Shift Registers

Table of contents

Registers . . . 1
Shift registers . . . 2
Serial parallel conversion . . . 3
Parallel load vs serial shift . . . 3
The mux-based bidirectional shift element . . . 6
The configurations a shift register can be wired into . . . 7
Universal shift register . . . 8
Wide (multi-bit) shift registers . . . 8
Related . . . 9

Registers and shift registers are the simplest useful sequential circuits[1] — groups of D flip-flops[2] sharing a common clock.

Registers

- A **register** is a group of flip-flops. An n**-bit register** consists of n flip-flops and is capable of storing n bits.
- A register is a sequential circuit **without any combinational logic** — it is just the storage elements and their shared control lines.
- Registers are used to store binary information (data/instructions) **inside a processor**.

A 4-bit register:

Signal	Connection
$I_0 \dots I_3$	the four parallel data inputs, one to each flip-flop's D input

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

Signal	Connection
$A_0 \ldots A_3$	the four parallel outputs, taken from each flip-flop's Q
`Clock`	a single common line driven to the C (clock) input of **all four** flip-flops
`Clear`	a single common line driven to the R (reset) input of all four flip-flops, drawn **active-low** (bubble on the input)

So on each rising clock edge all four flip-flops simultaneously capture their I inputs; asserting `Clear` forces all four outputs to 0 asynchronously (see flip-flops-and-latches[3] for asynchronous SET/CLR).

Shift registers

> A **shift register** is a register which is capable of shifting its binary information in one or both directions.

The 4-bit shift register example is built as a **chain**: the serial input SI feeds the D input of the first flip-flop; each flip-flop's Q output feeds the next flip-flop's D input; the last flip-flop's Q is the serial output SO. All four flip-flops share a common CLK line.

On each rising clock edge every stored bit moves one place along the chain, the bit currently on SI enters the first stage, and the bit that was in the last stage leaves at SO.

The **common clock is what makes the shift well-defined**. Every flip-flop samples its D input at the same instant, so each stage captures its neighbour's *old* value, not the new one the neighbour is simultaneously taking on; the whole word moves exactly one place per edge rather than racing along the chain. This is the point of the rule that in a synchronous sequential circuit all the sequential elements share a common clock signal — see sequential-circuits[4].

Worked through by hand, with stages labelled Q_0 (nearest SI) through Q_3 ($= SO$), starting from all zeros and shifting in the bit sequence 1, 0, 1, 1:

Clock edge	SI	Q_0	Q_1	Q_2	Q_3 ($= SO$)
(initial)	—	0	0	0	0
1	1	1	0	0	0
2	0	0	1	0	0
3	1	1	0	1	0

[3] courses/csse2010/flip-flops-and-latches.pdf
[4] courses/csse2010/sequential-circuits.pdf

Clock edge	SI	Q_0	Q_1	Q_2	Q_3 $(= SO)$
4	1	1	1	0	1

After 4 clock edges the 4 serially-supplied bits sit in the 4 flip-flops, and one further edge would push the first of them out of SO.

Serial parallel conversion

Shift registers can be used to do **serial-to-parallel conversion, and vice versa**. This is the reason they show up everywhere data has to travel over a single wire but be *used* a word at a time.

- **Serial in, parallel out**: drive the bits one per clock edge into SI; after n edges, the n flip-flop Q outputs — read as a group — are the parallel word. This is exactly the table above: clock 4 bits in, then read $Q_0Q_1Q_2Q_3$ in parallel.
- **Parallel in, serial out**: load the n bits into the flip-flops in one clock edge (a parallel load — see Figure 3), then read them off SO one bit per edge — the first bit is sitting at SO as soon as the load edge lands, and $n-1$ further edges bring the rest.

> **Note**
>
> The parallel outputs are simply the Q pins of the flip-flops already present in Figure 2, brought out as a group. Nothing extra is added to the circuit for serial-in/parallel-out at all.

