

**CpE358/CS381**

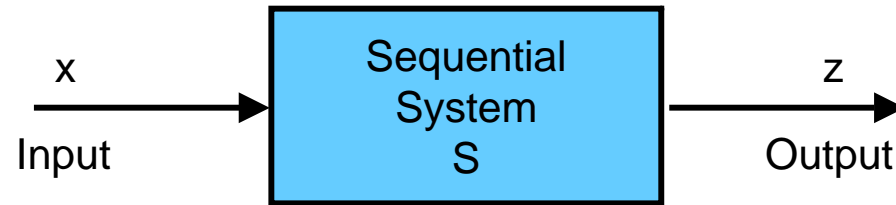
**Switching Theory and  
Logical Design**

**Class 16**

# Today

- Fundamental concepts of digital systems (Mano Chapter 1)
- Binary codes, number systems, and arithmetic (Ch 1)
- Boolean algebra (Ch 2)
- Simplification of switching equations (Ch 3)
- Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
- Combinatoric logical design including LSI implementation (Chapter 4)
- Flip-flops and state memory elements (Ch 5)
- Sequential logic analysis and design (Ch 5)
- Counters, shift register circuits (Ch 6)
- Hazards, Races, and time related issues in digital design (Ch 9)
- Synchronous vs. asynchronous design (Ch 9)
- Memory and Programmable logic (Ch 7)
- **Minimization of sequential systems**
- **Introduction to Finite Automata**

# Consider This Sequential System:



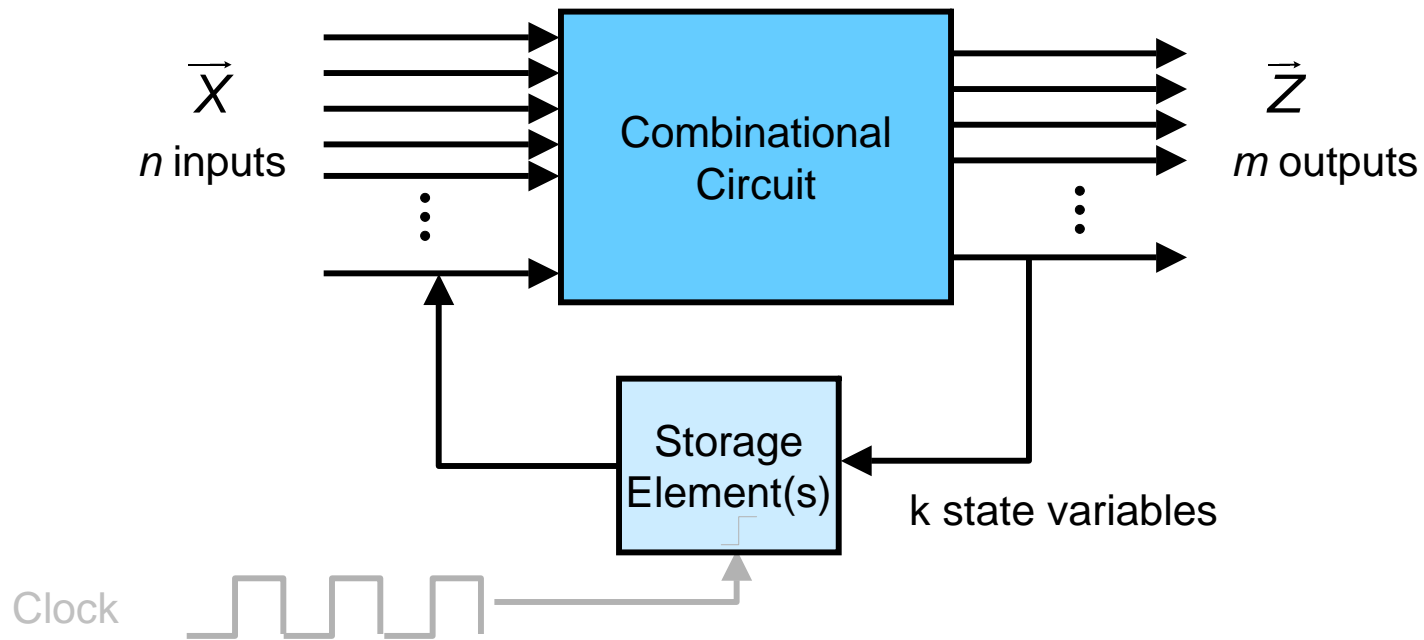
- Questions to ask:
  - Is S operating as intended?
    - Was the design correct?
    - Has a failure occurred?
  - Is S the simplest design that generates z for a given x?
    - Are all states necessary?
  - Can S be forced to go to a given state?
  - Are two systems,  $S_1$  and  $S_2$  distinguishable from each other?

- Reference for today's material: Hennie, Finite State Models for Logical Machines

# A Specific Sequential Design Issue

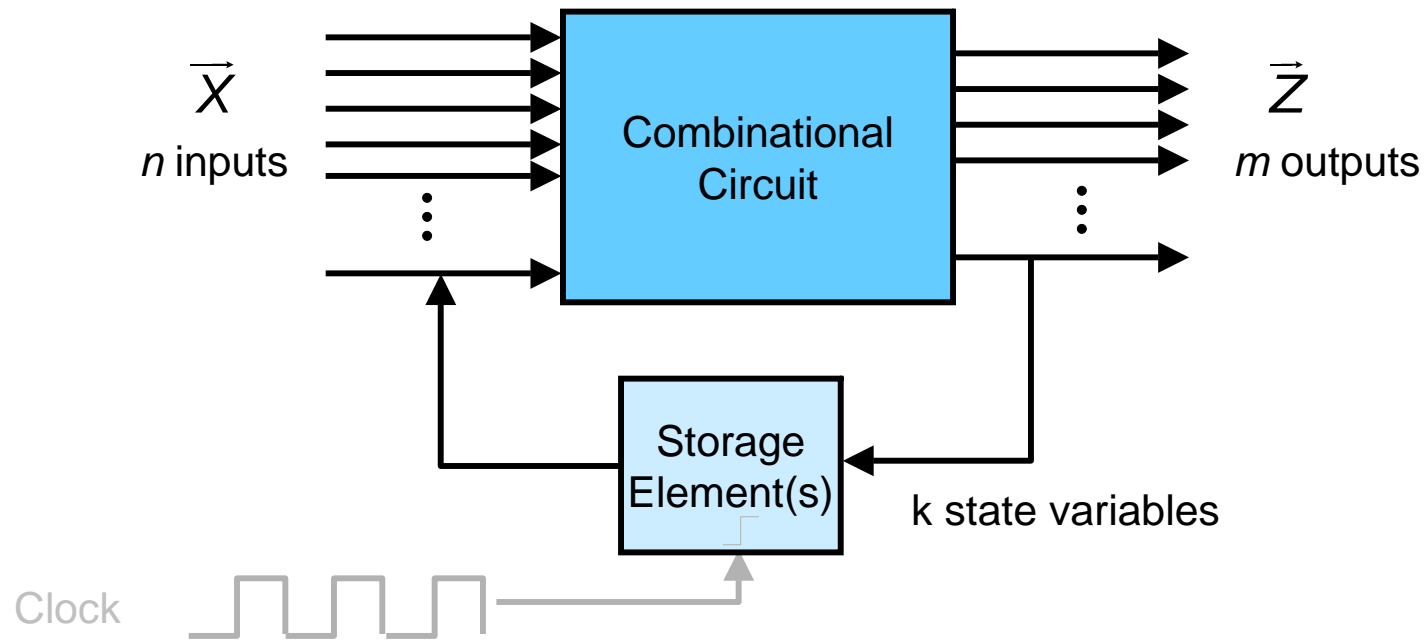
- Design Verification
  - Does system meet intended system requirements?
  - Does it perform properly over a range of operational conditions?
- Testing in Manufacture
  - Are there any faults in the product as produced?
- Field Diagnostics
  - Is the system still working properly?
  - If there is a fault, can it be localized?
- For each case, how much testing is needed?
- How much is practical?

# Sequential Design Testing



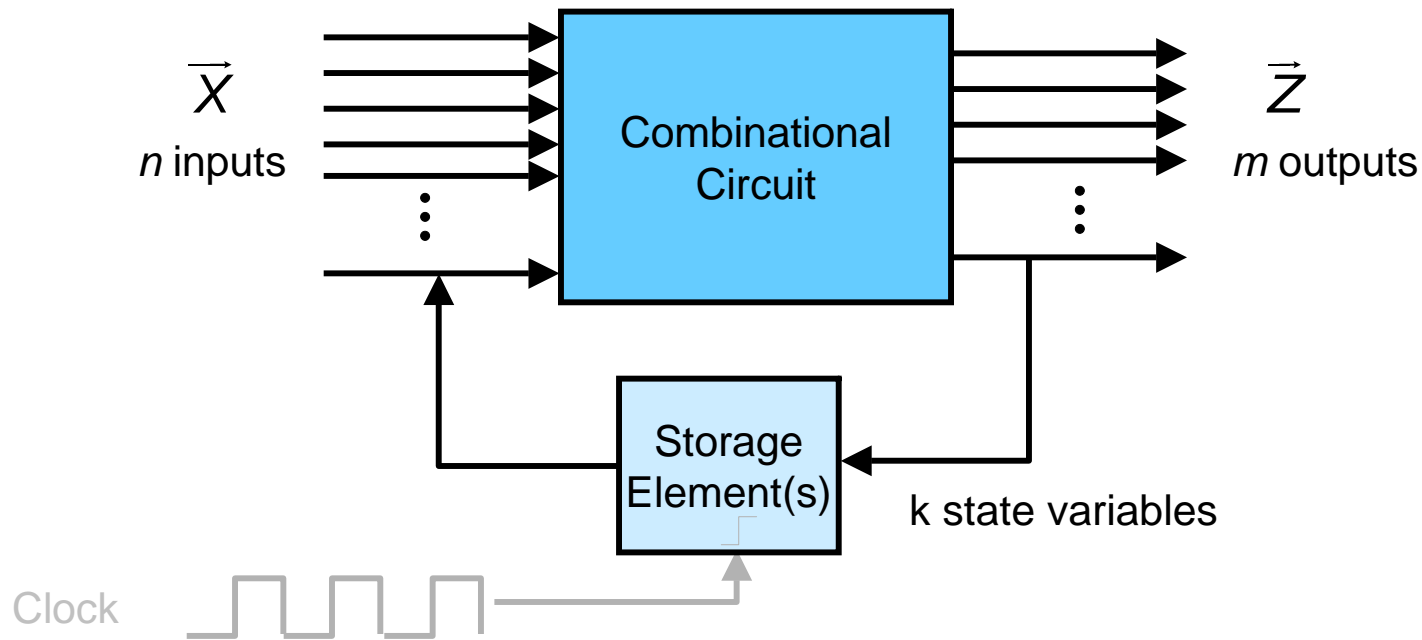
- How many tests need to be performed to find all possible faults in system?

