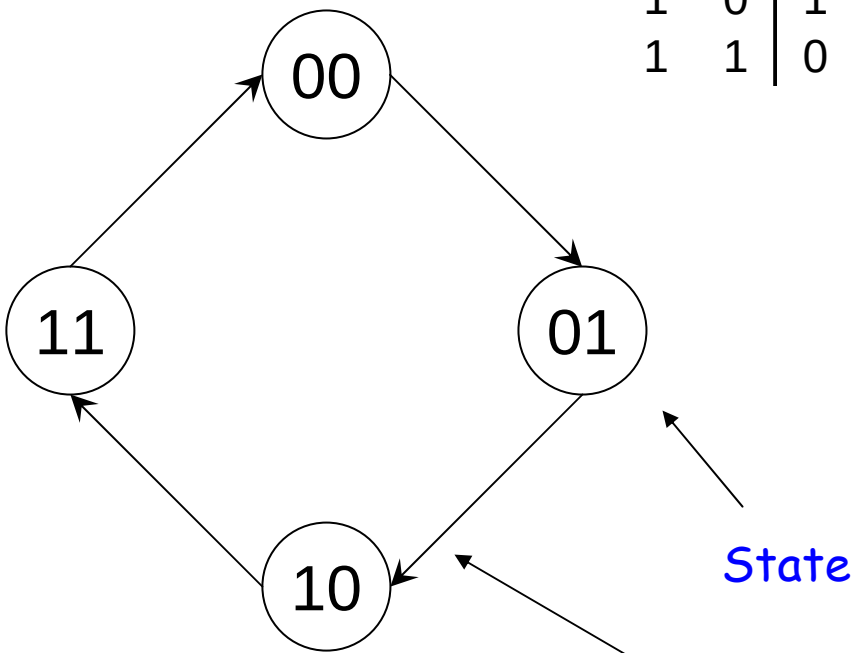
State Graphs
FSMs

Binary Counter State Graph

Q1	Q0	N1	N0
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



State graphs are graphical representations of TT's

They contain the same information: no more, no less

Design Procedure Using State Graphs

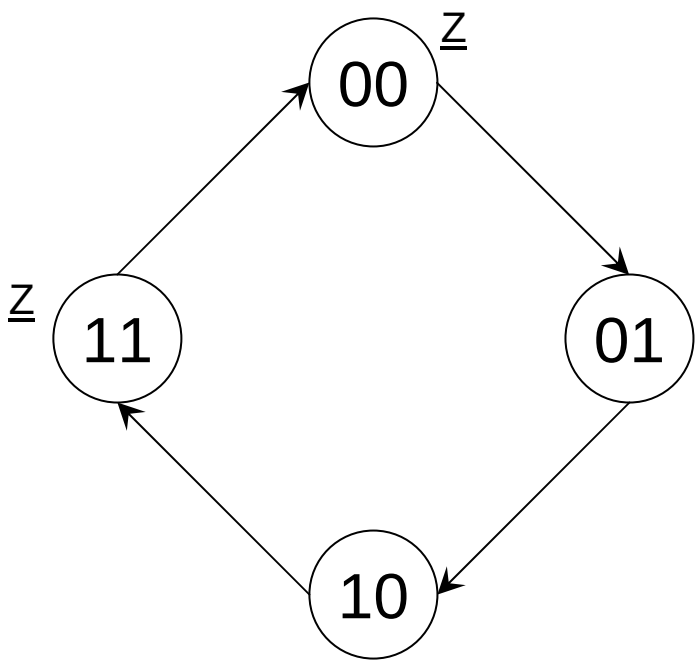
1. Draw the state graph

2. Create an equivalent transition table

3. If transition table contains input don't cares,
   - *unfold* it to a full transition table

4. Complete the design using KMaps, gates, FF's

State Graphs With Moore Outputs

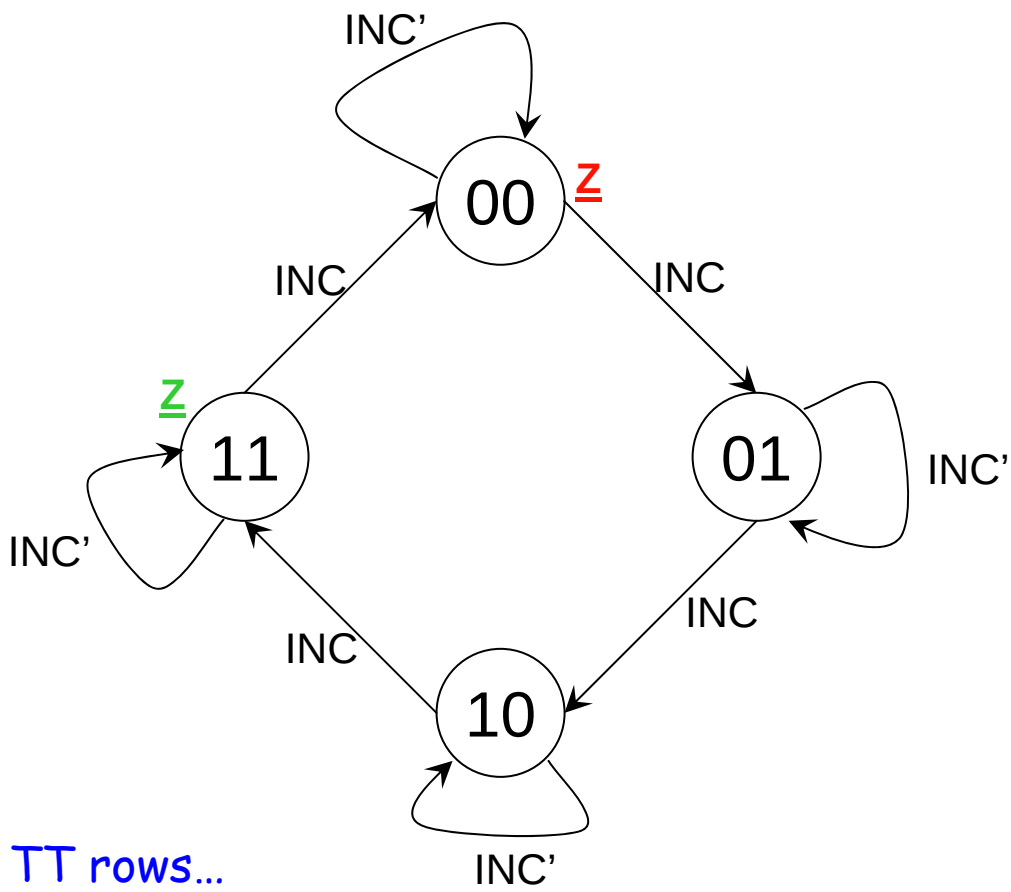


Q1	Q0	N1	N0	Z
0	0	0	1	1
0	1	1	0	0
1	0	1	1	0
1	1	0	0	1

Write the output next to the states it is asserted in…
Underline it to make it more clear

Another SG With A Moore Output

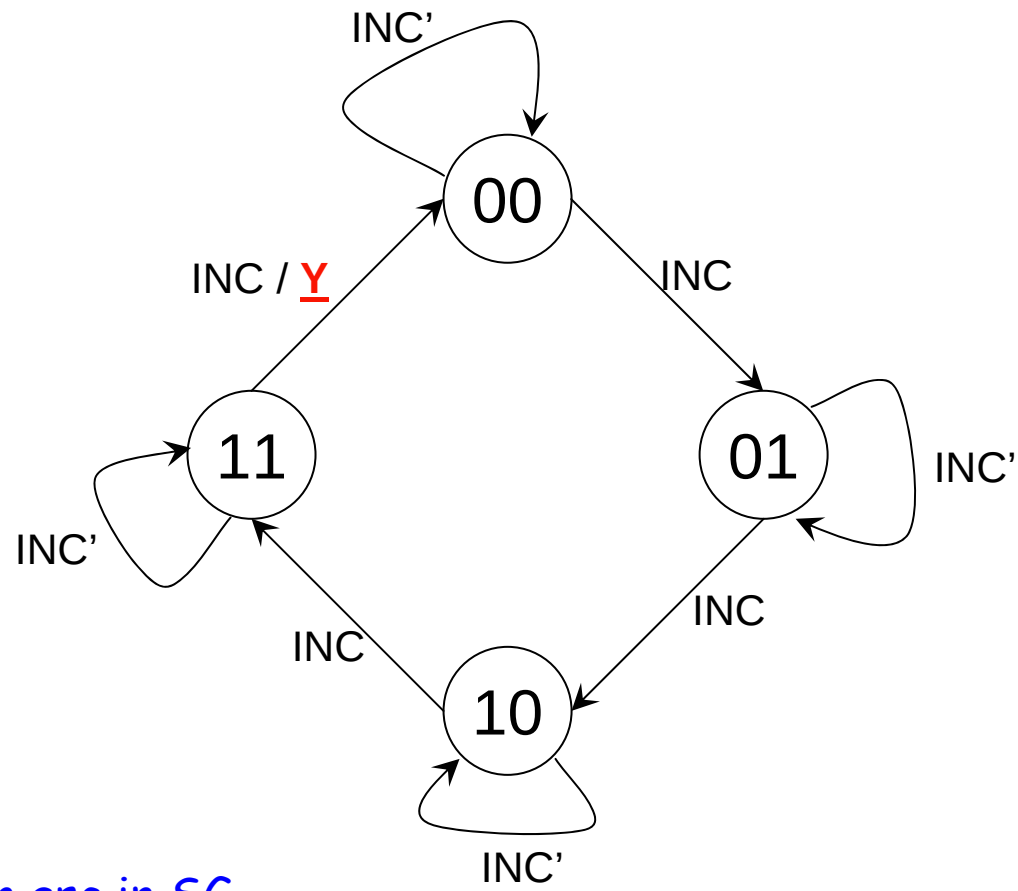
INC	Q1	Q0	N1	N0	Z
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	0	0	1



Moore output shows up in multiple TT rows…

State Graphs and Mealy Outputs

INC	Q1	Q0	N1	N0	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0
1	**1**	**1**	**0**	**0**	**1**



Mealy output is associated with an arc in SG

Properly Formed State Graphs

- A properly formed state graph is both:

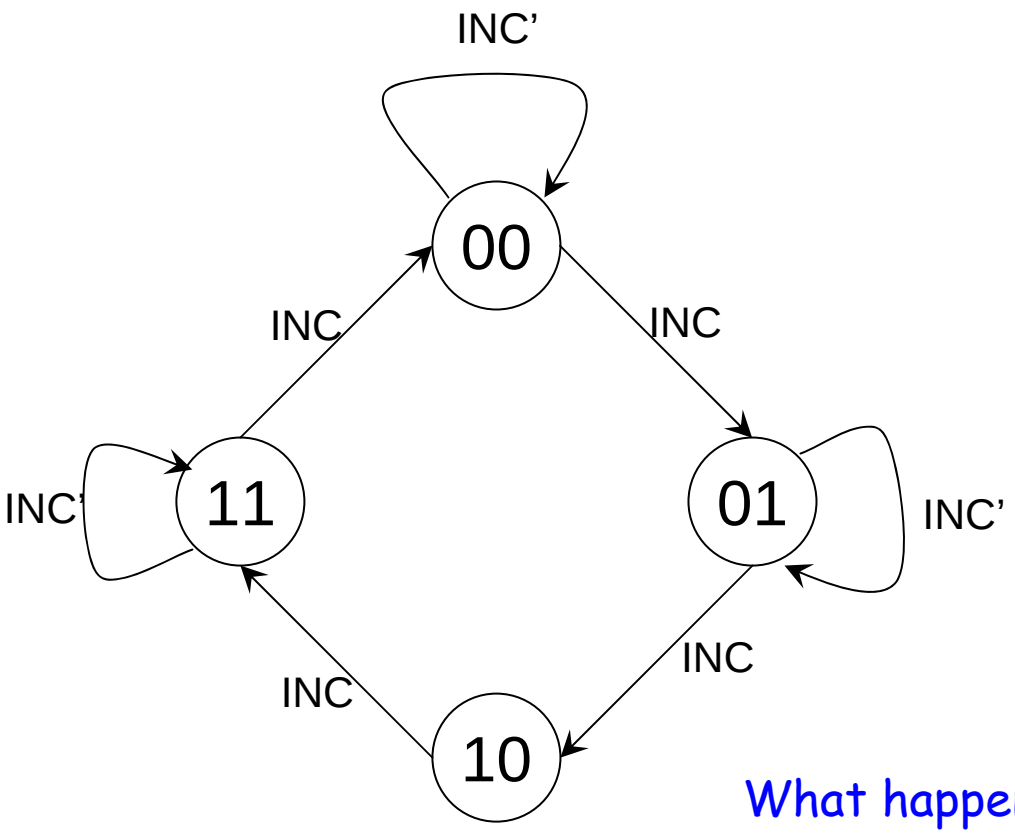
  Complete

  and

  Conflict-free

An Incomplete State Graph

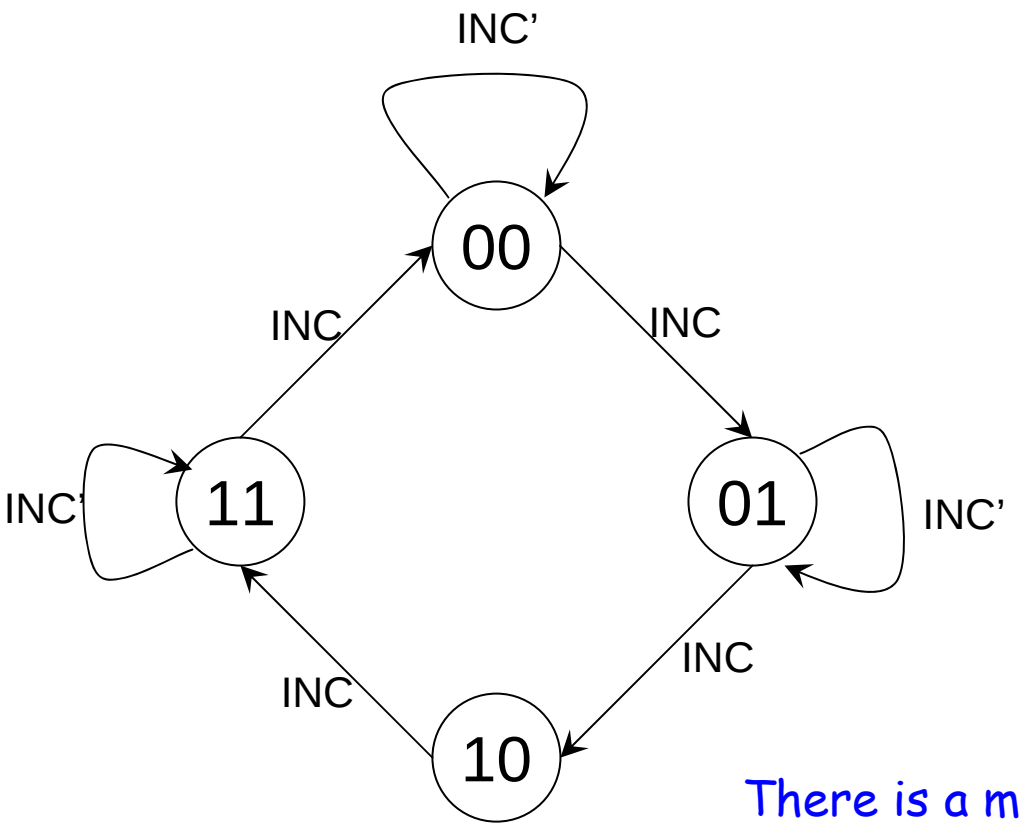
- This SG is incomplete. Can you see why?



What happens in state '10' when INC=0?

An Incomplete State Graph

- This SG is incomplete. Can you see why?



