Lab Work 2 - Combinational Logic Circuits

This module consists of a few parts:

1. Implementing Combinational Logic
2. Adders and Comparators
3. Other Combinational Logic Blocks

Implementing Combinational Logic

There are a few skills that are essential in digital electronics design:

- building logic circuits from schematics
- identifying the boolean equation for a logic circuit (schematic)
- simplifying a logic design (by simplifying boolean equation - algebra OR K-Map)
- drawing a logic circuit schematic from a boolean equation

If time permits, we will also look at the NAND/NOR gates as *universal* gates.

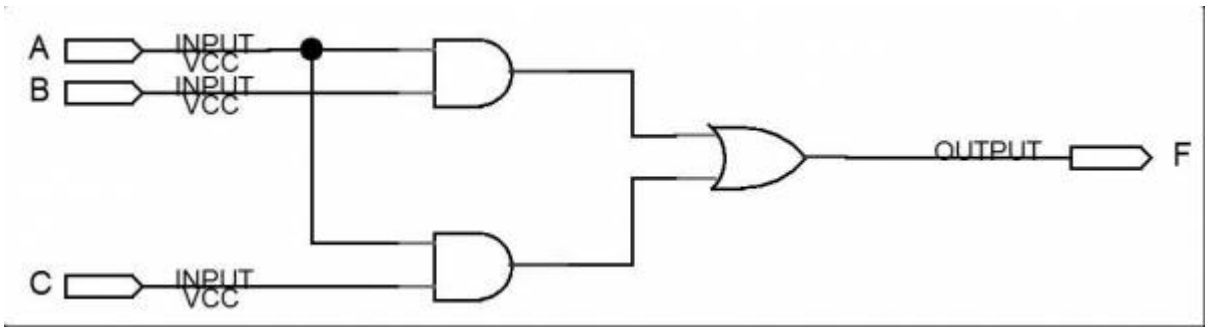
Note: After a while, you will realize that most (if not all) logic circuits can be implemented using AND-OR logic (a group of AND gates followed by a group of OR gates) implementation. In Boolean algebra, the equation for this implementation is in Sum-of-Product (SOP) form.

From Schematics to Breadboards

This is simply an extension to what have been done in previous lab sessions.

Exercise 2.1

Build the circuit shown below (Figure 2.1) on a breadboard:

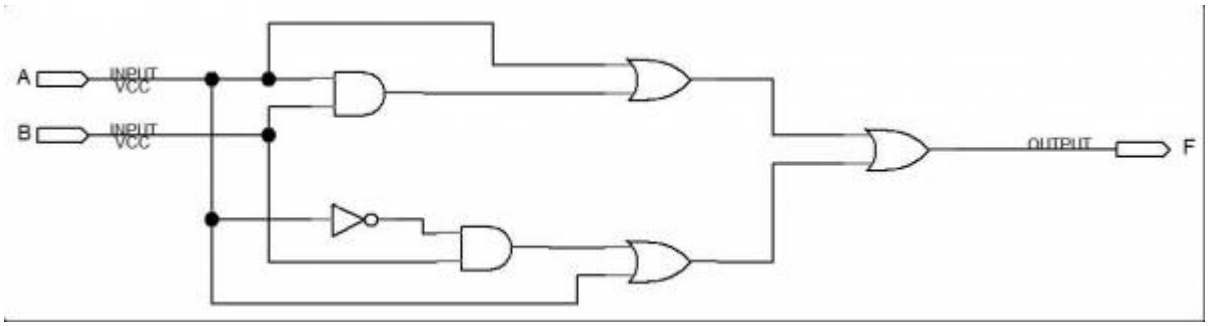


Build a truth table for the circuit.

Notice that the circuit is a direct implementation of SOP form boolean equation.

Exercise 2.2

Build the circuit shown below (Figure 2.2) on a breadboard:

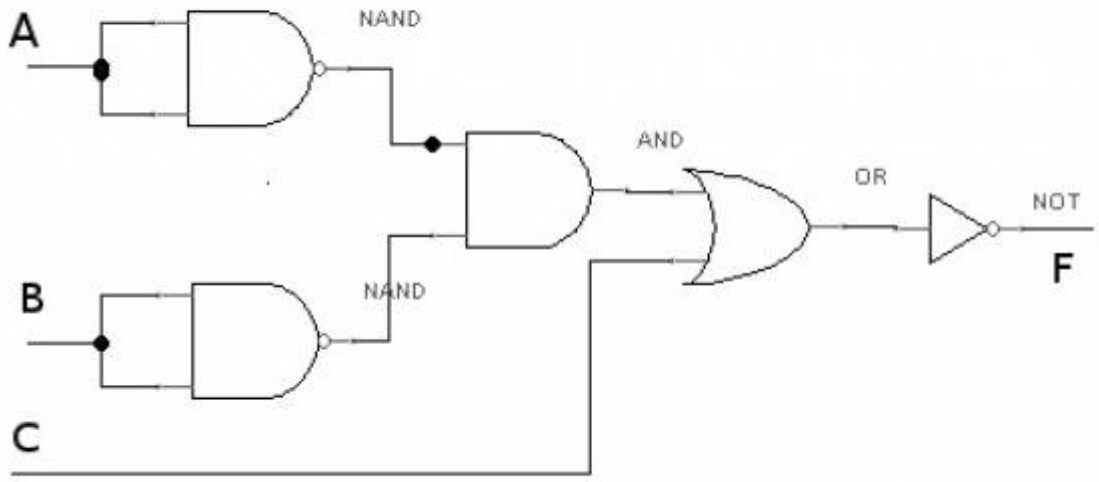


Build a truth table for the circuit.

Notice that the circuit is also a direct implementation of SOP form boolean equation.

Exercise 2.3

Build the circuit shown below (Figure 2.3) on a breadboard:



Build a truth table for the circuit.

From Schematics to Equations

Work related to this can be done on paper without having to have any hardware. However, some hardware verification is always desirable since that is what this is all about (implementing logic on

hardware).

Exercise 2.4

Derive Boolean equations for the circuits given in Figure 2.1 and Figure 2.2. You should get an equation in SOP form. For each equation,

- verify the equation using the truth table generated earlier
- rewrite the equation into a standard SOP form
- verify the 'extended' equation

Exercise 2.5

Derive Boolean equation for the circuit given in Figure 2.3.

- verify the equation using the truth table generated earlier
- try to redesign the circuit (draw schematic) so that it uses least number of ICs
- construct the new circuit and verify the design

From Equations to Schematics

The main task here is to identify the logic gates needed from a given Boolean equation.

Exercise 2.6

Consider this Boolean equation: $Y=AB+AC+A$

Simplify the equation. Build the circuit and construct a truth table.

***Note**: You can practice your knowledge by constructing standard SOP form of the given equation*

Exercise 2.7

Consider this Boolean equation: $Y=AB+BC(B+A)$

Simplify the equation. Build the circuit and construct a truth table.

***Note**: You can practice your knowledge by constructing standard SOP form of the given equation*

Exercise 2.8

Consider this truth table:

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Find the Boolean equation that satisfy the given truth table. Build the circuit and verify.

***Note**: You can practice your knowledge by trying to simplify the equation using K-Map*

Exercise 2.9

Consider this truth table:

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

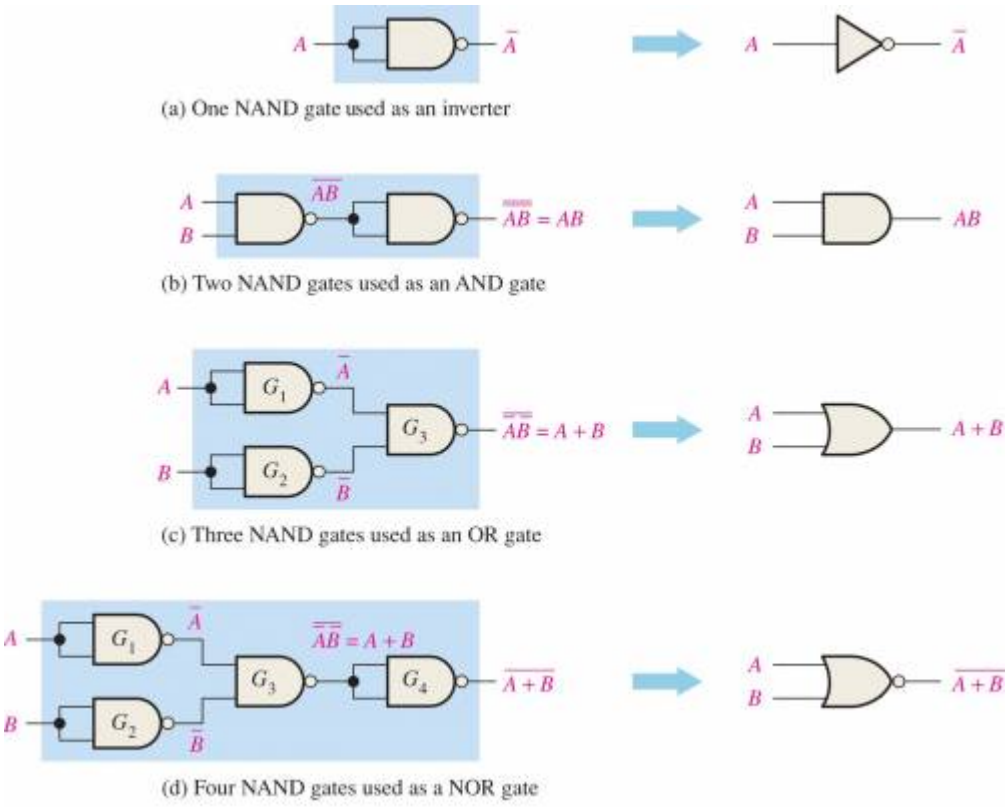
Find the Boolean equation that satisfy the given truth table. Build the circuit and verify.

