**DIGITAL DESIGN**

Let’s learn to design digital circuits, starting with a simple form of circuit:

* **COMBINATIONAL CIRCUIT**
  + Outputs depend solely on the present combination of the circuit inputs’ values
  + vs. **sequential circuit**: has “memory” that impacts outputs too, it keeps current state of the output with flip flops

Diagram

Description automatically generated

a and b 🡪 combinational c 🡪 sequential

- Switches

Electronic switches are the basis of binary digital circuits

* Diagram

  Description automatically generatedElectrical terminology
  + **Voltage**: Difference in electric potential between two points (volts, V)
    - Analogous to water pressure
  + **Resistance**: Tendency of wire to resist current flow (ohms, Ω)
    - Analogous to water pipe diameter
  + **Current**: Flow of charged particles (amps, A)
    - Analogous to water flow

Timeline

Description automatically generated

A switch has 3 parts:

* Source input, and output
  + Current tries to flow from source input to output
* Control input
  + Voltage that controls whether that current can flow

A picture containing timeline

Description automatically generatedThe amazing shrinking switch:

* 1930s: Relays
* 1940s: Vacuum tubes
* 1950s: Discrete transistor
* 1960s: Integrated circuits (ICs)
  + Initially just a few transistors on IC
  + Then tens, hundreds, thousands…

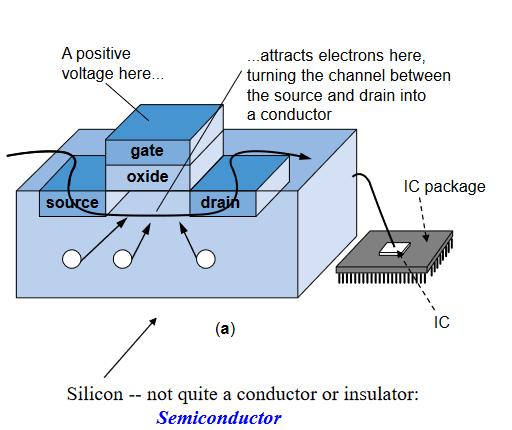
- Moore’s Law

IC capacity doubling about every 18 months for several decades

* Known as “Moore’s Law” after Gordon Moore, co-founder of Intel
  + Predicted in 1965 that components per IC would double roughly every year or so
* Today’s ICs hold billions of transistors
  + The first Pentium processor (early 1990s) needed only 3 million

- The CMOS (Complementary Metal Oxide Semiconductor) Transistor

Basic switch in modern ICs

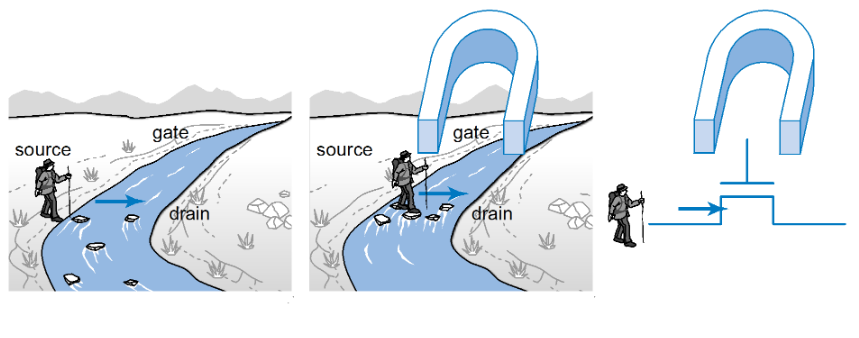
Silicon is quite important because sometimes it behaves like isolator, and sometimes it behaves like conductor.

Diagram

Description automatically generated

If you apply 1 to the gate, current is conducted from source to drain.

If you apply 0 to the gate, current is conducted from source to drain.



**BOOLEAN LOGIC GATES**

**Building Blocks for Digital Circuits**

**(Because Switches are Hard to Work With)**

Diagram

Description automatically generated

We create other components by using transistors.

Timeline

Description automatically generated- Boolean Algebra and its Relation to Digital Circuits

Developed mid-1800s by George Boole to formalize human thought.

Implement Boolean operators using transistors

* Call those implementations logic gates

A picture containing shape

Description automatically generatedChart, diagram, schematic, box and whisker chart

Description automatically generated

NOT GATE

Chart, box and whisker chart

Description automatically generated

OR GATE

Diagram

Description automatically generated

AND GATE

Diagram

Description automatically generated

F’in üstünü direkt kaldıramaz mıyız?

