Lecture 7 — Counters

Table of contents

Outline . . . 1
Schedule . . . 2
Worked: shift direction control . . . 2
Shift register types . . . 4
Counters . . . 4
Digital system application . . . 4
One-bit counter . . . 5
State . . . 6
Counter example — the 2-bit binary up counter . . . 6
Worked: what sequence does this counter count through? . . . 7
Key points . . . 10
Worked example — a non-binary sequence with a don't-care . . . 10
Next: state machines . . . 11
Reminders . . . 13

See [counters] for the reference material on counters, and shift-registers[1], sequential-circuits[2] and flip-flops-and-latches[3] for the background.

Counter design is a procedure: write the sequence down as a state table, read each D column off as a Boolean function of the Qs, then build it. Three circuits carry that idea — the one-bit toggle, the 2-bit binary up counter, and a 2-bit counter for a non-binary sequence with a don't-care state.

Outline

- A worked question on the mux-per-stage shift register

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

- Shift register types
- Counters, and binary counters
- A digital system application built around a counter
- State: current state, next state, and the D inputs
- Worked counter examples
- Next: state machines

Schedule

From the course teaching outline:

- **Week 4, Mon-Tue 17-18 Aug** — this lecture, *Lecture 7: Counters* (17 Aug).
- **Week 4, Thu-Fri 20-21 Aug** — *Lecture 8: Finite State Machines* (20 Aug).
- Learning labs this week: **Lab 6 Shift Registers** (P2, Mon-Tue) and **Lab 7 Counters** (P1, Thu-Fri).
- Week 5: *Lecture 9 ALU and Control Unit* (24 Aug), *Lecture 10 Introduction to AVR and Assembly Language* (27 Aug), with labs 8 (Finite State Machines) and 9 (AVR Assembly Programming 1).

Later in the schedule: the week 6 Thu-Fri **revision and help sessions for the lab exam**, and the week 7 **Lab Exam (Digital Logic)**, held during the scheduled lab sessions. Everything from this lecture onwards is lab-exam material three weeks later.

Worked: shift direction control

> Which of the following statements about the circuit below is true?
>
> A. When A is 1, flip-flop values are shifted to the right B. When A is 0, flip-flop values are shifted to the right C. When A is 0, flip-flop values stay the same D. When FN is 1, flip-flop values are shifted to the left E. When FN is 0, flip-flop values stay the same F. When FN is 0, flip-flop values are shifted to the left

The circuit is the mux-per-stage construction from last lecture[4], two stages deep.

Reading Figure 1 off, with Q_a = stage 1 and Q_b = stage 2:

$$D_a = \begin{cases} A & FN = 1 \\ Q_a & FN = 0 \end{cases} \qquad D_b = \begin{cases} Q_a & FN = 1 \\ Q_b & FN = 0 \end{cases}$$

- When $FN = 1$, A is loaded into stage 1 and stage 1's value moves into stage 2 — data shifts **left-to-right along the diagram**, one stage per clock.

[4] courses/csse2010/2026-08-13-shift-registers.pdf
[5] courses/csse2010/csse2010-diagram-notation.pdf