# Sequential Design Testing



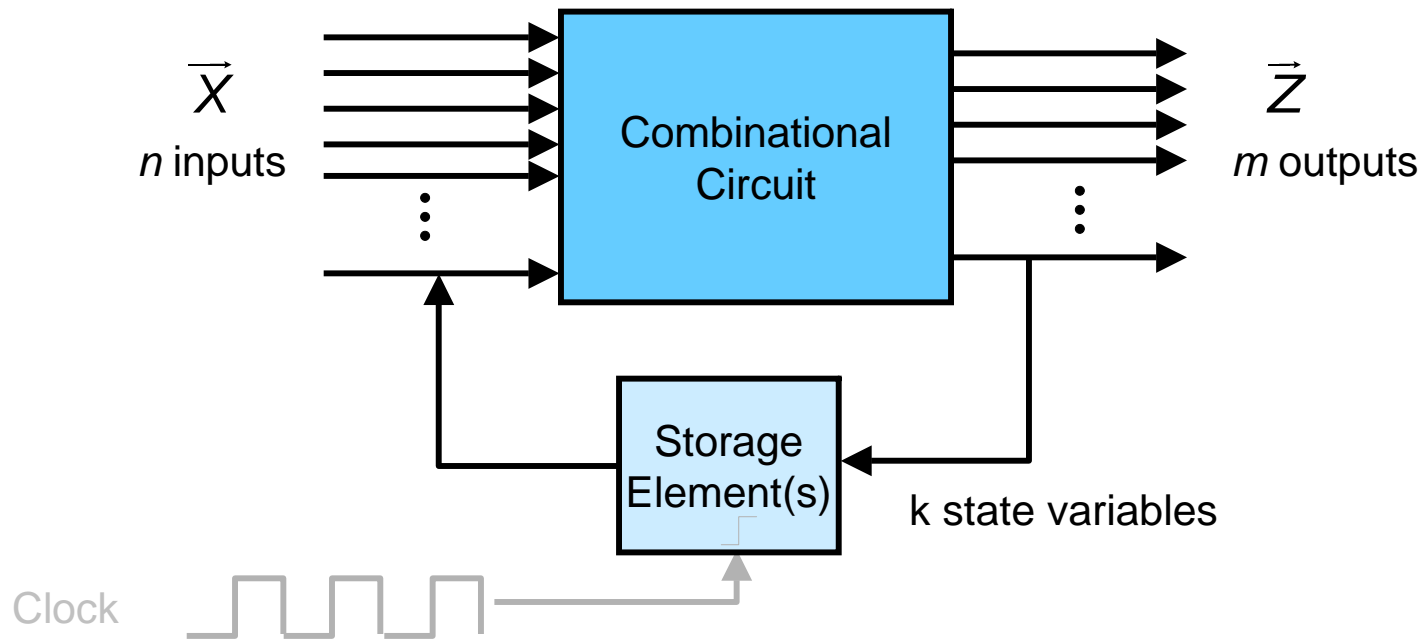
- How many tests need to be performed to find all possible faults in system?
  - **All storage elements need to be exercised**
  - **All combinatorial circuit elements need to be tested**

# Sequential Design Testing



- How many tests need to be performed to find all possible faults in system?
  - All storage elements need to be exercised
    - Testing all  $2^k$  states
    - Testing all  $2^k!$  possible state-state transitions
  - All combinatorial circuit elements need to be tested
    - Testing all  $2^n$  input combinations
    - Testing all  $2^n$  input combinations in all  $2^k$  states

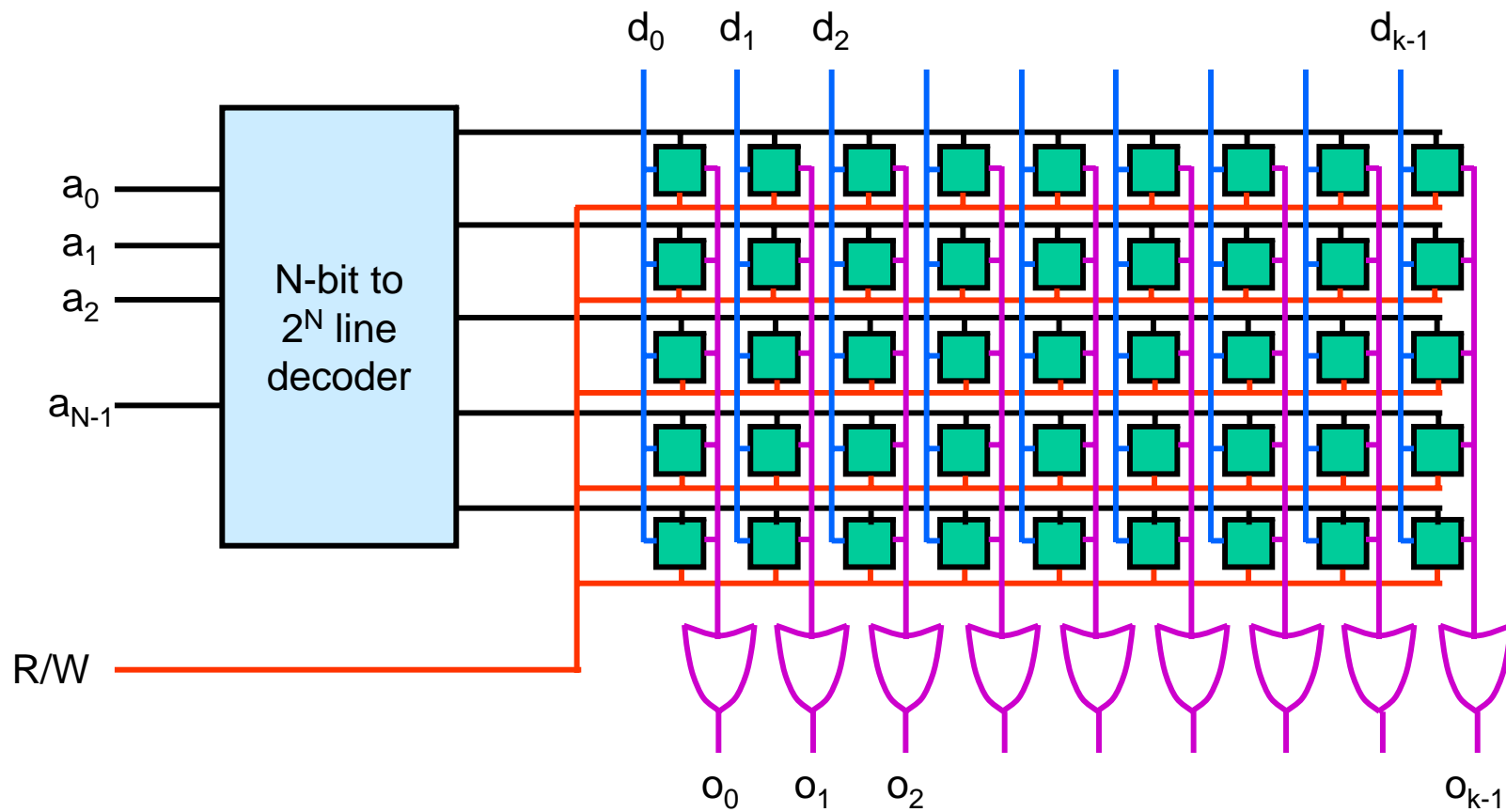
# Sequential Design Testing



- How many tests need to be performed to find all possible faults in system?
  - All storage elements need to be exercised
    - Testing all  $2^k$  states
    - Testing all  $2^k!$  possible state-state transitions
  - All combinatorial circuit elements need to be tested
    - Testing all  $2^n$  input combinations
    - Testing all  $2^n$  input combinations in all  $2^k$  states

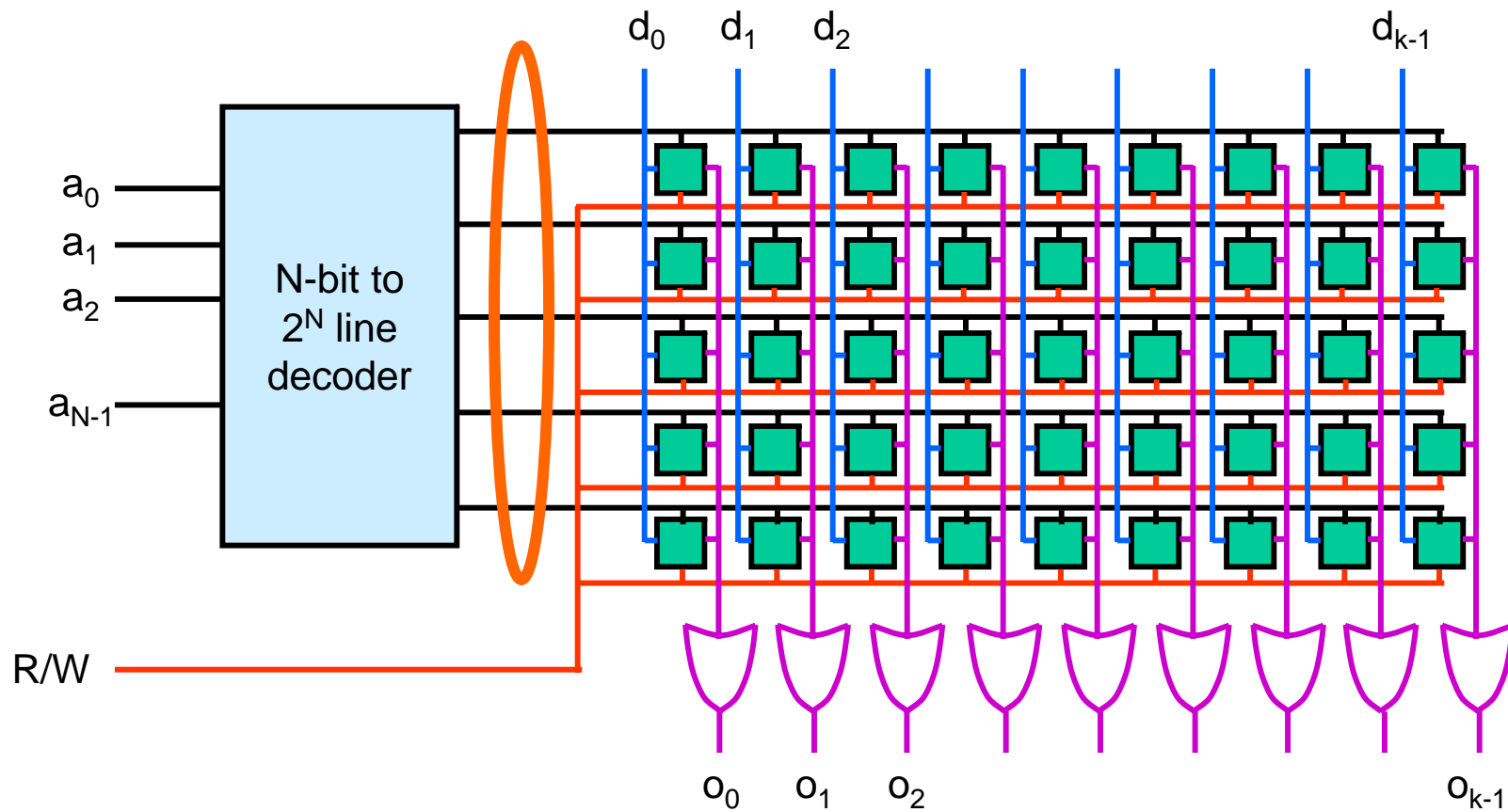
But internal state is generally not observable!

