1	1	1
0	1	1	0	x	1	1
0	1	1	1	x	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	x	1	1
1	0	1	1	x	1	1
1	1	0	0	x	0	0
1	1	0	1	x	1	0
1	1	1	0	x	1	1
1	1	1	1	x	1	1

$L1 = ON + MID \cdot DOOR$

$L2 = ON + (OFF)' \cdot DOOR$

```
% MATLAB code:

[off,mid,on,door] = a2d(0:15,4);

L1 = on | (mid & door);
L2 = on | (~off & door);

[off,mid,on,door,L1,L2] % print
```

don't cares are treated as1's

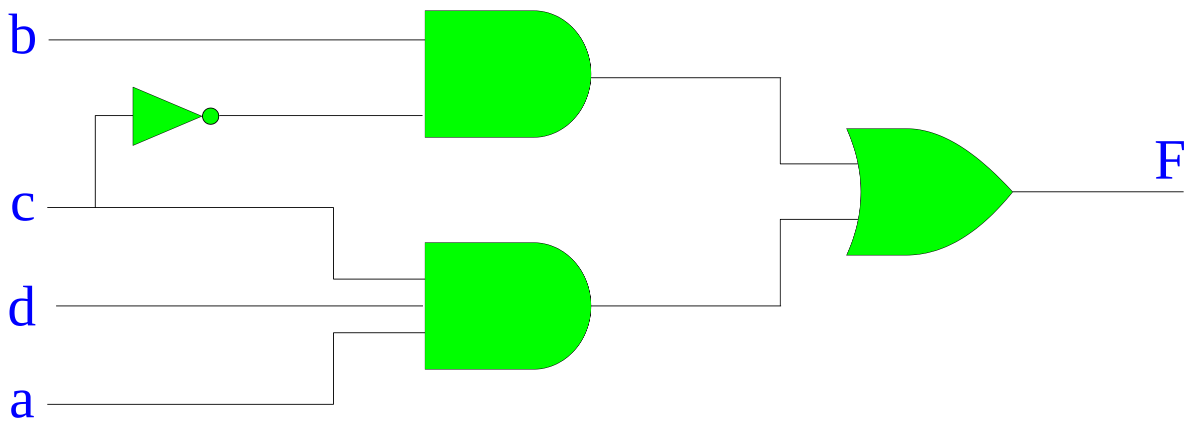
because not all don't cares were used in deriving L1 and L2, the full L1 and L2 columns are different. However, they agree on the relevant/realizable part of the truth table (shown in red)

Karnaugh map examples – 11

lab-2 derivations using K-maps

row	a b c d	F
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	0
3	0 0 1 1	0
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	0
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	0
11	1 0 1 1	1
12	1 1 0 0	1
13	1 1 0 1	1
14	1 1 1 0	0
15	1 1 1 1	1

$F = bc' + acd$

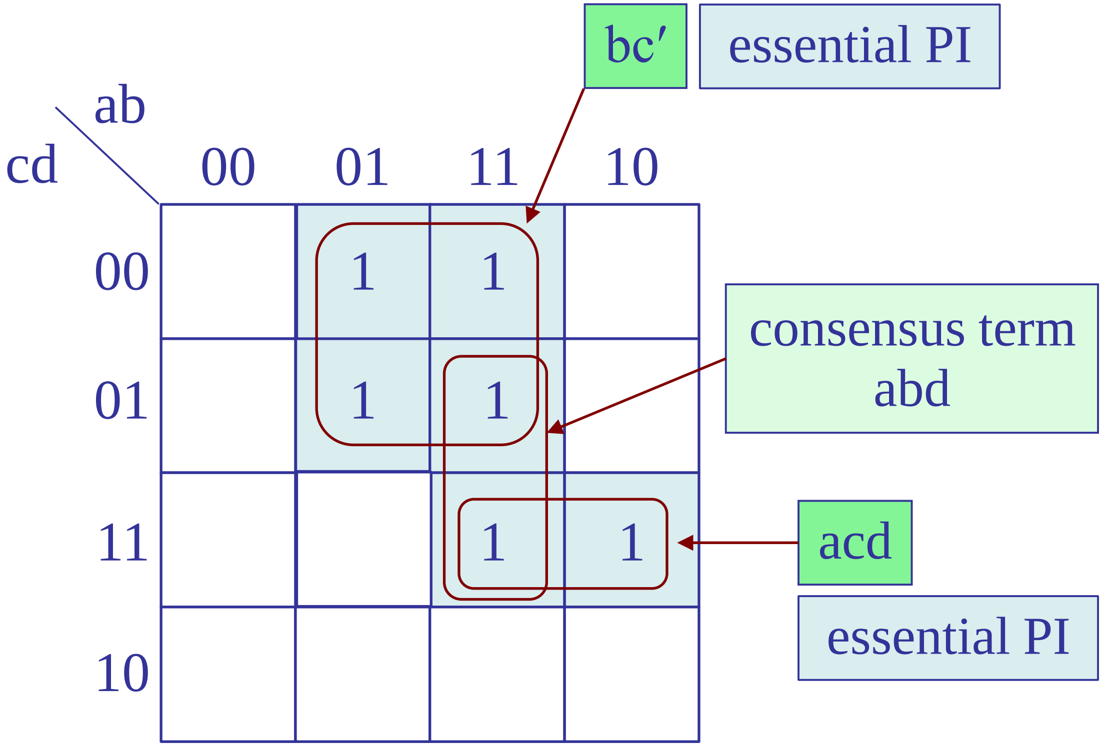



```
[a,b,c,d] = a2d(0:15,4);
F = (b & ~c) | (a & c & d);
[a,b,c,d,F]      % print truth table
```

Karnaugh map examples – 11

lab-2 derivations using K-maps

row	a b c d	F
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	0
3	0 0 1 1	0
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	0
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	0
11	1 0 1 1	1
12	1 1 0 0	1
13	1 1 0 1	1
14	1 1 1 0	0
15	1 1 1 1	1



$F = bc' + acd$

$F = bc' + acd + abd$

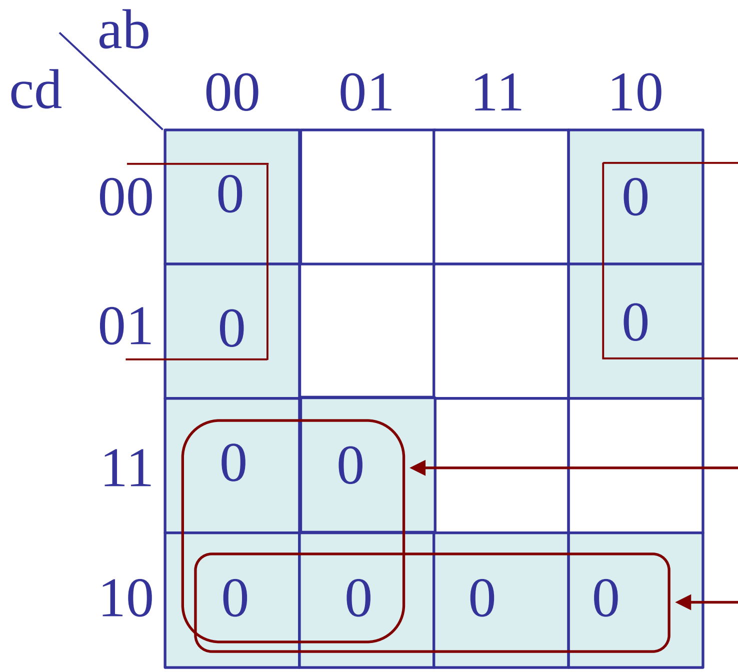
$F = a'bc'd' + a'bc'd + ab'cd + abc'd' + abc'd + abcd$

canonical minterm SOP expansion

Karnaugh map examples – 11

lab-2 derivations using K-maps

row	a b c d	F
0	0 0 0 0	0
1	0 0 0 1	0
2	0 0 1 0	0
3	0 0 1 1	0
4	0 1 0 0	1
5	0 1 0 1	1
6	0 1 1 0	0
7	0 1 1 1	0
8	1 0 0 0	0
9	1 0 0 1	0
10	1 0 1 0	0
11	1 0 1 1	1
12	1 1 0 0	1
13	1 1 0 1	1
14	1 1 1 0	0
15	1 1 1 1	1



if they were minterms for F′

b′c′

all three are essential PIs

a′c

cd′

F′ = a′c + b′c′ + cd′ = sum-of-products for F′

De Morgan

F = (a + c′)(b + c)(c′ + d) = product-of-sums

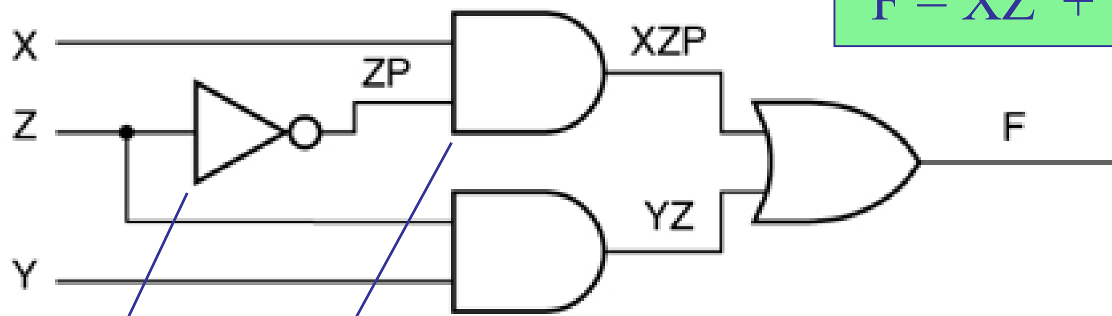
simplified POS expansion

20. Timing hazards – static hazards

effects of propagation delays

$F = XZ' + YZ$

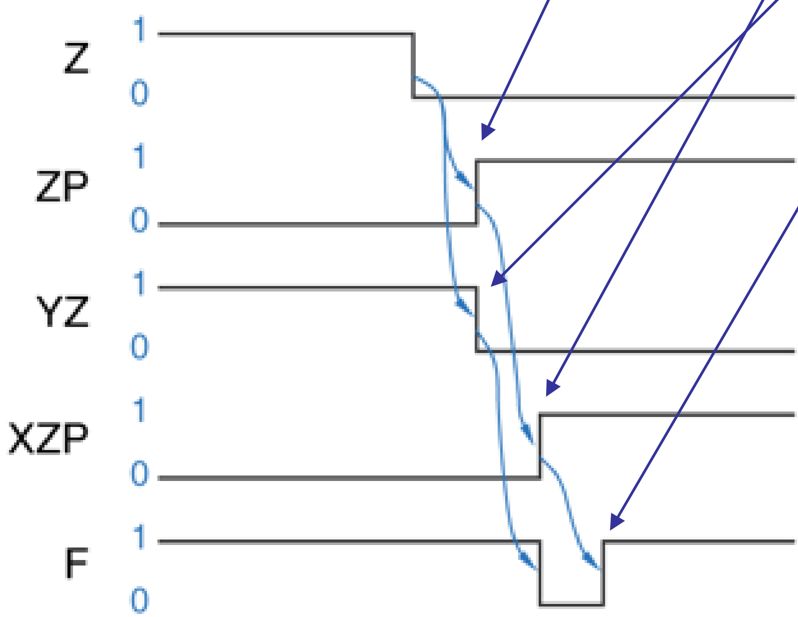
$X = 1$
$Y = 1$



delay through inverter

delays through AND gates

delay through OR gate

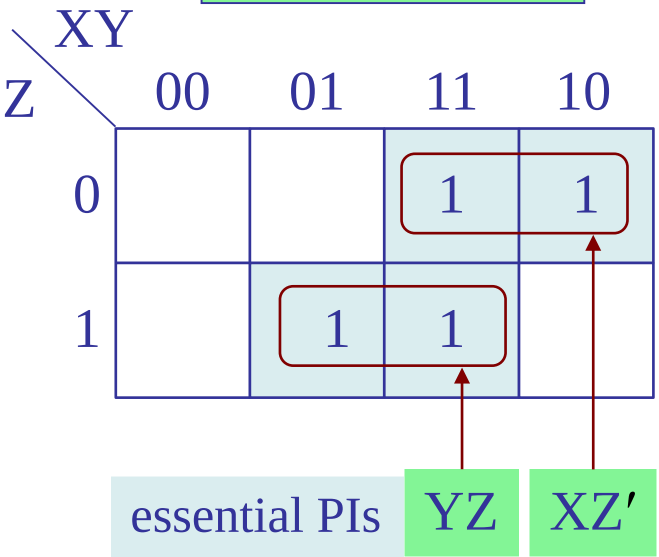


lab 2 and Wakerly, Fig.3-26

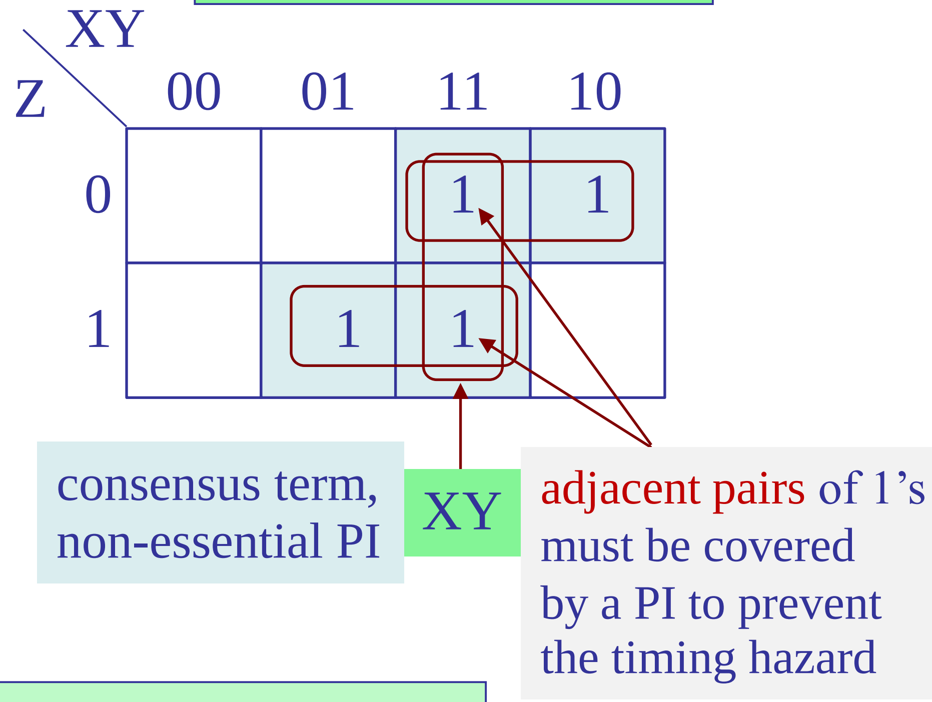
referred to as static-1 hazard,

note: the dual of the above circuit would exhibit a static-0 hazard, see also Wakerly Fig. 3-27