Parallel load vs serial shift

A plain shift register can only be filled one bit per clock. To also allow **parallel load** — all n bits written at once from n parallel inputs — each flip-flop's D input is fed from a **2-to-1 multiplexer** (see combinational-logic-blocks[5]) instead of directly from the previous stage:

Mux select	Mux data input chosen	Effect on that stage
"shift"	the previous stage's Q (or SI for the first stage)	serial shift by one place
"load"	that stage's parallel data input I_i	parallel load of the whole word in one clock edge

[5] courses/csse2010/combinational-logic-blocks.pdf

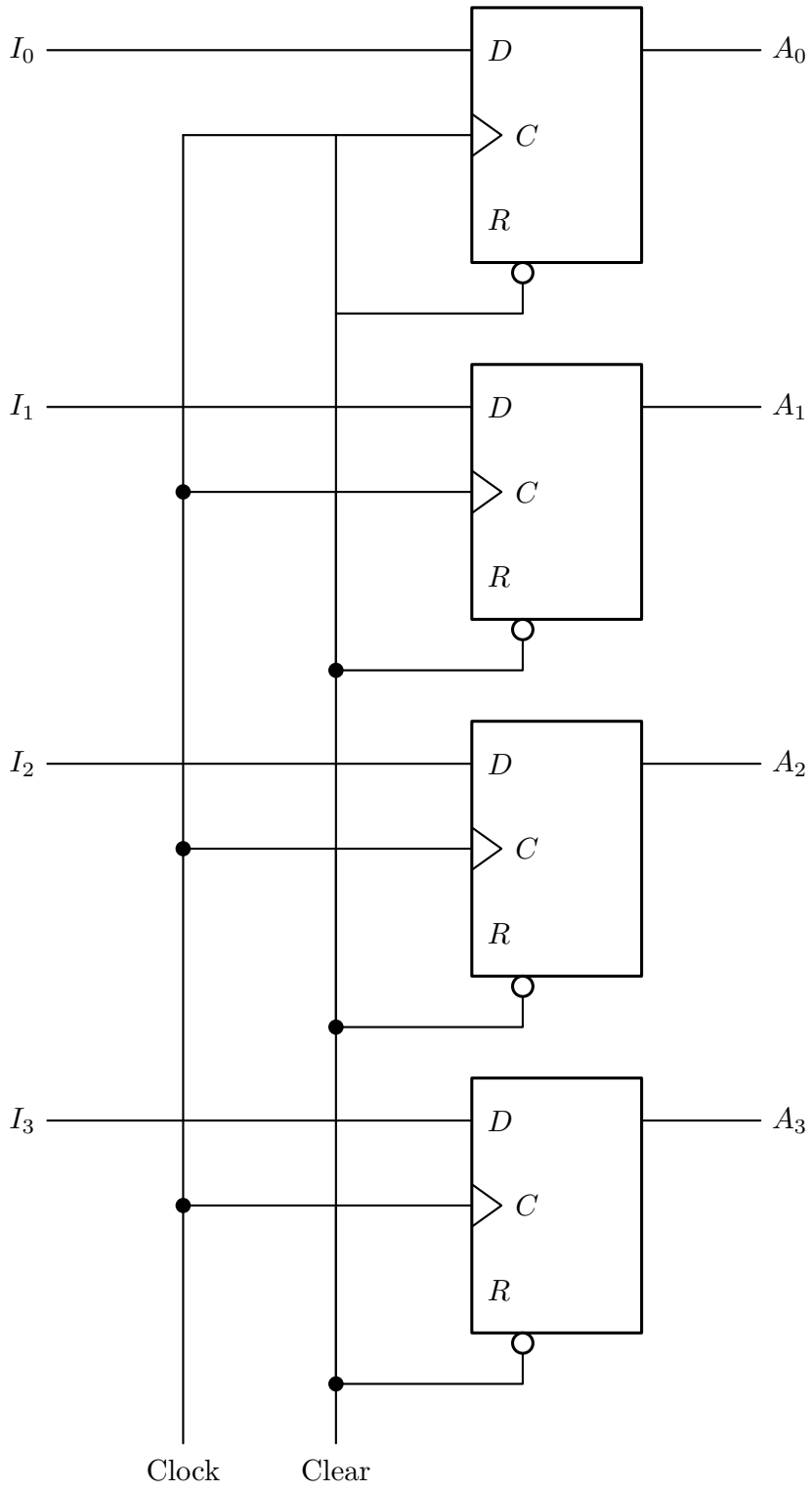

Figure 1: A 4-bit register (Mano, Digital Design, 3rd ed.). Four flip-flops, four parallel inputs, four parallel outputs, and exactly two shared control lines — no gate anywhere, which is what makes a register a sequential circuit without combinational logic. The pin labelling is Mano's, which differs from the rest of the course: the clock input is lettered C rather than CLK and the asynchronous clear is lettered R, marked active-low by the bubble on the pin. The course uses several labellings for both (see the diagram-notation note).