# Sequential Design Testing



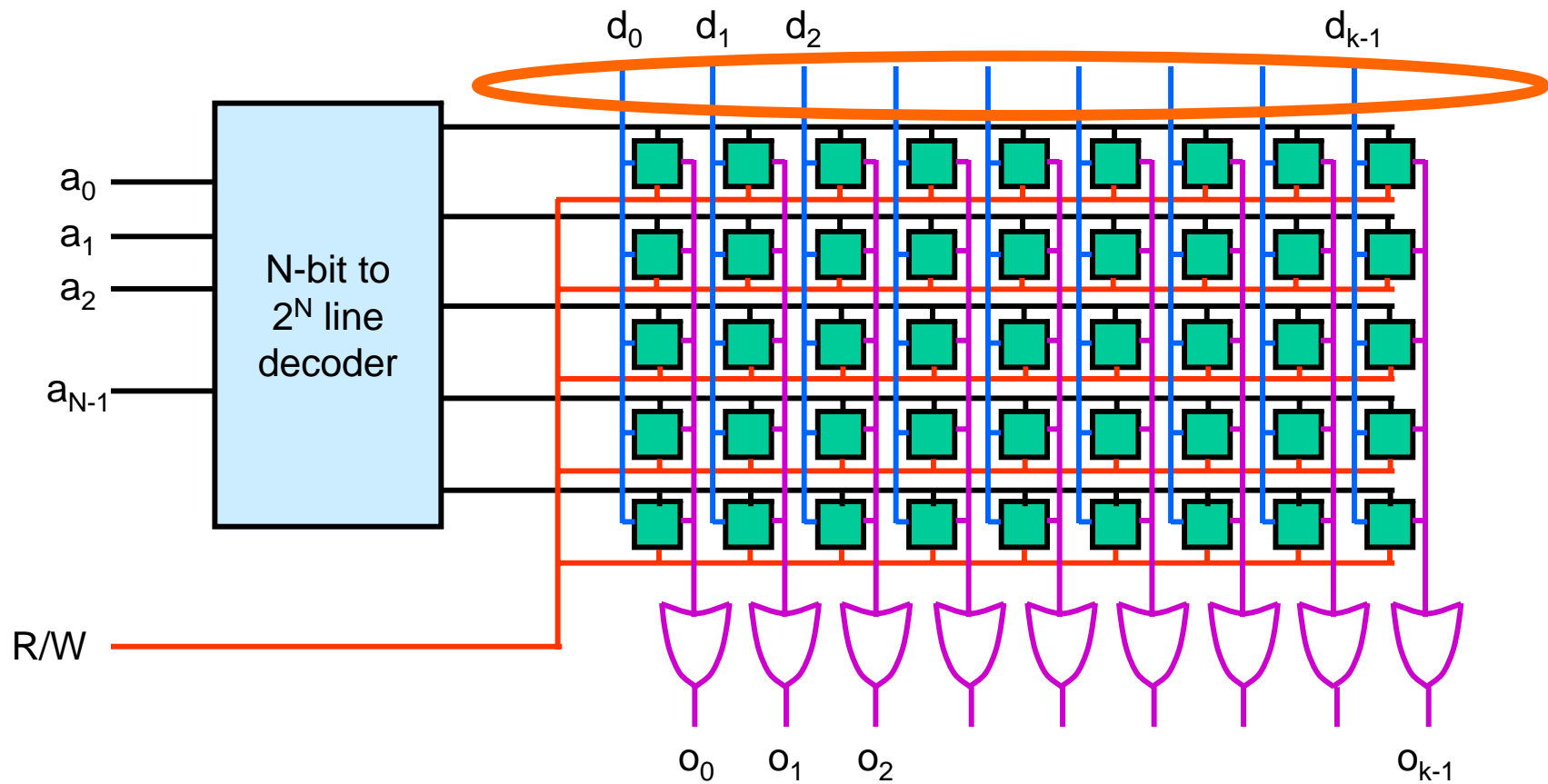
# Sequential Design Testing

Do select lines operate correctly?



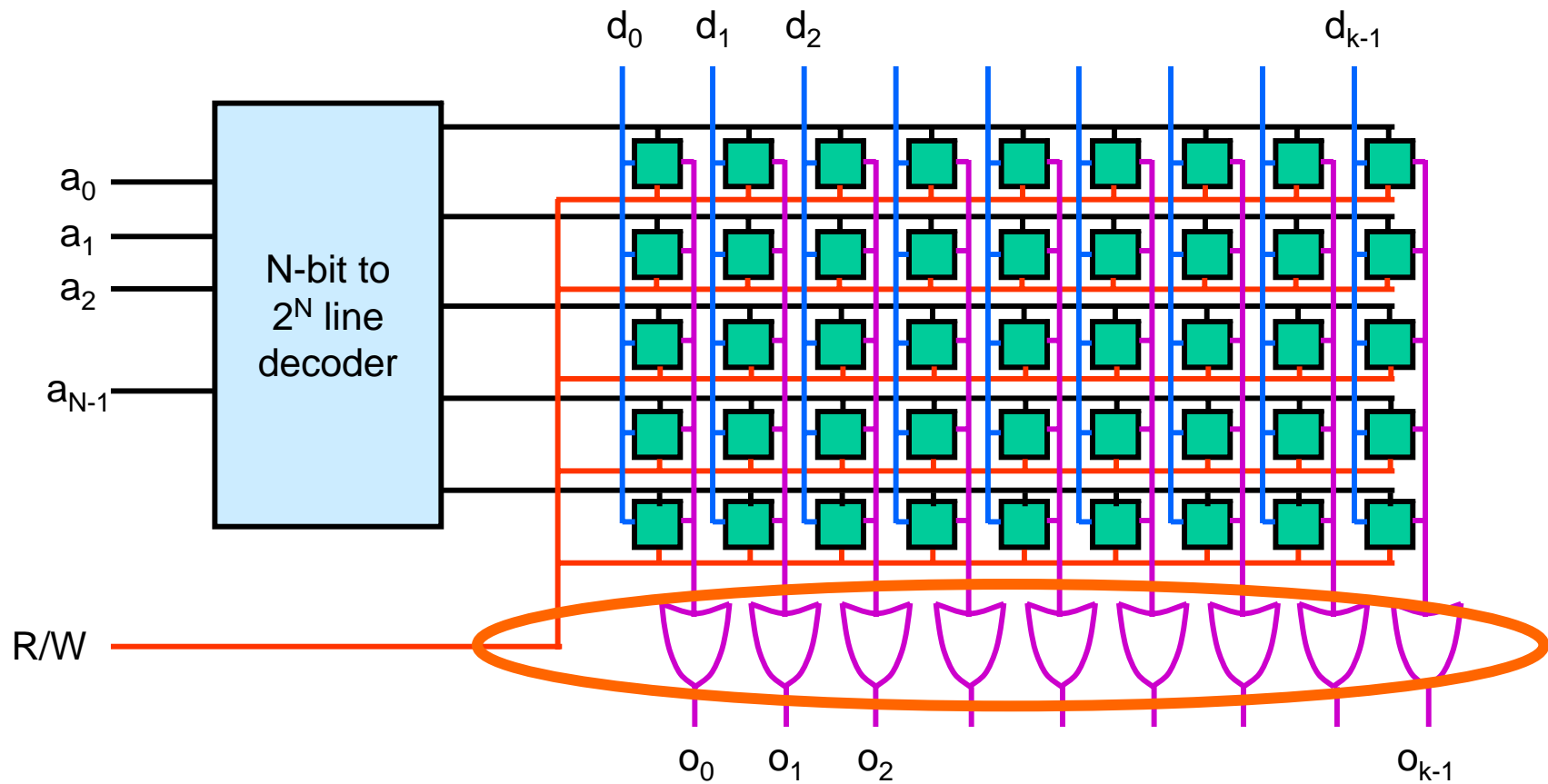
# Sequential Design Testing

Do data lines operate correctly?



