Counters

Table of contents

Binary counters . . . 1
4-bit binary counter — counting sequence . . . 2
Key points . . . 2
State: the design idea . . . 2
Design recipe . . . 3
Worked example — a 2-bit counter for $00 \to 10 \to 01 \to 00$ . . . 4
Applied exercise . . . 5
Related . . . 5

A **counter** is a multi-bit register that goes through a *determined sequence of states* (values) upon the application of input (clock) pulses. It is a synchronous sequential circuit[1] built from D flip-flops[2] plus some combinational logic in the feedback path.

Binary counters

A counter which follows the binary number sequence is a **binary counter**.

An n**-bit binary counter**:

- has n flip-flops;
- has 2^n different states;
- counts from 0 to $2^n - 1$, then wraps back to 0.

For example, a 2-bit binary counter counts

$$00 \to 01 \to 10 \to 11 \to 00 \to ...$$

i.e. $0 \to 1 \to 2 \to 3 \to 0 \to ...$

[1] courses/csse2010/sequential-circuits.pdf
[2] courses/csse2010/flip-flops-and-latches.pdf

4-bit binary counter — counting sequence

Bit 3	Bit 2	Bit 1	Bit 0	Value
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15
0	0	0	0	0

Each bit toggles at half the rate of the bit to its right — bit 0 changes every count, bit 1 every second count, bit 2 every fourth, bit 3 every eighth.

Key points

- The **next state is a function of the present state** (and possibly of external inputs).
- The count sequence **can** be the binary numbers but **does not have to be** — if it is, the counter is a binary counter; otherwise it is just some arbitrary cycle through states.
- These circuits are **synchronous**: all flip-flops share the same clock.
- **Asynchronous** counters exist too, but are not discussed in this course.

State: the design idea

This is the whole trick, and it is what makes counter design mechanical:

- The values stored in the flip-flops are the **current (present) state** of the circuit.
- The D inputs to the flip-flops are the **next state** — whatever is sitting on D at the clock edge becomes the new state.

- The D inputs are some **combinational function of the current state** (and of any external inputs).

So for a counter with state bits $Q_{n-1} \ldots Q_1 Q_0$, designing the counter means finding n Boolean expressions

$$D_i = f_i(Q_{n-1}, \ldots, Q_1, Q_0)$$

and building each one out of gates[3].

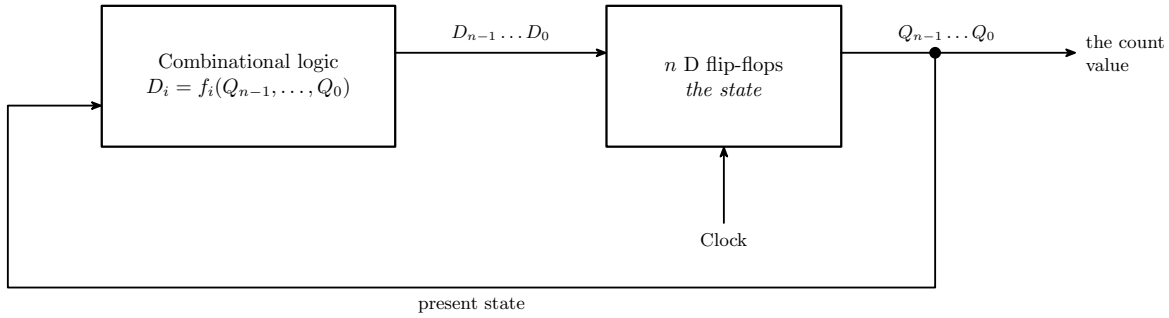

Figure 1: A counter is the general sequential-circuit block diagram with the external inputs deleted. That diagram, as Mano draws it, is a combinational circuit feeding flip-flops, the flip-flop outputs routed back round to the combinational circuit, and the clock entering the flip-flops on its own labelled stub. A counter may or may not have external inputs; removing them, as here, is what makes the design mechanical: with nothing else to depend on, each D is a function of the Q values alone, so the present-state/next-state table below is a complete specification of the circuit.

Design recipe

1. **Write down the count sequence** you want, as a cycle of state values.
2. **Build a present-state / next-state table.** One row per possible state — all 2^n of them, not just the ones in your sequence. Columns: present state $Q_{n-1} \ldots Q_0$, then next state $D_{n-1} \ldots D_0$. Fill each next-state entry by reading off what follows that state in your sequence.
3. **Handle unused states.** If your sequence doesn't use all 2^n states, the next state for the unused ones is a **don't-care** (X). You may set each X to whatever makes the algebra simplest — but check afterwards where the unused state actually goes, because a self-correcting counter (one whose unused states lead back into the cycle) is nicer than one that can get stuck.

