



SEQUENTIAL LOGIC • CHAPTER 6

Counters

Registers that step through a sequence of states, built entirely from D flip-flops.

Part 1 • Ripple and synchronous binary counters

Part 2 • Load, mod-N, ring and Johnson

Based on Mano & Ciletti, Digital Design (6th ed.), Sections 6.3–6.5. Two lectures, one deck.

PART 1 · WHAT IS A COUNTER?

A counter is a register that steps through a fixed sequence of states

Binary counter

n flip-flops count 0, 1, 2, ... $2^n - 1$, then wrap to 0.

Mod-N counter

Repeats a sequence of N states. A BCD counter is mod-10.

Any sequence

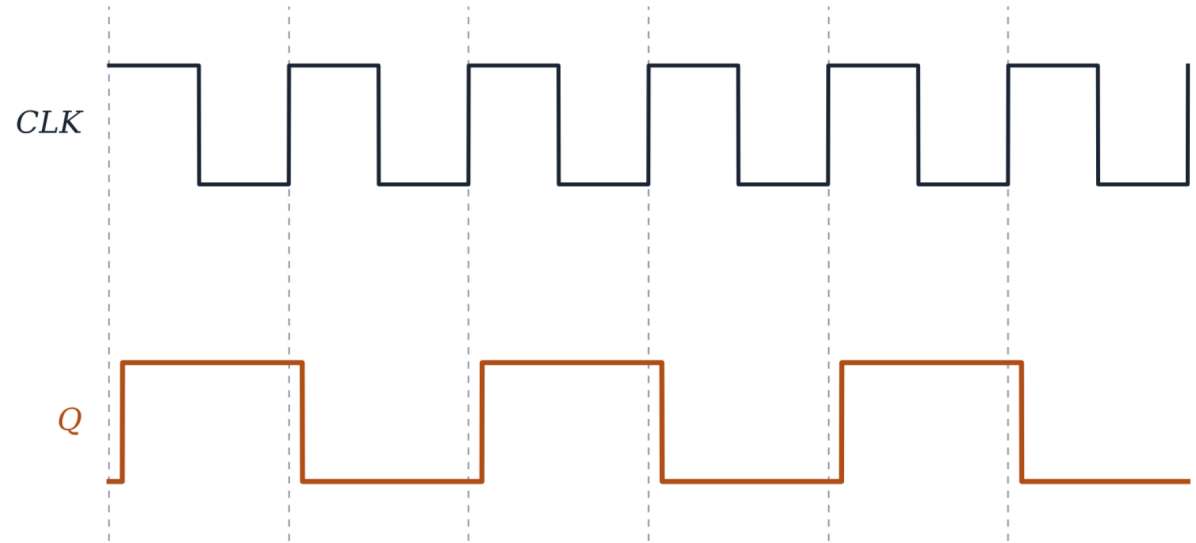
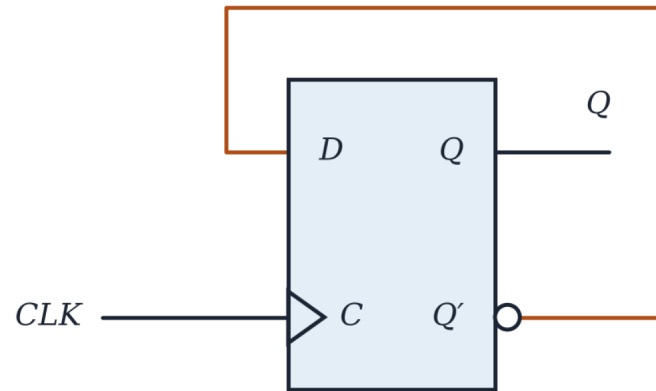
Need not be binary at all: ring and Johnson counters, in Part 2.

WHAT THEY'RE USED FOR

Counting events · dividing a clock frequency · generating timing signals · doing something exactly n times, like the serial adder's n shifts

PART 1 • THE BUILDING BLOCK

Feed Q' back into D , and the flip-flop toggles



D always equals the opposite of Q , so each clock edge **flips** Q .

Q completes one cycle every two clock cycles: a **divide-by-2** circuit.

PART 1 • LOOK AT THE COUNT

Which bits change from one count to the next?