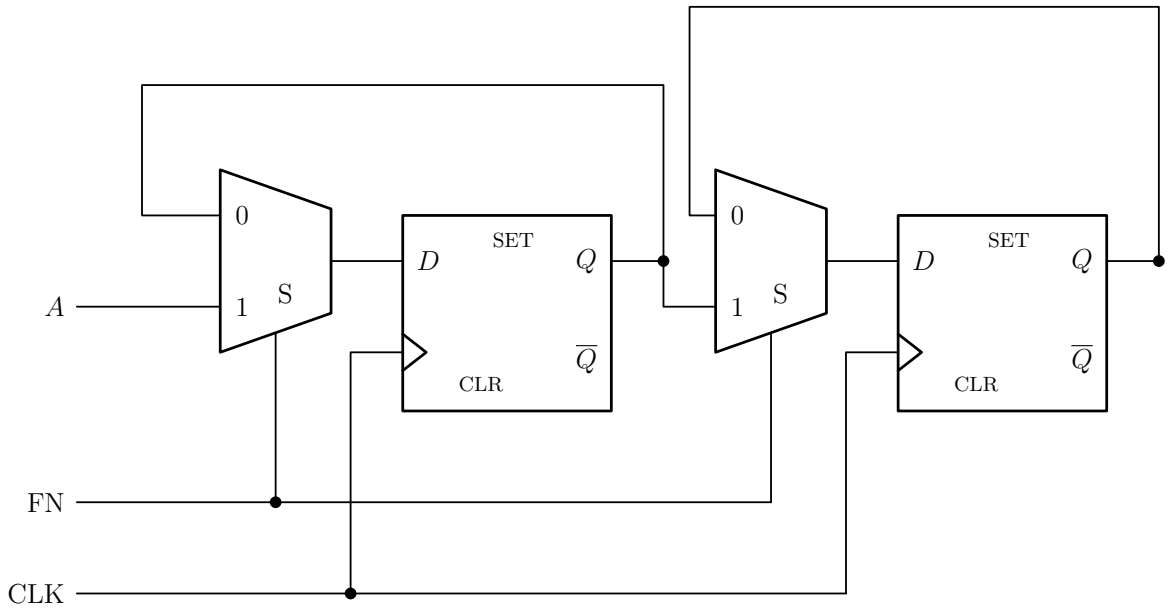

Figure 1: A two-stage shift register with a 2-to-1 mux in front of every flip-flop. Each mux's `0` input is fed back from *its own* flip-flop's Q — that is the hold path — and its `1` input takes the value from the stage before it (A from outside, for stage 1). One line, FN, selects both muxes, so FN enables the shift rather than choosing a direction; there is no left-shift path anywhere in the circuit. Each mux is drawn wide-edge-left with the select entering the bottom sloping edge (see csse2010-diagram-notation[5]). The one crossing, the CLK riser over the FN rail, carries no junction dot and so is not a connection.

- When $FN = 0$, each flip-flop's D input is its own Q, so every flip-flop reloads the value it already had — **nothing changes**.

FN is therefore a shift *enable*, not a direction control, so every option mentioning "shifted to the left" is out; and A is plain data rather than a control signal, so A/B/C are out too.

Answer: E — when FN is 0, flip-flop values stay the same.

Shift register types

There are seven shift-register configurations: serial in/shift right/serial out, serial in/shift left/serial out, parallel in/serial out (**PISO**), serial in/parallel out, parallel in/parallel out, rotate right and rotate left. Rotate right is a right shift with a wrap-around path.

Getting `0101` into a 4-bit register takes four clocks when it is shifted in serially (a **serial load**) and a single clock when it is loaded in parallel: n clocks versus one. The configurations and the clock-by-clock trace are in shift-registers[6].

Counters

An n-bit binary counter has n flip-flops and 2^n states, and counts 0 to $2^n - 1$. The definition of a counter and the 4-bit binary counting sequence table are in [counters]. The sequence runs `000…0`, `000…1`, , `111…1` and then back to `000…0`, which is 2^n states. The state count is the *length of the cycle*, not just the width of the register.

Digital system application

Where a counter sits in a real system — an automated tablet-bottling line (the example is from Floyd's *Digital Fundamentals*):

- A **keypad** for the number of tablets per bottle feeds an **encoder**, which feeds **Register A**. Register A drives **Decoder A** and a 7-segment display showing the tablets-per-bottle setting, and also a **code converter** producing the binary code for the preset number.
- On the conveyor, a **sensor** emits one pulse per tablet passing the valve. Those pulses are the clock of a **counter**, so the counter holds the binary code for the actual number of tablets in the bottle. A separate pulse **resets the counter to zero** when the next bottle is in place.
- A **comparator** tests the counter value against the preset value. Its output goes HIGH when $A = B$: HIGH closes the valve and advances the conveyor, LOW keeps the valve open.

