

Module 2

Combinational Logic

LOGIC DESIGN

PURPOSE: Implement Boolean functions by means of circuits.

The smallest building blocks are called **gates**.

CONVENTIONS

Digital circuits operate with two voltage levels of interest:

High
Low

We associate a **value** and a **symbol** to these voltages.

Voltage	Value	Symbol
High	True	1
Low	False	0

We also say that a signal is **asserted** (True value, High voltage) or **deasserted** (False value, Low voltage).

TWO KINDS OF LOGIC CIRCUITS

Combinatorial Circuits: Output = F (Input)

These circuits have no memory: the output lines depend only on the inputs. F is a logic function.

Sequential Circuits: Output = S (Input, State)

The circuits have memory: the output lines depend on the input and on the State which are values stored in memory.

TRUTH TABLES

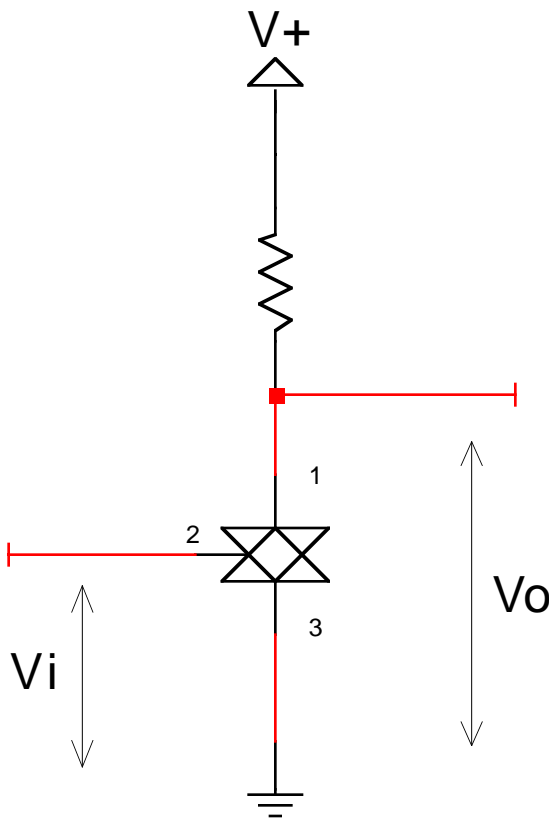
How to describe the values of the Outputs for each possible values of the Inputs? Build a **Truth Table**.

If n is the number of Inputs there are 2^n entries, one for each combination.

EXAMPLE

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

SWITCHING ELEMENTS



Logic circuits are comprised of switching elements. The 3 terminal device shown at left behaves as follows:

-if $V_{in} \geq V_t$, switch closed (terminals 1-3 shorted)

-if $V_{in} < V_t$, switch open (terminals 1-3 open)

The properties of the switch are such that

$$0 \leq V_t \leq V_+$$

Let V_i represent a binary-valued variable such that logical 1 corresponds to V_+ volts and logical 0 corresponds to 0 volts. The behaviour of the circuit can be expressed by the following truth table:

V_{in}	Switch State	V_{out}
V_+	CLOSED	0
0	OPEN	V_+

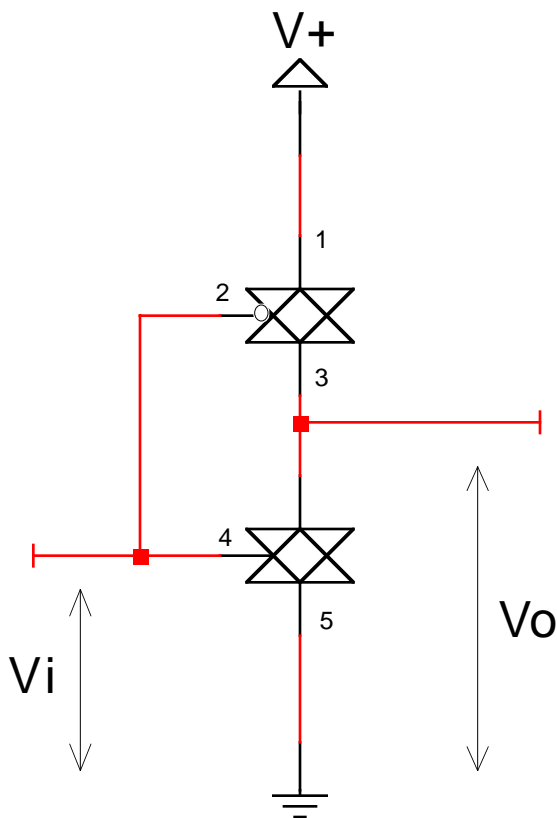
The logical function performed by the circuit is complement, hence it is referred to as an **INVERTER**.

SWITCHING ELEMENTS cont.

In current implementations, switches are fabricated using TRANSISTORS (BJT, MOS). For the purposes of this course we will assume that they behave as ideal switches.

The purpose of the resistor is to PULL UP the output to $V+$ when the switch is open; the switch serves to PULL DOWN the output to 0 when activated.

The configuration shown in the previous slide is referred to as PASSIVE PULL UP / ACTIVE PULL DOWN. It has the advantage of being simple, but takes a lot of space in silicon (resistors have large geometry) and a number of switching limitations (will become clearer in 304-323).



The configuration shown at left gets around some of these problems. Notice the bubble on the upper switch. It operates as follows:

-if $V_i \leq V_t$, 1-3 shorted

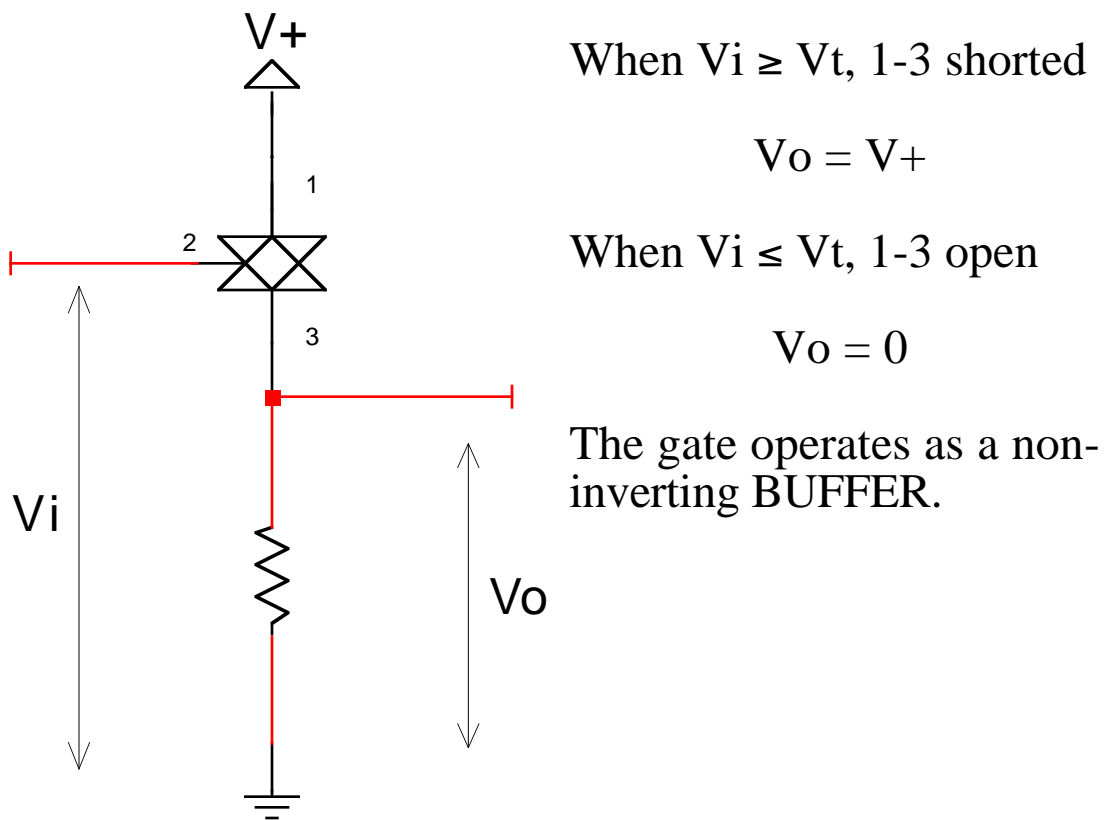
-if $V_i \geq V_t$, 1-3 open

i.e., upper switch closed, bottom switch open and vice-versa.

SWITCHING ELEMENTS cont.

The configuration shown in the previous slide using two switches is referred to as **ACTIVE PULL UP / ACTIVE PULL DOWN**. It is easily verified that it acts as an inverter.

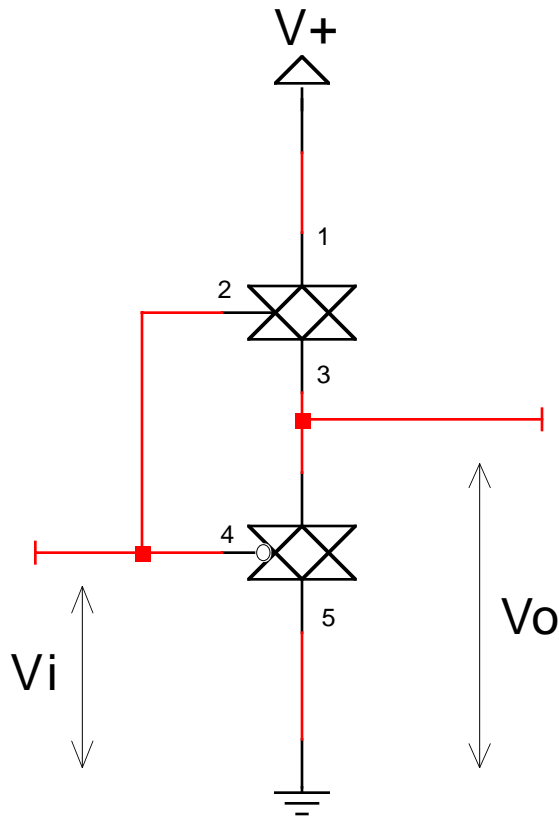
What happens when we swap pull-up and pull-down elements?



The configuration above is referred to as **ACTIVE PULL UP / PASSIVE PULL DOWN**.

SWITCHING ELEMENTS cont.

Analogously to the inverter, we can replace the passive pull-down element with a complementary switching element as follows:



It is easily verified that the circuit acts as a non-inverting BUFFER.

if $V_i \geq V+$, $V_o = V+$

if $V_i \leq V+$, $V_o = 0$

SWITCHING ELEMENTS cont.

We can summarize the behaviour of the circuits encountered thus far using the following table:

Pull-up	Pull-down	Function
passive	active (1)	inverter
active (0)	active (1)	inverter
active (1)	passive	buffer
active (1)	active (0)	buffer

where active (1) --> switch closed if $V_i \geq V_t$,

active (0) --> switch closed if $V_i < V_t$.

V_t is referred to as the *threshold voltage* of the switch and is a property of the particular device characteristics along with the biasing arrangement.

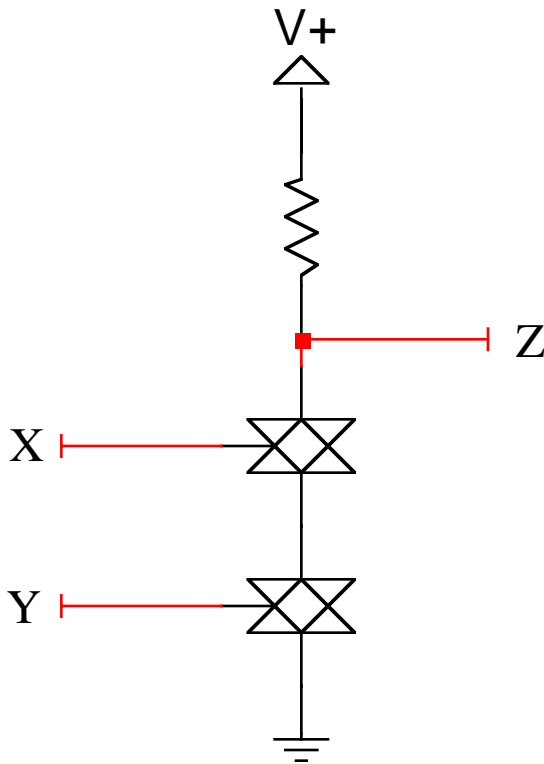
The 4 configurations listed in the table are generally separated into 2 classes, depending on whether a passive element is present or not. This has strong implications on the semiconductor fabrication technology used to implement the circuits.

Within each class there are two configurations, inverters and buffers, depending on whether the output logic level is the input level or its complement.

More complex gates are built from these basic structures.

ELEMENTS IN SERIES

Let's consider what happens when we connect two inverters in series as shown below. At the right of the circuit is a truth table describing the behaviour of the circuit.