Count	A3 A2 A1 A0	Bits that flip next
0	0 0 0 0	A0
1	0 0 0 1	A1 A0
2	0 0 1 0	A0
3	0 0 1 1	A2 A1 A0
4	0 1 0 0	A0
5	0 1 0 1	A1 A0
6	0 1 1 0	A0
7	0 1 1 1	A3 A2 A1 A0
8	1 0 0 0	...

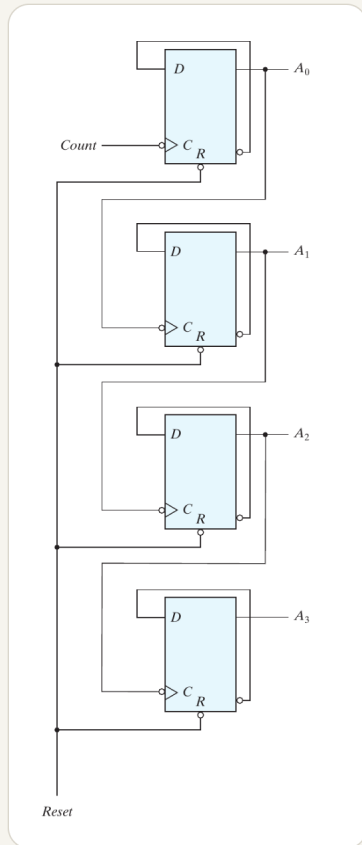
A0 flips on every count.

Each other bit flips when the bit just below it goes from 1 to 0.

So each stage is a toggle, triggered by the falling edge of the stage before it.

PART 1 • RIPPLE COUNTER

Chain the toggles: each stage clocks the next



Every stage is a toggle

Q' feeds D, exactly as on the previous slide.

Only A₀ sees the count pulses

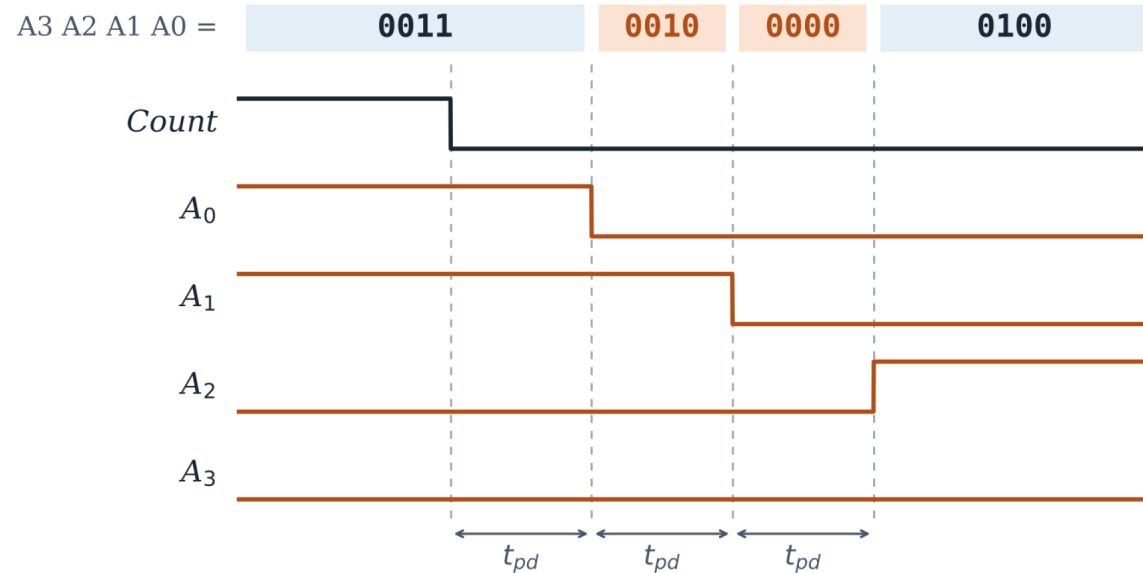
Stage i is clocked by A_{i-1} , not by a common clock.

Negative-edge triggered

The bubble on C means a stage toggles when the stage before it goes $1 \rightarrow 0$. That is exactly the pattern in the count table.

PART 1 · RIPPLE COUNTER

The problem: going from 3 to 4 passes through 2 and 0



Each stage waits for the one before it, so the change **ripples** through, one t_{pd} per stage.

