Lecture 6 — Sequential Circuits 1: Shift Registers

Table of contents

Outline . . . 1
Recap: flip-flops and sequential circuits . . . 1
Registers . . . 2
Shift registers . . . 3
Exercise: bidirectional shift register . . . 3
Universal shift register . . . 4
Wide shift registers . . . 4
Lab 06 preparation task . . . 4
Reminders . . . 7

See shift-registers[1] for the reference material, and sequential-circuits[2] and flip-flops-and-latches[3] for the material recapped at the start.

Outline

- Admin
- Sequential circuits
- Shift registers

Recap: flip-flops and sequential circuits

First met in Lecture 5[4]; the full detail is in the concept notes.

[1] courses/csse2010/shift-registers.pdf
[2] courses/csse2010/sequential-circuits.pdf
[3] courses/csse2010/flip-flops-and-latches.pdf
[4] courses/csse2010/2026-08-10-introduction-to-sequential-circuits.pdf

- **D flip-flop** — D input, Q output, CLK control input; Q copies (and remembers) D on the **rising edge** of CLK. Only D flip-flops are used in this course. Optional **asynchronous SET and CLR** inputs set/clear Q outside clock edges and are typically **active-low**. D flip-flops are what you build sequential circuits from — e.g. counters[5]. Full detail in flip-flops-and-latches[6].
- **Combinational vs sequential circuits** — combinational = logic gates only, output uniquely determined by the inputs, so you always get the same output for a given set of inputs. An example is the pair $AB + AC$ and $A(B + C)$ — two ANDs into an OR beside an OR into an AND, with truth tables that agree; see logic-gates[7]. Sequential = includes flip-flops, output determined by current inputs *and* current **state**, and can change when the clock ticks. See sequential-circuits[8].
- **Sequential circuits** — state = values in the flip-flops; present state vs next state; the output depends on input and state (or sometimes just the state), and the next state depends on inputs and state. Mano's block diagram is a combinational circuit feeding flip-flops, with the flip-flop outputs routed back round to the combinational circuit's inputs and `Clock pulses` entering the flip-flops directly. See sequential-circuits[9], and the counter-structure figure in counters[10] for the same diagram with its external inputs removed.
- **Synchronous sequential circuit** — storage elements can only change at discrete instants. The clock is a train of narrow pulses, and those instants are its rising edges. Outputs of storage elements change **only on the edges** of the control signal, unlike logic gates whose output changes whenever the input does. The rule: **in a synchronous sequential circuit, all the sequential elements share a common clock signal.**

Registers

A **register** is a group of flip-flops: an n-bit register is n flip-flops storing n bits. Two points:

- a register is a sequential circuit **without any combinational logic** — unlike the general sequential-circuit block diagram above, which has a combinational block in the loop;
- registers store binary information (data/instructions) **inside a processor**.

An example is a 4-bit register with parallel inputs $I_0..I_3$, parallel outputs $A_0..A_3$, and common `Clock` and (active-low) `Clear` lines. It is drawn as the register figure in the shift-registers note[11].

[5] courses/csse2010/counters.pdf
[6] courses/csse2010/flip-flops-and-latches.pdf
[7] courses/csse2010/logic-gates.pdf
[8] courses/csse2010/sequential-circuits.pdf
[9] courses/csse2010/sequential-circuits.pdf
[10] courses/csse2010/counters.pdf
[11] courses/csse2010/shift-registers.pdf

Shift registers

A **shift register** is a register capable of shifting its binary information in one or both directions. The example is Mano's Fig. 6-3: serial input $SI \rightarrow$ four D flip-flops in series $\rightarrow$ serial output SO, all on a common CLK. It is drawn, with a clock-by-clock trace, in the shift-registers note[12].

Two applications:

- **Serial parallel conversion** — shift registers can do serial-to-parallel conversion and vice-versa. The starting point is the plain chain of four D flip-flops on a common `clk`: a lead into the first D on the left, each stage's output running straight into the next stage's D. The conversion adds parallel taps to that chain.
- **Parallel load and serial shift** — four D flip-flops on a common `clk` again, with a **multiplexer in front of each flip-flop's D input** to choose between the shift source and the parallel input. See combinational-logic-blocks[13] for the mux, and the shift-registers note[14] for the construction, drawn out.

Exercise: bidirectional shift register

> Using the same multiplexer concept, draw a **3-bit shift register which allows data to be shifted in either direction**.
>
> Hint: build it from the element below, where **DIRN is 0 for left shift, 1 for right shift**.

