Lecture 2 — Intro to Logic Gates

Table of contents

Outline . . . 1
Digital logic . . . 2
Worked: 3-input NAND truth table . . . 2
Worked: 3-input XOR truth table . . . 3
Boolean logic functions . . . 3
Four representations of a logic function . . . 4
Logic diagram conventions . . . 4
Gates on ICs . . . 5
Logic function implementation (sum of products) . . . 5
Equivalent functions (simplification) . . . 6
De Morgan's law and equivalent circuits . . . 8
Universal (complete) gates . . . 8
Reminders . . . 8

See logic-gates[1] and boolean-algebra[2] for the reference definitions/tables.

Outline

- Introduction to Logic Gates
- Logic Diagrams
- Boolean Algebra and Logic Expressions

[1] courses/csse2010/logic-gates.pdf
[2] courses/csse2010/boolean-algebra.pdf

Digital logic

This is **level 0**, the Digital Logic level, of the abstraction stack from Lecture 1 (see 2026-07-27-course-intro-bits-bytes-binary[3]) — the bottom of the stack, and where the course starts.

- Digital circuits only have **two** logical levels present (binary):
  - logic '0' — usually a small voltage (e.g. around 0 volts);
  - logic '1' — usually a larger voltage (e.g. 0.8 to 5 volts, depending on the **logic family**, i.e. the type/size of the transistors).
- **Logic gates** are the building blocks of computers. Each gate has **one or more inputs** and **exactly one output**, and performs a logic operation (or function). Inputs and outputs can only be in two states, 1 & 0 — also called "true" & "false".
- 7 basic gate types: **NOT, AND, OR, NAND, NOR, XOR, XNOR** — see logic-gates[4] for symbols and truth tables.
- Every gate (and every logic function) can be described four equivalent ways: **logic symbol, truth table, Boolean expression, timing diagram**.

Filling in truth tables

Two habits:

- A 2-input gate's table has $2 \times 2 = 4$ rows; n inputs give 2^n rows.
- Don't memorise NAND/NOR/XNOR. Write the **AND / OR / XOR column first**, then invert it row by row — that is all the "N" prefix means. (Both worked questions below are done this way.)

Worked: 3-input NAND truth table

Which truth table is the 3-input NAND?

Write the 3-input **AND** column first — 0 everywhere except the last row — then invert it. The NAND is the table where $X = 1$ for every combination **except** $A = B = C = 1$:

A	B	C	AND	**NAND**
0	0	0	0	**1**
0	0	1	0	**1**
0	1	0	0	**1**
0	1	1	0	**1**

[3] courses/csse2010/2026-07-27-course-intro-bits-bytes-binary.pdf
[4] courses/csse2010/logic-gates.pdf

A	B	C	AND	**NAND**
1	0	0	0	**1**
1	0	1	0	**1**
1	1	0	0	**1**
1	1	1	1	**0**

The "output is HIGH when at least one input is LOW" wording carries over unchanged from two inputs to three.

Worked: 3-input XOR truth table

XOR is the **odd function** — the 2-input wording "exactly one input is HIGH" does *not* generalise; the correct statement is that the output is HIGH when an **odd number** of inputs are HIGH.

So the rows that are 1 are those with one or three 1s — $001, 010, 100, 111$:

A	B	C	**XOR**
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

Correspondingly XNOR is the **even function**.

Boolean logic functions

A logic function is written as an expression of **variables** (literals, e.g. A, B, X) combined by **functions** — $+$ (OR), $\cdot$ (AND), $\oplus$ (XOR), $^{-}$ (NOT) — with the output given as a function of the inputs, $X = f(A, B)$. Variables and functions can only take on the values **0** or **1**, and the rules for manipulating such expressions are **Boolean algebra**.

See boolean-algebra[5] for the full notation conventions, the laws/identities table, De Morgan's law and its gate equivalents, and the worked simplification example.

