Lecture 5 — Introduction to Sequential Circuits

Table of contents

Outline . . . 1
Recap . . . 2
Worked: multiplexer implementing a given function . . . 2
Circuits that remember values . . . 4
The D flip-flop . . . 4
Worked: D flip-flop output waveform . . . 5
SR latch from NOR gates . . . 6
Flip-flops vs latches . . . 7
Combinational vs sequential circuits . . . 8
Reminders . . . 8
See also . . . 8

See flip-flops-and-latches[1] and sequential-circuits[2] for the reference definitions, characteristic tables and symbols.

Outline

- Recap of combinational logic
- Circuits that remember values
- The D flip-flop: symbol, operation, characteristic table, waveforms
- The SR latch built from NOR gates
- Flip-flops vs latches; a real D flip-flop and its chips
- Combinational vs sequential circuits; synchronous sequential circuits

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

Recap

The course so far stacks up in three layers, each built on the one below:

1. **Binary representations** — unsigned, sign-magnitude, 1's complement, 2's complement, excess-2^{N-1} (see binary-number-representations[3], 2026-08-04-binary-arithmetic[4]).
2. **Logic gates** — NOT, AND, OR, NAND, NOR, XOR, XNOR — and Boolean algebra (see logic-gates[5], boolean-algebra[6]).
3. **Combinational logic circuits** — adder, adder/subtractor, multiplexer, decoder (see combinational-logic-blocks[7]).

The circuits built so far are the 4-bit ripple carry adder (four full adders chained by their carries), the 4-bit adder/subtractor with its mode select M ($M = 0$ add, $M = 1$ subtract), the 4:1 MUX with selects S_1S_0, and the 3:8 decoder with inputs A, B, C and outputs $D_0 \ldots D_7$. All of them are combinational. Sequential circuits start here.

Worked: multiplexer implementing a given function

Consider a 4:1 multiplexer with data inputs A, B, C, D on lines 0, 1, 2, 3 respectively and select inputs S_1S_0. What must A, B, C, D be so that the multiplexer output is

$$X = S_1.S_0 + \bar{S}_1.G$$

The options: (1) $A = 0, B = 0, C = 0, D = 1$; (2) $A = G, B = G, C = 0, D = 1$; (3) $A = G, B = 0, C = 0, D = 1$; (4) $A = 0, B = G, C = 0, D = 1$; (5) none of the above.

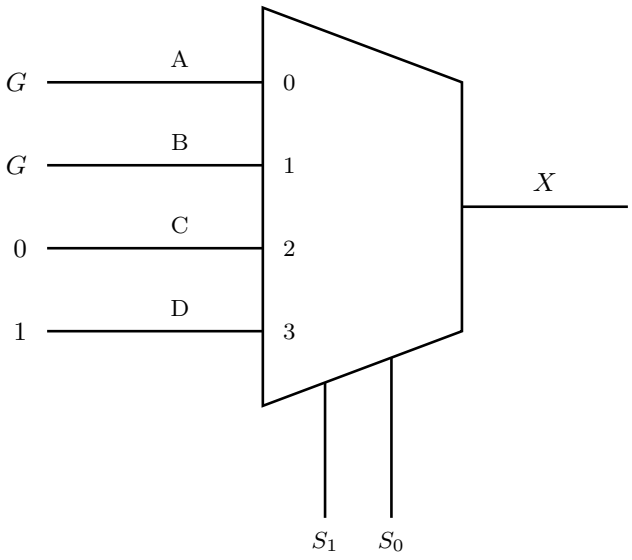
By select code. Walk the four select codes and ask what the multiplexer has to be passing in each case. A 4:1 MUX passes exactly one data input per select code, so whatever X evaluates to for a given S_1S_0 *is* the value that must be sitting on the addressed input.

The same thing algebraically. A 4:1 MUX passes whichever data input its select code addresses, so in general

$$X = \bar{S}_1\bar{S}_0.A + \bar{S}_1S_0.B + S_1\bar{S}_0.C + S_1S_0.D$$

Expand the target into the same four select terms. The $\bar{S}_1.G$ term covers *both* $\bar{S}_1\bar{S}_0$ and $\bar{S}_1S_0$, and the $S_1.S_0$ term is 1 for that one select code only:

[3] courses/csse2010/binary-number-representations.pdf
[4] courses/csse2010/2026-08-04-binary-arithmetic.pdf
[5] courses/csse2010/logic-gates.pdf
[6] courses/csse2010/boolean-algebra.pdf
[7] courses/csse2010/combinational-logic-blocks.pdf