[3] courses/csse2010/logic-gates.pdf

4. **Read each D column as a Boolean function of the Qs.** The table *is* a truth table with the Qs as inputs and D_i as the output, so write it in sum-of-products form: OR together one AND term per row where $D_i = 1$. See boolean-algebra[4].
5. **Simplify** each expression using the Boolean algebra laws (and the don't-cares), then draw the circuit: the combinational logic for D_i feeding flip-flop i, with every flip-flop on a **common clock**. See circuit-schematics[5] for schematic conventions and device-pinouts[6] for the 74HCT74 / 74HCT175 D flip-flop packages.

The smallest case falls straight out of the recipe: a **1-bit counter** counts $0 \to 1 \to 0 \to \ldots$, so its next state is always the complement of its present state, giving $D = \bar{Q}$ — a single flip-flop with its $\bar{Q}$ output wired back to its own D input (a *toggle*).

Worked example — a 2-bit counter for $00 \to 10 \to 01 \to 00$

Step 1–2 — present-state / next-state table. The sequence uses three of the four states; 11 is unused, so its next state is a don't-care.

Q_1	Q_0	D_1	D_0	comment
0	0	1	0	$00 \to 10$
0	1	0	0	$01 \to 00$
1	0	0	1	$10 \to 01$
1	1	X	X	unused state

Step 4 — read off the columns.

D_1 is 1 only in the row $Q_1Q_0 = 00$. Making the don't-care 1 would give $Q_1 \odot Q_0$ (XNOR), which is no simpler, so take the don't-care as 0 ($X_1 = 0$):

$$D_1 = \bar{Q}_1\bar{Q}_0$$

D_0 is 1 only in the row $Q_1Q_0 = 10$, which gives $D_0 = Q_1\bar{Q}_0$. Here the don't-care *does* help: setting it to 1 ($X_0 = 1$) makes $D_0 = 1$ for both rows with $Q_1 = 1$, so

$$D_0 = Q_1$$

Step 5 — the circuit. Two D flip-flops on a common clock. Flip-flop 0's D input is wired directly to flip-flop 1's Q_1 output — no gate at all. Flip-flop 1's D input is driven by a

[4] courses/csse2010/boolean-algebra.pdf
[5] courses/csse2010/circuit-schematics.pdf
[6] courses/csse2010/device-pinouts.pdf

2-input NOR of Q_1 and Q_0 (since $\bar{Q}_1\bar{Q}_0 = \overline{Q_1 + Q_0}$), or equivalently by an AND of the two $\bar{Q}$ outputs.

Check, including the unused state:

present	$D_1 D_0$	next
00	1 0	10
10	0 1	01
01	0 0	00
11	0 1	01

The cycle is right, and the unused state falls into the cycle on the next clock edge, $11 \to 01$, so the counter is **self-correcting** — a robust design. (Choosing $D_0 = Q_1\bar{Q}_0$ instead — don't-care set to 0 — would send $11 \to 00$, also fine, at the cost of a whole extra gate: an AND fed from Q_1 and the flip-flop's $\bar{Q}_0$ pin, where $D_0 = Q_1$ needs no gate at all.)

Applied exercise

week4-lab-lab-7-exercises[7] is exactly this recipe at $n = 3$: design a 3-bit synchronous counter that steps through an allocated 7-state sequence, with the eighth (missing) state treated as a don't-care.

Related

- flip-flops-and-latches[8] — the D flip-flop and its characteristic table.
- sequential-circuits[9] — state, present state / next state, synchronous sequential circuits.
- shift-registers[10] — the other family of registers built from a flip-flop chain.
- boolean-algebra[11] — sum of products and the simplification laws used in step 4–5.
- binary-number-representations[12] — the binary counting sequence itself.
- state-machines[13] — the same design recipe generalised: a counter is a state machine whose transitions carry no input condition.
- 2026-08-17-counters[14] — Lecture 7.
- csse2010-diagram-notation[15] — the drawing conventions the figures above follow.

