Logic Circuits
Semester 5

Ahmad Abu Zainab

Contents

Chapter 1

Number Systems and Conversion

- **Decimal** (Base 10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- **Binary** (Base 2): 0, 1
- **Octal** (Base 8): 0, 1, 2, 3, 4, 5, 6, 7
- **Hexadecimal** (Base 16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

1.1 Decimal and Binary

1.1.1 Decimal to Binary

1. Divide the decimal number by 2.
2. Keep the remainder.
3. Divide the quotient by 2.
4. Keep the remainder.
5. Repeat until the quotient is 0.
6. The binary number is the remainders in reverse order.

If the number has a decimal point then we multiply the decimal part by 2 the integer part of the result is the binary digit and the decimal part is multiplied by 2 again and so on until we obtain only a 1.

Example 1.1.1 (Convert 53_{10} to binary.)

$$\begin{aligned}
53/2 &= 26 \text{ remainder } 1\\
26/2 &= 13 \text{ remainder } 0\\
13/2 &= 6 \text{ remainder } 1\\
6/2 &= 3 \text{ remainder } 0\\
3/2 &= 1 \text{ remainder } 1\\
1/2 &= 0 \text{ remainder } 1
\end{aligned}$$

$$53_{10} = 110101_2.$$

Example 1.1.2 (Convert $.625_{10}$ to binary)

$$\begin{aligned} .625 \times 2 &= 1.25 \\ .25 \times 2 &= .5 \\ .5 \times 2 &= 1 \end{aligned}$$

$$.625_{10} = .101_2.$$

1.1.2 Binary to Decimal

1. Write the binary number.
2. Multiply each digit by 2^n where n is the position of the digit.
3. Add the results.

Example 1.1.3 (Convert 110101_2 to decimal.)

$$\begin{aligned} 110101_2 &= 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 16 + 0 + 4 + 0 + 1 \\ &= 53_{10} \end{aligned}$$

1.2 Binary and Hexidecimal

To convert from binary to hexadecimal we group the binary digits into groups of 4 starting from the right and convert each group to a hexadecimal digit.

$$1001101.010111_2 = \underbrace{0100}_{4}\ \underbrace{1101}_{D}\ .\ \underbrace{0101}_{5}\ \underbrace{1100}_{C} = 4D.5C_{16}.$$

1.3 Operations on Binary Numbers

1.3.1 Addition

$$\begin{array}{rccc} \text{carry} & 1 & & \\ & 0 & 1 & 1 \\ + & 1 & 1 & 0 \\ \hline & 1 & 0 & 1 \end{array}.$$

1.3.2 Subtraction

$$\begin{array}{rcccccc} \text{borrow} & & 1 & 1 & 1 & & \\ & 1 & 1 & 1 & 0 & 0 & 1 \\ - & & & 1 & 0 & 1 & 1 \\ \hline & 1 & 0 & 1 & 1 & 1 & 0 \end{array}.$$

1.3.3 Multiplication

$$
\begin{aligned}
0 \times 0 &= 0 \\
0 \times 0 &= 0 \\
0 \times 1 &= 0 \\
1 \times 1 &= 1
\end{aligned}
$$

$$
\begin{array}{ccccccccc}
 & & & & & 1 & 1 & 0 & 1 \\
\times & & & & & 1 & 0 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 0 & 0 & 0 & 0 & & \\
 & & 1 & 1 & 0 & 1 & & & \\
\hline
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array} .
$$

1.3.4 Division

Binary division is similar to decimal division, except it is much easier because the only two possible quotient digits are 0 and 1.
We start division by comparing the divisor with the upper bits of the dividend.
If we cannot subtract without getting a negative result, we move one place to the right and try again.
If we can subtract, we place a 1 for the quotient above the number we subtracted from and append the next dividend bit to the end of the difference and repeat this process with this modified difference until we run out of bits in the dividend.

1.4 Negative Numbers in Binary

Sign and Magnitude The leftmost bit is the sign bit and the rest of the bits are the magnitude.

$$-5_{10} = 1101_2.$$

One's Complement To negate a number we flip all the bits. $\bar{N} = (2^n - 1) - N$ where n is the number of bits.

$$-5_{10} = (2^4 - 1) - 5 = 10_{10} = 1010_2.$$

Two's Complement To negate a number we flip all the bits and add 1. $N^* = 2^n - N$ where n is the number of bits.

$$-5_{10} = 2^4 - 5 = 11_{10} = 1011_2.$$

Adding a negative number in two's complement is the same as subtracting the positive number. In one's complement we add the numbers and add the carry to the result.