[6] courses/csse2010/shift-registers.pdf

- The counter value also feeds an **adder**, whose other input is the running total held in **Register B**; the comparator's HIGH also causes the new sum to be stored back into Register B. Register B drives **Decoder B** (total display) and a **MUX/DEMUX** path that sends the accumulated total on to a computer for storage.

The counter, adder, register, decoder, comparator, encoder, mux and demux blocks covered so far in the course compose into a complete system — see combinational-logic-blocks[7] for the mux/decoder blocks and 2026-08-04-binary-arithmetic[8] for the adder.

One-bit counter

The circuit is a single D flip-flop, clocked on CLK, with a wire from its $\overline{Q}$ output looped back around to its D input.

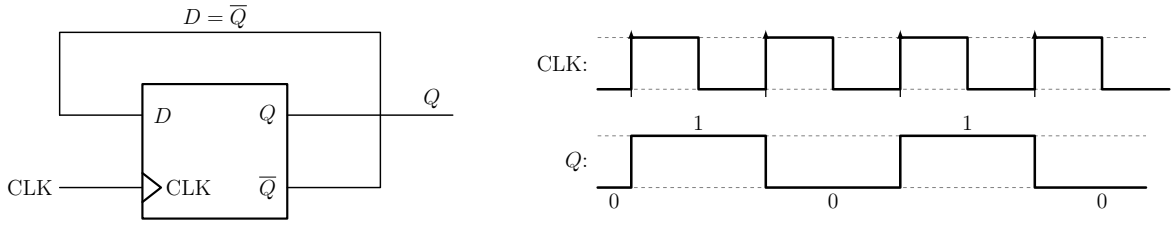

Figure 2: Left: one D flip-flop with $\overline{Q}$ looped back to D, so the next state is always the complement of the present state. The Q output line crosses the feedback wire with no junction dot, i.e. no connection. Right: the waveform, with an arrow on each rising clock edge and the value of Q, `0` or `1`, written on each segment. Q toggles on every rising edge, so the flip-flop divides the clock frequency by two.

The derivation takes three steps. First the equation: $D = \overline{Q}$. Then the roles: Q, the flip-flop's stored value, is the **present state**; D, what is sitting on the input at the clock edge, is the **next state**. Then those two written out as columns give the **state table**:

present state Q	next state D
0	1
1	0

Since D is the next state and $D = \overline{Q}$, the value inverts on every clock edge and the counter counts

[7] courses/csse2010/combinational-logic-blocks.pdf
[8] courses/csse2010/2026-08-04-binary-arithmetic.pdf

$$0 \to 1 \to 0 \to 1 \to \ldots$$

which is exactly the $n = 1$ binary counter ($2^1 = 2$ states, counting 0 to $2^1 - 1 = 1$). This is the standard *toggle* configuration, and the two-row table above is the smallest possible instance of the method the larger counters below use.

> **Present state or current state?**
>
> **Current state** and **present state** are synonyms. The course uses both and settles neither; see csse2010-diagram-notation[9].

State

Three statements make counter design mechanical. Take two D flip-flops on a common CLK, with inputs D_1 and D_0 still to be worked out:

- the values stored in the flip-flops are the **current state** of the circuit;
- the D inputs are the **next state**;
- the D inputs are some (combinational) **function of the current state and the inputs**.

So each D input is driven by its own combinational circuit, and each of those circuits is fed from the Q outputs. That is the whole architecture — flip-flops hold the state, a block of combinational logic computes the next one, and the clock steps between them. The design method that follows is written up as a recipe in [counters]; the underlying state terminology is in sequential-circuits[10].

Counter example — the 2-bit binary up counter

The problem is to **design** a **2-bit** counter for $00 \to 01 \to 10 \to 11 \to 00$. Design means a sequence is given and the circuit has to be found; the worked question after this one is analysis, where a circuit is given and the sequence has to be found.