$F = XZ' + YZ$



$F = XZ' + YZ + XY$



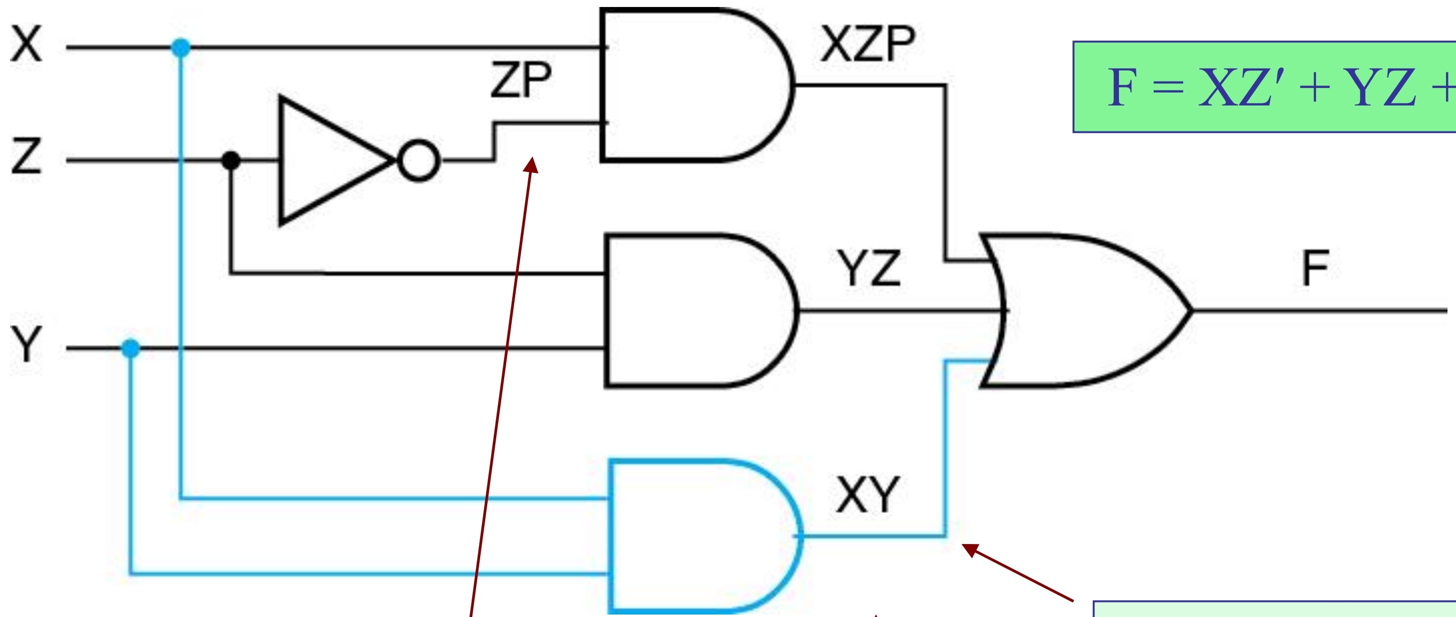
X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

MATLAB code for generating truth table

```
[X,Y,Z] = a2d(0:7,3);   % 3-bit binary pattern

F = (X & ~Z)|(Y & Z);

[X,Y,Z,F]               % print truth table
```

$$\mathbf{X \cdot A + X' \cdot B = X \cdot A + X' \cdot B + A \cdot B} \qquad \textbf{(consensus)}$$