Chapter 2

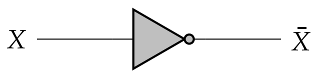
Boolean Algebra

2.1 Logic Gates

NOT Gate The output is the inverse of the input.

X	$\bar{X}$
0	1
1	0

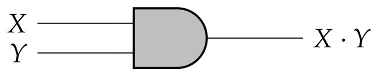
.



AND Gate The output is 1 if both inputs are 1.

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

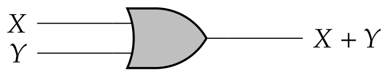
.



OR Gate The output is 1 if either input is 1.

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

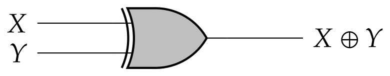
.



XOR Gate The output is 1 if either input is 1 but not both.

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

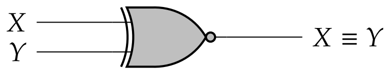
.



Equivalence Gate The output is 1 if both inputs are the same.

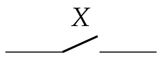
X	Y	$X \equiv Y$
0	0	1
0	1	0
1	0	0
1	1	1

.



2.1.1 Switches

If switch X is open, then we will define the value of X to be 0; if switch X is closed, then we will define the value of X to be 1.



2.2 Laws and Theorems

- Operations involving 0 and 1

$$\begin{aligned} X + 0 &= X \\ X + 1 &= 1 \\ X \cdot 1 &= X \\ X \cdot 0 &= 0 \end{aligned}$$

- Indempotent Laws

$$\begin{aligned} X + X &= X \\ X \cdot X &= X \end{aligned}$$

- Involution Law $$\bar{\bar{X}} = X.$$

- Laws of complementation

$$\begin{aligned} X + \bar{X} &= 1 \\ X \cdot \bar{X} &= 0 \end{aligned}$$

- Commutative Laws

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

- Associative Laws

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

- Distributive Laws

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

- Simplification Laws

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

$$\begin{aligned}
X \oplus 0 &= X \\
X \oplus 1 &= \bar{X} \\
X \oplus X &= 0 \\
X \oplus \bar{X} &= 1 \\
X \oplus Y &= Y \oplus X \\
X \oplus (Y \oplus Z) &= (X \oplus Y) \oplus Z = X \oplus Y \oplus Z \\
X \cdot (Y \oplus Z) &= X \cdot Y \oplus X \cdot Z \\
\overline{X \oplus Y} &= \bar{X} \oplus Y = X \oplus \bar{Y} = X \cdot Y + \bar{X} \cdot \bar{Y} = X \equiv Y
\end{aligned}$$

Consensus Theorem

$$X \cdot Y + \bar{X} \cdot Z + Y \cdot Z = X \cdot Y + \bar{X} \cdot Z.$$

in dual form

$$(X + Y)(\bar{X} + Z)(Y + Z) = (X + Y)(\bar{X} + Z).$$

An expression is in sum of products form if it is a sum of products of literals.

$$A\bar{B} + C\bar{D}E + A\bar{C}\bar{E}.$$

Likewise, an expression is in product of sums form if it is a product of sums of literals.

$$(A + B)(C + D)E.$$

2.3 De Morgan's Theorems

$$\begin{aligned}
\overline{X + Y} &= \bar{X} \cdot \bar{Y} \\
\overline{X \cdot Y} &= \bar{X} + \bar{Y}
\end{aligned}$$

2.4 Duality

Given an expression, we can obtain its dual by replacing all ANDs with ORs and all ORs with ANDs and all 0s with 1s and all 1s with 0s.

$$\begin{aligned}
(XYZ\cdots)^D &= X + Y + Z + \cdots \\
(X + Y + Z + \cdots)^D &= XYZ\cdots
\end{aligned}$$

The dual can also be found by taking the complement of the whole expression then complementing each variable individually

2.5 Minterm and Maxterm Expansions

Minterm A product term containing all the variables of the function in either true or complemented form.

Maxterm A sum term containing all the variables of the function in either true or complemented form.

Any Boolean function can be expressed as a sum of minterms or a product of maxterms.

