CS3350B Computer Organization
Chapter 2: Synchronous Circuits
Part 1: Gates, Switches, and Boolean Algebra

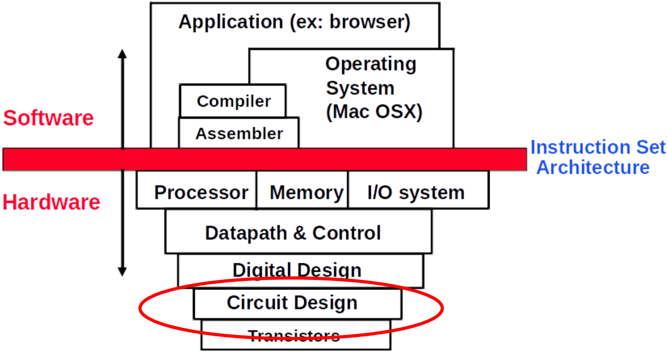
Iqra Batool

Department of Computer Science
University of Western Ontario, Canada

Monday January 29, 2024

Outline

Layers of Abstraction



After looking at high-level CPU and Memory we will now go down to the lowest level (that we care about).

Circuit Design vs Digital (Logic) Design

↳ Design of individual circuits vs Using circuits to implement some logic.

Circuit Design

Why do we care?

- Appreciate the limitations of hardware.
- Understand why some things are fast and some things are slow.
- Need circuit design to understand logic design.
- Need logic design to understand **CPU Datapath**.

If you are ever working with:

- Assembly, ISAs,
- Embedded Systems and circuits,
- Specialized computer/logic systems,

you will need circuit and logic design.

Digital Circuits

Everything is **digital:** represented by discrete, individual values.

↳ No gray areas or ambiguity.

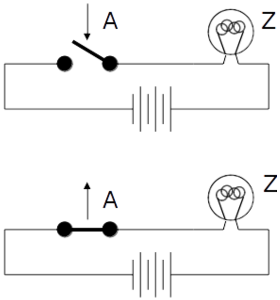
Must convert an **analog** – continuously variable – signal to digital.

For us, the analog signal is electricity (voltage).

↳ "High" voltage ⇒ 1

↳ "Low" voltage ⇒ 0

Physicality of Circuits



In the end, everything is a switch.

"Input" $\Rightarrow$ A
"Output" $\Rightarrow$ Z

If A is 0/false then switch is open.
If A is 1/true then switch is closed.

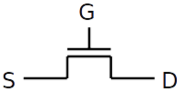
This circuit implements:

$$\mathbf{A} \equiv \mathbf{Z}$$

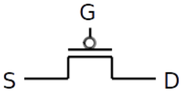
Transistors: Electrically Controlled Switches

MOS-FET: Metal-Oxide-Semiconductor Field-Effect **Transistor**

- Has a source (S), a drain (D), and a gate (G).
- Applying voltage to G allows current to flow between S and D.
- In reality, transistors, logic gates, SRAM, use CMOS (Complimentary-MOS). But we don't care about transistors really...



n-channel
opens when voltage at G is low,
closes when voltage at G is high



p-channel
closes when voltage at G is low,
opens when voltage at G is high

Flipping a transistor is *much faster* than moving a physical switch.

↳ Speed of switching a transistor directly related to speed of a CPU

Outline

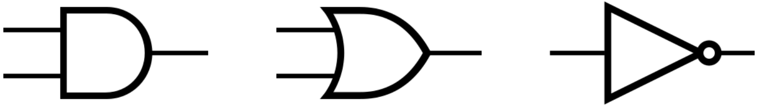
Logic as Circuits

Propositional Logic: A set of propositions (truth values) combined by some logical connectives.

- Truth values ≡ Binary digital signal
- Logical connectives ≡ **Logic gates**

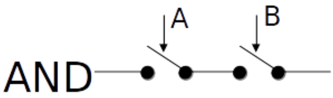
Logic Gate: A circuit implementing some logical expression/function.

The basics: **AND** ($\wedge$), **OR** ($\vee$), **NOT** ($\neg$).

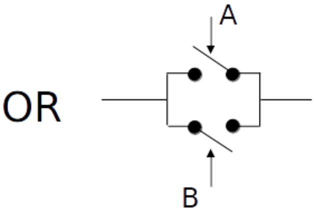


Arity of a function/gate is the number of inputs.

Gates as Switches

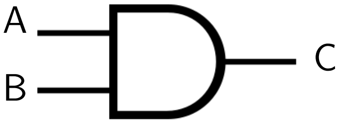


- Both A and B must be true/1 to get the circuit to complete.



