Lec 22:
Boolean Algebra II

Prof. Adam J. Aviv

GW

CSCI 1311 Discrete Structures I
Spring 2020

Digital Logic and Circuits

Digital Logic

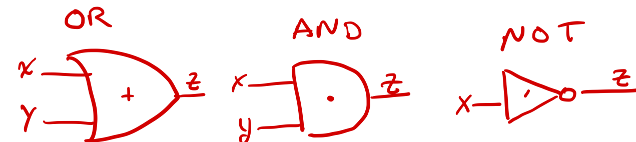
Recall, we've been discussing the Boolean Algebra of Digital Logic:

- $B = \{0, 1\}$
- $+$: logical OR
- $\cdot$: logical AND
- $'$: logical NOT (or negation)
- 1 : 1 (logical true)
- 0 : 0 (logical false)

An important feature of Digital Logic useful is that the set $B = \{0, 1\}$ has only two elements, 0 and 1, which are also the identities.

Digital Circuits

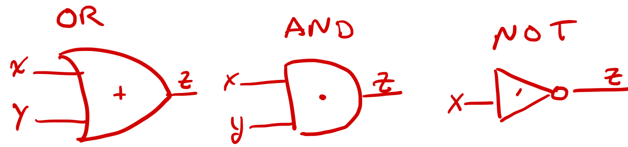
A Digital Circuit is a representation of digital logic where wires represent Boolean values, either a 0 or a 1, connected to a series of gates, representing Boolean operations.



The three primary gates are OR, AND, and NOT gates.

Gates

The gates of digital logic gates follow the same truth tables



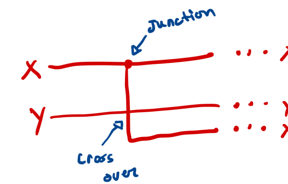
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

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

x	z
0	1
1	0

Wires

The wires of a digital circuit carry bits of information as either inputs or outputs of gates.

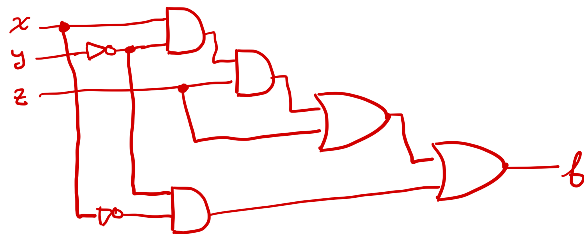


A wire must sometimes cross over each other or split for visualization purposes. If there is a junction, where the wire splits, we indicate that with a "dot", and if there is a cross over, there will be no "dot."

Exercise

Draw the circuit for the following Boolean function.
(***Note, I am simplifying*** $x \cdot y$ ***to*** xy ***for brevity and readability***)

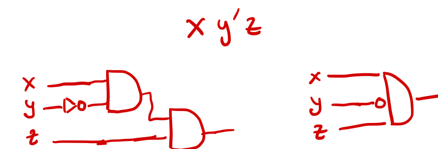
$$f(x, y, z) = xy'z + z + x'y'$$



Simplifying Circuits

Since AND and OR are associative, we can simplify the visual of the circuits when AND's or OR's are in sequence.

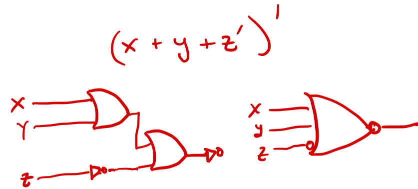
This logical AND gate (really comprised of two AND gates) takes three inputs where the output is the product.



Also, we can add a bubble (the ∘) to y to indicate that the complement is considered before taking the product.

NAND and NOR Gates

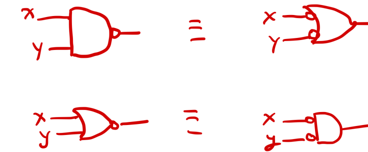
We can also simplify the complement of output of a gate



We put a bubble on the output of gate, it becomes an "N"-gate. In the above, this is a NOR gate. There is also a NAND gate.

Bubble Pushing and DeMorgan's Law

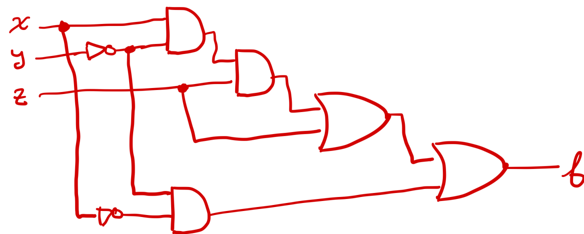
DeMorgan's law also applies to Digital Circuits, and you can visualize DeMorgan's law by "pushing bubles" through a gate.