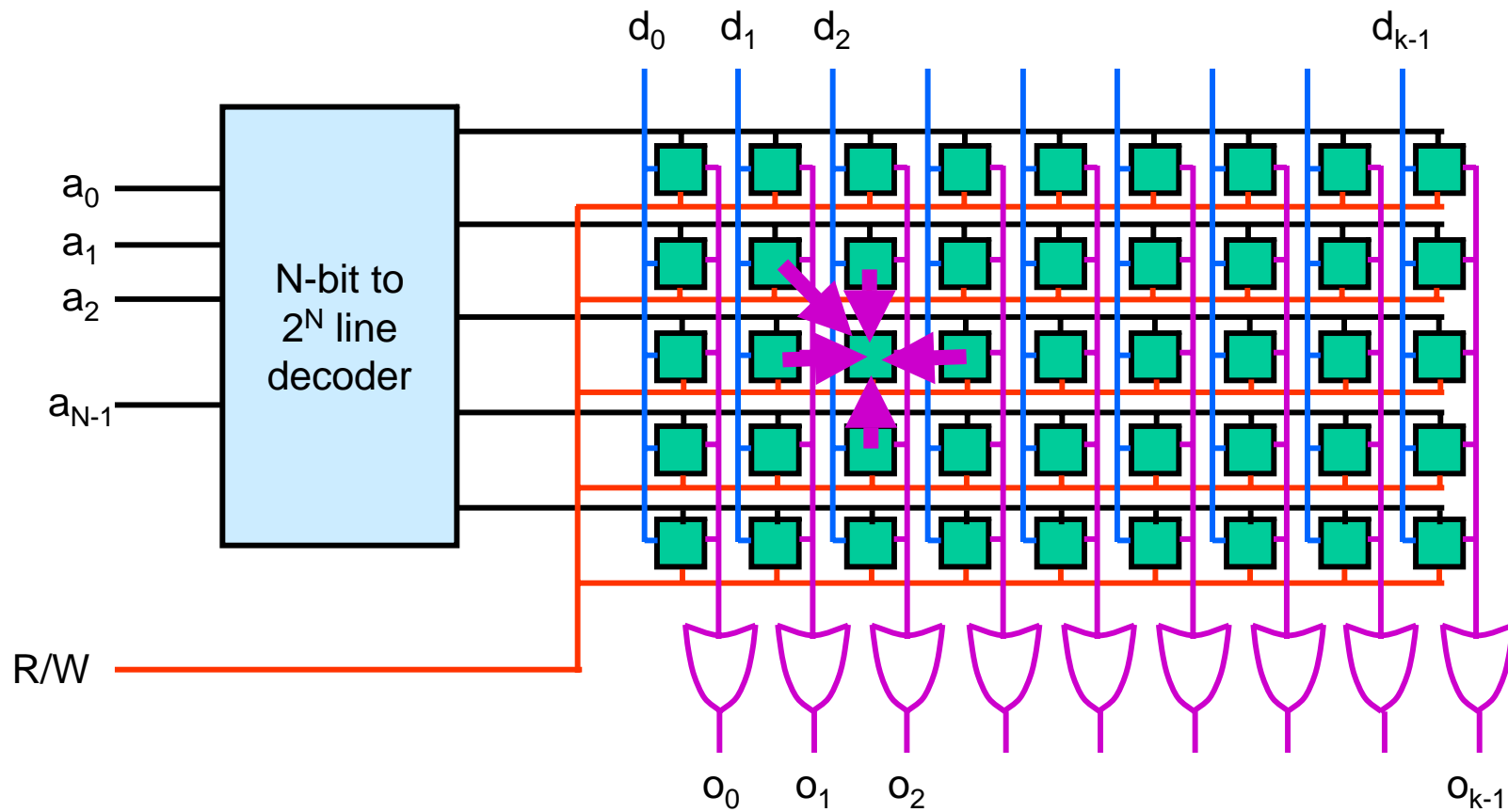
# Sequential Design Testing

Do output lines operate correctly?

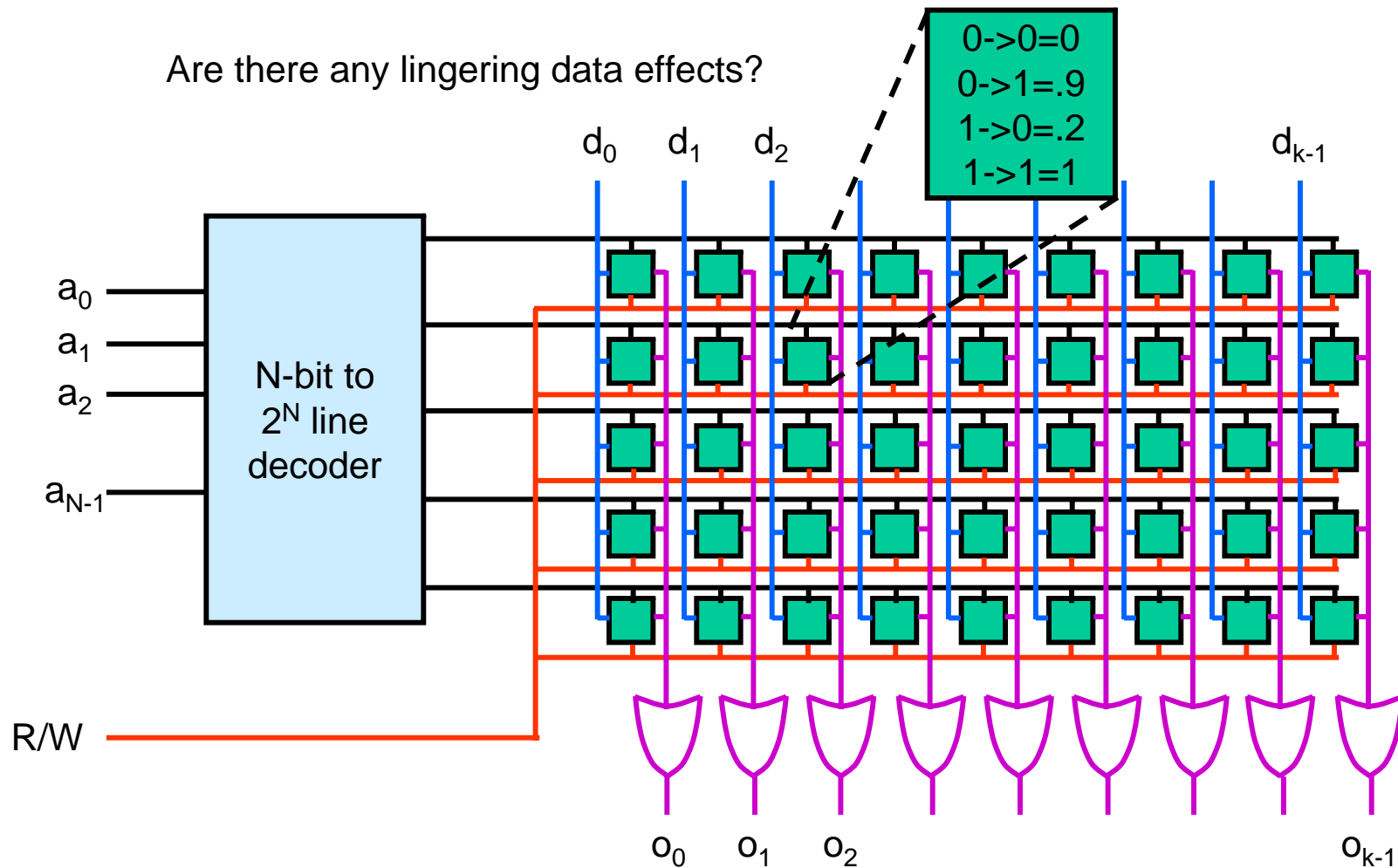


# Sequential Design Testing

Are there any cell-to-cell interactions?