[7] courses/csse2010/week4-lab-lab-7-exercises.pdf
[8] courses/csse2010/flip-flops-and-latches.pdf
[9] courses/csse2010/sequential-circuits.pdf
[10] courses/csse2010/shift-registers.pdf
[11] courses/csse2010/boolean-algebra.pdf
[12] courses/csse2010/binary-number-representations.pdf
[13] courses/csse2010/state-machines.pdf
[14] courses/csse2010/2026-08-17-counters.pdf
[15] courses/csse2010/csse2010-diagram-notation.pdf

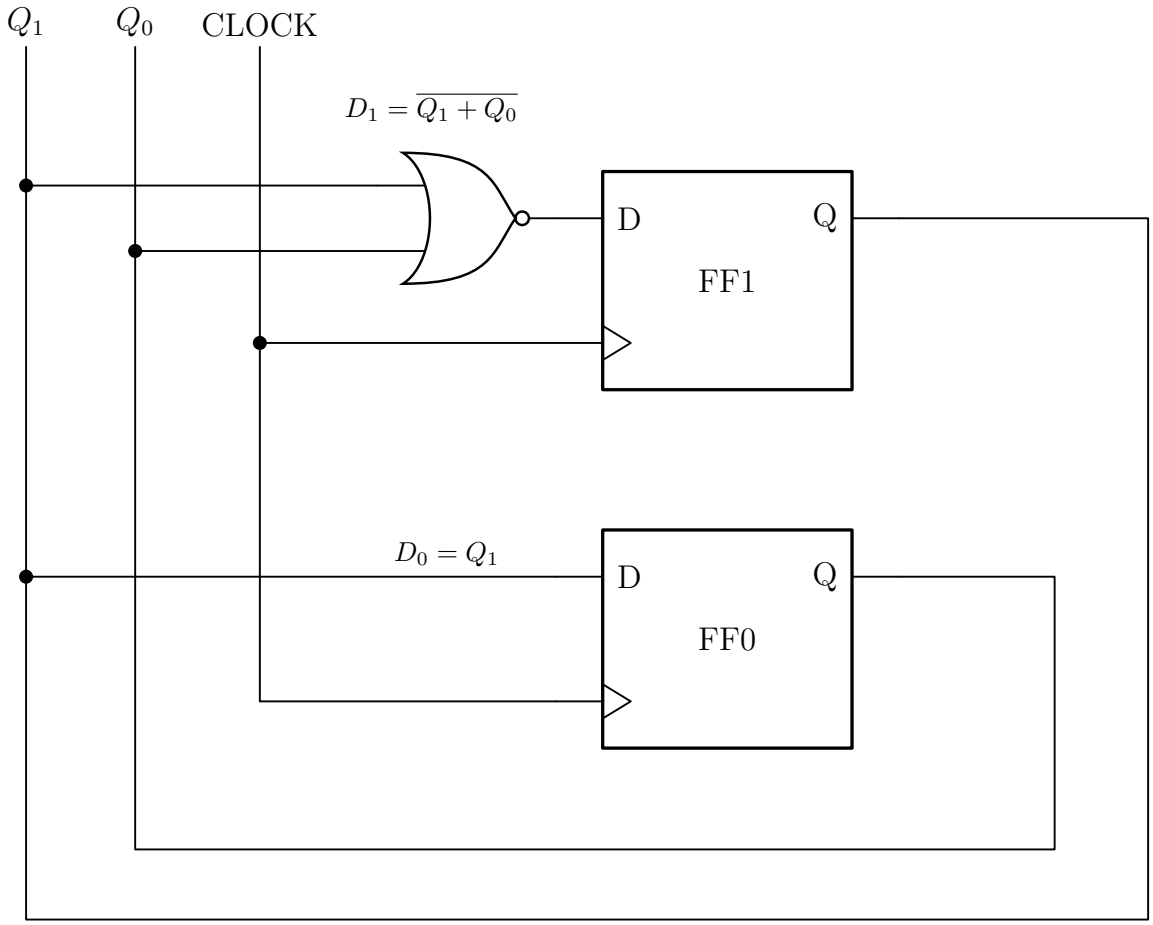

Figure 2: The whole counter, drawn from the derivation above. Each Q runs back to a vertical rail that the next-state logic taps, which is the physical form of the feedback loop in the block diagram earlier: D0 is FF1's Q wired straight across with no gate at all, and D1 is one NOR gate. Dots are connections; where the Q0 rail and the CLOCK rail cross the D0 wire there is no dot and no connection. The circuit takes the NOR form of D1; an AND of the two Q-bar outputs is the same function and would need the flip-flops' Q-bar pins instead.