PES
UNIVERSITY

Department of Computer Science & Engineering
Digital Design and Computer Organisation

UE20CS201

UNIT 1 Notes

Unit-1: Combinational Logic Design

Note: Additional Material

Boolean Functions

The binary variables and logic operations are used in Boolean algebra. The algebraic expression is known as **Boolean Expression**, is used to describe the **Boolean Function**. The Boolean expression consists of the constant value 1 and 0, logical operation symbols, and binary variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n, where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through 2^{n-1}.

Example 1: F=xy' z+p

We defined the Boolean function **F=xy' z+p** in terms of four binary variables x, y, z, and p. This function will be equal to 1 when x=1, y=0, z=1 or z=1.

Example 2: F(A,B,C,D)=A+BC'+D

Example 3: $F1 = x + y'z$

Truth table for the function F1

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Algebra

Boolean Algebra is the mathematics we use to analyse digital gates and circuits. We can use these "Laws of Boolean" to both reduce and simplify a complex Boolean expression in an attempt to reduce the number of logic gates required. *Boolean Algebra* is therefore a system of mathematics based on logic that has its own set of rules or laws which are used to define and reduce Boolean expressions.

The variables used in **Boolean Algebra** only have one of two possible values, a logic "0" and a logic "1" but an expression can have an infinite number of variables all labelled individually to represent inputs to the expression, For example, variables A, B, C etc, giving us a logical expression of A + B = C, but each variable can ONLY be a 0 or a 1.

Laws of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Logic minimization

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x\_ + y) = xx\_ + xy = 0 + xy = xy$.

2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y$.

3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x$.

4. $xy + x'z + yz = xy + x'z + yz(x + x')$

$= xy + x'z + xyz + x'yz$

$= xy(1 + z) + x'z(1 + y)$

$= xy + x'z$.

5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

PES UNIVERSITY

Department of Computer Science & Engineering
Digital Design and Computer Organisation
UE20CS201
UNIT 1 Notes

Unit-1: Combinational Logic Design- K-maps

what is K-map (Karnaugh map)

A k-map is a pictorial method used to minimize Boolean expressions without having to use Boolean algebra. theorems and equations. It can be thaught as pictorial representation of truth table. We can use 2-variables 3-variables or 4-variables K-maps to simplify the boolean functions / expressions.

Steps to solve boolean functions / expressions

1. select K-map according to the number of variables
2. Identify minterms or maxterms for a given problem.
3. For SOP (Sum of Products) place '1's in the cells of K-map for the respective miniterms '0' otherwise.
4. For POS (Product of sums) place 0's in the cells of the K-map for the respective max terms, 1's otherwise.
5. make the grouping of cells (1, 2, 4, 8, 16) cover maximum number of 1's or 0's in the group.
6. From the groups, obtain the product terms and

Sum them up for SOP form.

Note: The number of adjacent squares that may be combined must always represent a number that is a power, such as 1, 2, 4, 8

→ As more adjacent squares are combined we obtain a product term with fewer literals.

* considering 3-variable K-map, the number of literals in the ~~sop~~ product or sum term are.

① one square represents one minterm, giving a term with 3-literals.

② Two adjacent squares represent a term with two literals

③ Four adjacent squares represent a term with one literal.

④ eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

K-maps for 3-variables
examples on SOP form.

① Simplify the following boolean functions using 3-variable K-map.

ⓐ $F(x,y,z) = \Sigma(2,3,4,5)$

$x \backslash yz$	00	01	11	10
0			1	1
1	1	1		

$F = xy' + x'y$

(b) $F(x,y,z) = \Sigma(3,4,6,7)$

$x \backslash yz$	00	01	11	10
0			1	
1	1		1	1

xz'

yz

$F = yz + xz'$

(c) $F(x,y,z) = \Sigma(0,2,6,7)$

$x \backslash yz$	00	01	11	10
0	1			1
1			1	1

$F = xy + x'z'$

(d) $F(x,y,z) = \Sigma(0,2,3,4,6)$

$x \backslash yz$	00	01	11	10
0	1		1	1
1	1			1

$F = x'y + y'z' + yz'$

(e) $F(x,y,z) = \Sigma(1,4,5,6,7)$

$x \backslash yz$	00	01	11	10
0		1		
1	1	1	1	1

$F = x + y'z$.

K-map examples for POS form.