If the bubble pushes from the output of the gate towards its input, then it flips ANDs to ORs (or vice-versa) and puts bubbles on the input. If the bubble pushes in the other direction, from the input side, it flips the gate, placing a bubble on the output creating a NAND or NOR gate.

Exercise

Simplify the following circuit by combining all AND and OR's in sequence and adding bubbles.



Use DeMorgan's law and bubble pushing to flip all the gates to NAND and NOR gates.

Circuits from Truth Tables

State Machine

Suppose we wish to build circuits that embody some sort of programmatic logic, like counting using two bits: a_1, a_0, where each variable represents the bits of a binary number:

a_1	a_0	
0	0	$\equiv 0$
0	1	$\equiv 1$
1	0	$\equiv 2$
1	1	$\equiv 3$

Counting would add 1, and then cycle back around to 0 after adding 1 to 3.

$$0 \to 1 \to 2 \to 3 \to 0 \to 1 \to 2 \to 0 \dots$$

Counting Digital Logic

How can we encode counting in digital logic?

$$00 \to 01 \to 10 \to 11 \to 00 \to 01 \to 10 \to 11 \to 00 \dots$$

Consider two boolean functions that produce the next bits from the current one

$$f_0(a_0, a_1) \to b_0$$
$$f_1(a_0, a_1) \to b_1$$

Where given the input state a_0, a_1, we learn the next state b_0, b_1.

Truth table of state transition

a_1	a_0	f_0
0	0	1
0	1	0
1	0	1
1	1	0

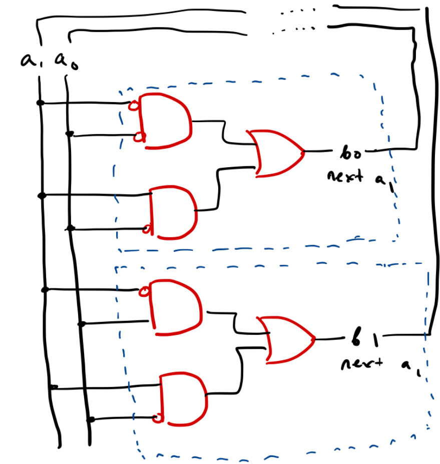
a_1	a_0	f_1
0	0	0
0	1	1
1	0	1
1	1	0

And we can write the DNF of f_0 and f_1 from the truth table

$$f_0(a_0, a_1) = a_1' a_0' + a_1 a_0'$$
$$f_1(a_0, a_1) = a_1' a_0 + a_1 a_0'$$

Two bit counter

Combining the two boolean functions, we can create a complete two bit adder circuit



Exercise

Consider a three bit (a_2, a_1, a_0)adding machine that always adds 2 to its input. That is the following sequences are encoded.

$$000 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 000 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 000 \ldots$$

$$001 \rightarrow 011 \rightarrow 101 \rightarrow 111 \rightarrow 001 \rightarrow 011 \rightarrow 101 \rightarrow 111 \rightarrow 001 \ldots$$

Derive the boolean functions f_2, f_1, f_0 that describe the bits of the next state in the sequence.

Solution

a_2	a_1	a_0	f_0
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	1

a_2	a_1	a_0	f_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

a_2	a_1	a_0	f_2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$f_0(a_2, a_1, a_0) = a_2'a_1'a_0 + a_2'a_1a_0 + a_2a_1'a_0 + a_2a_1a_0$$
$$f_1(a_2, a_1, a_0) = a_2'a_1'a_0' + a_2'a_1'a_0 + a_2a_1'a_0' + a_2a_1'a_0$$
$$f_2(a_2, a_1, a_0) = a_2'a_1a_0' + a_2'a_1a_0 + a_2a_1'a_0' + a_2a_1'a_0$$

Karnaugh Maps

By-2 three-bit adder

$$000 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 000 \rightarrow 010 \rightarrow 100 \rightarrow 110 \rightarrow 000 \ldots$$
$$001 \rightarrow 011 \rightarrow 101 \rightarrow 111 \rightarrow 001 \rightarrow 011 \rightarrow 101 \rightarrow 111 \rightarrow 001 \ldots$$

a_2	a_1	a_0	f_0
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	1

a_2	a_1	a_0	f_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

a_2	a_1	a_0	f_2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$f_0(a_2, a_1, a_0) = a_2'a_1'a_0 + a_2'a_1a_0 + a_2a_1'a_0 + a_2a_1a_0$$
$$f_1(a_2, a_1, a_0) = a_2'a_1'a_0' + a_2'a_1'a_0 + a_2a_1'a_0' + a_2a_1'a_0$$
$$f_2(a_2, a_1, a_0) = a_2'a_1a_0' + a_2'a_1a_0 + a_2a_1'a_0' + a_2a_1'a_0$$

Simpler expressions?