Hayır. Ya 0 (0 V - 0.6 V) ya 1 geçirmemiz lazım. Eğer üstü kaldırırsak ve değerler 0, 0 olursa hiçbir şey geçmez. Hiçbir şey geçmemesi 0 demek değildir. 0’ı voltajla geçirmemiz lazım, yani 0 için de elektrik lazım.

Icon

Description automatically generatedYou can do this. Can think of as AND(a, b, c).

Diagram

Description automatically generatedSeat Belt Warning Light System

s=1: seat belt fastened

k=1: key inserted

Boolean equation:

* Seat belt not fastened, and key inserted
* w = NOT(s) AND k

A picture containing diagram

Description automatically generated

Diagram, schematic

Description automatically generated

Diagram

Description automatically generated

=

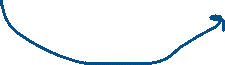
switches

logic gates

Diagram, schematic

Description automatically generated

Her logic gate bir dirençtir. Bir yerden 0 bir yerden 1 gelse kısa devre olur. 1, 0 yoluna girer.



**BOOLEAN ALGEBRA**

a AND b 🡪 a.b (product)

a OR b 🡪 a + b (sum)

NOT(a) 🡪 a’

Precedence (left to right):

1. ()
2. ‘
3. .
4. +

Terminology

F(a, b, c) 🡪 F is function; a, b, c are variables (inputs)

F(a, b, c) = a’.b.c + a.b.c’ + a.b + c

* Variables 🡪 a, b, c
* Literals (appearance of variables) 🡪 a’, a, b, c, c’
* Product term (term combined by production) 🡪 a’.b.c , a.b.c’ , a.b , c
* Sum of products (equation written as OR of product terms)
  + F(a, b, c) = a’.b.c + a.b.c’ + a.b + c
    - This one is sum of product
  + F(a, b, c) = (a + b).c + d
    - This one is not sum of product

Properties of Boolean Algebra

* a + a = a
* a.a = a
* Commutative
* Identity
  + 0 + a = a
  + 1.a = a
* Null elements
  + a + 1 = 1
  + a.0 = 0
* Complement
  + a + a’ = 1
  + a.a’ = 0
  + (a’)’ = a
  + a + b = b + a
  + a.b = b.a
* Distributive
  + a.(b + c) = a.b + a.c
  + a + (b.c) = (a + b).(a + c)
* Associate
  + (a + b) + c = a + (b + c)
  + (a.b).c = a.(b.c)

F(a, b, c) = a.b.c + a.b.c’ = a.b.(c + c’) = a.b

x + x’.z = x + (x’.z) = (x + x’).(x + z) = x + z

De Morgan’s Rule

* (a + b)’ = a’.b’
* (a.b)’ = a’ + b’

(a’ + b + c’)’ = a.b’.c

a’ + b’ + c’ = (a.b.c)’

(a + b + c)’ = (a’.b’.c’)

Example - Door opening system

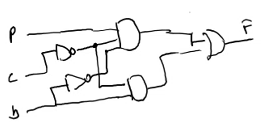
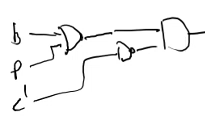
p = 1 if a person is detected

b = 1 force it to be open

c = 1 force it to be closed

F = c’.b + c’b’p

F = c’.(b + b’.p) = c’.[(b + b’).(b + p)] = c’.(b + p)



*So simplification is very important!*

Complement of a Function

F = x’.y.z + x’.y’.z 🡪 F’ = (x’.y.z)’.(x’.y’.z)’ = (x + y’ + z’).(x + y + z’)

F = x.(y’.z’ + y.z) 🡪 F’ = x’ + (y’.z’ + y.z)’ = x’ + (y’.z’)’.(y.z)’ = x’ + (y + z).(y’ + z’)

Representations of Boolean Functions

English 1

F outputs 1 when a is 0 and b is 0, or when a is 0 and b is 1

English 2

F outputs 1 when a is 0, regardless of b’s value

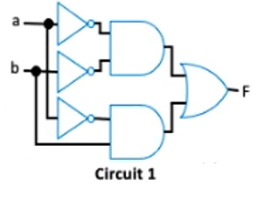
Equation 1

F(a, b) = a’.b’ + a’.b

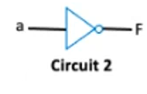
Equation 2

F(a, b) = a’