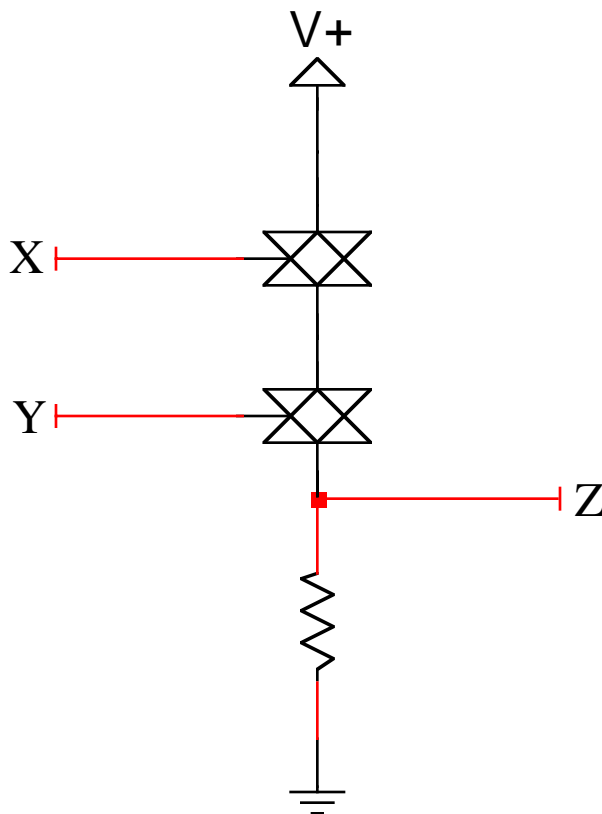
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

The above table is easily verified by noting that the only condition under which Z can be 0 is for BOTH switches to be active, i.e. X=1 and Y=1.

The circuit corresponds to a 2-input NAND gate. Larger NAND gates can be fabricated by placing additional active (1) switches in series (up to the practical limit determined by electronic circuit considerations).

ELEMENTS IN SERIES cont.

Recall that an inverter is turned into a non-inverting buffer by transposing the passive pull-up with the active (1) pull-down. This suggests a way of converting a NAND gate into a AND gate.



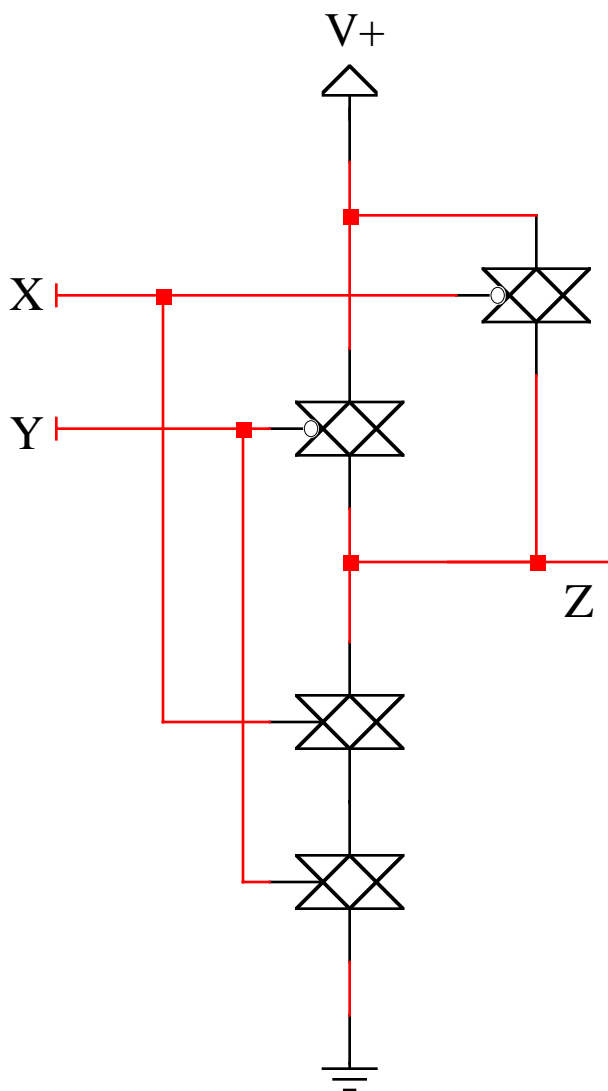
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

The above truth table is easily verified. By default, $Z = 0$ if either switch is off. Hence the only way in which $Z = 1$ is if **BOTH** X and Y are set to 1.

This confirms our supposition that transposing pull-up and pull-down elements negates the initial function. As with the NAND gate shown previously, multiple inputs are accommodated by placing additional switch elements in series.

ELEMENTS IN SERIES cont.

The case for series connection of elements with an active pull-up or pull-down is slightly more complex since a switch can have only a single control input.



As before, we can determine the function of the circuit at left by exhaustively exploring its truth table:

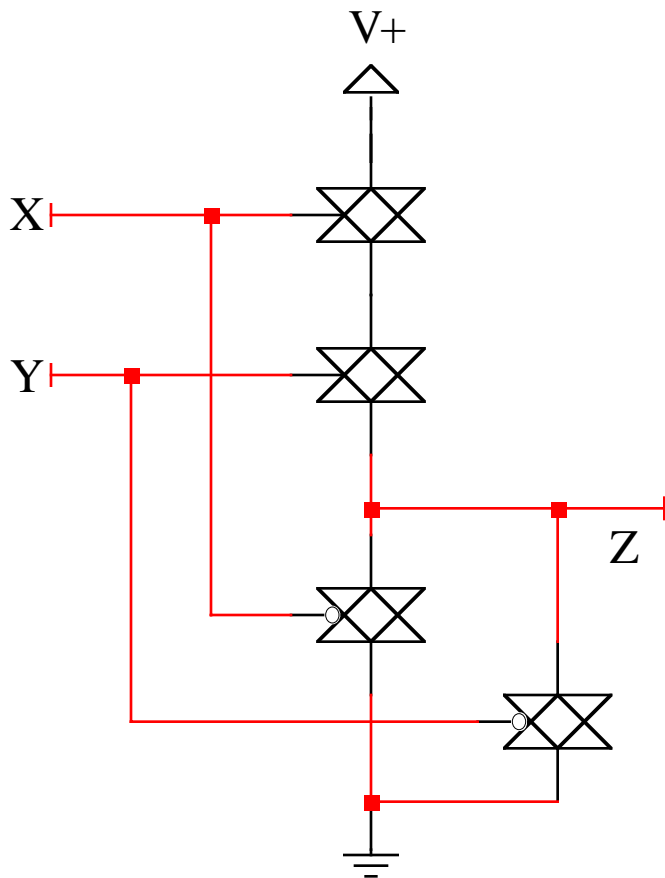
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

If either X or Y is 0, one of the two series elements is open, and one of the parallel elements closed so that $Z = 1$. The only case in which $Z = 0$ is if both X and Y are 1.

The circuit corresponds to a NAND gate with the additional features of smaller geometry (i.e. Size) and better switching characteristics.

ELEMENTS IN SERIES cont.

We would expect that swapping the pull-up and pull-down structures should result in an AND gate.



We can test this hypothesis by verifying the truth table corresponding to the circuit.

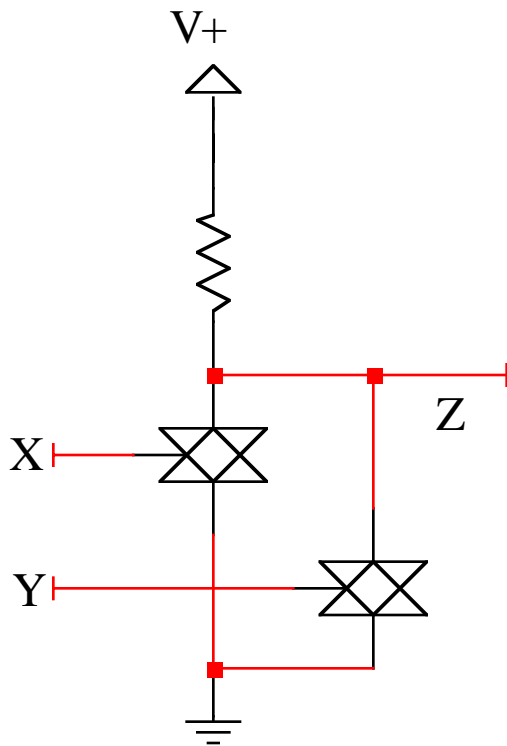
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

A 0 at X or Y opens the path to V+ and shorts the path to ground. Only X and Y both 1 will set Z to 1.

Hence the circuit corresponds to an AND gate. To increase the number of inputs, additional pull-ups are connected in series and pull-downs in parallel.

ELEMENTS IN PARALLEL

We now repeat our investigation for switching elements connected in parallel. Consider what happens when two inverters are connected in parallel as shown below.



The truth table is determined (as before) by evaluating the circuit output for each input combination.

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

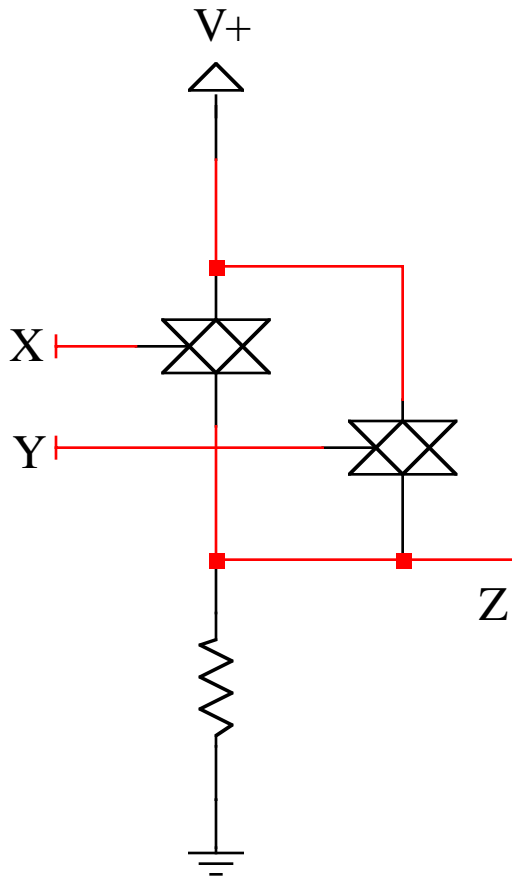
The only possible way for Z to be 1 is for both switches to be off.

It is easily verified from the truth table that the circuit corresponds to a NOR gate.

This is not entirely unexpected and is related to the principle of DUALITY which we will study shortly.

ELEMENTS IN PARALLEL cont.

Again, swapping the pull-up and pull-down elements should change the sense of the gate, i.e., from NOR to OR.



Evaluating the circuit for each possible input combination yields the following truth table.

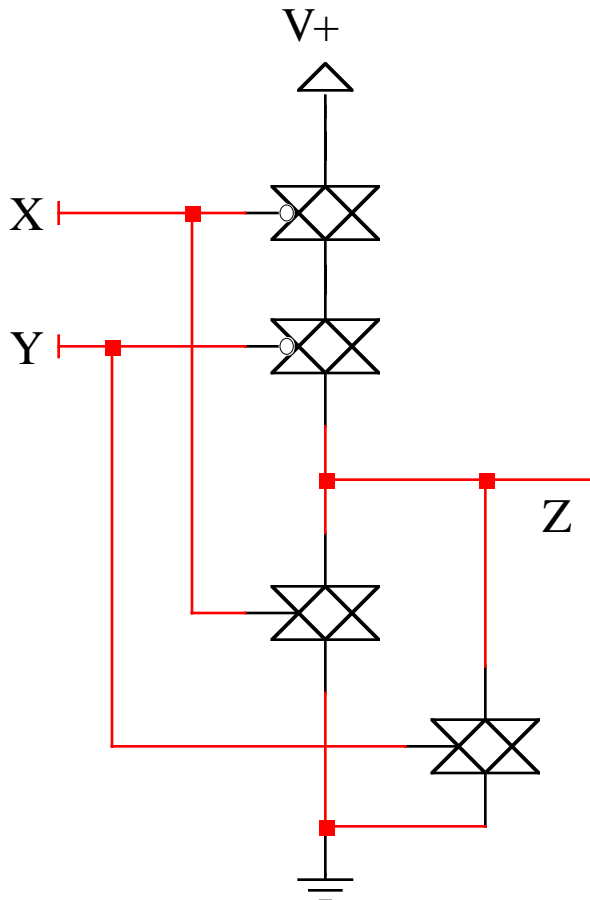
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

The only possible case for a 0 output is for both switches to be off. Otherwise the output is pulled high if either or both switches are on.

The circuit corresponds to an OR gate as we suspected.

ELEMENTS IN PARALLEL cont.

Parallel connection of elements with an active pull-up or pull down involves replacing the passive element with active (0) switching elements in series. First, consider the case where the active (0) elements are on top.