- Either A or B can be true/1 to get the circuit to complete.

Logic Gates In Detail: AND



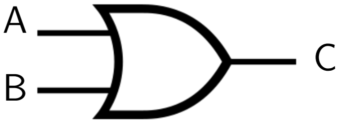
$$A \wedge B \equiv C$$

$$A \cdot B \equiv C$$

Truth Table for AND

A	B	$A \wedge B \equiv C$
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates In Detail: OR



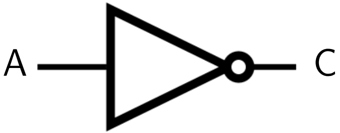
$$A \vee B \equiv C$$

$$A + B \equiv C$$

Truth Table for OR

A	B	$A \vee B \equiv C$
0	0	0
0	1	1
1	0	1
1	1	1

Logic Gates In Detail: NOT



$$\neg A \equiv C$$

$$\overline{A} \equiv C$$

Truth Table for NOT

A	$\neg A \equiv C$
0	1
1	0

More Interesting Logic Gates: NAND

$$\neg(A \wedge B) \equiv C$$

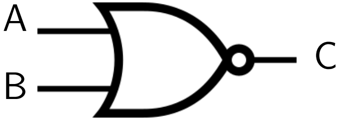
$$\overline{A \cdot B} \equiv C$$

$$A \mid B$$

Truth Table for NAND

A	B	$\overline{A \cdot B} \equiv C$
0	0	1
0	1	1
1	0	1
1	1	0

More Interesting Logic Gates: NOR



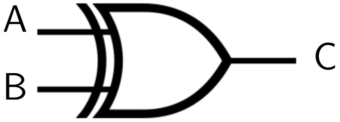
$$\neg(A \vee B) \equiv C$$

$$\overline{A + B} \equiv C$$

Truth Table for NOR

A	B	$\overline{A + B} \equiv C$
0	0	1
0	1	0
1	0	0
1	1	0

More Interesting Logic Gates: XOR (Exclusive OR)



$A \oplus B \equiv C$

Truth Table for XOR

A	B	$A \oplus B \equiv C$
0	0	0
0	1	1
1	0	1
1	1	0

Outline

The Algebra of Logic Gates

Due to the equivalence of truth values and binary digital signals, **Boolean Algebra** is heavily used discussing circuitry.

Associativity:

$$(A + B) + C \equiv A + (B + C)$$
$$(A \cdot B) \cdot C \equiv A \cdot (B \cdot C)$$

Commutativity:

$$A + B \equiv B + A$$
$$A \cdot B \equiv B \cdot A$$

Distributivity:

$$A + (B \cdot C) \equiv (A + B) \cdot (A + C)$$
$$A \cdot (B + C) \equiv (A \cdot B) + (A \cdot C)$$

Identity:

$$A + 0 \equiv A$$
$$A \cdot 1 \equiv A$$

Annihilation:

$$A + 1 \equiv 1$$
$$A \cdot 0 \equiv 0$$

Idempotence:

$$A + A \equiv A$$
$$A \cdot A \equiv A$$

Boolean Algebra: More Interesting Laws

Absorption:

$$A \cdot (A + B) \equiv A$$
$$A + (A \cdot B) \equiv A$$

Double Negation

$$\overline{\overline{A}} \equiv A$$

Complementation:

$$A + \overline{A} \equiv 1$$
$$A \cdot \overline{A} \equiv 0$$

De Morgan's Laws:

$$\overline{A + B} \equiv \overline{A} \cdot \overline{B}$$
$$\overline{A \cdot B} \equiv \overline{A} + \overline{B}$$

Look familiar?
↳ Definitions of NOR and NAND.

Proving De Morgan's Laws

Proof by Exhaustion:

- The easiest way to prove something is to write out each expression's truth table.

$$\overline{A + B} \equiv \overline{A} \cdot \overline{B}$$

A	B	A + B	$\overline{A+B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Simplifying Expressions with Boolean Algebra (1/2)

$$\overline{xyz} + \overline{xy}z$$

$$\begin{aligned} \overline{xyz} + \overline{xy}z &\equiv \overline{xy}(\overline{z} + z) && \text{Factor } \overline{xy} \\ &\equiv \overline{xy}(1) && \text{Complementation of } z \\ &\equiv \overline{xy} && \text{Identity with } \overline{xy} \end{aligned}$$

x	y	z	$\overline{xyz}$	$\overline{xy}z$	$\overline{xyz} + \overline{xy}z$
0	0	0	1	0	1
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

‘

Note: $\overline{AB} \implies \overline{A} \cdot \overline{B}$; otherwise use $\overline{A \cdot B}$ or $\overline{(A\,B)}$ for $A \mid B$.