Circuit 1



Circuit 2



Truth Table

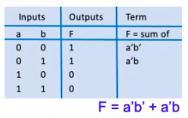


Converting Among Representations

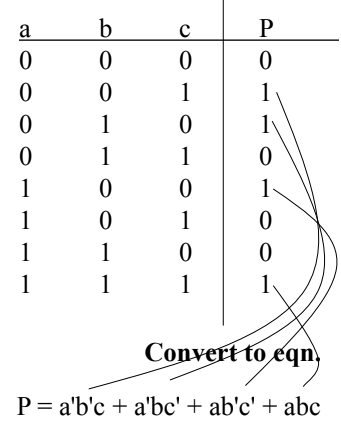
Truth table to equation

If you have truth table, first convert it to an equation (which we can then convert to circuit).

1 olan outputları bul, a ve b’den AND ile 1 elde etmeye çalış, çıkan sonuçları OR işlemine sok.

 Table

Description automatically generated



Example:

Parity bit: Extra bit added to data, intended to enable detection of error (a bit changed unintentionally)

* e.g., errors can occur on wires due to electrical interference

Even parity: Set parity bit so total number of 1s (data + parity) is even

* e.g., if data is 001, parity is 1 🡪 0011 has even number of 1s

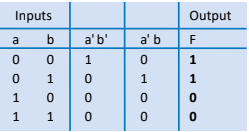
Want equation, but easiest to start from truth table for this example

Equation to truth table

Circuit to truth table

First convert to circuit to equation, then equation to table.

F = a’.b’ + a’.b

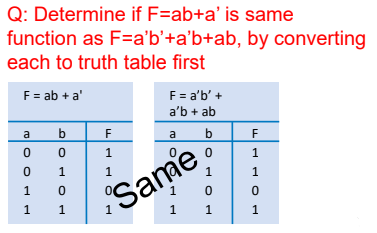


How can we determine if 2 functions are same?

Recall door opening system, more than 1 same equations

Solution: Convert to truth tables

* Only ONE truth table representation of a given function



You can put long equations to truth table and produce equation from that truth table. You may get a simpler equation.

EXAMPLE

F(a, b, c, d) = a’.b.c + a’.b.c’.d + c.b + a.c’.d + b’.c.d

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | d | a’.b.c | a’.b.c’.d | c.b | a.c’.d | b’.c.d | F |
| 0 | 0 | 0 | 0 | 0 | … | … | … | … | … |
| 0 | 0 | 0 | 1 | 0 | … | … | … | … | … |
| 0 | 0 | 1 | 0 | 0 | … | … | … | … | … |
| 0 | 0 | 1 | 1 | 0 | … | … | … | … | … |
| 0 | 1 | 0 | 0 | 0 | … | … | … | … | … |
| 0 | 1 | 0 | 1 | 0 | … | … | … | … | … |
| 0 | 1 | 1 | 0 | 1 | … | … | … | … | … |
| 0 | 1 | 1 | 1 | 1 | … | … | … | … | … |
| 1 | 0 | 0 | 0 | 0 | … | … | … | … | … |
| 1 | 0 | 0 | 1 | 0 | … | … | … | … | … |
| 1 | 0 | 1 | 0 | 0 | … | … | … | … | … |
| 1 | 0 | 1 | 1 | 0 | … | … | … | … | … |
| 1 | 1 | 0 | 0 | 0 | … | … | … | … | … |
| 1 | 1 | 0 | 1 | 0 | … | … | … | … | … |
| 1 | 1 | 1 | 0 | 0 | … | … | … | … | … |
| 1 | 1 | 1 | 1 | 0 | … | … | … | … | … |

At each product term to turn these table into equation you will have 4 variables (a, b, c, d).

Canonical Forms of the Expressions

x,y 🡪 variables, input for the circuit

You can create 4 different combinations by using AND gate:

* xy
* x’y
* xy’
* x’y’

These 4 combinations are minterms.