[5] courses/csse2010/boolean-algebra.pdf

Four representations of a logic function

For a function of n inputs:

1. **Truth table** — lists the output for all 2^n combinations of the inputs. Best to list the input combinations in a systematic way (i.e. counting up in binary).
2. **Boolean function** (or **equation**) — describes the conditions under which the function output is 1.
3. **Logic diagram** — a combination of logic symbols joined by wires.
4. **Timing diagram** — see timing-diagrams[6].

Logic diagram conventions

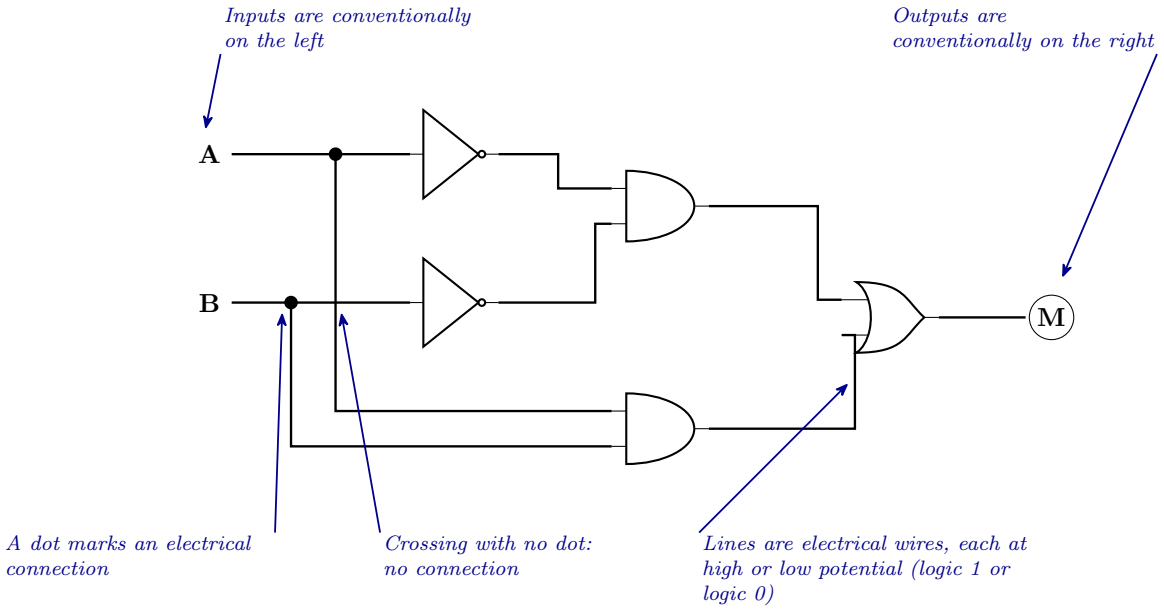

Figure 1: The drawing conventions for a logic diagram, on a two-input example. The two junction dots and the dot-free crossing next to them are the point: A branches down to the lower AND gate and crosses B's wire on the way, and only the dots say which wires are actually joined.

- Inputs are conventionally drawn on the **left**, outputs on the **right**.
- Lines represent **electrical wires**. A wire can be at high or low potential — logic 1 or logic 0 respectively.

[6] courses/csse2010/timing-diagrams.pdf

- A filled **dot** at a junction indicates an electrical connection; wires that cross **without** a dot are *not* connected.

Reading Figure 1 off the page as practice: the top AND gets $\bar{A}$ and $\bar{B}$, the bottom AND gets A and B, so $M = \bar{A}\bar{B} + AB$ — which is 1 exactly when A and B are equal, i.e. XNOR.

Gates on ICs

- Example: the **74HCT00** IC has four 2-input NAND gates in a single 14-pin DIP package — drawn out, with its pin numbers, in logic-gates[7].
- Vcc = power (e.g. 5V), GND = ground (0V). Pin spacing 0.1"×0.3"; chip ~15mm long.
- The internal drawing of the chip is a *schematic*; the black plastic part on the bench is the *IC*. See circuit-schematics[8] for the difference and for how chips get labelled on a buildable drawing.

Logic function implementation (sum of products)

Any logic function can be implemented as the **OR of AND combinations** of its inputs — called **sum of products**. From the truth table:

1. For each '1' in the output column, write down the AND combination of inputs that gives that 1 (complementing each input that is 0 in that row).
2. OR these together.

