DIGITAL LOGIC CIRCUITS

Digital logic circuits ⇨ electronic circuits that handle information encoded in binary form (deal with signals that have only two values, **0** and **1**)

- ***Digital*** …. computers, watches, controllers, telephones, cameras, ...

★ BINARY NUMBER SYSTEM

Number ….in whatever base	*Decimal value of the given number*
Decimal: **1,998**	$= 1\text{x}10^3 + 9\text{x}10^2 + 9\text{x}10^1 + 8\text{x}10^0 = 1{,}000+900+90+8$ = **1,998**
Binary: **11111001110**	$= 1\text{x}2^{10} + 1\text{x}2^9 + 1\text{x}2^8 + 1\text{x}2^7 + 1\text{x}2^6 + 1\text{x}2^3 + 1\text{x}2^2 + 1\text{x}2^1 =$ $1{,}024+512+258+128+64+8+4+2$ = **1,998**

Powers of 2

N	2^N	*Comments*
0	1	
1	2	
2	4	
3	8	
4	16	
5	32	
6	64	
7	128	
8	256	
9	512	
10	1,024	"Kilo"as 2^{10} is the closest power of 2 to 1,000 (decimal)
11	2,048	
...	...	
15	32,768	2^{15} Hz often used as clock crystal frequency in digital watches
...	...	
20	1,048,576	"Mega" as 2^{20} is the closest power of 2 to 1,000,000 (decimal)
...	...	
30	1,073,741,824	"Giga" as 2^{30} is the closest power of 2 to 1,000,000,000(decimal)

Negative Powers of 2

N <0	2^N
-1	$2^{-1} = 0.5$
-2	$2^{-2} = 0.25$
-3	$2^{-3} = 0.125$
-4	$2^{-4} = 0.0625$
-5	$2^{-5} = 0.03125$
-6	$2^{-6} = 0.015625$
-7	$2^{-7} = 0.0078125$
-8	$2^{-8} = 0.00390625$
-9	$2^{-9} = 0.001953125$
-10	$2^{-10} = 0.0009765625$
…	

Binary numbers less than 1

Binary	*Decimal value*
0.101101	$= 1x2^{-1} + 1x2^{-3} + 1x2^{-4} + 1x2^{-6} =$ **0.703125**

HEXADECIMAL

Binary:
11111001110

111 1100 1110

$\underbrace{111}_{7}\ \underbrace{1100}_{12}\ \underbrace{1110}_{14}$ <== *Decimal*

Hexadecimal: **7CE** = $7\text{x}16^2 + 12\text{x}16^1 + 14\text{x}16^0 =$ **1998**

Binary	Decimal	**Hexadecimal**
0000	0	**0**
0001	1	**1**
0010	2	**2**
0011	3	**3**
0100	4	**4**
0101	5	**5**
0110	6	**6**
0111	7	**7**
1000	8	**8**
1001	9	**9**
1010	10	**A**
1011	11	**B**
1100	12	**C**
1101	13	**D**
1110	14	**E**
1111	15	**F**

LOGIC OPERATIONS AND TRUTH TABLES

Digital logic circuits handle data encoded in binary form, i.e. signals that have only two values, **0** and **1**.

- Binary logic dealing with "true" and "false" comes in handy to describe the behaviour of these circuits: **0** is usually associated with "**false**" and **1** with "**true**."

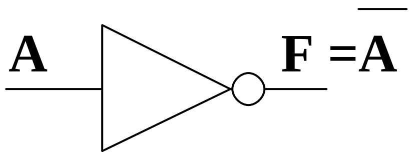
- Quite complex digital logic circuits (e.g. entire computers) can be built using a few *types of basic circuits* called **gates,** each performing a single elementary logic operation : *NOT*, *AND*, *OR*, ***NAND***, *NOR*, etc..

- Boole's binary algebra is used as a formal / mathematical tool to describe and design complex binary logic circuits.

GATES

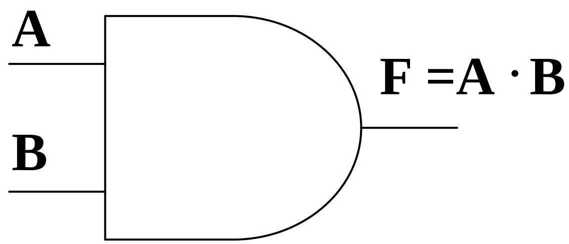
A	$\overline{A}$
0	1
1	0

NOT



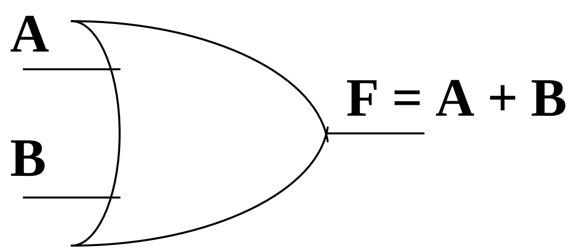
A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

AND



A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1

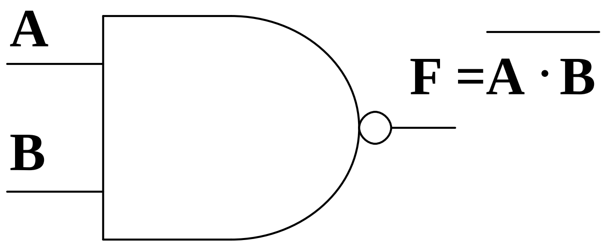
OR



… more GATES

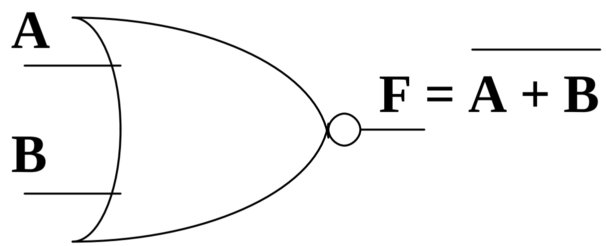
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

NAND



A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

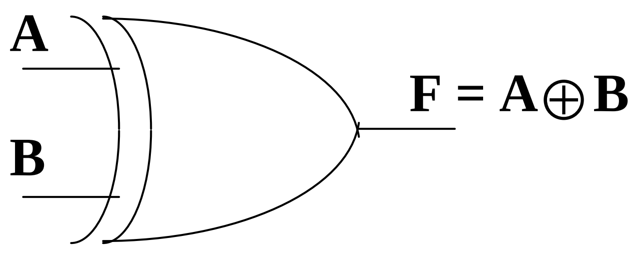
NOR



… and more GATES

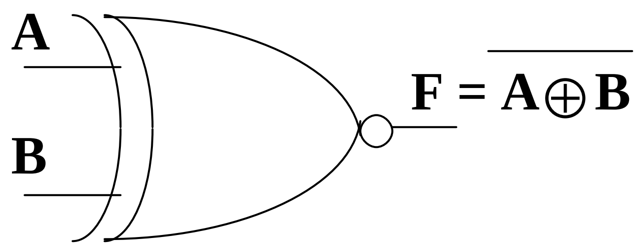
XOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



EQU or *XNOR*

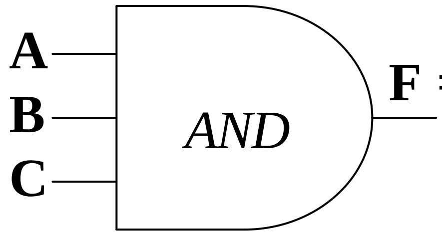
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1



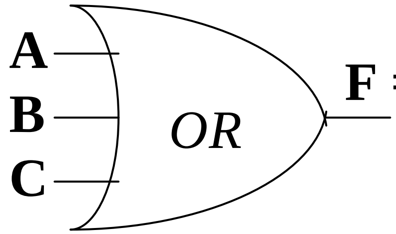
GATES … with more inputs

EXAMPLES OF GATES WITH THREE INPUTS

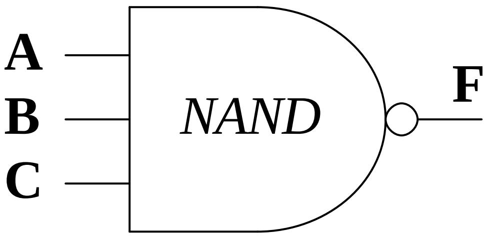
A	B	C	$A\cdot B\cdot C$	$A+B+C$	$\overline{A\cdot B\cdot C}$	$\overline{A+B+C}$
0	0	0	0	0	1	1
0	0	1	0	1	1	0
0	1	0	0	1	1	0
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	0	1	0	1	1	0
1	1	0	0	1	1	0
1	1	1	1	1	0	0



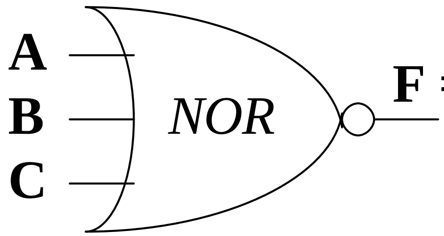
F $= A\cdot B\cdot C$



F $= A+B+C$



F $= \overline{A\cdot B\cdot C}$

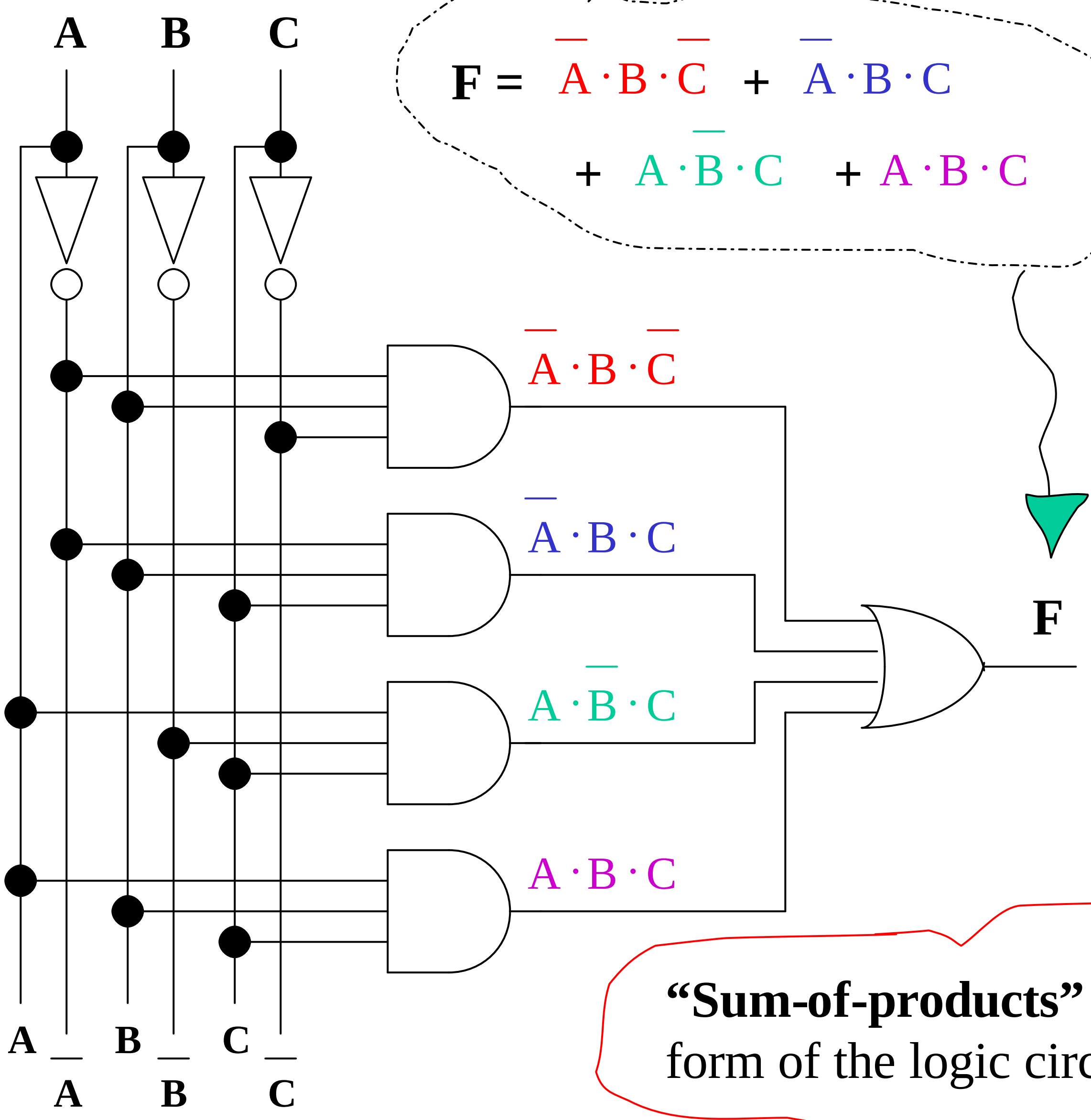


F $= \overline{A+B+C}$

Logic Gate Array that Produces an Arbitrarily Chosen Output

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$F = \bar{A}\cdot B\cdot \bar{C} + \bar{A}\cdot B\cdot C + A\cdot \bar{B}\cdot C + A\cdot B\cdot C$$



"Sum-of-products" form of the logic circuit.