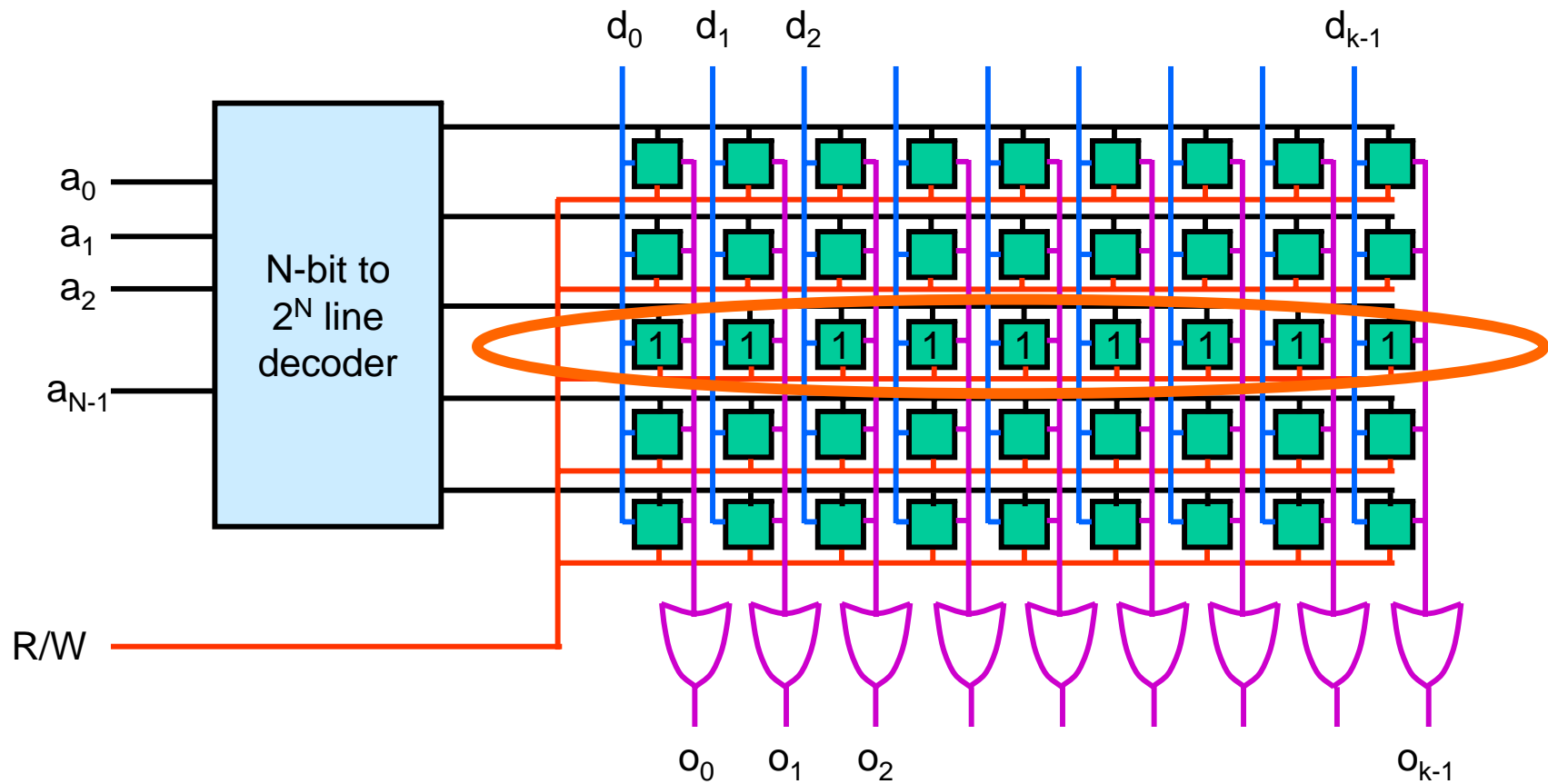


# Sequential Design Testing



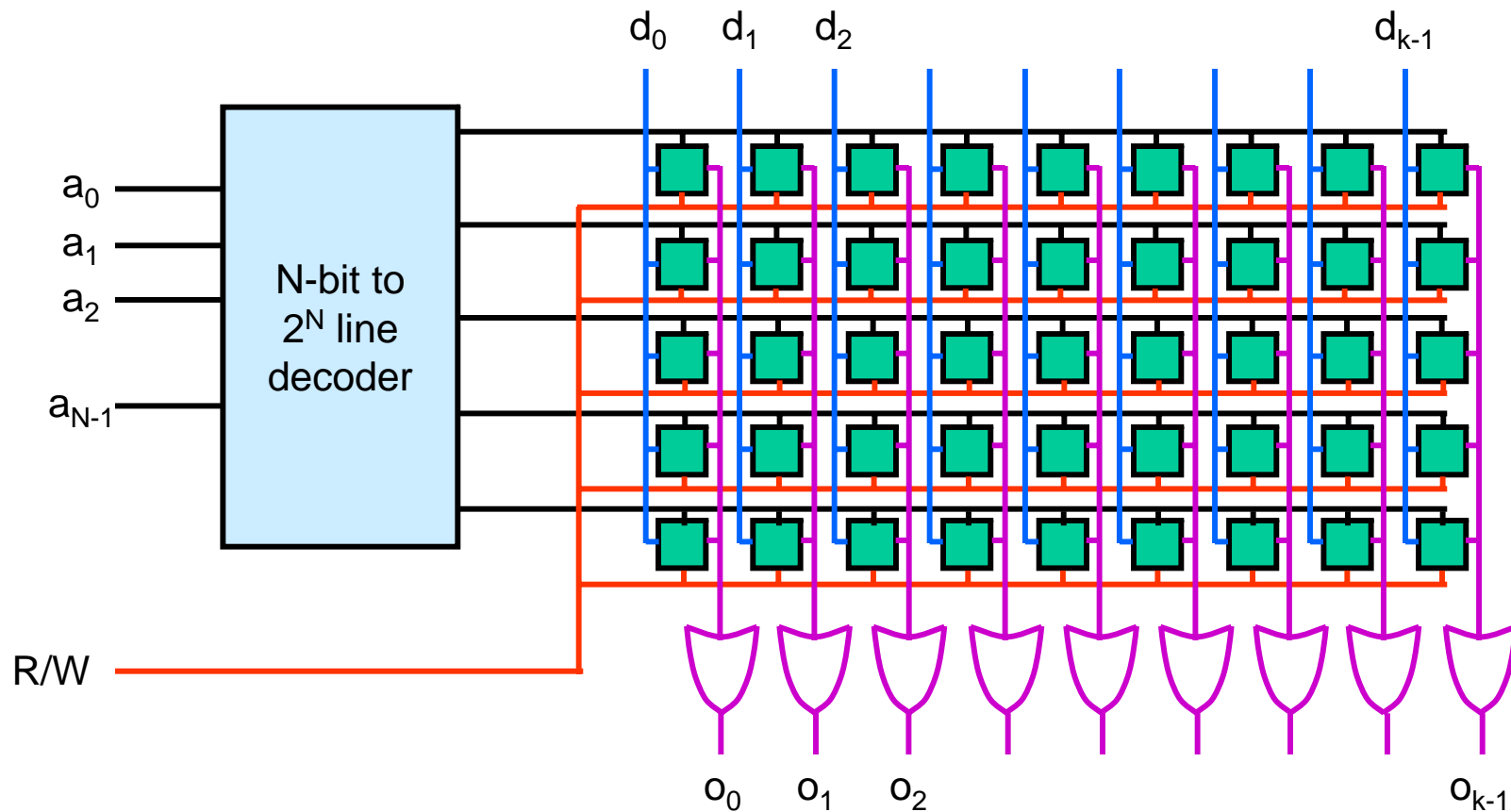
# Sequential Design Testing

Are there any data pattern effects?



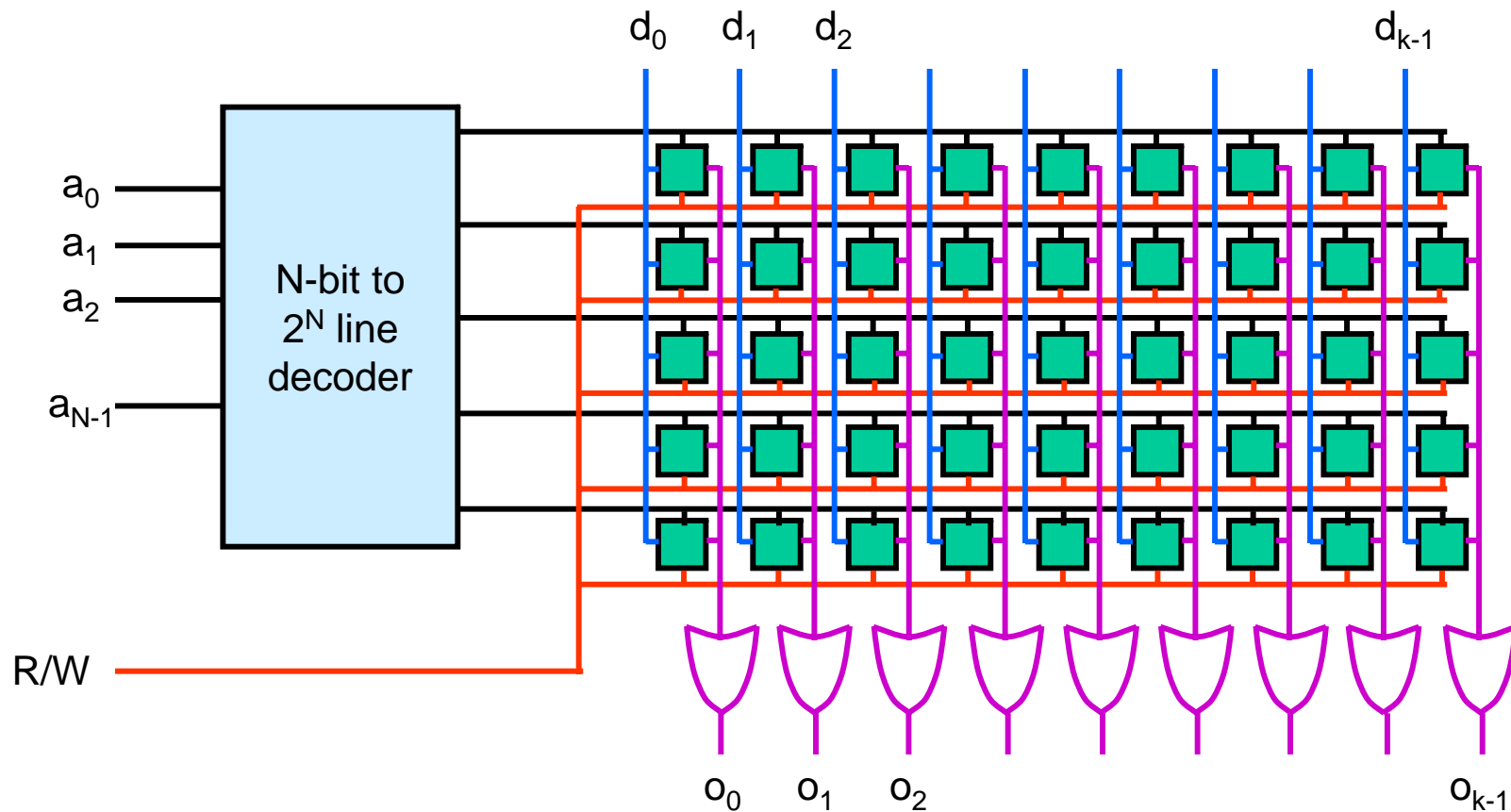
# Sequential Design Testing

How many test conditions are there for a  $2^N$  word x K-bit memory array?



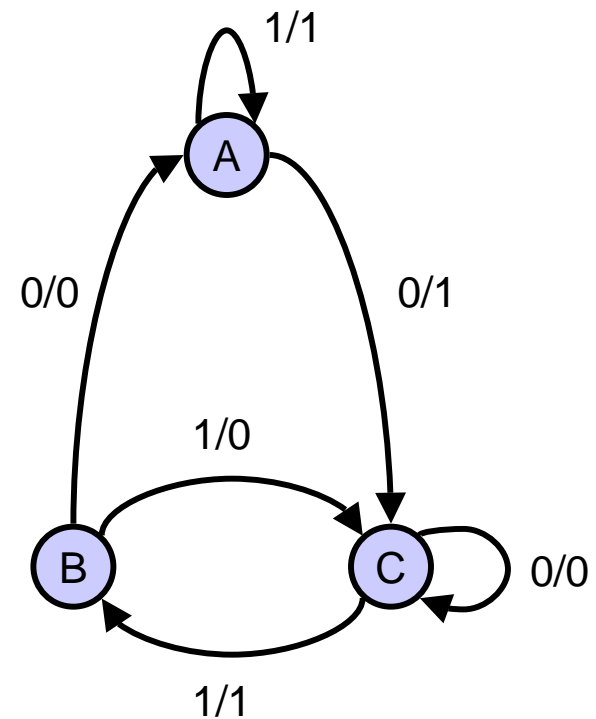
# Sequential Design Testing

How many test conditions are there for a  $2^N$  word x K-bit memory array?  
 $2^{NK}$  states,  $(2^{NK})!$  single state-to-single state transitions



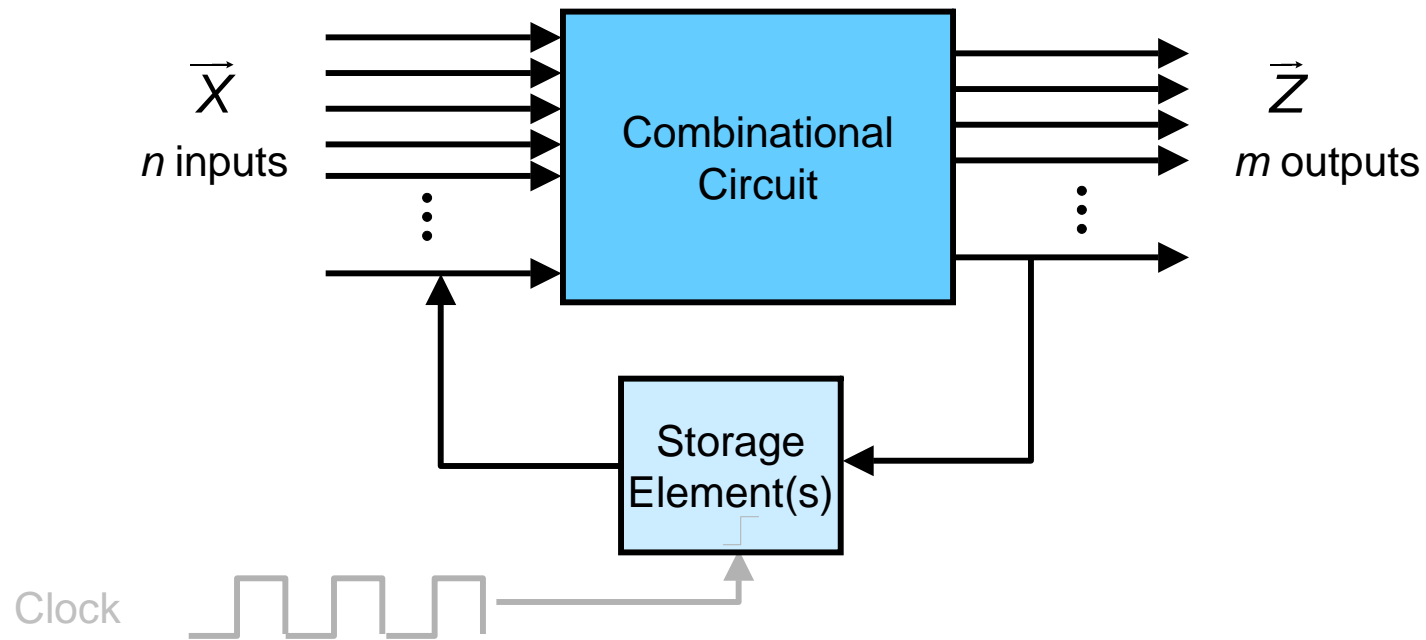
# State Diagram/State Table

Present State	Next State/Output (input 0)	Next State/Output (input 1)
A	C/1	A/1
B	A/0	C/0
C	C/0	B/1