Start from two D flip-flops on a common CLK with **nothing connected** to their D inputs, and a current-state/next-state table with columns $Q_1, Q_0 \mid D_1, D_0$ and a row for each of the four states. Filling in the next state for each row gives the **state table**:

[9] courses/csse2010/csse2010-diagram-notation.pdf

[10] courses/csse2010/sequential-circuits.pdf

Q_1	Q_0	D_1	D_0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Reading the two next-state columns as Boolean functions of Q_1 and Q_0 gives two equations:

$$D_0 = \overline{Q_0} \qquad\qquad D_1 = Q_1 \oplus Q_0$$

D_0 is 1 exactly when Q_0 is 0, so bit 0 toggles every clock — it is the one-bit counter of Figure 2, unchanged. $D_1 = Q_1 \oplus Q_0$ means bit 1 takes its own value flipped whenever $Q_0 = 1$ and keeps it whenever $Q_0 = 0$ — which is the carry rule for binary counting: bit 1 flips exactly on the clocks where bit 0 rolls over from 1 to 0.

As a check, step the circuit through a clock train and write down the state at each rising edge: 00, 01, 10, 11, 00. The circuit walks the sequence it was designed from. This is the direct model for the 3-bit design in Lab 7.

Worked: what sequence does this counter count through?

> What sequence does the counter below count through? (Assume Q_1Q_0 starts at 00.)
>
> A. 00 → 11 → 10 → 01 → 00 B. 00 → 10 → 11 → 01 → 00 C. 00 → 11 → 01 → 10 → 00 D. 00 → 01 → 11 → 10 → 00 E. 00 → 01 → 10 → 11 → 00 F. 00 → 10 → 01 → 11 → 00

This is the design problem run backwards: the circuit is given, the sequence has to be recovered.

Tabulating from Figure 4:

Q_1	Q_0	$D_1 = \overline{Q_1}$	$D_0 = \overline{Q_1 \oplus Q_0}$	next state
0	0	1	1	11
1	1	0	1	01
0	1	1	0	10
1	0	0	0	00