The truth table corresponding to this circuit is determined by evaluating the response to each of the 4 possible input combinations.

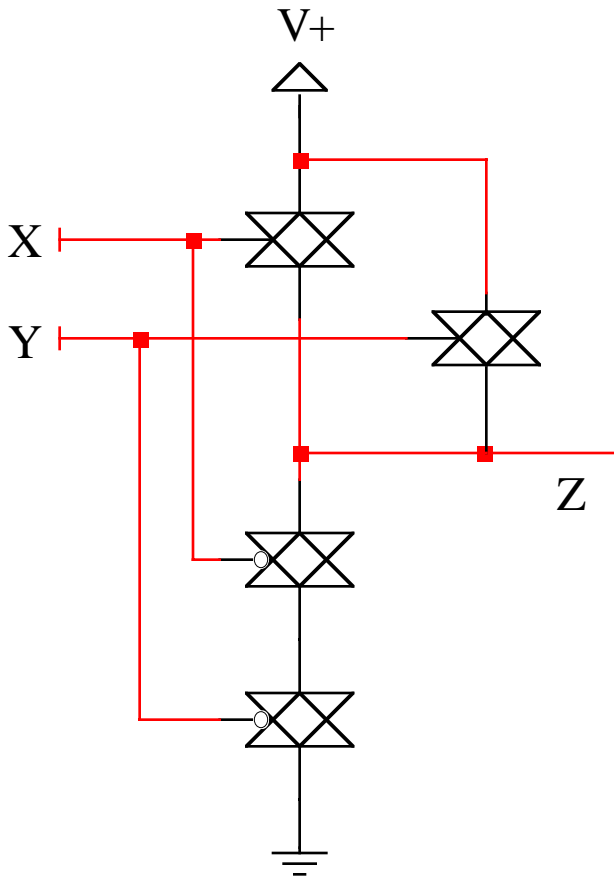
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

A 1 on either X or Y open circuits the path to V+ and short circuits the path to ground.

A 0 on both X and Y open circuits both of the lower switches and short circuits the path to V+. From the truth table it can be seen that the circuit corresponds to a NOR gate.

ELEMENTS IN PARALLEL cont.

Finally, to obtain a positive OR gate using both active pull-up and pull-down elements, we swap the upper and lower elements as follows.



Using the same procedure as before, we obtain the following truth table for the circuit.

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

A 1 on X or Y open circuits the bottom leg and closes one of the two switches in parallel, resulting

in an output of 1. The only combination that can result in a 0 output is for both inputs to be 0, hence the circuit is a positive OR gate.

BOOLEAN ALGEBRA

Logic functions, which we have represented thus far using truth tables, can also be expressed in terms of logic equations. This gives rise to Boolean Algebra (after the mathematician, 1815-1864). One can take two views:

To a mathematician: *A ring with multiplicative identity in which every element is an idempotent.* ($a \text{ times } a = a$)

To a computer engineer: *A notation to describe logical relationships between two-valued variables.* ($c = f(a,b)$)

All variables have the values 0 or 1. There are three operators defined as follows:

A	B	NOT A	NOT B	A+B	A•B
0	0	1	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	1	1

The three operators are also referred to as

A + B	LOGICAL SUM
A • B	LOGICAL PRODUCT
NOT	LOGICAL NEGATION

Boolean algebra is governed by rules and axioms which enable logic equations to be manipulated.

PROPERTIES & IDENTITIES

Properties of 0 and 1.

- Identity law: $A + 0 = A$ and $A \cdot 1 = A$

Proof by enumeration:

If $A = 0$, $0 + 0 = 0$. Now if $A = 1$, $1 + 0 = 1$

If $A = 0$, $0 \cdot 1 = 0$. Now if $A = 1$, $1 \cdot 1 = 1$

- Zero and One laws: $A + 1 = 1$ and $A \cdot 0 = 0$

Proof by enumeration:

If $A = 0$, $0 + 1 = 1$. Now if $A = 1$, $1 + 1 = 1$.

If $A = 0$, $0 \cdot 0 = 0$. Now if $A = 1$, $1 \cdot 0 = 0$.

- Inverse laws: $\overline{A} + A = 1$ and $A \cdot \overline{A} = 0$

- Idempotence: $A + A = A$ and $A \cdot A = A$

- Commutative laws: $A + B = B + A$ and $A \cdot B = B \cdot A$

PROPERTIES & IDENTITIES: cont.

- Associative Laws: $A + (B + C) = (A + B) + C$
 $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Distributive Laws: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
 $A + (B \cdot C) = (A + B) \cdot (A + C)$
- Absorption Laws: $A + (A \cdot B) = A$
 $A \cdot (A + B) = A$
- DeMorgan's Laws: $\overline{A + B} = \overline{A} \cdot \overline{B}$
 $\overline{A \cdot B} = \overline{A} + \overline{B}$

Note: All the above properties come in pairs. Each is formally obtained from its dual by:

1. Exchanging the operators \cdot and $+$
2. Exchanging the constants 0 and 1

How this duality property can be proved?

CANONICAL FORMS

Two fundamental algebraic forms to describe a truth table:

- MINTERM or SUM OF PRODUCTS form,
- MAXTERM or PRODUCT OF SUMS form.

Consider the truth table for a full-adder:

	Cin	Ai	Bi	Sum	Cout
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

A minterm is a CONJUNCTION over the input variables ([Cin,Ai,Bi] in this case) that is true whenever the corresponding output is true.

For example, notice that Sum is true when [Cin,Ai,Bi]=[0,0,1]. The corresponding logical expression would then be

$$\overline{\text{Cin}} \ \overline{\text{Ai}} \ \text{Bi} .$$

CANONICAL FORMS cont.

Using this approach we can completely specify the logical functions for Sum and Cout by taking the logical sum of each minterm as follows:

$$\begin{aligned}\text{Sum} &= \overline{C_{in}} \overline{A_i} B_i + \overline{C_{in}} A_i \overline{B_i} + C_{in} \overline{A_i} \overline{B_i} + C_{in} A_i B_i , \\ \text{Cout} &= \overline{C_{in}} A_i B_i + C_{in} \overline{A_i} B_i + C_{in} A_i \overline{B_i} + C_{in} A_i B_i .\end{aligned}$$

This form is referred to as a SUM OF PRODUCTS, often abbreviated as $\Sigma\Pi$.

A maxterm is a DISJUNCTION over the input variables that is false whenever the corresponding output is false.

Again, notice that Sum is false when $[C_{in}, A_i, B_i] = [0, 0, 0]$. The corresponding logical expression would be $C_{in} + A_i + B_i$. Using this approach we can write expressions for Sum and Cout by taking the logical product of each maxterm as follows:

$$\begin{aligned}\text{Sum} &= (C_i + A_i + B_i)(C_{in} + \overline{A_i} + \overline{B_i})(\overline{C_{in}} + A_i + \overline{B_i})(\overline{C_{in}} + \overline{A_i} + B_i) \\ \text{Cout} &= (C_{in} + A_i + B_i)(C_{in} + A_i + \overline{B_i})(C_{in} + \overline{A_i} + B_i)(\overline{C_{in}} + A_i + B_i).\end{aligned}$$

This is referred to as a PRODUCT OF SUMS, $\Pi\Sigma$.

If the SUM OF PRODUCTS and PRODUCT OF SUMS forms both describe the same function, then we should be able to prove this algebraically.

CANONICAL FORMS cont.

Using the expression for Sum as an example, start by writing an expression for its complement as sum of products:

$$\overline{\text{Sum}} = \overline{\text{Cin}} \overline{\text{Ai}} \overline{\text{Bi}} + \overline{\text{Cin}} \text{Ai} \text{Bi} + \text{Cin} \overline{\text{Ai}} \text{Bi} + \text{Cin} \text{Ai} \overline{\text{Bi}}$$

Now apply de Morgan's law to the right hand side:

$$\overline{\text{Sum}} = \overline{\text{Cin} + \text{Ai} + \text{Bi}} + \overline{\text{Cin} + \overline{\text{Ai}} + \overline{\text{Bi}}} + \overline{\overline{\text{Cin}} + \text{Ai} + \text{Bi}} + \overline{\overline{\text{Cin}} + \overline{\text{Ai}} + \text{Bi}}$$

Apply de Morgan one more time:

$$\text{Sum} = (\text{Cin} + \text{Ai} + \text{Bi})(\text{Cin} + \overline{\text{Ai}} + \overline{\text{Bi}})(\overline{\text{Cin}} + \text{Ai} + \overline{\text{Bi}})(\overline{\text{Cin}} + \overline{\text{Ai}} + \text{Bi})$$

Which is exactly what is required.

So, given a truth table, we now have a mechanism for expressing it as a logical function in one of two canonical forms.

But what about its implementation as an electronic circuit?

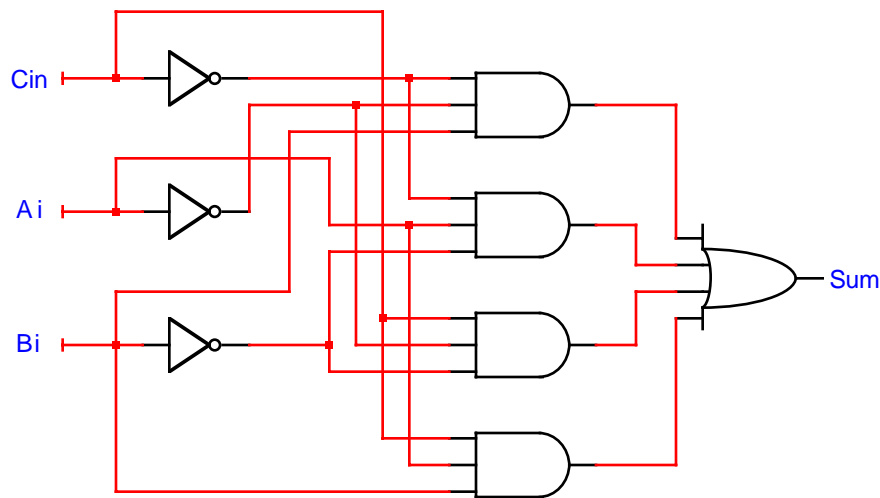
CANONICAL FORMS TO LOGIC CIRCUITS

Given that we can transform a truth table into a logic equation, how do we transform the equation into an equivalent logic circuit?

First, consider the expression we obtained earlier for the sum function expressed as a Sum of Products:

$$\text{Sum} = \overline{\text{Cin}} \overline{\text{Ai}} \text{Bi} + \overline{\text{Cin}} \text{Ai} \overline{\text{Bi}} + \text{Cin} \overline{\text{Ai}} \overline{\text{Bi}} + \text{Cin} \text{Ai} \text{Bi}$$

Each product corresponds to an AND gate, and the sum which combines them together to an OR gate. A direct circuit implementation might look as follows.



Notice how the expression maps **DIRECTLY** to the logic circuit implementation.

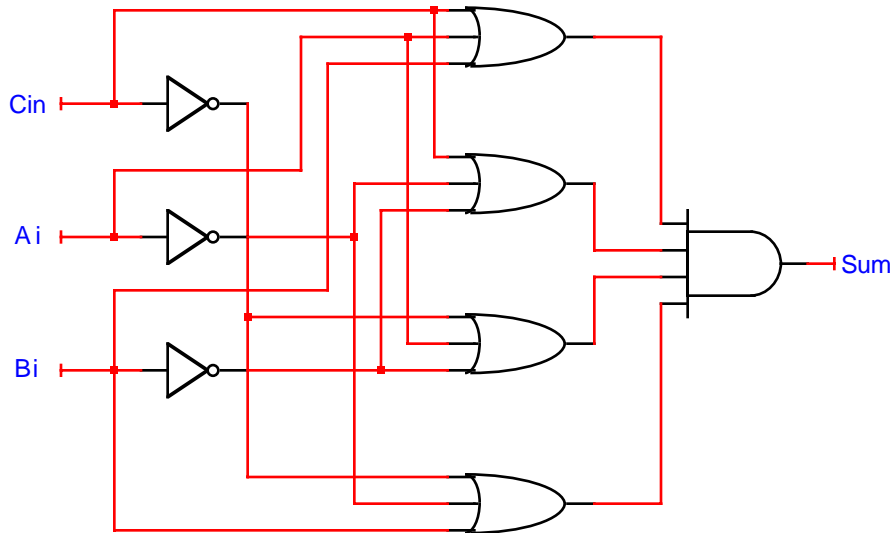
Observe: the complexity of the resulting circuit is directly related to the complexity of the expression.

CANONICAL FORMS TO LOGIC CIRCUITS cont.