For a moment the outputs show **false counts**: 0010 and 0000.

With n stages, the count is only correct after up to $n \times t_{pd}$.

PART 1 • YOUR TURN

Make the ripple counter count down

Wanted sequence

0000 → 1111 → 1110 → 1101 → 1100 → ...

Step 1: when does a bit flip when counting down?

Step 2: change one connection per stage in Fig. 6.8(b).

Answer

Counting down, a bit flips when the bit below it goes **0 → 1**.

So clock each stage from the previous stage's **Q'** instead of Q. Q' falls exactly when Q rises.

PART 1 · SYNCHRONOUS COUNTER

Give every flip-flop the same clock. Then each one needs a rule: flip now, or not?

0 0 1 1

A0 = 1, so **A1 flips**. A1 = 1 too, so **A2 flips**.

0 1 1 1

A2 A1 A0 are all 1, so **A3 flips** as well.

0 1 0 1

A1 = 0, so neither A2 nor A3 flips. Only A0 and A1 do.

Rule: bit i flips when **all bits below it are 1**. A0 always flips.

PART 1 · SYNCHRONOUS COUNTER

"Flip if" becomes an XOR in front of D

THE XOR TRICK

Expression	Meaning for D
$A \oplus 0 = A$	Keep the bit
$A \oplus 1 = A'$	Flip the bit

So $D = A \oplus (\text{condition to flip})$ does exactly what we want.

4-BIT UP COUNTER

$$D_0 = A_0 \oplus 1 = A_0'$$

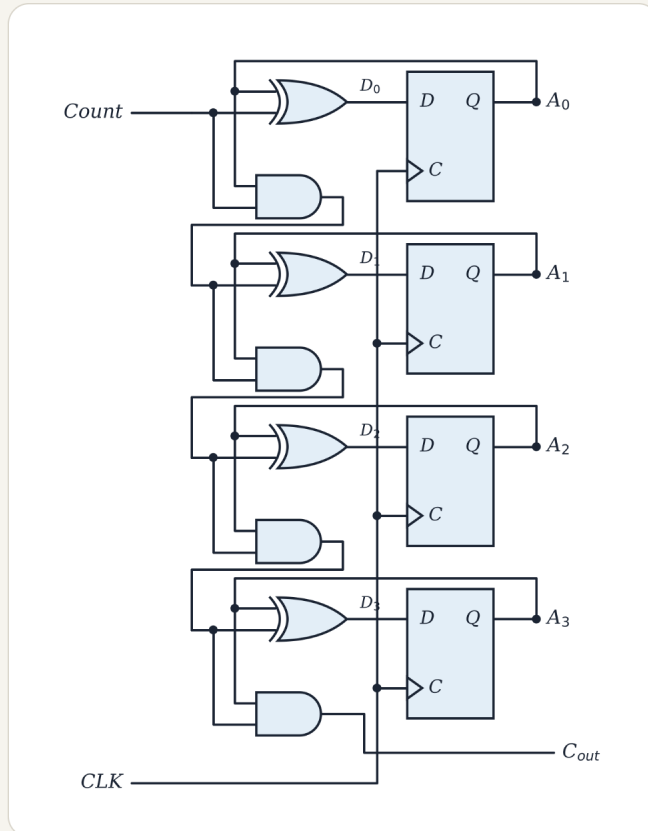
$$D_1 = A_1 \oplus A_0$$

$$D_2 = A_2 \oplus A_1 \cdot A_0$$

$$D_3 = A_3 \oplus A_2 \cdot A_1 \cdot A_0$$

PART 1 • SYNCHRONOUS COUNTER

The circuit: XOR per bit, AND chain, one clock



One clock drives every flip-flop, so all bits change together. No false counts.

The AND chain builds each flip condition: Count, Count·A₀, Count·A₀·A₁, ...

Count enters at the top. With Count = 1 the equations are those on the last slide.

Count = 1 when Count = 1 and the counter is at 1111: the next edge wraps to 0000.