Example 2.5.1 (Minterm)

$$f(A,B,C) = \underbrace{\bar{A}\bar{B}\bar{C}}_{000} + \underbrace{\bar{A}\bar{B}C}_{001} + \underbrace{\bar{A}BC}_{011} + \underbrace{\bar{A}B\bar{C}}_{010} + \underbrace{ABC}_{111} = \Sigma m(0,1,3,2,7).$$

Example 2.5.2 (Maxterm)

$$f(A,B,C) = (\underbrace{\bar{A}+\bar{B}+\bar{C}}_{111}) \cdot (\underbrace{\bar{A}+\bar{B}+C}_{110}) \cdot (\underbrace{\bar{A}+B+C}_{100}) \cdot (\underbrace{\bar{A}+B+\bar{C}}_{101}) \cdot (\underbrace{A+B+C}_{000}) = \Pi M(7,6,4,5,0).$$

Chapter 3

Karnaugh Maps

A Karnaugh map is a graphical representation of a truth table.

It is useful for simplifying Boolean expressions, by grouping together adjacent cells containing 1s.

We first draw a grid with the number of rows and columns equal to the number of variables.

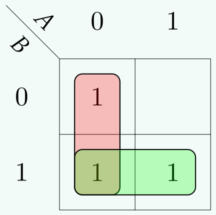
We then label the rows and columns with the binary values of the variables using a Gray code. (000, 001, 011, 010, 110, 111, 101, 100) No 2 adjacent numbers differ by more than 1 bit

We then fill in the cells with the output of the function.

We then group together adjacent cells containing 1s in powers of 2 (1, 2, 4, 8, 16, 32, 64, 128, 256, etc.)

We then go over the groups and write down the variables that remain the same throughout the group.

Example 3.0.1 (Two Variable K-map)
Take the following K-map



We look at the first group is 0 and 2(vertical), in this group A is not changing and it's value is 0. So the first group corresponds to $\bar{A}$.

The second group is 2 and 3(horizontal), in this group B is not changing and it's value is 1. So the second group corresponds to B.

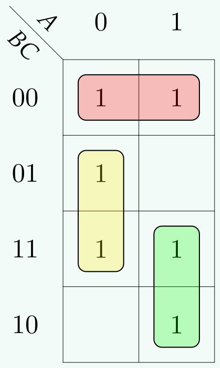
Therefore the simplified expression is

$$f(A,B) = \bar{A} + B.$$

Example 3.0.2 (Three Variable K-map)
Take the following function in min-term form

$$f(A,B,C) = \Sigma m(0,1,3,4,6,7).$$

We can represent this function in a K-map as follows



First group(red) the only changing variable is A. So it's excluded in the term of the group which is $\bar{B}\bar{C}$.
Second group(yellow) the only changing variable is B. So it's excluded in the term of the group which is $\bar{A}C$.
Third group(green) the only changing variable is C. So it's excluded in the term of the group which is AB.

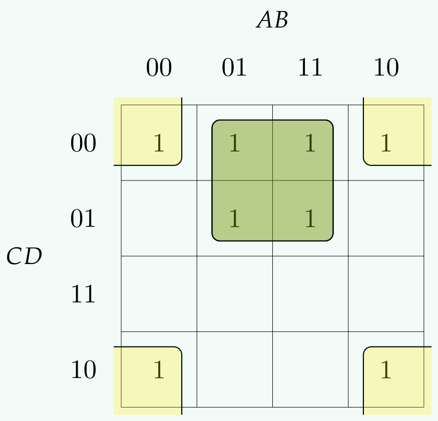
Therefore the simplified expression is

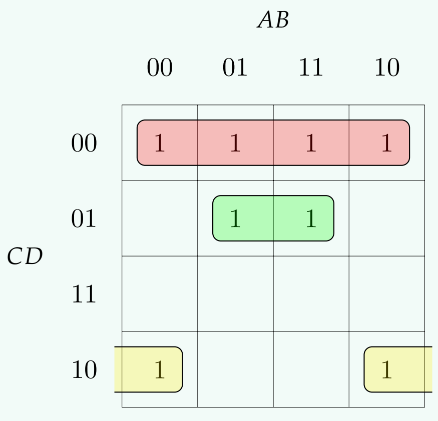
$$f(A,B,C) = \bar{B}\bar{C} + \bar{A}C + AB.$$

Example 3.0.3 (Four Variable K-map)
Take the following function in min-term form

$$f(A,B,C,D) = \Sigma m(0,2,4,5,8,10,12,13).$$

We can represent this function in a K-map as follows





First K-map gives us the simplified form as

$$f(A,B,C,D) = B\bar{C} + \bar{B}\bar{D}.$$

And the second K-map gives us

$$f(A,B,C,D) = \bar{C}\bar{D} + B\bar{C}D + \bar{B}C\bar{D}.$$

To untilize maxterms in a K-map we first fill the maxterm cells with 0s and the rest with 1s, and then we follow the same procedure as before. This will give us $\bar{f}$ then we simply compute the inverse to get f.