Simplifying Expressions with Boolean Algebra (2/2)

Sometimes a truth table is too challenging...

↳ For v variables a truth table has 2^v rows.

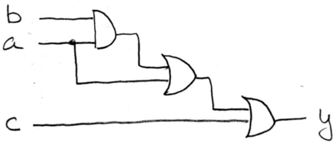
$$\overline{(\overline{x} + \overline{z})}\,(abcd + xz) \implies \text{6 variables, 64 rows}$$

Instead we can simplify using the laws of Boolean algebra:

$$\begin{aligned}
\overline{(\overline{x} + \overline{z})}\,(abcd + xz) &\equiv \overline{\overline{x}}\,\overline{\overline{z}}\,(abcd + xz) && \text{De Morgan's Law} \\
&\equiv xz\,(abcd + xz) && \text{Double negation of } x \text{ and } z \\
&\equiv xz && \text{Absorption}
\end{aligned}$$

Simplifying Expressions for Simplified Circuits

$$y = ((ab) + a) + c$$



$$
\begin{aligned}
y &\equiv (ab + a) + c & \\
&\equiv a\,(b + 1) + c & \text{Factor } a \\
&\equiv a\,(1) + c & \text{Annihilaltion} \\
&\equiv a + c & \text{Identity}
\end{aligned}
$$

$\Downarrow$

Canonical Forms

Different standard or **canonical forms**.

- **Conjunctive Normal Form** (CNF) ⇒ AND of ORs
  - ↳ "Product-of-sums"
- **Disjunctive Normal Form** (DNF) ⇒ ORs of ANDs
  - ↳ "Sum-of-products"

CNF $(a+b)\cdot(\overline{a}+b)\cdot(\overline{a}+\overline{b})$

DNF $ab+\overline{a}b+\overline{a}\overline{b}$

- Every variable should appear in every sub-expression.
  - ↳ Products for DNF, Sums for CNF.
  - ↳ Some authors call this "Full DNF" or "Full CNF".
- Every boolean expression can be converted to a canonical form.
- DNF more useful and practical ⇒ truth tables.

Truth Tables and Disjunctive Normal Forms

We can get a DNF expression directly from a truth table.

- a, b, c are inputs, f is output.
- Create one product term for every entry in the table with $f \equiv 1$.
- Put $\overline{x}$ in product if x is false in that row.
- Put x in product if x is true in that row.
- OR all products together.

a	b	c	f
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	0
1	1	1	1

$$\Longrightarrow \qquad \overline{abc} + \overline{a}b\overline{c} + a\overline{bc} + abc$$

Functional Completeness

Functional Completeness - A set of functions (operators) which can adequately describe every operation and outcome in an algebra.

- For Boolean algebra the classical set of operators: $\{+, \cdot, \neg\}$ is functionally complete but not **minimal**.
- Thanks to De Morgan's Law we only need one of AND or OR.
- The sets $\{+, \neg\}$ and $\{\cdot, \neg\}$ are both functionally complete and minimal.
  - ↳ **minimal** - removing any one of the operators would make the set functionally *incomplete*.
- NAND alone is functionally complete; so is NOR alone.

NAND is Functionally Complete

NAND alone is functionally complete.

- NAND $\equiv\ |$
- To prove functional completeness simply show that the operators of the set can mimic the functionality of the set $\{+, \cdot, \neg\}$.

X	$\overline{X}$	$X \cdot X$	$\overline{X \cdot X}$
0	1	0	1
1	0	1	0

$\neg X \equiv X \mid X$

X	Y	$A \equiv X \vert Y$	$A \vert A$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

$X \cdot Y \equiv \overline{X|Y} \equiv (X|Y) \mid (X|Y)$

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X} \vert \overline{Y}$
0	0	1	1	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1

$X + Y \equiv \overline{\overline{X + Y}} \equiv \overline{\overline{X} \cdot \overline{Y}} \equiv (X|X) \mid (Y|Y)$

Summary

Boolean algebra can simplify circuits.

- Remove variables that the output does not depend on.
- Simplifies expression, removing needless gates.
- Space and time complexity improved!

Truth tables, canonical forms, functional completeness.

Help generating truth tables:

- `https://web.stanford.edu/class/cs103/tools/truth-table-tool/`