Next, consider the Product of Sums expression we obtained earlier:

$$\text{Sum} = (\text{Cin} + \text{Ai} + \text{Bi})(\text{Cin} + \overline{\text{Ai}} + \overline{\text{Bi}})(\overline{\text{Cin}} + \text{Ai} + \overline{\text{Bi}})(\overline{\text{Cin}} + \overline{\text{Ai}} + \text{Bi})$$

A direct implementation of this expression results in the following circuit:



Again, the complexity of the resulting circuit is directly related to the complexity of the product-of-sums expression.

If we could somehow MINIMIZE these expressions, the result would be a SIMPLER CIRCUIT.

This process is referred to as LOGIC MINIMIZATION.

TRANSFORMATION OF FORMS

In earlier notes we observed how logic gates could be negated by swapping pull-up and pull-down elements.

While this is possible in theory, technological constraints often make this impossible in practice. One common family of logic, TRANSISTOR-TRANSISTOR LOGIC (TTL), is based entirely on inverter structures. AND and OR gates are fabricated by cascading NAND and NOR gates with inverters. This adds additional switching delay, so circuits composed of NAND and NOR gates are preferred.

We can achieve a suitable transformation by applying De Morgan's law to our canonical forms.

Example:

$$\text{Sum} = \overline{C_{in}} \overline{A_i} \overline{B_i} + \overline{C_{in}} \overline{A_i} B_i + C_{in} \overline{A_i} \overline{B_i} + C_{in} A_i B_i$$

1. Negate terms
2. Change + 's to • 's
3. Negate expression

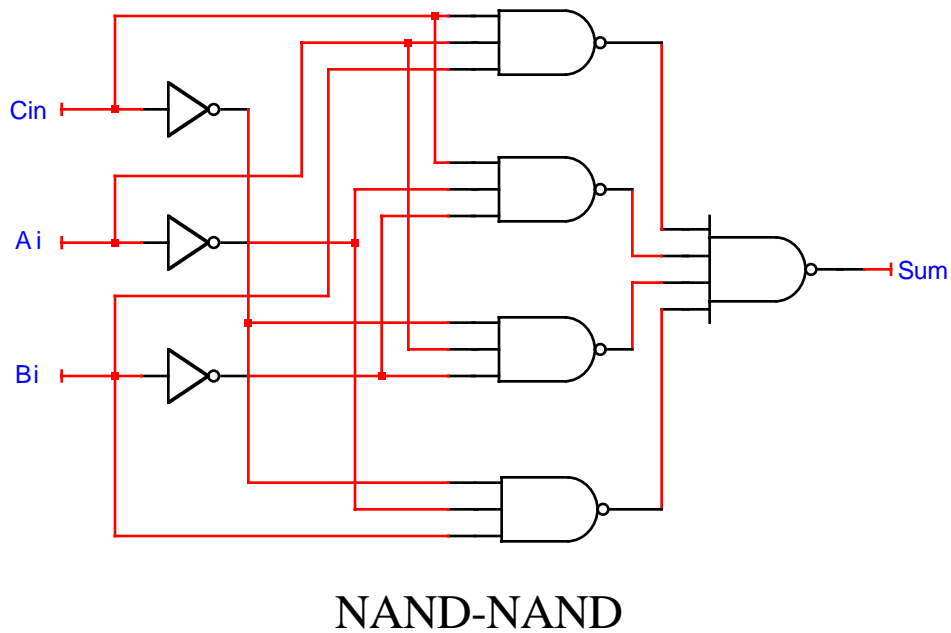
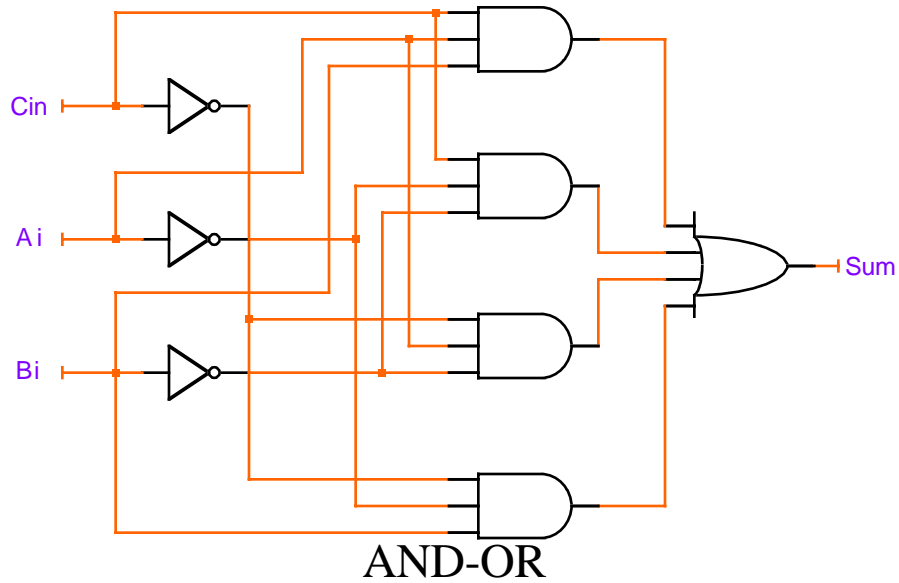
$$\text{Sum} = \overline{\overline{C_{in}} \overline{A_i} \overline{B_i} \cdot \overline{C_{in}} \overline{A_i} B_i \cdot C_{in} \overline{A_i} \overline{B_i} \cdot C_{in} A_i B_i}$$

The resulting expression contains 4 NAND terms combined together in a NAND.

We refer to this as NAND-NAND logic.

TRANSFORMATION OF FORMS: cont.

Compare the AND-OR and NAND-NAND circuits:



The NAND-NAND has about half the propagation delay of the AND-OR (excluding the inverters) for TTL.

TRANSFORMATION OF FORMS: cont.

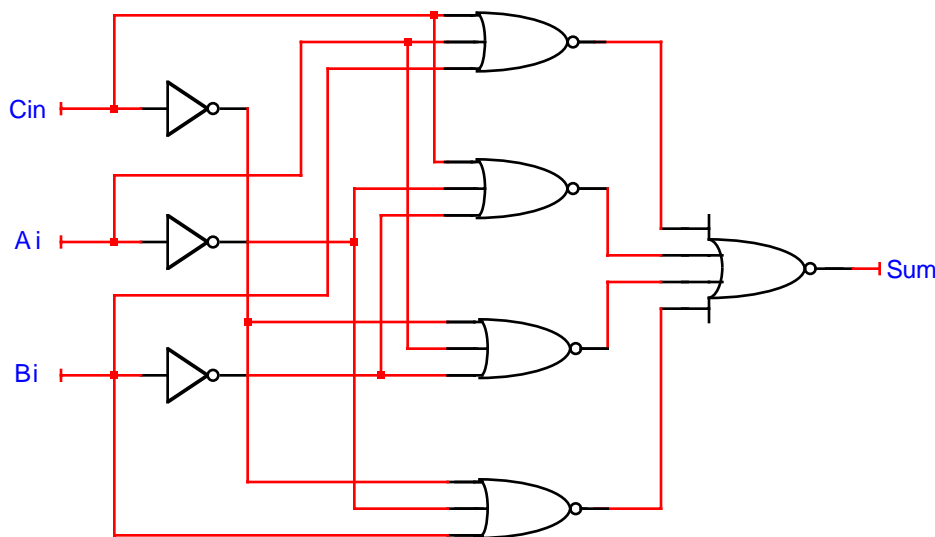
We can apply De Morgan's law to the $\prod\Sigma$ form as well:

$$\text{Sum} = (\text{Cin} + \text{Ai} + \text{Bi})(\text{Cin} + \overline{\text{Ai}} + \overline{\text{Bi}})(\overline{\text{Cin}} + \text{Ai} + \overline{\text{Bi}})(\overline{\text{Cin}} + \overline{\text{Ai}} + \text{Bi})$$

Applying De Morgan we obtain:

$$\text{Sum} = \left[\overline{(\text{Cin} + \text{Ai} + \text{Bi})} + \overline{(\text{Cin} + \overline{\text{Ai}} + \overline{\text{Bi}})} + \overline{(\overline{\text{Cin}} + \text{Ai} + \overline{\text{Bi}})} + \overline{(\overline{\text{Cin}} + \overline{\text{Ai}} + \text{Bi})} \right]'$$

The resulting expression contains 4 NOR gates combined together in a NOR with the following equivalent circuit.



NOR-NOR

As with NAND-NAND, propagation delay is reduced.

LOGIC MINIMIZATION

The simplification of logic equations to minimal form is referred to as LOGIC MINIMIZATION. This has a direct impact on the complexity of the resulting circuit implementation.

Consider the following truth table

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

The sum of products corresponding to this table is

$$X = \overline{A} \overline{B} \overline{C} \overline{D} + \overline{A} \overline{B} C D + \overline{A} B \overline{C} \overline{D} + A \overline{B} \overline{C} \overline{D} \\ + A \overline{B} C \overline{D} + A \overline{B} C D + A B \overline{C} \overline{D} + A B C \overline{D}$$

LOGIC MINIMIZATION: cont.

Algebraic minimization strategy: factor into terms that differ in 1 variable. Repeat this procedure until no further simplification is possible.

$$1. X = \overline{A} \overline{C} \overline{D} (\overline{B} + B) + A \overline{C} \overline{D} (B + \overline{B}) + \overline{B} CD(\overline{A} + A) + AC \overline{D} (B + \overline{B})$$

$$X = \overline{A} \overline{C} \overline{D} + A \overline{C} \overline{D} + \overline{B} CD + AC \overline{D}$$

Observe that the second term may be combined with either the first or last term. Idempotence says that any of the terms in the above expression may be replicated without changing anything, so...

$$2. X = \overline{A} \overline{C} \overline{D} + A \overline{C} \overline{D} + \overline{B} CD + AC \overline{D} + A \overline{C} \overline{D} \\ = \overline{C} \overline{D} (\overline{A} + A) + \overline{B} CD + A \overline{D} (C + \overline{C}) \\ = \overline{C} \overline{D} + \overline{B} CD + A \overline{D}$$

Now instead of requiring 8 4-input AND gates and 1 8-input OR gate, the circuit has been reduced to a total of 4 gates: 1 3-input OR, 1 3-input AND, and 2 2-input ANDs (neglecting the inverters again).

These manipulations are not always obvious! For this reason a convenient graphical method has been developed - Karnaugh Maps.

KARNAUGH MAPS

Another more compact way of representing the truth table is to use a KARNAUGH MAP.

X		CD			
		00	01	11	10
AB	00	1	0	1	0
	01	1	0	0	0
	11	1	0	0	1
	10	1	0	1	1

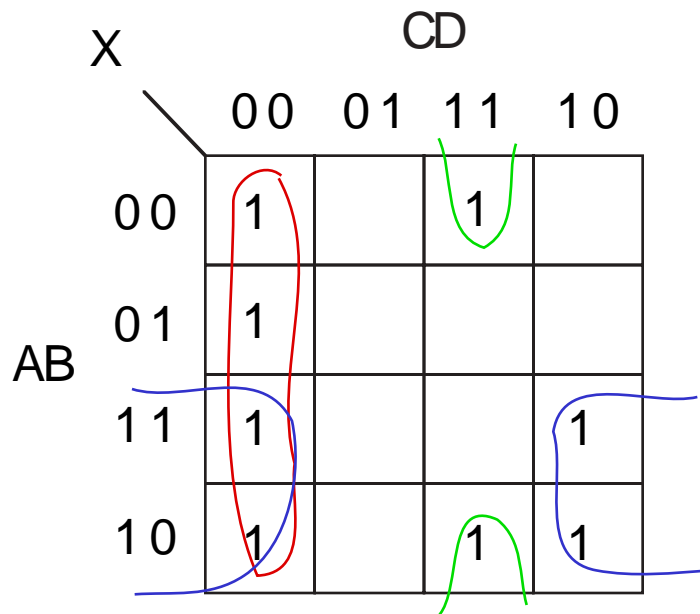
Notice how each truth table entry is represented by a corresponding cell in the Karnaugh map.

Also pay close attention to the ordering! Each cell differs in exactly 1 variable when moving in a vertical or horizontal direction.

The map also wraps around the edges (bottom row to top row, leftmost column to rightmost column).

Let's return to the minimization example shown earlier and examine its relationship to the map shown above.

KARNAUGH MAPS: cont.



Notice the 3 groups circled in different colors. What is their significance?

Answer: they are comprised of minterms that DIFFER ONLY IN A SINGLE VARIABLE.