INC	Q1	Q0	N1	N0
0	0	0	0	0
0	0	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

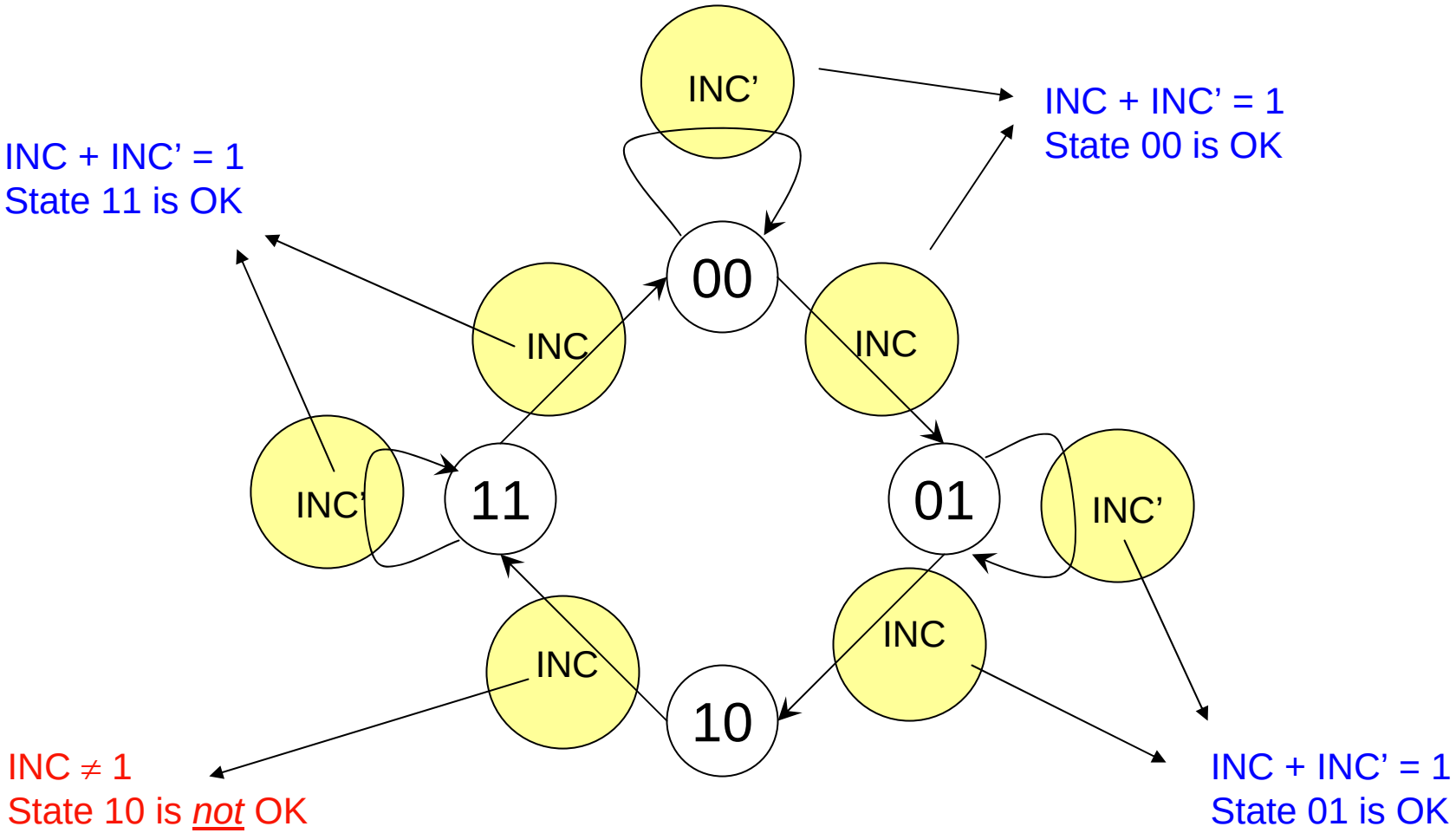
There is a missing row in the TT as well…

Complete State Graphs

- For a SG to be complete:
  - Paths leaving each state must cover all cases

- To check:
  - OR together conditions on all arcs leaving a given state
  - If result = '1', state is complete
  - If result ≠ '1', state is incomplete

Checking for Completeness



Alternate Check for Completeness

- Full transition table (no input don't cares) should have 2^n rows where:

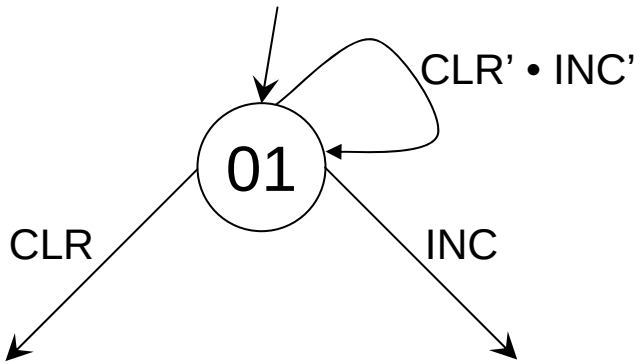
  n = (#inputs + #state variables)

INC	Q1	Q0	N1	N0
0	0	0	0	0
0	0	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

This only has 7 rows, something is missing…

Additional Completeness Considerations

- If some input combination will never occur
  - Don't need to enforce completeness

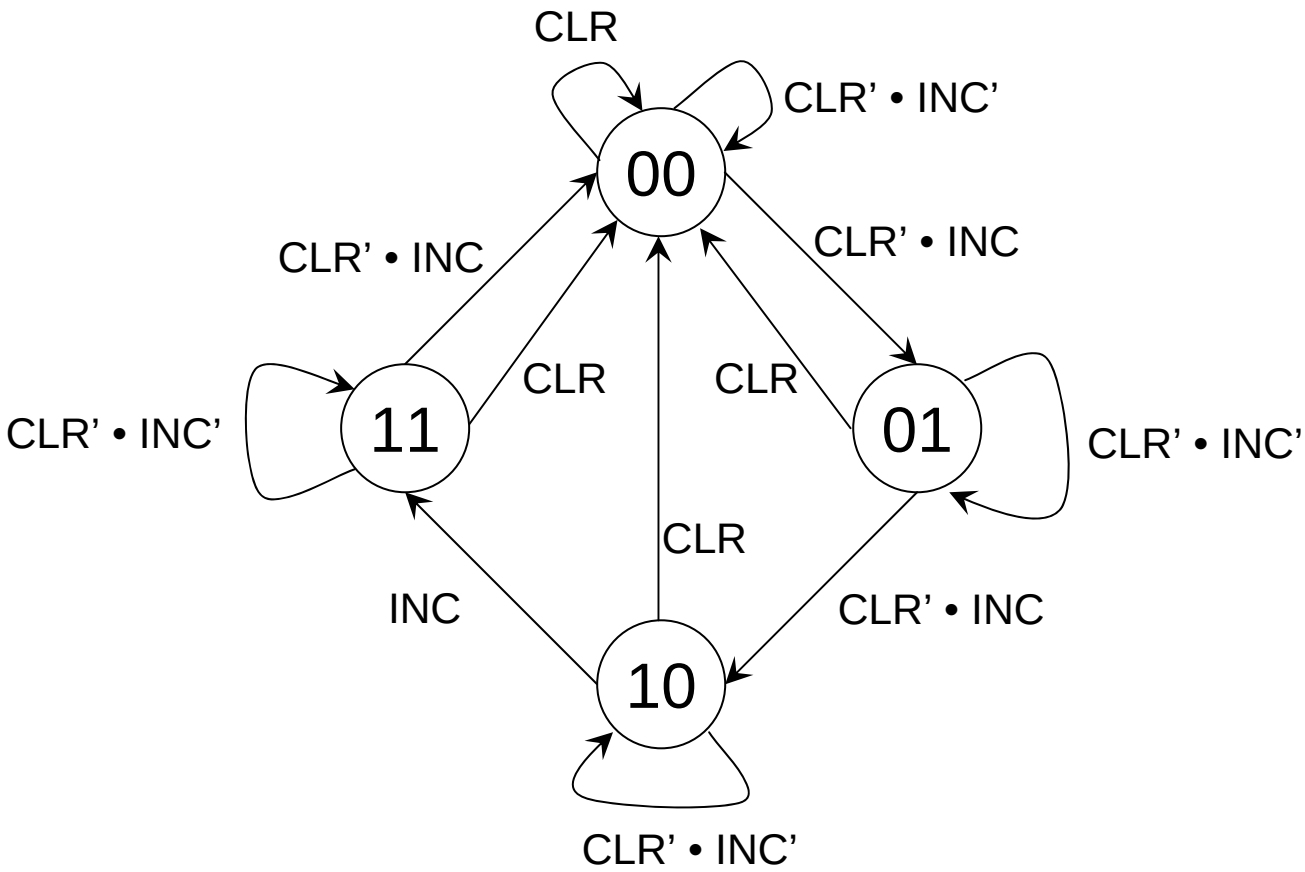


What about the case of CLR•INC?

If INC=CLR='1' will *never* occur,
this incomplete state is OK

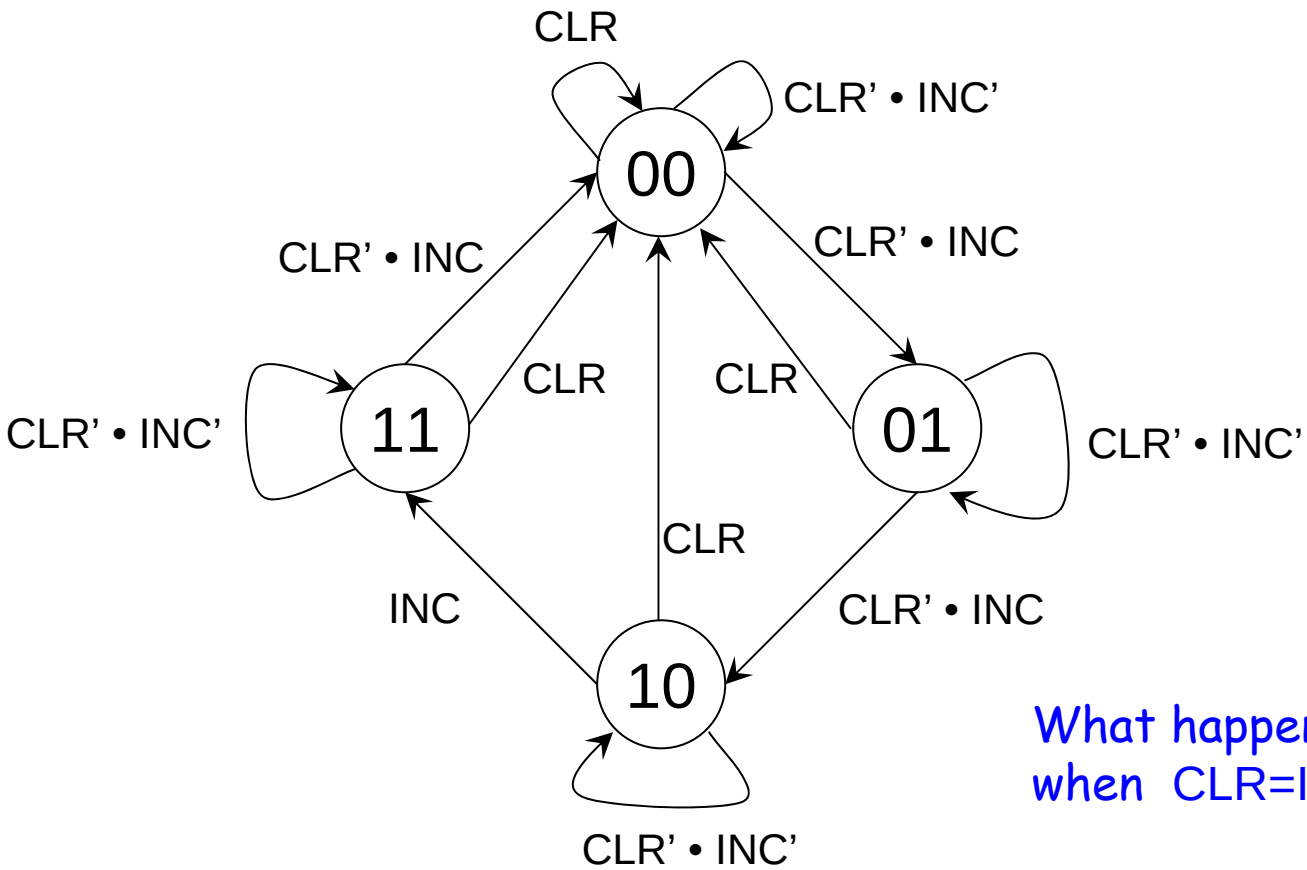
Conflicts in State Graphs

- This SG has a conflict, can you find it?



Conflicts in State Graphs

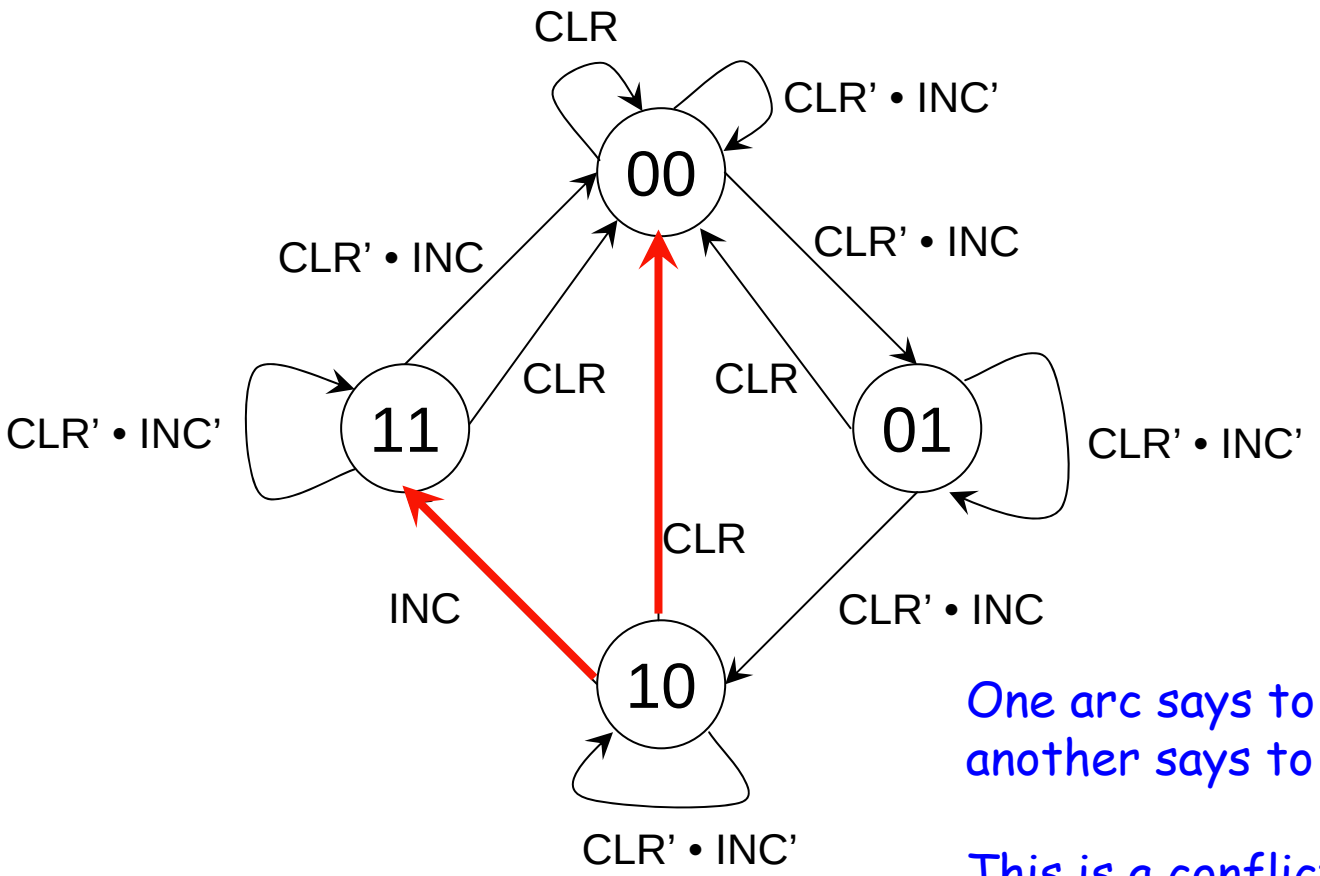
- This SG has a conflict, can you find it?



What happens in state '10' when CLR=INC='1'?

Conflicts in State Graphs

- This SG has a conflict, can you find it?