(a) $F(x,y,z) = \Pi(0,3,6,7)$

$x \backslash yz$	00	01	11	10
0	0		0	
1			0	0

$F' = x'y'z' + y'z' + xy$

$F = (x+y+z)(y+z)(x'+y')$

(b) $F(x, y, z) = \Pi(0, 2, 3, 4, 6)$

$x \backslash yz$	00	01	11	10
0	0		0	0
1	0	1		0

$F' = z' + x'y$

$F = z(x + y')$

K-map - 4 variables.

Simplify the following boolean functions using 4-variable K-maps

(a) $f(a, b, c, d) = \Sigma(1, 3, 12, 13, 14, 15)$

$a'b'd$.

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

ab

$f = ab + a'b'd$

(b) $f(a, b, c, d) = \Sigma(1, 5, 9, 10, 11, 14, 15)$

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

$a'b$

$b'c'$

$f = a'b + b'c'$

Product of sums simplification. (POS)

In POS, cells containing '0' are grouped. Function obtained is F'

(a) $F(A,B,C,D) = \Pi(3,4,6,7,11,12,13,14,15)$

K-map

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

$$F' = CD + AB + BD'$$

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

POS

(b) $F(A,B,C,D) = \Pi(1,3,5,7,13,15)$

K-map

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

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

$$F = (A+D')(B'+D')$$

(c) $F(A,B,C,D) = \Pi(1,3,4,6,9,11,12,14)$

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

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

$$F = (B'+D)(B+D')$$

4- Variable K maps(SOP)

a) $F(a,b,c,d) = \Sigma(1, 5, 9, 10, 11, 14, 15)$

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

$F = a'c'd + ac + ab'd$

b) $F(a,b,c,d) = \Sigma(0, 2, 4, 5, 6, 7, 8, 10, 13, 15)$

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

$F = b'd' + a'b + bd$

c) $F(a,b,c,d) = \Sigma(1, 4, 5, 6, 7, 13)$

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

$F = a'b + a'c'd + bc'd$

Prime implicants

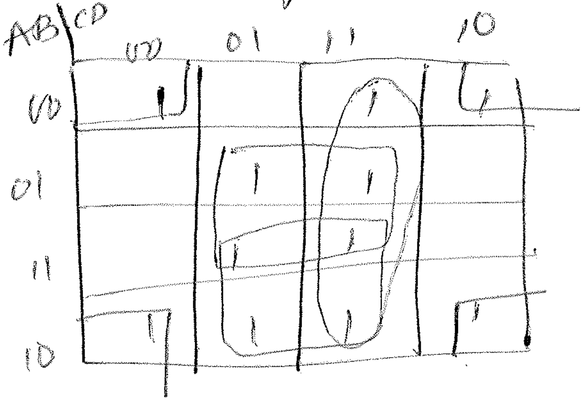
A *Prime implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map. If a minterm in a square is covered by only one prime implicant, then that prime implicant is said to be essential.

Examples

Simplify the following finding all the prime and essential prime implicants.