BOOLEAN ALGEBRA

AND rules

$$\mathbf{A \cdot A = A}$$

$$\mathbf{A \cdot \overline{A} = 0}$$

$$\mathbf{0 \cdot A = 0}$$

$$\mathbf{1 \cdot A = A}$$

$$\mathbf{A \cdot B = B \cdot A}$$

$$\mathbf{A \cdot (B \cdot C) = (A \cdot B) \cdot C}$$

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

$$\mathbf{\overline{A} \cdot \overline{B} = \overline{A + B}}$$

" Proof ":

A B C	A· (B+C)	A·B+A·C
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	0	0
1 0 0	0	0
1 0 1	1	1
1 1 0	1	1
1 1 1	1	1

***OR* rules**

$$\mathbf{A + A = A}$$

$$\mathbf{A + \overline{A} = 1}$$

$$\mathbf{0 + A = A}$$

$$\mathbf{1 + A = 1}$$

$$\mathbf{A + B = B + A}$$

$$\mathbf{A + (B + C) = (A + B) + C}$$

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

$$\mathbf{\overline{A} + \overline{B} = \overline{A \cdot B}}$$

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

DeMorgan's Theorem

$$\overline{A}\cdot\overline{B} = \overline{A+B}$$

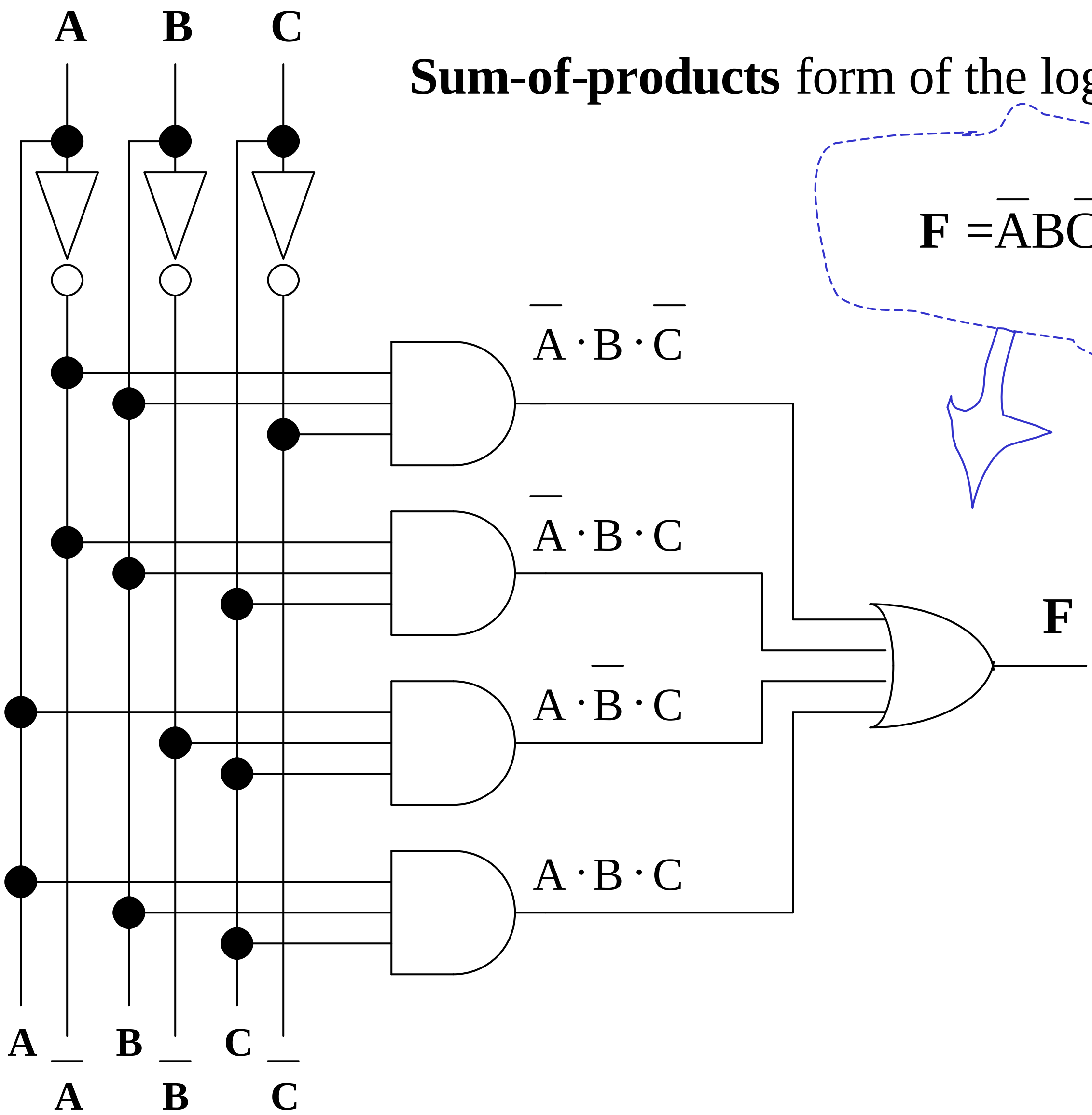
$$\overline{A}+\overline{B} = \overline{A\cdot B}$$

A	B	$\overline{A}\cdot\overline{B}$	$\overline{A+B}$	$\overline{A}+\overline{B}$	$\overline{A\cdot B}$
0	0	1	1	1	1
0	1	0	0	1	1
1	0	0	0	1	1
1	1	0	0	0	0

Simplifying logic functions using Boolean algebra rules

Sum-of-products form of the logic function:

$$\mathbf{F} = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$



Simplifying logic functions using Boolean algebra rules … continued

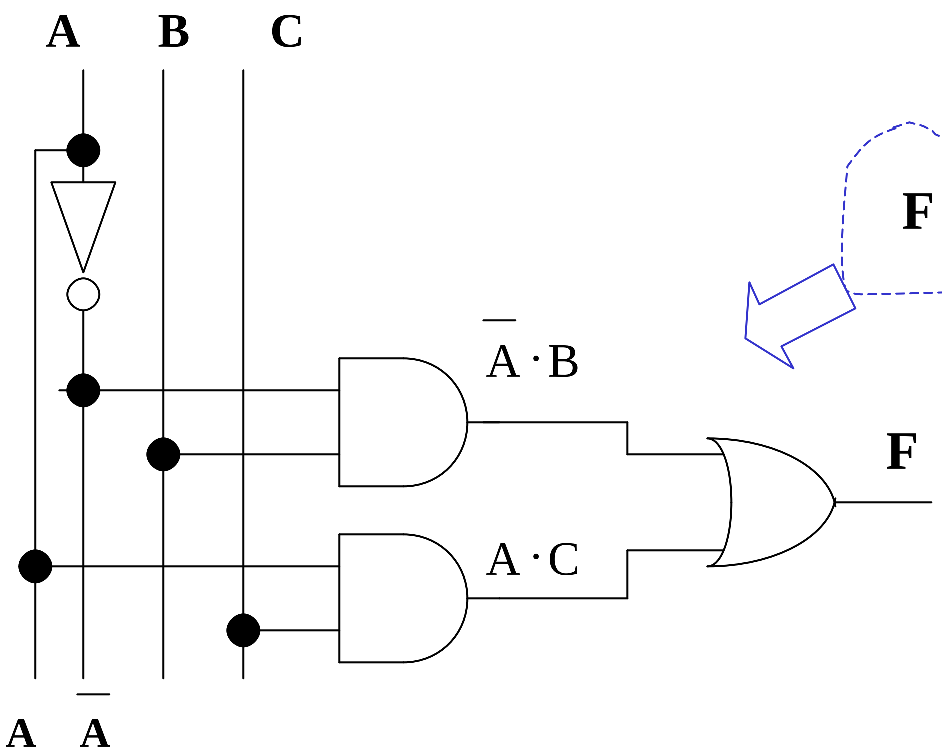
$$F = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + ABC$$

$$F = (\overline{A}B\overline{C} + \overline{A}BC) + (A\overline{B}C + ABC)$$

$$F = \overline{A}(B\overline{C} + BC) + A(\overline{B}C + BC)$$

$$F = \overline{A}B\underbrace{(\overline{C} + C)}_{1} + AC\underbrace{(\overline{B} + B)}_{1}$$

$F = \overline{A}B + AC$



Simplifying logic functions using Karnaugh maps

- **Karnaugh map** => graphical representation of a truth table for a logic function.

- Each line in the truth table corresponds to a square in the Karnaugh map.

- The Karnaugh map squares are labeled so that horizontally or vertically adjacent squares differ only in one variable. (*Each square in the top row is considered to be adjacent to a corresponding square in the bottom row. Each square in the left most column is considered to be adjacent to a corresponding square in the right most column.*)

	A	**B**	**C**	**F**
(0)	0	0	0	...
(1)	0	0	1	...
(2)	0	1	0	...
(3)	0	1	1	...
(4)	1	0	0	...
(5)	1	0	1	...
(6)	1	1	0	...
(7)	1	1	1	...

C \ **A B**	**00**	**01**	**11**	**10**
0	*0*	*2*	*6*	*4*
1	*1*	*3*	*7*	*5*

Karnaugh map

Simplifying logic functions of 4 variables using Karnaugh maps

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

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*

Simplifying logic functions using Karnaugh maps … looping

- The logic expressions for an output can be simplified by properly combining squares (**looping**) in the Karnaugh maps which contain **1**s.

- Looping a pair of adjacent **1**s eliminates the variable that appears in both direct and complemented form.

	A	B	C	F
(0)	0	0	0	0
(1)	0	0	1	0
(2)	0	1	0	1
(3)	0	1	1	1
(4)	1	0	0	0
(5)	1	0	1	1
(6)	1	1	0	0
(7)	1	1	1	1

C \ **A B**	**00**	**01**	**11**	**10**
0	0	**1**	0	0
1	0	**1**	**1**	**1**

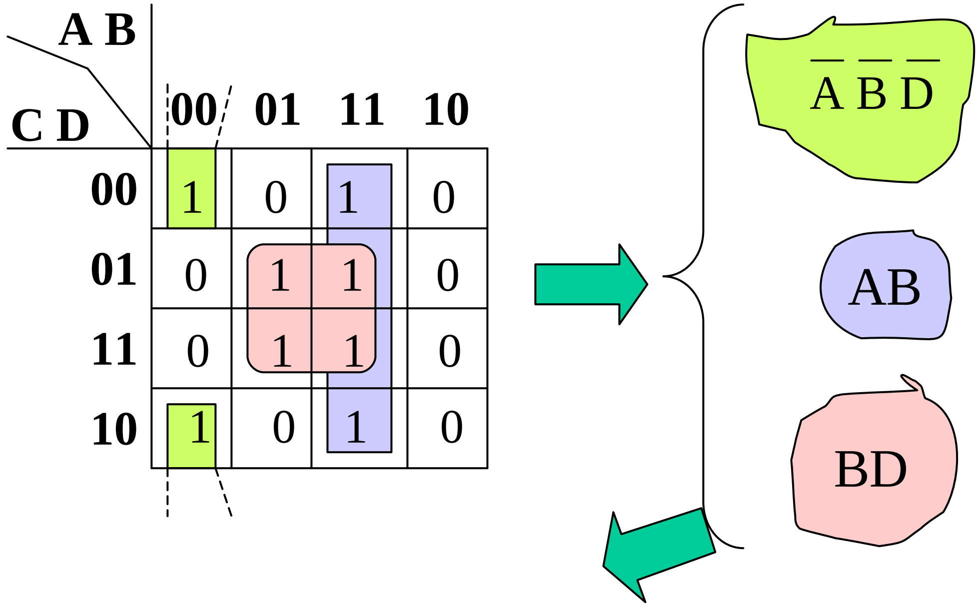
$\overline{A}B$ AC

$$F = \overline{A}B + AC$$

Simplifying logic functions using Karnaugh maps … more looping

	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	0
(4)	0	1	0	0	0
(5)	0	1	0	1	1
(6)	0	1	1	0	0
(7)	0	1	1	1	1
(8)	1	0	0	0	0
(9)	1	0	0	1	0
(10)	1	0	1	0	0
(11)	1	0	1	1	0
(12)	1	1	0	0	1
(13)	1	1	0	1	1
(14)	1	1	1	0	1
(15)	1	1	1	1	1

Looping a *quad* of adjacent **1**s eliminates the two variables that appears in both direct and complemented form.