20. Timing hazards – static hazards



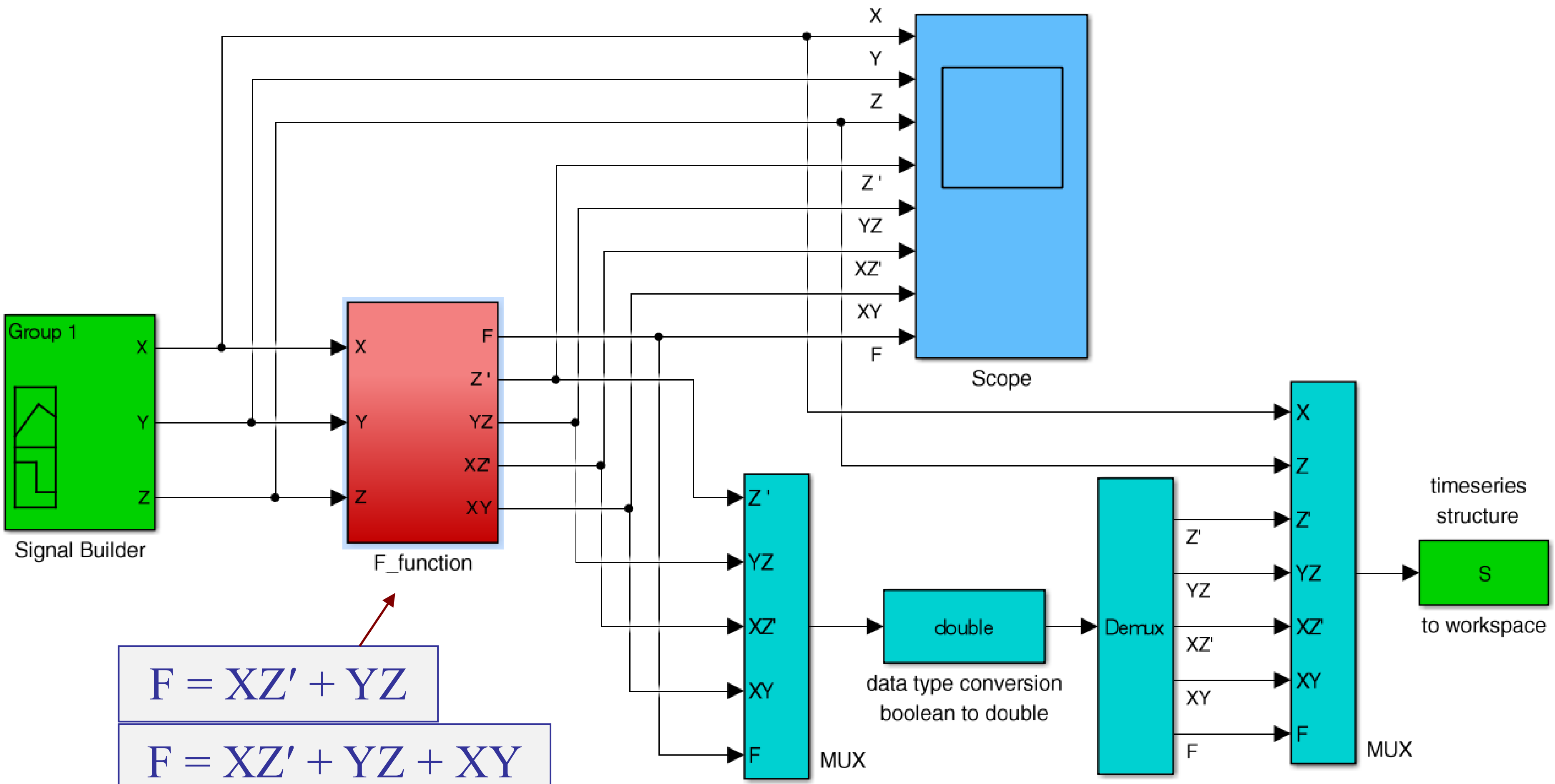
$F = XZ' + YZ + XY$

consensus term

look at these individual outputs

Wakerly, Fig.3-29

Simulink Implementation



$$F = XZ' + YZ$$

$$F = XZ' + YZ + XY$$

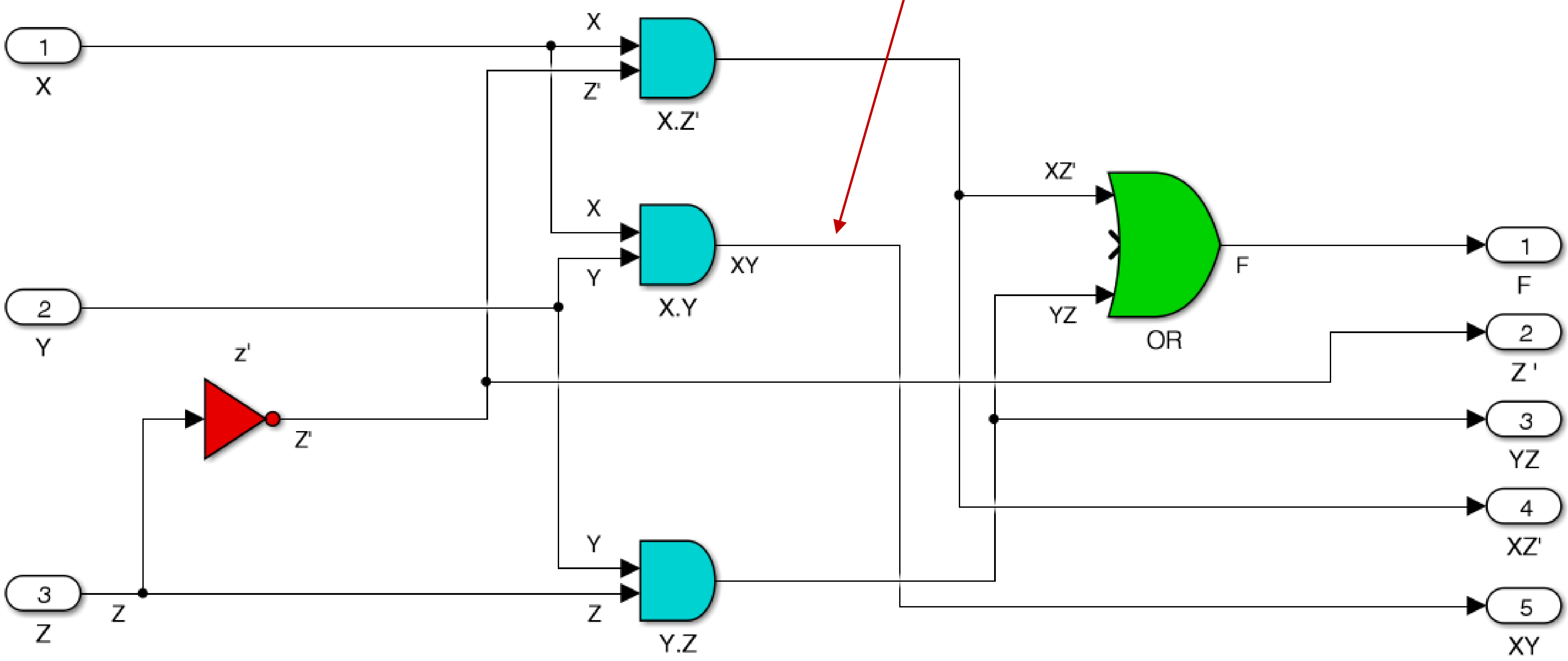
Simulink file: fig326a.slx

without delays

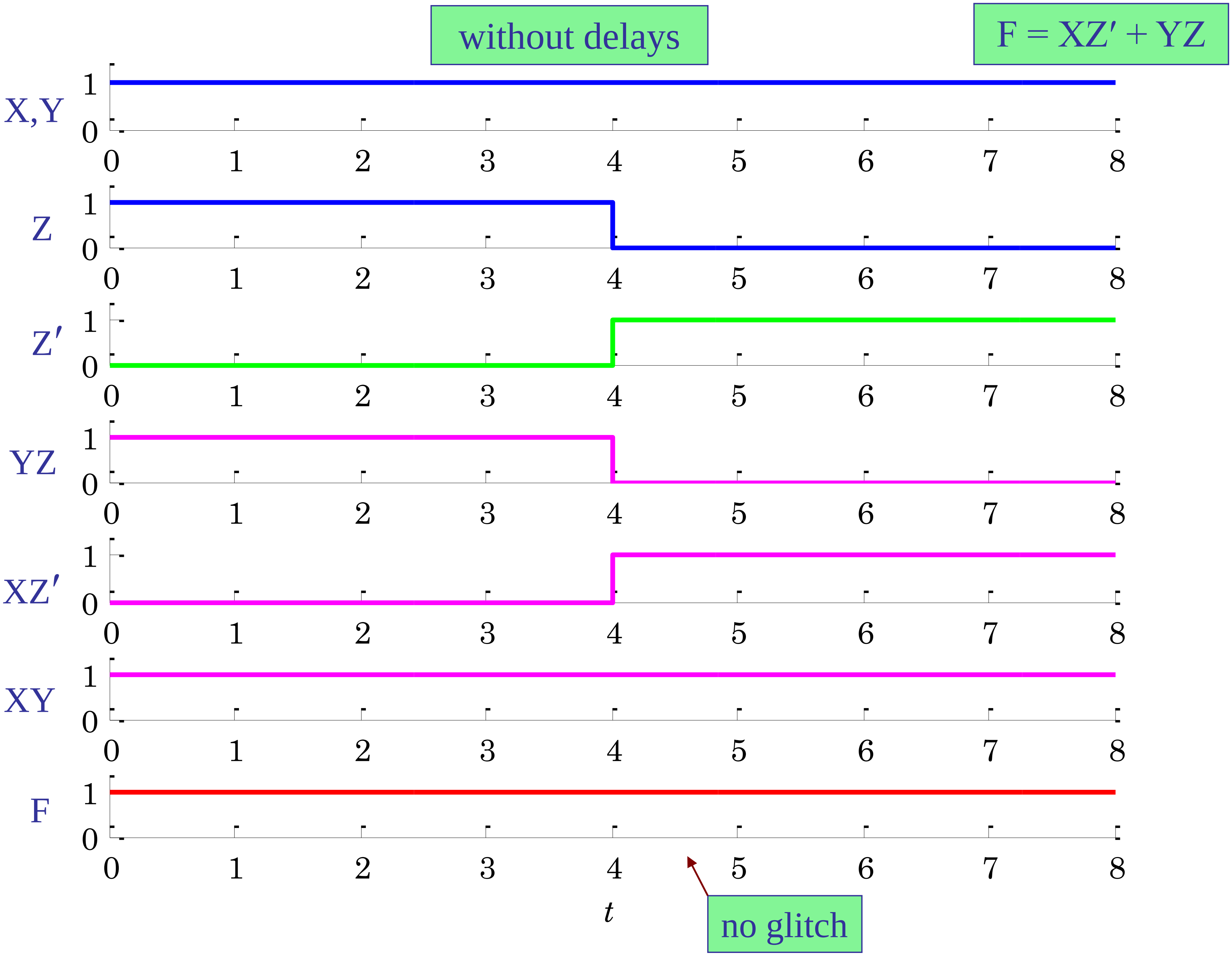
$$F = XZ' + YZ$$

$$F = XZ' + YZ + XY$$

consensus term



Simulink: subfunction contained in fig326a.slx



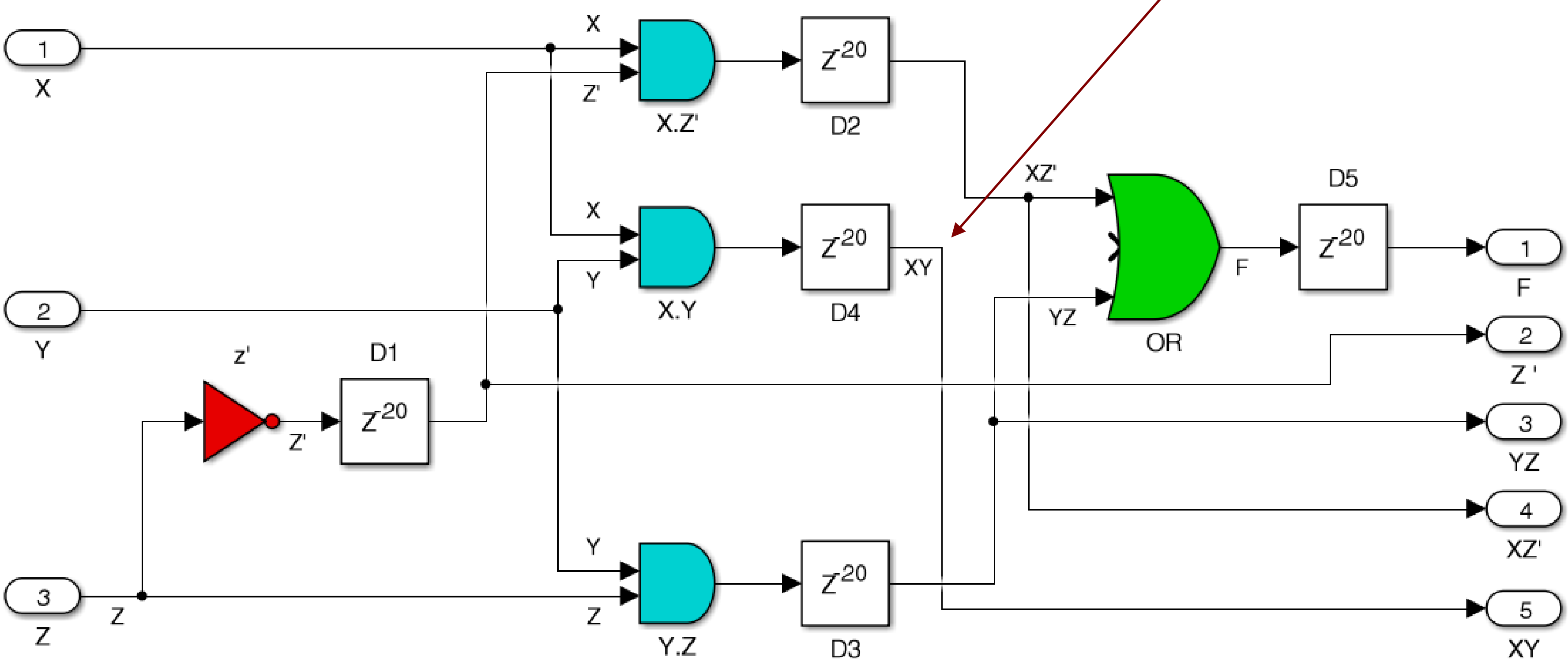
without delays
F = XZ′ + YZ
X,Y
Z
Z′
YZ
XZ′
XY
F
t
no glitch

with delays

$F = XZ' + YZ$

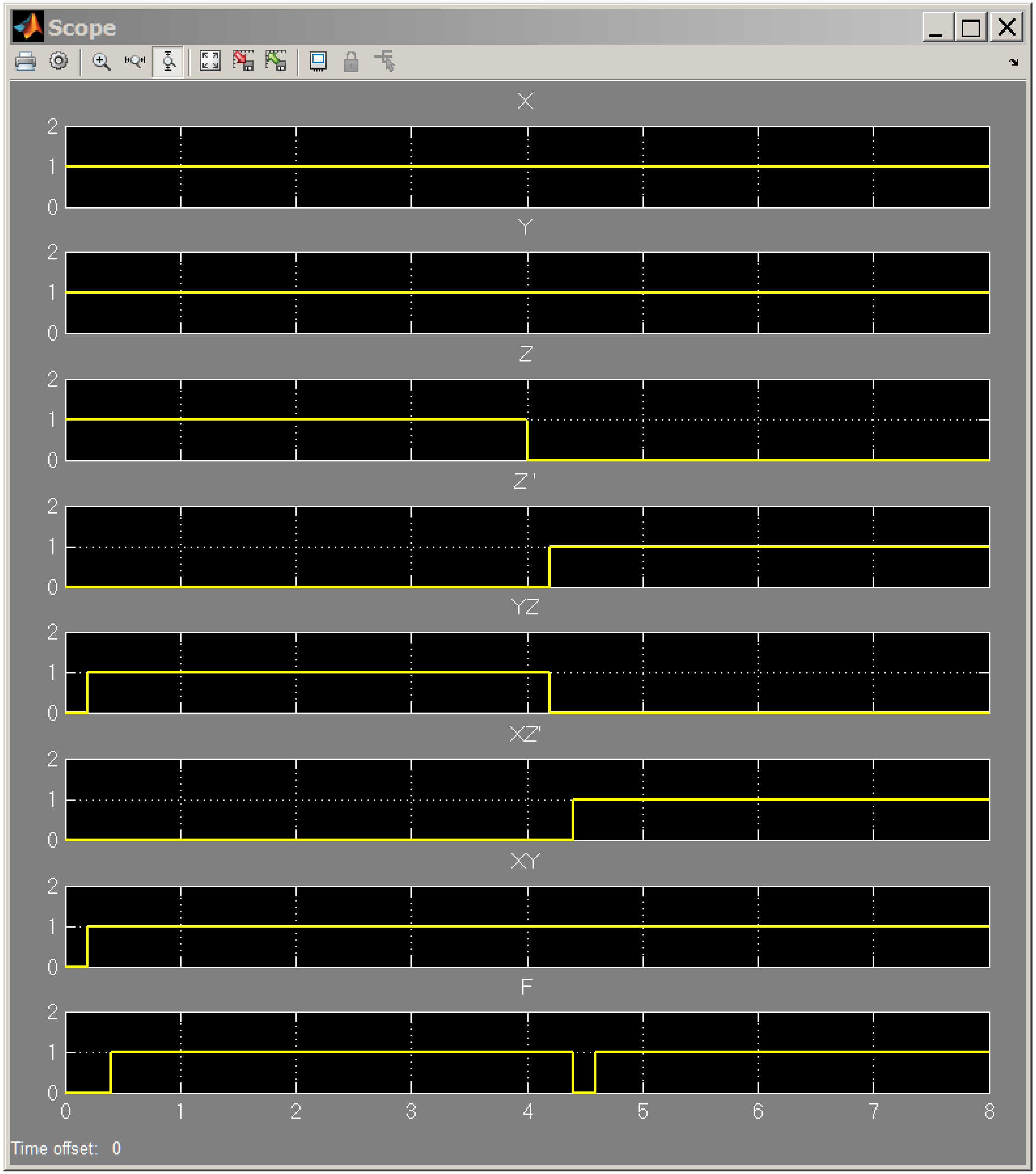
subfunction contained in fig326a.slx

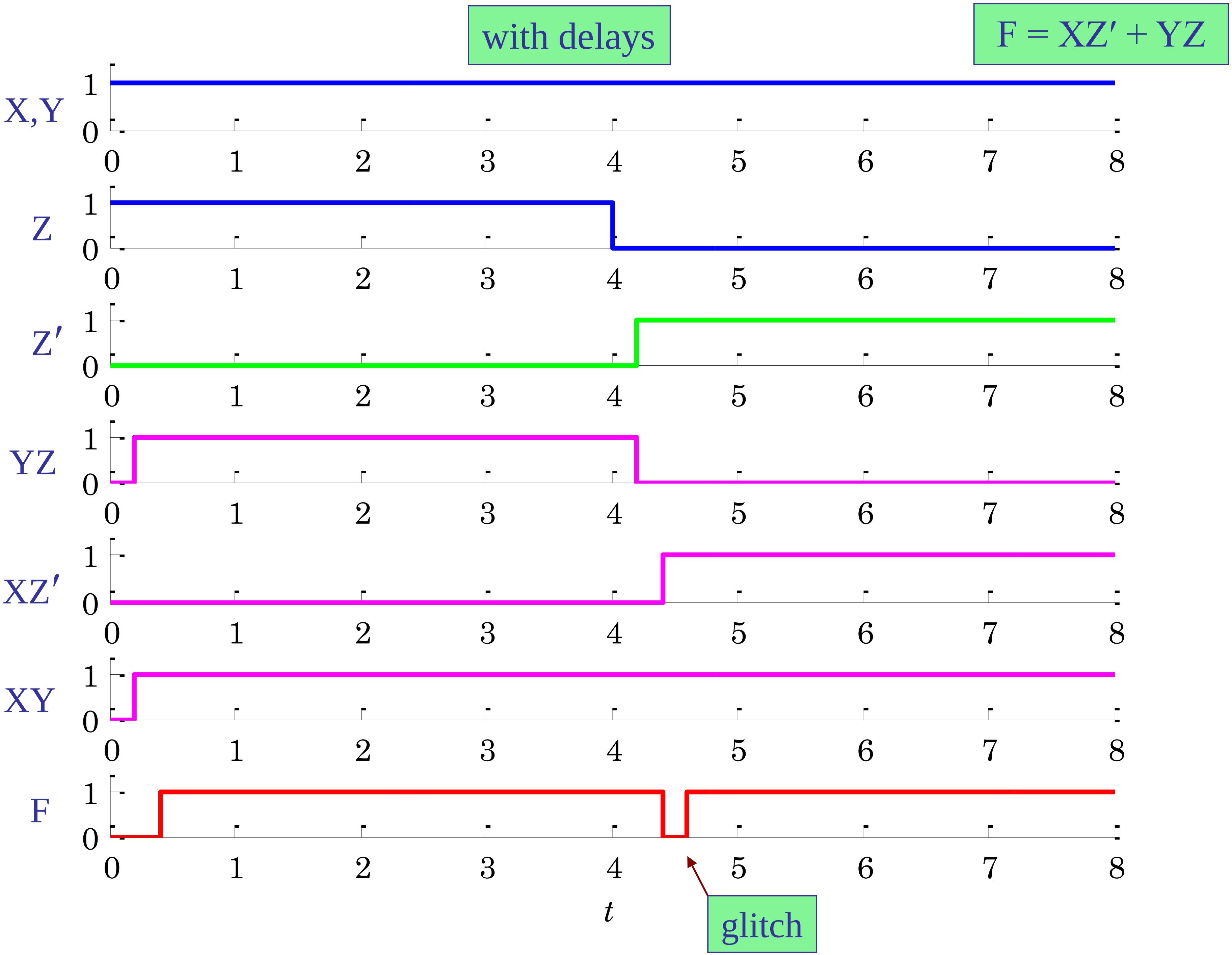
with consensus term XY not connected



note: here, the simulation time is $0 < t < 8$ (sec or any other time unit), and the sampling time interval is, by default, $T_s = 0.01$ time units, so that there are 100 samples per time unit, thus, delay by 20 samples would correspond to $20/100 = 0.2 = 1/5$ of a time unit.