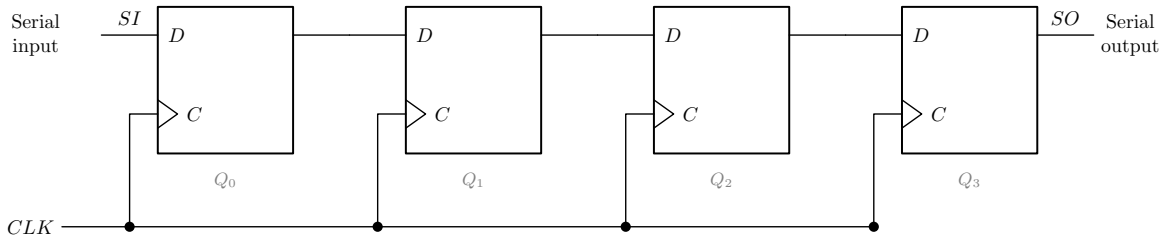

Figure 2: The 4-bit shift register (Mano's Fig. 6-3). The whole device is four D flip-flops and one wire each: the only structural difference from the register above is where each D input comes from. The grey stage names Q0 to Q3 are there so the trace table below can be read against the drawing.

The select line is common to all the muxes, so on each clock edge the register either shifts by one place or loads the whole parallel word.

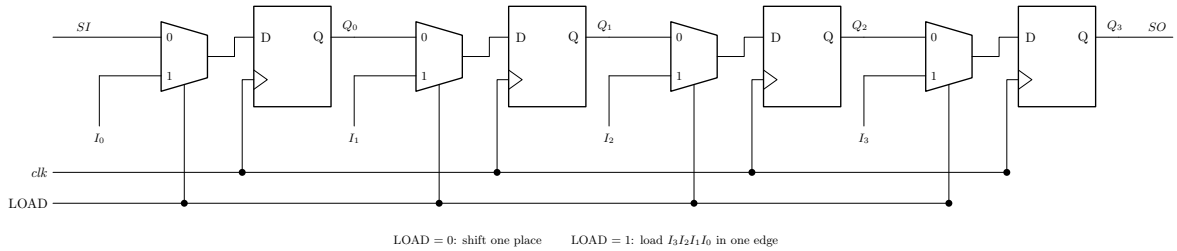

Figure 3: The parallel-load shift register: one 2-to-1 multiplexer in front of every flip-flop's D input, with a single LOAD line steering all four at once. Follow the top mux input across and it is the plain chain of the previous figure; follow the bottom input and every stage is loaded from its own I pin in one edge. The Q pins do double duty as the parallel output. Which mux input is which is a free choice: here LOAD equals 0 for shift and 1 for load.

Adding a hold. The same trick gives a register an *enable*: widen each mux by one input and wire that input to the stage's own Q, so on a "hold" edge every flip-flop reloads the value it already has and the word does not move. This is the standard way to stop a register without gating the clock — gating the clock risks glitching it — and it is what a "recirculating" input means. It follows from the mux construction, and a past-paper question turns on it (csse2010-sample-midsem-1[6] Q18, where each mux's input 0 is its own stage's Q looped back, so the control line chooses shift or hold).

[6] courses/csse2010/csse2010-sample-midsem-1.pdf

The mux-based bidirectional shift element

A single building block makes a shift register that shifts in **either** direction (drawn in the lecture note[7]):

- a **2-to-1 multiplexer** whose output drives the D input of one D flip-flop;
- the mux's select line is a control signal called **DIRN**;
- **DIRN = 0** selects mux input **0** → **left shift**;
- **DIRN = 1** selects mux input **1** → **right shift**;
- the flip-flop's Q is that stage's output, and the flip-flop is clocked from the common clock.