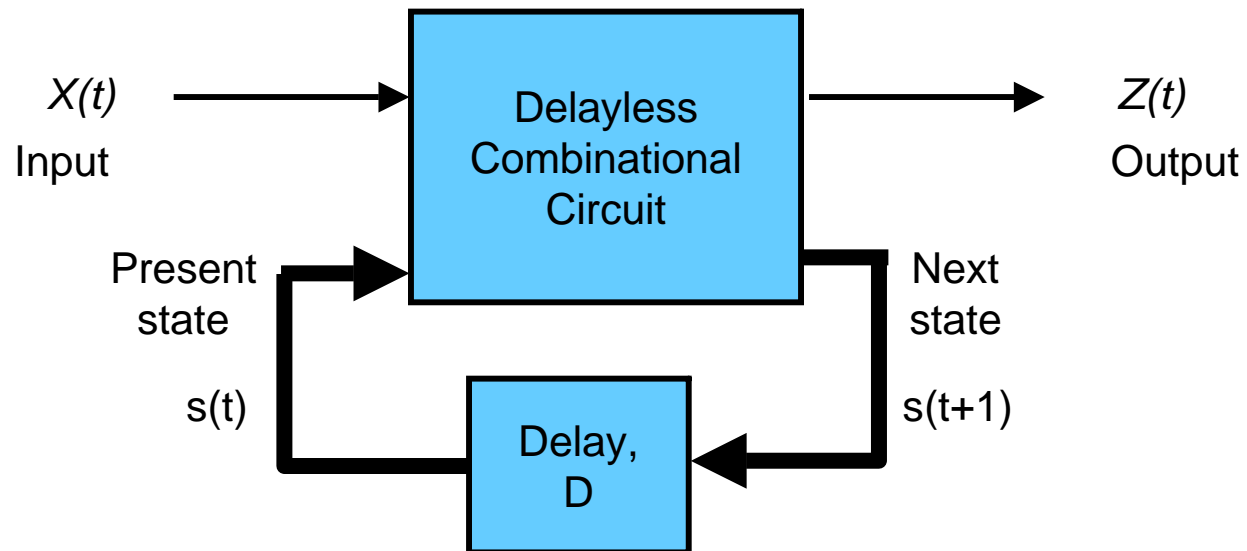


- Reference for today's material: Hennie, Finite State Models for Logical Machines, Wiley, 1968

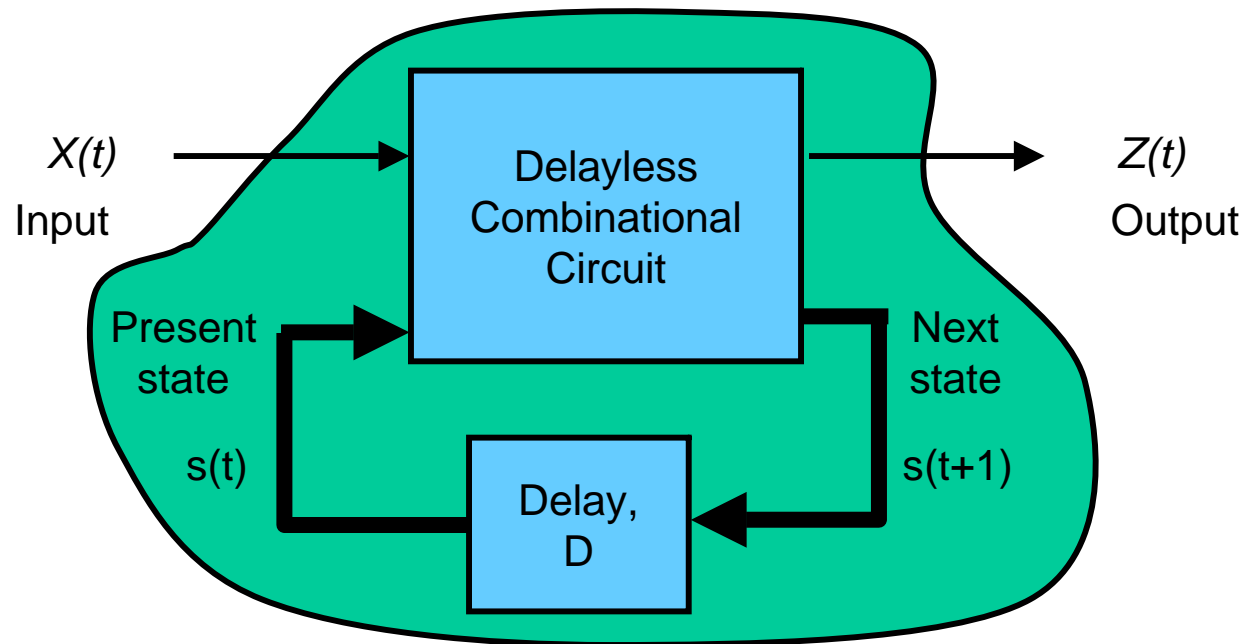
# Finite State Machine Model



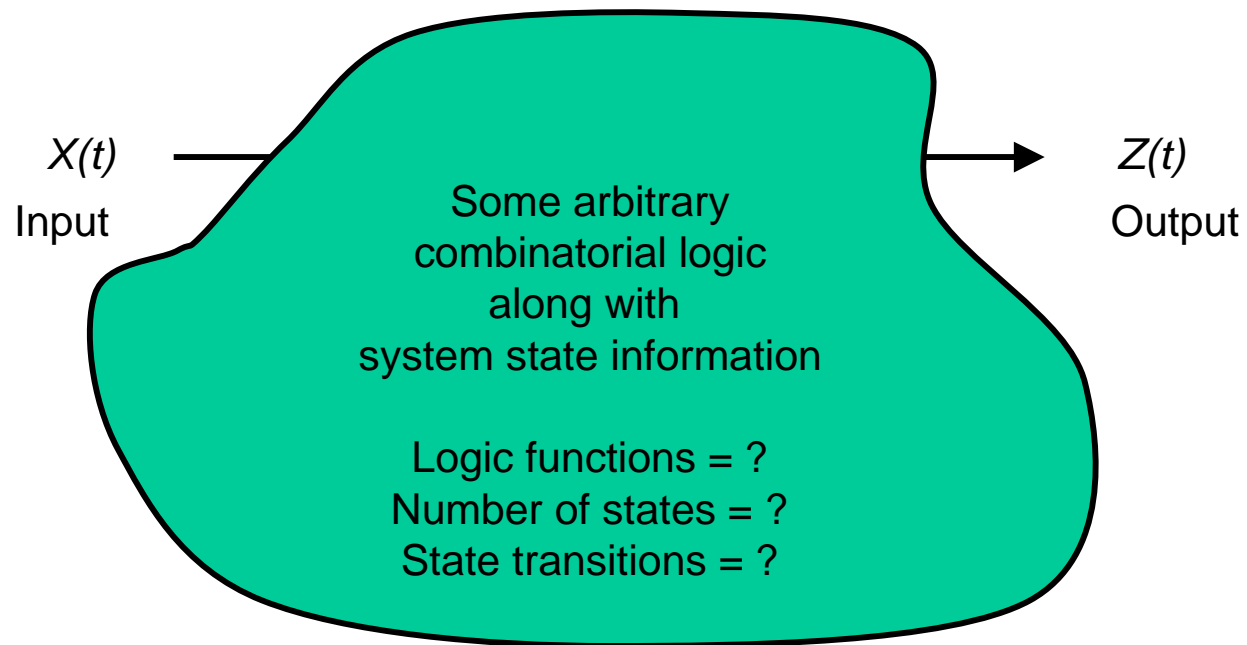
# Finite State Machine Model



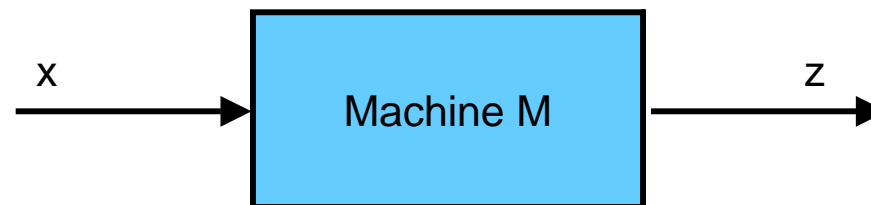
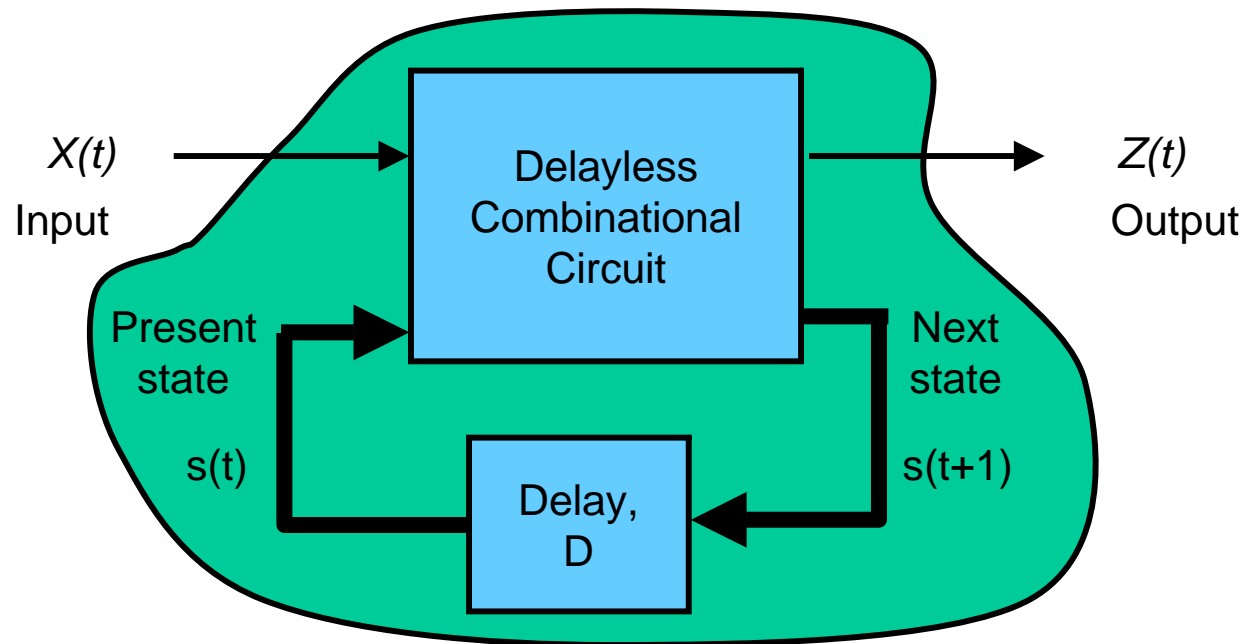
# Finite State Machine Model