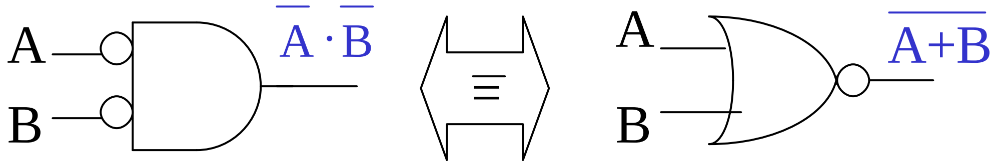
$$F = \overline{A}\,\overline{B}\,\overline{D} + AB + BD$$

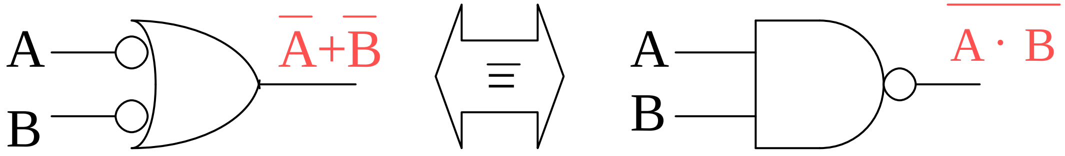
DeMorgan's Theorem

$$\overline{A}\cdot\overline{B} = \overline{A + B}$$

$$\overline{A} + \overline{B} = \overline{A \cdot B}$$

Equivalent Gate Symbols

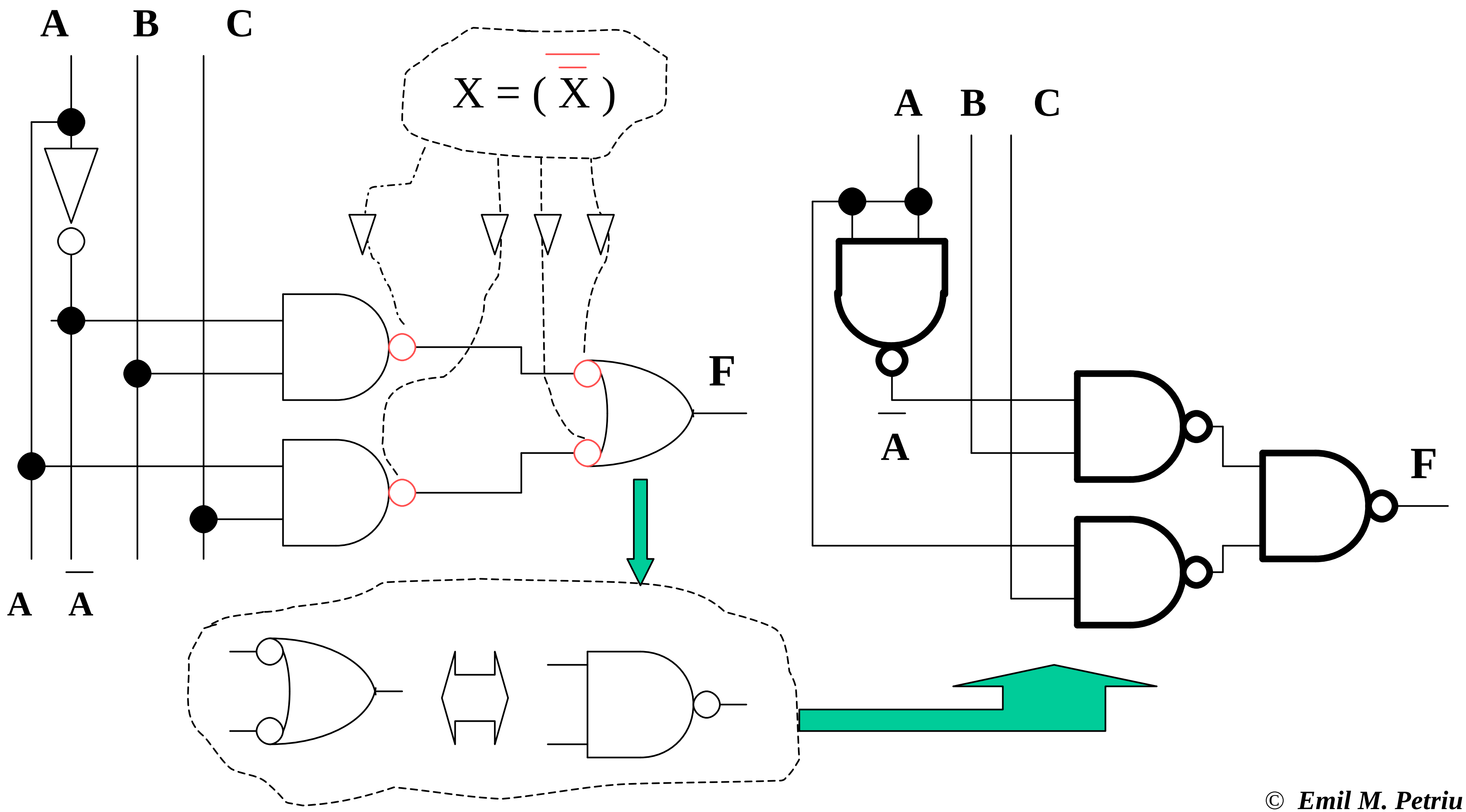




NAND gate implementation of the "sum-of-product" logic functions

$$F = \overline{A}B + AC$$

- NAND gates are faster than ANDs and ORs in most technologies



ADDING BINARY NUMBERS

Adding two bits:

0+	0+	1+	1+
0	1	0	1
0	1	1	1 0

Carry (over) — Sum

The binary number **10** is equivalent to the decimal **2**

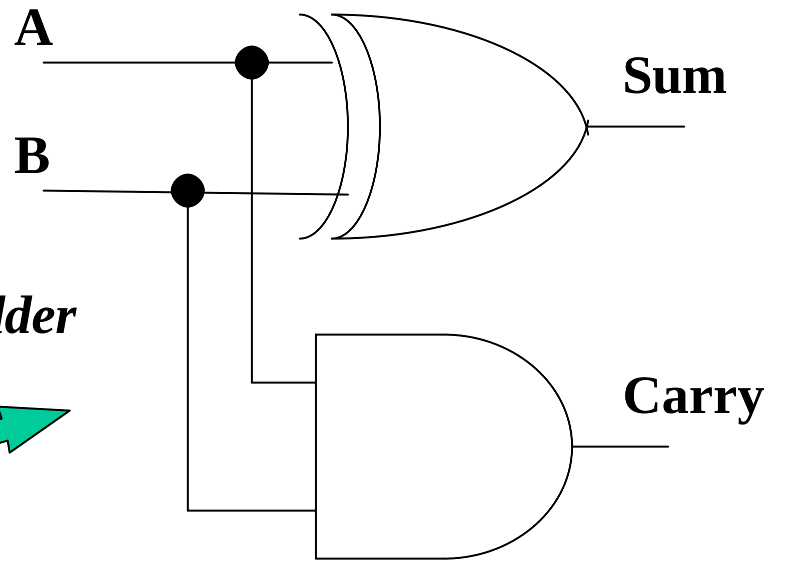
Truth table

Inputs A	Inputs B	Outputs Carry	Outputs Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\text{Sum} = A \oplus B$$

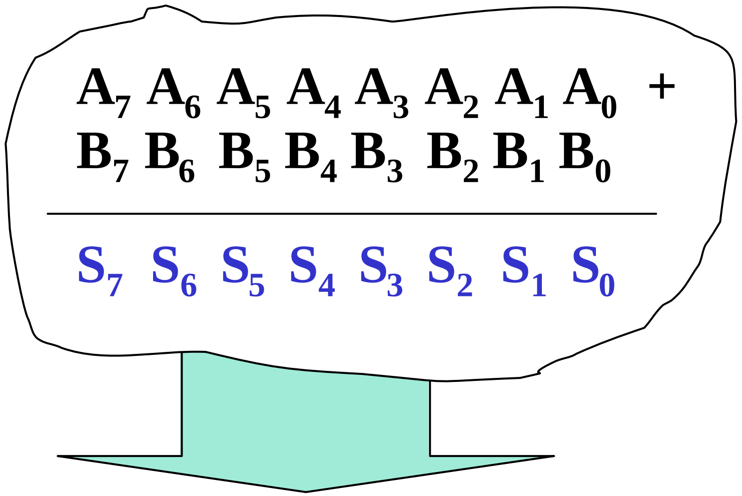
$$\text{Carry} = A \cdot B$$

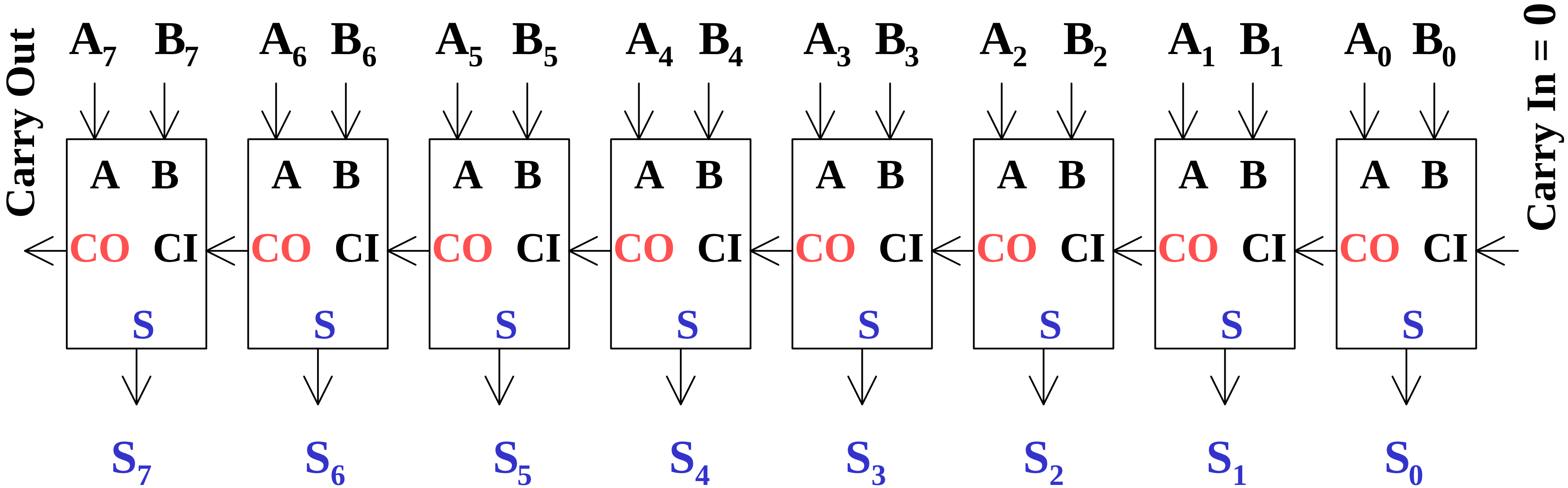
Half-Adder circuit



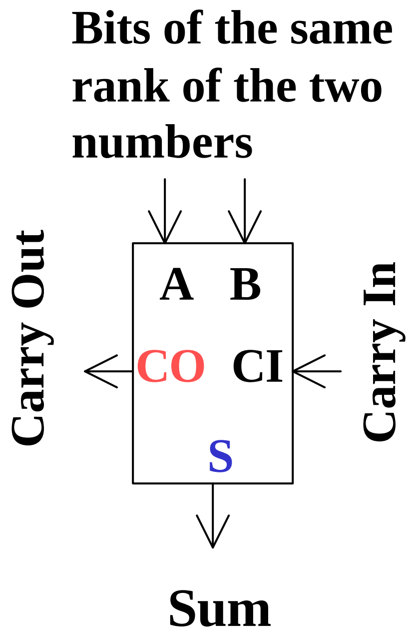
Adding multi-bit numbers:

108_D +	→	0 1 1 0 1 1 0 0 +	
90_D	→	0 1 0 1 1 0 1 0	
198_D	←	1 1 0 0 0 1 1 0	⇐ Sum
		0 1 1 1 1 0 0 0	⇐ Carry





Full Adder



Inputs			Outputs	
A	B	CI	CO	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S

CI \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

CO

CI \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$A \cdot B$ $B \cdot CI$ $A \cdot CI$

$$S = \overline{A} \cdot \overline{B} \cdot CI + \overline{A} \cdot B \cdot \overline{CI} + A \cdot \overline{B} \cdot \overline{CI} + A \cdot B \cdot CI$$

$$C = A \cdot B + B \cdot CI + A \cdot CI$$

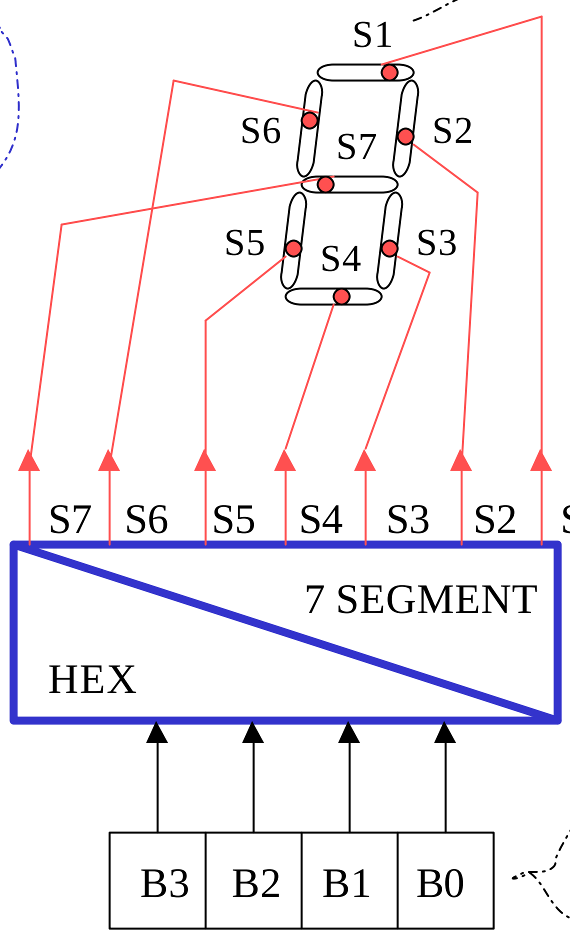
HEX-TO-7 SEGMENT DECODER

This examples illustrates how a practical problem is analyzed in order to generate truth tables,and then how truth table-defined functions are mapped on Karnaugh maps.

	B3	B2	B1	B0
(0)	0	0	0	0
(1)	0	0	0	1
(2)	0	0	1	0
(3)	0	0	1	1
(4)	0	1	0	0
(5)	0	1	0	1
(6)	0	1	1	0
(7)	0	1	1	1
(8)	1	0	0	0
(9)	1	0	0	1
(A)	1	0	1	0
(B)	1	0	1	1
(C)	1	1	0	0
(D)	1	1	0	1
(E)	1	1	1	0
(F)	1	1	1	1

Four-bit "machine" representation of the hex digits

"Natural" (i.e.as humans write) representation of the "hex" digits.

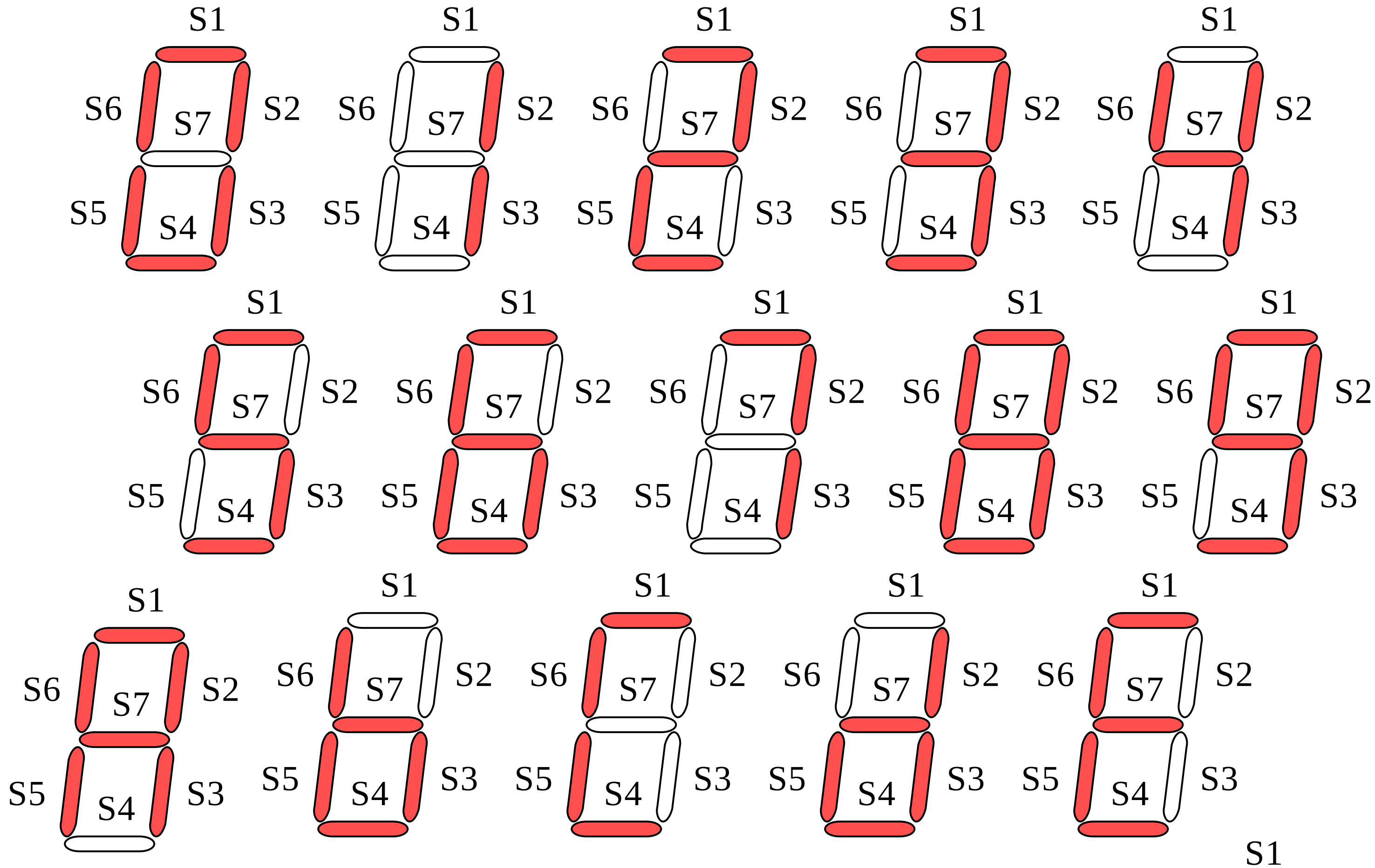


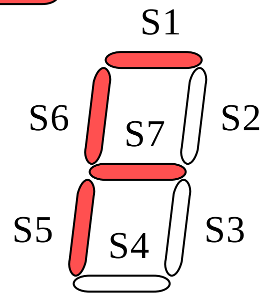
7-segment display to allow the user see the natural representation of the hex digits.

Binary outputs; when such a signal is = 1 then the corresponding segment in the 7-segment display is on (you see it), when the signal is = 0 then the segment is off (you can't see it).

The four-bit representation of the hex digits

	B3	B2	B1	B0
(0)	0	0	0	0
(1)	0	0	0	1
(2)	0	0	1	0
(3)	0	0	1	1
(4)	0	1	0	0
(5)	0	1	0	1
(6)	0	1	1	0
(7)	0	1	1	1
(8)	1	0	0	0
(9)	1	0	0	1
(A)	1	0	1	0
(B)	1	0	1	1
(C)	1	1	0	0
(D)	1	1	0	1
(E)	1	1	1	0
(F)	1	1	1	1





We are developing ad-hoc"binary-hex logic" expressions used just for our convenience in the problem analysis process. Each expression will enumerate only those hex digits when the specific display-segment is "on":

S1 = 0+2 +3+5+6+7+8+9+A+C+E+F

S2 = 0+1+2+3+4+7+8+9+A+D

S3 = 0+1+3+4+5+6+7+8+9+A+B+D

S4 = 0+2+3+5+6+8+9+B+C+D+E

S5 = 0+2+6+8+A+B+C+D+E+F

S6 = 0+4+5+6+8+9+A+B+C+E+F

S7 = 2+3+4+5+6+8+9+A+B+D+E+F

Hex-to-7 segment

	B3	B2	B1	B0
(0)	0	0	0	0
(1)	0	0	0	1
(2)	0	0	1	0
(3)	0	0	1	1
(4)	0	1	0	0
(5)	0	1	0	1
(6)	0	1	1	0
(7)	0	1	1	1
(8)	1	0	0	0
(9)	1	0	0	1
(A)	1	0	1	0
(B)	1	0	1	1
(C)	1	1	0	0
(D)	1	1	0	1
(E)	1	1	1	0
(F)	1	1	1	1

S1 = 0+2 +3+5+6+7+8+9+A+C+E+F

S2 = 0+1+2+3+4+7+8+9+A+D

S3 = 0+1+3+4+5+6+7+8+9+A+B+D

S4 = 0+2+3+5+6+8+9+B+C+D+E

S5 = 0+2+6+8+A+B+C+D+E+F

S6 = 0+4+5+6+8+9+A+B+C+E+F

S7 = 2+3+4+5+6+8+9+A+B+D+E+F

B1 B0 \ B3 B2	00	01	11	10
00	*0*	*4*	*C*	*8*
01	*1*	*5*	*D*	*9*
11	*3*	*7*	*F*	*B*
10	*2*	*6*	*E*	*A*

As we are using ad-hoc “binary-hex logic” equations, (i.e. binary S… outputs as functions of hex variables) it will useful in this case to have hex-labeled Karnaugh map, instead of the usual 2-D (i.e. two dimensional) binary labeled K maps. This will allow for a more convenient mapping of the “binary-hex” logic equations onto the K-maps.

Hex-to-7 segment

Mapping the ad-hoc "binary-hex logic" equations onto Karnaugh maps:

B1 B0 \ B3 B2	00	01	11	10
00	*0*	*4*	*C*	*8*
01	*1*	*5*	*D*	*9*
11	*3*	*7*	*F*	*B*
10	*2*	*6*	*E*	*A*

S1 = 0+2 +3+5+6+7+8+9+A+C+E+F

S2 = 0+1+2+3+4+7+8+9+A+D

S3 = 0+1+3+4+5+6+7+8+9+A+B+D

S4 = 0+2+3+5+6+8+9+B+C+D+E

S1

B1 B0 \ B3 B2	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	1	1	1	0
10	1	1	1	1

S2

B1 B0 \ B3 B2	00	01	11	10
00	1	1	0	1
01	1	0	1	1
11	1	1	0	0
10	1	0	0	1

S3

B1 B0 \ B3 B2	00	01	11	10
00	1	1	0	1
01	1	1	1	1
11	1	1	0	1
10	0	1	0	1

S4

B1 B0 \ B3 B2	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	0	1
10	1	1	1	0

◆ Hex-to-7 segment

B1 B0 \ B3 B2	00	01	11	10
00	*0*	*4*	*C*	*8*
01	*1*	*5*	*D*	*9*
11	*3*	*7*	*F*	*B*
10	*2*	*6*	*E*	*A*

S5

B1 B0 \ B3 B2	00	01	11	10
00	1	0	1	1
01	0	0	1	0
11	0	0	1	1
10	1	1	1	1

S6

B1 B0 \ B3 B2	00	01	11	10
00	1	1	1	1
01	0	1	0	1
11	0	0	1	1
10	0	1	1	1

S7

B1 B0 \ B3 B2	00	01	11	10
00	0	1	0	1
01	0	1	1	1
11	1	0	1	1
10	1	1	1	1

S5 = 0+2+6+8+A+B+C+D+E+F

S6 = 0+4+5+6+8+9+A+B+C+E+F

S7 = 2+3+4+5+6+8+9+A+B+D+E+F

SYSTEMS of LOGIC FUNCTIONS ▷ 2- bit Comparator

	A_1	A_0	B_1	B_0	F_1	F_2	F_3
(0)	0	0	0	0	1	0	0
(1)	0	0	0	1	0	0	1
(2)	0	0	1	0	0	0	1
(3)	0	0	1	1	0	0	1
(4)	0	1	0	0	0	1	0
(5)	0	1	0	1	1	0	0
(6)	0	1	1	0	0	0	1
(7)	0	1	1	1	0	0	1
(8)	1	0	0	0	0	1	0
(9)	1	0	0	1	0	1	0
(10)	1	0	1	0	1	0	0
(11)	1	0	1	1	0	0	1
(12)	1	1	0	0	0	1	0
(13)	1	1	0	1	0	1	0
(14)	1	1	1	0	0	1	0
(15)	1	1	1	1	1	0	0