Note:-

Sometimes we don't care about some of the outputs of a function (Don't care conditions). We can represent this in a K-map by filling the cells with don't care conditions with an X. An example is

$$f(A,B,C) = \Sigma m(0,1,3,4,5,7) + d(2,6).$$

Chapter 4

Combinational Logic

Combinational Logic A logic circuit whose output depends only on the current input values.

Sequential Logic A logic circuit whose output depends on the current input values and the past input values.

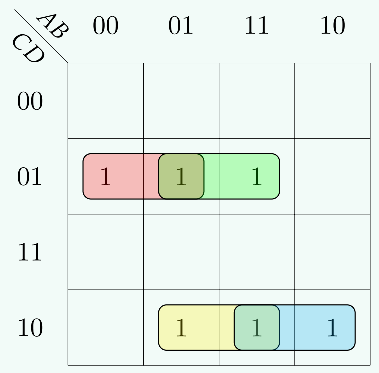
The maximum number of gates cascaded in series between a circuit input and the output is referred to as the number of *levels* of gates.

Example 4.0.1

We need to realize the 2 level circuit and the 3 level circuit. The 2 level circuit is realized by cascading 2 gates in series and the 3 level circuit is realized by cascading 3 gates in series.

$$f(a,b,c,d) = \Sigma m(1,5,6,10,13,14).$$

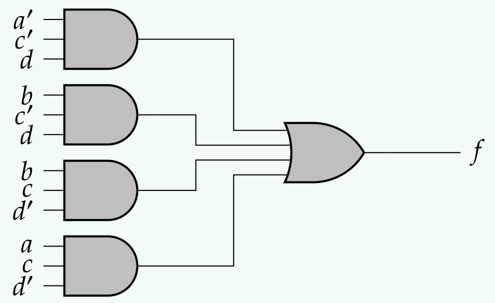
First we construct the k-map



Giving us

$$f = a'c'd + bc'd + bcd' + acd'.$$

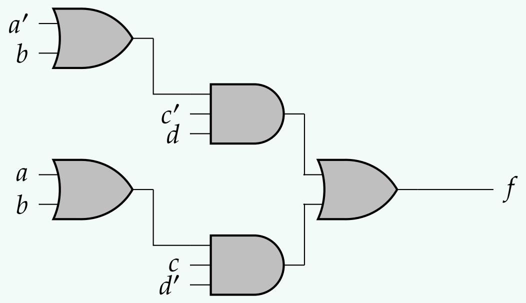
Yielding the following circuit



The 3 level circuit is realized by cascading 3 gates in series. We obtain the expression for the 3 level circuit by factoring f as follows

$$f = c'd(a' + b) + cd'(a + b).$$

Giving us the following circuit



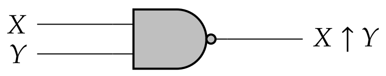
We do this again for max-term form. Then we compare all the circuits and choose the one with the least number of gates.

4.1 NAND and NOR Gates

NAND Gate The output is 0 if both inputs are 1.

X	Y	$X \uparrow Y$
0	0	1
0	1	1
1	0	1
1	1	0

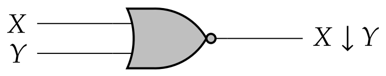
.



NOR Gate The output is 1 if both inputs are 0.

X	Y	$X \downarrow Y$
0	0	1
0	1	0
1	0	0
1	1	0

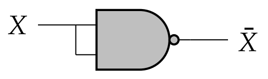
.



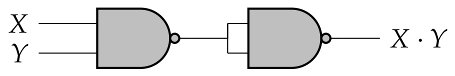
A set of gates is said to be functionally complete if all Boolean functions can be realized using only gates from the set. AND and NOT gates are functionally complete since we can realize all Boolean functions using only AND and NOT gates (including OR).

