

LECTURE 7 · FINITE STATE MACHINES

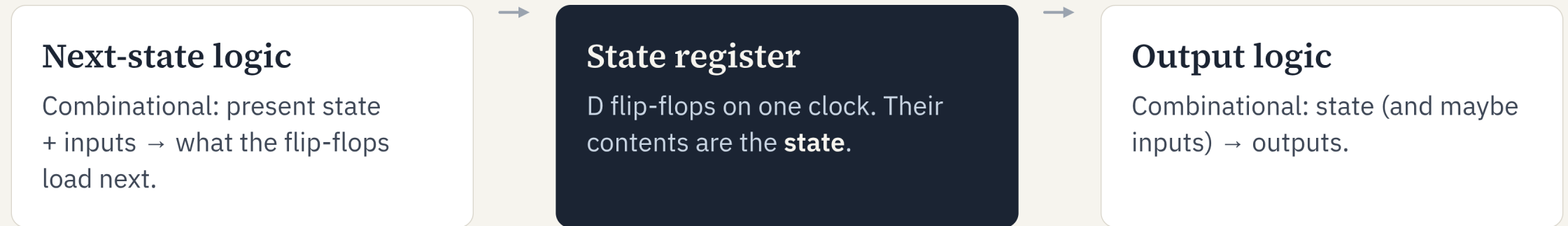
State Machines: Analysis and Design

Reading what a clocked circuit does, then building one from a word description.

Based on Mano & Ciletti, Digital Design (6th ed.), Sections 5.5 (analysis with D flip-flops) and 5.8 (design with D flip-flops)

THE BIG PICTURE

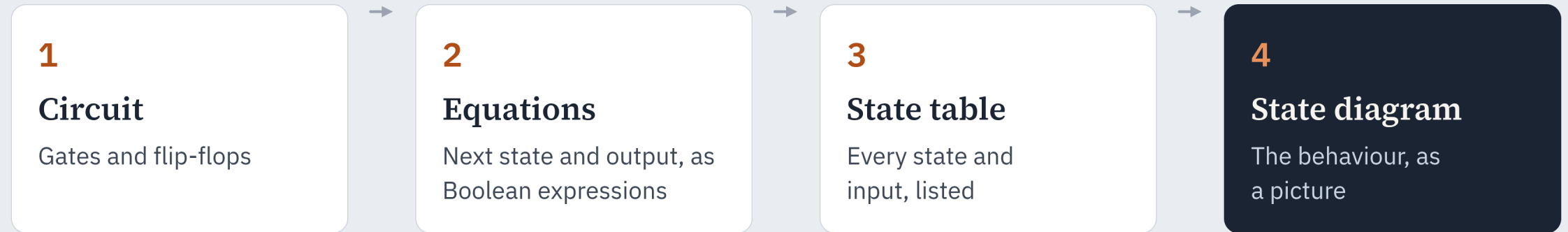
Every clocked circuit we have built has the same three parts



Circuit we built	Its state	Its next-state logic
Counter	The count	XOR + AND chain (or count + 1)
Serial adder	The carry	The full adder's carry-out