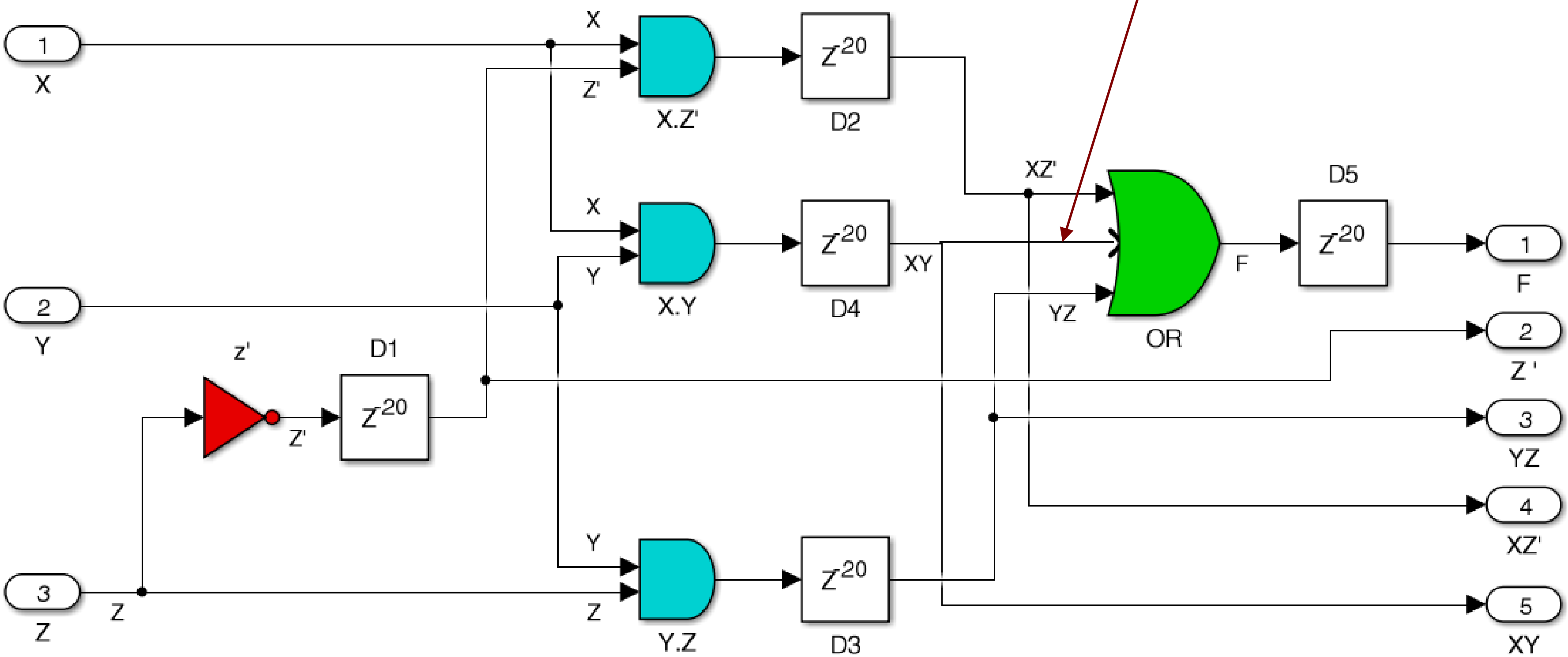
with delays



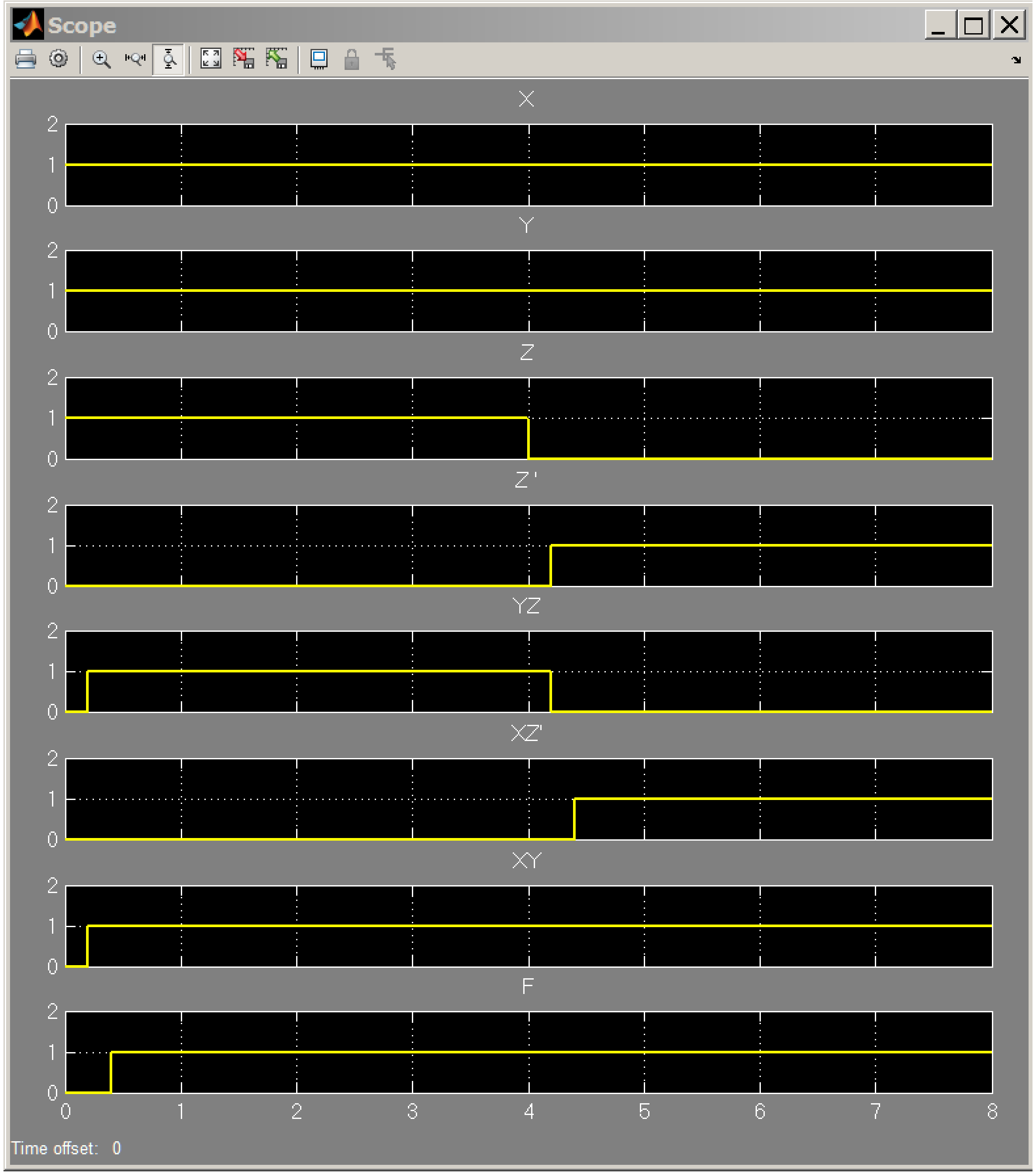

with delays
F = XZ′ + YZ
X,Y
Z
Z′
YZ
XZ′
XY
F
t
glitch

```matlab
% fig326m.m - found on Canvas
% import data from Simulink into MATLAB for plotting

t   = S.time;          % time range, length 801 by default
X   = S.data(:,1);
Z   = S.data(:,2);
Zp  = S.data(:,3);
YZ  = S.data(:,4);
XZp = S.data(:,5);
XY  = S.data(:,6);
F   = S.data(:,7);     % extract computed output

figure;
   subplot(7,1,1); stairs(t,X,'b-'); yaxis(0,2,0:2);
   subplot(7,1,2); stairs(t,Z,'b-'); yaxis(0,2,0:2);
   subplot(7,1,3); stairs(t,Zp,'g-'); yaxis(0,2,0:2);
   subplot(7,1,4); stairs(t,YZ,'m-'); yaxis(0,2,0:2);
   subplot(7,1,5); stairs(t,XZp,'m-'); yaxis(0,2,0:2);
   subplot(7,1,6); stairs(t,XY,'m-'); yaxis(0,2,0:2);
   subplot(7,1,7); stairs(t,F,'r-'); yaxis(0,2,0:2);
   xlabel('\itt');
```

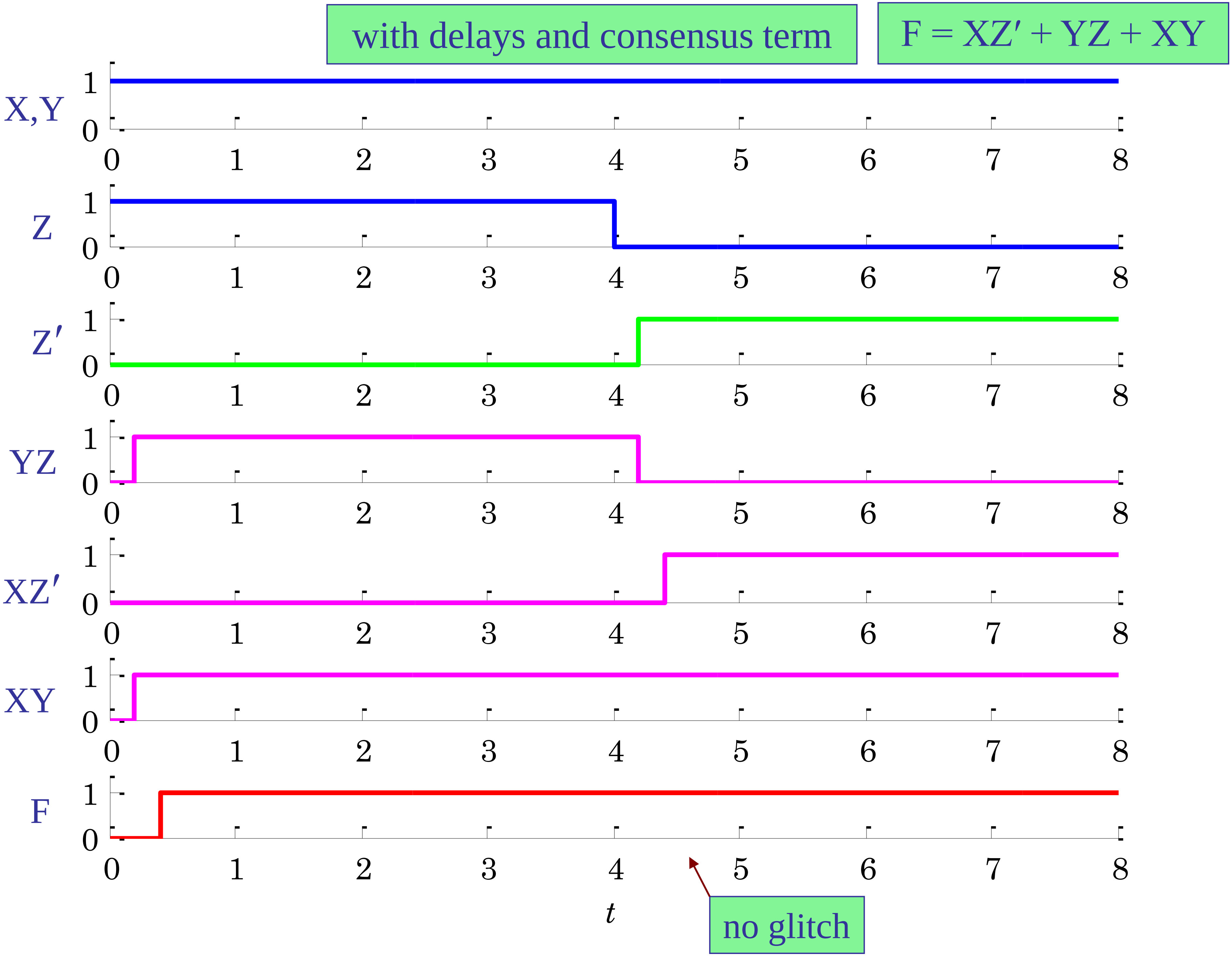
with consensus term XY connected



$$F = XZ' + YZ + XY$$



Scope
X
Y
Z
Z'
YZ
XZ'
XY
F
0
1
2
3
4
5
6
7
8
Time offset: 0



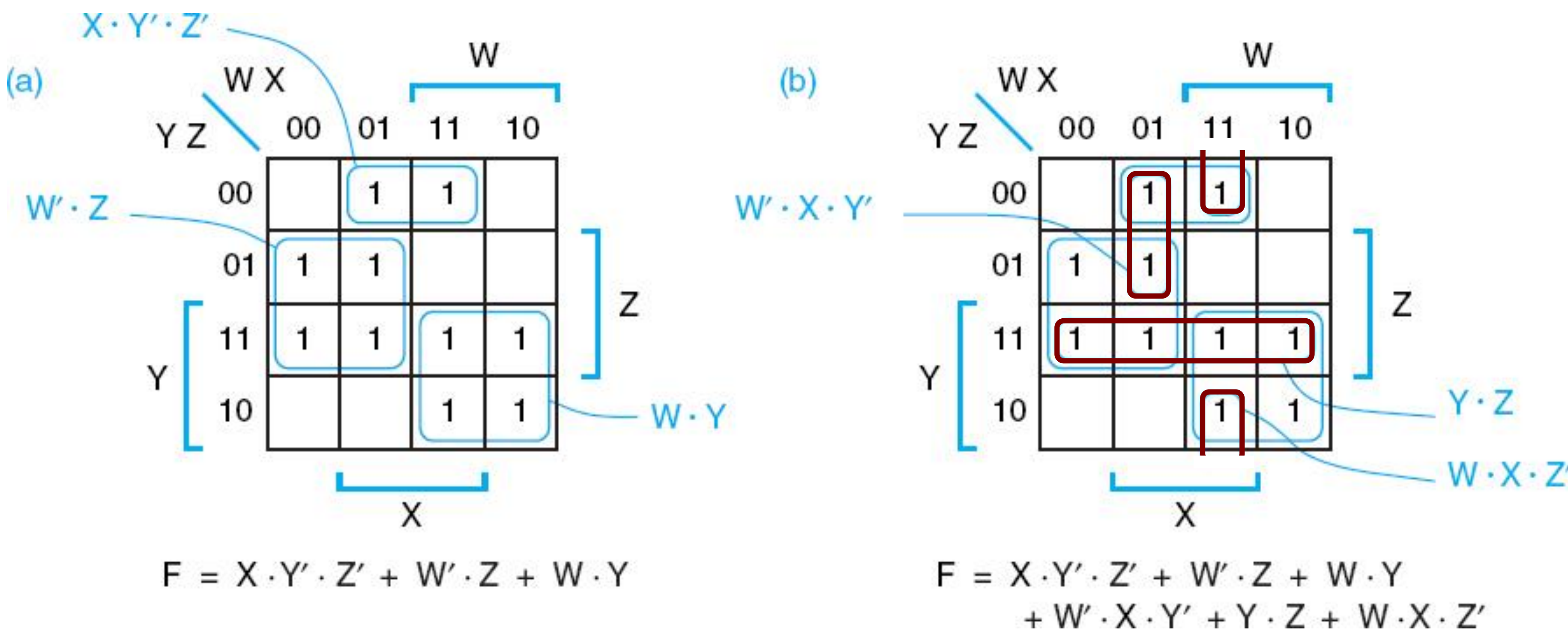
with delays and consensus term
F = XZ′ + YZ + XY
X,Y
Z
Z′
YZ
XZ′
XY
F
1
0
0 1 2 3 4 5 6 7 8
t
no glitch

another example – Wakerly Fig. 3-30

K-map for a sum-of-products circuit

(a) as originally designed

(b) with extra product terms fo cover static-1 hazards



adjacent pairs of 1’s must be covered by a PI to prevent timing hazards two more examples [cf. Brown & Vranesic]

adjacent pairs of 1's must be covered by a PI to prevent timing hazards

cd \ ab	00	01	11	10
00			1	
01			1	
11	1	1	1	
10	1	1	1	

$F = a'c + ab$

$\quad = a'c + ab + bc$

consensus term

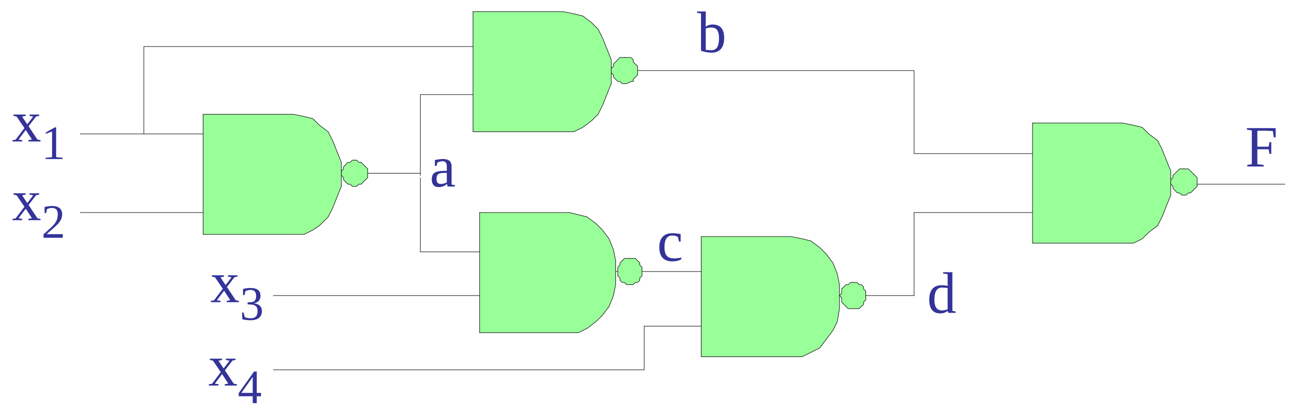
cd \ ab	00	01	11	10
00				
01			1	1
11	1	1	1	1
10	1	1		