(a) $F(A,B,C,D) = \Sigma(0,2,3,5,7,8,9,10,11,13,15)$

K-map

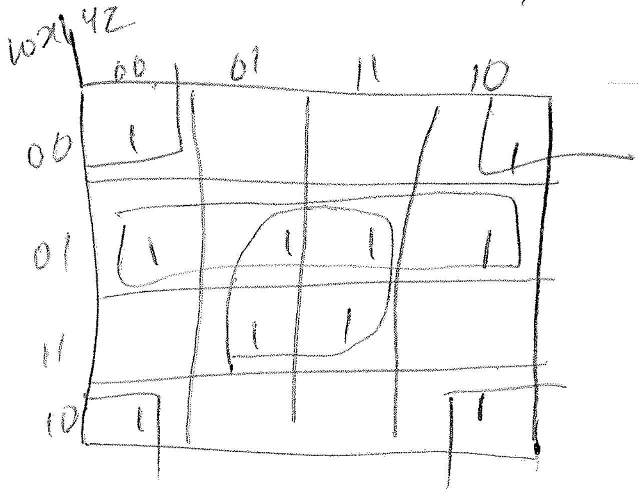


$F = B'D' + CD + BD + AD$

Prime implicants (PI): $B'D'$, CD, BD & AD

Essential PI = $B'D'$; BD

(b) $F(w,x,y,z) = \Sigma(0,2,4,5,6,7,8,10,13,15)$



$F = wx' + xz + x'z'$

Prime implicants : wx', xz, $x'z'$

Essential PI = xz, $x'z'$

PES UNIVERSITY

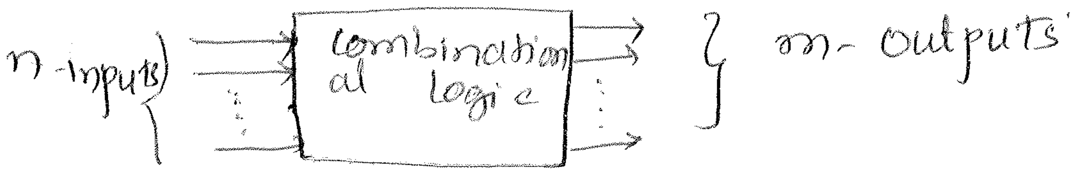
Department of Computer Science & Engineering
Digital Design and Computer Organisation
UE20CS201
UNIT 1 Notes

Unit-1: Combinational Logic Design

NOTE: Additional material.

Combinational logic

* combinational circuits consists of logic gates whose outputs at any time are determined by the present combination of inputs.



for n input variables, there are 2^n possible binary input combinations. Each input combination there will be one output value.

examples: Adders, subtractors, comparitors, multiplexers, demux, encoders, decoders & seven-segment decoder

Design procedure for combinational circuits

steps involved in design procedure.

1. From the specifications of the problem statement, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs

contd.

3) obtain the simplified boolean functions for each output as a function of the input variables.

4) Draw the logic diagram and verify the correctness of the design.

Examples

1) Design a combinational circuit with 3 inputs and one output.

(a) The output is '1' when the binary value of the input is less than 3. The output is '0' otherwise.

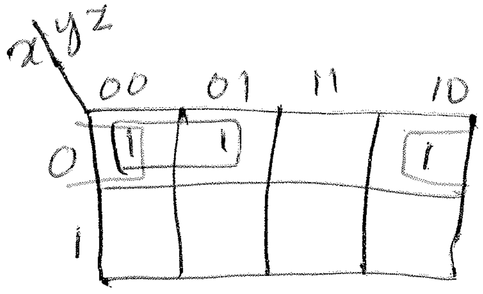
(b) The output is 1 when the binary value of the inputs is an even number.

Solution

(a) Truth table.

Inputs x	y	z	Output F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

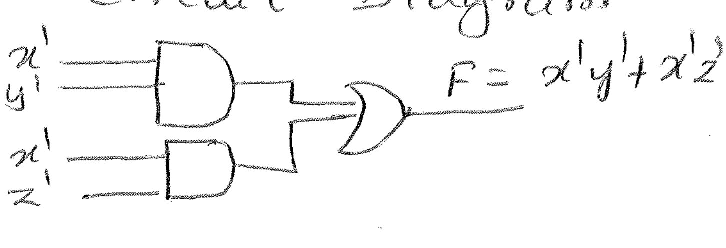
K-map.



equation:

$$F = x'y' + x'z'$$

Circuit Diagram



(b) Truth table

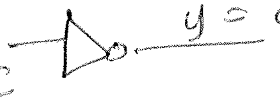
Inputs a	b	c	output y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

K-map

a \ bc	00	01	11	10
0	1			1
1	1			1

equation: $y = c'$

circuit diagram.



Note: '0' is considered as even number.

Example 2:

Design a combinational circuit with 3 inputs x, y, z and 3 outputs A, B, C. ~~cases~~ when the binary input is 0, 1, 2, 3 the binary output is '1' greater than input ~~case~~ and when the binary input is 4, 5, 6, 7 the binary output is one less than the input.

Solution.

Truth table.

Inputs x	y	z	outputs A	B	C
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

K-map for output A.

x \ yz	00	01	11	10
0			1	
1		1	1	1

equation:

$A = xy + yz + xz$

K-map for output B

$x \backslash yz$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Equation

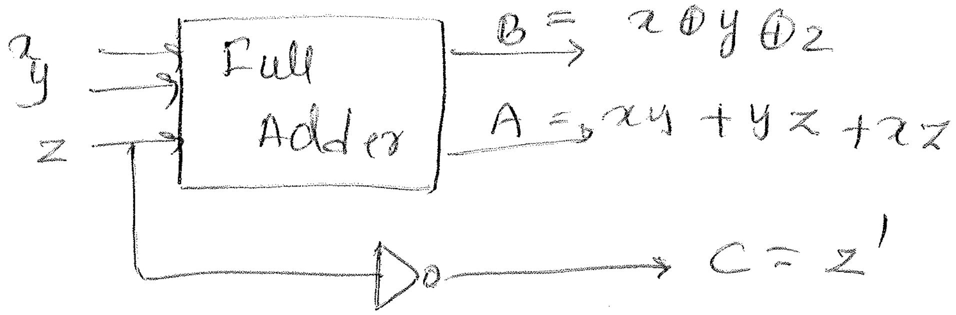
$B = x \oplus y \oplus z$

K-map for output C

$x \backslash yz$	00	01	11	10
0	1			1
1	1			1

$C = z'$

Circuit Diagram



Adder/Subtractor, overflow

Positive integers (including zero) can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with binary digits. It is customary to represent the sign with a bit placed in the leftmost position of the number. The convention is to make the sign bit 0 for positive and 1 for negative.

It is important to realize that both signed and unsigned binary numbers consist of a string of bits when represented in a computer. The user determines whether the number is signed or unsigned. If the binary number is signed, then the leftmost bit represents the sign and the rest of the bits represent the number. If the binary number is assumed to be unsigned, then the leftmost bit is the most significant bit of the number.

For example,

1. The string of bits 01001 can be considered as 9 (unsigned binary) or as +9 (signed binary) because the leftmost bit is 0.
2. The string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned number and the binary equivalent of -9 when considered as a signed number. This is because the 1 that is in the leftmost position designates a negative and the other four bits represent binary 9.

   The representation of the signed numbers in the above example is referred to as the *signed-magnitude* convention. In this notation, the number consists of a magnitude and a symbol (+ or -) or a bit (0 or 1) indicating the sign.

The signed-magnitude system is used in ordinary arithmetic, but is awkward when employed in computer arithmetic because of the separate handling of the sign and the magnitude. Therefore, the signed-complement system is normally used.

Adder/Subtractor, Overflow

Binary Addition Rules

Arithmetic rules for binary numbers are quite straightforward, and similar to those used in decimal arithmetic. The rules for addition of binary numbers are:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = (1)0

Sum=0 and carry =1

Binary addition is carried out just like decimal, by adding up the columns, starting at the right and working column by column towards the left.

Example 1

	Decimal	Binary
	2	10
	1 +	01 +
Answer	3	11

Example 2

	Decimal	Binary
	3	0011
	1 +	0001 +
Carry	_	0110
	4	0100

Arithmetic Addition

1. Perform the arithmetic addition of the following numbers with 8-bit to represent a number (1-bit for sign and 7 bits for magnitude)

a. (+6) + (+13) b. (-6) + (+13) c). (+6) + (-13) d. (- 6) + (-13)

Solution

a) +6 : 0000 0110
\+ +13 0000 1101
+19 0001 0011

b) -6 :
\+ +13

Represent 6 with 8-bits
6: 0000 0110
Find 1's complement
1111 1001
Find 2's complement of -6
* Add '1' to 1's compleme nt
1's comp
~~0000 0110~~ → 1111

Adding 1 to 1's complement

1111 1001 + 1 = 1111 1010

$$\begin{array}{rcl} & -6 : & 1111\ 1010 \\ + & +13 : & +\ 0000\ 1101 \\ \hline & +7. & \boxed{1}\ 0000\ 0111 \Rightarrow +7 \end{array}$$

ignore
Carry generated out of MSB bit

(c) +6 + (−13)

Finding 2's complement of −13.
Finding 2's complement of −13 with 8-bits.

* represent 13 with 8-bits.

13 - 0000 1101

Find 1's complement of 13 = 1111 0010

Find 2's complement, by adding "1" to 1's complement.

$$\begin{array}{r} 1111\ 0010 \\ +\ \ \ \ \ \ \ \ \ \ 1 \\ \hline 1111\ 0011 \end{array}$$

$$\begin{array}{rcl} & +6 — & 0000\ 0110 \\ + & -13 — & +\ 1111\ 0011 \\ \hline & -7 & \boxed{1}111\ 1001 \end{array}$$

Check the sign bit of the result.
if the sign bit is '1', then the result is in 2's complemented form. Hence take the 2's complement of result to get the value.