Compare two 2-bit numbers:

A=B ⇨ $F_1 = \Sigma\,(0,5,10,15)$

A>B ⇨ $F_2 = \Sigma\,(4,8,9,12,13,14)$

A<B ⇨ $F_3 = \Sigma\,(1,2,3,6,7,11)$

2- bit Comparator

A=B ⇨ $F_1 = \Sigma\,(0,5,10,15)$

B_1B_0 \ A_1A_0	00	01	11	10
00	*0*	*4*	*12*	*8*
01	*1*	*5*	*13*	*9*
11	*3*	*7*	*15*	*11*
10	*2*	*6*	*14*	*10*

B_1B_0 \ A_1A_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$\mathbf{F_1} = 1$ when both numbers, A and B, are equal which happens when all their bits of the same order are identical, i.e. $A_0 \overline{\equiv} B_0$ *AND* $A_1 \overline{\equiv} B_1$

$$F_1 = \overline{(A_0 \oplus B_0)} \bullet \overline{(A_1 \oplus B_1)}$$

2- bit Comparator

$$A<B \Rightarrow F_3 = \Sigma\,(1,2,3,6,7,11)$$

$B_1 B_0$ \ $A_1 A_0$	00	01	11	10
00	*0*	*4*	*12*	*8*
01	*1*	*5*	*13*	*9*
11	*3*	*7*	*15*	*11*
10	*2*	*6*	*14*	*10*

$B_1 B_0$ \ $A_1 A_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$F_3 = \overline{A_0} B_1 B_0 + \overline{A_1} B_1 + \overline{A_1}\,\overline{A_0} B_0$$

2- bit Comparator

A>B ⇨ $F_2 = \Sigma\,(4,8,9,12,13,14)$

B_1B_0 \ A_1A_0	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

B_1B_0 \ A_1A_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$F_2 = \overline{F_1 + F_3}$$

F_1+F_3

B_1B_0 \ A_1A_0	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	1	1	1	1
10	1	1	0	1

F_1

B_1B_0 \ A_1A_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

F_3

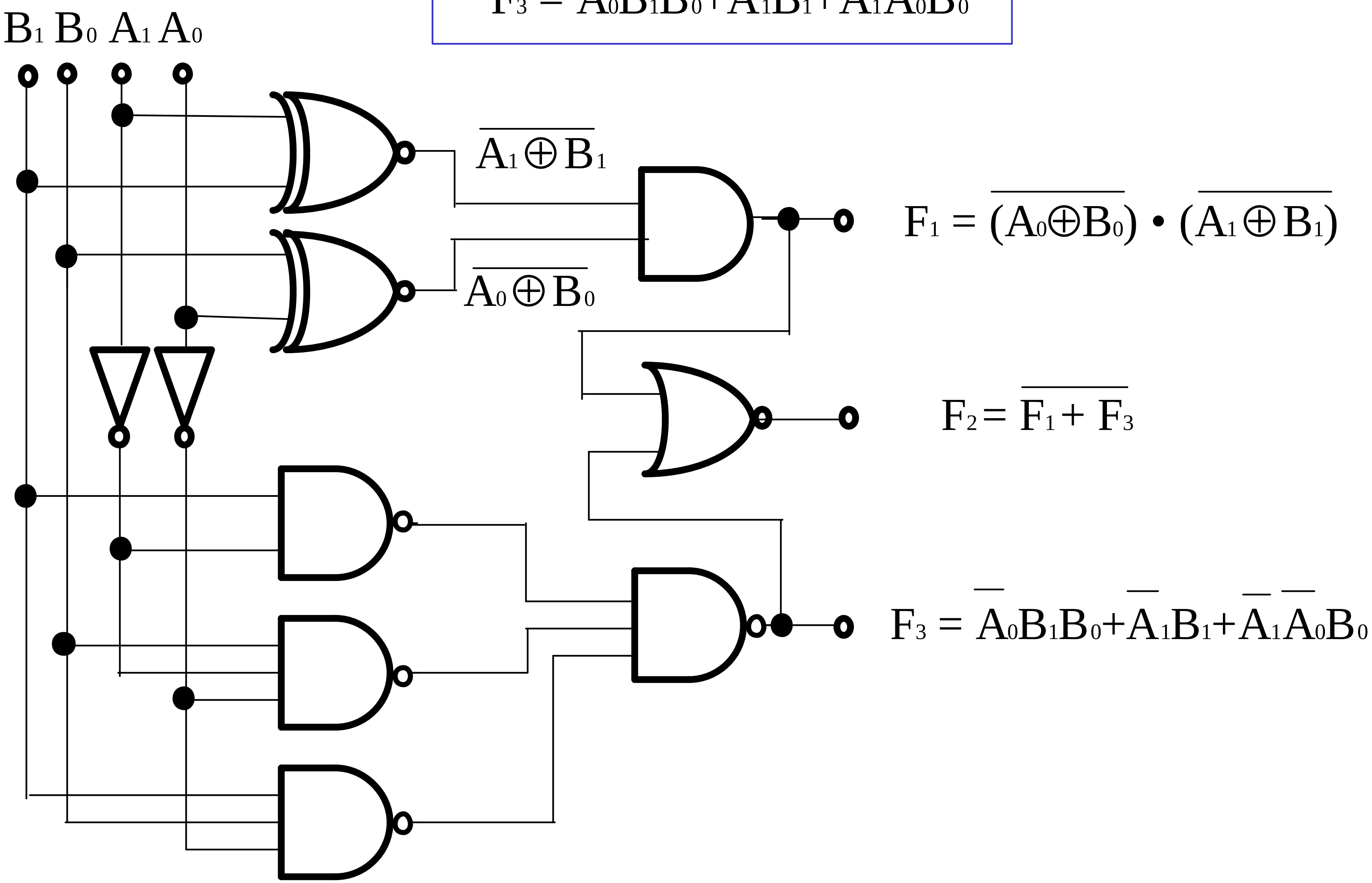
B_1B_0 \ A_1A_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

2- bit Comparator

$$F_1 = \overline{(A_0 \oplus B_0)} \bullet \overline{(A_1 \oplus B_1)}$$

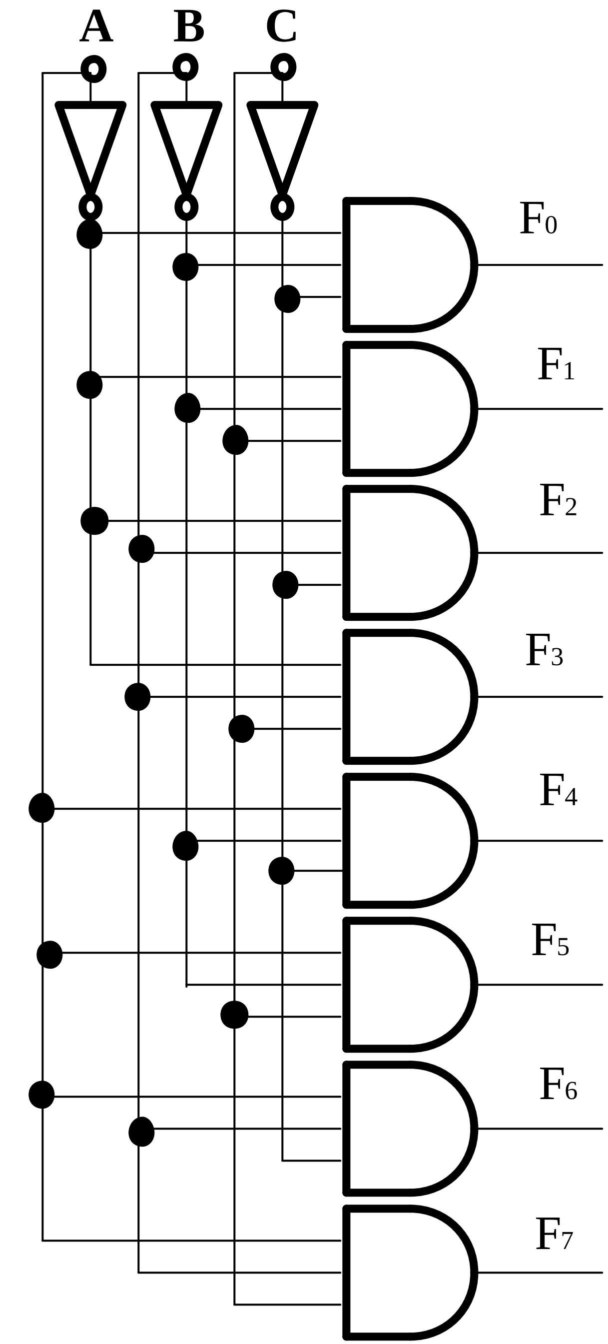
$$F_2 = \overline{F_1 + F_3}$$

$$F_3 = \overline{A_0}B_1B_0 + \overline{A_1}B_1 + \overline{A_1}\,\overline{A_0}B_0$$



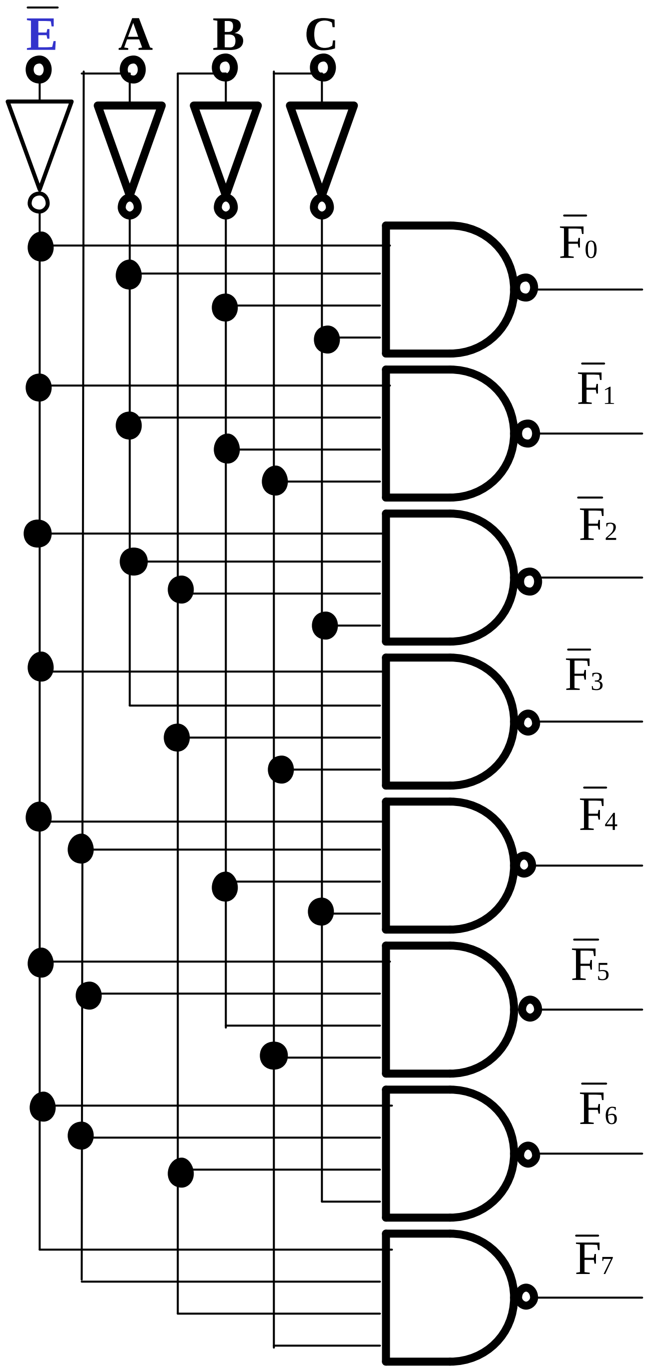
3-to-8 Decoder

	A	B	C	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
(0)	0	0	0	1	0	0	0	0	0	0	0
(1)	0	0	1	0	1	0	0	0	0	0	0
(2)	0	1	0	0	0	1	0	0	0	0	0
(3)	0	1	1	0	0	0	1	0	0	0	0
(4)	1	0	0	0	0	0	0	1	0	0	0
(5)	1	0	1	0	0	0	0	0	1	0	0
(6)	1	1	0	0	0	0	0	0	0	1	0
(7)	1	1	1	0	0	0	0	0	0	0	1