ANALYSIS

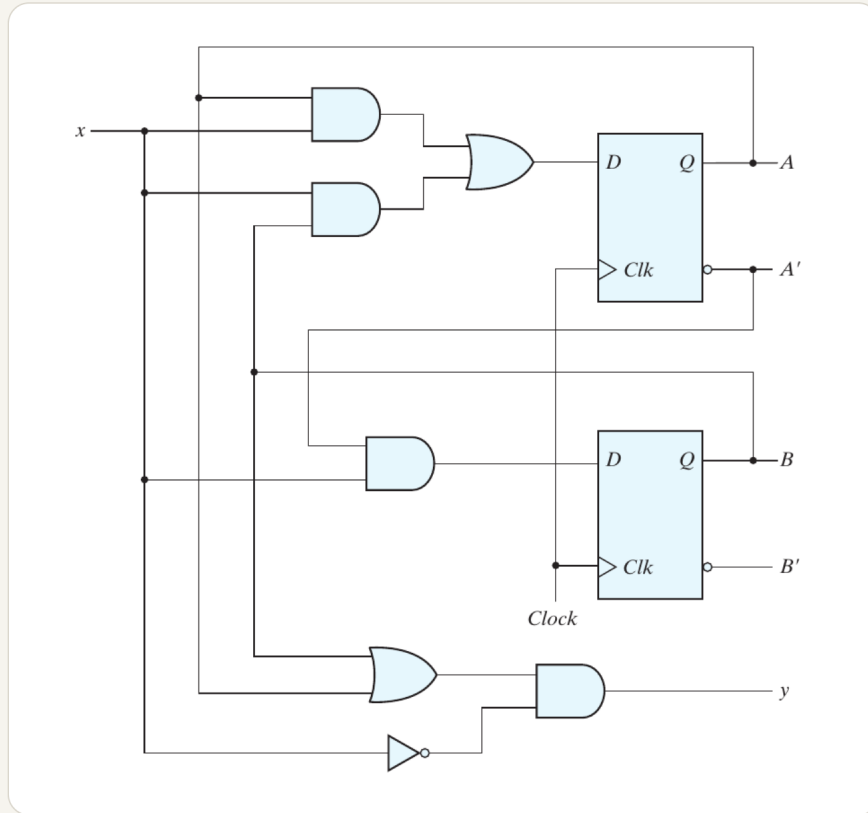
Analysis: given a circuit, find out what it does



Design is the same path in reverse: from a description of behaviour to a circuit.

STEP 1 • THE CIRCUIT

Two D flip-flops, one input x , one output y



Trace each D input back through the gates to A , B and x .

The two flip-flops hold the **present state AB**. With 2 flip-flops there are 4 possible states.

y is formed from the state and the input.

STEP 2 • EQUATIONS

Read the equations off the gates

FLIP-FLOP INPUT EQUATIONS

$$DA = Ax + Bx$$

$$DB = A'x$$

$$y = (A + B)x'$$

STATE EQUATIONS

$$A(t+1) = Ax + Bx$$

$$B(t+1) = A'x$$

For a D flip-flop, $Q(t+1) = D$, so the **state equation is the input equation**. The right-hand side uses present values: the values just before the clock edge.

STEP 3 • STATE TABLE

List every present state and input

Present AB	x	Next AB	Output y
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	1
0 1	1	1 1	0
1 0	0	0 0	1
1 0	1	1 0	0
1 1	0	0 0	1
1 1	1	1 0	0

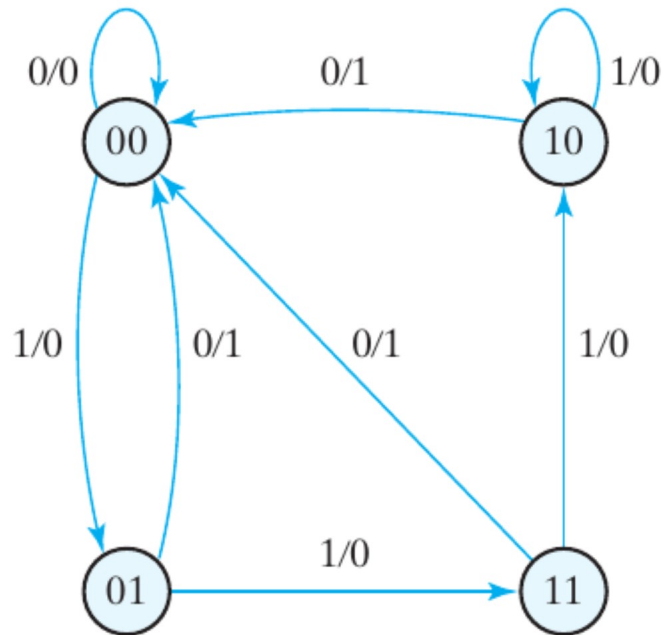
m flip-flops and n inputs give **2 $m+n$** rows: here $2^3 = 8$.

Fill **Next AB** from the state equations, and **y** from the output equation, exactly like a truth table.

Next state happens *after* the edge; y is the output *now*.

STEP 4 • STATE DIAGRAM

Circles are states, arrows are transitions



Each arrow is one row of the state table: from the present state, for one input value, to the next state.