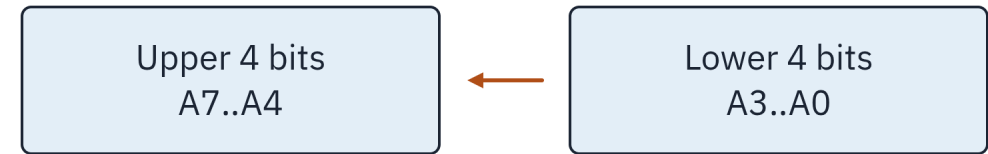
Count enable and Cout come for free

Count = 0 means hold

Every flip condition becomes 0, so every $D_i = A_i \oplus 0 = A_i$.

The counter pauses while the clock keeps running.
Same rule as last lecture: **gate the data, not the clock.**

Chain counters with Cout



Lower Cout drives upper Count. Same clock for both: an 8-bit synchronous counter.

PART 1 • COMPARISON

Ripple versus synchronous

	Ripple	Synchronous
Clock	Each stage clocked by the previous output	One clock shared by all stages
Outputs change	One after another	All together
False counts	Yes, briefly, during the ripple	No
Settling time	Grows with n: up to n flip-flop delays	One flip-flop delay
Extra logic	None	An XOR per bit and an AND chain

PART 1 · THE PATTERN TO REMEMBER

$$D = Q \oplus (\text{when to flip})$$

Every counter in Part 1 is this one idea. Only the "when to flip" logic changes: all lower bits 1 for counting up, all lower bits 0 for counting down.

Keep this in mind: later we will package the XOR together with the flip-flop and give it a name of its own.

PART 1 · UP-DOWN COUNTER

Counting down: a bit flips when all lower bits are 0

DOWN ONLY

$$\begin{aligned}D_0 &= A_0 \oplus 1 \\D_1 &= A_1 \oplus A_0' \\D_2 &= A_2 \oplus A_1' \cdot A_0' \\D_3 &= A_3 \oplus A_2' \cdot A_1' \cdot A_0'\end{aligned}$$

Same structure; the AND chain uses the complemented outputs.

BOTH, WITH UP AND DOWN CONTROLS

$$\begin{aligned}\text{flip}_i &= \text{Up} \cdot (A_{i-1} \cdots A_0) \\&+ \text{Up}' \cdot \text{Down} \cdot (A_{i-1}' \cdots A_0')\end{aligned}$$

Up	Down	Operation
1	X	Count up
0	1	Count down
0	0	No change

PART 1 · YOUR TURN

Three quick problems

01

Write D2, D1, D0 for a **3-bit synchronous down counter** using D flip-flops. Check that 000 goes to 111 and 100 goes to 011.

02

A 4-bit counter is clocked at 16 MHz with Count = 1. Which output is a square wave at 1 MHz?

03

A 4-bit ripple counter has $t_{pd} = 10$ ns per stage. How long does 0111 $\rightarrow$ 1000 take to settle? Which false counts appear on the way?

PART 1 · ANSWERS

Three quick problems

01 $D0 = A0'$ $D1 = A1 \oplus A0'$ $D2 = A2 \oplus A1' \cdot A0'$
000: D = 1, 1, 1 → 111. 100: $D2 = 1 \oplus 1 = 0$, $D1 = 1$, $D0 = 1$ → 011.

02 **A3.** Each stage halves the frequency: $A0 = 8$ MHz, $A1 = 4$, $A2 = 2$, $A3 = 1$ MHz.

03 **40 ns.** All four stages flip in turn: 0111 → 0110 → 0100 → 0000 → 1000. Three false counts.

PART 1 • RECAP

What we built in Part 1

Toggle

D = Q' flips on every edge and divides the clock by 2.

Ripple counter

Toggles clocking each other. Simple, but slow to settle and shows false counts.

Synchronous counter

One clock. $D_i = A_i \oplus$ (all lower bits 1). Count enable holds; Cout chains.

Up-down counter

Same form; counting down flips when all lower bits are 0.

