Combinational Logic Blocks

Table of contents

Reference note for the standard combinational building blocks: the adder-subtractor, the multiplexer, and the decoder. See logic-gates[1] for the gate symbols/truth tables and boolean-algebra[2] for the notation used below.

What makes a circuit combinational

A **combinational circuit** is a combination of logic gates with n inputs and m outputs:

- Each output can be expressed as a function of the n input variables.
- The **output depends on the current inputs only** — there is no memory of previous inputs. (Contrast with sequential-circuits[3], where the output also depends on stored state.)
- It can always be written as a truth table with n input columns, m output columns, and 2^n rows (one per possible input combination).

Binary subtraction

$A - B$ is usually implemented as $A + (-B)$, where:

- A and B are multi-bit quantities;
- "+" here means **addition**, not OR;

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

- $-B$ means the **two's complement** of B (see binary-number-representations[4]).

The two's complement of B is calculated by **flipping all the bits and adding 1**. So a subtractor is just an adder with two extra pieces: something that conditionally inverts B, and something that adds the extra 1.

The controlled inverter (XOR trick)

We need a gate that flips a bit *only sometimes*: given a mode bit M,

$$Z = \bar{B} \text{ when } M = 1, \qquad Z = B \text{ when } M = 0$$

The slide poses this as a question and is otherwise blank. Derived from the XOR truth table: XOR with one input held at 0 passes the other input through unchanged, and XOR with one input held at 1 inverts it. So the gate is a **2-input XOR**:

$$Z = B \oplus M$$

M	B	$Z = B \oplus M$	Effect
0	0	0	pass through
0	1	1	pass through
1	0	1	invert
1	1	0	invert

This is called a **controlled inverter**.

Adder-subtractor circuit

A 4-bit adder-subtractor is built from a 4-bit binary adder (a ripple-carry adder — see 2026-08-04-binary-arithmetic[5]) plus four XOR gates:

- Data input A ($A_0 \ldots A_3$) goes **straight** into the adder's A inputs.
- Data input B ($B_0 \ldots B_3$) goes into the adder's B inputs **via one XOR gate per bit**; the second input of every XOR gate is the shared **mode select** line M.
- $M = 0$ means **add**, $M = 1$ means **subtract**.
- The adder produces the data output $S_0 \ldots S_3$, plus a carry-out C_4 and taking a carry-in C_0.

[4] courses/csse2010/binary-number-representations.pdf
[5] courses/csse2010/2026-08-04-binary-arithmetic.pdf

What should the carry-in be? The slide asks this and leaves it blank. Derived: with $M = 1$ the XOR gates supply $\bar{B}$, and two's complement needs $\bar{B} + 1$ — the extra +1 is supplied by feeding it in as the carry-in. With $M = 0$ we want plain addition, which needs a carry-in of 0. Both cases are satisfied by

$$C_0 = M$$

so the mode select line is wired to both the XOR gates and the adder's carry-in.

A 4-bit adder is available as an off-the-shelf IC — the 74HCT283, see device-pinouts[6].

Multiplexer (mux)

A **multiplexer** has:

- 2^n **data** inputs,
- **1** output,
- n **control** (or **select**) inputs, which *select* one of the data inputs to be "sent" or "steered" to the output.

4-to-1 multiplexer

Data inputs $D_0 \ldots D_3$, select inputs $S_1 S_0$, output F. The logic symbol is the characteristic trapezoid with the data inputs on the wide (left) edge, F on the narrow (right) edge, and the select inputs entering the bottom.

Function table:

S_1	S_0	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Note this is a **function table**, not a full truth table — the full truth table would need $2^6 = 64$ rows for the six inputs $S_1, S_0, D_0, D_1, D_2, D_3$.

[6] courses/csse2010/device-pinouts.pdf

4-to-1 mux logic circuit implementation

- S_1 and S_0 each feed an inverter, giving $\bar{S}_1$ and $\bar{S}_0$ as well as the true forms.
- Four **3-input AND gates**, one per data input. Each AND gate takes its data input plus the select-literal pair that identifies it: D_0 with $\bar{S}_1\bar{S}_0$, D_1 with $\bar{S}_1 S_0$, D_2 with $S_1\bar{S}_0$, D_3 with $S_1 S_0$.
- The four AND outputs feed a single **4-input OR gate** whose output is F.

That is exactly the sum-of-products form:

$$F = D_0\bar{S}_1\bar{S}_0 + D_1\bar{S}_1 S_0 + D_2 S_1\bar{S}_0 + D_3 S_1 S_0$$

Only the AND gate whose select literals match the current S_1S_0 can be 1, so exactly one data input reaches the OR gate at a time.

2-to-1 multiplexer

Data inputs D_0 and D_1, one control input S_0, output F.

Function table:

S_0	F
0	D_0
1	D_1

Expanded to a full truth table over all three inputs:

S_0	D_0	D_1	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

i.e. $F = D_0\bar{S}_0 + D_1 S_0$. When $S_0 = 0$ the output tracks D_0 and ignores D_1; when $S_0 = 1$ it tracks D_1 and ignores D_0.

A quad 2-input multiplexer is available as an off-the-shelf IC — the 74HCT157, see device-pinouts[7].

Using a mux to implement a logic function

Because the data inputs can be tied to constant 0 or 1, an n-select mux with its selects wired to the function's variables can implement *any* function of those n variables: set data input D_i to the value the function should take when the selects equal i. See the worked polling question in 2026-08-06-combinational-logic[8].

Decoder

A **decoder** converts an n-bit input to a logic 1 on **exactly one** of its 2^n outputs (the one whose index equals the input value); all other outputs are 0.

3-to-8 decoder

Inputs A, B, C (with A the most significant), outputs $D_0 \ldots D_7$.

A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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**

(The slide tabulates only the first four rows and elides the rest with "…"; the remaining four rows are filled in above from the definition — the 1 always sits in the column whose index is the binary value of ABC.)

[7] courses/csse2010/device-pinouts.pdf
[8] courses/csse2010/2026-08-06-combinational-logic.pdf

3-to-8 decoder logic circuit implementation

- Each of A, B, C is fanned out to an inverter, so all six literals $A, \bar{A}, B, \bar{B}, C, \bar{C}$ are available on a vertical bus.
- **Eight 3-input AND gates**, one per output. Each taps the three literals matching its index:

$$D_0 = \bar{A}\bar{B}\bar{C}, \quad D_1 = \bar{A}\bar{B}C, \quad D_2 = \bar{A}B\bar{C}, \quad D_3 = \bar{A}BC$$
$$D_4 = A\bar{B}\bar{C}, \quad D_5 = A\bar{B}C, \quad D_6 = AB\bar{C}, \quad D_7 = ABC$$

Each AND gate is the minterm for one input combination, so exactly one is high at any time.