Label **x/y**: input / output. 00 → 01 marked **1/0** means "in 00 with x = 1, y is 0, and the next edge goes to 01".

The output on an arrow belongs to the present state, before the edge.

WHAT DOES IT DO?

Run an input stream through the diagram

Clock cycle	1	2	3	4	5	6	7	8
State AB	00	01	11	10	00	01	00	00
Input x	1	1	1	0	1	0	0	1
Output y	0	0	0	1	0	1	0	0

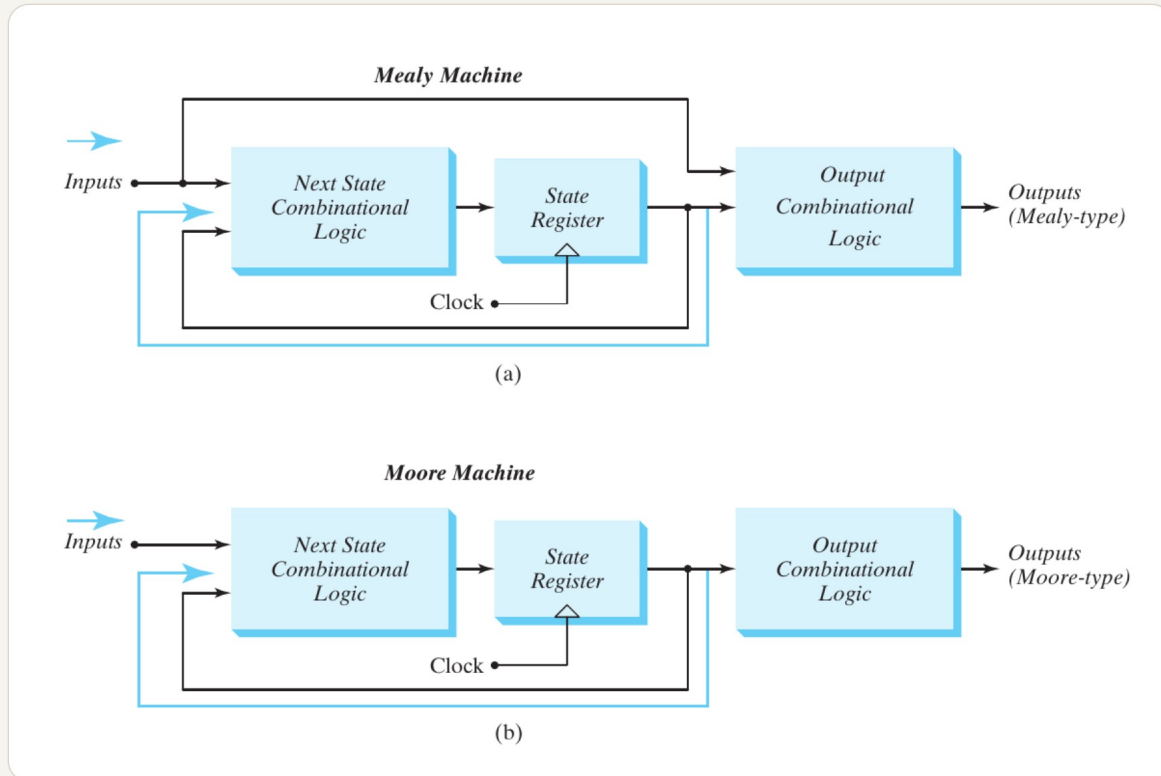
y = 1 exactly when a **0** arrives after one or more **1**s.

The 0 in cycle 7 gives y = 0: no 1 came just before it.

Mano calls this circuit a **zero detector**.

MEALY AND MOORE

Two kinds of finite state machine



Mealy

Output = $f(\text{state}, \text{inputs})$. Note the extra wire from the inputs to the output logic.

Moore

Output = $f(\text{state})$ only.

The zero detector is Mealy: y depends on x .