COUNTERS · PART 2

Controlling counters, and other sequences

Parallel load

Mod-N and BCD counting

Ring and Johnson counters

PART 2 · COUNTER WITH PARALLEL LOAD

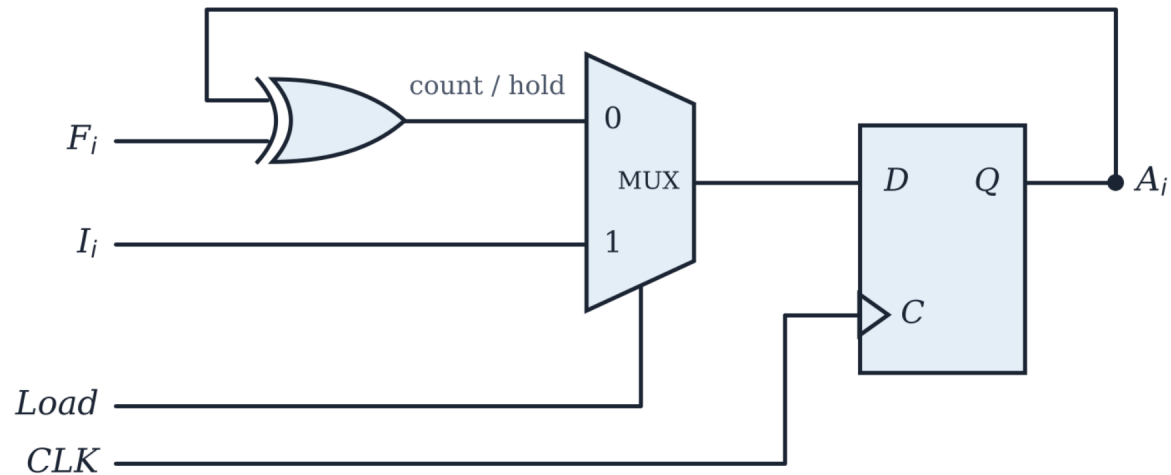
A counter that can also be loaded with any starting value

Clear_b	CLK	Load	Count	Function
0	X	X	X	Clear to 0 (immediately, no clock)
1	↑	1	X	Load inputs I3..I0
1	↑	0	1	Count to the next binary state
1	↑	0	0	No change

Load has priority over Count. Why load a counter? To start counting from any value, and, as we will see, to make it count to any N.

PART 2 · COUNTER WITH PARALLEL LOAD

Build it with D: a mux chooses "load" or "count"

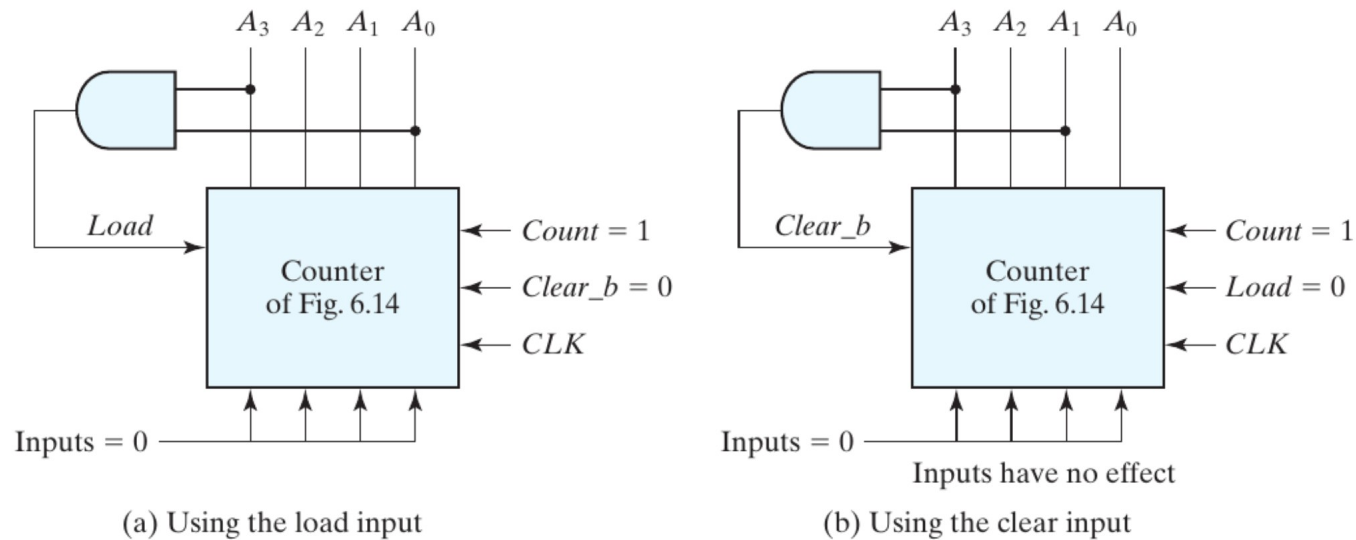


$$D_i = \text{Load} \cdot I_i + \text{Load}' \cdot (A_i \oplus F_i)$$

$F_i = \text{Count} \cdot A_{i-1} \cdots A_0$, from the AND chain in Part 1.

