

Registers, Shift Registers & the Serial Adder

From one flip-flop to a word of bits: storing it, moving it, and computing with it one bit per clock.



WHERE WE ARE

We can store one bit. Now we need a whole word.

LAST TIME

D latch

Level-sensitive: Q follows D the whole time enable is active.



LAST TIME

D flip-flop

Edge-triggered: D is captured only at the clock edge, then held for a full cycle.



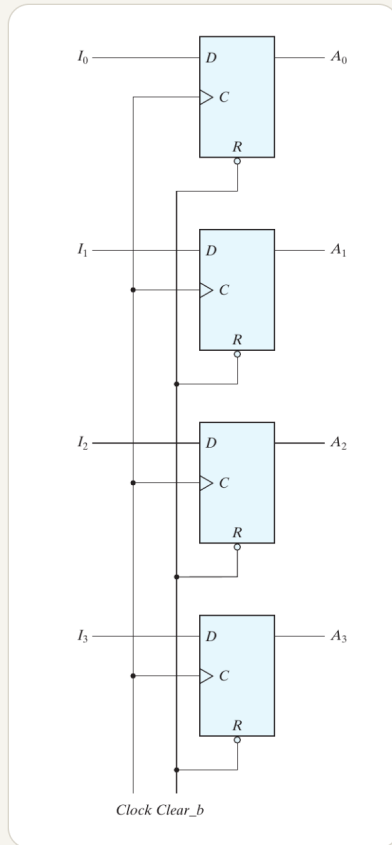
TODAY

n flip-flops

Store a word, hold it, shift it, and add with it, all on one shared clock.

REGISTERS

A register is n flip-flops sharing one clock



Parallel load

The values of *I₃* *I₂* *I₁* *I₀* just before the rising edge appear on *A₃* *A₂* *A₁* *A₀* just after it. All four bits load together.

Asynchronous clear

Clear_b = 0 resets every flip-flop at once, without waiting for the clock. Keep it at 1 during normal operation.

Outputs always readable

A₃ to *A₀* can be read any time; they change only at a clock edge.

Figure 6.1, Four-bit register. Mano & Ciletti, Digital Design, 6th ed.

A QUESTION FOR THE CLASS

A D flip-flop has no "hold" input. Every clock edge loads D. So how does a register keep its value?

Idea 1: hold the inputs constant

Works, but the data bus feeding the register can't be used for anything else.

Idea 2: stop the clock with a gate

AND the clock with a Load signal. Tempting, but a bad idea. Why?

REGISTERS

Why we don't put gates in the clock path

01

Skew

The AND gate delays this register's clock. Its edge now arrives later than every other flip-flop's, so the system is no longer synchronous.

02

False edges

If Load changes while the clock is high, the gate's output gets an extra rising edge, and the register loads when it shouldn't.