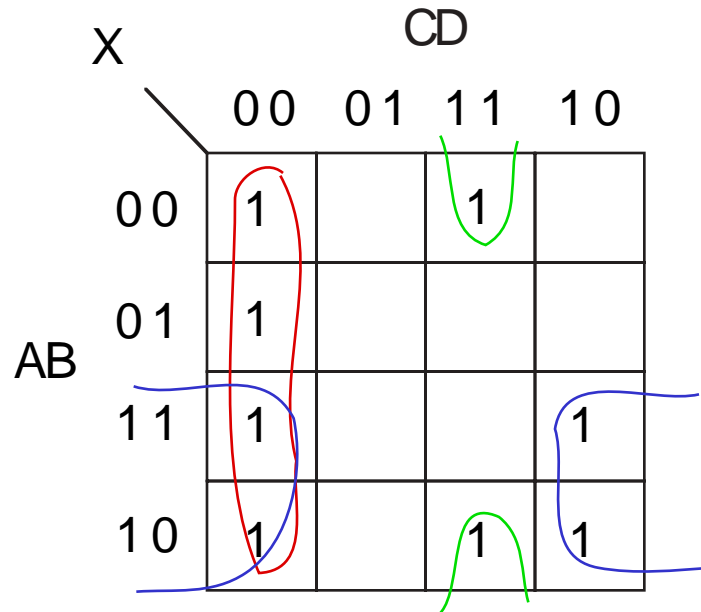
Consider the group circled in red: 0000 0100 1100 1000.

In going from top to bottom (left to right above), each successive minterm differs by only a single bit.

The same is true for the blue group: 1100 1000 1010 1110

And also for the green: 0011 1011

KARNAUGH MAPS: cont.



The terms circled in red (first column) correspond to the following minterm expression:

$$\overline{A} \overline{B} \overline{C} \overline{D} + \overline{A} \overline{B} \overline{C} D + A \overline{B} \overline{C} \overline{D} + A \overline{B} \overline{C} D$$

We already know that this reduces to $\overline{C} \overline{D}$ - the map simply makes this relationship explicit.

Notice that the $\overline{C} \overline{D}$ term is common to each cell and that A and B each take on values of 0 and 1.

KARNAUGH MAPS: cont.

In general, for an n-bit minterm:

- A group of 2 minterms can be combined if they have n-1 variables in common - 1 variable is eliminated.
- A group of 4 minterms can be combined if they have n-2 variables in common - 2 variables are eliminated.
- A group of 8 variables can be combined if they have n-3 variables in common - 3 variables are eliminated and so on. So far we've identified 1 group of 4 ones. Are there any others? Yes, the terms circled in blue corresponding to the following minterm expression:

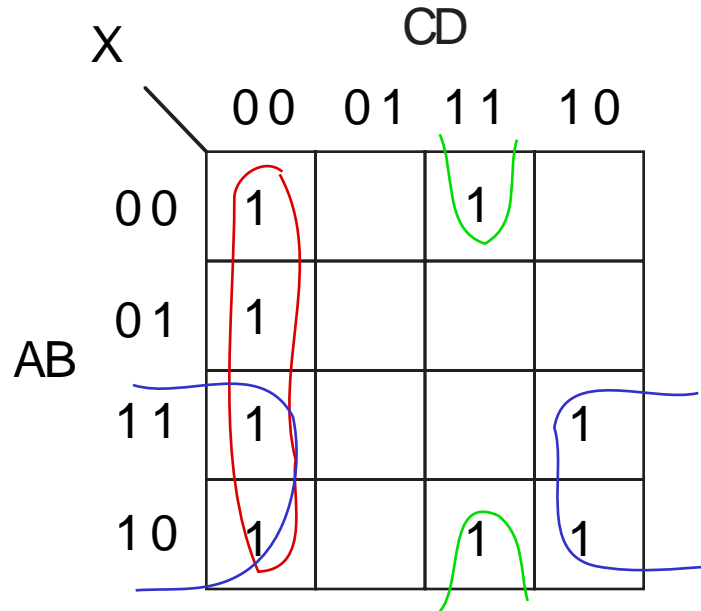
$$AB\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$

Notice that the first two terms also appear in the previous expression. That's OK, terms can be replicated as needed (idempotence).

Again from the earlier algebraic manipulation we know that this reduces to $A\overline{D}$. This can be observed directly from the Karnaugh map - each term has $A\overline{D}$ in common with B and C taking on 0 and 1.

KARNAUGH MAPS: cont.

To complete the minimization procedure we have to account for each 1 that appears in the map. At this point the terms $\overline{A} \overline{B} CD$ and $A \overline{B} CD$ are left. Notice that they differ only in A - the green circle shown on the map - which reduces to $\overline{B} CD$.



Combining the results we obtain the following minimal expression for X :

$$X = \overline{C} \overline{D} + \overline{B} CD + A \overline{D} .$$

KARNAUGH MAPS: cont.

As you might suspect (principal of duality) the Karnaugh map probably has something to do with maxterms as well.

		CD			
		00	01	11	10
AB	00		0		0
	01		0	0	0
	11		0	0	
	10		0		

First consider the terms circled in blue (the column under 01). The maxterm expression is

$$(A + B + C + \bar{D}) (A + \bar{B} + C + \bar{D}) (\bar{A} + \bar{B} + C + \bar{D}) (\bar{A} + B + C + \bar{D})$$

We would probably suspect that variables A and B can be eliminated, i.e., that the minimal expression corresponding to this product of sums is $(C + \bar{D})$.

KARNAUGH MAPS: cont.

There's one way to find out - simplify the expression algebraically. Here's a really neat trick.

Take the dual form, i.e.,

$$\begin{aligned} & \text{Dual} \{ (A + B + C + \overline{D}) (A + \overline{B} + C + \overline{D}) \\ & \quad (\overline{A} + \overline{B} + C + \overline{D}) (\overline{A} + B + C + \overline{D}) \} \\ &= (ABC \overline{D}) + (A \overline{B} C \overline{D}) + (\overline{A} \overline{B} C \overline{D}) + (\overline{A} B C \overline{D}) \\ &= (AC \overline{D})(B + \overline{B}) + (\overline{A} C \overline{D})(\overline{B} + B) \\ &= (AC \overline{D}) + (\overline{A} C \overline{D}) \\ &= (C \overline{D})(A + \overline{A}) = C \overline{D} \end{aligned}$$

This gives us another expression that we can simplify much more easily. IT IS NOT EQUIVALENT.

To convert back we need to take the dual form of the result.

$$\text{Dual} \{ C \overline{D} \} = (C + \overline{D})$$

KARNAUGH MAPS: cont.

X		CD			
		00	01	11	10
AB	00		0		0
	01		0	0	0
	11		0	0	
	10		0		

So the same properties of variable elimination also hold for maxterms. Let's complete the procedure by enumerating the remaining 0's.

For the term circled in green (center of the map)

$$(\overline{B} + \overline{D})$$

and the remaining term circled in red

$$(A + \overline{C} + D)$$

KARNAUGH MAPS: cont.

		CD			
		00	01	11	10
AB	00		0		0
	01		0	0	0
	11		0	0	
	10		0		

Combining all the maxterms, the minimal expression for X is

$$\Pi\Sigma = (C + \overline{D}) (\overline{B} + \overline{D}) (A + \overline{C} + D)$$

For comparison, the minterm expression for X is

$$\Sigma\Pi = \overline{C} \overline{D} + \overline{B} CD + A \overline{D}.$$

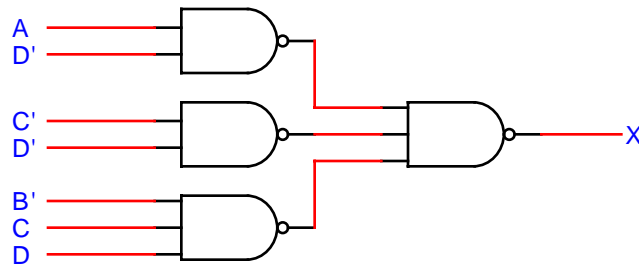
KARNAUGH MAPS: cont.

Circuit Implementations

We can use de Morgan's law to convert each of the canonical forms into NAND-NAND and NOR-NOR representations respectively.

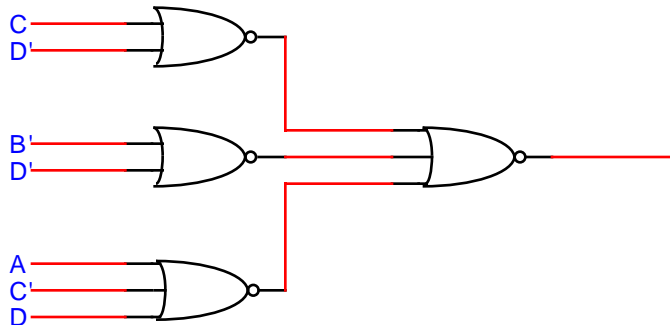
$$\begin{aligned}\Sigma\Pi &= \overline{C} \overline{D} + \overline{B} CD + A \overline{D} \\ &= \overline{\overline{\overline{C} \overline{D}} \overline{\overline{\overline{B} CD} \overline{\overline{A} \overline{D}}}}\end{aligned}$$

NAND-NAND



$$\begin{aligned}\Pi\Sigma &= (C + \overline{D}) (\overline{B} + \overline{D}) (A + \overline{C} + D) \\ &= \overline{\overline{(C + D)} + \overline{(B + D)} + \overline{(A + \overline{C} + D)}}\end{aligned}$$

NOR-NOR



DON'T CARES

Consider the truth table shown below:

C	B	A	X
0	0	0	d
0	0	1	0
0	1	0	1
0	1	1	d
1	0	0	1
1	0	1	0
1	1	0	d
1	1	1	0

The d's in the output correspond to "don't cares", i.e., values for which the output is permitted to be either 0 or 1.

Since they can be taken either way, they are usually selected so that the simplest function is obtained.

Consider the Karnaugh map for this truth table.

		BA			
		00	01	11	10
C	0	d	0	d	1
	1	1	0	0	d

Notice how the don't cares are used to obtain a minimal expression.


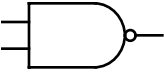


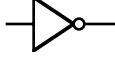


The d's corresponding to minterms 000 & 110 are treated as 1's, and the remaining minterm as 0.

This corresponds to $X = \overline{A}$, instead of $X = \overline{C} B \overline{A} + C \overline{B} \overline{A}$

COMMON BUILDING BLOCKS

Combinational logic is often defined in terms of standard building blocks ranging from simple gates to complex functions such as ALU's (arithmetic and logical units).

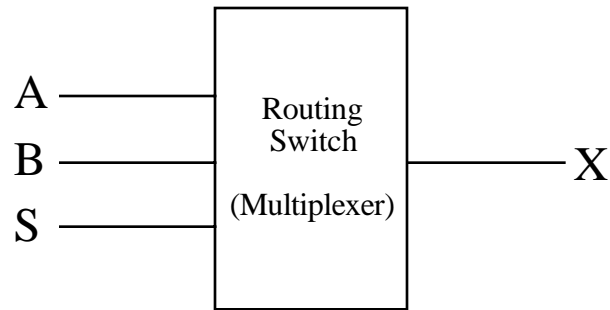
The Basic Gates:

• AND Gate	$A \cdot B$	
• NAND Gate	$\overline{A \cdot B}$	
• OR Gate	$A + B$	
• NOR Gate	$\overline{A + B}$	
• NOT Gate	\overline{A}	
• XOR Gate	$A \overline{B} + \overline{A} B$	
• XNOR Gate	$AB + \overline{A} \overline{B}$	

COMMON BUILDING BLOCKS: cont.

The Multiplexer is a routing switch that selects 1 of n lines for output depending on the state of a set of control inputs. The case for n=2 is shown below.