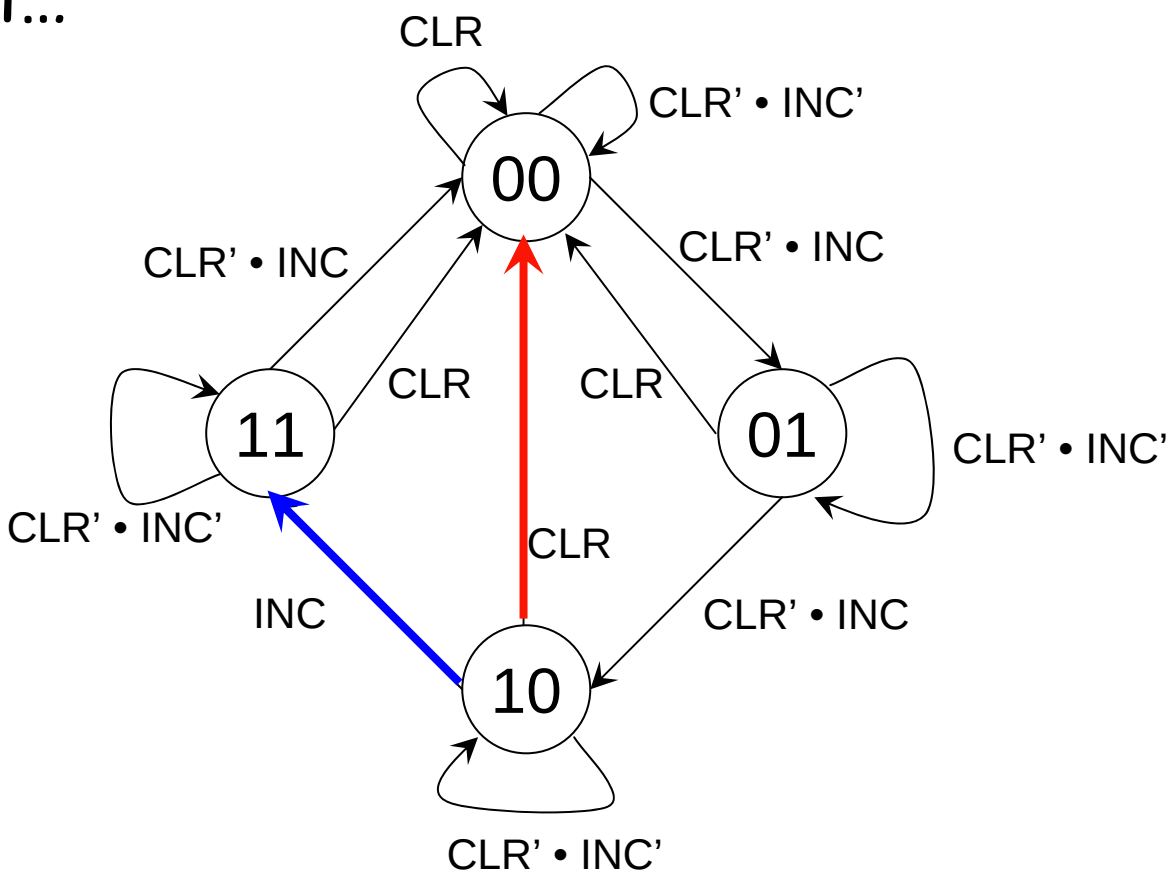
One arc says to go to state 00, another says to go to state 11

This is a conflict…

Conflicts in State Graphs

- The corresponding transition table has a problem as well…

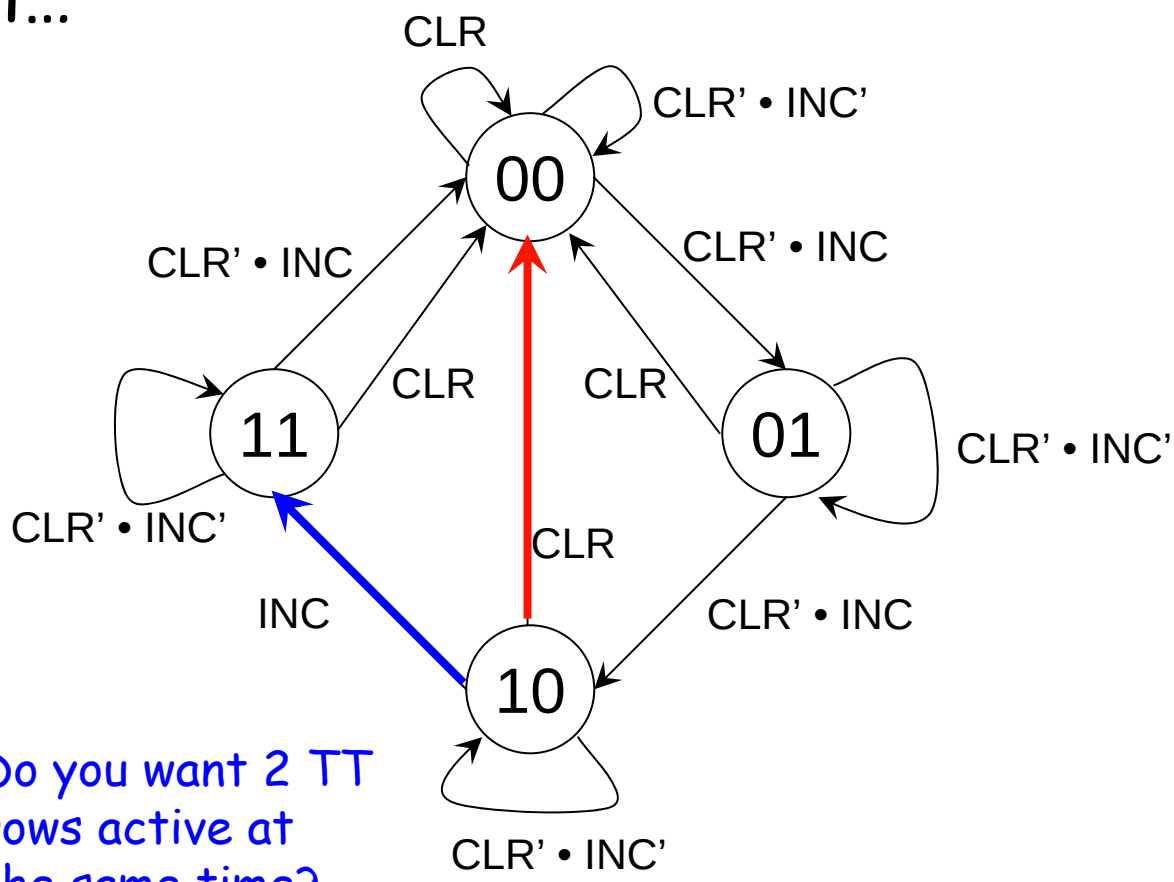
	CLR	INC	Q1	Q0	N1	N0
	0	0	0	0	0	0
	0	0	0	1	0	1
	0	0	1	0	1	0
	0	0	1	1	1	1
	0	1	0	0	0	1
	0	1	0	1	1	0
→	1	-	1	0	0	0
→	-	1	1	0	1	1
	0	1	1	1	0	0
	1	-	-	-	0	0



Conflicts in State Graphs

- The corresponding transition table has a problem as well…

	CLR	INC	Q1	Q0	N1	N0
	0	0	0	0	0	0
	0	0	0	1	0	1
	0	0	1	0	1	0
	0	0	1	1	1	1
	0	1	0	0	0	1
	0	1	0	1	1	0
→	1	-	1	0	0	0
→	-	1	1	0	1	1
	0	1	1	1	0	0
	1	-	-	-	0	0

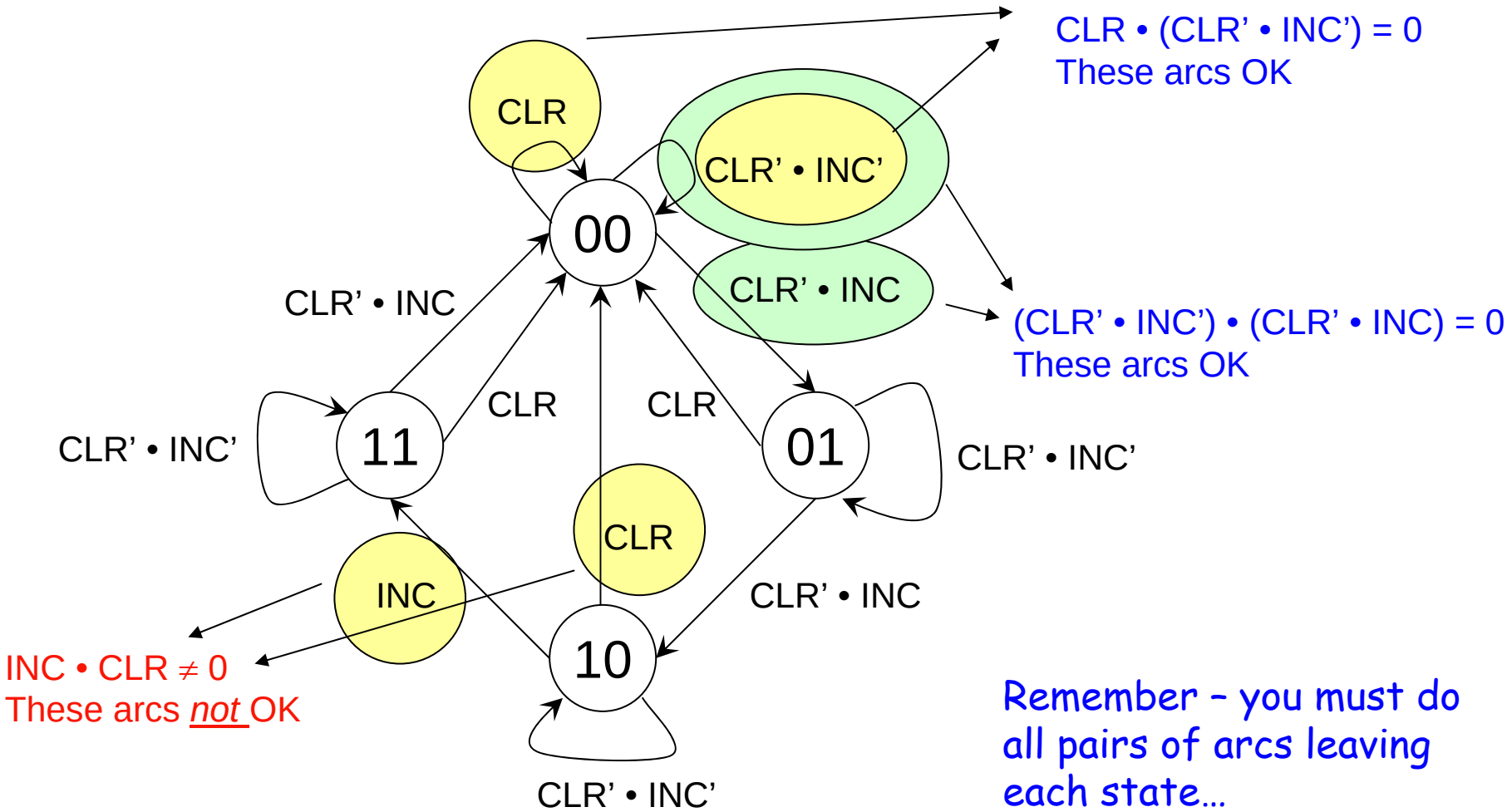


Do you want 2 TT rows active at the same time?

Conflict-Free State Graphs

- For a State Graph to not have conflicts:
  - Paths leaving each state must not conflict with one another
- To check:
  - For each pair of arcs leaving a given state, AND together their conditions
  - If result = '0', arcs have no conflict
  - If result ≠ '0', arcs have a conflict

Checking for Conflicts



Remember – you must do all pairs of arcs leaving each state…

Alternate Check for Conflicts

- Full transition table (no input don't cares) should have 2n rows where:

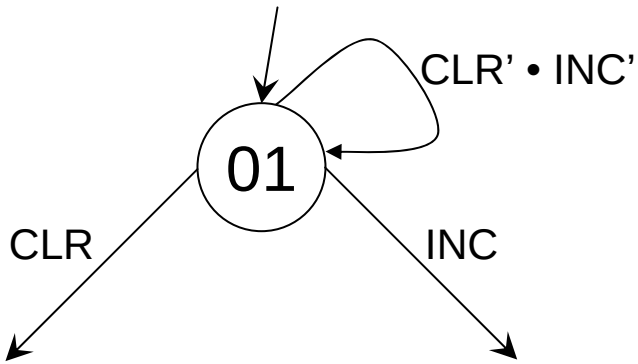
  n = (#inputs + #state variables)

CLR	INC	Q1	Q0	N1	N0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	0
1	**0**	**1**	**0**	**0**	**0**
1	**1**	**1**	**0**	**0**	**0**
0	**1**	**1**	**0**	**1**	**1**
1	**1**	**1**	**0**	**1**	**1**
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

This has 18 rows, something is wrong…

Additional Conflict Considerations

- If some input combination will never occur
  - Don't need to worry about conflicts

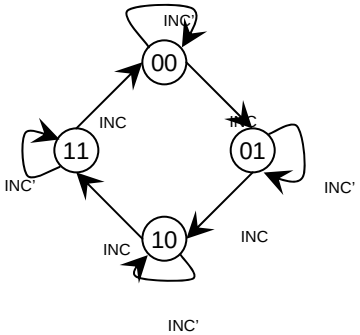


If INC=CLR='1' will *never* occur, the conflict shown here is OK.
But, you will have to make a decision on how to write the TT

Summary - Properly Formed State Graphs

- A properly formed state graph is both
  - Complete
  - Conflict-free

- Perform tests to ensure you have covered all the cases once and only once



Summary – SG's vs. TT's

INC	Q1	Q0	N1	N0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

- A state graph is simply a graphical way of writing a Transition Table
  - No additional information
- You should be adept at converting between them
- Design is always done from TT to KMaps to gates/FF's

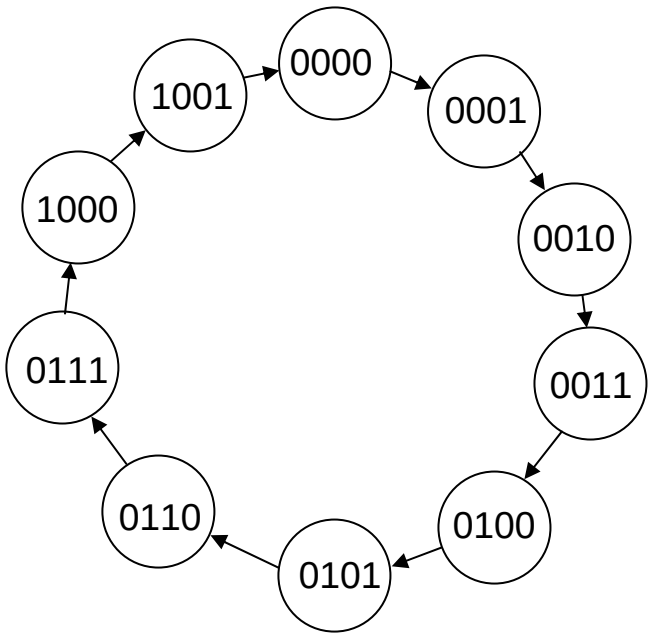
Finite State Machines

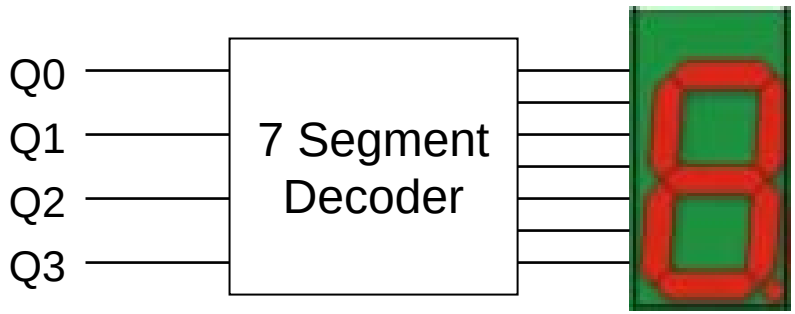
State Machine Concepts

- State, current state, next state, state registers
- IFL, OFL, Moore outputs, Mealy outputs
- Transition tables
  - With output don't cares (X's)
  - With input don't cares (-'s)
- State graphs
  - And their correspondence to TT's

Counters as State Machines

- A counter ***is*** a state machine
  - Where the state encodings are significant