Chaining n of these elements and wiring, for each stage, mux input 0 to the neighbour on the left-shift side and mux input 1 to the neighbour on the right-shift side gives an n-bit **bidirectional shift register**: one shared DIRN line decides which way the whole register shifts on the next clock edge.

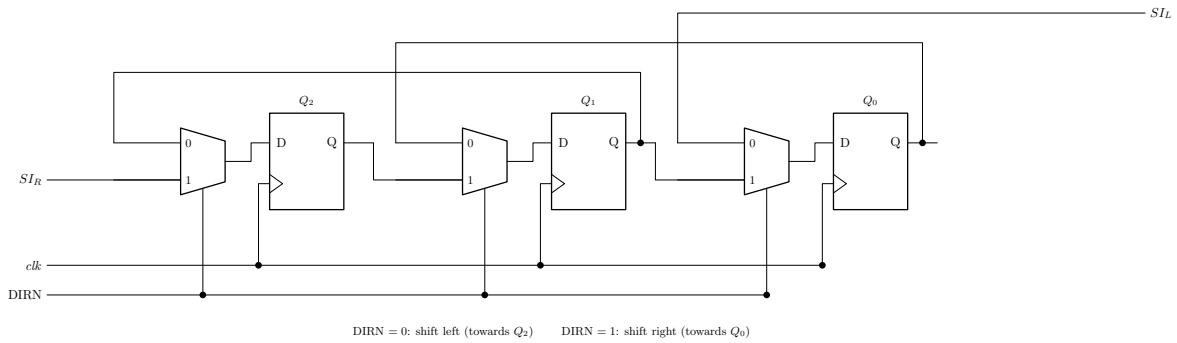

Figure 4: A 3-bit bidirectional shift register: three copies of the DIRN element, one shared DIRN line and one shared clock. Q2 is the most significant bit, on the left. Mux input 0 of every stage comes from the stage on its right (left shift, bits moving towards Q2); mux input 1 comes from the stage on its left (right shift). The two ends need a serial input each, because on each direction one end has no neighbour to take a bit from. Wires that cross without a dot are not connected.

For a 3-bit register with bits $Q_2Q_1Q_0$ (Q_2 most significant), taking *left shift* to mean bits move towards the more-significant end:

Stage	Mux input 0 (DIRN=0, left shift)	Mux input 1 (DIRN=1, right shift)
Q_2	Q_1	SI_R (serial input for right shifts)
Q_1	Q_0	Q_2

[7] courses/csse2010/2026-08-13-shift-registers.pdf

Stage	Mux input 0 (DIRN=0, left shift)	Mux input 1 (DIRN=1, right shift)
Q_0	SI_L (serial input for left shifts)	Q_1

One clock edge applied to a 3-bit register holding $Q_2Q_1Q_0 = 110$, with both serial inputs held at 0:

DIRN	Operation	Before ($Q_2Q_1Q_0$)	After ($Q_2Q_1Q_0$)
0	left shift	110	100
1	right shift	110	011

See 2026-08-13-shift-registers[8] for the exercise that asks for this register.

The configurations a shift register can be wired into

There are **seven named configurations**, parts (a) to (g) of a figure from Floyd, recapped at the start of Lecture 7[9]:

- serial in / shift right / serial out
- serial in / shift left / serial out
- parallel in / serial out
- serial in / parallel out
- parallel in / parallel out
- rotate right
- rotate left

The first six are all the same flip-flop chain differing only in *where each D comes from* and *which pins are brought out*: adding a mux per stage buys the parallel input, tapping the Q pins buys the parallel output, and swapping the direction of the chain buys left instead of right. The last two, the **rotates**, are the chain with its serial output wired back to its serial input, so no bit is lost and the word cycles.

[8] courses/csse2010/2026-08-13-shift-registers.pdf
[9] courses/csse2010/2026-08-17-counters.pdf

Universal shift register

A **universal shift register** is the combination of all of the above capabilities in one device: it can hold its value, shift left, shift right, and be parallel-loaded, with the operation selected by control inputs.