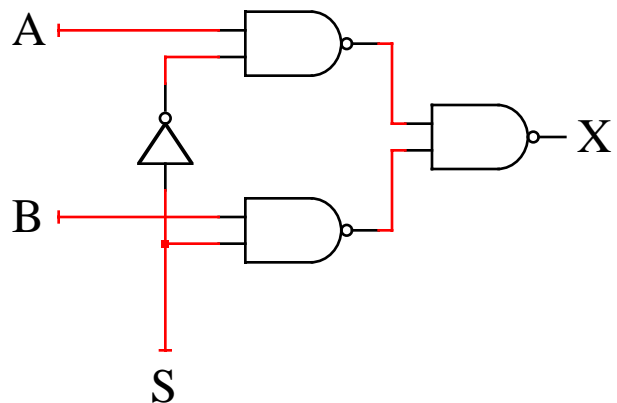
S	A	B	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



We can synthesize this circuit using formal methods (e.g. Karnaugh Map), but the logic equation is simple enough to infer from inspection:

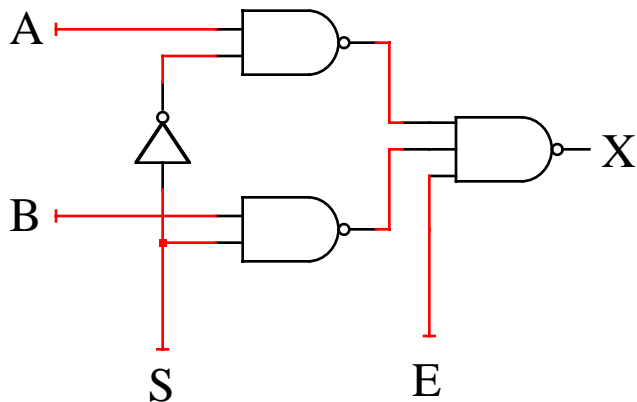
$$X = \bar{S} A + S B$$

		AB			
		00	01	11	10
S	0	0	0	1	1
	1	0	1	1	0



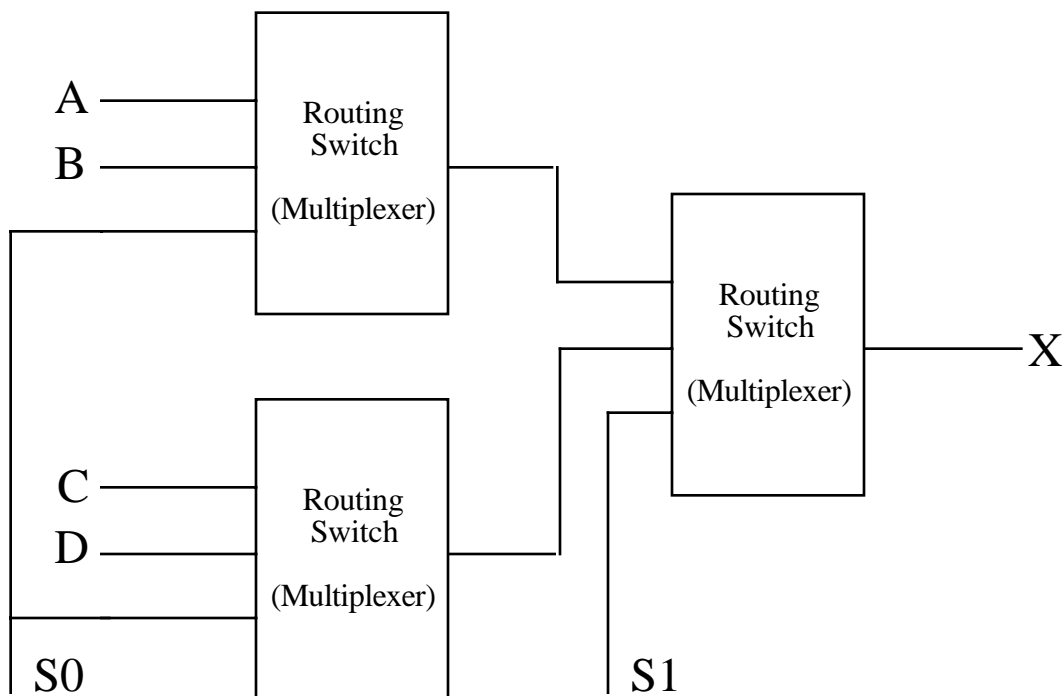
COMMON BUILDING BLOCKS: cont.

Sometimes an additional control line is added to the basic multiplexer as follows:



The enable (E) line is used to synchronize changes at X with an external timing signal such as a clock line.

Larger multiplexers can be formed by cascading smaller ones as shown below (e.g. binary trees).



COMMON BUILDING BLOCKS: cont.

Propagation Delay:

Is defined as the interval between the application of the input signal and the point at which the output produces a stable response.

Consider the case of the 2-input multiplexer. The propagation delay is exactly 2 gate delays (assuming S is stable).

The 4-input multiplexer, made from two 2-input multiplexers, adds an additional 2 units of delay.

One can fabricate an 8-input multiplexer, by combining two 4-input multiplexers using a 2-input multiplexers.

The structure corresponds to a BINARY TREE, where the depth of the tree is given as:

$$\text{Depth} = \text{Log}_2(\# \text{ Inputs})$$

Each level of the tree corresponds to the propagation delay of a 2-input multiplexer, i.e., 2 gate delays.

Hence, a 256-input multiplexer built in this fashion would have a propagation delay of 16 gate delays.

COMMON BUILDING BLOCKS: cont.

2'S COMPLEMENT OVERFLOW DEECTOR

Recall that overflow in 2's complement addition can be determined from the sign (msb) bits of the two arguments and the resulting sum.

Ah	Bh	Σ h	OF
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

		$B\Sigma$			
		00	01	11	10
A	0		1		
	1				1

		$B\Sigma$			
		00	01	11	10
A	0	0		0	0
	1	0	0	0	

The resulting minterm and maxterm expressions are, respectively:

$$\begin{aligned}
 \text{OF} &= \overline{A} \overline{B} \Sigma + AB \overline{\Sigma} \\
 &= (B + \Sigma) (\overline{A} + \overline{\Sigma}) (A + \overline{B}) \quad \begin{array}{l} \text{NAND-NAND} \\ \text{NOR-NOR} \end{array}
 \end{aligned}$$

COMMON BUILDING BLOCKS: cont.

DECODERS

As the name implies, a decoder takes an n-bit input and produces 2^n outputs, i.e., a single output line is true for each input combination. The table shown below corresponds to a 3-bit decoder.

Inputs			Outputs							
S2	S1	S0	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

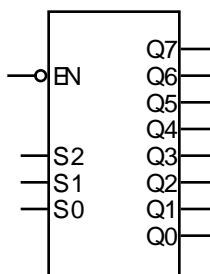


table shown

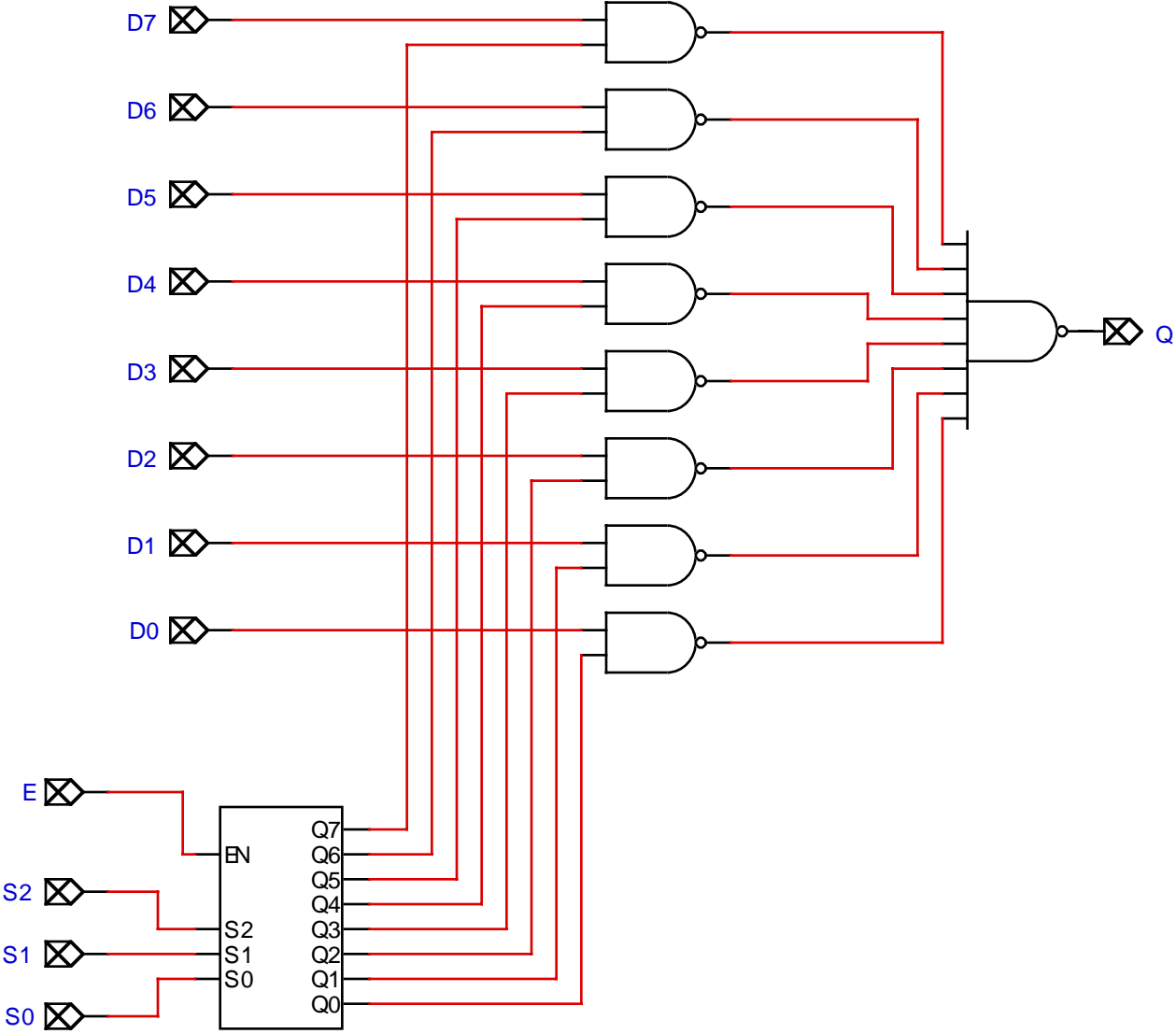
The circuit shown at left corresponds to the truth table shown above with one minor exception - the enable input.

When enable is not asserted, i.e., logic 1 in this case (notice the inversion bubble), outputs are all 0. Otherwise the decoder functions according to the truth

above.

COMMON BUILDING BLOCKS: cont.

A Better Multiplexer



COMMON BUILDING BLOCKS: cont.

ENCODERS

The encoder performs the opposite function of the decoder, it takes n inputs and produces m outputs, where $n = 2^m$.

Inputs							Outputs		
I7	I6	I5	I4	I3	I2	I1	Q2	Q1	Q0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	1
0	0	0	0	0	1	0	0	1	0
0	0	0	0	1	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	1
0	1	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1	1	1

Notice that the truth table does NOT contain 2^n rows!
The implicit assumption is that only 1 input can be asserted at any time. Hence the 3 logic equations corresponding to Q2, Q1, and Q0 are as follows:

$$Q2 = I4 + I5 + I6 + I7$$

$$Q1 = I2 + I3 + I6 + I7$$

$$Q0 = I1 + I3 + I5 + I7$$

COMMON BUILDING BLOCKS: cont.

COMPARATORS

A single-bit comparator takes 2 inputs A and B and asserts one of three outputs depending on whether $A > B$, $A = B$, or $A < B$, i.e.,

$G = 1$ if and only $A > B$

$E = 1$ if and only $A = B$

$L = 1$ if and only $A < B$

Truth table:

Inputs		Outputs		
A	B	G	E	L
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

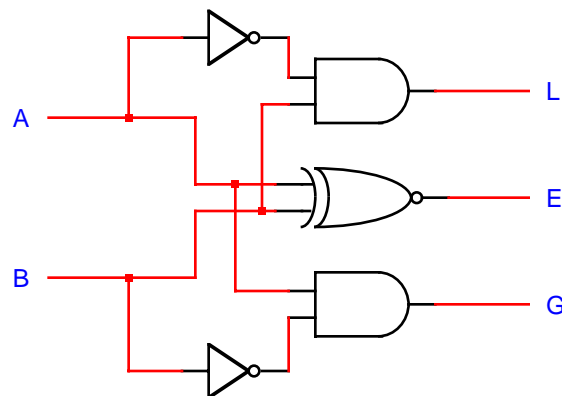
Logic equations

$$G = A \bar{A} \bar{B}$$

$$L = \bar{A} A B$$

$$E = \bar{A} \bar{A} \bar{B} + A A B$$

Can be implemented with an AND gate, two inverters and an XNOR gate.



COMMON BUILDING BLOCKS: cont.

FULL ADDER

C _i	A _i	B _i	Σ	C _o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The truth table at left corresponds to a FULL ADDER.

Using standard techniques, we can derive a logic function describing this table in either of the two canonical forms.

Notice that the expression for Σ does not simplify in either canonical form.