Applying 2's complement to the result

$$\begin{array}{r} 0000\ 0110 \\ +\quad 1 \\ \hline 0000\ 0111 \\ \hline \end{array} \Rightarrow 7 \text{ (magnitude)}$$

sign bit - 1

Hence the ans/result will be -7

(d) $(-6) + (-13)$

$$\begin{array}{rcl} -6 & : & 1111\ 1010 \\ + \ -13 & : & 1111\ 0011 \\ \hline -19 & & \boxed{1}\ 1110\ 1101 \end{array}$$

ignore

* Here the result is -ve. Hence find the 2's complement of the result.

2's complement of result

$$\begin{array}{r} 0001\ 0010 \\ +\quad 1 \\ \hline 0001\ 0011 \end{array} \Rightarrow 19 \text{ (magnitude)}$$

sign = 1

Hence the result will be -19

2. Convert decimal +49 and +29 to binary, using the signed-2's-complement representation and enough digits to accommodate the numbers. Then perform the binary equivalent of (+29) + (-49), (-29) + (+49), and (-29) + (-49). Convert the answers back to decimal and verify that they are correct.

Solution:

Decimal --> Binary

+ 29 --> 00011101

+ 49 --> 00110001

- 29 --> 11100011

1's complement of 29 +1 = 11100010+1 = 11100011

- 49 --> 11001111

1's compliment of 49 +1 = 11001110+1 = 11001111

Now we apply the normal binary arithmetic to these converted numbers:

(a)

$$
\begin{array}{rcl}
+29 & = & 00011101 \\
+\ -49 & = & 11001111 \\
\hline
-20 & & 11101100
\end{array}
$$

* Result is -ve. Hence find the 2's complement of result.

2's complement of 11101100 is

$$
\begin{array}{r}
00010011 \\
+\qquad\quad 1 \\
\hline
00010100
\end{array}
= 20
$$

Hence the result is -20.

(b) $(-29) + (+49)$

$$
\begin{array}{rcl}
-29 & = & 11100011 \\
+\ +49 & = & 00110001 \\
\hline
+20 & & 1\,00010100 \rightarrow (+20)
\end{array}
$$

ignore

(c) $(-29) + (-49)$

$$
\begin{array}{rcl}
-29 & = & 11100011 \\
+\ -49 & = & 11001111 \\
\hline
-\cancel{7}8 & & \boxed{1}\,10110010 \text{ - ans}
\end{array}
$$

ignore

Here the result -ve, hence find the 2's complement of the ans/result

2's complement of 10110010

$$
\begin{array}{r}
01001101 \\
+ \quad\quad 1 \\
\hline
01001110
\end{array}
$$

64 32 16 8 4 2 1

$64 + 14 = 78$

$\boxed{\text{ANS} = -78}$

Binary Subtraction

The rules for subtraction of binary numbers are again similar to decimal. When a large digit is to be subtracted from a smaller one, a 'borrow' is taken from the next column to the left. In decimal subtractions the digit 'borrowed in' is worth ten, but in binary subtractions the 'borrowed in' digit must be worth 2_{10} or binary 10_2.

Note: All the subtractions can be done with addition.

Example 1: 10101 – 00111 = 10101 + (−00111) = 21 − 7 = 14

2's complement of −00111 ⇒ 11000 + 1 = 11001

10101 + 11001 =

```
        10101
      + 11001
ignore [1] 01110   ⇒ (14)
```

Example 2: 10101 – 10111

10101 + (−10111)

2's complement of 10111 = 01000 + 1 = 01001

```
   10101
 + 01001
   11110   −ve = (−2)
```

Example 3: 1101 –(-1001)

2's complement

1101 + (+1001) =

```
        1101
      + 1001
ignore [1] 0110   (+6)   the result is overflow
```

Example 4: 1101 – (-1110)

1101 + (+1110) =

```
        1101
      + 1110
ignore [1] 1011   (−5) - result is overflow
```

Example 5: (-1101) – (-1110) in five-bit register

−1101 + (+1110) = −13 + 14 = +1

−1101 ⇒ 10011 - (2's complement of 1101)

```
   10011
 + 01110
 [1] 00001   ⇒ +1
 ignore
```

References

1. “Digital Design and Computer Architecture”, David Money Harris, Sarah L Harris, 2nd Edition, Morgan Kaufmann, 2012.

2. “Digital Design”, M Morris Mano, Michael D Ciletti, 6th Edition, Pearson, 2018.

3. https://www.javatpoint.com/boolean-functions-in-digital-electronics

4. https://learnabout-electronics.org/Digital/dig13.php