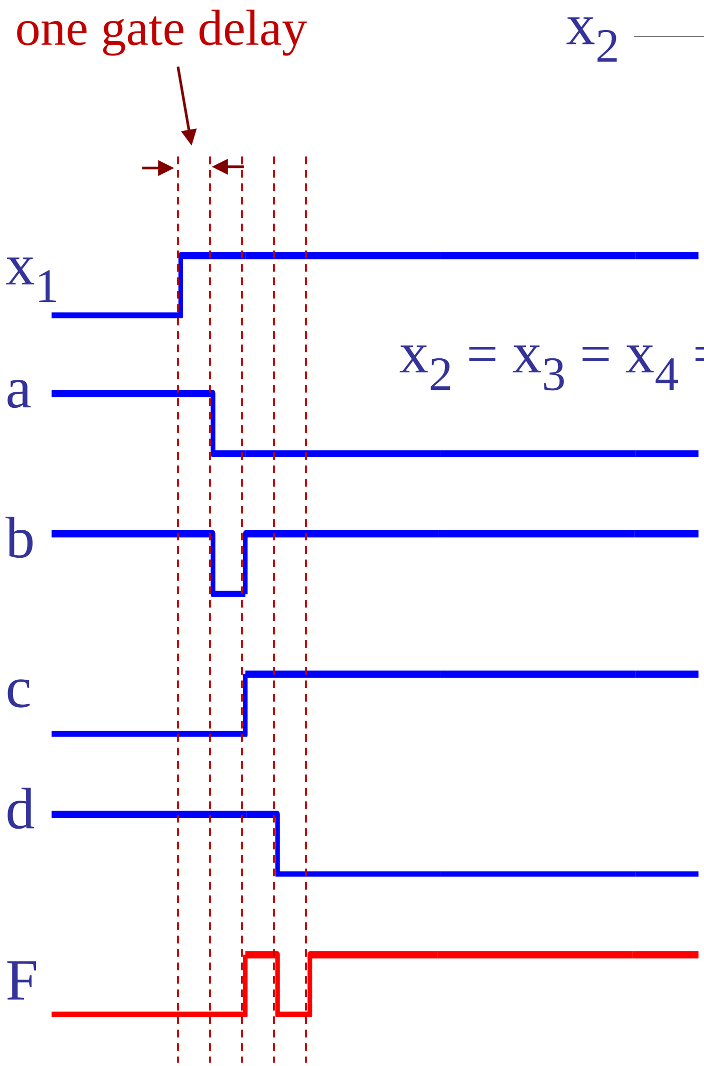
$F = a'c + ad$

$\quad = a'c + ad + cd$

consensus term

20. Timing hazards – dynamic hazards



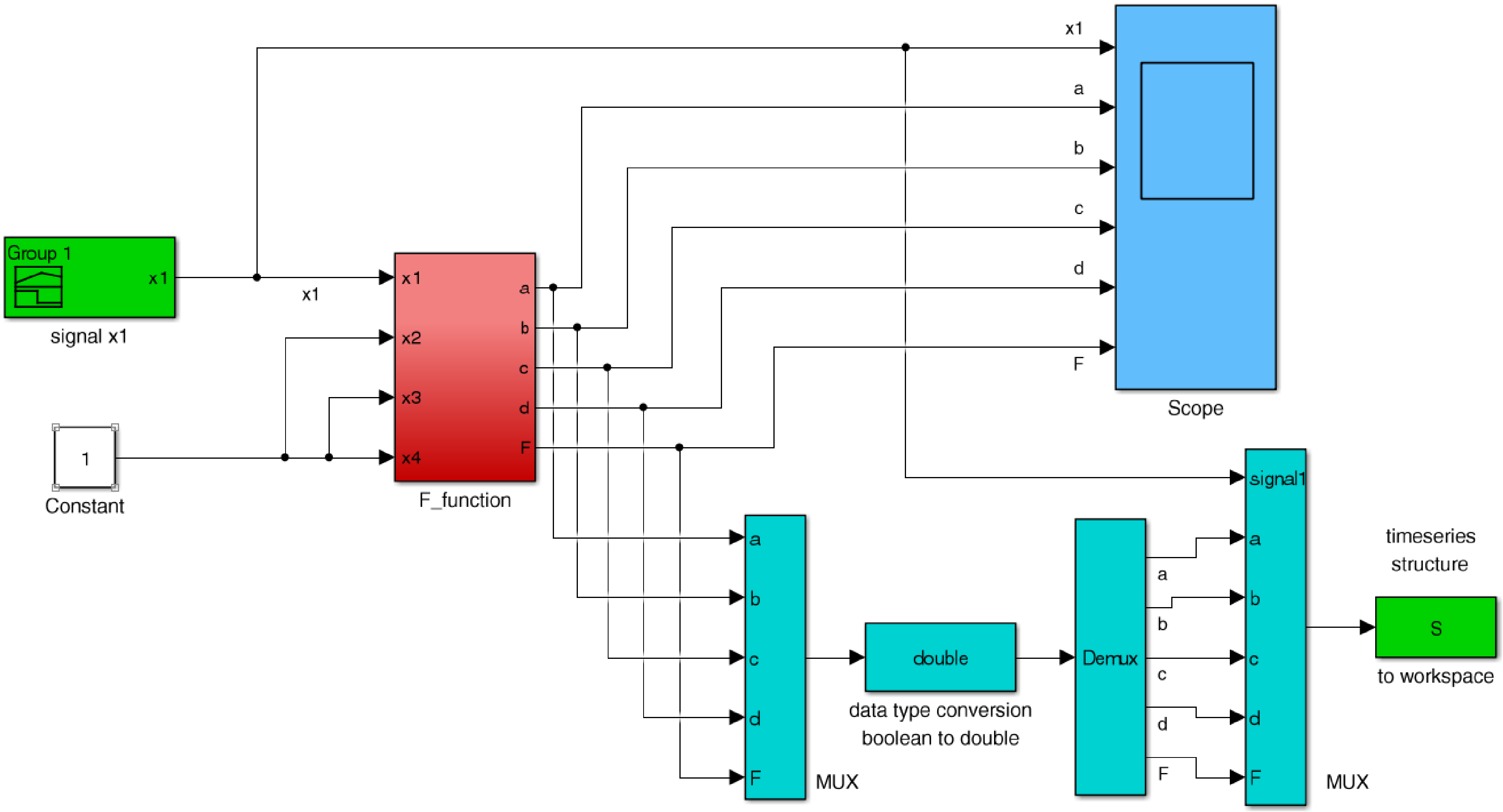


$x_2 = x_3 = x_4 = 1$

dynamic hazard example in which the output undergoes a brief oscillation before settling to its correct value, caused by the delays along different propagation paths.

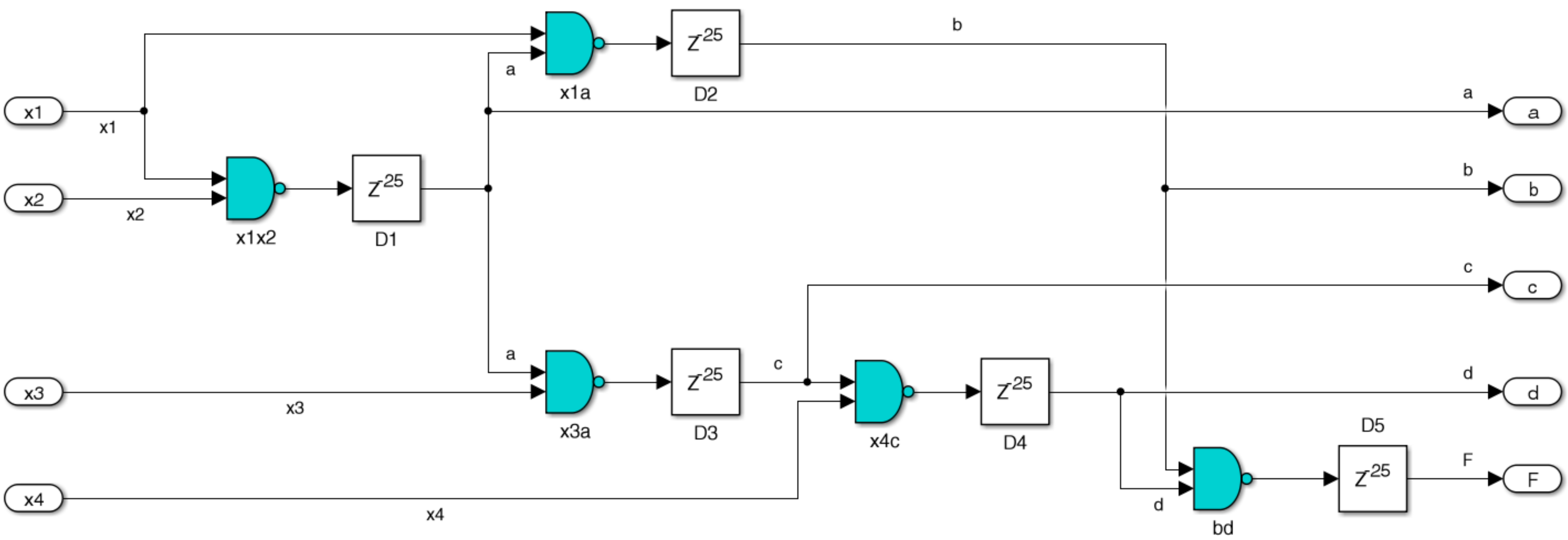
[References: S. Brown & Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3/e, McGraw-Hill, 2014, also ,Wakerly, Sect. 3.4.3.]

Simulink Implementation



Simulink file: dhazard1.slx, with delays

Simulink file: dhazard2.slx, without delays



$$x_2 = x_3 = x_4 = 1$$

$$a = (x_1 \cdot x_2)' = x_1'$$

$$b = (x_1 \cdot a)' = (x_1 \cdot a)'$$

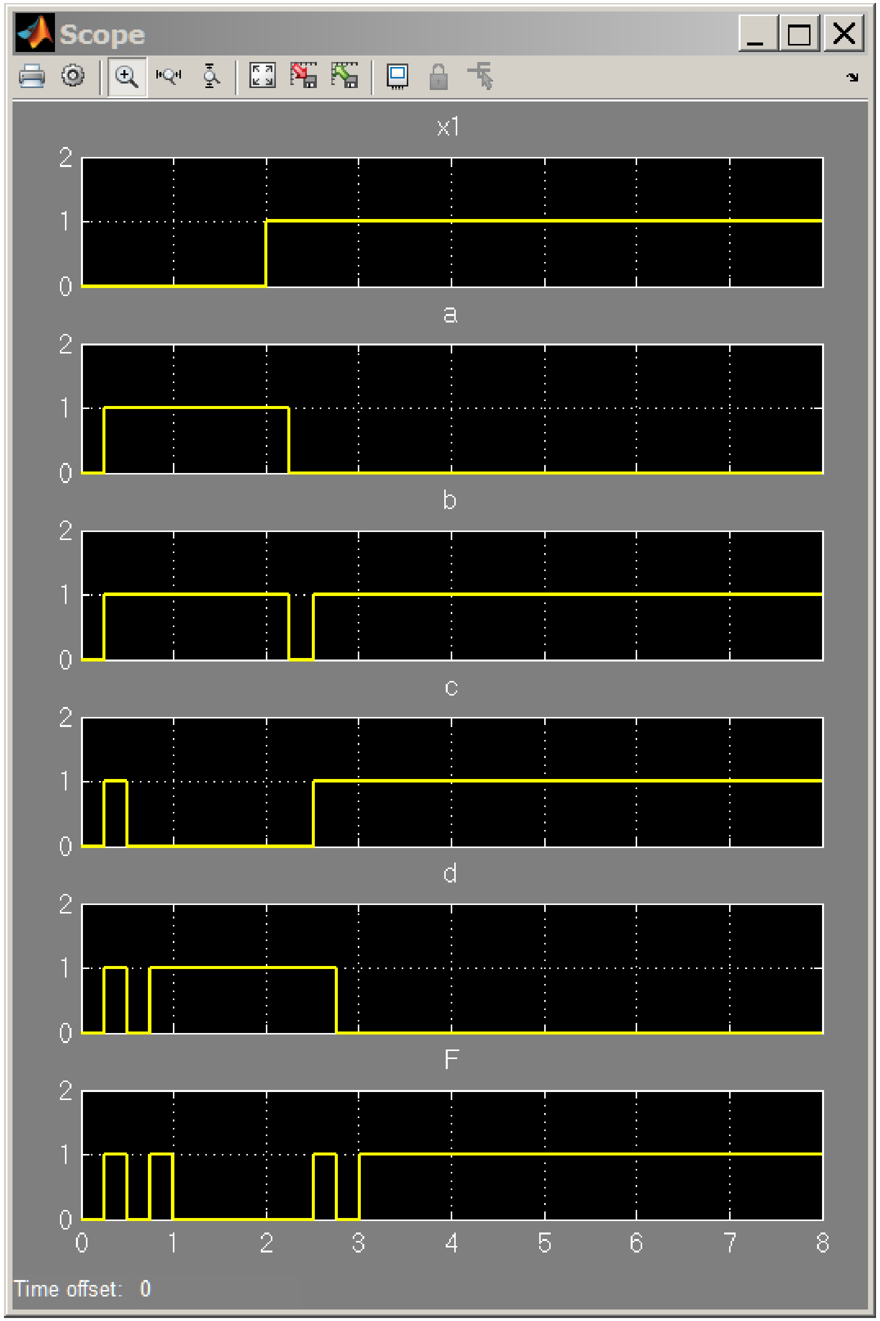
$$c = (x_3 \cdot a)' = (1 \cdot a)' = a'$$

$$d = (x_4 \cdot c)' = (1 \cdot c)' = c' = a$$

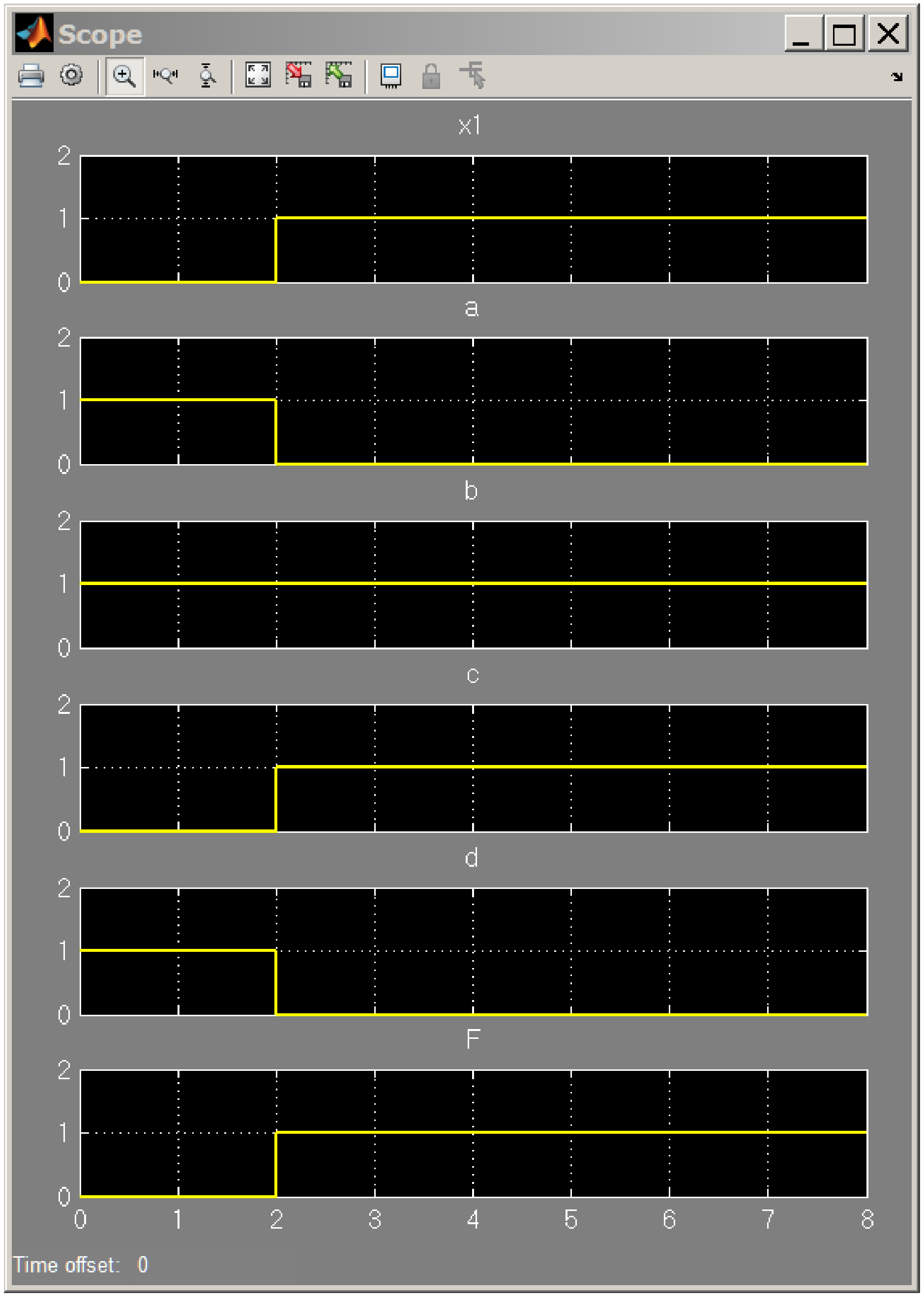
$$F = (b \cdot d)'$$

(without delays)