		A _i B _i			
		00	01	11	10
Σ	0		①		①
	1	①		①	

		A _i B _i			
		00	01	11	10
Σ	0	①		①	
	1		①		①

$$\begin{aligned} \Sigma &= \overline{C_i} \overline{A_i} \overline{B_i} + C_i A_i B_i + \overline{C_i} \overline{A_i} B_i + \overline{C_i} A_i \overline{B_i} \\ &= (C_i + A_i + B_i)(C_i + \overline{A_i} + \overline{B_i})(\overline{C_i} + A_i + \overline{B_i})(\overline{C_i} + \overline{A_i} + B_i) \end{aligned}$$

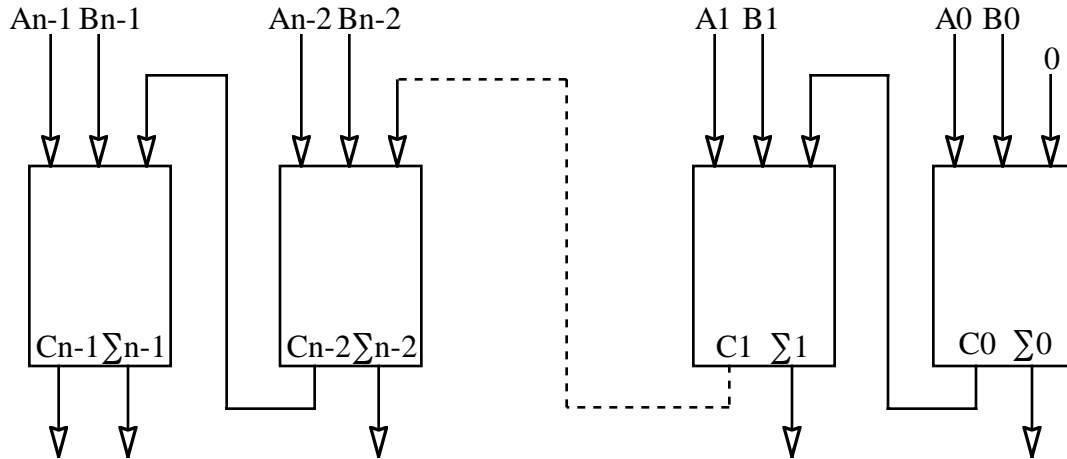
		A _i B _i			
		00	01	11	10
C _o	0			①	
	1		①	①	①

		A _i B _i			
		00	01	11	10
C _o	0	①	①		①
	1	①			

$$\begin{aligned} C_o &= A_i B_i + C_i B_i + C_i A_i \\ &= (A_i + B_i)(C_i + A_i)(C_i + B_i) \end{aligned}$$

COMMON BUILDING BLOCKS: cont.

Word Width Full Adders



An n -bit full-adder can be fabricated by cascading together n full-adder modules as shown above.

What are the PROPAGATION DELAYS corresponding to Σ and C_o ?

- 3 gate delays for Σ (negating inputs adds 1 delay)
- 2 gate delays for C_o (there are no inverted inputs)

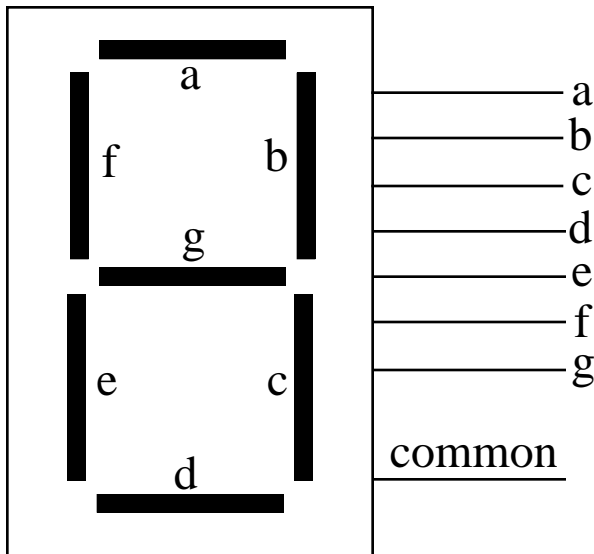
What is the propagation delay for the n -bit adder?

- $(n-1) \times 2$ gate delays to propagate carry to the last full adder in the chain +
- 3 gate delays to produce the final sum

Total = $2n + 1$ gate delays

EXAMPLE: 7 SEGMENT DISPLAY

These are common in counters, calculators, digital watches, computer consoles, etc.



Principle: Seven light emitting diodes (lamps) seperately fed, with common return lead.

By lighting up different combinations of the 7 segments, numerals and some letters can be produced.

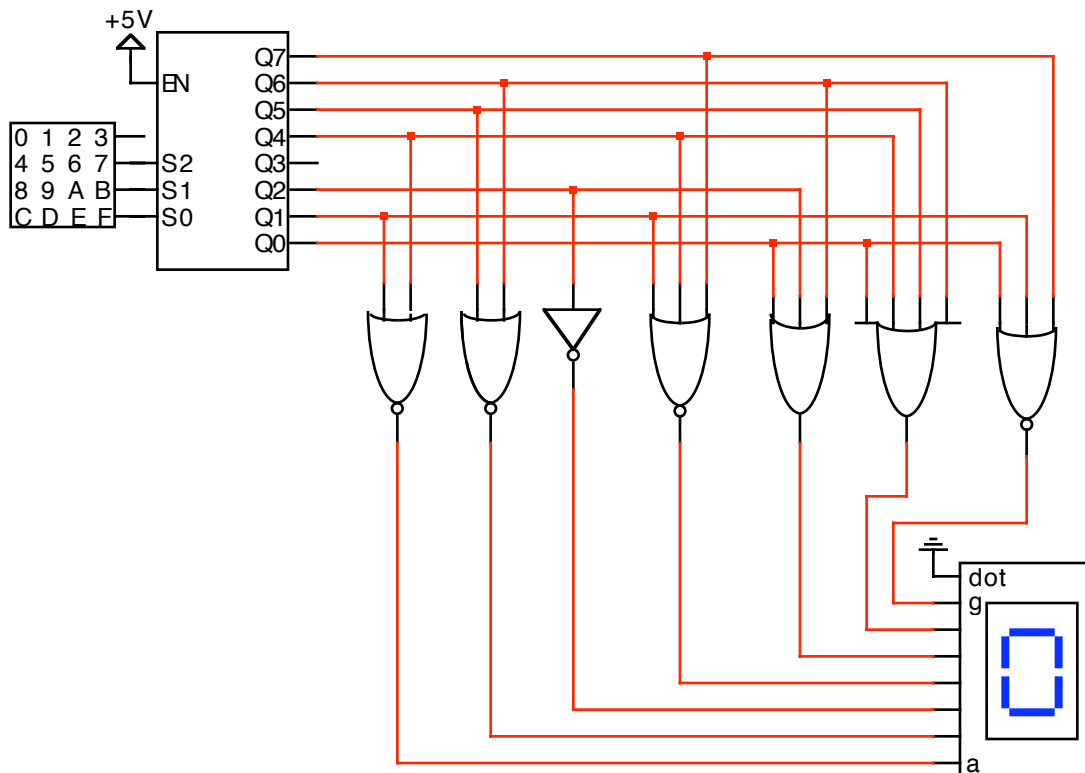
The segments to be lit have no clear mathematical relationship to numbers. They are related to numerals, so a specially designed driving circuit is required.

Input Code			Oct al	Display Code						
B2	B1	B0		a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	1	1	0	1	1	0	0	0	0
0	1	0	2	1	1	0	1	1	0	1
0	1	1	3	1	1	1	1	0	0	1
1	0	0	4	0	1	1	0	0	1	1
1	0	1	5	1	0	1	1	0	1	1
1	1	0	6	1	0	1	1	1	1	1
1	1	1	7	1	1	1	0	0	0	0

EXAMPLE: 7 SEGMENT DISPLAY

Design Approach:

Using standard techniques we can derive 7 logic equations corresponding to the display code shown in the truth table. A more straightforward approach is to use a 3-to-8 decoder as shown in the diagram below.



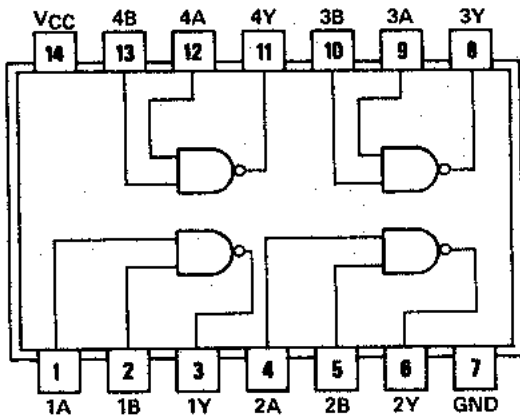
A 3-to-8 decoder is used to assert each octal code. Each segment combines 1's (OR) or 0's (NOR), depending on which are fewer.

e.g. segment a is OFF for 1 & 4, therefore $a = \text{NOR}(1,4)$
segment e is ON for 0,2,6, therefore $e = \text{OR}(0,2,6)$

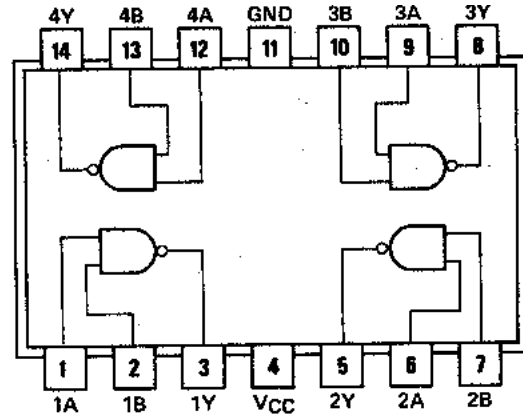
DIGITAL LOGIC TECHNOLOGIES

Transistor-Transistor Logic (TTL)

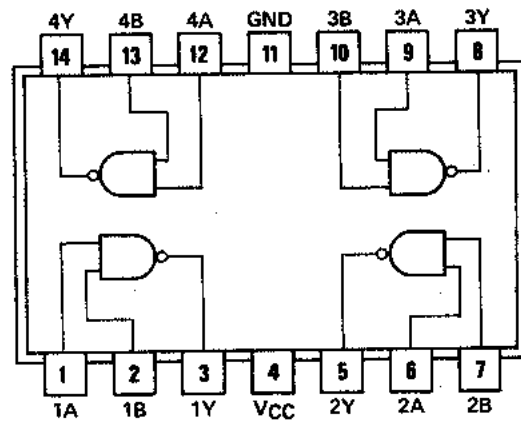
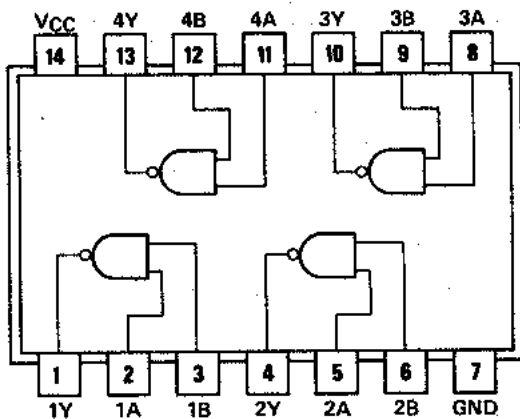
Introduced early 70's, low-medium density, used for "glue logic", still used to define functional building blocks in more modern technologies.



- | | |
|------------------------|------------------------|
| SN5400 (J) | SN7400 (J, N) |
| SN54H00 (J) | SN74H00 (J, N) |
| SN54L00 (J) | SN74L00 (J, N) |
| SN54LS00 (J, W) | SN74LS00 (J, N) |
| SN54S00 (J, W) | SN74S00 (J, N) |



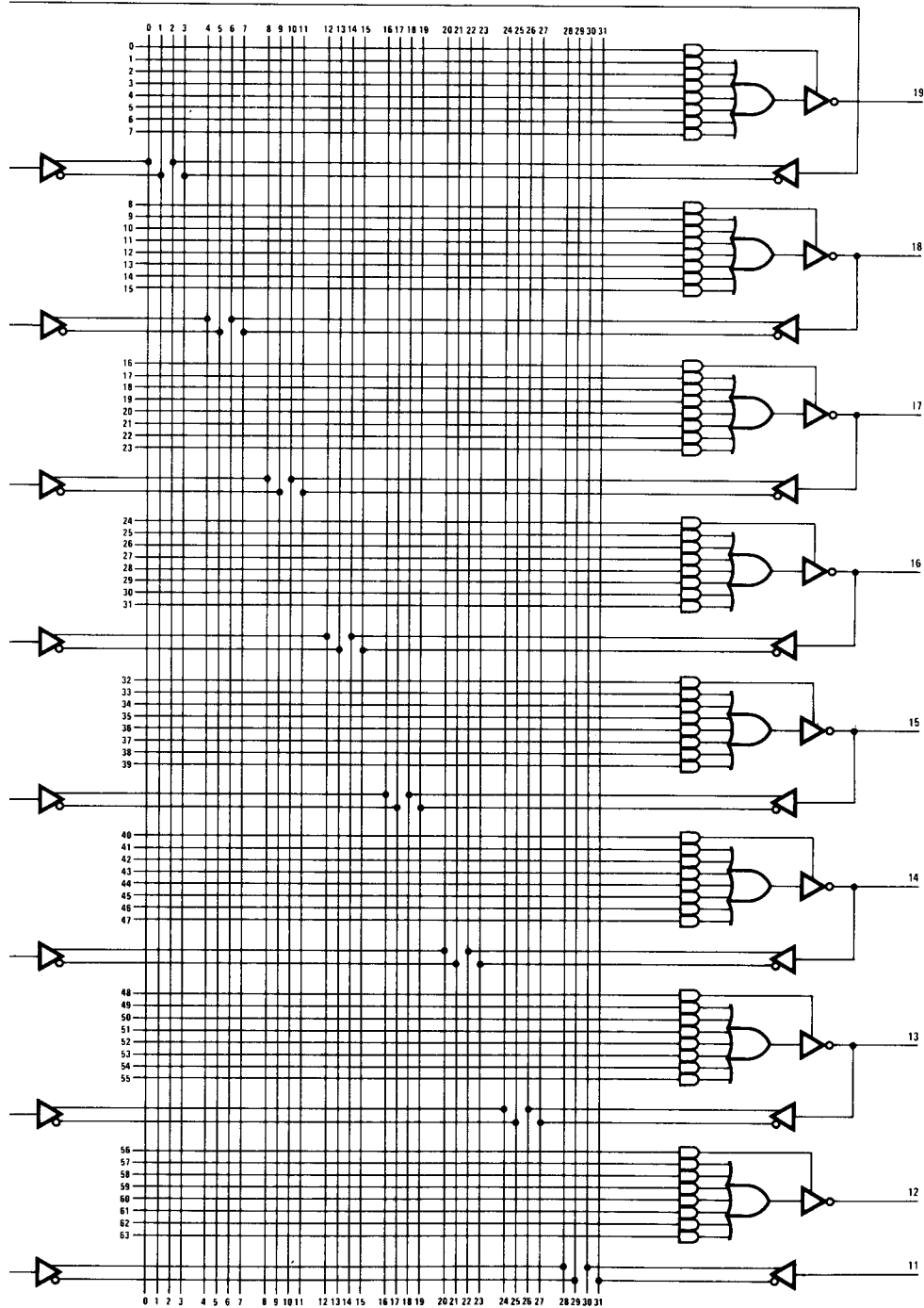
- | |
|--------------------|
| SN5400 (W) |
| SN54H00 (W) |
| SN54L00 (T) |



SN5401 (W)

DIGITAL LOGIC TECHNOLOGIES cont.