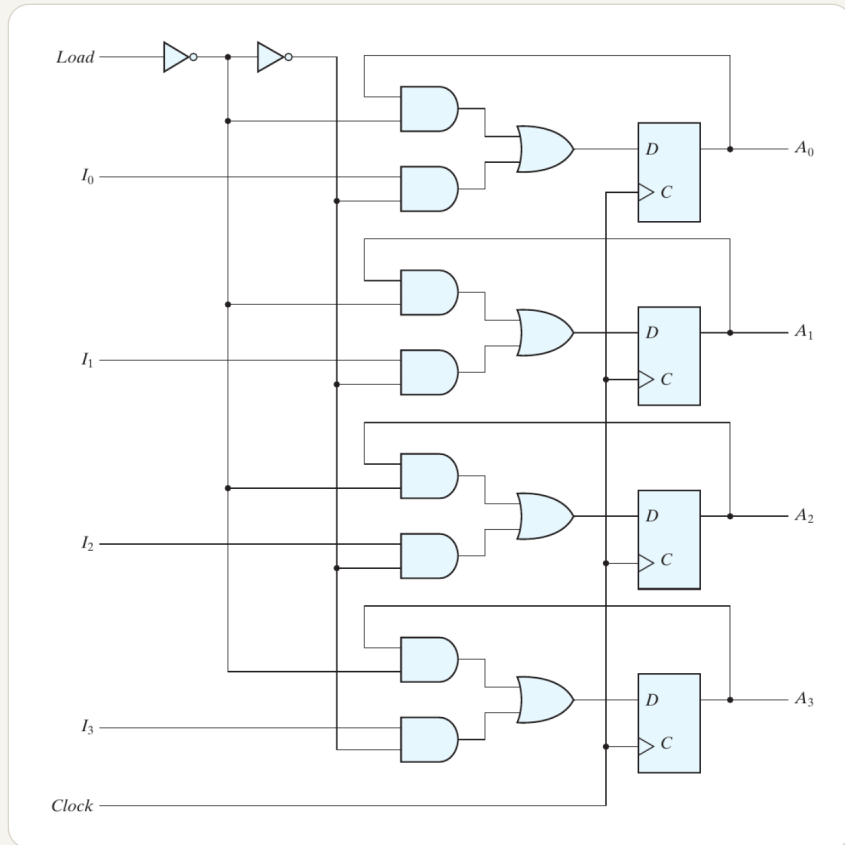
Rule

Gate the data, not the clock

Let the clock reach every flip-flop untouched. Decide with logic on D what gets loaded.

REGISTERS

Hold by feeding Q back into D



$$D_i = \text{Load} \cdot I_i + \text{Load}' \cdot A_i$$

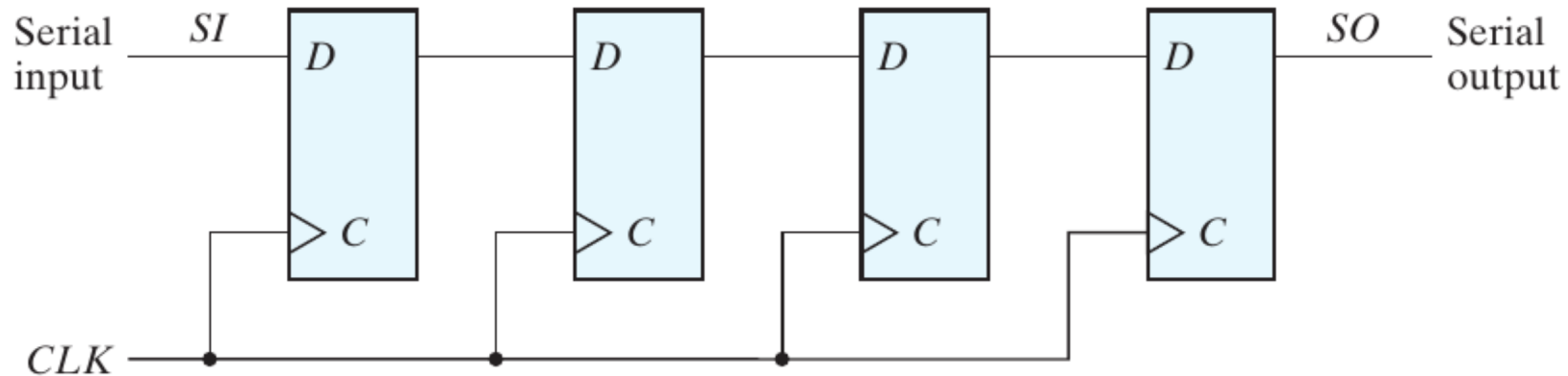
Load	At the next clock edge
1	New data I ₃ ..I ₀ is loaded
0	A ₃ ..A ₀ reloads itself: no change

The AND–OR pair is a **2-to-1 multiplexer** in front of each D. The clock goes to every flip-flop without passing through any gate.

Figure 6.2, Four-bit register with parallel load. Mano & Ciletti, Digital Design, 6th ed.

SHIFT REGISTERS

Chain the flip-flops: each Q drives the next D



Serial input, SI

Sets what enters the leftmost stage.