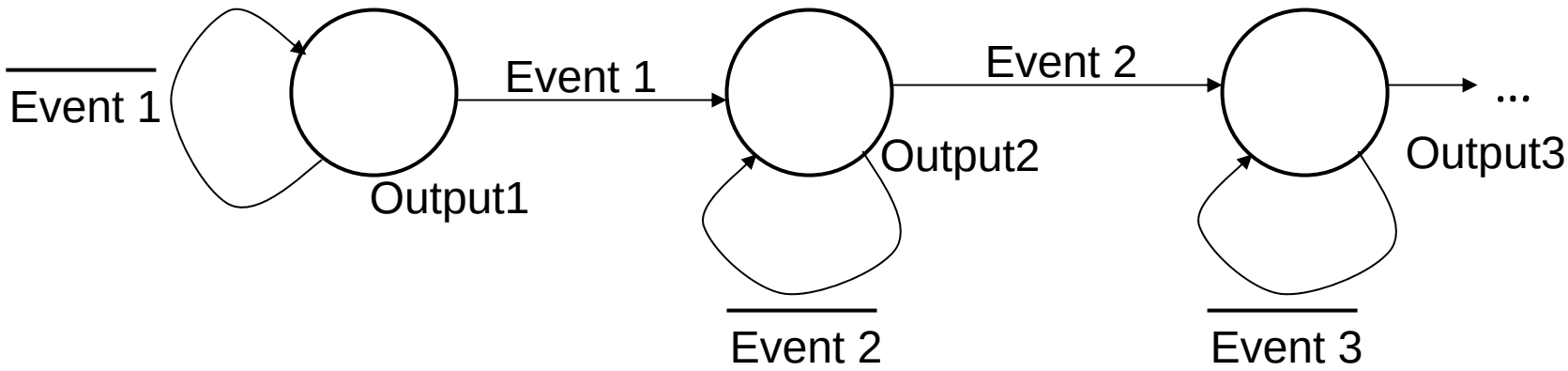


State Machines

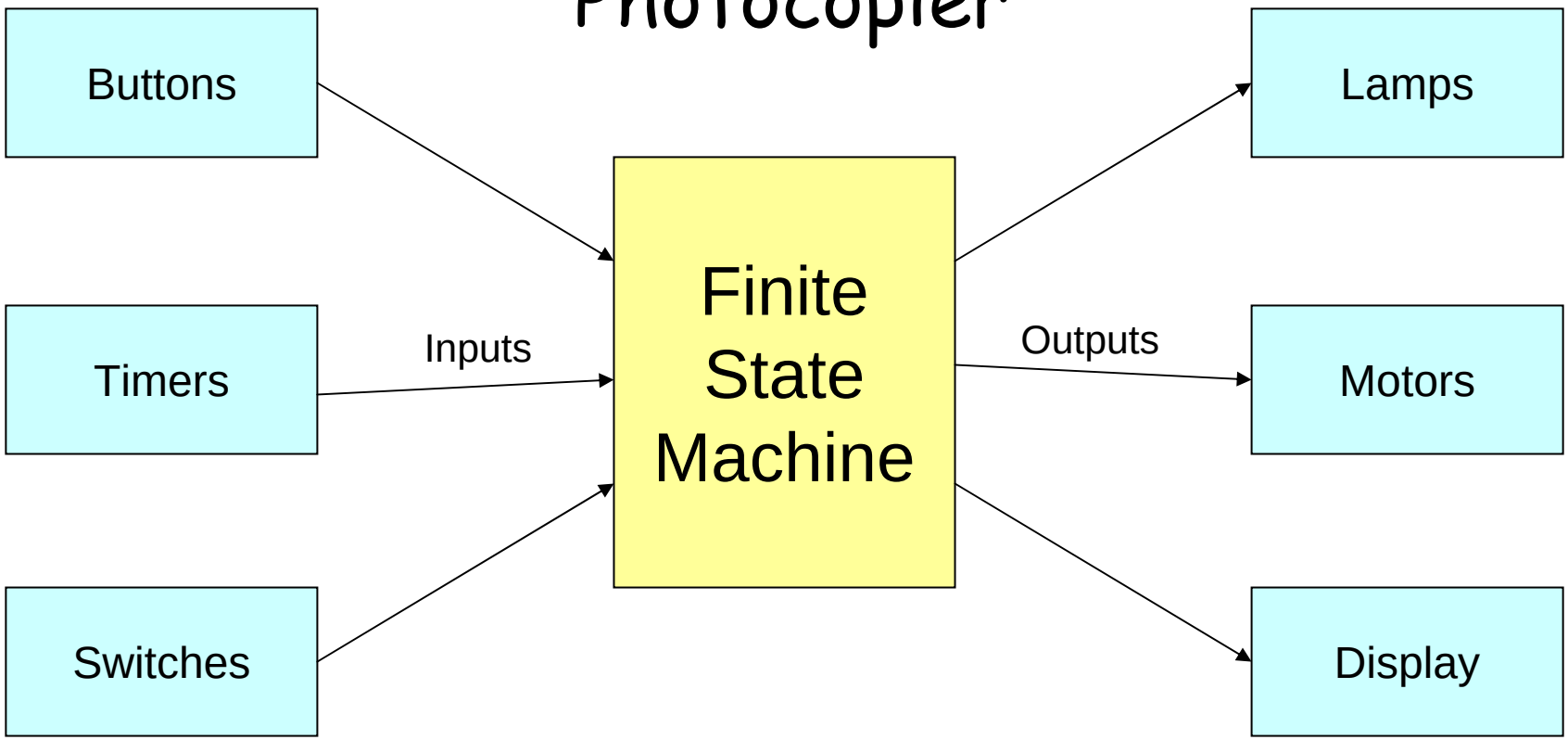
- A state machine is a *sequential circuit* which progresses through a series of states in reponse to inputs
  - The output values are usually significant
  - The state encodings are usually not significant
    - Unlike with counters

State Encodings

- In this machine, the state encodings don't matter
- The output values do…

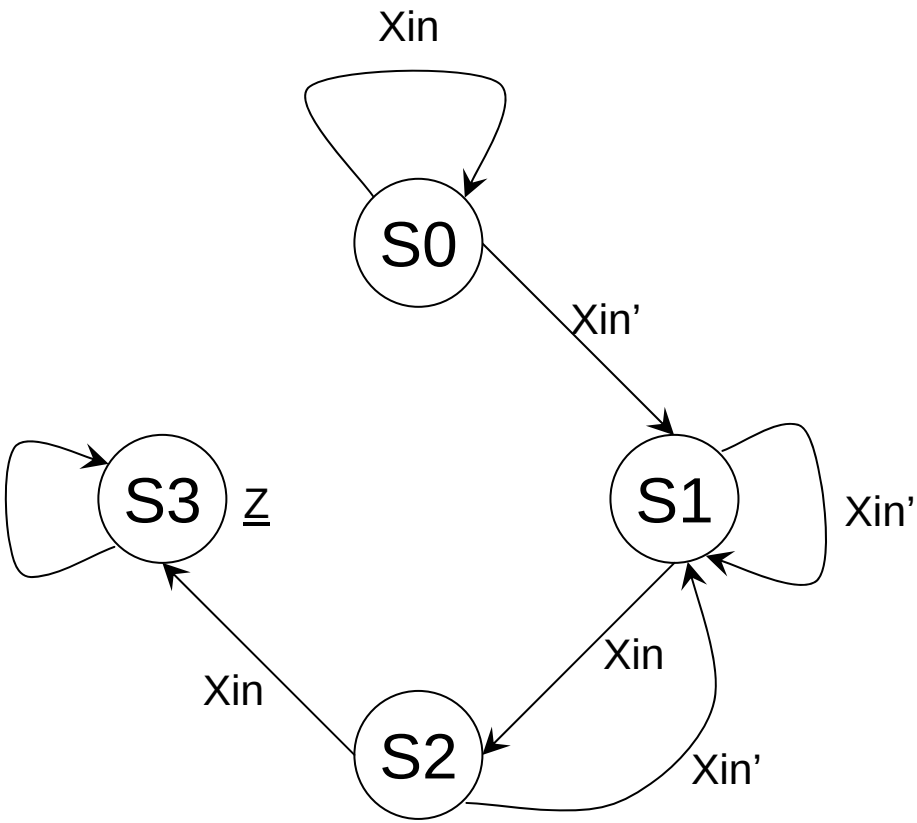


A State Machine Controller for a Photocopier



1) FSM receives inputs from copier
2) FSM generates control outputs in response
3) States help it remember where it is in the copy process…

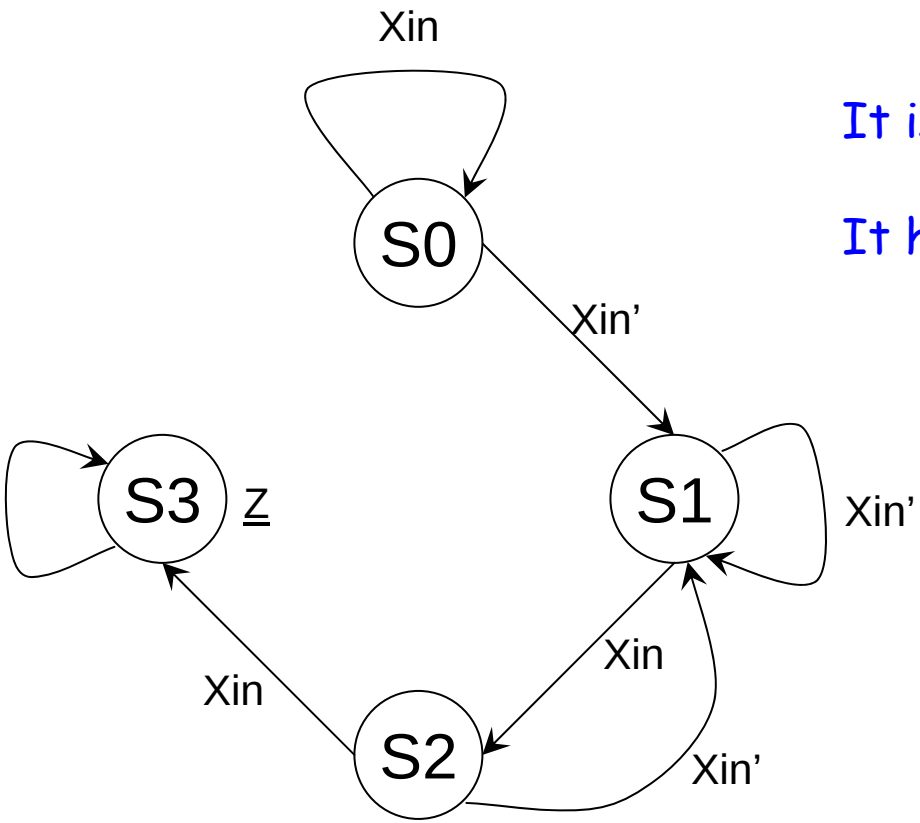
A Sequence Detector FSM



Because the encodings don't matter, we will use symbolic state names

What does this machine do?

A Sequence Detector FSM

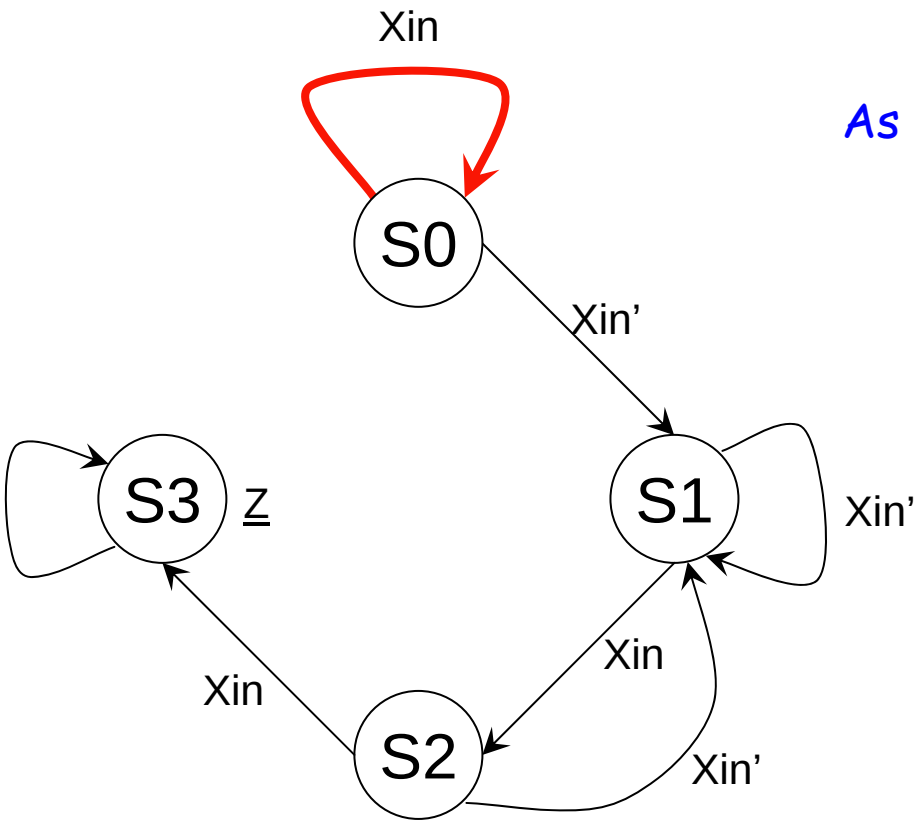


It is a sequence detector.

It has 1 input: 'Xin' and one output: 'Z'

It detects the sequence 0..1..1 on the input.
When detected, output 'Z' is asserted.

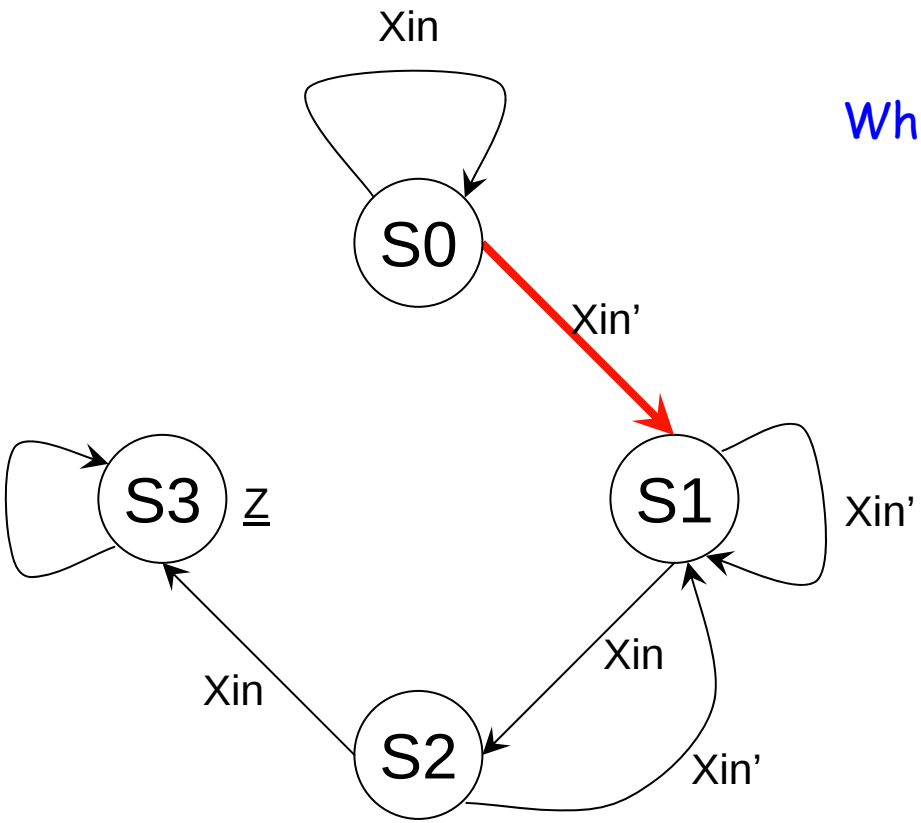
A Sequence Detector FSM



As long as Xin=1, we stay in state S0

1..1..1..1..1..

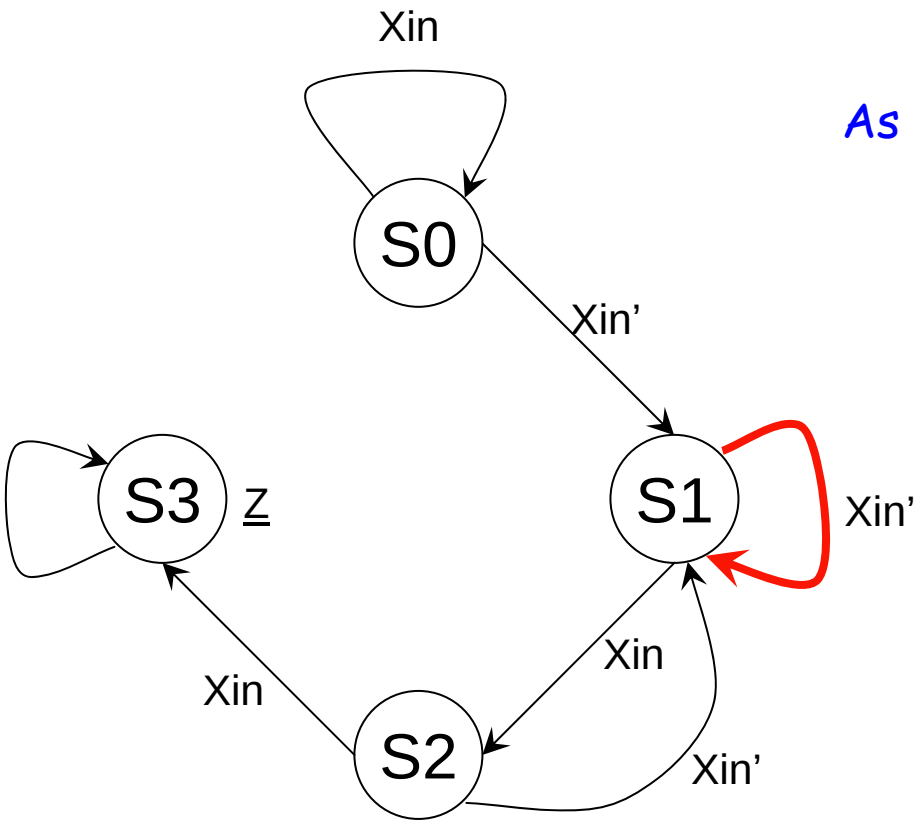
A Sequence Detector FSM



When Xin=0, we go to state S1

1..1..1..1..1..0..