Answer. Instantiate three copies of the element, share one `DIRN` line and one clock across all three, and for each stage wire

- **mux input 0** (selected when DIRN = 0, left shift) to the neighbouring stage on the left-shift *source* side, and
- **mux input 1** (selected when DIRN = 1, right shift) to the neighbouring stage on the right-shift *source* side,

with an external serial input supplying whichever end of the register has no neighbour in that direction. Writing the bits as $Q_2Q_1Q_0$ with Q_2 most significant, and taking "left shift" to move bits towards the more-significant end:

Stage	Mux input 0 (DIRN=0, left)	Mux input 1 (DIRN=1, right)
Q_2	Q_1	SI_R

[12] courses/csse2010/shift-registers.pdf
[13] courses/csse2010/combinational-logic-blocks.pdf
[14] courses/csse2010/shift-registers.pdf

Stage	Mux input 0 (DIRN=0, left)	Mux input 1 (DIRN=1, right)
Q_1	Q_0	Q_2
Q_0	SI_L	Q_1

Sanity check on $Q_2Q_1Q_0 = 110$ with both serial inputs at 0: one edge with DIRN = 0 gives 100; one edge with DIRN = 1 gives 011. The full wired-up register is drawn in the shift-registers note[15].

The point of the exercise: **a bidirectional shift register is just a unidirectional one with a mux per flip-flop**, and the direction control is a single line shared by every mux, so the whole register commits to one direction per clock edge.

Universal shift register

A universal shift register does hold / shift left / shift right / parallel load in one device, selected by control inputs. See the shift-registers note[16].

Wide shift registers

Shift registers can shift **multiple bits at a time**. An example is a **4-stage, 8-bit queue**: four 8-bit registers in a row (inputs $A..H$, outputs $Q_1..Q_8$, each with an `ENB` enable), each register's outputs feeding the next register's inputs as an 8-bit bus, all on a common clock. A byte at `Input` emerges at `Output` four clock edges later. It is drawn in the shift-registers note[17].

This is also the shape the lab preparation task takes, two stages deep and four bits wide instead of four stages of eight.

Lab 06 preparation task

> **2-digit lock/unlock circuit**: the user inputs two decimal digits (4 bits each) **AB** in serial. The circuit matches the two input digits with a code (say **CD**) and unlocks if the input matches the code (i.e. **AB = CD**).

The **preparation-task handout** specifies the architecture precisely, and it is not what "in serial" might suggest:

[15] courses/csse2010/shift-registers.pdf
[16] courses/csse2010/shift-registers.pdf
[17] courses/csse2010/shift-registers.pdf

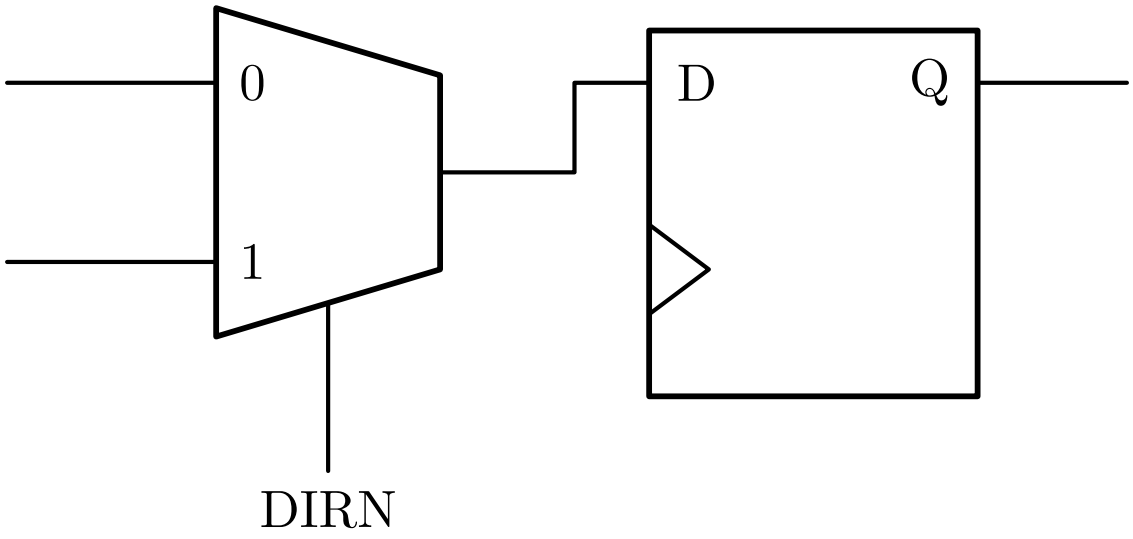

Figure 1: The hint element: a 2-to-1 multiplexer whose data inputs are labelled only 0 and 1, whose select input DIRN enters through the bottom sloping edge, and whose output drives the D input of one edge-triggered D flip-flop. The mux inputs have no names and the clock input no label. One mux plus one flip-flop is a stage, and DIRN picks which neighbour that stage listens to.