# Finite State Machine Model

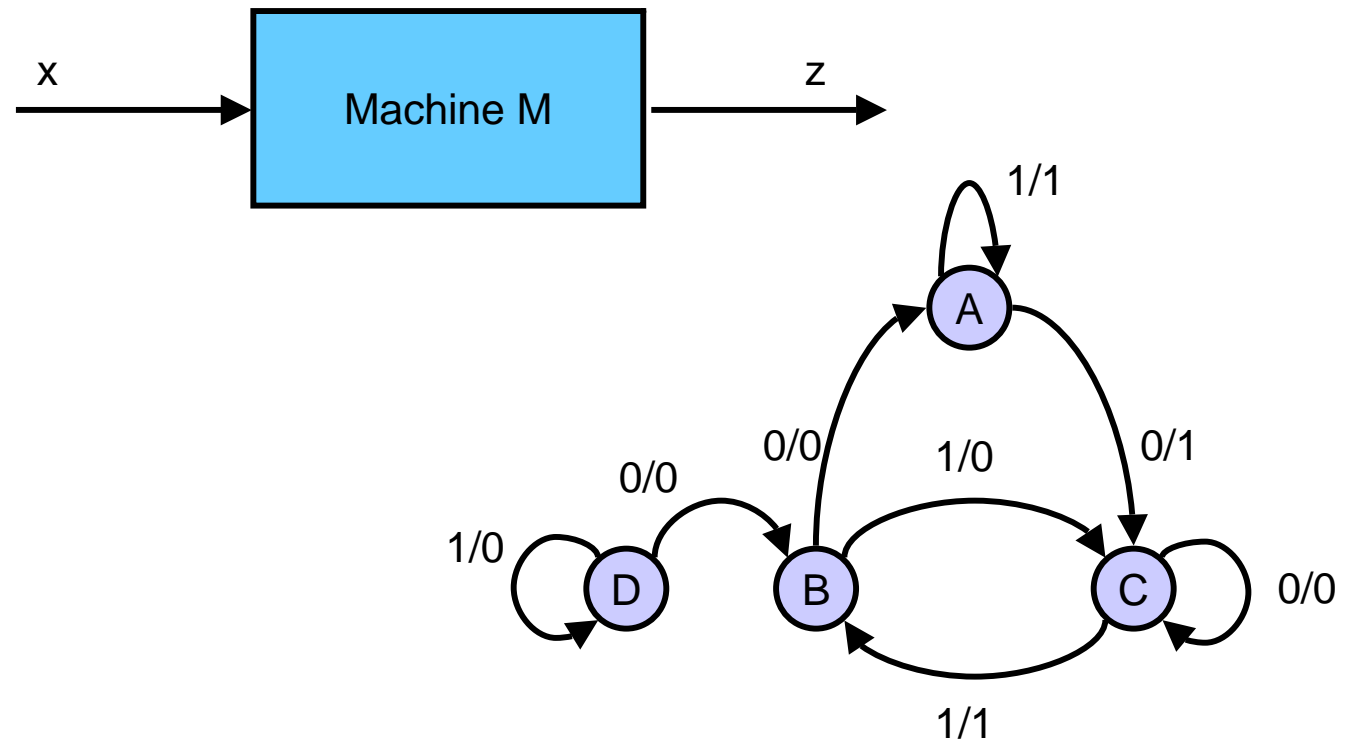


# Finite State Machine Model



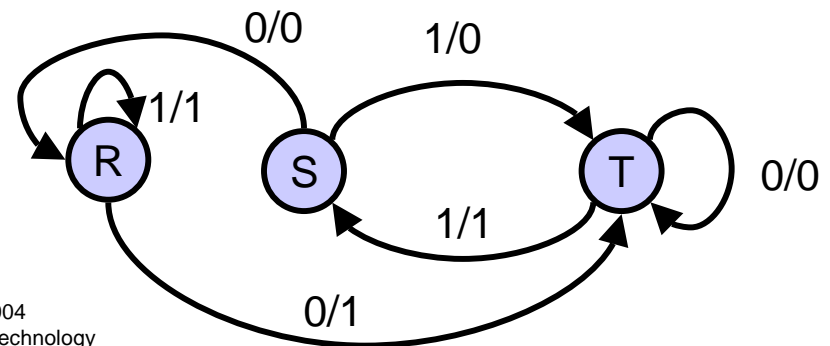
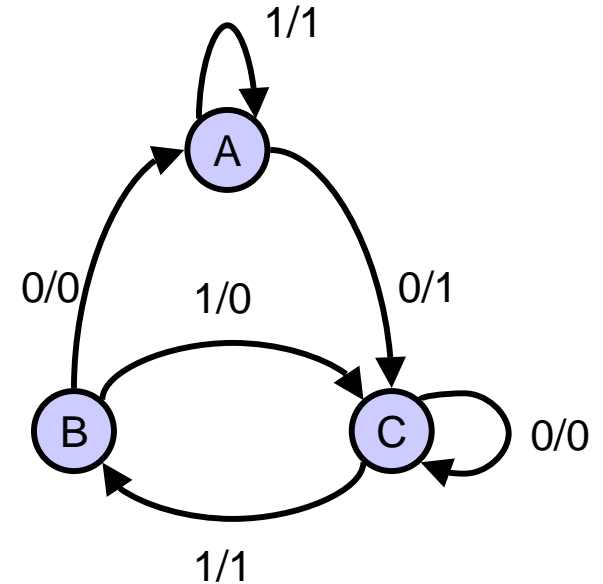
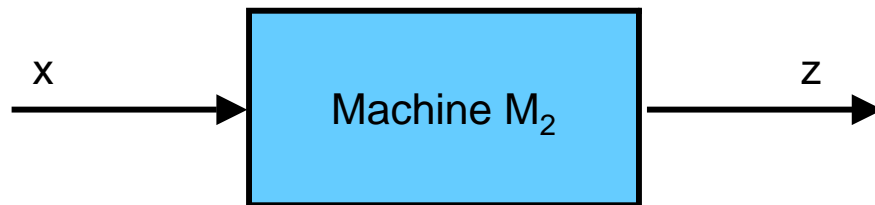
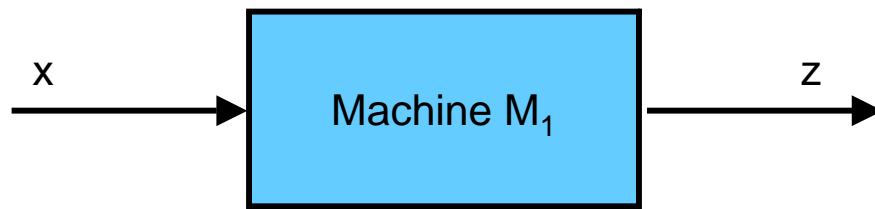
# Fundamental Properties of FSMs

- Accessibility
  - Is state S accessible?