[11] courses/csse2010/csse2010-diagram-notation.pdf

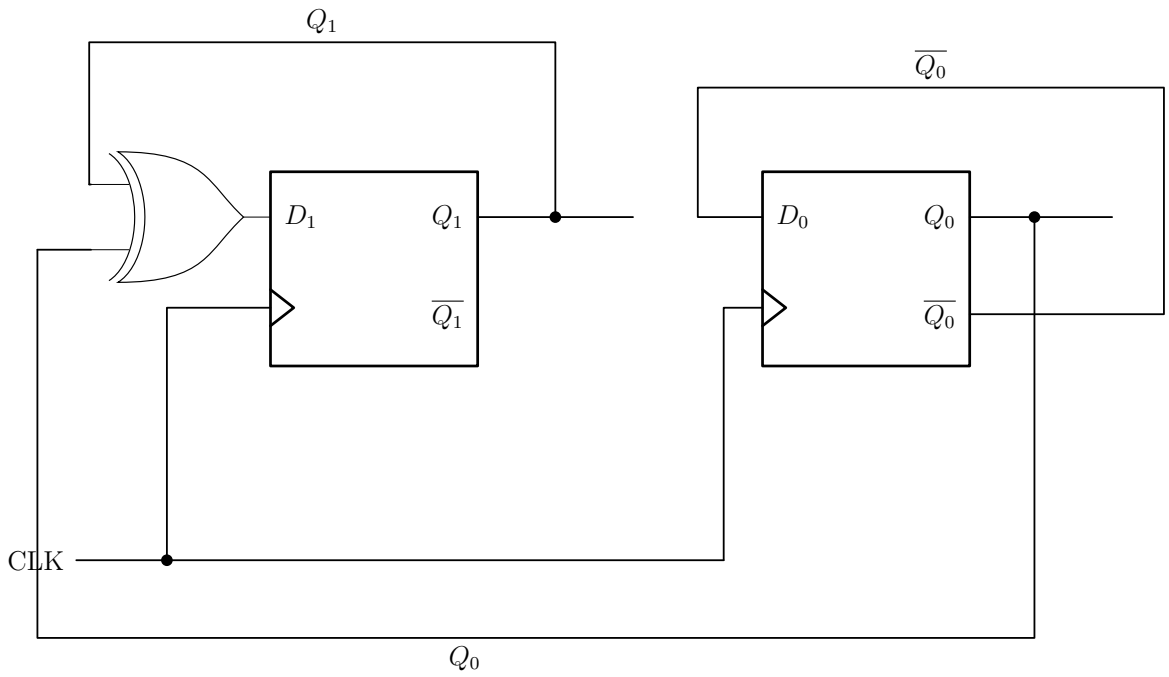

Figure 3: The 2-bit synchronous binary up counter for $00 \to 01 \to 10 \to 11 \to 00$. $\overline{Q_0}$ loops back to D_0 (bit 0 toggles every clock); an XOR of Q_1 and Q_0 drives D_1. Both flip-flops share one clock, which is what makes the counter synchronous. The gate is drawn the conventional way round — inputs on the left, output on the right — a rule the course schematics guide states and that holds without exception (see csse2010-diagram-notation[11]). The feedback to D_0 is taken from the $\overline{Q}$ pin at the lower right of the flip-flop.

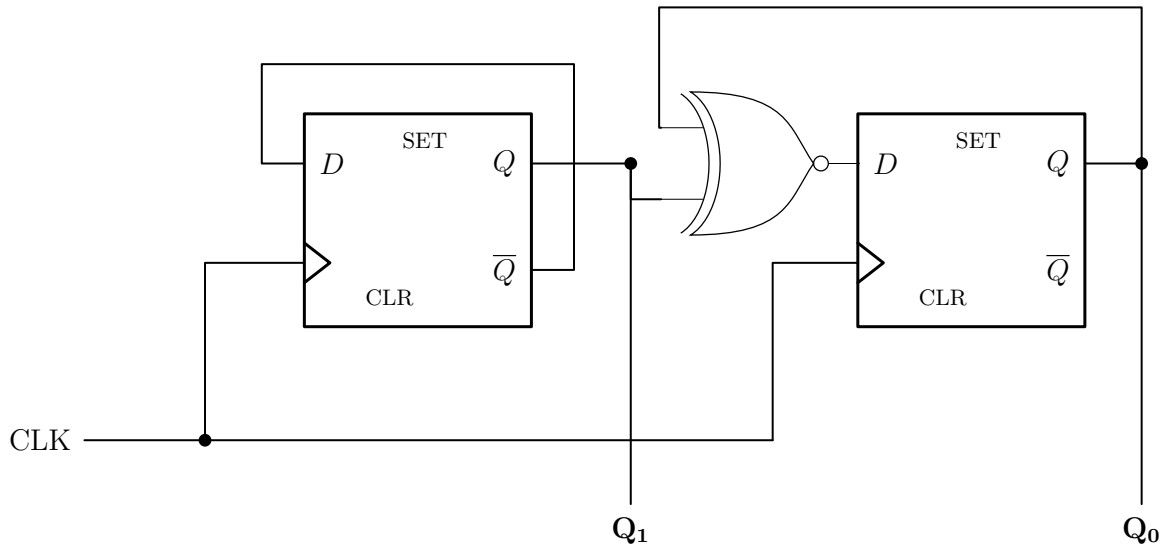

Figure 4: The counter to analyse. The left flip-flop's $\overline{Q}$ is wired back to its own D, so $D_1 = \overline{Q_1}$ and Q_1 toggles every clock. The right flip-flop's D is driven by a two-input **XNOR** — an OR body with the extra input-side arc *and* an inversion bubble on the output — whose inputs are Q_1 and Q_0, so $D_0 = \overline{Q_1 \oplus Q_0}$. Everything in the answer turns on that gate symbol. SET and CLR are lettered inside the boxes and are unused.

Starting from 00: $00 \to 11 \to 01 \to 10 \to 00 \to \ldots$

Answer: C.