Since we can realize AND and NOT gates using only NAND gates, NAND gates are also functionally complete.

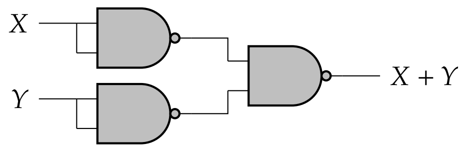
1. **NOT**



2. **AND**



3. **OR**



4.2 Design of Two-Level NAND and NOR-Gate Circuits

A circuit composed of AND and OR gates can be converted to a circuit composed of NAND and NOR gates by using the formula $f = \overline{f'}$.

Example 4.2.1
Given the function

$$F = A + BC' + B'CD.$$

We can convert it to several forms

$$\begin{aligned} F &= [(A + BC' + B'CD)']' && \text{AND-OR} \\ &= [A' \cdot (B + C')' \cdot (B' + C + D)']' && \text{NAND-NAND} \\ &= [A' \cdot (B' + C) \cdot (B + C' + D')]' && \text{OR-NAND} \\ &= A + (B' + C)' + (B + C' + D')' && \text{NOR-OR} \end{aligned}$$

The procedure for designing a two-level NAND-NAND circuit is as follows

1. Find a minimum sum-of-products expression for F.
2. raw the corresponding two-level AND-OR circuit.
3. Replace all gates with NAND gates leaving the gate interconnections unchanged.
4. If the output gate has any single literals as inputs, complement these literals.

The procedure for designing a two-level NOR-NOR circuit is as follows

1. Find a minimum product-of-sums expression for F.
2. Draw the corresponding two-level OR-AND circuit.

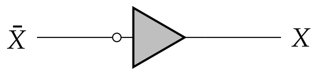
3. Replace all gates with NOR gates leaving the gate interconnections unchanged.
4. If the output gate has any single literals as inputs, complement these literals.

To design a multi-level NAND-gate circuit we perform the following procedure

1. Simplify the function.
2. Draw the corresponding AND-OR circuit. The output gate should be an OR gate.
3. Replace all gates with NAND gates leaving the gate interconnections unchanged.
4. Take the output-gate at level 1. The literal inputs to even levels remain unchanged while the literal inputs to odd levels are complemented.

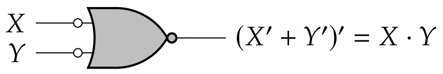
4.2.1 Alternative Representations

We will sometimes use a shorter circuit representation

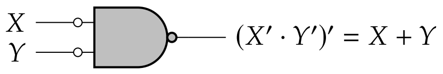


to represent a circuit using a NOT gate.

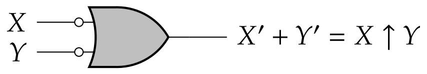
1. **AND**



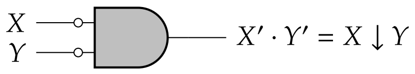
2. **OR**



3. **NAND**



4. **NOR**



These alternative representations are useful for simplifying circuits when converting to and from NAND-NAND and AND-OR.
The following procedure may be used to convert to a NAND (or NOR) circuit:

1. We add an inversion bubble to the input of the odd levels.
2. If an inverted input is connected to an inverted output, we leave it as is.
3. If an inverted input is connected to a non-inverted output (or vice-versa), we add an inversion bubble to the non-inverted terminal to cancel the inversion.
4. If an literal is connected to an inverted input, we complement the literal.

4.3 Design of Combinational Circuits

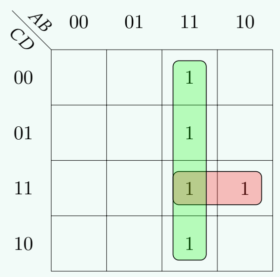
Digital design systems' solution often requires implementing multiple functions. While we can implement each function separately, it is often more efficient to implement all the functions together.