- A **two-stage, 4-bit-wide shift register — 8 flip-flops.** The register is *digit*-serial, not bit-serial: each shift moves a whole 4-bit digit from stage 1 to stage 2.
- **Four switches** set a digit value in parallel (0_{10}–15_{10}, i.e. 0000_2–1111_2); a **push button** clocks that digit into stage 1, pushing stage 1's previous contents into stage 2.
- Two LEDs — one for **unlocked**, one for **locked** — plus a **reset** push button. After reset both stages hold `0000`; at all times other than a match, "locked" is on and "unlocked" is off.
- The lock remembers **the last two digits entered**, so it opens on any sequence ending in the code. If the code is 34 and you enter 5, 3, 4, it unlocks on the third digit.

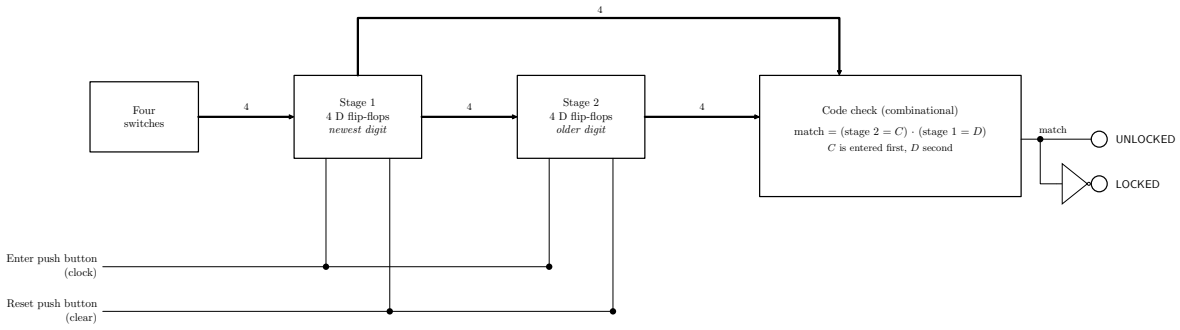

Figure 2: The combination lock at block level. Inputs on the left feed flip-flops in the centre, which feed flip-flops on the right, so the older digit ends up on the right. Hence the match condition shown: the first code digit C is compared against stage 2 and the second digit D against stage 1, not the other way round. Bus widths are lettered beside the heavy lines.

Your code is the **last two digits of your student number**, with any zeroes changed to $10_{10} = 1010_2$; and if the two digits are identical, replace the second with $11_{10} = 1011_2$. Enter the left digit first. (E.g. 40123456 → 5 then 6; 46543210 → 1 then 10; 44444444 → 4 then 11.)

The ordering trap, from the handout's closing NB:

> NB: The natural and eminently reasonable way to begin laying out the logic diagram and/or circuit schematic is to have the inputs on the left, feeding into a set of flipflops in the centre, which in turn feed into a set of flipflops on the right. In this case, the entered code moves left to right, such that the older of the two values is stored in the flipflops on the right and the newer on the left. Ensure that your code checking logic reflects this ordering.

Since you enter the left code digit first, that digit is the *older* one by the time the lock should open, so it is the one sitting in **stage 2**. That is the asymmetry Figure 2 spells out; getting it backwards is a lock that opens on the code reversed.

Deliverables: a **logic diagram** *and* a **circuit schematic**, using the device pinouts from Blackboard (see device-pinouts[18] and circuit-schematics[19]). The handout offers Logisim as an option, not a requirement — *"You may also simulate your logic diagram in Logisim before coming to lab sessions"* — and says the circuit is **built during the practical session**, along with variations on it. The comparison itself is combinational — see combinational-logic-blocks[20]. The lab session this feeds is week4-lab-sequential-circuits-shift-registers[21].

The handout recommends watching the week 3, Lecture 6 recording before attempting this task.

Reminders

- **Labs 6 and 7** (next week, week 4) have **preparation tasks** which should be attempted **before** coming to the labs.
- Attempt the weekly exercise and quizzes.

[18] courses/csse2010/device-pinouts.pdf
[19] courses/csse2010/circuit-schematics.pdf
[20] courses/csse2010/combinational-logic-blocks.pdf
[21] courses/csse2010/week4-lab-sequential-circuits-shift-registers.pdf