S_1	S_0	$X = S_1.S_0 + \overline{S_1}.G$	input
0	0	$0.0 + \overline{0}.G = G$	$A = G$
0	1	$0.1 + \overline{0}.G = G$	$B = G$
1	0	$1.0 + \overline{1}.G = 0$	$C = 0$
1	1	$1.1 + \overline{1}.G = 1$	$D = 1$

Figure 1: The multiplexer with the answer wired on, and the working beside it. Each row substitutes one select code into the target expression and reads off the value the addressed data input must carry; the four results down the right-hand column are the answer. Rows 1 and 2 both give G — the term S1-bar . G says nothing about S0, so both of the S1 = 0 inputs have to carry G.

$$X = \bar{S}_1\bar{S}_0.G + \bar{S}_1 S_0.G + S_1\bar{S}_0.0 + S_1 S_0.1$$

Matching term by term against the MUX expression: $A = G$, $B = G$, $C = 0$, $D = 1$.

Answer: option 2.

The trap: it's tempting to put G on only one of the two $\bar{S}_1$ lines, but $\bar{S}_1.G$ says nothing about S_0, so *both* $\bar{S}_1$ inputs must carry G. That is what separates option 2 from options 3 and 4.

Two wider points:

- Each of the four data inputs is one of $\{1, 0, G, \bar{G}\}$. Those are the only four things a data input can ever need to be when a multiplexer is used to implement a function of its select variables plus one extra variable G — a constant, that variable, or its complement.
- Using lookup/selection hardware to realise arbitrary logic is the idea programmable logic (FPGAs) is built on.

Circuits that remember values

- The output of any logic gate or **combinational circuit** depends on the **current value of the inputs only**. A two-input adder block is an example: x and y in, S out.
- If an input changes, the output can change too — and the previous value is **lost forever**.
- In a **sequential circuit**, the current output depends not only on the current inputs but also on the **past** outputs.
- Circuits with **memory** can remember values even when the input changes.

That motivates the whole rest of the course: to build anything that accumulates, counts, or holds a result, you need a storage element.

The D flip-flop

The first memory element is the D flip-flop: **D** is the data input, **Q** the output, and **CLK** the control input — the clock. Q copies the value of D — and remembers it — whenever CLK goes from 0 to 1 (the *rising edge*). A D flip-flop stores 1 bit of information.

Everything below depends on telling the two kinds of clock edge apart.

The full symbol set, the characteristic table and a first worked waveform are in flip-flops-and-latches[8].

The characteristic table is the tabular definition of a flip-flop's operation. Its right-hand column $Q(t+1)$ is the *next* output: what the output will be on the next clock edge.

[8] courses/csse2010/flip-flops-and-latches.pdf

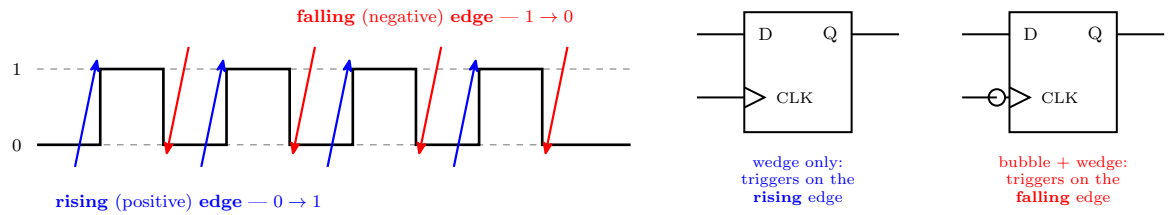


Figure 2: A clock waveform with an up-arrow on every 0-to-1 transition and a down-arrow on every 1-to-0 transition, beside the flip-flop symbols that trigger on each. Colour is only to keep the two edge types apart; it carries no other meaning. A D flip-flop responds to exactly one of these two events, and which one is settled by whether its clock pin carries a bubble.

A D flip-flop remembers a single bit, so n D flip-flops remember n bits and an n-bit **register** is by definition n D flip-flops. JK and T flip-flops also exist but are not covered in this course. Flip-flops can be made out of logic gates.

Worked: D flip-flop output waveform

For the rising-edge D flip-flop above, what is the output waveform for Q, given the D and CLK waveforms in the figure?