It is last lecture's parallel-load register, with "hold" upgraded to "count or hold".

A BCD counter: detect 9, then load 0



(a) At count 1001, $A_3 \cdot A_0 = 1$ asserts Load. The next edge loads 0000 instead of counting.

Why is $A_3 \cdot A_0$ enough? Among 0000 to 1001, only 1001 has both bits set.

Result: 0, 1, ... 9, 0, ... ten states, all decided at clock edges.

Why not just clear the counter at 10?

FIG. 6.15(B): ASYNCHRONOUS CLEAR

... 1001 → 1010 → 0000

The NAND sees 1010 and clears at once, without waiting for a clock edge.

So 1010 **does** appear, for a few gate delays. A1 gives a short false pulse: a **glitch**.

FIG. 6.15(A): SYNCHRONOUS LOAD

... 1001 → 0000

The decision is made during 1001 and acted on at the next clock edge.

No extra state, no glitch. **Prefer this.**

Same lesson as gating the clock: let the clock edge decide when things change.

PART 2 · YOUR TURN

Use the loadable counter. What drives Load, and what goes on I3..I0?

01

Mod-6 counter

0, 1, 2, 3, 4, 5, 0, ...

02

Count 3 to 12

3, 4, ... 11, 12, 3, ...

Hint: find the smallest AND gate that is 1 only at the last state, among the states the counter actually reaches.

PART 2 · ANSWERS

Use the loadable counter. What drives Load, and what goes on I3..I0?

01

Mod-6 counter

Last state 5 = 0101. **Load** = $A_2 \cdot A_0$; inputs 0000.

Only 5 has $A_2 = A_0 = 1$ among 0 to 5.

02

Count 3 to 12

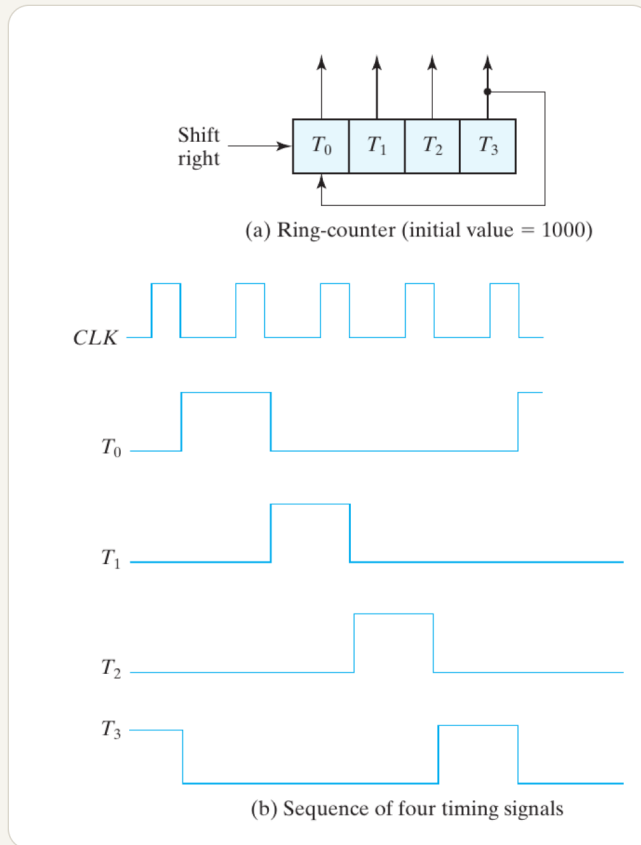
Last state 12 = 1100. **Load** = $A_3 \cdot A_2$; inputs 0011.

Only 12 has $A_3 = A_2 = 1$ among 3 to 12.

Recipe: detect the last state, load the first. The counter then repeats N states: a **divide-by-N** counter.