Worked example — truth table for M:

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	**1**	**1**	**1**
1	0	0	0
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	**1**	**1**	**1**

Four rows are 1, so there are four product terms:

[7] courses/csse2010/logic-gates.pdf
[8] courses/csse2010/circuit-schematics.pdf

$$M = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

which simplifies to

$$M = AB + BC + AC$$

This is the **majority function**: $M = 1$ whenever at least two of the three inputs are 1. (It comes back as the design task in week2-lab-majority-function-circuit-design[9].)

> **i** The simplification step
>
> The route is to duplicate ABC using the idempotent law $A + A = A$, then pair it with each of the other three terms and use the distributive and inverse laws (all in boolean-algebra[10]):
>
> $$\bar{A}BC + ABC = BC(\bar{A} + A) = BC$$
> $$A\bar{B}C + ABC = AC(\bar{B} + B) = AC$$
> $$AB\bar{C} + ABC = AB(\bar{C} + C) = AB$$

The equivalent logic diagram

This is the *unsimplified* $\bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$ drawn literally — three inverters, four 3-input ANDs and a 4-input OR. It is what sum of products gives you before any algebra.

Equivalent functions (simplification)

Sum of products does **not** necessarily produce the circuit with the minimum number of gates. The Boolean function can be *manipulated* into an equivalent function using the rules of Boolean algebra (see boolean-algebra[11]).

Example: $Z = AB + AC = A(B + C)$ by the distributive law.

A second worked simplification, CODE $= C + CD + \overline{(D + \bar{D})}$, collapses all the way down to CODE $= C$ — the steps are in boolean-algebra[12].

[9] courses/csse2010/week2-lab-majority-function-circuit-design.pdf
[10] courses/csse2010/boolean-algebra.pdf
[11] courses/csse2010/boolean-algebra.pdf
[12] courses/csse2010/boolean-algebra.pdf