***Note**: You can practice your knowledge by trying to simplify the equation using K-Map*

Universal Gates

The NAND and NOR gates are also known as universal gates due to the fact that they can be used to emulate other logic gates.

Exercise 2.10

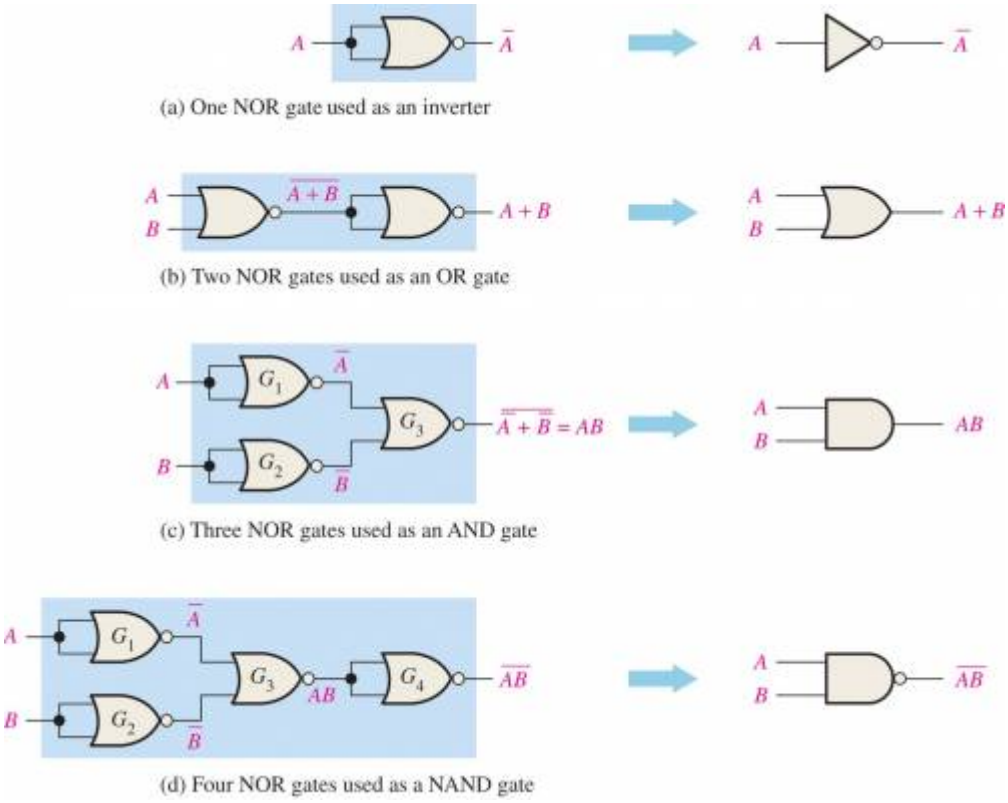


***Disclaimer**: This image is extracted from resources available for Digital Fundamentals 11th Edition (Global Edition)*

Use NAND gates to implement these equations:

- $X = \overline{A} + B$
- $X = A\ \overline{B}$

Exercise 2.11



***Disclaimer**: This image is extracted from resources available for Digital Fundamentals 11th Edition (Global Edition)*

Use NOR gates to implement these equations:

- $X=\overline{A}+B$
- $X=A\ \overline{B}$

Adders and Comparators

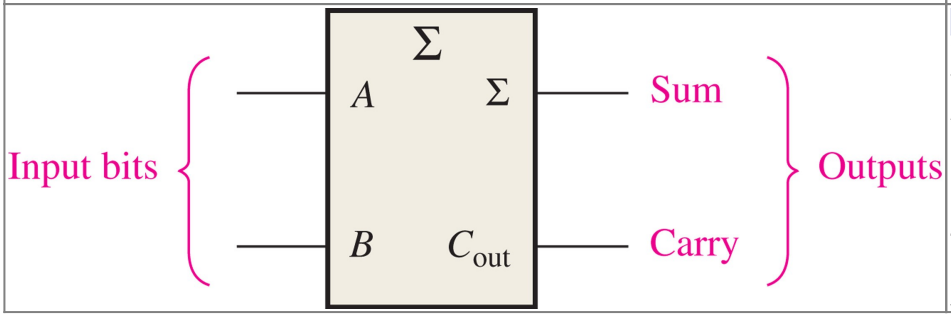
One of the most useful combinational logic circuit is an adder. It is the core component of any Arithmetic Unit - used in binary multipliers and even floating-point arithmetic units. Meanwhile, a comparator is useful as a decision making circuitry - it usually compares the magnitude of two binary values.

Half-Adder

A half-adder sums two 1-bit values and provides two 1-bit values (sum and carry).

Half-Adder

Symbol



Truth Table

TABLE 6–1

Half-adder truth table.

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

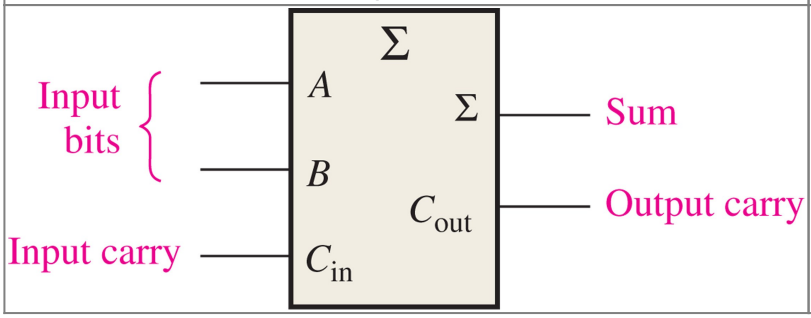
***Disclaimer**: The images above are extracted from resources available for Digital Fundamentals 11th Edition (Global Edition)*

Full-Adder

A full-adder sums three 1-bit values and provides two 1-bit values (sum and carry).

Full-Adder

Symbol



Truth Table

TABLE 6–2

Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C_{in} = input carry, sometimes designated as *CI*
C_{out} = output carry, sometimes designated as *CO*
Σ = sum
A and B = input variables (operands)

***Disclaimer**: The images above are extracted from resources available for Digital Fundamentals 11th Edition (Global Edition)*

Comparator

There are three possible output bits of a comparator (depending on application requirement): equality (==), less than (<) and greater than (>).

Comparator Output	Description
EQ (==)	Output is at logic HI when the first value is exactly the same as the second value
LT (<)	Output is at logic HI when the first value is less than the second value
GT (>)	Output is at logic HI when the first value is greater then the second value

Truth Table for a 1-bit Comparator:

A	B	EQ	LT	GT
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

***Note**: A 2-bit comparator cannot be built by simply cascading two 1-bit logic circuits.*

Things To Do

THING 1 Build a 1-bit half-adder circuit and verify.

THING 2 Build a 1-bit full-adder circuit using 2×1-bit half-adders. Verify. *Trivia: What is the least number of ICs (of 2-input logic gates) needed to implement this?*

THING 3 (Optional?) Build a 2-bit adder and verify.

THING 4 (Optional?) Construct a truth table for 1-bit subtractor. Build the circuit and verify.

THING 5 (Optional?) Build a 4-bit ripple carry adder and verify.

THING 6 (Optional?) Build a 4-bit carry look ahead (CLA) adder and verify.

THING 7 Construct a truth table for 2-bit comparator (3 outputs). Get the Boolean expression for each output. Build the circuit and verify.

THING 8 (Optional?) Build a 4-bit comparator (3 outputs) and verify.

ask your instructor for more...

Other Combinational Logic Blocks

In this section, we are going to look at other commonly-used combinational logic circuits (decoder, encoder, multiplexer and demultiplexer.