We can create the combinations by using OR gate:

* x + y
* x’ + y
* x + y’
* x’ + y’

These 4 combinations are maxterms.

x, y, z are variables:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | y | z | MINTERMS | DESIGNATION | MAXTERMS | DESIGNATION |
| 0 | 0 | 0 | x’y’z’ | m0 | x + y + z | M0 |
| 0 | 0 | 1 | x’y’z | m1 | x + y + z’ | M1 |
| 0 | 1 | 0 | x’yz’ | m2 | x + y’ + z | M2 |
| 0 | 1 | 1 | x’yz | m3 | x + y’ + z’ | M3 |
| 1 | 0 | 0 | xy’z’ | m4 | x’ + y + z | M4 |
| 1 | 0 | 1 | xy’z | m5 | x’ + y + z’ | M5 |
| 1 | 1 | 0 | xyz’ | m6 | x’ + y’ + z | M6 |
| 1 | 1 | 1 | xyz | m7 | x’ + y’ + z’ | M7 |

Minterms make the function 1.

Maxterms make the function 0.

(minterm)’ = maxterm 🡪 (x’y’z’)’ = x + y + z 🡪 (m0)’ = M0

Boolean algebra express a function as sum of minterms or product of maxterms.

F = x’y’z + xy’z’ + xyz = m1 + m4 + m7 = ∑ (1, 4, 7)

F(a, b, c) = ∑(1, 4, 5, 6, 7) = m1 + m4 + m5 + m6 + m7

F’(a, b, c) = ∑(0, 2, 3) = m0 + m2 + m3

[F’(a, b, c)]’ = (m0 + m2 + m3)’ = m0’m2’m3’ = M0.M2.M3 = (0, 2, 3) = F(a, b, c)



Standard Forms / Canonical Forms

In canonical forms, the terms have to include all the variables.

But in standard, it does not have to.

F(x, y, z) = xy + x’z 🡪 standard bc xy is a term and it doesn’t include all the variables.

F(x, y, z) = xyz’ + x’yz 🡪 canonical

Conversion from standard to canonical form

Can be done with 2 ways:

* boolean algebra
* truth table

Lets say function: F1 = xy + x’z

boolean algebra

F1 = xy + x’z = xy1 + x’z1 = xy(z + z’) + x’z(y + y’) = xyz + xyz’ + x’zy + x’zy’

F(x, y, z) = ∑ (1, 3, 6, 7) = (0, 2, 4, 5) 🡪 You have to know their places from the table.



truth table

xyz’

xyz

x’yz

x’y’z

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | y | z | xy | x’z | F | DESIGNATION |
| 0 | 0 | 0 | 0 | 0 | 0 | m0 |
| 0 | 0 | 1 | 0 | 1 | 1 | m1 |
| 0 | 1 | 0 | 0 | 0 | 0 | m2 |
| 0 | 1 | 1 | 0 | 1 | 1 | m3 |
| 1 | 0 | 0 | 0 | 0 | 0 | m4 |
| 1 | 0 | 1 | 0 | 0 | 0 | m5 |
| 1 | 1 | 0 | 1 | 0 | 1 | m6 |
| 1 | 1 | 1 | 1 | 0 | 1 | m7 |

F(x, y, z) = xy + x’z = x’y’z + x’yz + xyz’ + xyz = ∑ (1, 3, 6, 7)

TRUTH TABLE’A KOYARKEN İLK VARIABLE İÇİN YARISI 0, DİĞER YARISI 1; İKİNCİ İÇİN İLK YARININ YARISI 0, YARISI 1 VE İKİNCİ YARININ YARISI 0 YARISI 1 ŞEKLİNDE DEVAM ETMELİSİN.

Multiple-Output Circuits

Many circuits have more than one output

Can give each a separate circuit, or can share gates

Ex: F = ab + c’ , G = ab + bc

Diagram, schematic

Description automatically generated

Multiple-Output Example: BCD to 7-Segment Converter

Diagram

Description automatically generated

We use w, x, y, z bc 4 is the least number that we can create minimum 10 combinations.

Diagram

Description automatically generated

For remaining 6 cases, I will not show any numbers.

a = w’x’y’z’ + … 🡪 her bir term a’yı 1 yapan kombinasyonlar. tabloda gördüğün gibi a’yı 8 kombinasyon 1 yapıyor.

Combinational Logic Design Process

|  |  |
| --- | --- |
| **STEP** | **DESCRIPTION** |
| Step 1 : Capture behaviour  Capture the function | Create a truth table or equations, whichever is most natural for the given problem, to describe the desired behavior of each output of the combinational logic. |
| Step 2 : Convert to circuit  2A: Create equations  2B: Implement as a gate-based circuit | This substep is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.  For each output, create a circuit corresponding to the output’s equation. (Sharing gates among multiple outputs is OK optionally.) |

Example: Three 1s Pattern Detector

Problem: Detect three consecutive 1s in 8-bit input: abcdefgh

* 00011101 🡪 1
* 10101011 🡪 0
* 11110000 🡪 1

8 input var. Truth table ile yapsan 2^8 satır yapar. Onun için mintermler ile yapacağız.