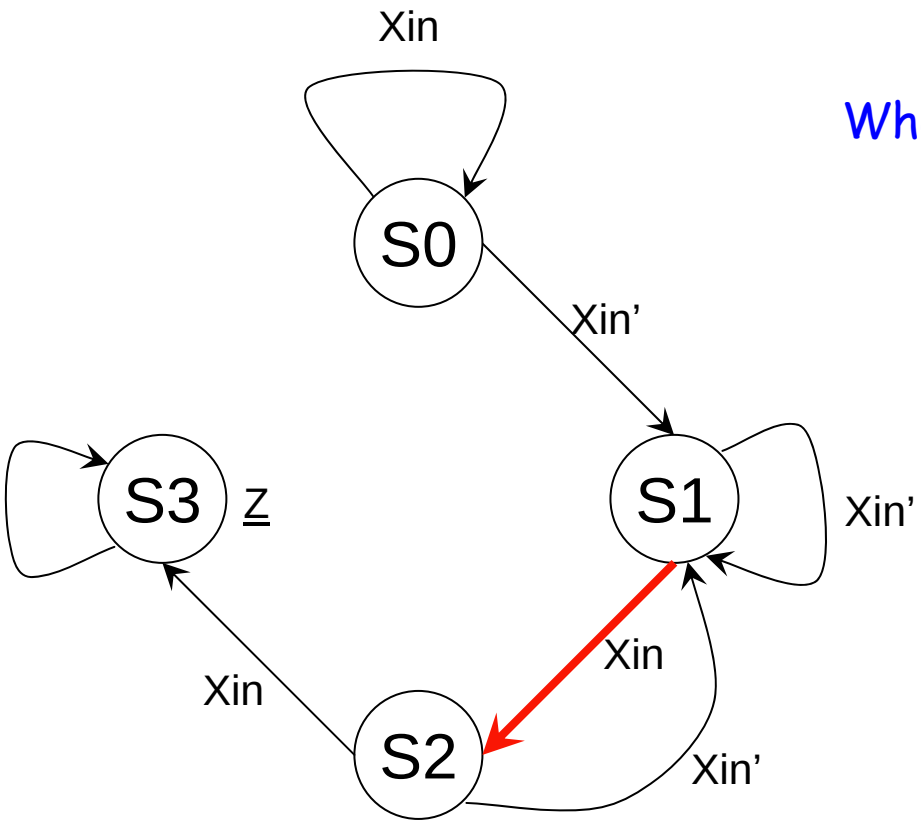
A Sequence Detector FSM



As long as Xin=0, we stay in state S1

1..1..1..1..1..0..0..0..0..

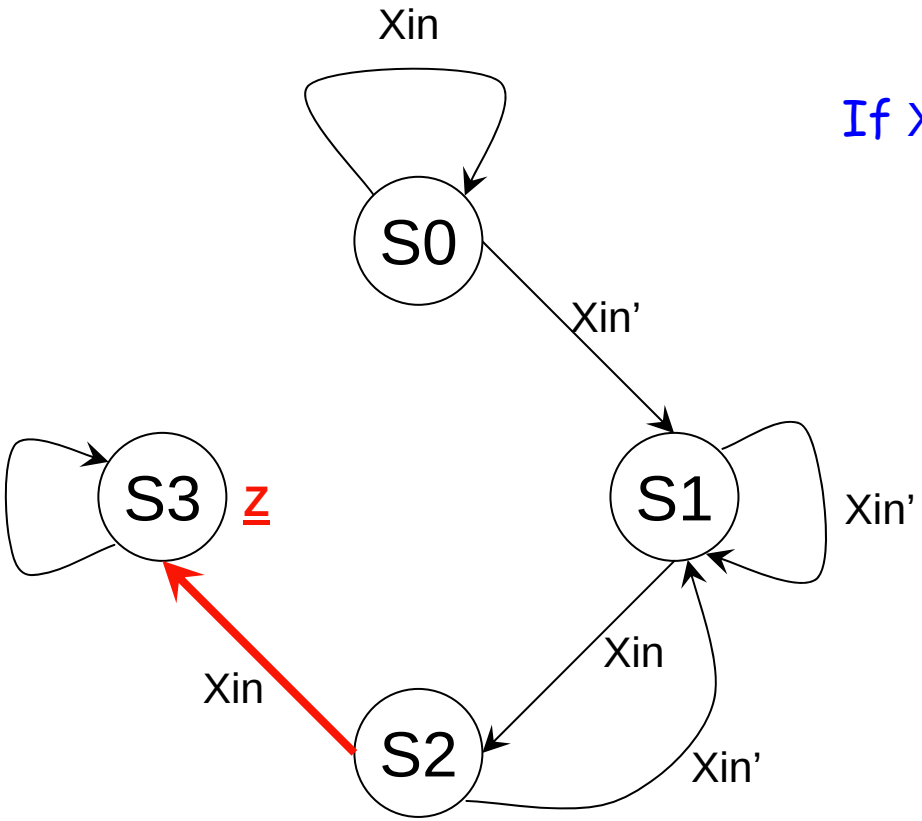
A Sequence Detector FSM



When Xin=1, we go to state S2

1..1..1..1..1..0..0..0..0..1..

A Sequence Detector FSM

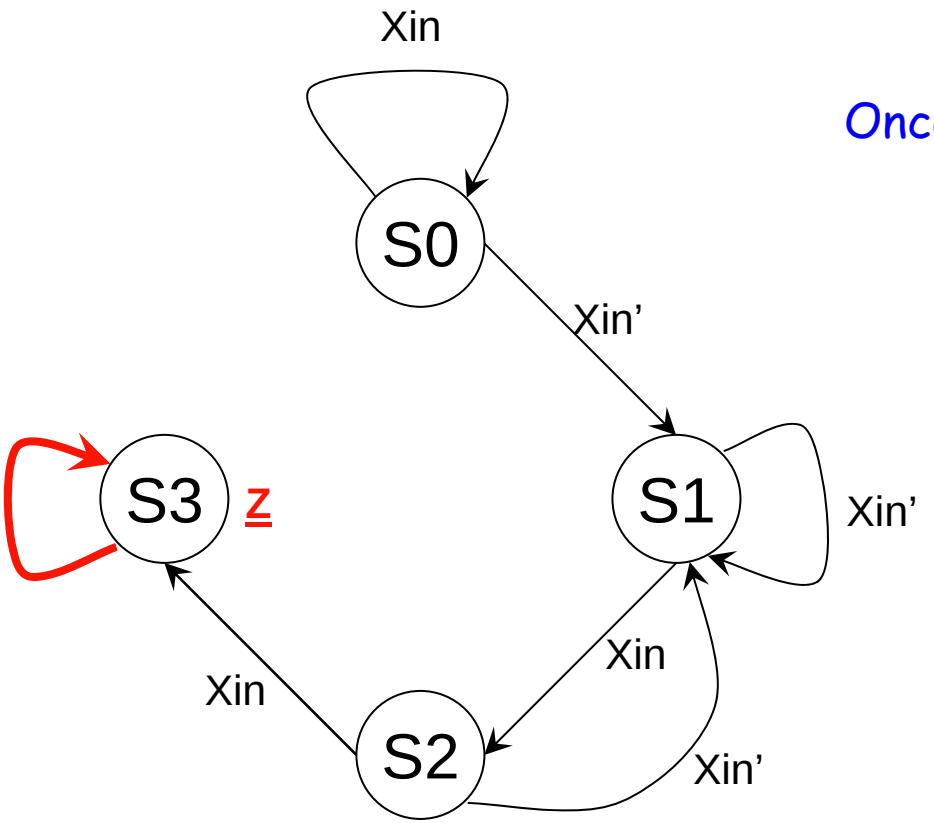


If Xin=1 again, we go to state S3

1..1..1..1..1..0..0..0..0..1..1

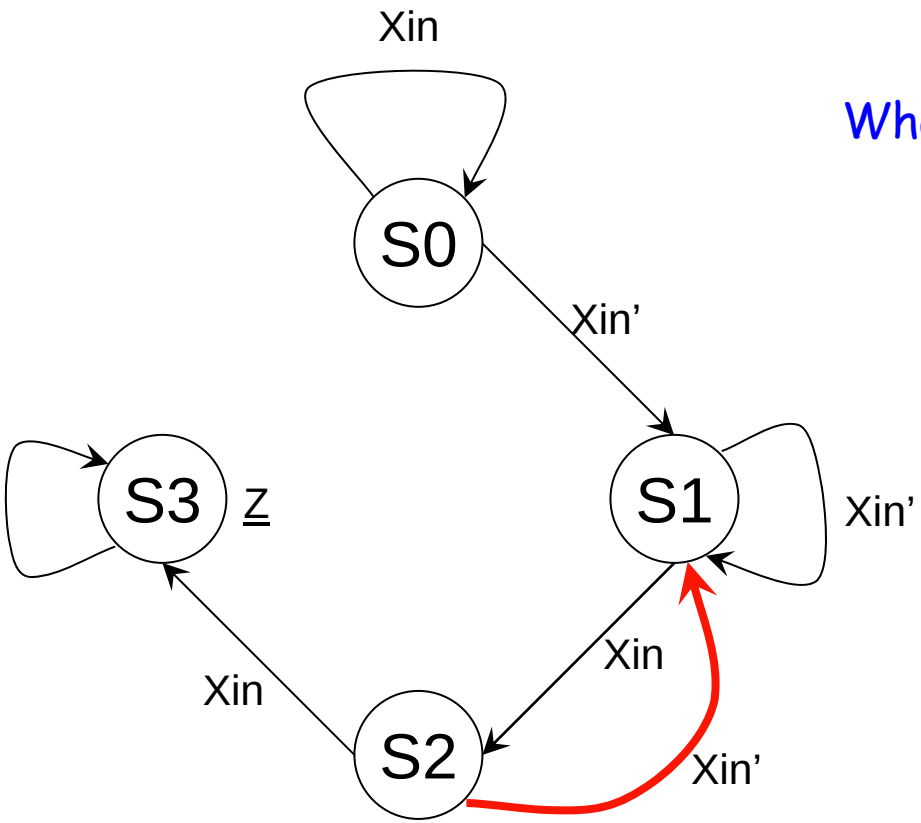
SUCCESS! Raise the Z output

A Sequence Detector FSM



Once we enter state S3, we never leave…

A Sequence Detector FSM



What if we don't see the second Xin=1?

1..1..1..1..1..0..0..0..0..1..0..

Implementing the Sequence Detector FSM

1. Create symbolic Transition Table
2. Assign state encoding
3. Create conventional Transition Table
4. Do standard implementation steps

Xin	CS	NS	Z
0	S0	S1	0
1	S0	S0	0
0	S1	S1	0
1	S1	S2	0
0	S2	S1	0
1	S2	S3	0
-	S3	S3	1

Symbolic TT

S0 = 00
S1 = 01
S2 = 10
S3 = 11

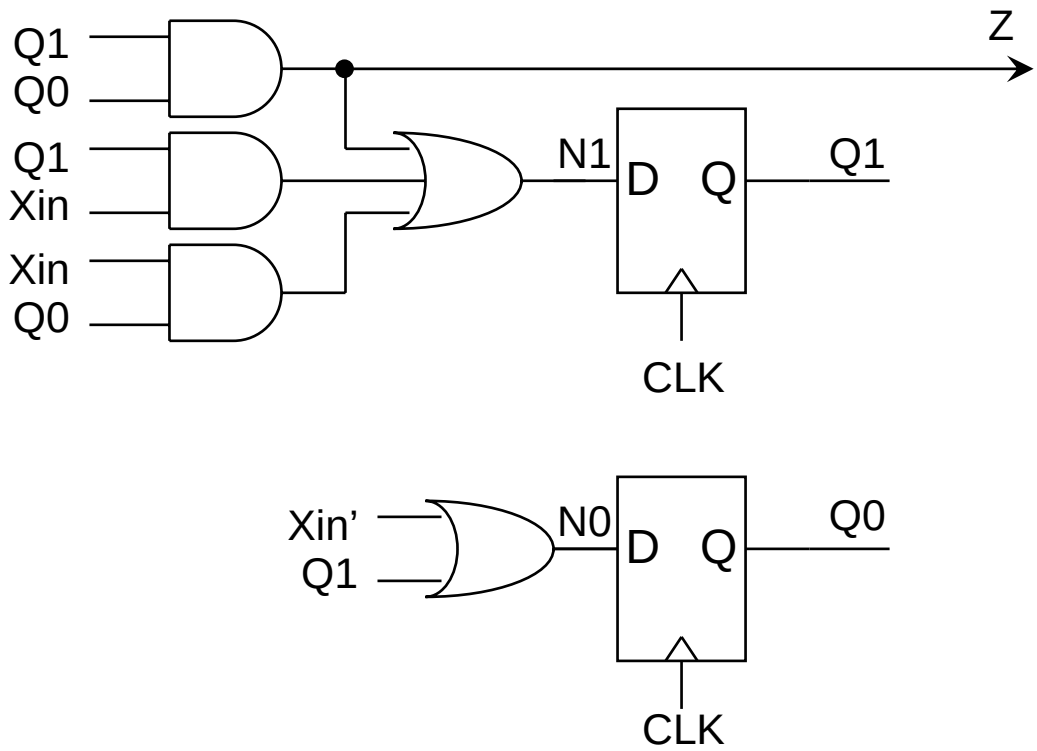
State Assignment

Xin	Q1	Q0	N1	N0	Z
0	0	0	0	1	0
1	0	0	0	0	0
0	0	1	0	1	0
1	0	1	1	0	0
0	1	0	0	1	0
1	1	0	1	1	0
-	1	1	1	1	1

Conventional TT

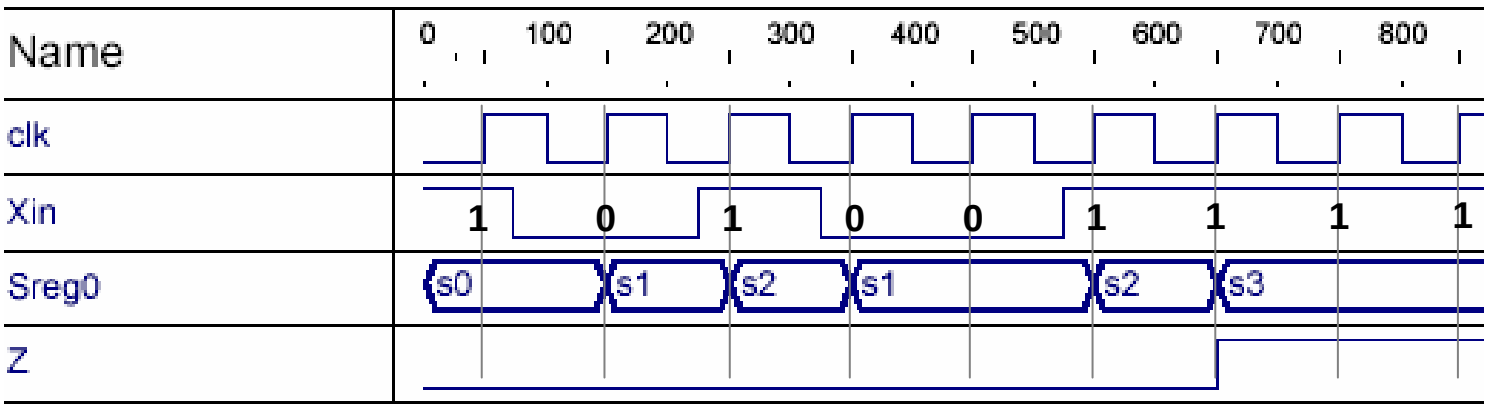
Sequence Detector Implementation

N1 = Q1•Q0 + Xin•Q1 + Xin•Q0
N0 = Xin' + Q1
Z = Q1•Q0



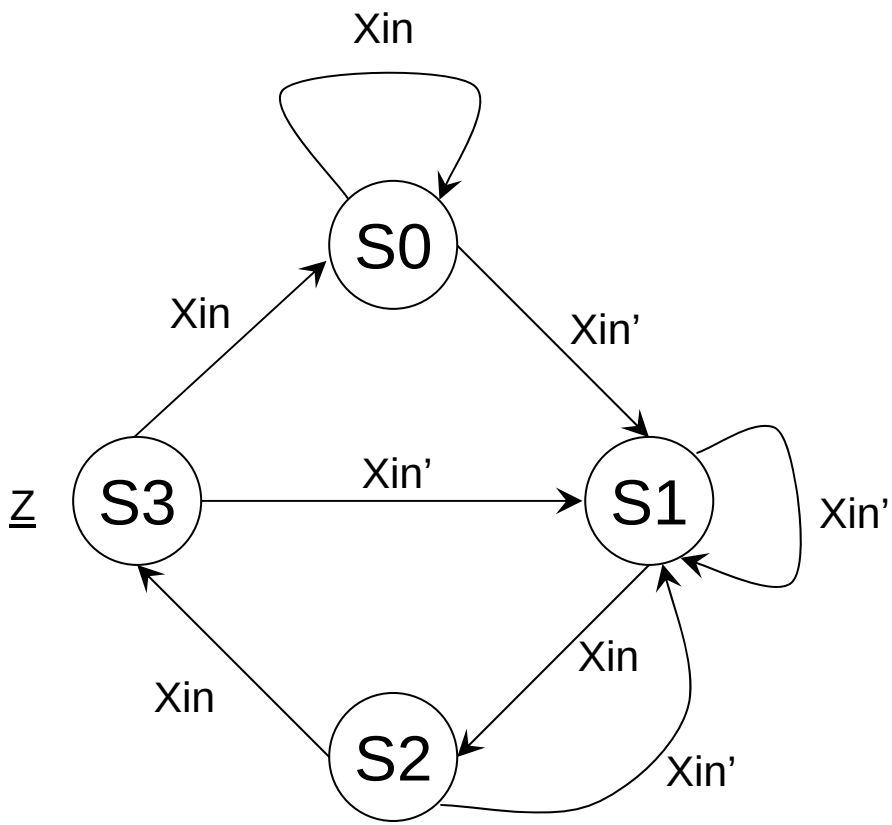
A Problem With the Sequence Detector

- This FSM only detects first occurrence of 011 on input…
- Here is a timing diagram
  - actual screenshot from a simulation