Structurally it is the two constructions above stacked: four operations need a **4-to-1 mux per flip-flop** and **two select lines**, with the four mux inputs being the stage's own Q (hold), its left neighbour, its right neighbour, and its parallel input I_i.

Wide (multi-bit) shift registers

A shift register does not have to shift one *bit* at a time — it can shift a whole multi-bit word per clock edge. An example is an **8-bit wide, 4-stage queue**:

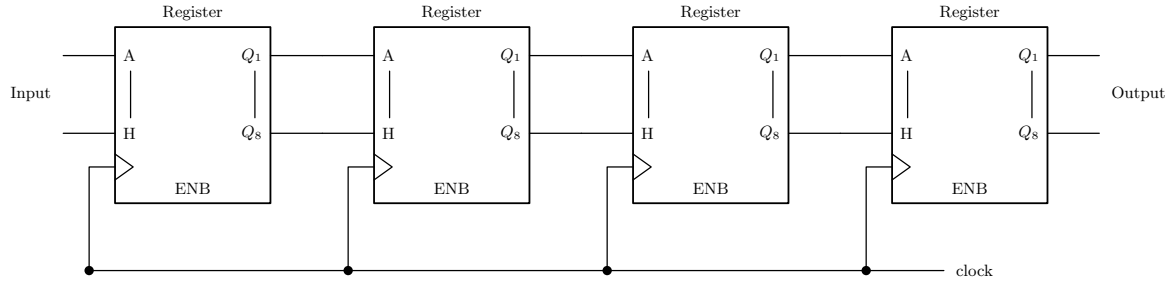

Figure 5: The 4-stage 8-bit queue. Each block is a whole 8-bit register rather than a single flip-flop: the vertical line between A and H, and between Q1 and Q8, is shorthand for the six pins in between. Functionally identical to the 1-bit chain — one stage's outputs are the next stage's inputs, everything on one clock — but each stage now holds a byte, so a byte presented at Input reaches Output four clock edges later.

- four **registers** in a row, each 8 bits wide (inputs labelled A through H, outputs labelled Q_1 through Q_8);
- the 8-bit `Input` bus feeds the first register's 8 data inputs;
- each register's 8 Q outputs feed the next register's 8 data inputs, as a bus;
- the last register's outputs are the 8-bit `Output`;
- all four registers share a **common clock** line, and each has an `ENB` (enable) input.

Functionally this is the same chain as the 1-bit shift register, but each "stage" holds a byte rather than a bit — so it behaves as a 4-deep queue of bytes: a byte presented at `Input` appears at `Output` four clock edges later.

This is also the shape the **Lab 06 combination lock** takes, at a smaller size: a two-stage, 4-bit-wide shift register, so that each press of the "enter" button moves a whole digit from the first stage to the second. See the lecture note[10] and week4-lab-sequential-circuits-shift-registers[11].

Related

- flip-flops-and-latches[12] — the D flip-flop these are built from
- sequential-circuits[13] — state, present/next state, synchronous clocking
- combinational-logic-blocks[14] — the multiplexer used for load/direction selection
- counters[15] — the other classic thing built from a row of D flip-flops
- serial-communication[16] · avr-usart[17] — a shift register at each end of a serial link, turning bytes into a stream of bits and back
- avr-spi[18] — two 8-bit shift registers joined in a ring, so that a byte goes each way at once
- device-pinouts[19] — 74HCT175 quad D flip-flop and 74HCT157 quad 2-input multiplexer pinouts
- csse2010-diagram-notation[20] — the drawing conventions the figures above are checked against

[10] courses/csse2010/2026-08-13-shift-registers.pdf
[11] courses/csse2010/week4-lab-sequential-circuits-shift-registers.pdf
[12] courses/csse2010/flip-flops-and-latches.pdf
[13] courses/csse2010/sequential-circuits.pdf
[14] courses/csse2010/combinational-logic-blocks.pdf
[15] courses/csse2010/counters.pdf
[16] courses/csse2010/serial-communication.pdf
[17] courses/csse2010/avr-usart.pdf
[18] courses/csse2010/avr-spi.pdf
[19] courses/csse2010/device-pinouts.pdf
[20] courses/csse2010/csse2010-diagram-notation.pdf