Every clock edge

All bits move one place to the right, together.

Serial output, SO

The bit leaving the rightmost stage.

SHIFT REGISTERS

Why the chain needs flip-flops, not latches

WITH D LATCHES

Data races through

While the clock is high, every latch is transparent at once. SI passes through stage 1, straight into stage 2, then 3, then 4, all in one pulse.

How far it gets depends on gate delays and pulse width: **unpredictable**.

WITH EDGE-TRIGGERED D FLIP-FLOPS

Exactly one step per edge

At the edge, every stage samples its neighbour's *old* output. Outputs change only a propagation delay later, after sampling is over.

Each bit moves **one place**, every time.

SHIFT REGISTERS

Trace: shifting in the serial stream 1, 0, 1, 1

Edge	SI	Q1	Q2	Q3	Q4
Start	–	0	0	0	0
After T1	1	1	0	0	0
After T2	0	0	1	0	0
After T3	1	1	0	1	0
After T4	1	1	1	0	1

The first bit in ends up in Q4, the farthest stage.

After 4 edges, 4 bits that arrived one at a time can be read **all at once** from Q1..Q4.

That is serial-to-parallel conversion.

SHIFT REGISTERS

Four ways data can go in and come out

SISO

Serial in, serial out

Delays a bit stream by n clocks.

SIPO

Serial in, parallel out

Receiver: collects bits from one wire into a word.

PISO

Parallel in, serial out

Transmitter: sends a word down one wire.

PIPO

Parallel in, parallel out

An ordinary storage register.

Why go serial? Over a long distance, one wire is far cheaper than n wires. The transmitter converts parallel to serial; the receiver converts back.

SHIFT REGISTERS

Serial transfer: copying A into B over one wire

Pulse	Register A	Register B
Initial	1011	0010
After T1	1101	1001
After T2	1110	1100
After T3	0111	0110
After T4	1011	1011

SO of A → SI of B. A's rightmost bit enters B's leftmost stage each pulse.

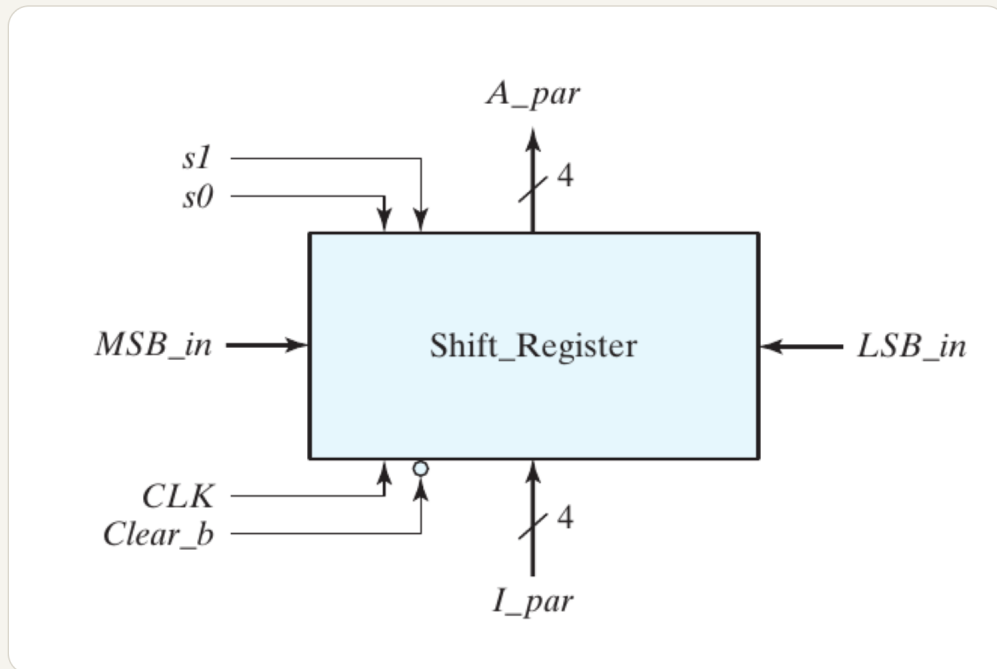
SO of A → SI of A too. A recirculates, so after 4 shifts A is back to 1011, not lost.