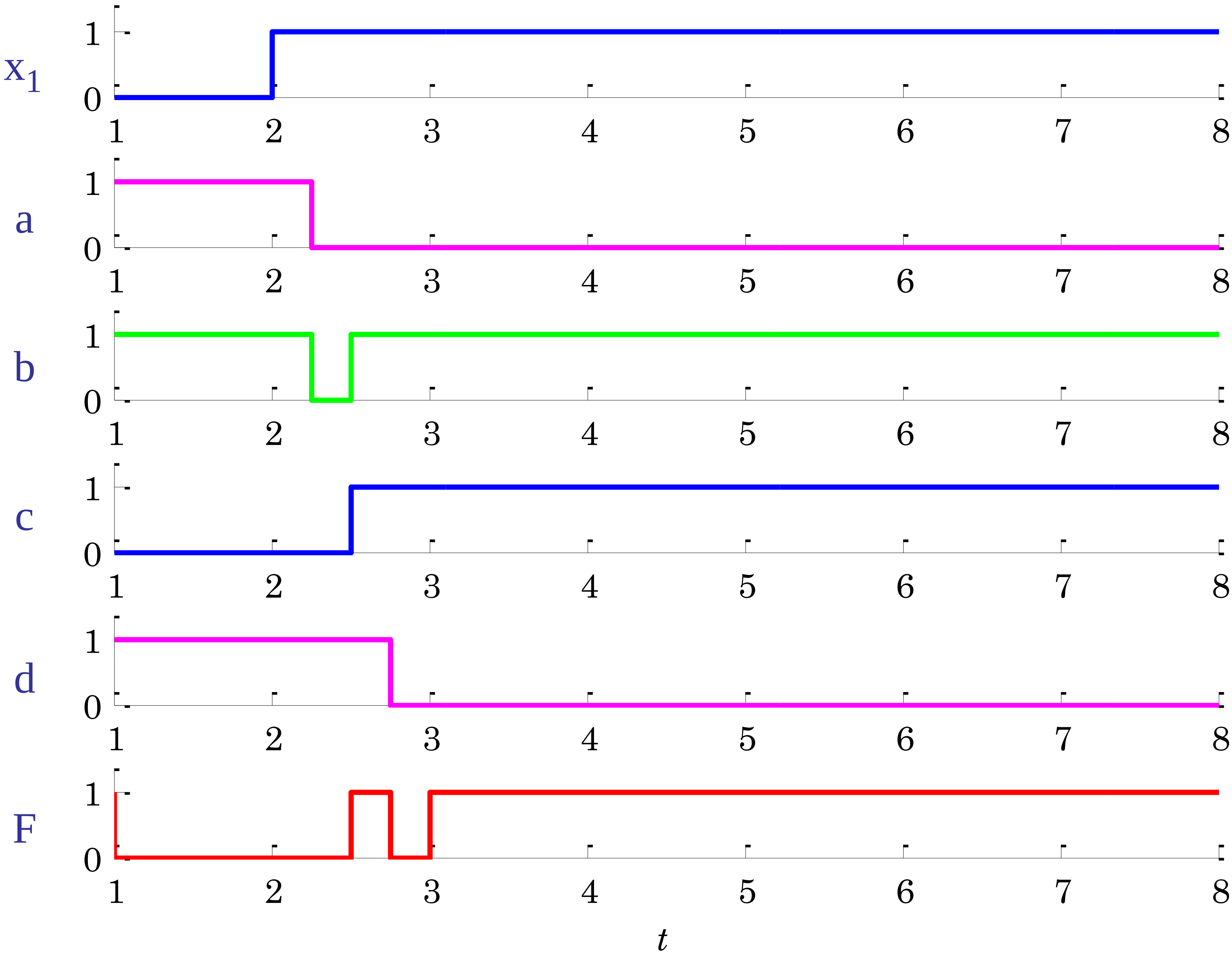
with delays



without delays



```
set(0,'DefaultAxesFontSize',8);

t = S.time;          % extract data from timeseries S
x1 = S.data(:,1);
a = S.data(:,2);
b = S.data(:,3);
c = S.data(:,4);
d = S.data(:,5);
F = S.data(:,6);

figure;
subplot(6,1,1); stairs(t,x1,'b-'); yaxis(0,1.5,0:1);
subplot(6,1,2); stairs(t,a,'m-'); yaxis(0,1.5,0:1);
subplot(6,1,3); stairs(t,b,'g-'); yaxis(0,1.5,0:1);
subplot(6,1,4); stairs(t,c,'b-'); yaxis(0,1.5,0:1);
subplot(6,1,5); stairs(t,d,'m-'); yaxis(0,1.5,0:1);
subplot(6,1,6); stairs(t,F,'r-'); yaxis(0,1.5,0:1);
xlabel('\itt');
```


x1
a
b
c
d
F
0
1
1
2
3
4
5
6
7
8
t

ideal timing diagram – truth table

```
% MATLAB code for generating truth table

[x1,x2,x3,x4] = a2d(0:15,4);     % 4-bit binary pattern

a = ~(x1 & x2);
b = ~(x1 & a);
c = ~(x3 & a);
d = ~(x4 & c);

F = ~(b & d);

[x1,x2,x3,x4,a,b,c,d,F]          % print truth table
```

the operations **&** and | are vectorized

without delays:

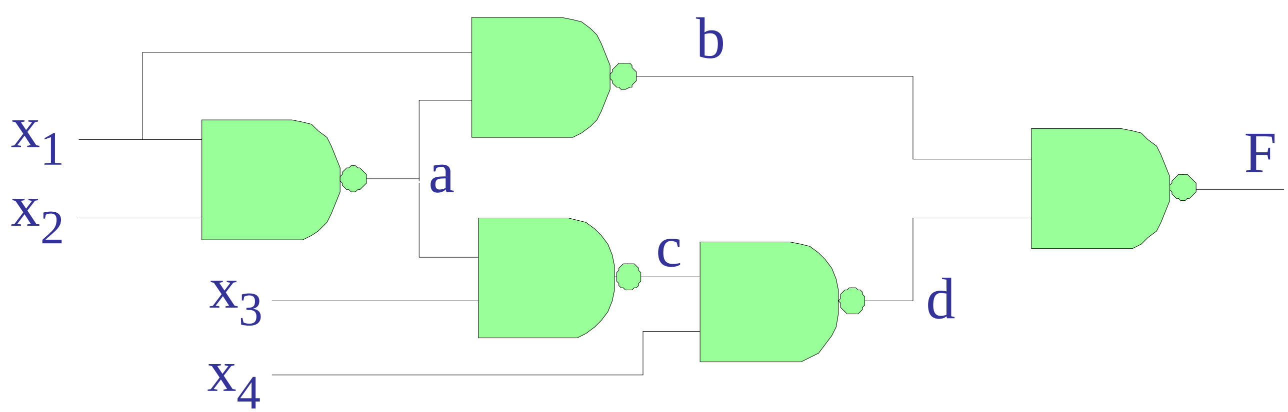
$a = (x_1 \cdot x_2)'$

$b = (x_1 \cdot a)'$

$c = (x_3 \cdot a)'$

$d = (x_4 \cdot c)'$

$F = (b \cdot d)'$

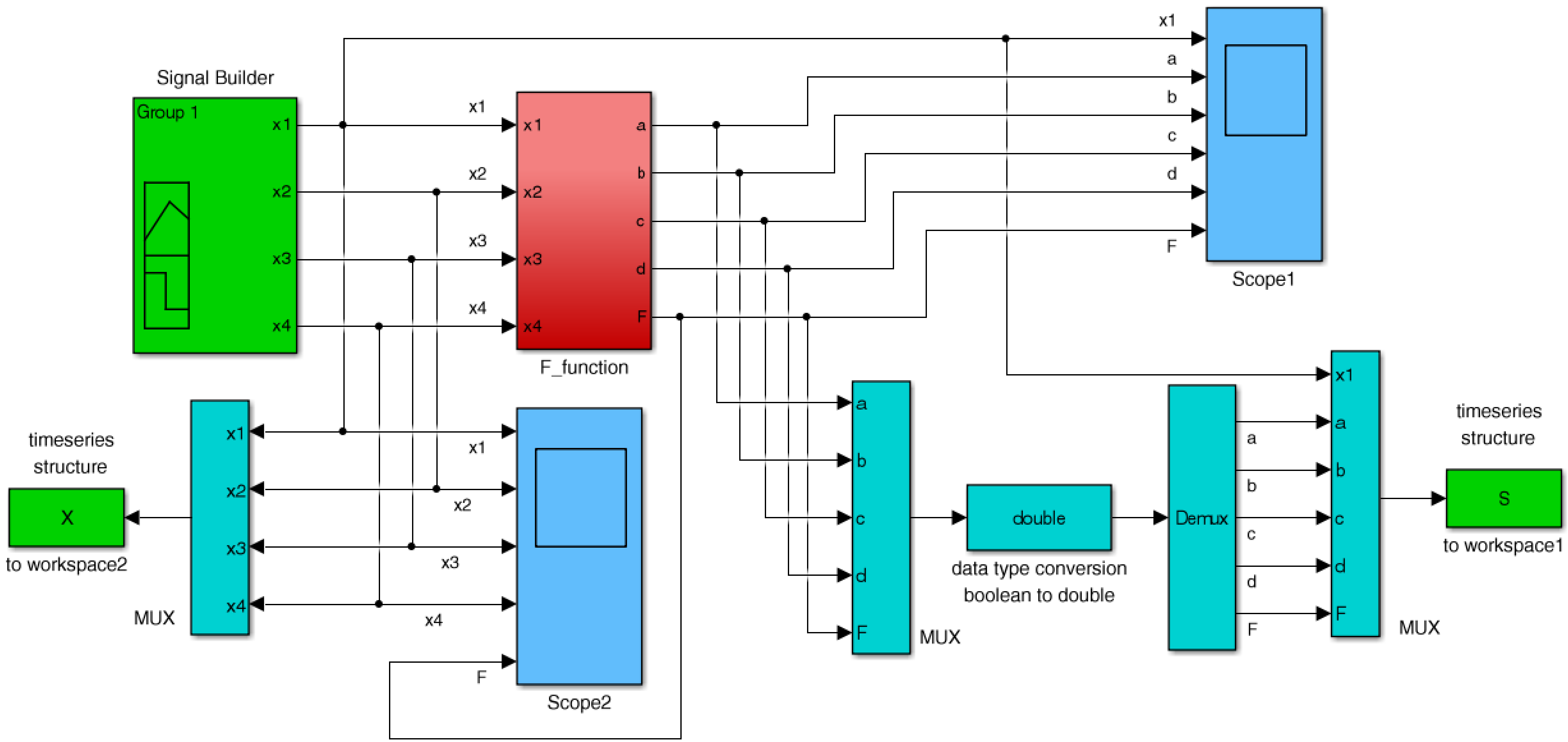


ideal timing diagram – truth table

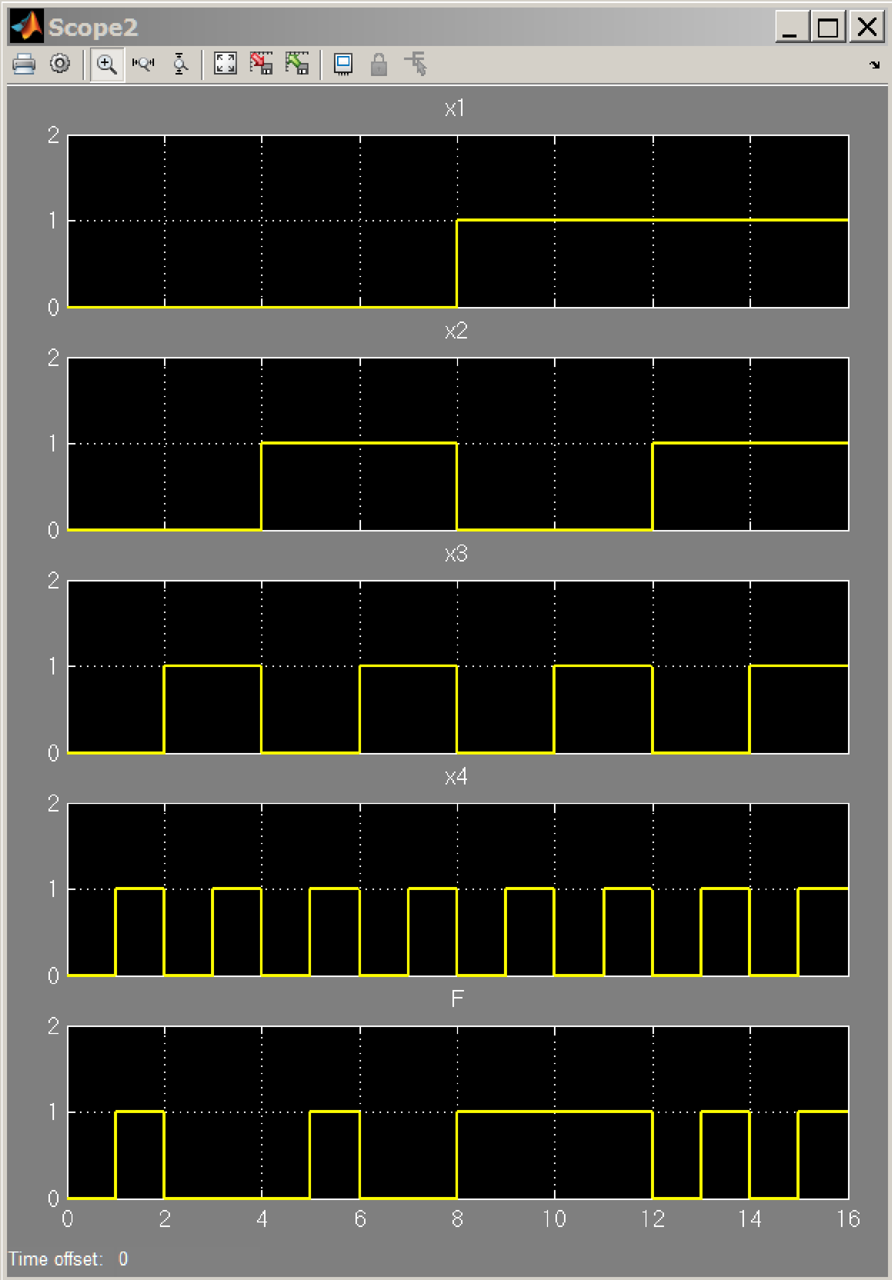
x1	x2	x3	x4	a	b	c	d	F
0	0	0	0	1	1	1	1	0
0	0	0	1	1	1	1	0	1
0	0	1	0	1	1	0	1	0
0	0	1	1	1	1	0	1	0
0	1	0	0	1	1	1	1	0
0	1	0	1	1	1	1	0	1
0	1	1	0	1	1	0	1	0
0	1	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1	1
1	0	0	1	1	0	1	0	1
1	0	1	0	1	0	0	1	1
1	0	1	1	1	0	0	1	1
1	1	0	0	0	1	1	1	0
1	1	0	1	0	1	1	0	1
1	1	1	0	0	1	1	1	0
1	1	1	1	0	1	1	0	1