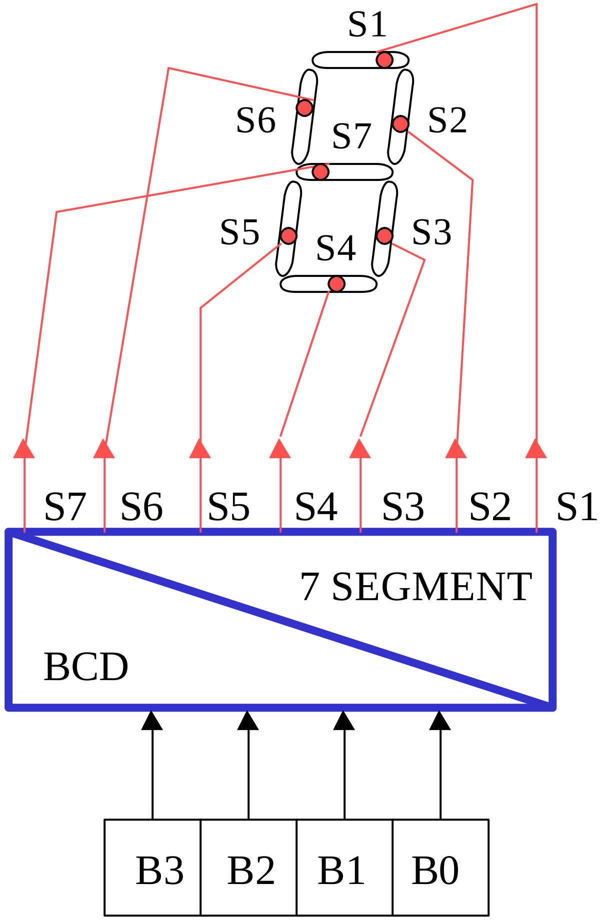


3-to-8 Decoder (74 138)

	A	B	C	$\overline{E}$	$\overline{F}_0$	$\overline{F}_1$	$\overline{F}_2$	$\overline{F}_3$	$\overline{F}_4$	$\overline{F}_5$	$\overline{F}_6$	$\overline{F}_7$
(x)	x	x	x	1	1	1	1	1	1	1	1	1
(0)	0	0	0	0	0	1	1	1	1	1	1	1
(1)	0	0	1	0	1	0	1	1	1	1	1	1
(2)	0	1	0	0	1	1	0	1	1	1	1	1
(3)	0	1	1	0	1	1	1	0	1	1	1	1
(4)	1	0	0	0	1	1	1	1	0	1	1	1
(5)	1	0	1	0	1	1	1	1	1	0	1	1
(6)	1	1	0	0	1	1	1	1	1	1	0	1
(7)	1	1	1	0	1	1	1	1	1	1	1	0



BCD-TO-7 SEGMENT DECODER



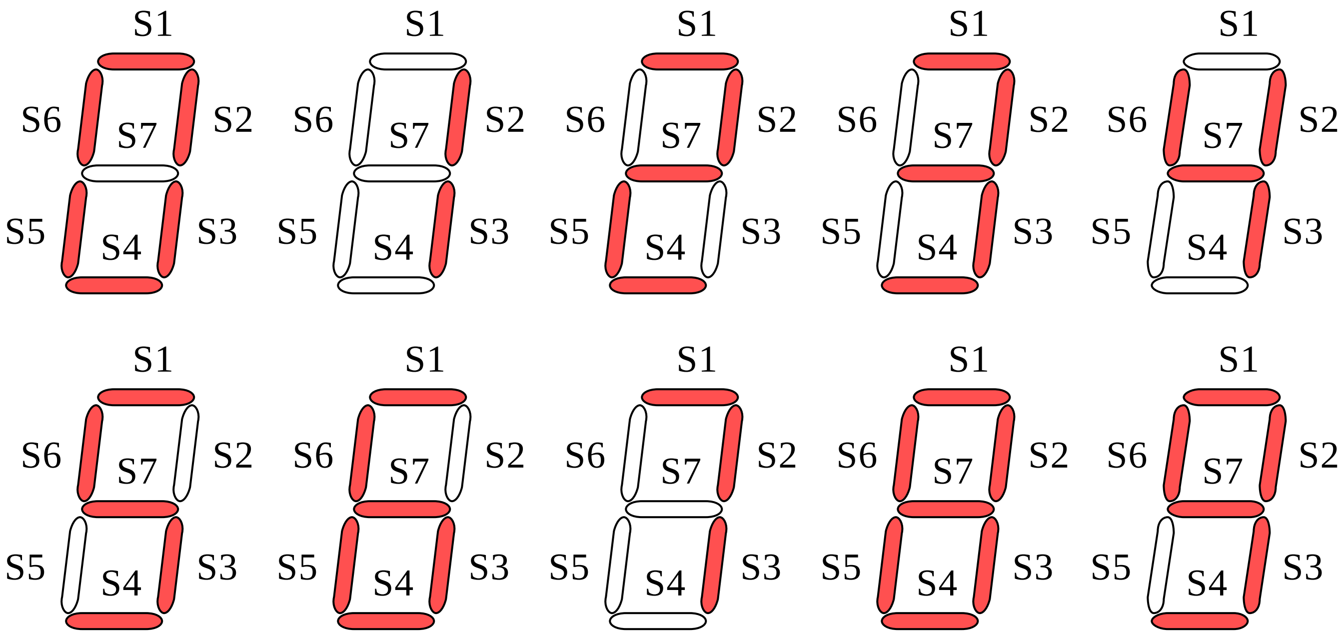
	B3	B2	B1	B0
(0)	0	0	0	0
(1)	0	0	0	1
(2)	0	0	1	0
(3)	0	0	1	1
(4)	0	1	0	0
(5)	0	1	0	1
(6)	0	1	1	0
(7)	0	1	1	1
(8)	1	0	0	0
(9)	1	0	0	1
(x)	1	0	1	0
(x)	1	0	1	1
(x)	1	1	0	0
(x)	1	1	0	1
(x)	1	1	1	0
(x)	1	1	1	1

B1 B0 \ **B3 B2**	**00**	**01**	**11**	**10**
00	0	4	x	8
01	1	5	x	9
11	3	7	x	x
10	2	6	x	x

"Don't Care" states/situations. As it is expected that these states are never going to occur, then we may just as well use them as fill-in "1s" in a Karnaugh map if this helps to make larger loopings

BCD-to-7 segment

	B3	B2	B1	B0
(0)	0	0	0	0
(1)	0	0	0	1
(2)	0	0	1	0
(3)	0	0	1	1
(4)	0	1	0	0
(5)	0	1	0	1
(6)	0	1	1	0
(7)	0	1	1	1
(8)	1	0	0	0
(9)	1	0	0	1
(x)	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



S1 = 0+2 +3+5+6+7+8+9

S2 = 0+1+2+3+4+7+8+9

S3 = 0+1+3+4+5+6+7+8+9

S4 = 0+2+3+5+6+8+9

S5 = 0+2+6+8

S6 = 0+4+5+6+8+9

S7 = 2+3+4+5+6+8+9

BCD-to-7 segment

B1 B0 \ B3 B2	**00**	**01**	**11**	**10**
00	0	4	x	8
01	1	5	x	9
11	3	7	x	x
10	2	6	x	x

S1 = 0+2 +3+5 +6+7+8+9

S1

B1 B0 \ B3 B2	00	01	11	10
00	1	0	x	1
01	0	1	x	1
11	1	1	x	x
10	1	1	x	x

$$S1 = \boxed{B3} + \overline{B2}\,\overline{B0} + \boxed{B1} + \boxed{B2B1} + \boxed{B2B0}$$

S2 = 0+1+2+3 +4+7+8+9

S2

B1 B0 \ B3 B2	00	01	11	10
00	1	1	x	1
01	1	0	x	1
11	1	1	x	x
10	1	0	x	x

$$S2 = \boxed{B3} + \overline{B2} + B1B0 + \overline{B1}\,\overline{B0}$$

◆ BCD-to-7 segment

B1 B0 \ B3 B2	**00**	**01**	**11**	**10**
00	*0*	*4*	*x*	*8*
01	*1*	*5*	*x*	*9*
11	*3*	*7*	*x*	*x*
10	*2*	*6*	*x*	*x*

S3 = 0+1+3+4+5+6+7+8+9

S3

B1 B0 \ B3 B2	00	01	11	10
00	1	1	x	1
01	1	1	x	1
11	1	1	x	x
10	0	1	x	x

$$S3 = \boxed{B3} + B1B0 + \boxed{\overline{B1}} + \boxed{B2}$$

S4 = 0+2+3+5+6+8+9

S4

B1 B0 \ B3 B2	00	01	11	10
00	1	0	x	1
01	0	1	x	1
11	1	0	x	x
10	1	1	x	x

$$S4 = \boxed{B3} + \overline{B2}\,\overline{B0} + \boxed{\overline{B2}B1} + \boxed{B2\overline{B1}B0} + \boxed{B1\overline{B0}}$$

BCD-to-7 segment

B1 B0 \ B3 B2	**00**	**01**	**11**	**10**
00	*0*	*4*	*x*	*8*
01	*1*	*5*	*x*	*9*
11	*3*	*7*	*x*	*x*
10	*2*	*6*	*x*	*x*

S7 = 2+3+4+5+6+8+9

S7

B1 B0 \ B3 B2	**00**	**01**	**11**	**10**
00	0	1	x	1
01	0	1	x	1
11	1	0	x	x
10	1	1	x	x

$$\mathbf{S7} = \boxed{B3} + \boxed{\overline{B2}B1} + \boxed{\overline{B1}B2} + \boxed{B1\overline{B0}}$$

S5 = 0+2+6+8

S5

B1 B0 \ B3 B2	**00**	**01**	**11**	**10**
00	1	0	x	1
01	0	0	x	0
11	0	0	x	x
10	1	1	x	x

$$\mathbf{S5} = \overline{B2}\,\overline{B0} + \boxed{B1\overline{B0}}$$

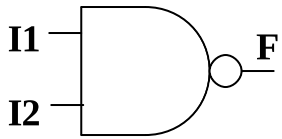
S6 = 0+4+5+6+8+9

S6

B1 B0 \ B3 B2	**00**	**01**	**11**	**10**
00	1	1	x	1
01	0	1	x	1
11	0	0	x	x
10	0	1	x	x

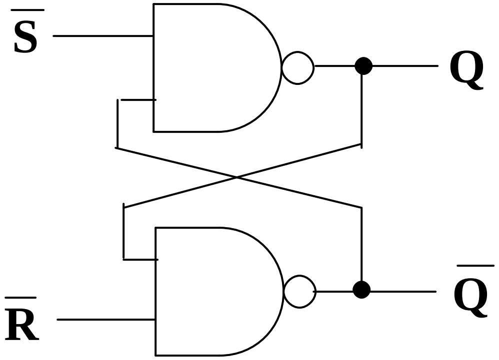
$$\mathbf{S6} = \boxed{B3} + \overline{B1}\,\overline{B0} + \boxed{\overline{B1}B2} + \boxed{B2\overline{B0}}$$