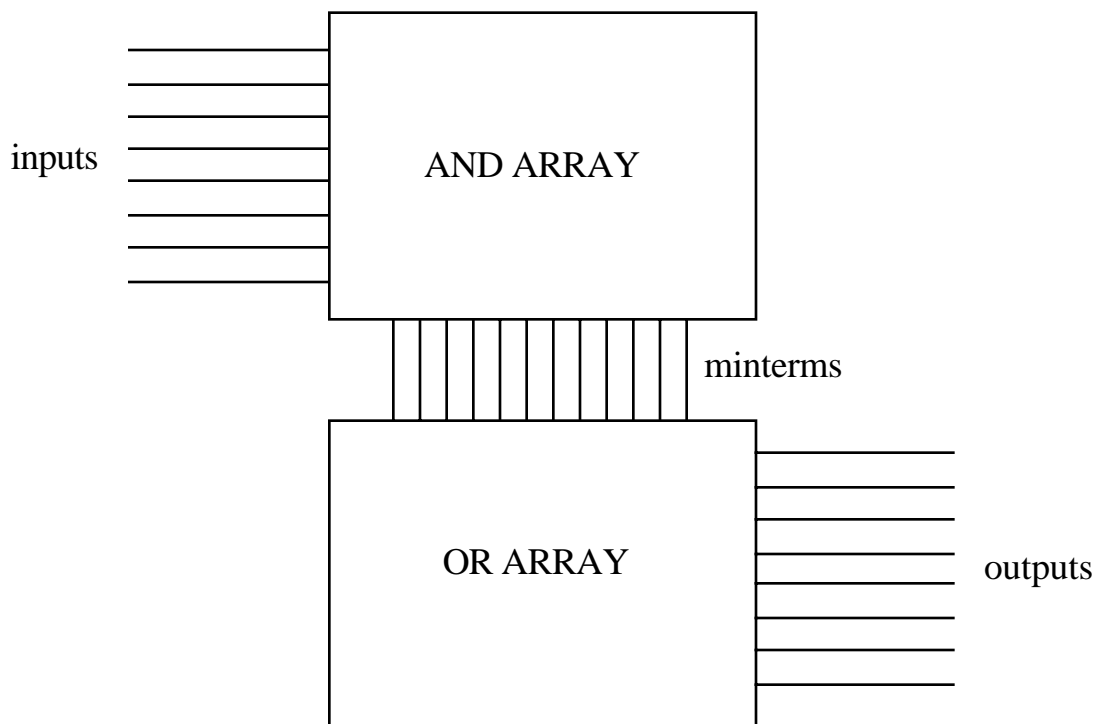
Programmable Array Logic (PALs)



DIGITAL LOGIC TECHNOLOGIES cont.

The PAL shown on the previous slide consists of a fixed number of multiple-input AND gates combined through OR terms. Connections to each AND are programmable, so there is considerable flexibility in the range of logic functions that can be implemented. The only limitation is the fixed number of minterms.

The Programmable Logic Array (PLA) shown below is a more general structure than the PAL. It is comprised of an AND plane and an OR plane. The size of the AND array is equal to the number of inputs, while the size of the OR array is equal to the number of terms times the number of outputs. In a PLA connections are programmable in BOTH the AND array and the OR array.



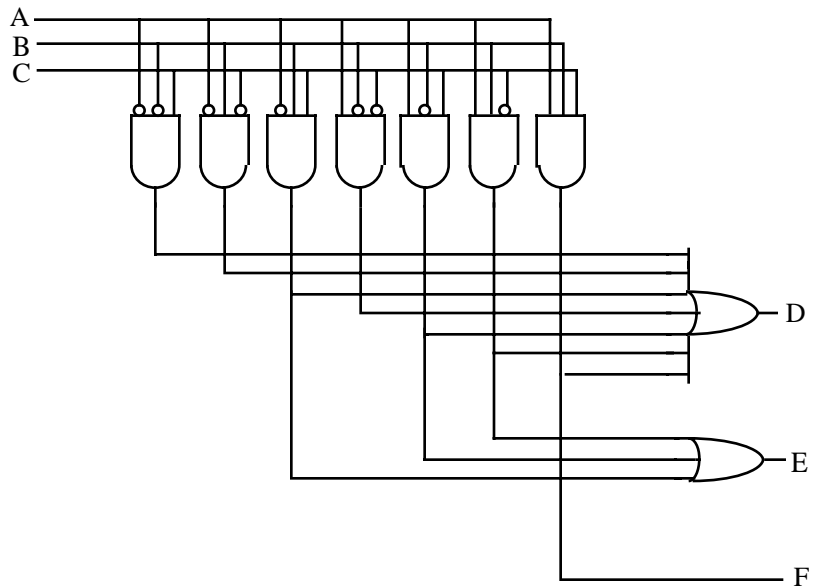
DIGITAL LOGIC TECHNOLOGIES cont.

EXAMPLE:

Consider the following truth table

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

There are seven unique terms with at least one true value in the outputs, so there will be seven columns in the AND plane. The number of rows in the AND plane is three since there are three inputs. There are three rows in the OR plane since there are three outputs. Once programmed, the PLA has the circuit at right.



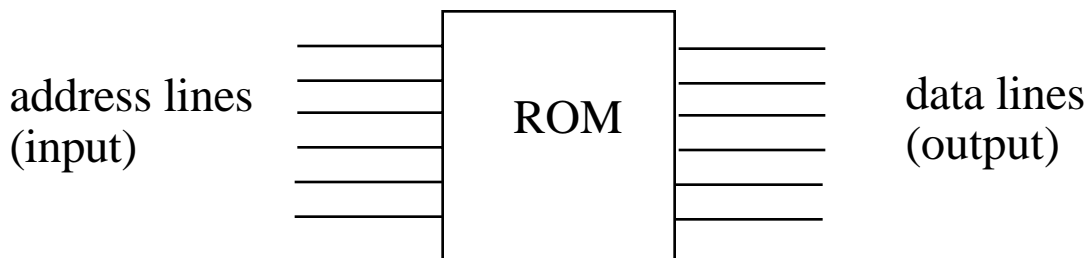
DIGITAL LOGIC TECHNOLOGIES cont.

READ-ONLY MEMORY (ROM)

Another form of structured logic that can be used to implement logic functions is the read-only memory. It has a fixed set of locations usually set at creation time that can be addressed and read. There are also programmable ROM's or PROM's, and some that can be erased and reprogrammed called EPROM's.

A ROM has N inputs (address lines) which can be used to specify 2^N memory locations. Each location is M bits wide, hence the output (data lines) has M lines.

A ROM can encode a logic function directly from the truth table with N inputs and M outputs. Each entry stores a row of the truth table.



ROMs are fully decoded: one full word for each possible input. So, a ROM always contains more entries than a PLA. As the number of inputs grows, the number of entries grows exponentially. PLA's grow much slower with the number of terms. ROM's are however sometimes convenient when the logic function may change but the number of input and outputs remains fixed.

Lookup Tables

Multiplexers as combinational logic

Full adder design using two 8-input multiplexers

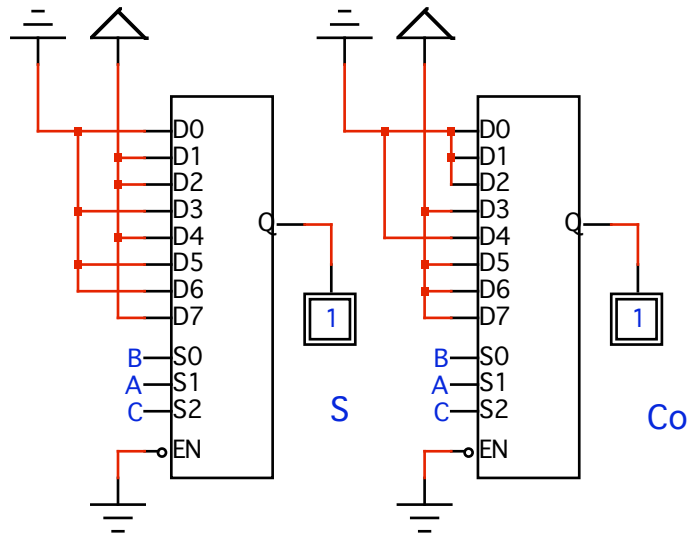
C	A	B	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Observe: multiplexer inputs are the output columns of the truth table.

1 — C
0 —

1 — A
0 —

1 — B
0 —



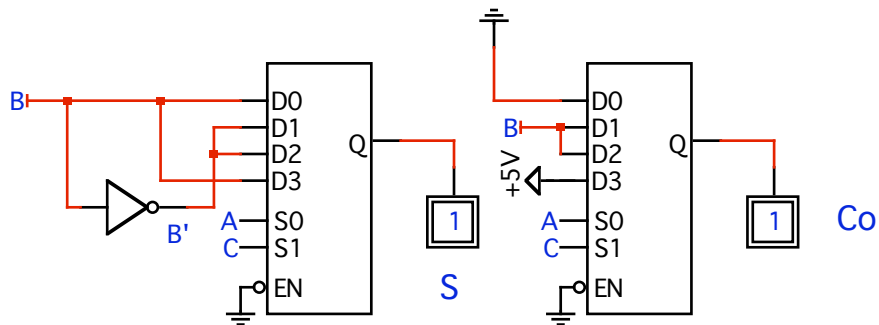
Multiplexers as combinational logic: continued

C	A	B	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

C	A	B	S	
0	0	0	0	S=B
0	0	1	1	S=B
0	1	0	1	$S=\overline{B}$
0	1	1	0	$S=\overline{B}$
1	0	0	1	$S=\overline{B}$
1	0	1	0	S=B
1	1	0	0	S=B
1	1	1	1	S=B

C	A	B	Co	
0	0	0	0	Co=0
0	0	1	0	Co=0
0	1	0	0	Co=B
0	1	1	1	Co=B
1	0	0	0	Co=B
1	0	1	1	Co=B
1	1	0	1	Co=1
1	1	1	1	Co=1

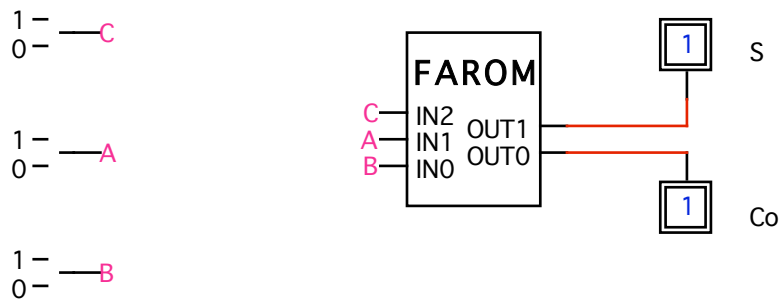
Full adder design using two 4-input multiplexers



Direct implementation using ROM

Full adder design using two 8-input multiplexers

C	A	B	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Inputs C, A, and B are used to form the 3-bit address to the PROM shown above, hence each input indexes a particular memory cell.

The cells are correspondingly programmed with the outputs of the truth table.

If you used LogicWorks to generate a PROM, the entries to this device would be, from lowest to highest address: 0, 2, 2, 1, 2, 1, 1, 3.