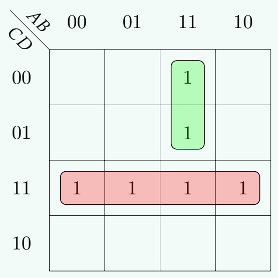
Example 4.3.1

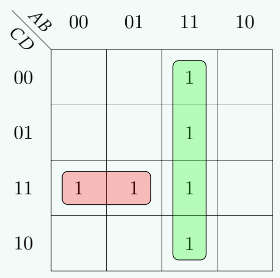
We need to implement the following functions

$$f_1 = \Sigma m(11, 12, 13, 14, 15)$$
$$f_2 = \Sigma m(3, 7, 11, 12, 13, 15)$$
$$f_3 = \Sigma m(3, 7, 12, 13, 14, 15)$$

We can implement each function separately as follows

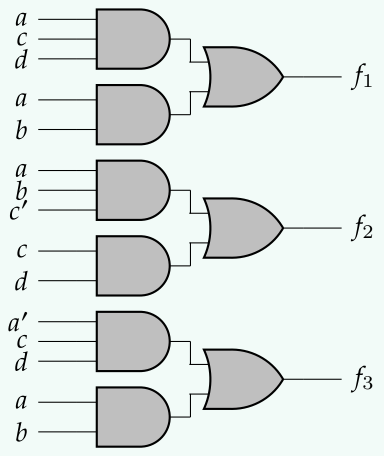




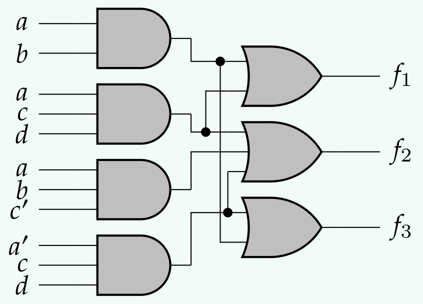


$$f_1 = ab + acd$$
$$f_2 = abc' + cd$$
$$f_3 = a'cd + ab$$

The obtained circuit for all 3 separately looks like



By grouping terms we can simplify the circuit to



The simplified circuit has 7 gates while the original circuit has 9 gates.

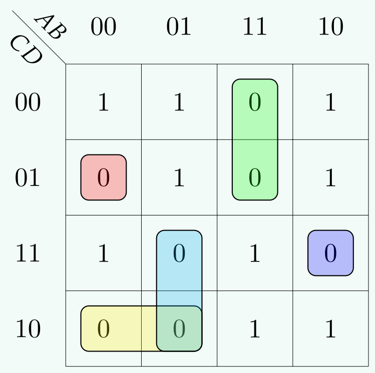
4.4 Design of Circuits with Limited Gate Fan-In

When designing a circuit we may be limited by the number of inputs a gate can have. We can overcome this by factoring the function into a multi-level circuit.

Example 4.4.1

We need to implement the following function using only 3 input NOR gates.

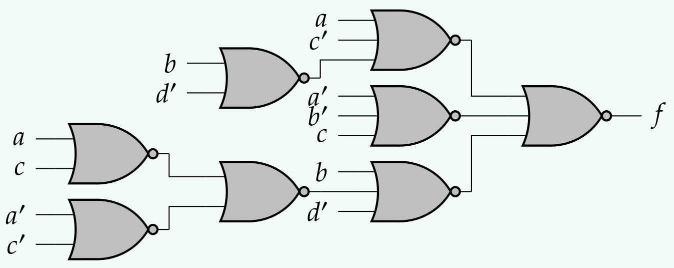
$$f = \Sigma m(0, 3, 4, 5, 8, 9, 10, 14, 15).$$



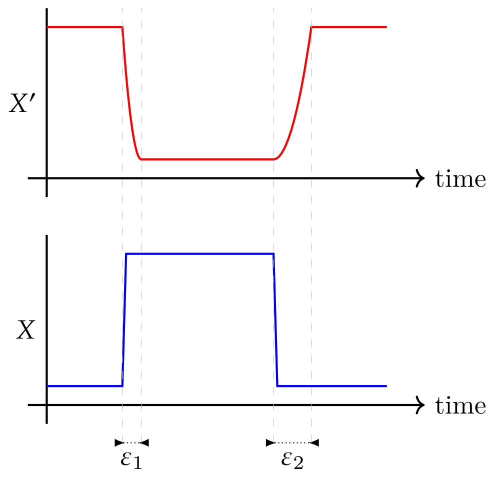
CD \ AB	00	01	11	10
00	1	1	0	1
01	0	1	0	1
11	1	0	1	0
10	0	0	1	1