B's old contents fall off its right end.

**Shift must be enabled for exactly 4 clocks.
Something has to count.**

UNIVERSAL SHIFT REGISTER

One register that can do everything



Hold: keep the value

Shift right: new bit from MSB_in

Shift left: new bit from LSB_in

Parallel load: from I_par

Plus: clock, clear, n parallel outputs

Two mode bits, $s1$ $s0$, pick one of four operations.

UNIVERSAL SHIFT REGISTER

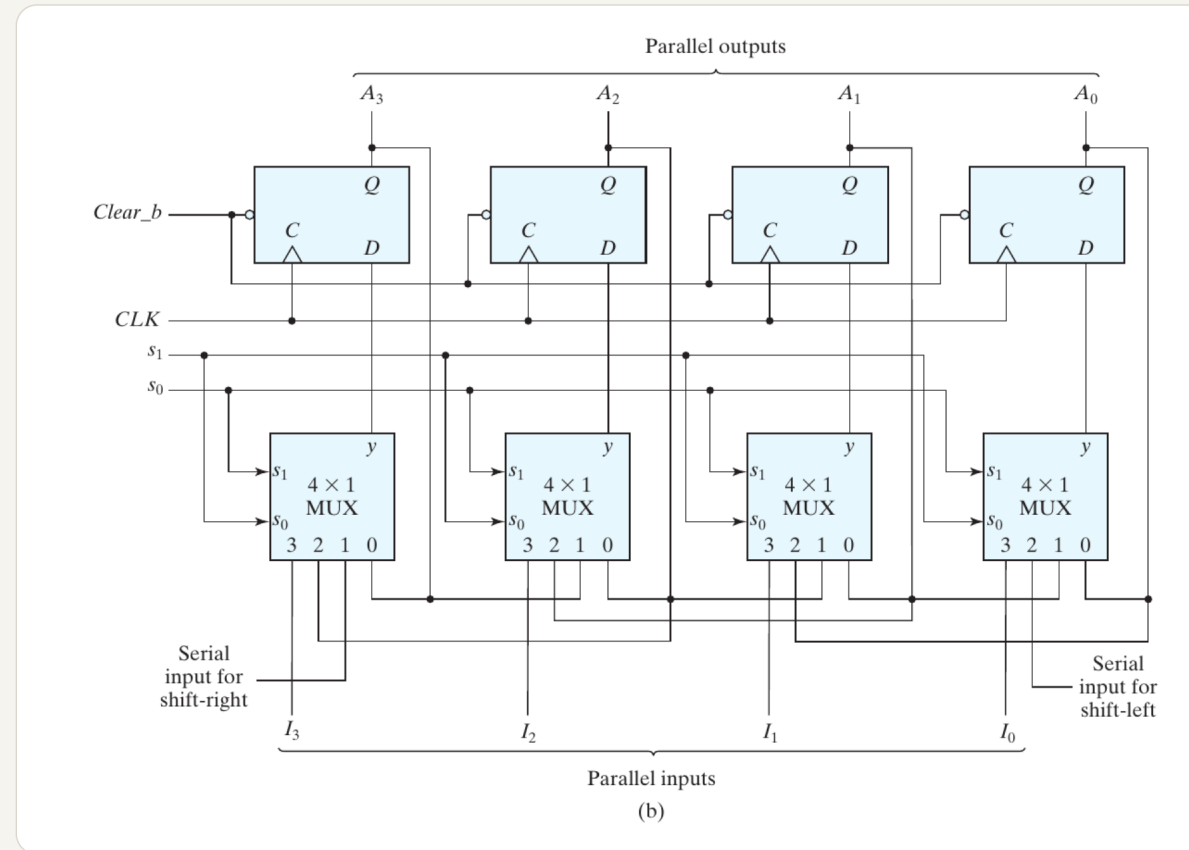
Same trick, bigger mux: a 4-to-1 mux in front of each D

s1	s0	Operation	Mux input selected for Di
0	0	No change	0: its own output A_i
0	1	Shift right	1: left neighbour A_{i+1} (A_3 takes MSB_in)
1	0	Shift left	2: right neighbour A_{i-1} (A_0 takes LSB_in)
1	1	Parallel load	3: parallel input I_i