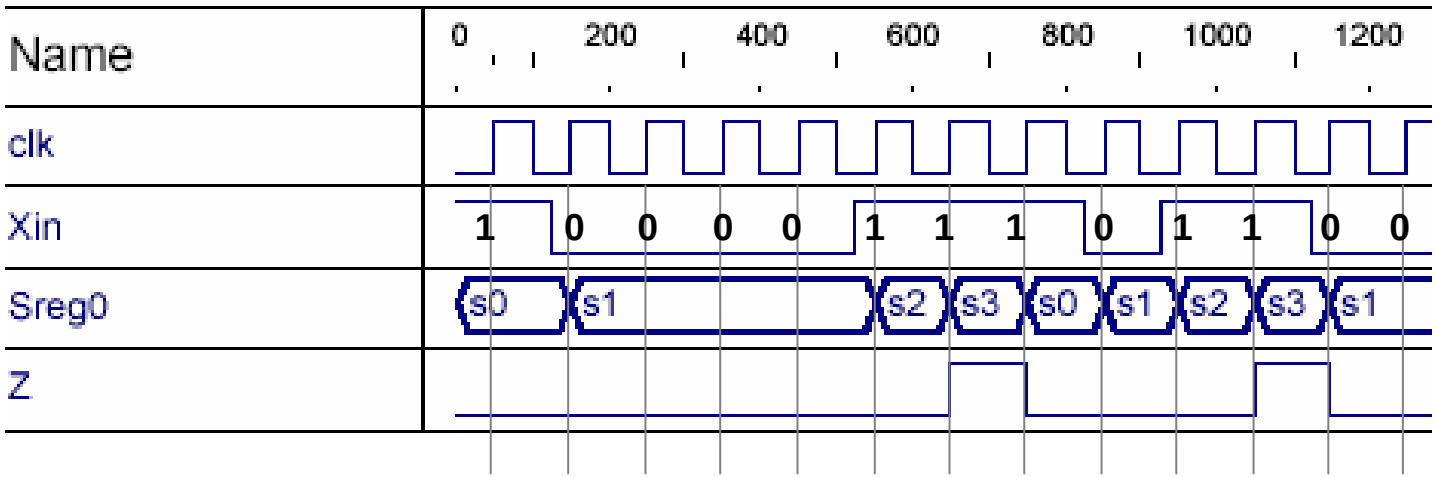


Improved Sequence Detector

- This starts over each time 011 is detected…



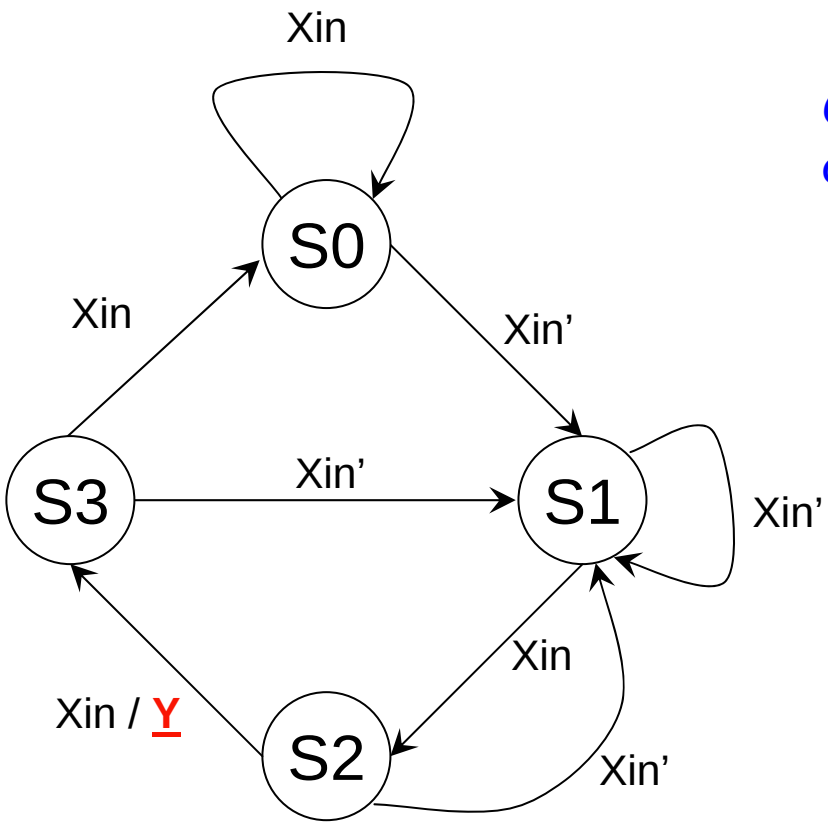
Improved Detector Timing Diagram



Z=1 any time state is S3

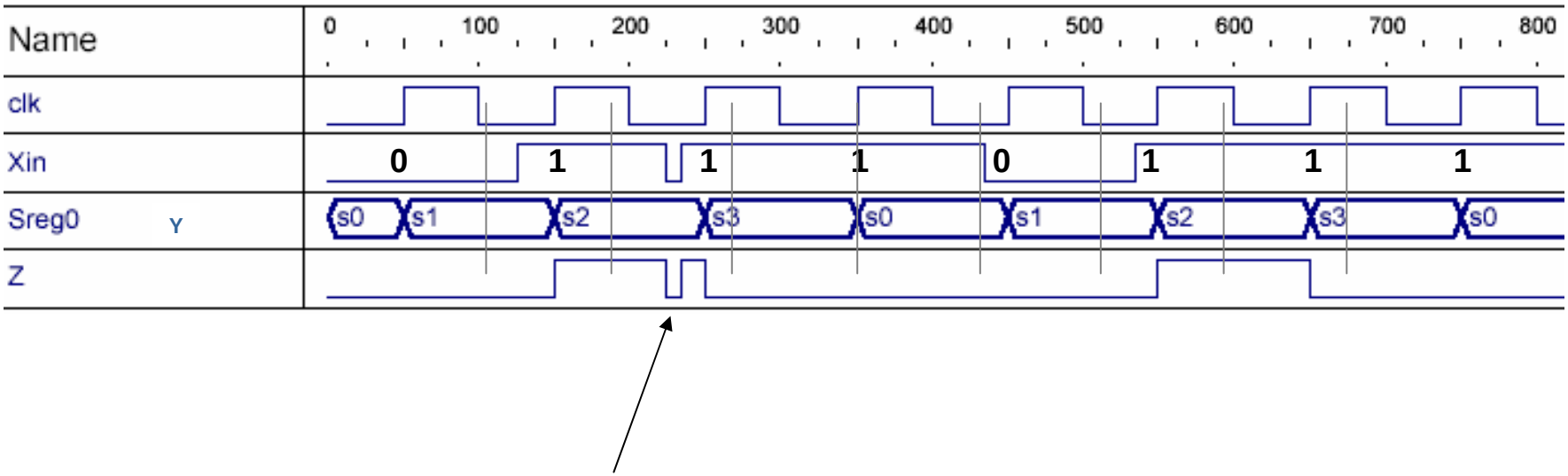
Looking at Xin, does it look like Z is a cycle late?

Mealy Version of Sequence Detector



Output is asserted *during* the second '1' of the 011 sequence…

Mealy Version Timing Diagram

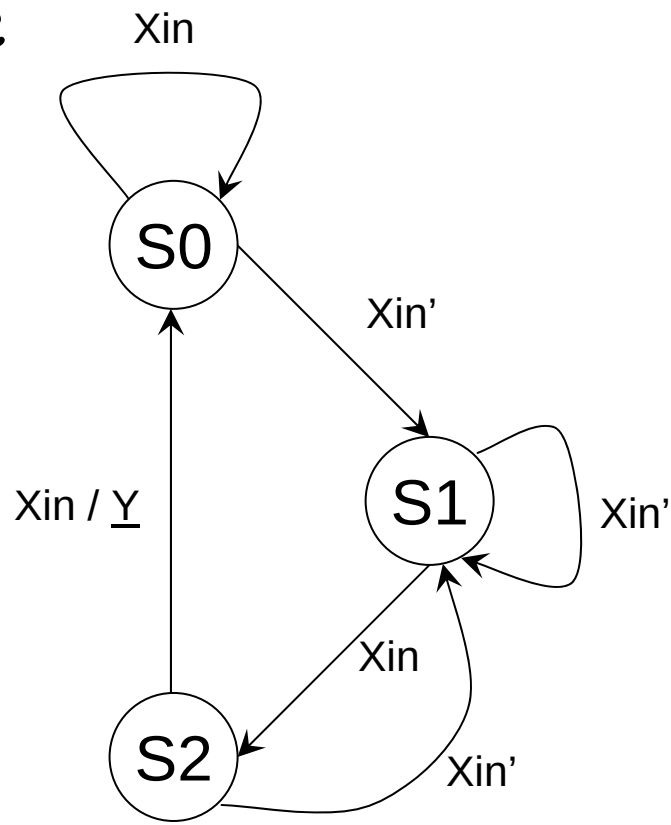


Note how Mealy output follows input during state S2

Output is a cycle earlier than in Moore machine – it appears in S2 rather than in S3

Simplified Mealy Sequence Detector

- A characteristic ofMealy machines is they often require fewer states than Moore machines
- Here is simplified but equivalent FSM



Example FSM's

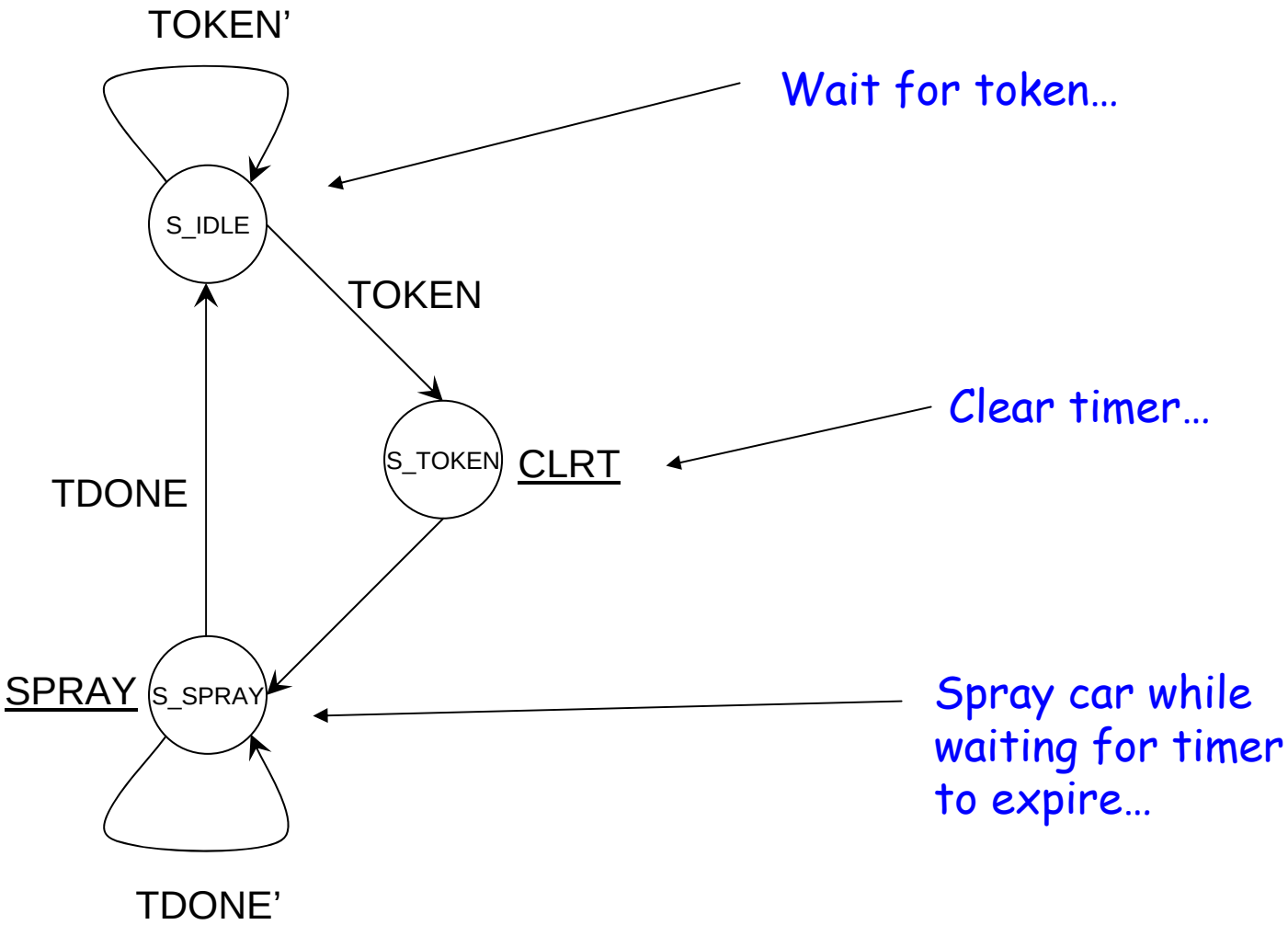
Two Car Wash Controllers

Basic Car Wash FSM Operation

1. Wait for a token to be inserted
2. Reset timer
3. Turn on water pump until timer expires
4. Start over

- This assumes existence of:
  - Token acceptance mechanism
  - Timer
  - Digitally-controlled water pump