PART 2 • SHIFT-REGISTER COUNTERS

Ring counter: one 1 circulating in a shift register



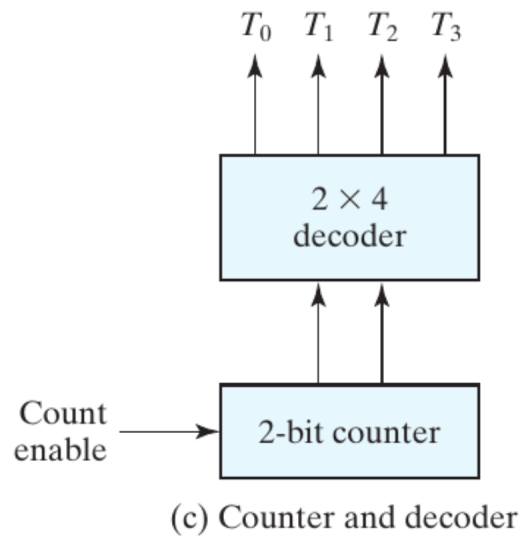
Shift right, with the last output fed back to the first input. Start at 1000.

1000 → 0100 → 0010 → 0001 → 1000

Each output is 1 for exactly one clock in four: four **timing signals** $T_0..T_3$, ready to use with no decoding.

k flip-flops give only k states, and the register must be preset to 1000.

The alternative: a binary counter plus a decoder

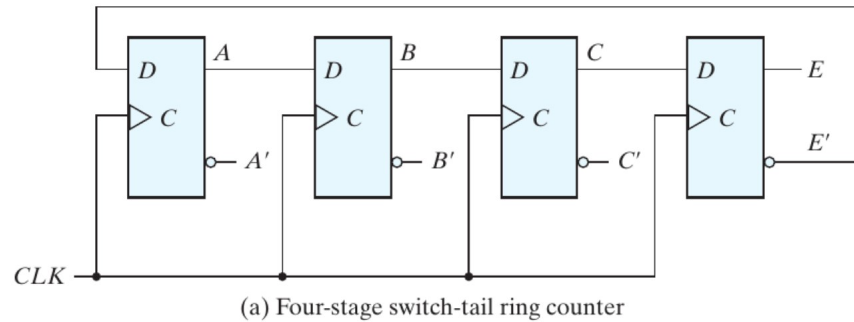


A 2-bit counter steps 00, 01, 10, 11; the decoder turns each state into one of $T_0..T_3$. Same four timing signals.

For 16 timing signals	Flip-flops	Gates
Ring counter	16	none
Counter + decoder	4	16 four-input ANDs

PART 2 • SHIFT-REGISTER COUNTERS

Johnson counter: feed back the complement



Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding

One change from the ring counter: E' , not E , goes back to the first D . Also called a switch-tail ring counter.

1s fill in from the left, then 0s do: **8 states from 4 flip-flops**. In general, $2k$ states from k .

Each state is decoded by a **2-input AND** of neighbouring bits.

PART 2 · UNUSED STATES

4 flip-flops have 16 states. The Johnson counter uses 8. What about the rest?

IF NOISE PUTS IT IN 0100 (A B C E)

0100 → 1010 → 1101 → 0110 → 1011 → 0101 → 0010 → 1001 → 0100 → ...

The 8 unused states form their own loop. The counter **never** returns to the valid sequence.

$$DC = (A + C) \cdot B$$

Mano's fix replaces $DC = B$. The valid sequence is unchanged, and every unused state now reaches it within a few clocks: **self-correcting**.

Three ways to make 8 timing signals

Method	Flip-flops	Decoding gates	Glitch-free outputs?
Ring counter	8	None	Yes
Johnson counter	4	8 two-input ANDs	Yes, one bit changes per clock
3-bit counter + decoder	3	8 three-input ANDs	Not always

The trade-off again: more flip-flops buy simpler, cleaner decoding. Fewer flip-flops cost wider gates.

SUMMARY

Every counter was a D flip-flop plus logic choosing its next value

XOR in front of D

Binary counting: $D = Q \oplus$ (when to flip).

Mux in front of D

Load or count. Detect the last state, load the first: mod-N.

Neighbour into D

Shift-register counters: ring and Johnson, for timing signals.

COMING NEXT

Sequential multiplier: a counter tells the datapath when to stop.

T and JK flip-flops, state machines: the toggle packaged as one element, and a method for designing any sequence.