MEALY AND MOORE

How they differ in time

	Mealy	Moore
Output changes	Whenever the input changes	Only just after a clock edge
Reacts to an input	In the same cycle	One cycle later
Glitches	Possible, if the input is noisy mid-cycle	None from the input
States needed	Often fewer	Often more
Read the output	Just before the active clock edge	Any time during the cycle

Rule of thumb: when an output leaves your module or controls something important, prefer Moore or register the Mealy output.

THE PROCEDURE

Seven steps from words to gates

1	Word description → state diagram	The creative step: decide what each state remembers.
2	Reduce the number of states	Optional; its own lecture.
3	Assign binary codes to states	Binary, Gray or one-hot.
4	Write the binary-coded state table	Mechanical.
5	Choose the flip-flop type	D throughout this lecture.
6	Derive input and output equations	K-maps, with unused states as don't-cares.
7	Draw the logic diagram	Gates and flip-flops.

Steps 1 and 3 need thought. Steps 4 to 7 are bookkeeping, and in Verilog the synthesis tool does them for you.

STEP 1 • EXAMPLE

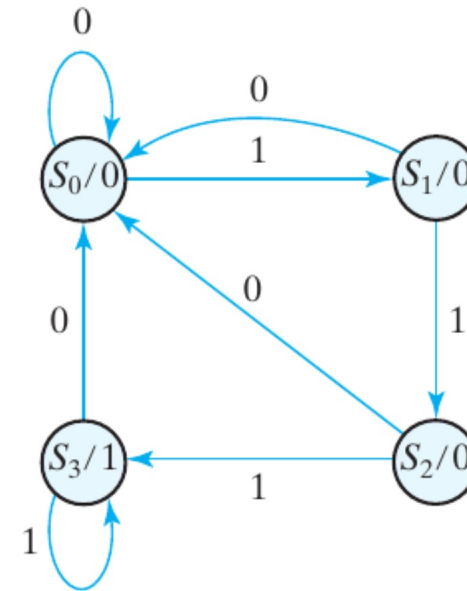
A sequence detector: three or more 1s

"Design a circuit that detects three or more consecutive 1s in a string of bits arriving on input **x**. Output **y** = 1 while the condition holds."

One bit arrives per clock cycle.

Moore machine: **y** depends on the state only.

Example: $x = 0\ 1\ 1\ 1\ 1\ 0 \rightarrow y$ becomes 1 after the third 1 and stays 1 after the fourth.



STEP 1 • THINKING

Each state answers: what do I remember?

S0

No 1s at the end of the input so far (or just reset).

S1

Exactly one 1 at the end.

S2

Exactly two 1s at the end.

S3

Three or more 1s at the end. Output 1.

A state is **everything the machine must remember about the past** to decide what to do next.

Here, the length of the current run of 1s, counting up to three. Anything longer looks the same as three.

STEPS 3 AND 4

The binary-coded state table

A B	x	A B (next)	y
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	0
0 1	1	1 0	0
1 0	0	0 0	0
1 0	1	1 1	0
1 1	0	0 0	1
1 1	1	1 1	1

Step 3: binary assignment $S0 = 00$, $S1 = 01$, $S2 = 10$, $S3 = 11$.