Basic Car Wash FSM SG

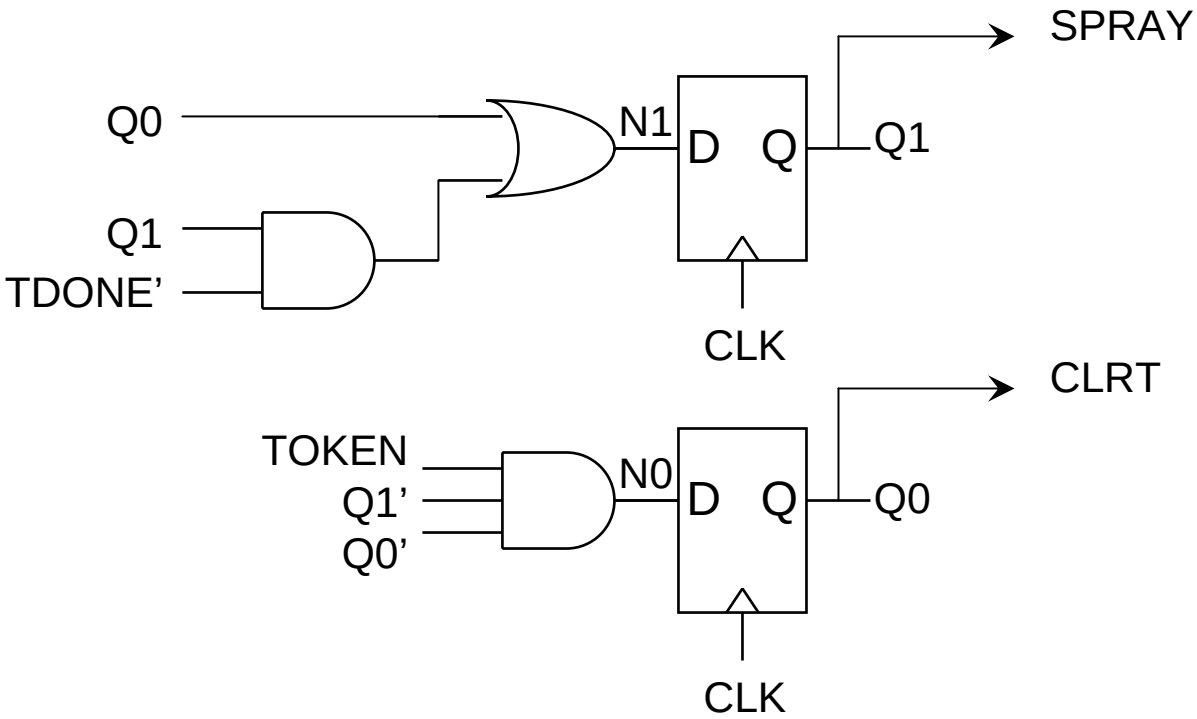


TOKEN	TDONE	CS	NS	CLRT	SPRAY
0	-	S_IDLE	S_IDLE	0	0
1	-	S_IDLE	S_TOKEN	0	0
-	-	S_TOKEN	S_SPRAY	1	0
-	0	S_SPRAY	S_SPRAY	0	1
-	1	S_SPRAY	S_IDLE	0	1

TOKEN	TDONE	Q1 Q0	N1 N0	CLRT	SPRAY
0	-	0 0	0 0	0	0
1	-	0 0	0 1	0	0
-	-	0 1	1 0	1	0
-	0	1 0	1 0	0	1
-	1	1 0	0 0	0	1

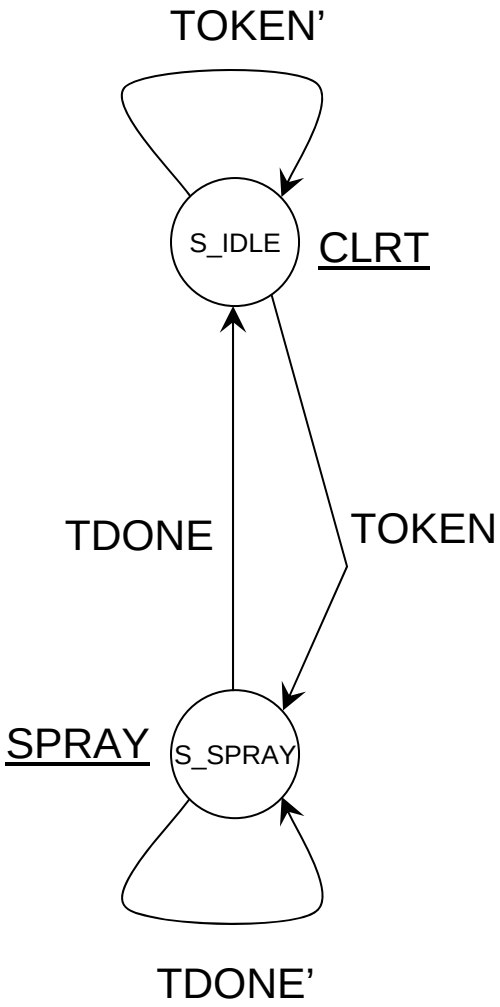
TOKEN	TDONE	Q1	Q0	N1	N0	CLRT	SPRAY
0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0
0	0	1	0	1	0	0	1
0	0	1	1	X	X	X	X
0	1	0	0	0	0	0	0
0	1	0	1	1	0	1	0
0	1	1	0	0	0	0	1
0	1	1	1	X	X	X	X
1	0	0	0	0	1	0	0
1	0	0	1	1	0	1	0
1	0	1	0	1	0	0	1
1	0	1	1	X	X	X	X
1	1	0	0	0	1	0	0
1	1	0	1	1	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	X	X	X	X

Basic Car Wash Implementation



This is what you get from a KMap solution

Simplified State Graph



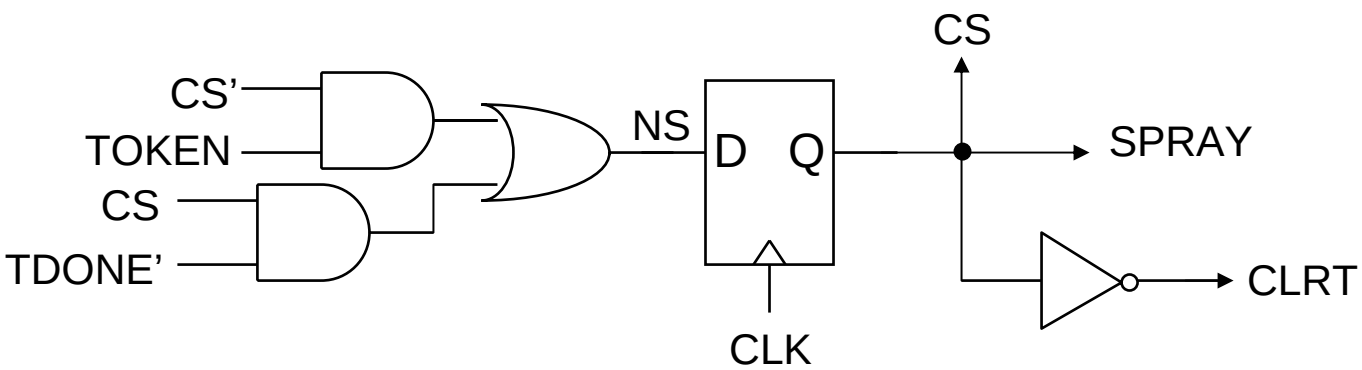
Clear timer while waiting for a token and eliminate a state

TOKEN	TDONE	CS	NS	CLRT	SPRAY
0	-	S_IDLE	S_IDLE	1	0
1	-	S_IDLE	S_SPRAY	1	0
-	0	S_SPRAY	S_SPRAY	0	1
-	1	S_SPRAY	S_IDLE	0	1

Simplified Car Wash Implementation

State Encoding: S_IDLE = 0, S_SPRAY = 1

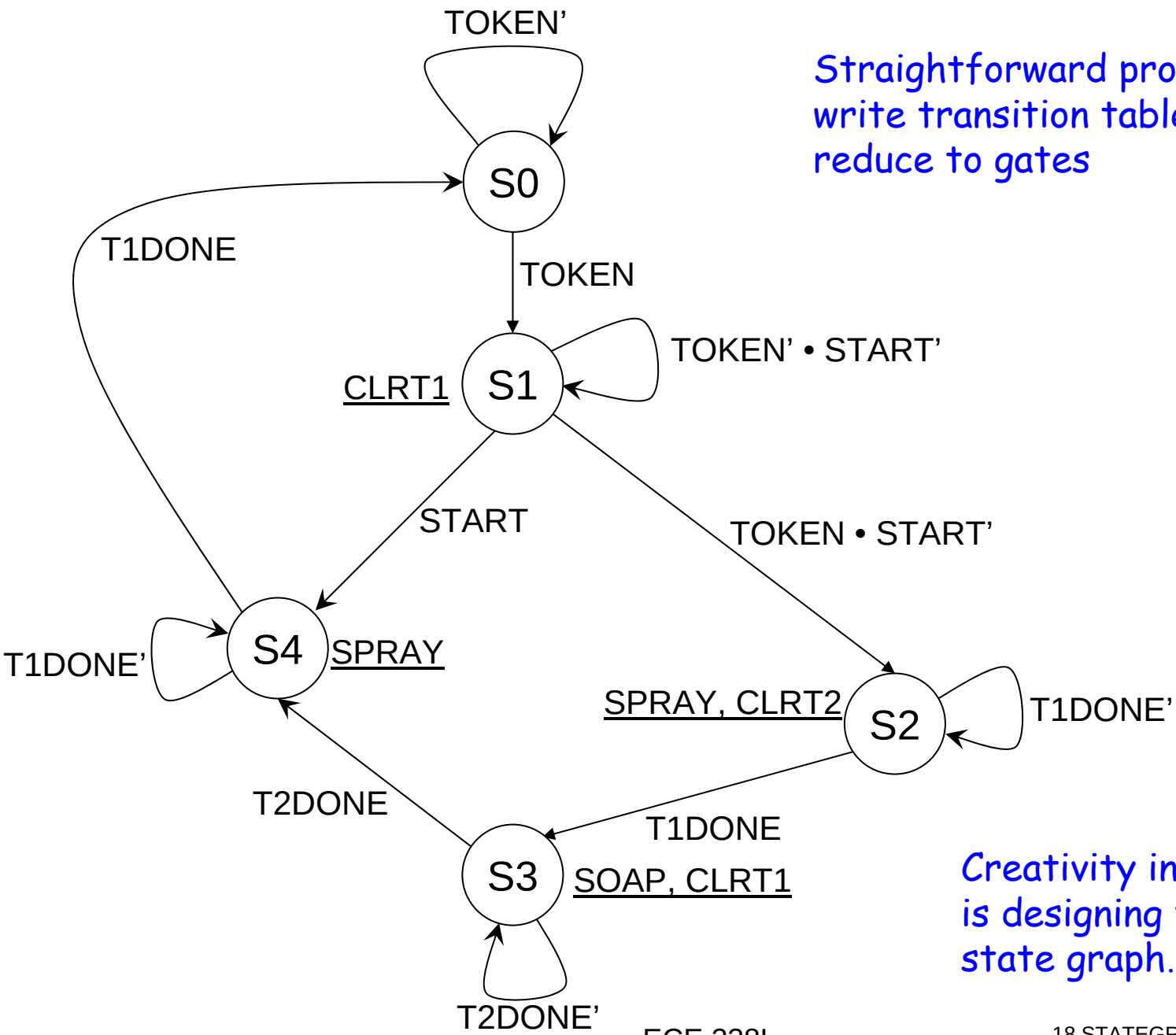
NS = CS'•TOKEN + CS•TDONE'
CLRT = CS'
SPRAY = CS



A Fancy Car Wash Controller

- Two types of washes:
  - Regular wash (spray only) = 1 token
  - Deluxe wash (spray, soap, spray) = 2 tokens

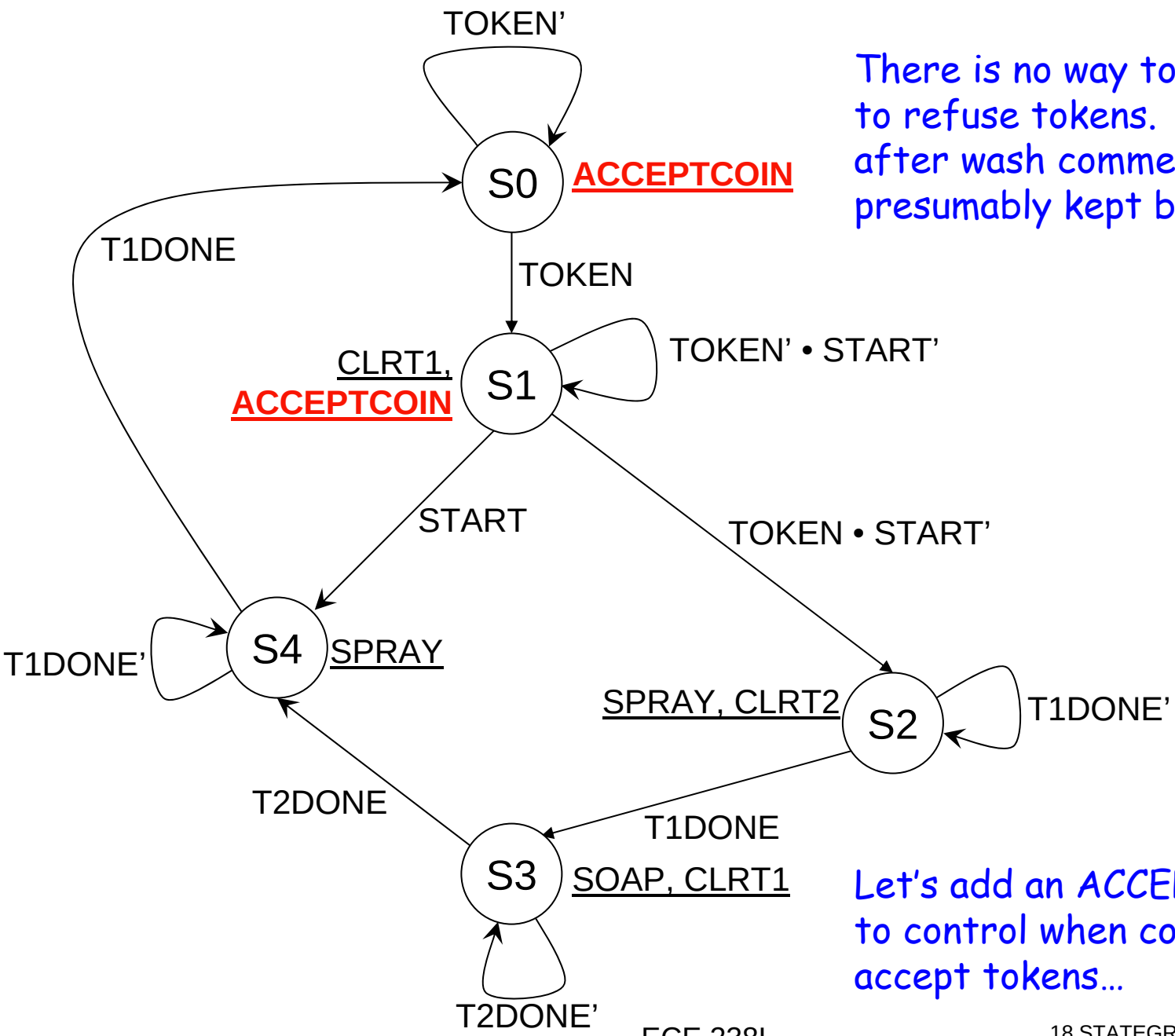
- Customer:
  - Inserts 1 token and pushes START for Regular
  - Inserts 2 tokens for Deluxe



Straightforward process to write transition table and reduce to gates

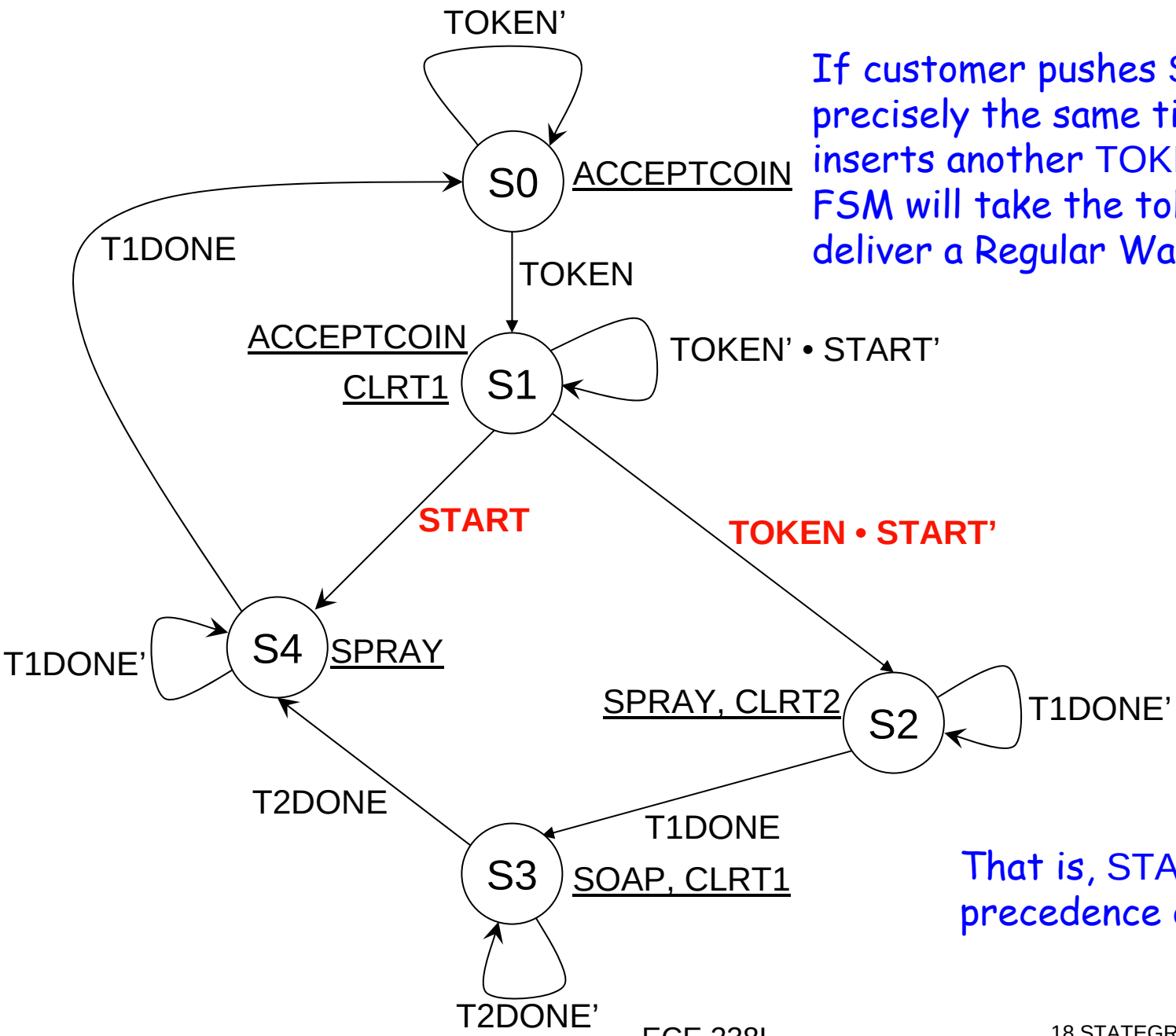
Creativity in FSM design is designing the initial state graph…