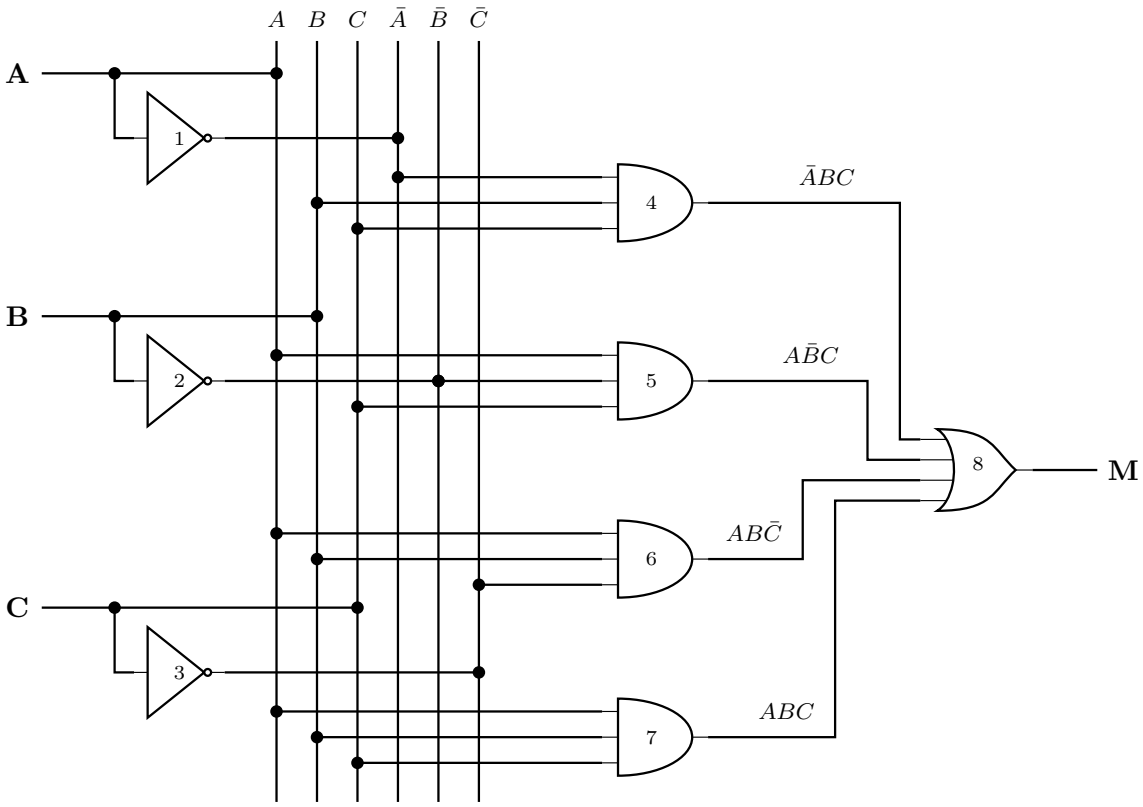

Figure 2: The sum-of-products expression built directly as a circuit. Gates 1-3 generate the complements; the six vertical rails carry A, B, C and their complements past every gate; gates 4-7 form one product term per '1' row of the truth table; gate 8 ORs them into M. Only the dots are connections - every other crossing is just two wires passing.

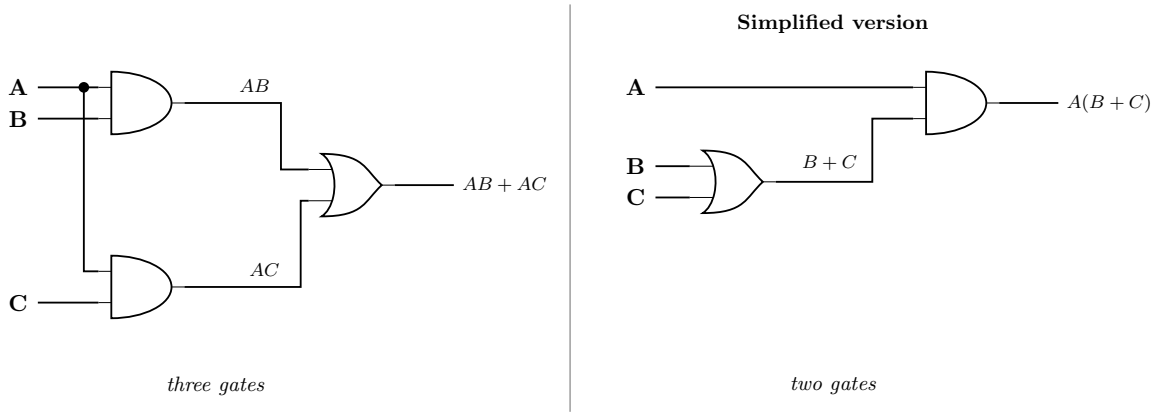

Figure 3: The same function drawn twice. The left circuit is the sum-of-products form AB + AC and needs three gates; factoring A out with the distributive law gives A(B+C), which needs two. Both have identical truth tables - the algebra is what buys the saving.

De Morgan's law and equivalent circuits

AND and OR can be interchanged **if you invert the inputs and the output** — the four gate equivalences are drawn in boolean-algebra[13]. Exercise: use truth tables to check that they are valid.

Universal (complete) gates

All circuits can be built from **NAND-only** or **NOR-only** gates (called *complete* gates), because NOT, AND, and OR can each be constructed purely from NAND gates, or purely from NOR gates — the six constructions are drawn in logic-gates[14]. The reason this matters in practice: it is **easier to build NAND and NOR gates from transistors** than AND/OR directly.

Reminders

- Attend Learning Lab sessions for the second half of this week — only the session you're signed up to. (Sign-on issues: email eait.mytimetable@uq.edu.au.)
- The slides used in a Learning Lab are released after the last session that week.
- If specific preparation is needed for a lab you will be told; by default, review the previous lectures.

[13] courses/csse2010/boolean-algebra.pdf
[14] courses/csse2010/logic-gates.pdf

- From week 2, you'll have both P2 (Mon-Tue) and P1 (Thu-Fri) lab sessions.