The gate reading rests entirely on the **symbol** — the extra input-side arc plus the output bubble is an XNOR, not a NOR. Don't try to settle it by elimination: an XOR reading gives $D_0 = Q_1 \oplus Q_0$, $D_1 = \overline{Q_1}$ and the sequence $00 \to 10 \to 01 \to 11 \to 00$, which *is* one of the listed options (F).

This counter is **not** a binary counter. Its sequence is a legitimate four-state cycle that simply isn't the binary count — exactly the case the second key point below covers.

Key points

Four key points, all recorded in [counters]:

- the **next state is a function of the previous 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*;
- these circuits are **synchronous**: all flip-flops share the same clock;
- **asynchronous** counters exist as well, but the course does not discuss them.

Worked example — a non-binary sequence with a don't-care

The problem:

> Design a 2-bit counter that counts $00 \to 10 \to 01 \to 00 \to$

Step 1 — the state table. Present state on the left, next state on the right. The fourth row, present state 11, is an **unused state**. Its next-state entries are written X_1X_0, where an X is a **don't care term**:

Q_1	Q_0	D_1	D_0	
0	0	1	0	$00 \to 10$
1	0	0	1	$10 \to 01$
0	1	0	0	$01 \to 00$
1	1	X_1	X_0	unused state

Step 2 — spend the don't-cares. Write down *both* choices for each column, then pick:

$$D_0: \quad X_0 = 0 \Rightarrow D_0 = Q_1\overline{Q_0} \qquad X_0 = 1 \Rightarrow D_0 = Q_1 \quad \checkmark$$
$$D_1: \quad X_1 = 0 \Rightarrow D_1 = \overline{Q_1}\,\overline{Q_0} \quad \checkmark \qquad X_1 = 1 \Rightarrow D_1 = \overline{Q_1 \oplus Q_0} \quad \times$$

$X_0 = 1$ is chosen because it is strictly simpler: adding the 11 row to the single 10 row lets Q_0 drop out entirely and leaves $D_0 = Q_1$, a bare wire. $X_1 = 1$ is **rejected** even though it also looks tidy — an XNOR is one gate, same as an AND. The reason is the next step.

Step 3 — check the unused state. With $X_1X_0 = 01$ the transitions are

$$00 \to 10 \to 01 \to 00, \qquad 11 \to 01$$

This is a **robust design**: the unused state falls straight back into the cycle, so a power-up glitch into 11 costs one clock and nothing more. Taking $X_1 = 1$ instead would have given $11 \to 11$ and the counter would have hung there forever — which is why that option is rejected. The final equations are

$$D_1 = \overline{Q_1}\,\overline{Q_0} \qquad\qquad D_0 = Q_1$$

As a check, step the circuit through a clock train again: the state at each rising edge is 00, 10, 01, 00, and from 11 it drops into 01. The full write-up of this example, including the sum-of-products working, is in [counters]; week4-lab-lab-7-exercises[13] is the same procedure at $n = 3$.

Next: state machines

More sequential circuits come next: **state machines**. In the general synchronous sequential circuit, inputs and fed-back state go into a combinational circuit, whose outputs are both the circuit outputs and the flip-flop inputs; the flip-flops are driven by clock pulses and their outputs are fed back (see sequential-circuits[14]).

A counter may or may not have external inputs. This is the link between the two: a counter is the special case of a state machine whose transition logic reads nothing but the state itself, which is why none of the four counter circuits above has an input other than the clock. See state-machines[15] and 2026-08-20-state-machines[16].