4-bit binary pattern

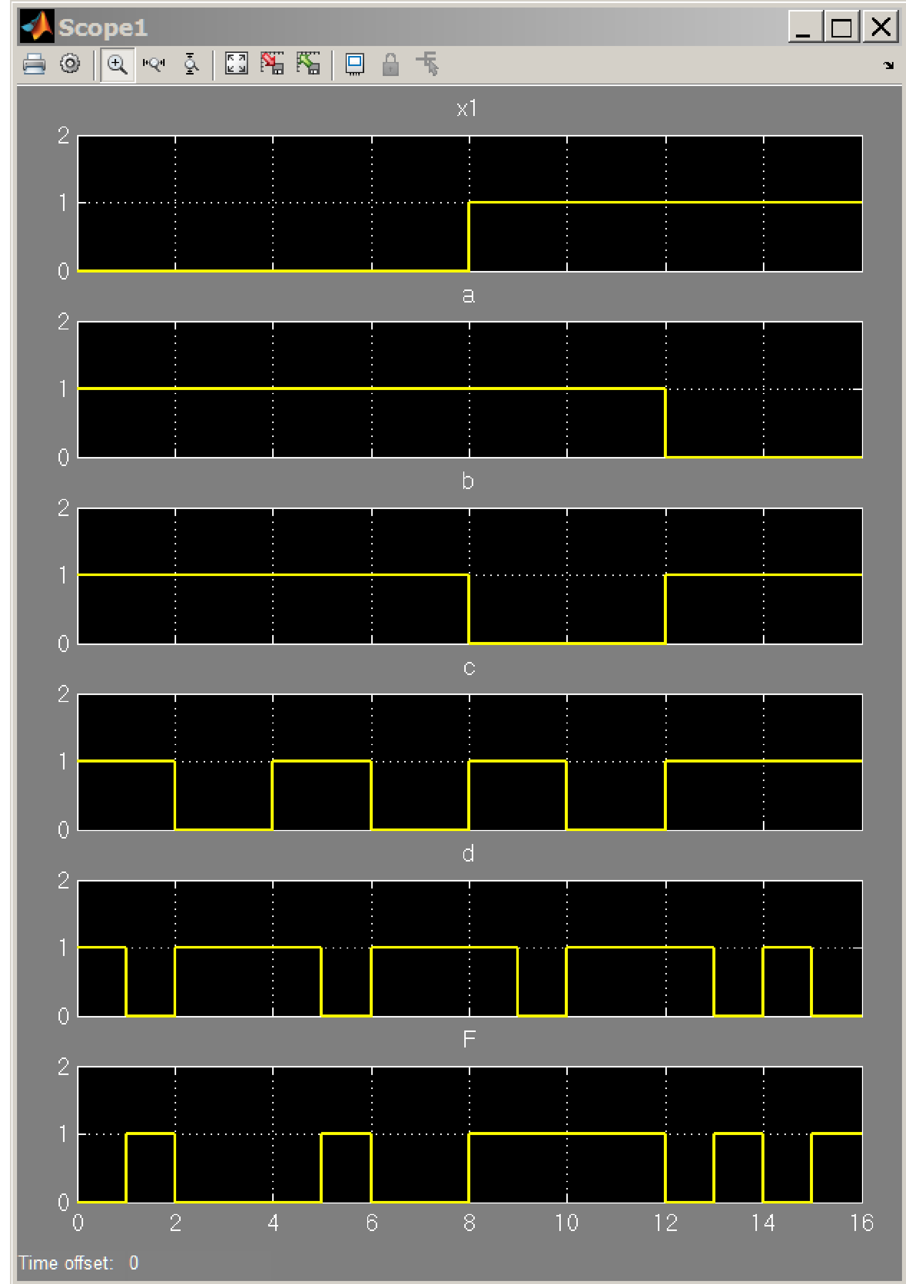
ideal timing diagram – truth table



Simulink file: dhazard3.slx, without delays

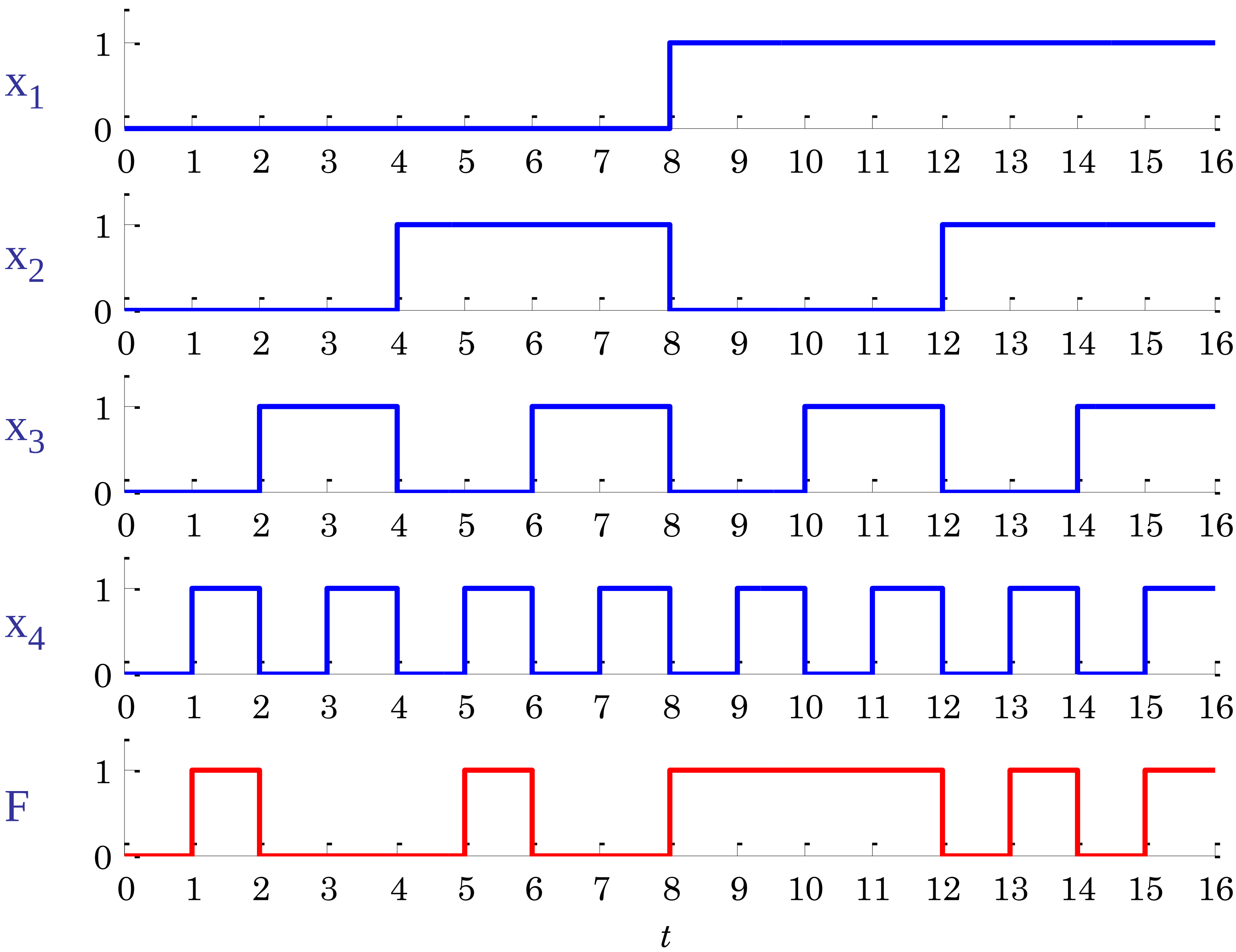


x_1, x_2, x_3, x_4, F signals

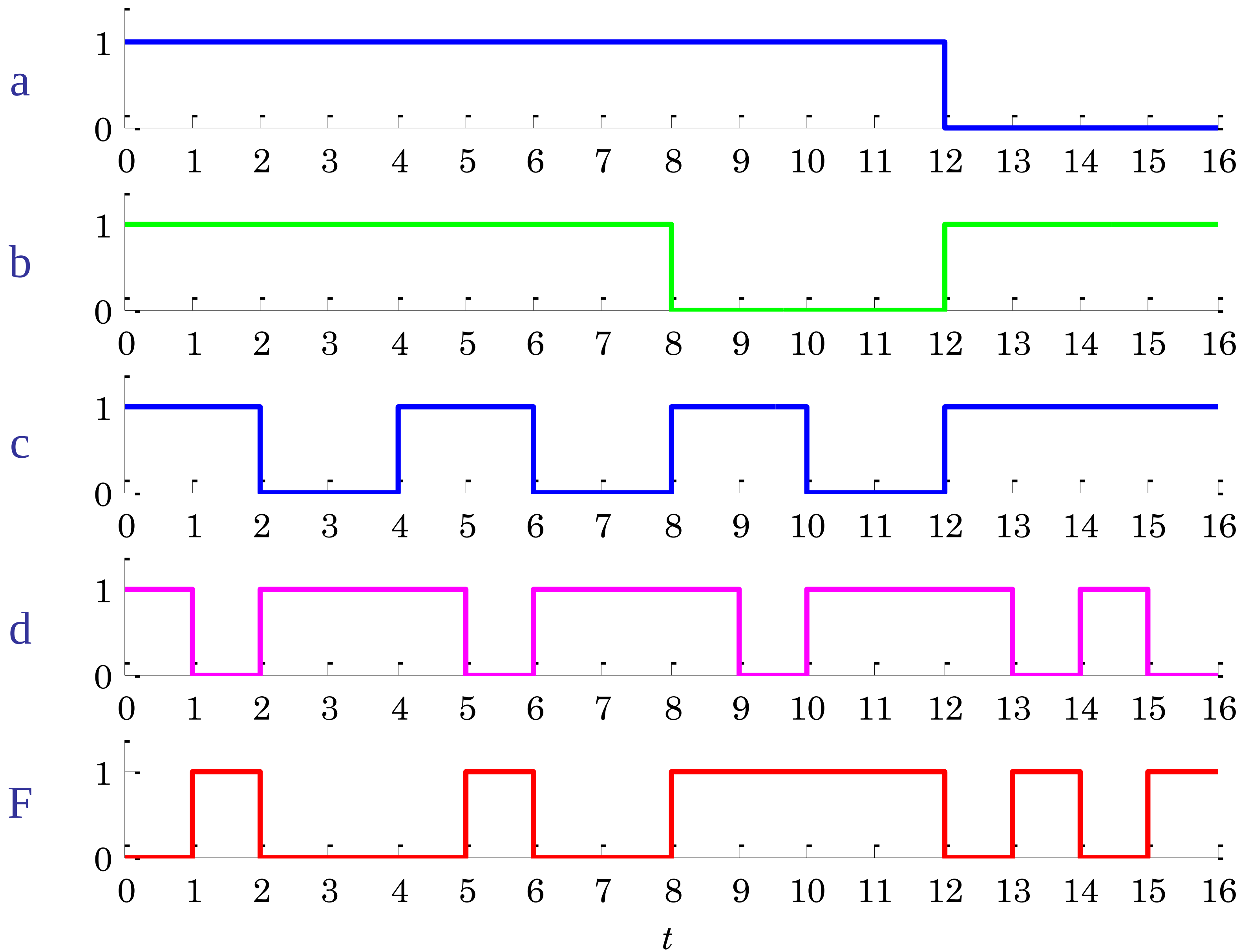


x_1, a, b, c, d, F signals

x_1, x_2, x_3, x_4, F signals



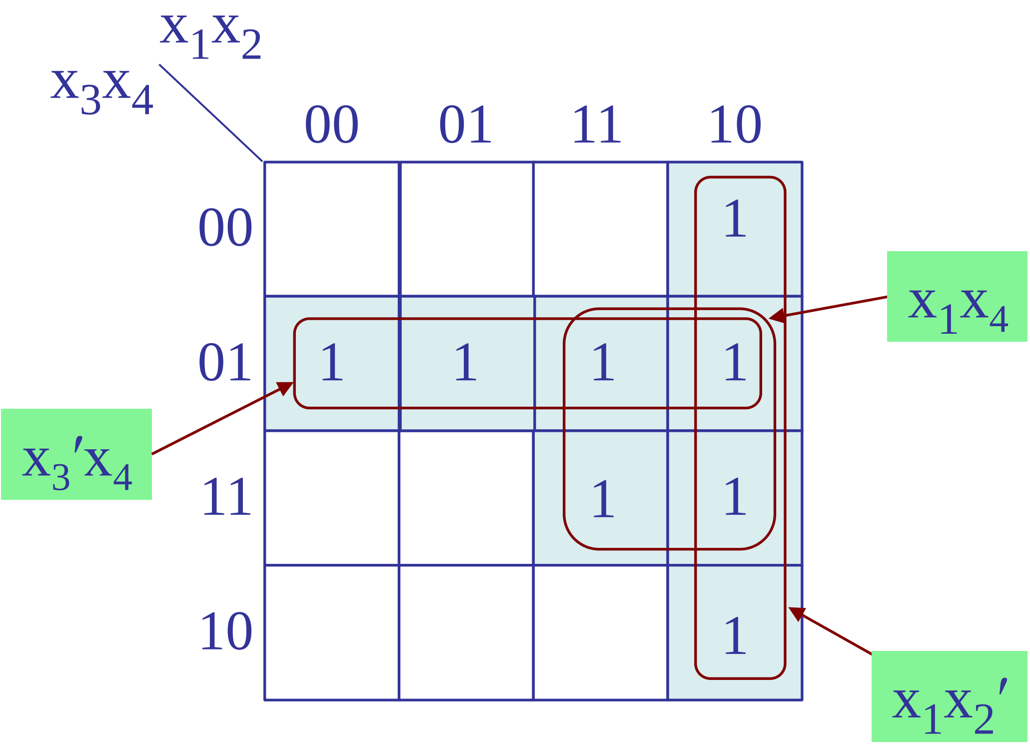
x_1, a, b, c, d, F signals

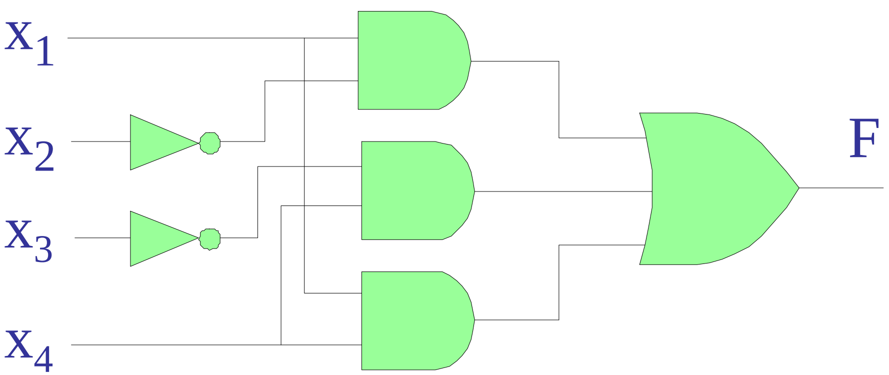


```matlab
t = S.time;          % extract data from structure S
a = S.data(:,2);
b = S.data(:,3);
c = S.data(:,4);
d = S.data(:,5);
F = S.data(:,6);

x1 = X.data(:,1);    % extract data from structure X
x2 = X.data(:,2);
x3 = X.data(:,3);
x4 = X.data(:,4);

figure;
subplot(5,1,1); stairs(t,x1,'b-'); yaxis(0,1.5,0:1)
subplot(5,1,2); stairs(t,x2,'b-'); yaxis(0,1.5,0:1)
subplot(5,1,3); stairs(t,x3,'b-'); yaxis(0,1.5,0:1)
subplot(5,1,4); stairs(t,x4,'b-'); yaxis(0,1.5,0:1)
subplot(5,1,5); stairs(t,F,'r-'); yaxis(0,1.5,0:1)
xlabel('\itt');

figure;
subplot(5,1,1); stairs(t,a,'b-'); yaxis(0,1.5,0:1)
subplot(5,1,2); stairs(t,b,'g-'); yaxis(0,1.5,0:1)
subplot(5,1,3); stairs(t,c,'b-'); yaxis(0,1.5,0:1)
subplot(5,1,4); stairs(t,d,'m-'); yaxis(0,1.5,0:1)
subplot(5,1,5); stairs(t,F,'r-'); yaxis(0,1.5,0:1)
xlabel('\itt');
```