Decoder

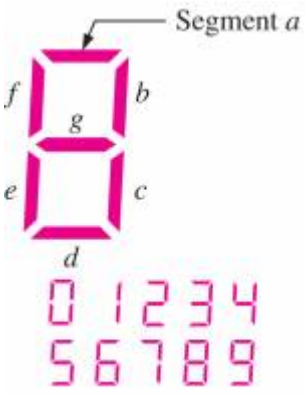
A decoder is supposed to detect/determine specific bit combinations (code). For an n-bit binary code, there can be up to 2^n combinations (thus, as many outputs). A common decoder is usually known as an $nto2^n$ decoder (n input, 2^n output).

Truth Table for a 3-8 Decoder (active HI output):

A_2	A_1	A_0	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

7-segment Display

This is a 'classic' device used to display decimal digits 0 to 9. It consists of 7 LED segments (hence the name) arranged as such to enable it display decimal digits (***Note**: It can also been used to display certain alphabets/letters*). To display binary or BCD value, a decoder is required to 'convert' the value into an output that can be used to drive the 7-segment.

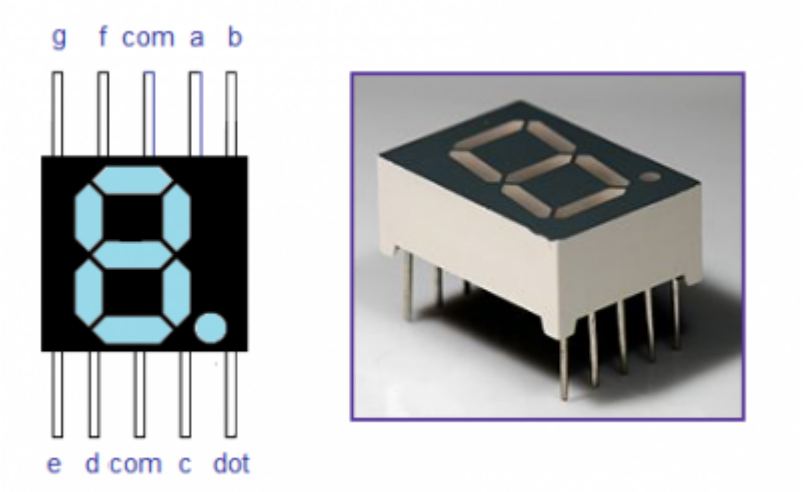


***Disclaimer**: The image above is extracted from resources available for Digital Fundamentals 11th Edition (Global Edition)*

Truth Table for a 7-segment Decoder (active HI output):

A_3	A_2	A_1	A_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

There are two types of 7-segments: common anode and common cathode. A common anode 7-segment has the anode terminals of all LED in the package connected the the COM (common) pin. The same goes for common cathode.



***Disclaimer**: The image above is taken from http://www.circuitstoday.com/interfacing-seven-segment-display-to-8051. I will remove the image if the copyright owner asks me to do so.*

Encoder

An encoder does the *inverse* decoder function. Therefore, it usually accepts a group of bits that has only 1 bit active at a time to represent a specific value or pattern. This can be converted by the encoder into a coded format (e.g. binary or BCD).

Notice that an encoded form usually has lower number of bits, so it can also be seen as a 'compression' function.

Truth Table for a BCD Encoder:

Decimal Digit	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Multiplexer

A multiplexer is a selection logic block. Multiple input lines can be selected to drive a single output line. Since the selector signal is in binary form, a multiplexer is usually found as a 2^n to 1 selection block (where n is the number of selector bits and $n>0$).

Truth Table for a 4-1 Multiplexer:

Selector Bits		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Note that in the above truth table, data inputs (D_3-D_0)have been 'compressed'. If each of the 4 input bits is listed with all possibilities, we would need a 64-row table!

Demultiplexer

A demultiplexer is the inverse of a multiplexer (*duh!*). A single source signal can be routed to any one of multiple output lines, depending on the selector signal. One thing should be noted here is that a decoder can actually be used as a demultiplexer!

Things To Do

THING 1 Build a truth table for 2-4 decoder. Build the logic circuit and verify.

THING 2 Get the Boolean expression for a 7-segment decoder assuming we need to display **the numbers 0 to 3 only**. Build the logic circuit and verify.

THING 3 Build the logic circuit for a 4-1 multiplexer and verify.

THING 4 (Optional) Repeat **THING 2** so that it can display the full range 0-9.

THING 5 (Optional) Build a logic circuit to implement 1-4 demultiplexer. Feed a 1kHz TTL signal to the input. Verify that the output is visible on the selected output channel. ***Hint****: The circuit is available in the textbook!*

THING X (Optional) Consider using tristate buffer(s) in (de-)multiplexer circuits. What is the (dis-)advantage(s) of using this implementation?