[12] courses/csse2010/csse2010-diagram-notation.pdf
[13] courses/csse2010/week4-lab-lab-7-exercises.pdf
[14] courses/csse2010/sequential-circuits.pdf
[15] courses/csse2010/state-machines.pdf
[16] courses/csse2010/2026-08-20-state-machines.pdf

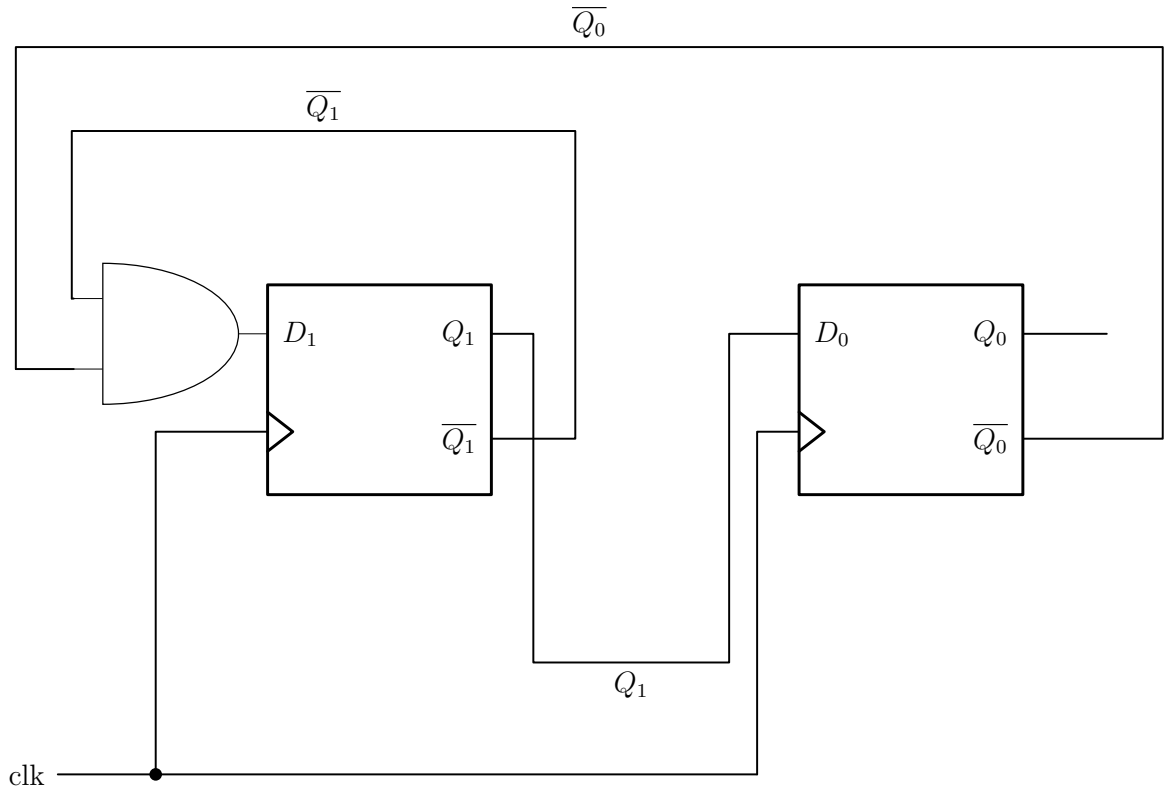

Figure 5: The implementation of $D_1 = \overline{Q_1}, \overline{Q_0}$ and $D_0 = Q_1$ for $00 \to 10 \to 01 \to 00$. One AND gate over the two complement outputs is the whole of the combinational logic; D_0 needs no gate at all, being Q_1 wired straight across. As in Figure 3 the gate is drawn inputs-left/output-right. The clock is lettered `clk` here and `CLK` in the other figures; the course uses both and settles neither (see csse2010-diagram-notation[12]).

Reminders

- **Lab 7 (Counters)** runs in the P1 sessions this week (Thu-Fri) — the pre-lab schematic for the 3-bit synchronous counter must be drawn beforehand. See week4-lab-lab-7-exercises[17], and device-pinouts[18] for the flip-flop packages.
- Lab 6 (Shift Registers) runs in the P2 sessions (Mon-Tue) this week.
- The centrally scheduled 90-minute in-semester theory exam is on **Saturday 5 September**, and the Digital Logic **lab exam is during the scheduled lab sessions in week 7**.

[17] courses/csse2010/week4-lab-lab-7-exercises.pdf
[18] courses/csse2010/device-pinouts.pdf