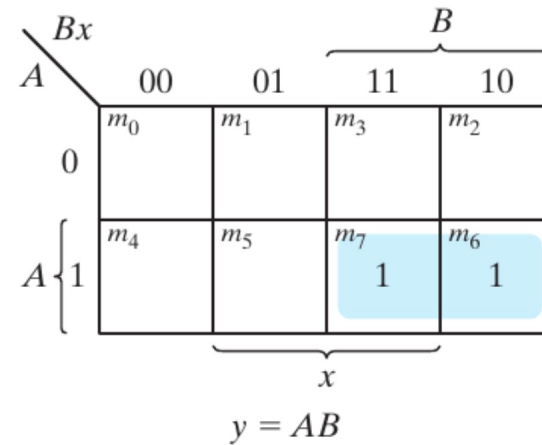
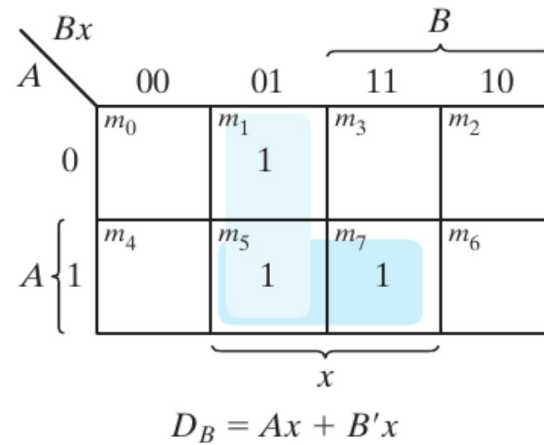
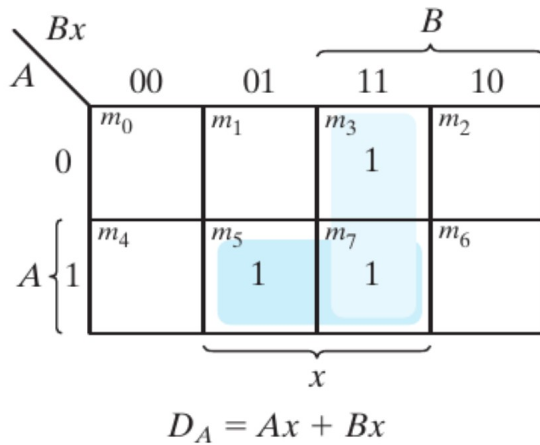
Step 4: one row per present state and input: $2^2 \times 2 = 8$ rows.

With D flip-flops, the next-state columns *are* the D inputs: $\{DA\} = A(t+1)$, $\{DB\} = B(t+1)$.

y is a function of A and B only: it repeats on both rows of a state.

STEP 6

Equations from Karnaugh maps



$$\mathbf{DA} = \mathbf{Ax} + \mathbf{Bx}$$

$$\mathbf{DB} = \mathbf{Ax} + \mathbf{B'x}$$

$$\mathbf{y} = \mathbf{AB}$$

Read each 1 from the next-state columns of the table.

Same maps, same rules as combinational logic.

CHECK

Trace the equations on a test input

Cycle	1	2	3	4	5	6	7	8	9	10
input x	0	1	1	1	1	0	1	1	1	0
state AB	00	00	01	10	11	11	00	01	10	11
output y = AB	0	0	0	0	1	1	0	0	0	1

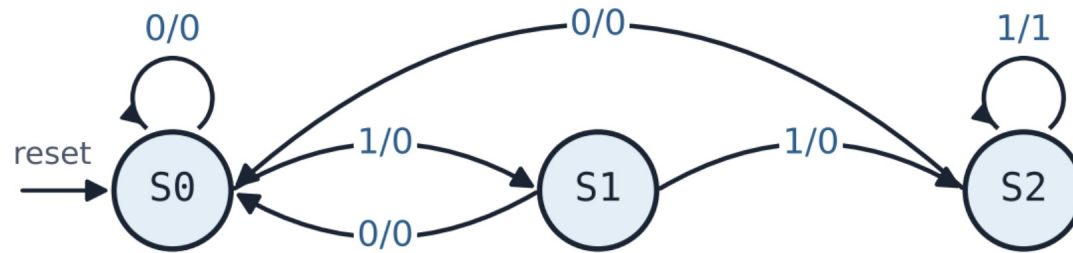
Next state of each column, from $DA = Ax + Bx$ and $DB = Ax + B'x$, is the state in the following column.

$y = 1$ in cycles 5 and 6 (after the third and fourth 1s) and in cycle 10. Correct.

Hand traces like this become the expected values in a self-checking testbench.

MEALY OR MOORE

The same detector as a Mealy machine



Three states, not four.

The output is on the arrow: 1/1 means "input 1, output 1".

$y = 1$ **during** the cycle of the third 1, one cycle earlier than Moore.

Mealy: fewer states, faster response, but the output can glitch with x . Moore: one extra state, output stable all cycle.