MEMORY ELEMENTS: LATCHES AND FLIP-FLOPS

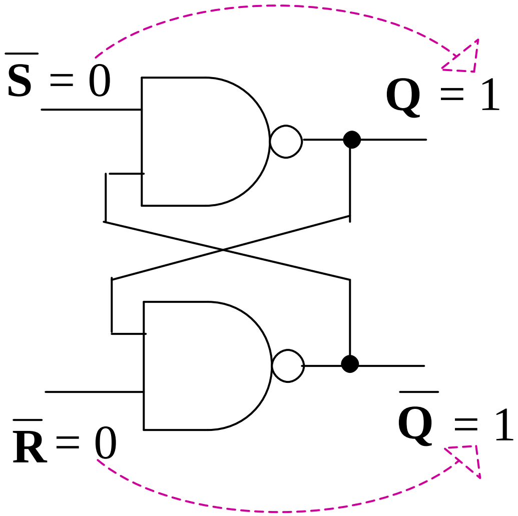


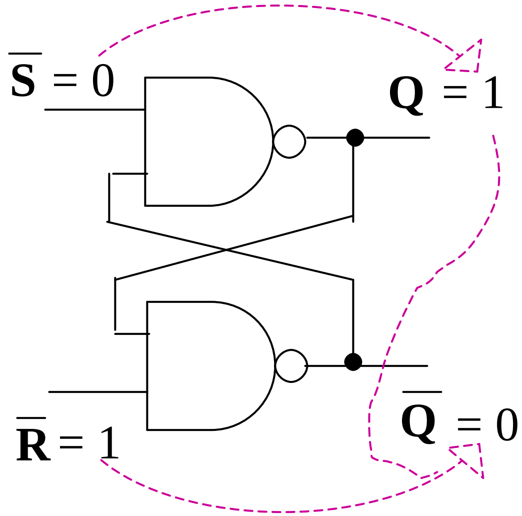
I1	I2	F
0	0	1
0	1	1
1	0	1
1	1	0

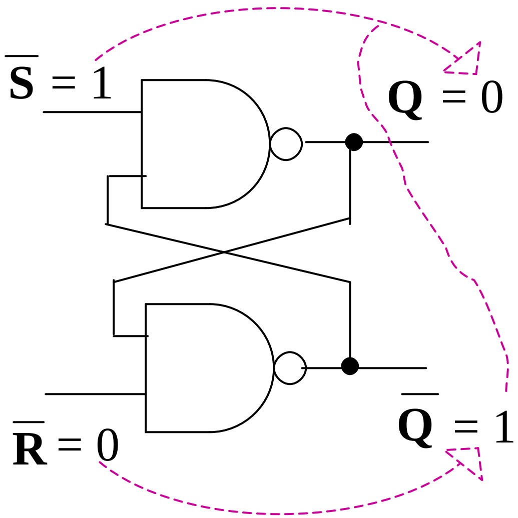
R-S Latch
(Reset-Set)

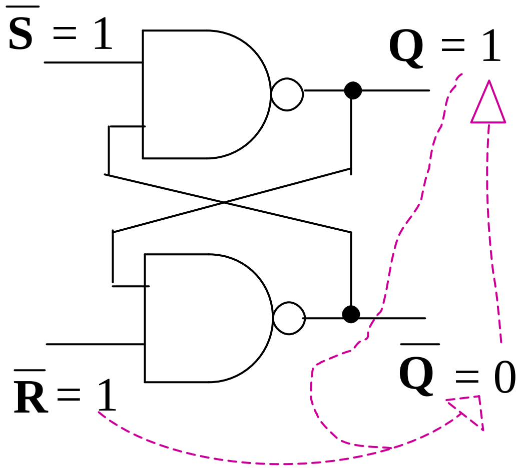


$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	0	1	1	**Weird** state
0	1	1	0	**Set** state
1	0	0	1	**Reset** state
1	1	Q	$\overline{Q}$	**Hold** state

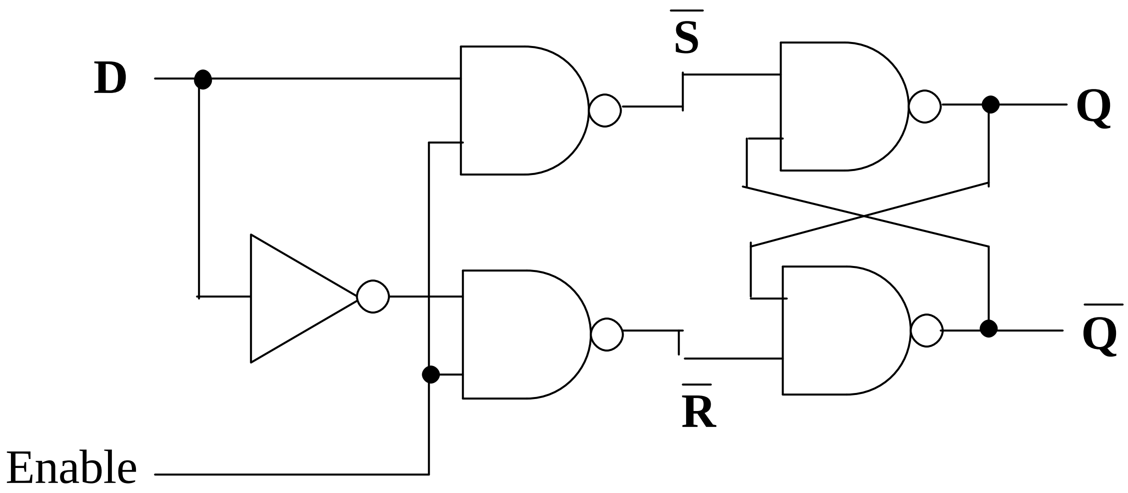








✦ ***D*** (Transparent) **Latch**



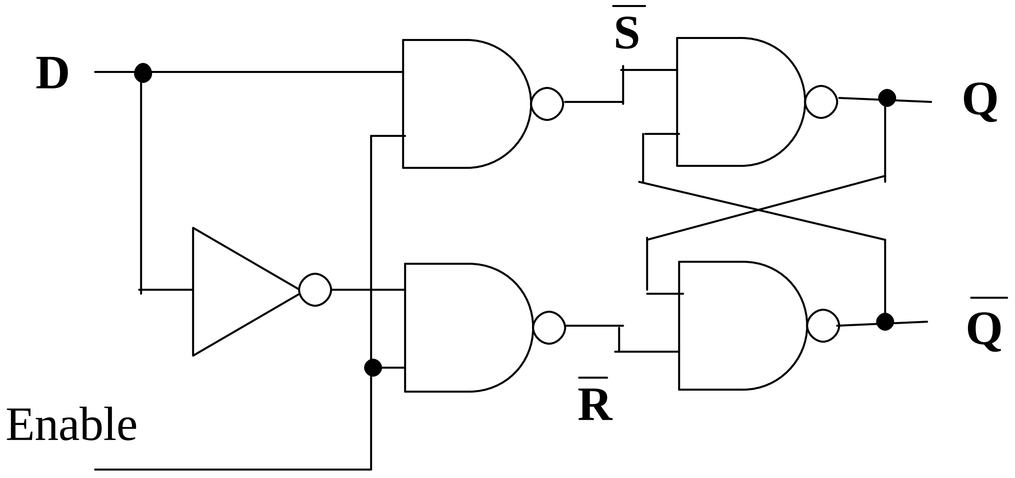
Enable	**D**	**Q**	**$\overline{Q}$**
0	x	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

Enable	**D**	**$\overline{S}$**	**$\overline{R}$**	**Q**	**$\overline{Q}$**
0	0	1	1	Q	$\overline{Q}$
0	1	1	1	Q	$\overline{Q}$
1	0	1	0	0	1
1	1	0	1	1	0

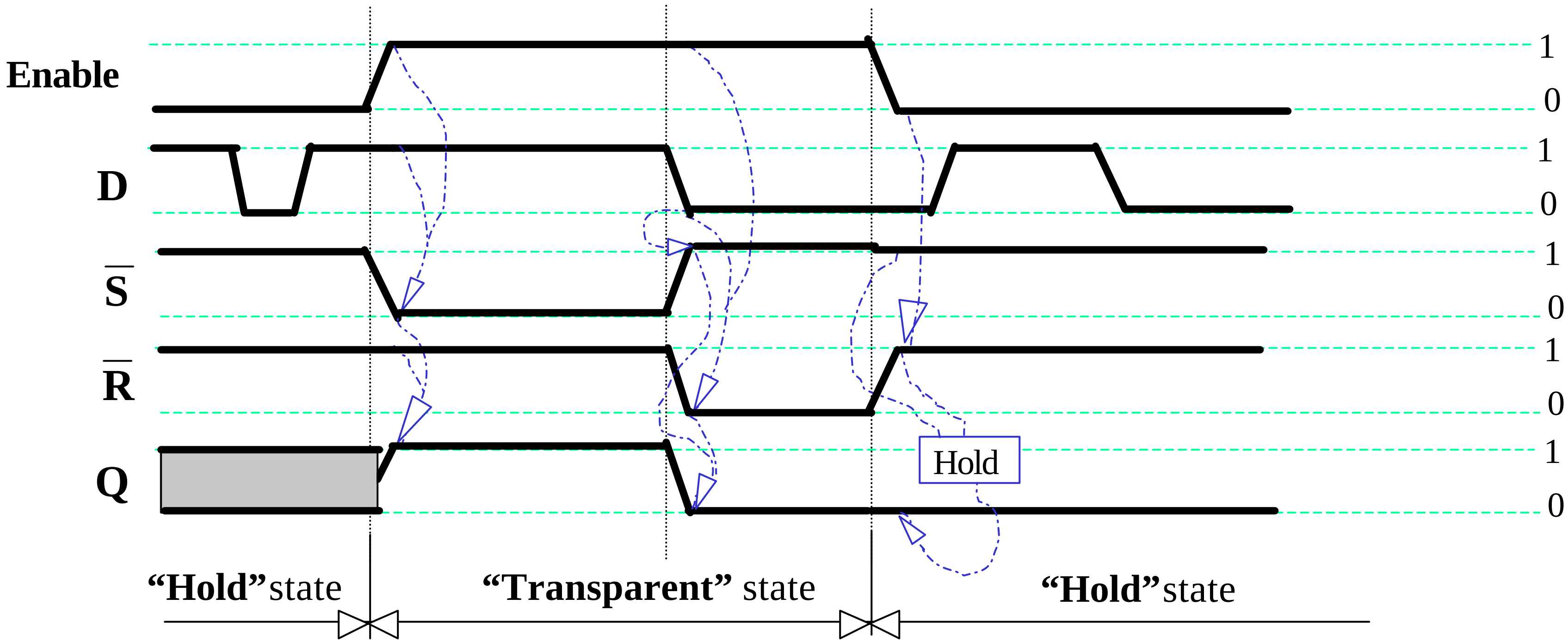
$\overline{S}$	**$\overline{R}$**	**Q**	**$\overline{Q}$**
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Q	$\overline{Q}$

When the ***Enable*** input is =1 (i.e. TRUE or HIGH) the information present at the ***D*** input is stored in the latch and will "appear as it is" at the ***Q*** output (=> it is like that there is a "*transparent*" path from the ***D*** input to the ***Q*** output)