Reading the figure. The clock has four rising edges, each marked with an arrow. D is low across the first clock pulse, rises after that pulse has ended, stays high across the second and third clock pulses, and falls again *after* the third rising edge but before the fourth.

Rising clock edge	D at that edge	Q after the edge
1st	0	0
2nd	1	1
3rd	1	1 (no change)
4th	0	0

So Q goes high at the **2nd** rising clock edge and stays high until the **4th** rising clock edge, where it returns to 0. Q is *not* a copy of D: it lags D's rise (waiting for the next clock edge) and lags D's fall (holding the 1 until the next clock edge).

Answer: option 3 — the trace that rises at the second rising clock edge and falls at the fourth rising clock edge.

Why the others are wrong:

- **Option 1** rises and falls exactly with D — that's D itself, i.e. no clocking at all.

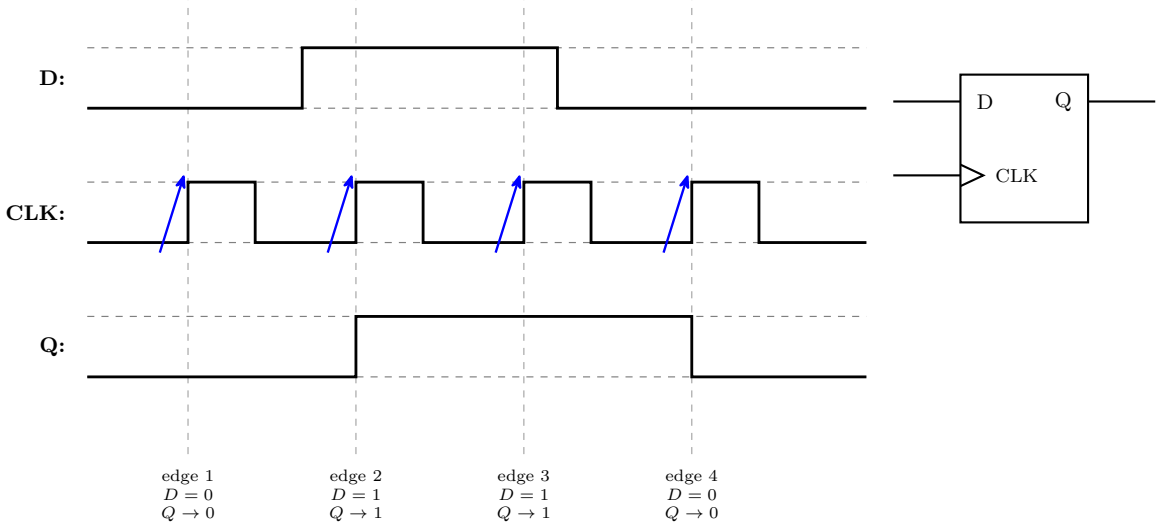


Figure 3: The D and CLK waveforms with the answer, option 3, drawn in as Q. The dashed verticals are the four rising clock edges, each also marked with a blue arrow. Everything about Q is decided at those four instants and nowhere else — which is why Q's edges never line up with D's.

- **Option 2** rises correctly at the second rising edge but falls at the *falling* edge of the third clock pulse — reacting to the wrong edge, as a negative-edge-triggered device would.
- **Option 4** goes high and low several times inside the window where D is high, dropping back to 0 between clock pulses instead of holding — it is following the clock rather than storing anything, so nothing is being remembered at all.

The general check, worth applying to any question of this shape: a rising-edge flip-flop's output can only change *at* a marked rising edge. Any candidate trace with a transition anywhere else is wrong on sight, whatever D is doing there.

SR latch from NOR gates

Two cross-coupled NOR gates make a storage element: S into the top gate (output $\bar{Q}$), R into the bottom gate (output Q), each gate's output fed back into the other gate's spare input. This circuit is the **active high SR latch**.

The truth table: $SR = 00$ holds (Q unchanged, $Q = Q_{n-1}$ — the storage, or remember, row), 01 resets to 0, 10 sets to 1, and 11 is *invalid*. With $S = R = 1$ the outputs are $Q = \bar{Q} = 0$, and the two outputs coming out equal is what makes the combination invalid.

The analysis turns on the identity $\overline{0 + x} = 1 \cdot \bar{x} = \bar{x}$ — a NOR with a 0 on one input is an inverter for the other. Its companion is $\overline{1 + x} = 0$ (a NOR with a 1 on one input is pinned to

0); it is the same De Morgan step and the other half of the analysis, and flip-flops-and-latches[9] carries both.