STEP 1 : CAPTURE THE FUNCTION

* Truth table or equation?
  + Truth table is too big: 2^8 = 256 rows
  + Equation: create terms for each possible case of three consecutive 1s
* y = abc + bcd + cde + def + efg + fgh 🡪 6 minterms

Diagram, schematic

Description automatically generatedSTEP 2A : CREATE EQUATION 🡪 already done

STEP 2B : IMPLEMENT AS A GATE-BASED CIRCUIT

Example: Number of 1s Counter

Problem: Output in binary on two outputs yz the # of 1s on three inputs

* 010 🡪 01
* 101 🡪 10
* 000 🡪 00

STEP 1 : CAPTURE THE FUNCTION

* Truth table or equation?
  + Truth table is straightforward

Table

Description automatically generated

STEP 2A: CREATE EQUATIONS

* y = a’bc + ab’c + abc’ + abc
* z = a’b’c + a’bc’ + ab’c’ + abc
* Optional: Let’s simplify y:
  + y = a’bc + ab’c + ab(c’ + c) = a’bc + ab’c + ab

STEP 2B : IMPLEMENT AS A GATE-BASED CIRCUIT

Diagram, schematic

Description automatically generated

Simplifying Notations

Used in previous circuit

Diagram, schematic

Description automatically generated

List inputs multiple times

* Less wiring in drawing

Diagram, schematic

Description automatically generated

Draw inversion bubble rather than inverter. Or list input as complemented.

Example: Keypad Converter

Diagram

Description automatically generated

STEP 1 : CAPTURE BEHAVIOUR

* Truth table too big (2^7 rows); equations not clear either
* Informal table can help

A picture containing chart

Description automatically generated

None of the buttons has pressed, show nothing. That’s why we have prime of all variables.

Example: Sprinkler Controller

Microprocessor outputs which zone to water (e.g., cba = 100 means zone 6) and enables watering (e=1)

Decoder should set appropriate valve to 1.

Diagram

Description automatically generated

Decoder sets 1 of the outputs to 1, remainings are 0 at a time.

3 input yeterdi (2^3 ‘ten 8 output), neden 4 input yaptık 🡪 hiçbirinin açılmama durumu için.

Diagram

Description automatically generated

MORE GATES

A picture containing table

Description automatically generated

Chart, box and whisker chart

Description automatically generated

**AND**

**OR**

NAND same as AND with power & ground switched

* nMOS conducts 0s well, but not 1s (reasons beyond our scope) -- so NAND is more efficient

Likewise, NOR same as OR with power/ground switched

Example Uses

Aircraft lavatory sign example

* S = (abc)’

Diagram

Description automatically generated

Detecting all 0s

* Use NOR

Icon

Description automatically generated with medium confidence

Detecting equality

* Use XNOR

Diagram

Description automatically generated

Detecting odd # of 1s

* Use XOR
* Useful for generating “parity” bit common for detecting errors

Completeness of NAND

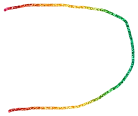
Any boolean function can be implemented using just NAND gates. Why?

* Need AND, OR, and NOT
* NOT : 1-input NAND (or 2-input NAND with inputs tied together)
* Shape

  Description automatically generated with low confidenceAND: NAND followed by NOT
* OR: NAND preceded by NOTs 🡪
* Thus, NAND is a universal gate
  + Can implement any circuit using just NAND gates

Likewise for NOR

Let’s create NOT gate with NAND gate:



Example:

F = a’b’c + a’c + bc’ --------> standard form

Express this function as a sum of minterms and product of maxterms 🡪 both canonical forms

F(a,b,c) = ∑ (1,2,3,6) 🡪 sum of minterms

F(a,b,c) = π(0,4,5,7) 🡪 product of maxterms

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | b | c | a’b’c | a’c | bc’ | F | index |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 2 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 3 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 5 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 6 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 7 |

Example:

F = AB + C

Draw that circuit by using only NAND gates.

(F’)’ = ((AB + C)’)’

F = ((AB)’.C’)’

A pair of glasses

Description automatically generated with medium confidence