If we "squint" at the truth table, much simpler Boolean expressions become apparent.

a_2	a_1	a_0	f_0
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	1

a_2	a_1	a_0	f_1
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

a_2	a_1	a_0	f_2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$\mathbf{f_0(a_2, a_1, a_0) = a_0}$$
$$\mathbf{f_1(a_2, a_1, a_0) = a_1'}$$
$$\mathbf{f_2(a_2, a_1, a_0) = a_2'a_1 + a_2a_1'}$$

Gates = Performance = Money+

It's important to have simplify logical expressions.

The more gates, the slower a process can take, and the more money it would take to make the hardware.

We should strive to simplified expressions where we can.

Simplifying Expressions via Equivalences

Because we know that both expressions share the same truth table, we should be able to reduce one into the other

$$\begin{aligned}
f_0(a_2, a_1, a_0) &= a_2'a_1'a_0 + a_2'a_1a_0 + a_2a_1'a_0 + a_2a_1a_0 & \\
&= a_2'a_0(a_1' + a_1) + a_2a_0(a_1' + a_1) & \text{Distributive} \\
&= a_2'a_0(1) + a_2a_0(1) & \text{Complement Law} \\
&= a_2'a_0 + a_2a_0 & \text{Identity} \\
&= a_0(a_2' + a_2) & \text{Distributive} \\
&= a_0(1) & \text{Complement Law} \\
&= \boxed{a_0} & \text{Identity}
\end{aligned}$$

Exercise

Simplify the following expressions

$$f_1(a_2, a_1, a_0) = a_2'a_1'a_0' + a_2'a_1'a_0 + a_2a_1'a_0' + a_2a_1'a_0$$

$$f_2(a_2, a_1, a_0) = a_2'a_1a_0' + a_2'a_1a_0 + a_2a_1'a_0' + a_2a_1'a_0$$

(Un)Distribute, Complement, Identity, and Repeat . . .

If we have an expression in DNF, we can find reductions/simplifications:

- (un)distributing variables
- leave two complements
- cancel the complements

Can we visualize this process?

Karnaugh Maps

a_2	a_1	a_0	f_0
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	1

		11 a_2a_1	01 $a_2'a_1$	00 $a_2'a_1'$	01 a_2a_1'
1	a_0	1	1	1	1
0	a_0'	0	0	0	0

In the K-Map, every adjacent cell (with wrap around) can cancel out. We seek out a covering of 1's to reveal cancellations.

The goal is to cover all the 1's to derive the simplified expression

Rules of K-Map Coverings

1. Each covering corresponds to a product term in the result. The final formula is the sum of product terms.
2. All 1's must be covered.
3. A cover must contain an even number of cells, or just a single cell
4. Covers can overlap
5. The goal is to cover all the 1's with as few of covers as possible.

Sample Coverings (1)

Vertical Cover: Doesn't z and z' will cancel: xy

		11 xy	01 $x'y$	00 $x'y'$	01 xy'
1	z	1	0	1	1
0	z'	1	0	0	0

Horizontal Cover: Adjacent complements (x' and x) cancel: zy'

Sample Coverings (2)

Coverings can wrap around edges. This is a horizontal covering where y' and y cancel: xz

		11 xy	01 $x'y$	00 $x'y'$	01 xy'
1	z	1	0	0	1
0	z'	0	0	1	0

A single 1 is covered alone if it cannot be covered with a neighbor: x'y'z'

Sample Coverings (3)

In a group of four, you can do two cancellation.

		11 xy	01 $x'y$	00 $x'y'$	01 xy'
1	z	1	1	1	1
0	z'	0	1	1	0

4x Horizontal Cover: x cancels x' and y cancles y', leaving z

4x Vertical Cover: z cancels z' and y' cancels y', leaving x'

Cover's can overlap!

Exercise

Simplify the following expression using a Karnaugh Map

$$f(x, y, z) = x'yz + x'y'z + xy'z + x'y'z'$$

Four Way K-Map

Same rules apply in four way, with two variables per column/row

		11 xy	01 $x'y$	00 $x'y'$	01 xy'
11	zw	0	0	1	1
01	$z'w$	0	1	1	1
00	$z'w'$	0	1	1	0
10	$z'w'$	1	0	0	1

$$f(x, y, z, w) = xz'w + wy' + x'z'$$

Exercise

Find a simplified expression using a K-map for the following truth table

x	y	z	w	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0