Circuit, both identities and the full row-by-row derivation are in flip-flops-and-latches[10].

Homework: re-analyse the same S-R latch with the NOR gates replaced by NAND gates, and complete the truth table. The result is the **active low SR latch**: the inputs become $\bar{S}$ and $\bar{R}$ and, importantly, the gate outputs swap over so that the *top* gate now carries Q. That relabelling is what keeps "set" meaning set. Drawn and worked through in flip-flops-and-latches[11].

Flip-flops vs latches

Latches are **level** triggered — they respond to changes of logic level on their inputs. Modify a latch so it responds instead to an **edge** (a momentary transition) of a control input, and you have a **flip-flop**: it stores 1 bit while being sensitive only to a clock edge, so it changes its output only *at* the edges and holds the value in between.

A clock has a positive edge (0→1) and a negative edge (1→0), so a D flip-flop can be positive- or negative-edge triggered — see Figure 2 above. In a **positive edge triggered D flip-flop**, input D is copied to output Q at the positive edge of the clock. In between clock edges the flip-flop is not responsive, so it stores the value.

JK and T flip-flops exist but are **not covered in this course**. The **state** of a flip-flop is the value it is storing. Flip-flops are more useful than latches.

The four schematic symbols (plain / bubble / wedge / bubble-and-wedge on the clock pin, with the wedge meaning edge-triggered and the bubble meaning falling edge) are drawn and tabulated in flip-flops-and-latches[12].

A real D flip-flop's internal schematic is six NAND gates in three cross-coupled pairs. Its $\overline{\text{PRE}}$ and $\overline{\text{CLR}}$ inputs are **active low** and **asynchronous**: they reach into the latches directly and do not wait for a clock edge. The symbol is in flip-flops-and-latches[13].

The chips that package them: the **74HCT74** (dual D flip-flop) and the **74HCT273** (eight flip-flops, i.e. one byte). See device-pinouts[14] for the 74HCT74 pinout — the 74HCT273 is not in the course's device-symbols document, so there is no pinout for it in the vault.

[9] courses/csse2010/flip-flops-and-latches.pdf
[10] courses/csse2010/flip-flops-and-latches.pdf
[11] courses/csse2010/flip-flops-and-latches.pdf
[12] courses/csse2010/flip-flops-and-latches.pdf
[13] courses/csse2010/flip-flops-and-latches.pdf
[14] courses/csse2010/device-pinouts.pdf

Combinational vs sequential circuits

The formal contrast:

	Combinational	Sequential
Contents	Logic gates only, no flip-flops	Includes flip-flops
Output	Uniquely determined by the inputs — the same inputs always give the same output	Determined by the current inputs **and** the current **state** (the values in the flip-flops)
When it can change	Whenever an input changes	Only when the clock 'ticks'
Blocks met so far	Adders, multiplexers, decoders, demultiplexers, encoders	Counters, registers

A combinational example, first met in Lecture 2: A into an AND gate whose other input is $B + C$, giving $A(B + C)$ — a circuit with no memory at all.

The general block diagram (combinational logic + flip-flops + feedback path, with the clock entering the flip-flops only) and the definition of a **synchronous** sequential circuit — storage elements can only change at discrete instants of time, namely the clock edges — are drawn in sequential-circuits[15].

Reminders

Coming up:

- **Lab 4** (Mon–Tue this week) — combinational logic. Make sure you attempt the preparation task. Use logic ICs or Logisim software to test your circuits.
- **Lab 5** (Thu–Fri this week) — flip-flops. Use Logisim software to test the circuits. **Kit loaning happens in Lab 5.**

See also

- flip-flops-and-latches[16] · sequential-circuits[17] — the reference notes for everything above
- combinational-logic-blocks[18] — the multiplexer in the first worked question
- timing-diagrams[19] — how the waveform figures are read

[15] courses/csse2010/sequential-circuits.pdf
[16] courses/csse2010/flip-flops-and-latches.pdf
[17] courses/csse2010/sequential-circuits.pdf
[18] courses/csse2010/combinational-logic-blocks.pdf
[19] courses/csse2010/timing-diagrams.pdf

- week3-lab-logisim-and-flip-flops[20] — Lab 5
- 2026-08-13-shift-registers[21] — Lecture 6, the next one

[20] courses/csse2010/week3-lab-logisim-and-flip-flops.pdf
[21] courses/csse2010/2026-08-13-shift-registers.pdf