Enhancements to Fancy Controller



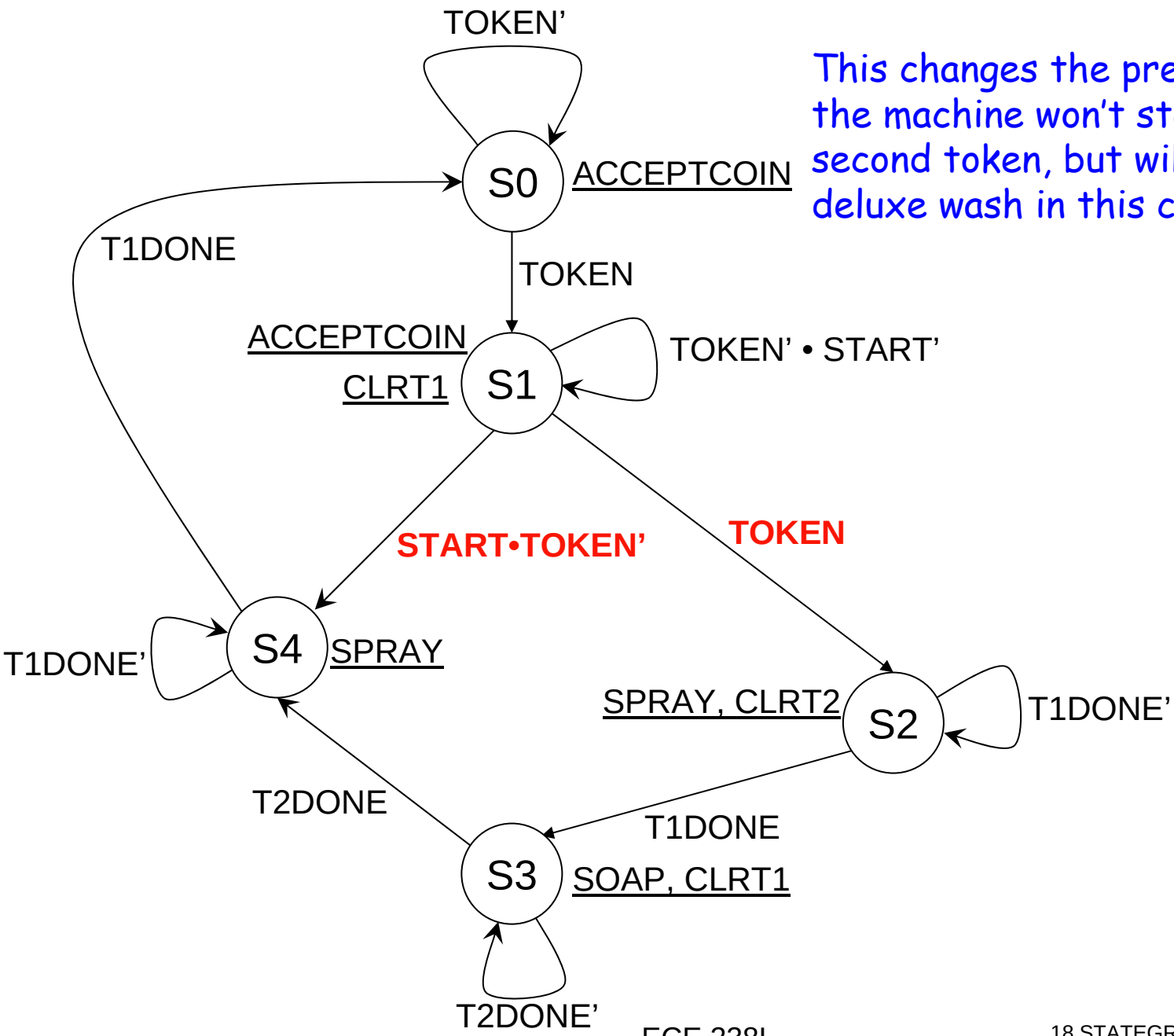
There is no way to cause coinbox to refuse tokens. Coins inserted after wash commences are presumably kept by the machine.

Let's add an ACCEPTCOIN output to control when coinbox will accept tokens…



If customer pushes START at precisely the same time as he inserts another TOKEN, the FSM will take the token, but deliver a Regular Wash.

That is, START has precedence over TOKEN



This changes the precedence so the machine won't steal the second token, but will give a deluxe wash in this case.

Completeness and Conflict Revisited

- State S1 is a typical problem spot
- Carefully analyze the state graph to ensure no conflicts exist and that all cases covered.

Resetting State Machines

- Ability to reset the FSM is essential for testing most systems

- Always include a reset capability
  - Add CLR signal to state graph
  - Use flip flops with clear inputs

  - Either method will work

Another Example

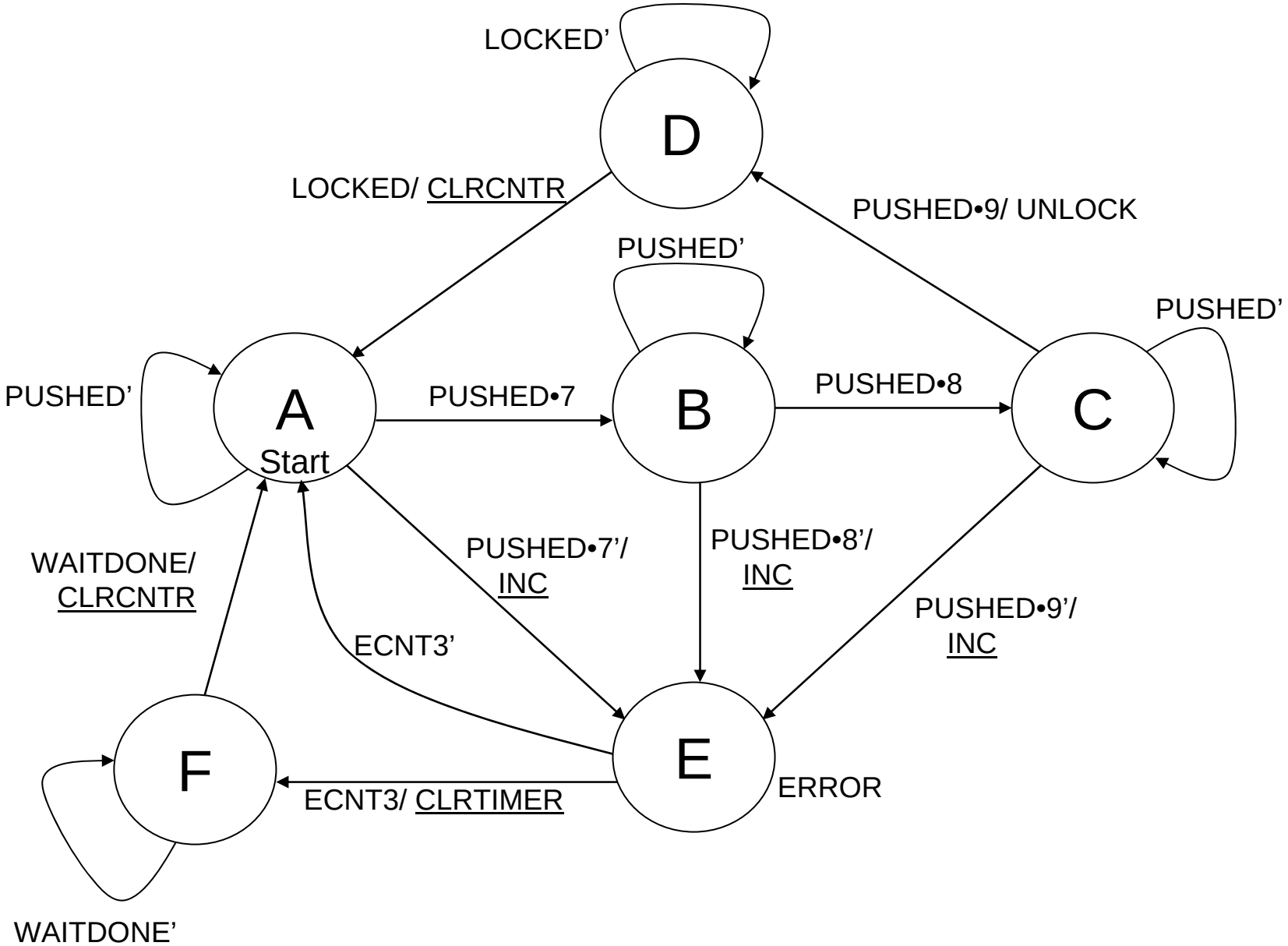
An Electronic Key Lock

Example: Electronic Key Lock

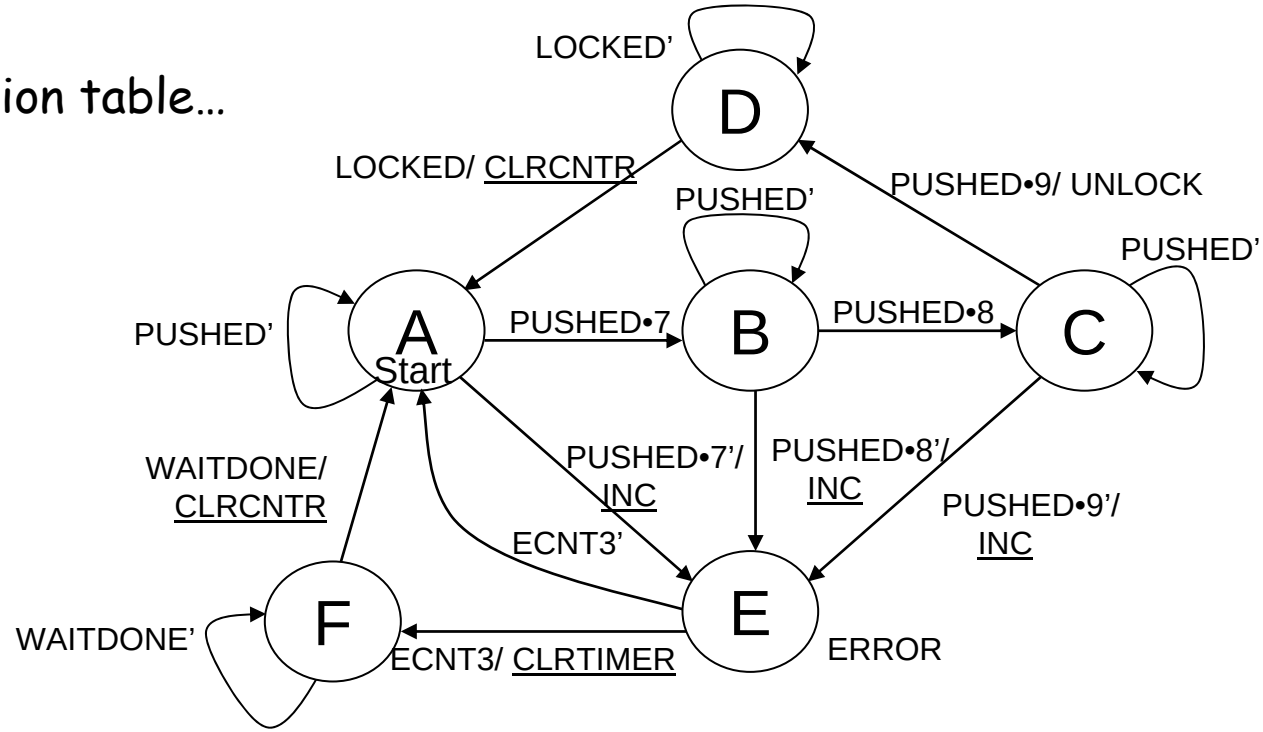
1) There are 10 keypads 0-9
2) The unlock sequence is 7..8..9
3) When a pad is pushed the signal for that number is asserted
4) When *any* of the keypads are pushed a *PUSHED* signal is asserted
5) If you push a number out of sequence, you get an error indicator and you get to start over
6) After three wrong tries, you must wait ½ hr. before trying again
7) If you entered the correct sequence, the lock unlocks.
8) Once the lock has opened, nothing happens until it is manually locked again.

Inputs: Keypads signals 0-9
PUSHED signal
ECNT3 (error count = 3)
WAITDONE
LOCKED

Outputs: INC
CLRTIMER
CLRCNTR
ERROR
UNLOCK


LOCKED'
D
LOCKED/ CLRCNTR
PUSHED•9/ UNLOCK
PUSHED'
PUSHED'
PUSHED'
A
Start
PUSHED•7
B
PUSHED•8
C
PUSHED•7'/
INC
PUSHED•8'/
INC
WAITDONE/
CLRCNTR
PUSHED•9'/
INC
ECNT3'
F
E
ERROR
ECNT3/ CLRTIMER
WAITDONE'

Let's create the transition table…



PUSHED	7	8	9	ECNT3	WAITDONE	LOCKED	STATE	NEXTSTATE	INC	ERROR	CLRCNTR	CLRTIMER	UNLOCK
0	-	-	-	-	-	-	A	A	0	0	0	-	0
1	0	-	-	-	-	-	A	E	1	0	0	-	0
1	1	-	-	-	-	-	A	B	0	0	0	-	0
0	-	-	-	-	-	-	B	B	0	0	0	-	0
1	-	0	-	-	-	-	B	E	1	0	0	-	0
1	-	1	-	-	-	-	B	C	0	0	0	-	0
0	-	-	-	-	-	-	C	C	0	0	0	-	0
1	-	-	0	-	-	-	C	E	1	0	0	-	0
1	-	-	1	-	-	-	C	D	0	0	-	-	1
-	-	-	-	-	-	0	D	D	-	0	-	-	0
-	-	-	-	-	-	1	D	A	0	0	1	-	0
-	-	-	-	0	-	-	E	A	0	1	0	-	0
-	-	-	-	1	-	-	E	F	-	1	0	1	0
-	-	-	-	-	0	-	F	F	-	0	0	0	0
-	-	-	-	-	1	-	F	A	0	0	1	0	0

Now let's expand State A

PUSHED	7	8	9	ECNT3	WAITDONE	LOCKED	STATE	NEXTSTATE	INC	ERROR	CLRCNTR	CLRTIMER	UNLOCK
0	0	0	0	0	0	0	A	A	0	0	0	-	0
0	0	0	0	0	0	1	A	A	0	0	0	-	0
0	0	0	0	0	1	0	A	A	0	0	0	-	0
0	0	0	0	0	1	1	A	A	0	0	0	-	0
0	0	0	0	1	0	0	A	A	0	0	0	-	0
0	0	0	0	1	0	1	A	A	0	0	0	-	0
0	0	0	0	1	1	0	A	A	0	0	0	-	0
0	0	0	0	1	1	1	A	A	0	0	0	-	0
0	0	0	1	0	0	0	A	A	0	0	0	-	0
0	0	0	1	0	0	1	A	A	0	0	0	-	0
0	0	0	1	0	1	0	A	A	0	0	0	-	0
0	0	0	1	0	1	1	A	A	0	0	0	-	0
0	0	0	1	1	0	0	A	A	0	0	0	-	0
0	0	0	1	1	0	1	A	A	0	0	0	-	0
0	0	0	1	1	1	0	A	A	0	0	0	-	0

This is going to get ugly fast…

- 7 inputs + 3 state bits = 1024 rows in TT
- Can you do a 10-variable KMap?
- There is a technique called One-Hot state machines we can use instead…