Compare with the parallel-load register: there a 2-to-1 mux chose between *hold* and *load*. Here a 4-to-1 mux chooses among four sources. The clock path is still untouched.

UNIVERSAL SHIFT REGISTER

The full circuit: four flip-flops, four muxes



Trace the A2 mux

Input	Wired to
0	A2 (hold)
1	A3 (shift right)
2	A1 (shift left)
3	I2 (load)

Real part: the 74194 (74HC194) 4-bit universal shift register.

YOUR TURN

Start at A3 A2 A1 A0 = 0000. Fill in the register after each edge.

Edge	s1 s0	Operation	Other input	A3 A2 A1 A0
T1	1 1	Load	I = 1010	????
T2	0 1	Shift right	MSB_in = 1	????
T3	1 0	Shift left	LSB_in = 0	????
T4	0 0	No change	—	????

Bonus: after T3, is the register back where it was after T1? Was that guaranteed?

ANSWER

Start at A3 A2 A1 A0 = 0000. Fill in the register after each edge.

Edge	s1 s0	Operation	Other input	A3 A2 A1 A0
T1	1 1	Load	I = 1010	1 0 1 0
T2	0 1	Shift right	MSB_in = 1	1 1 0 1
T3	1 0	Shift left	LSB_in = 0	1 0 1 0
T4	0 0	No change	—	1 0 1 0

Not guaranteed. The shift right pushed A0 = 0 out and it was lost. We got 1010 back only because LSB_in happened to be 0.

SERIAL ADDER

A ripple-carry adder spreads the carry across space. A serial addder spreads it across time.

RIPPLE CARRY (CH. 4)



n full adders, one clock. The carry ripples through wires.

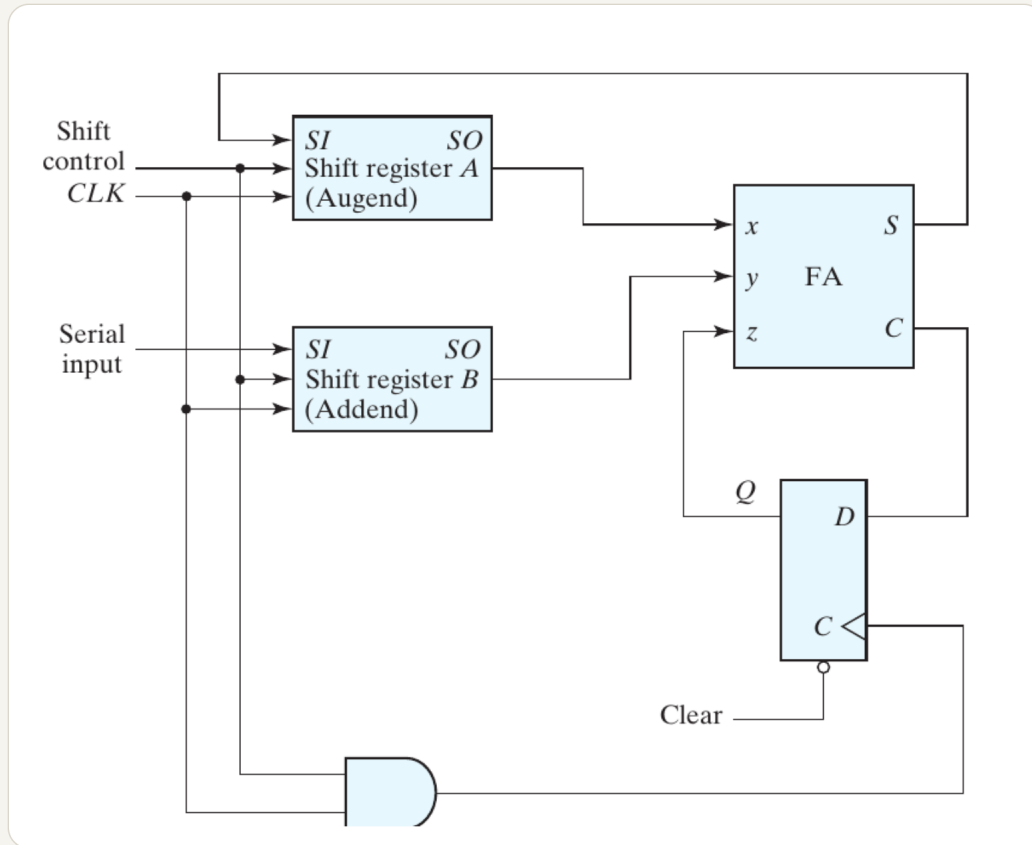
SERIAL (TODAY)