The simplest form (two-level) requires 2 4-input gate and 1 5-input gate. We can factor the function to remedy this

$$
\begin{aligned}
f' &= b'd(a'c' + ac) + a'c(b + d') + abc' \\
f &= [b + d' + (a + c)(a' + c')][a + c' + b'd][a' + b' + c]
\end{aligned}
$$



4.5 Gate Delays and Timing Diagrams

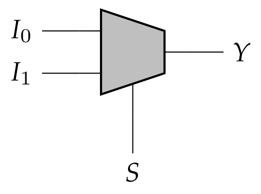


Chapter 5

Multiplexers, Decoders, and Programmable Logic Devices

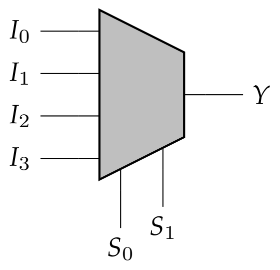
5.1 Multiplexers

A multiplexer is a combinational circuit that selects one of many inputs and directs it to a single output.



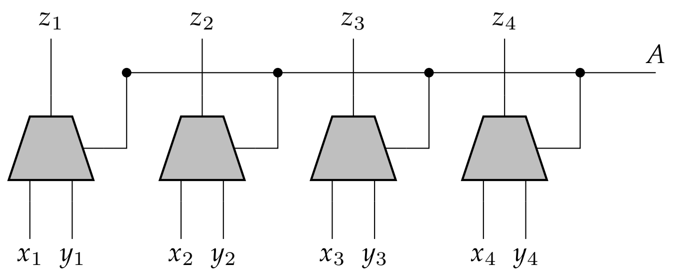
$$Y = S' \cdot I_0 + S \cdot I_1.$$

The multiplexer has 2^n inputs, n select inputs, and 1 output. The select inputs determine which input is connected to the output.



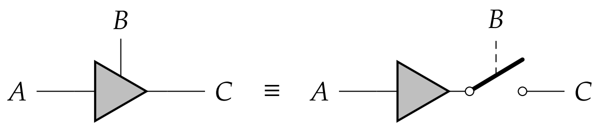
$$Y = S_0'S_1'I_0 + S_0'S_1I_1 + S_0S_1'I_2 + S_0S_1I_3.$$

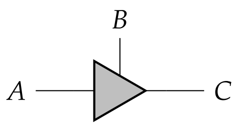
Multiplexers are typically used in digital design systems to select data to be processed.



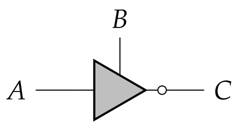
5.2 Three-State Buffers

A three-state buffer is a circuit that can be in one of three states: high, low, or high impedance. It is used to connect multiple outputs to a single input.

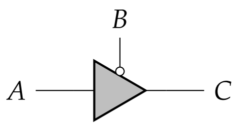




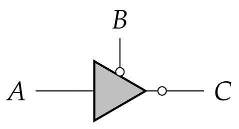
B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1



B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0



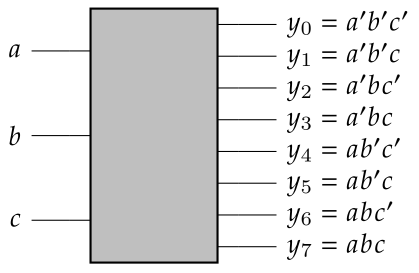
B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z



B	A	C
0	0	1
0	1	0
1	0	Z
1	1	Z

5.3 Decoders

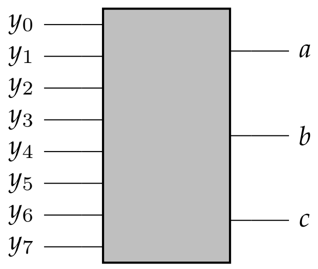
A decoder is a combinational circuit that converts a binary code to the associated minterm for that pin.



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

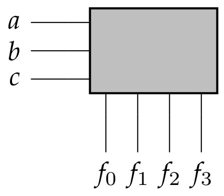
5.4 Encoders

An encoder is a combinational circuit that performs the inverse operation of a decoder. It converts a minterm to the associated binary code.



5.5 Read-Only Memory

A read-only memory (ROM) consists of an array of semiconductor devices that are interconnected to store an array of binary data. Once binary data is stored in the ROM, it can be read out whenever desired, but the data that is stored cannot be changed under normal operating conditions.



The truth table for the ROM depends on the data stored. The ROM can be thought off as an array of values and the input abc acts an index to the array.