x1	x2	x3	x4	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

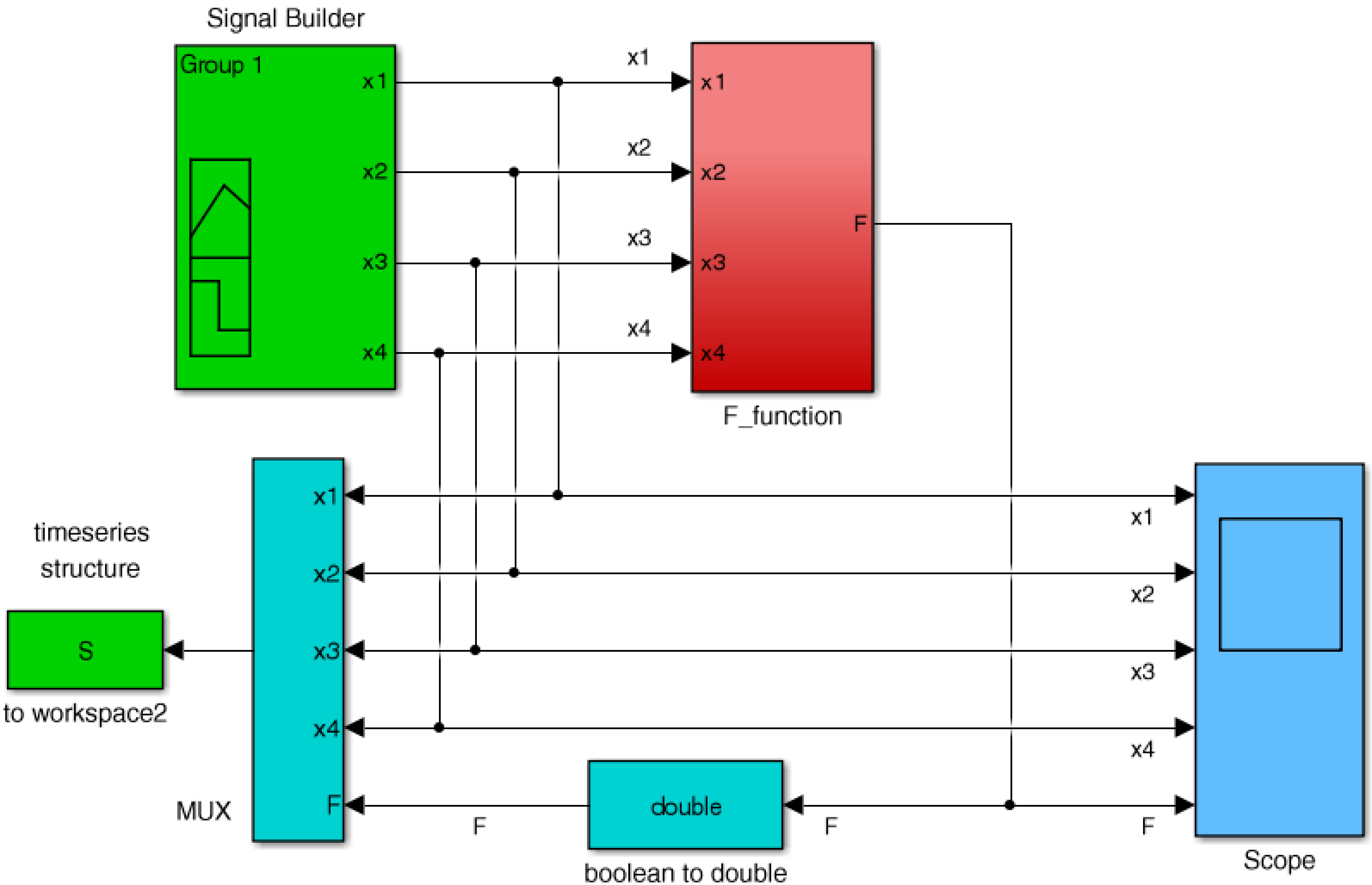
4-bit binary pattern





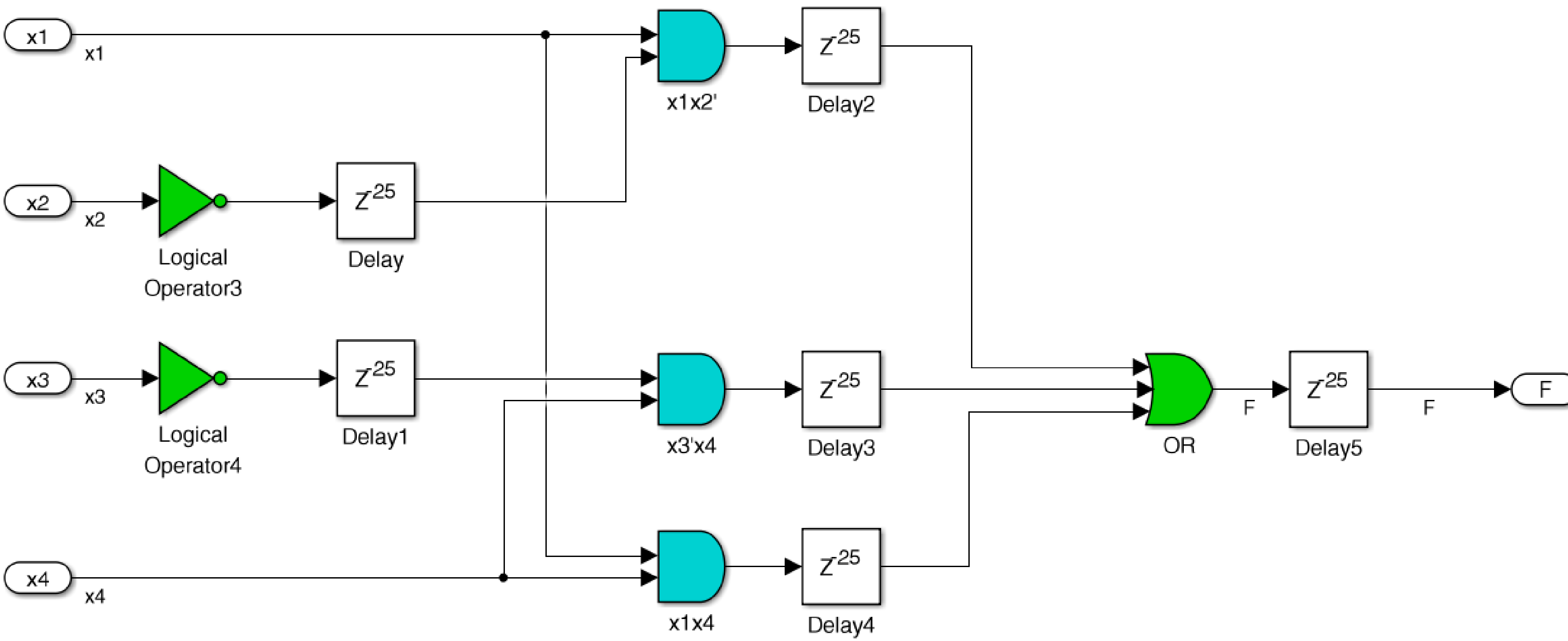
$$F = x_1x_4 + x_1x_2' + x_3'x_4$$

$$F = x_1x_4 + x_1x_2' + x_3'x_4$$



Simulink file: dhazard4.slx, with delays

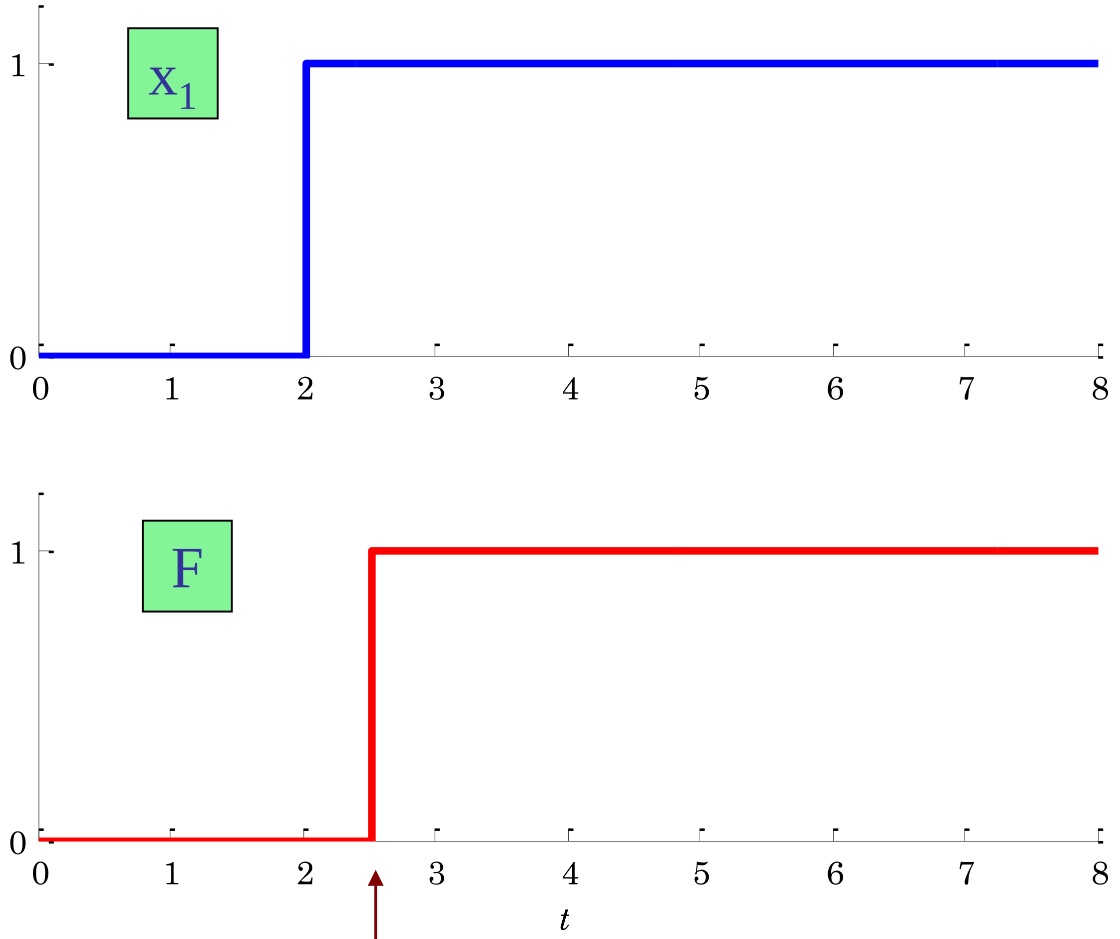
$$F = x_1x_4 + x_1x_2' + x_3'x_4$$



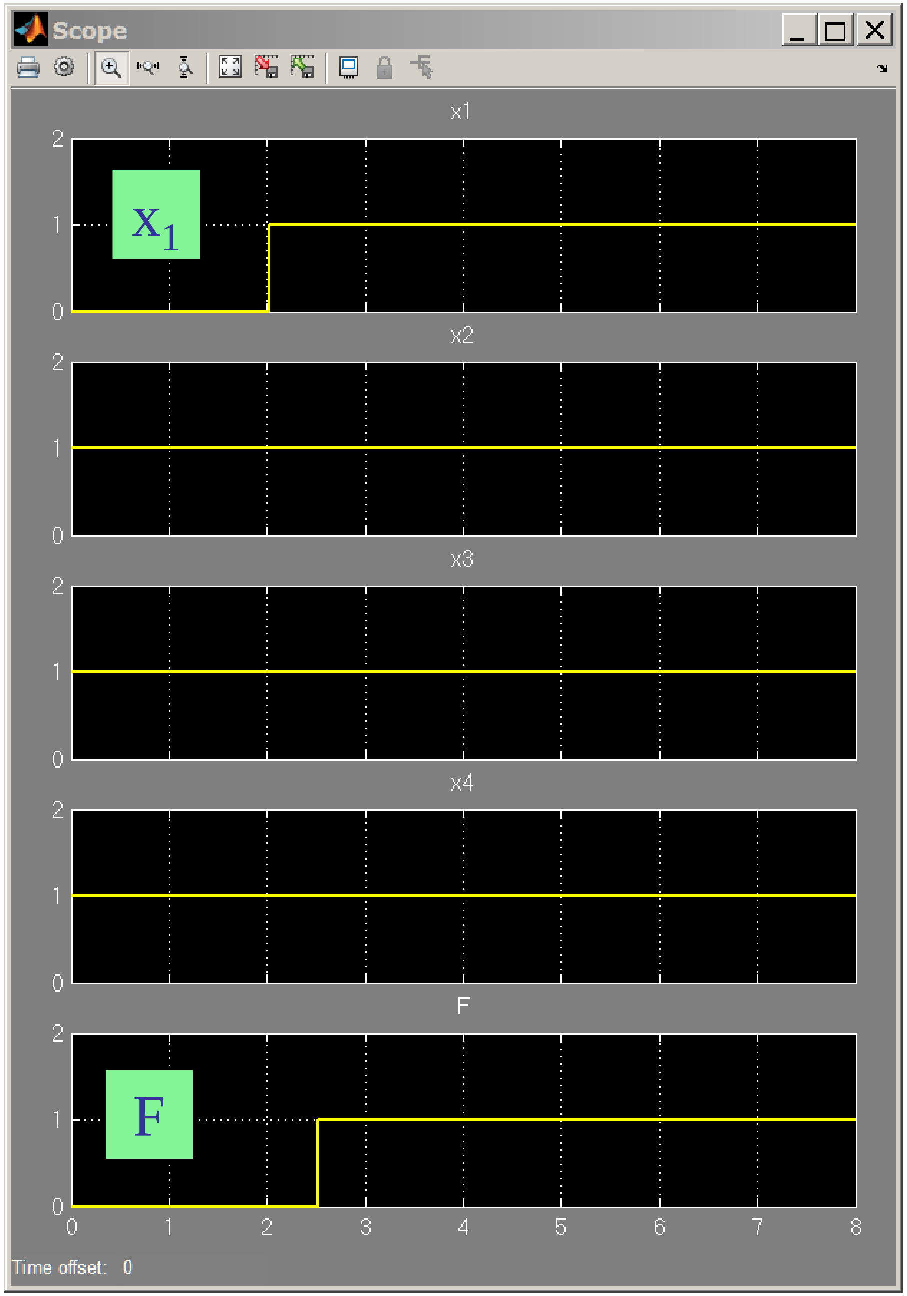
does not exhibit any static or dynamic hazards, apart from an overall delay

$x_2 = x_3 = x_4 = 1$

$F = x_1 x_4 + x_1 x_2' + x_3' x_4$



overall two-gate delay, relative to x_1



$$x_2 = x_3 = x_4 = 1$$

$$F = x_1x_4 + x_1x_2' + x_3'x_4$$