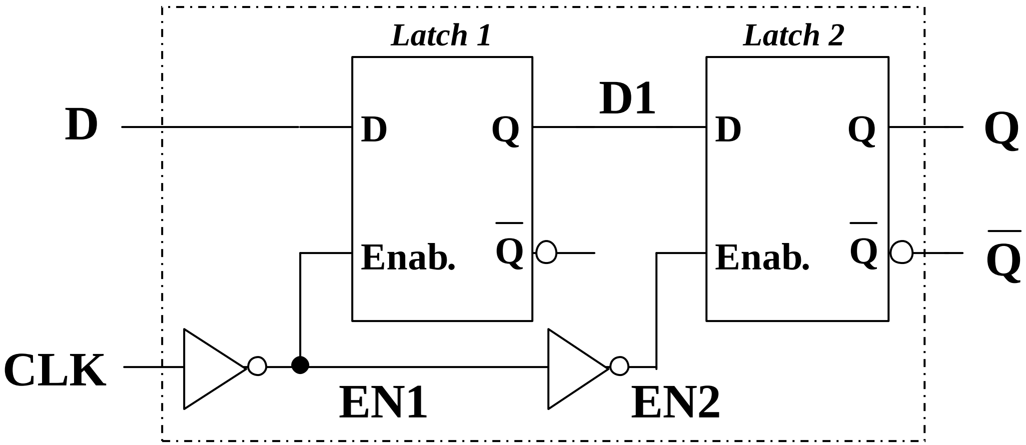
◆ *D* Latch

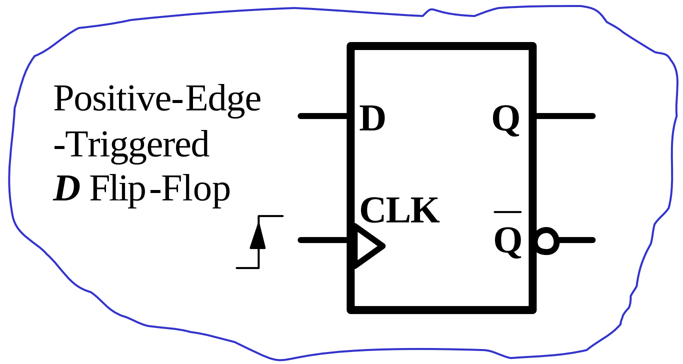


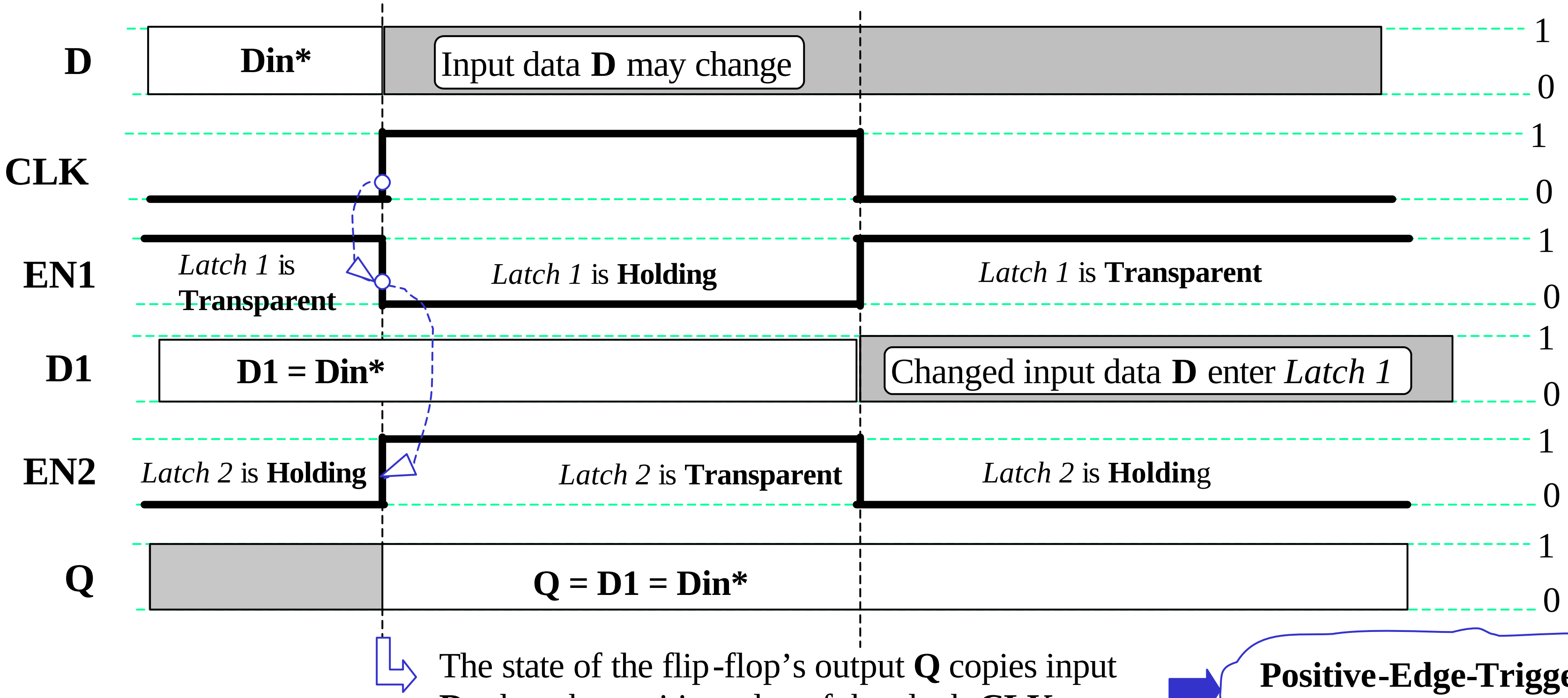
Enable	D	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1	Q	$\overline{Q}$
0	1	1	1	Q	$\overline{Q}$
1	0	1	0	0	1
1	1	0	1	1	0



Synchronous *D* Flip-Flop



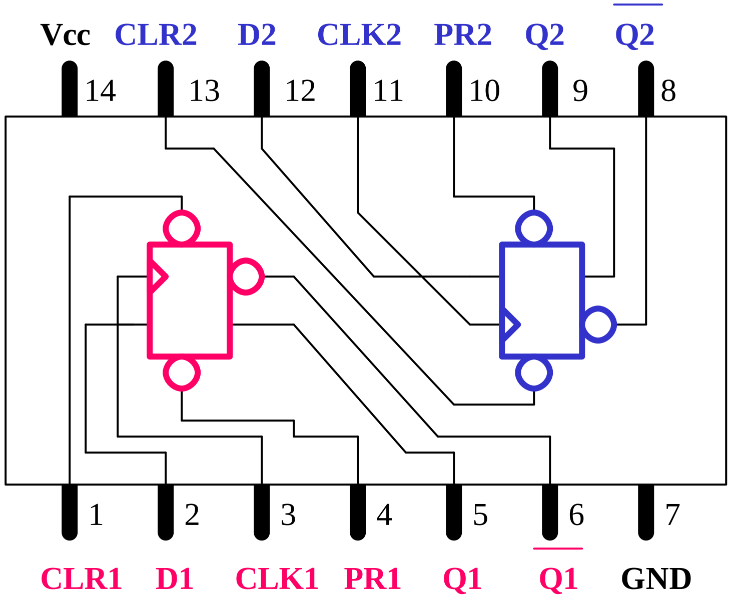




The state of the flip-flop's output **Q** copies input **D** when the positive edge of the clock **CLK** occurs

Positive-Edge-Triggered *D* Flip-Flop

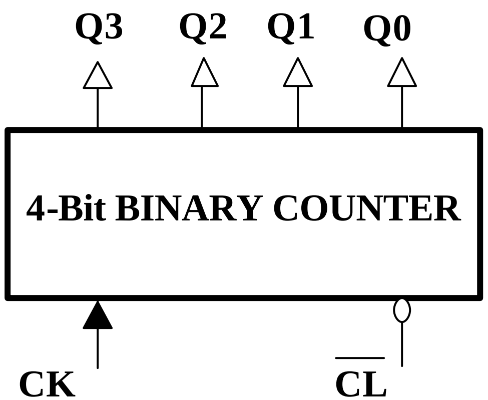
Synchronous *D* Flip-Flop



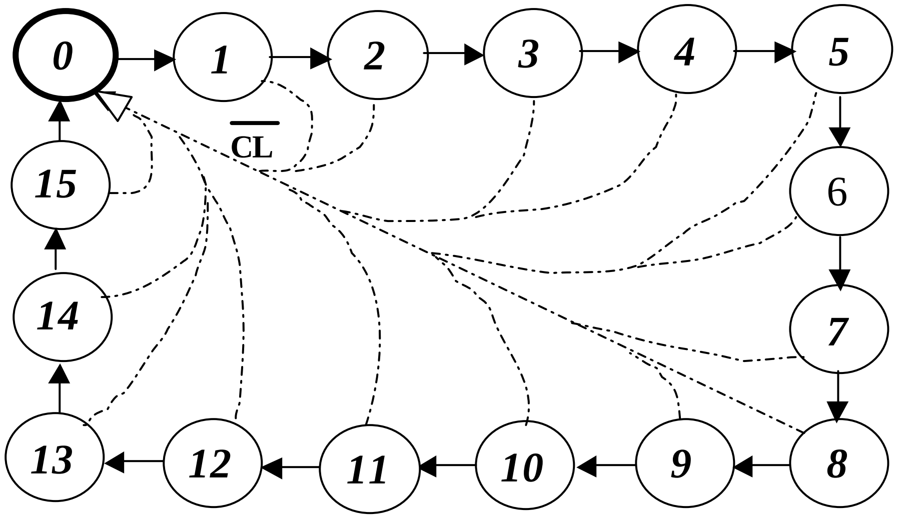
Connection diagram of the ***7474*** **Dual Positive-Edge-Triggered D Flip-Flops** with Preset and Clear.

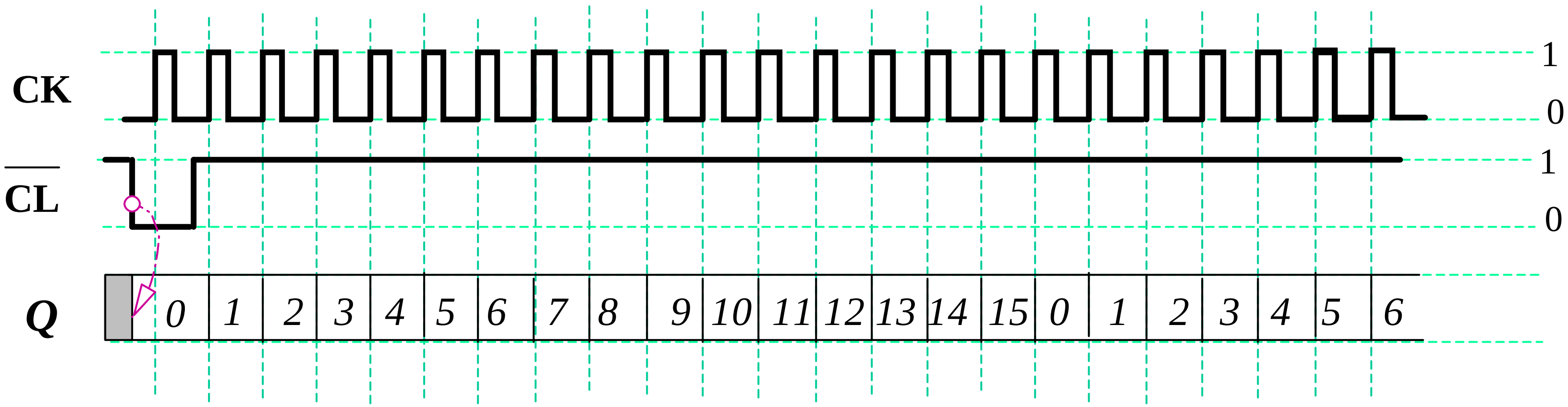
COUNTERS

4-Bit Synchronous Counter using *D* Flip-Flops



$$Q = \sum_{i=0}^{3} Qi \cdot 2i$$





Synchronous 4-bit Counter

DECIMAL STATE Q	BINARY STATE OF THE COUNTER Q3	Q2	Q1	Q0	FLIP FLOP INPUTS (for the next state) D3	D2	D1	D0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	1	1	0	0
12	1	1	0	0	1	1	0	1
13	1	1	0	1	1	1	1	0
14	1	1	1	0	1	1	1	1
15	1	1	1	1	0	0	0	0
0	0	0	0	0				

Using D flip-flops has the distinct advantage of a straightforward definition of the **flip-flop inputs: the current state of these inputs is the next state of the counter**. The logic equations for all four flip-flop inputs **D3, D2, D1**, and **D0** are derived from this truth table as functions of the current states of the counter's flip-flops: **Q3, Q2, Q1**, and **Q0**. Karnaugh maps can be used to simplify these equations.

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

D3

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	0	1
10	0	0	1	1

D2

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	0	1	1	0

D1

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

D0

Q1 Q0 \ Q3 Q2	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

Synchronous 4-bit Counter

D3

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	0	1
10	0	0	1	1

D3 = $Q3 \cdot \overline{Q2} + Q3 \cdot \overline{Q1} + Q3 \cdot \overline{Q0} + \overline{Q3} Q2 \cdot Q1 \cdot Q0$

D2

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	0	1	1	0

D2 = $Q2 \cdot \overline{Q0} + Q2 \cdot \overline{Q1} + \overline{Q2} \cdot Q1 \cdot Q0$

D1

Q1 Q0 \ Q3 Q2	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

D1 = $\overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$

D0

Q1 Q0 \ Q3 Q2	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

D0 = $\overline{Q0}$

Synchronous 4-bit Counter

$\mathbf{D0} = \overline{Q0}$

$\mathbf{D1} = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0}$

$\mathbf{D2} = Q2 \cdot \overline{Q0} + Q2 \cdot \overline{Q1} + \overline{Q2} \cdot Q1 \cdot Q0$

$\mathbf{D3} = Q3 \cdot \overline{Q2} + Q3 \cdot \overline{Q1} + Q3 \cdot \overline{Q0}$

$+ \overline{Q3} \cdot Q2 \cdot Q1 \cdot Q0$