EXAMPLE 2 · STEP 1

A Mealy detector for 1 0 1

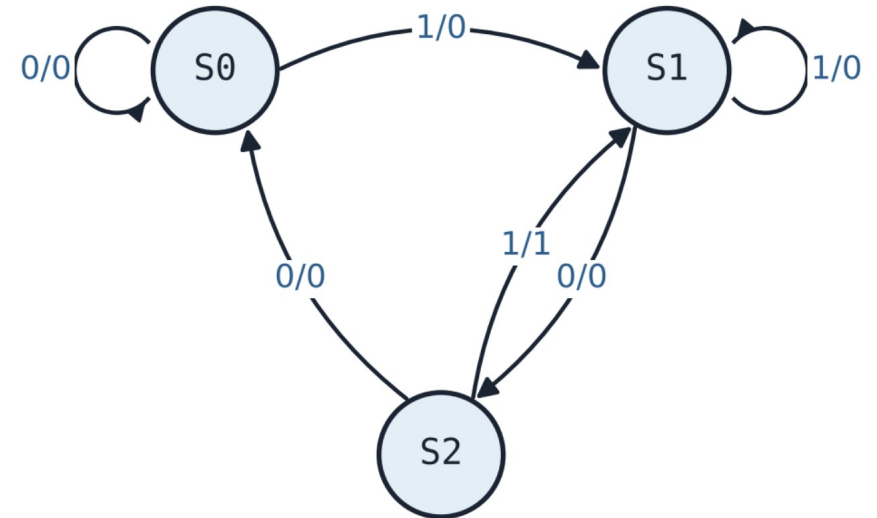
Output $z = 1$ in the cycle the last three inputs are **1 0 1**.
Overlap allowed: 10101 gives two detections.

S0: nothing useful seen.

S1: last input was 1.

S2: last two inputs were 1 0.

From S2, a 1 completes 101 *and* is the start of the next one, so go to S1.



STEPS 3 AND 4

State table with an unused state

A B	x	A B (next)	z
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	1 0	0
0 1	1	0 1	0
1 0	0	0 0	0
1 0	1	0 1	1
1 1	0	X X	X
1 1	1	X X	X

Three states need two flip-flops: S0 = 00, S1 = 01, S2 = 10.

Code **11 is unused**. The machine never goes there in normal operation.

So its rows are **don't-cares** (X): the K-maps may treat them as 0 or 1.

Mealy: z can differ between the two rows of one state.

STEP 6

Don't-cares give very small equations

DA

A \ Bx	00	01	11	10
0	0	0	0	1
1	0	0	X	X

DB

A \ Bx	00	01	11	10
0	0	1	1	0
1	0	1	X	0

z

A \ Bx	00	01	11	10
0	0	0	0	0
1	0	1	X	X

$$\mathbf{DA} = Bx' \quad \mathbf{DB} = x \quad \mathbf{z} = Ax$$

The don't-cares made every equation a single product term. $\mathbf{DB} = x$: B simply remembers the last input.

CHECK

What happens in the unused state?

A B	x	DA = Bx'	DB = x	next	z = Ax
1 1	0	1	0	1 0 (S2)	0
1 1	1	0	1	0 1 (S1)	1

Good

From 11 the machine reaches a valid state in one clock. It is **self-correcting**.

Bad

In 11 with $x = 1$, $z = 1$: a **false detection**.

Fix if it matters

Use $z = AB'x$, one more literal. Or rely on reset never letting the machine reach 11.

UNUSED STATES

Three ways to handle unused states

1. **Don't-care, then check.** Smallest logic. Analyse every unused state afterwards.
2. **Send unused states to S0.** Write it into the table. Slightly more logic, no surprises.
3. **Always have a reset.** It puts the machine in a known state at start-up.

```
case (state)
  S0: next = x ? S1 : S0;
  S1: next = x ? S1 : S2;
  S2: next = x ? S1 : S0;
  default: next = S0;
endcase
```

In Verilog, a **default** branch is option 2. Make it a habit in every FSM.

SUMMARY

Analysis and design are the same path, two directions

Analysis

Circuit → equations → state table → state diagram.

Mealy or Moore

Does the output use the input? Moore outputs change only after edges.

Design

Words → states that remember → codes → table → equations → circuit.

Unused states

Check them, give a default, always reset.

Next: Lecture 8 writes these machines in Verilog, starting from the 101 detector.