One full adder, used n times. The carry waits in a flip-flop.

SERIAL ADDER

Two shift registers, one full adder, one carry flip-flop



Before starting: A holds the augend, B the addend, and the carry flip-flop is cleared to 0.

During each clock: the FA adds $x = A_0$, $y = B_0$ and $z = Q$, the stored carry.

At each edge: both registers shift right, S enters A's left end, and C is stored in Q.

After n edges: A holds the sum. Its old bits were shifted out as the sum bits came in.

Figure 6.5, Serial adder. Mano & Ciletti, Digital Design, 6th ed.

SERIAL ADDER

Trace: A = 0101 (5), B = 0111 (7)

State	A	B	Q (carry)	FA sees x y z	S	C
Start	0101	0111	0	1 1 0	0	1
After T1	0010	0011	1	0 1 1	0	1
After T2	0001	0001	1	1 1 1	1	1
After T3	1000	0000	1	0 0 1	1	0
After T4	1100	0000	0	done	–	–

A = 1100 = 12. Each row's S enters A's left end and its C becomes the next row's Q. Final Q = 0, so no overflow out of 4 bits.

SERIAL ADDER

The carry flip-flop stores state, not just data

Q now	x	y	Q next	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

It is the full-adder truth table, renamed: carry-in becomes the **present state** Q, and carry-out becomes the **next state**.

$$S = x \oplus y \oplus Q$$

$$Q(\text{next}) = xy + xQ + yQ$$

S depends on the present inputs *and* on what happened at earlier clock edges. That is exactly what made the SR latch sequential.

What the datapath can't do by itself

01

Clear the carry first

Q still holds the last carry from the previous addition. Forget to clear it and the new sum is off by one.

02

Shift exactly n times

One shift too many and the sum bits start going round again. One too few and the sum is incomplete.

Next

Who counts to n?

A control unit built around a **counter**: the topic of the next lecture.

SERIAL ADDER

The trade-off: hardware versus clock cycles

For an n-bit add	Ripple-carry (parallel)	Serial
Full adders	n	1
Extra flip-flops	none	1 (carry)
Operand registers	parallel load	shift registers
Clock cycles	1	n
Delay per cycle	about n full-adder delays	about 1 full-adder delay

Serial is never faster. Total time is still about n full-adder delays, plus flip-flop overhead each cycle.

The saving grows with n. Registers cost area too, so for small words the gap is modest.

SUMMARY

Everything today was D flip-flops plus a mux or an adder

Register

n flip-flops, one clock. Hold with a mux on D, never a gate on the clock.

Shift register

Q to the next D. Works only because flip-flops are edge-triggered.

Universal shift register

A 4-to-1 mux per bit: hold, shift right, shift left, load.

Serial adder

One full adder and a carry flip-flop: less hardware, n clock cycles.

Think about for next time: connect a D flip-flop's Q' to its own D. What does Q do on each clock edge?