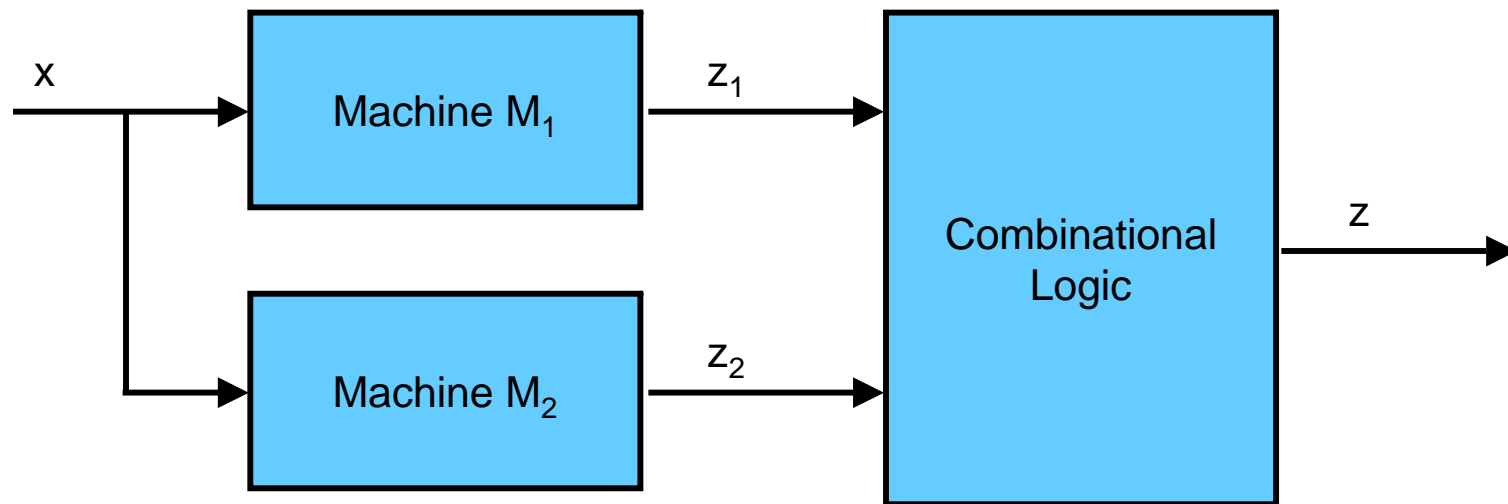


# Fundamental Properties of FSMs

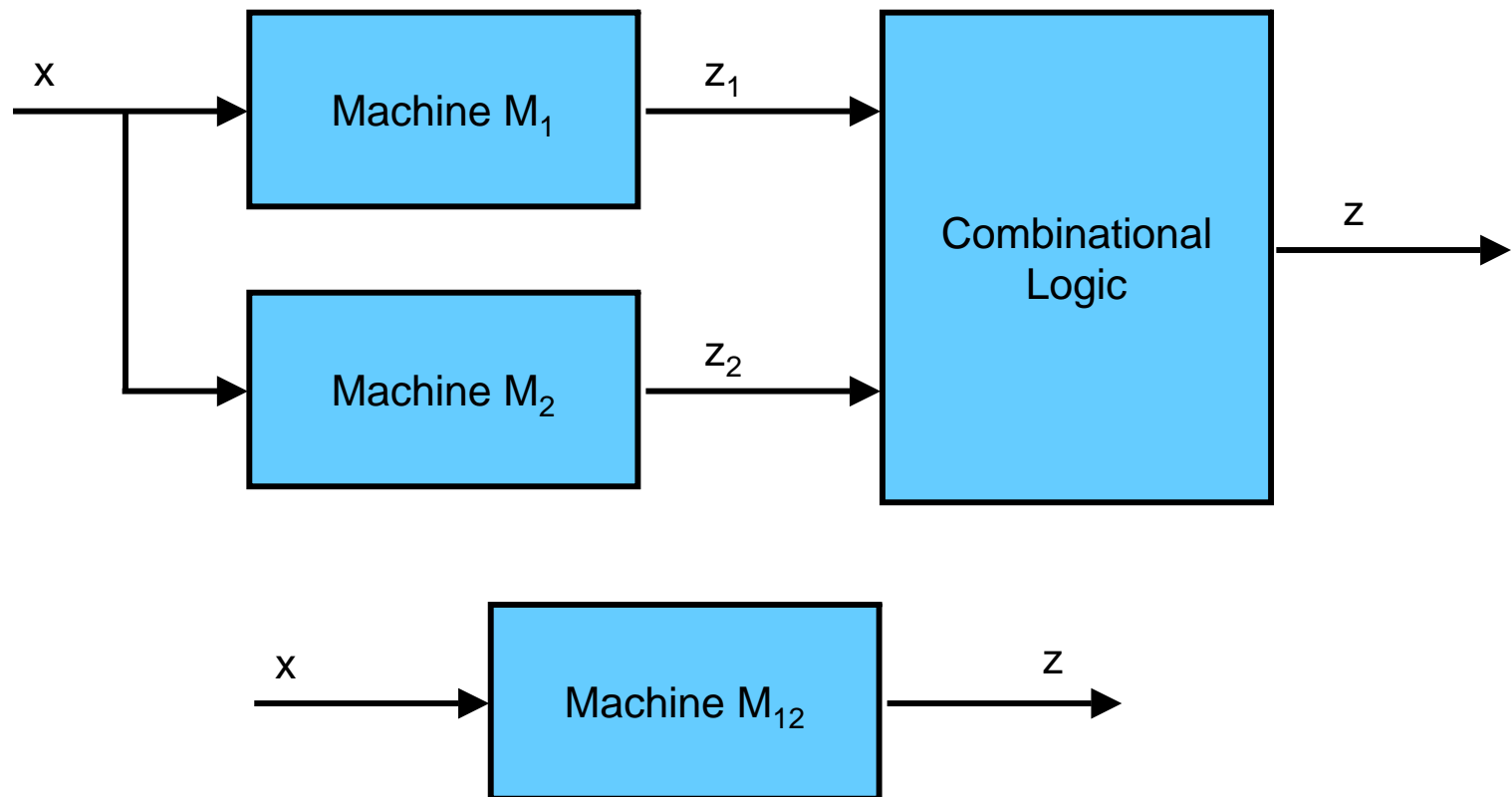
- Distinguishability
  - Can  $M_1$  be distinguished from  $M_2$ ?



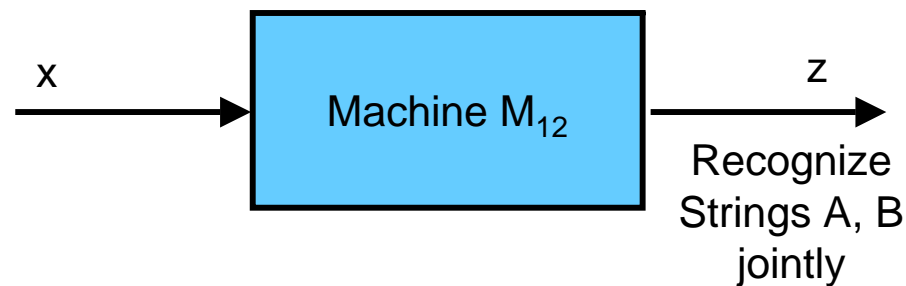
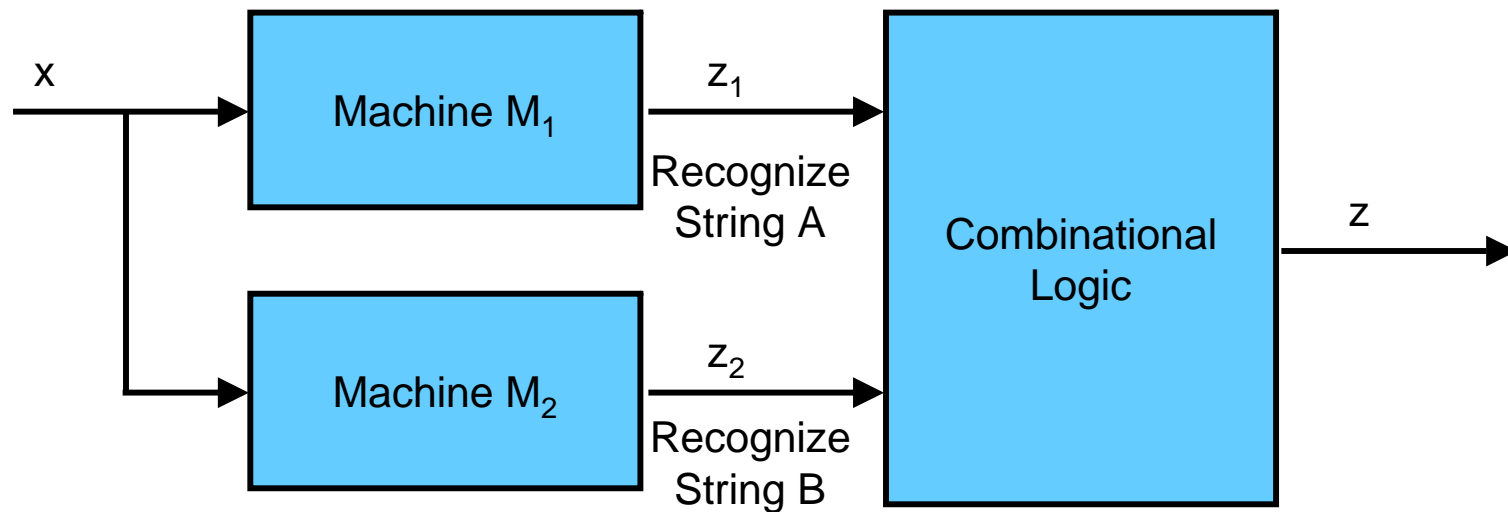
# Combinations of Machines



# Combinations of Machines



# Combinations of Machines



# Combinations of Machines

- Recognize strings that have an odd number of 1s *and* contain a block of four contiguous 1s

# Combinations of Machines

- Recognize strings that have an odd number of 1s *and* contain a block of four contiguous 1s

State	0	1
A	A 0	B 1
B	B 1	A 0

# Combinations of Machines

- Recognize strings that have an odd number of 1s *and* contain a block of four contiguous 1s

State	0	1
A	A 0	B 1
B	B 1	A 0

State	0	1
A	A 0	B 0
B	A 0	C 0
C	A 0	D 0
D	A 0	E 1
E	E 1	E 1

# Combinations of Machines

- Recognize strings that have an odd number of 1s **and** contain a block of four contiguous 1s

State	0	1
A	A 0	B 1
B	B 1	A 0

State	0	1
A	A 0	B 0
B	A 0	C 0
C	A 0	D 0
D	A 0	E 1
E	E 1	E 1

State	0	1
AA	AA 0	BB 0
BB	BA 0	AC 0
BA	BA 0	AB 0
AC	AA 0	BD 0
AB	AA 0	BC 0
BD	BA 0	AE 0
BC	BA 0	AD 0
AE	AE 0	BE 1
AD	AA 0	BE 1
BE	BE 1	AE 0

# Combinations of Machines

- Recognize strings that have an odd number of 1s *and* contain a block of four contiguous 1s

State	0	1
A	A 0	B 1
B	B 1	A 0

State	0	1
A	A 0	B 0
B	A 0	C 0
C	A 0	D 0
D	A 0	E 1
E	E 1	E 1

State	0	1
AA	AA 0	BB 0
BB	BA 0	AC 0
BA	BA 0	AB 0
AC	AA 0	BD 0
AB	AA 0	BC 0
BD	BA 0	AE 0
BC	BA 0	AD 0
AE	AE 0	BE 1
AD	AA 0	BE 1
BE	BE 1	AE 0

Machines designed as compositions of two or more machines may be easier to design and maintain

# Capabilities of FSMs

- Transform an input signal into an output pattern
  - An arbitrarily long string of periodic inputs must eventually produce a periodic output
  - Systems that do not have this property cannot be built using FSMs
  
- Recognize the occurrence of a specific input signal or pattern
  - Intermediate outputs are not important, only the final output

# Capabilities of FSMs

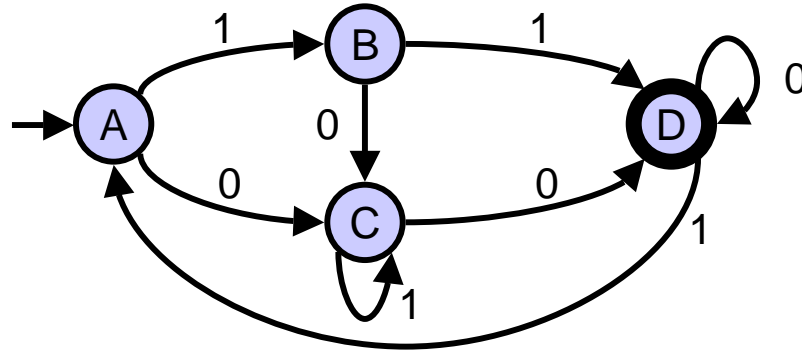
- Transform an input signal into an output pattern
  - An arbitrarily long string of periodic inputs must eventually produce a periodic output
  - Systems that do not have this property cannot be built using FSMs

**WHY?**

- Recognize the occurrence of a specific input signal or pattern
  - Intermediate outputs are not important, only the final output

# Deterministic vs. Nondeterministic FSMs

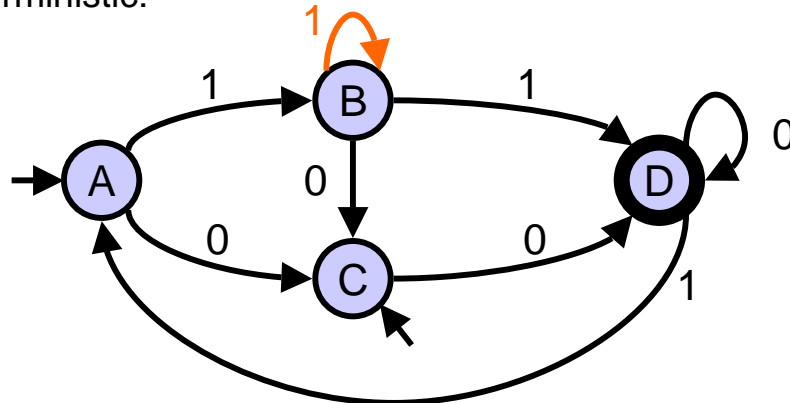
- Deterministic:



State	0	1	Out
A	C	B	0
B	C	D	0
C	D	C	0
D	D	A	1

- Known starting state, unique successor states

- Nondeterministic:



State	0	1	Out
A	C	B	0
B	C	<b>B,D</b>	0
C	D	-	0
D	D	A	1

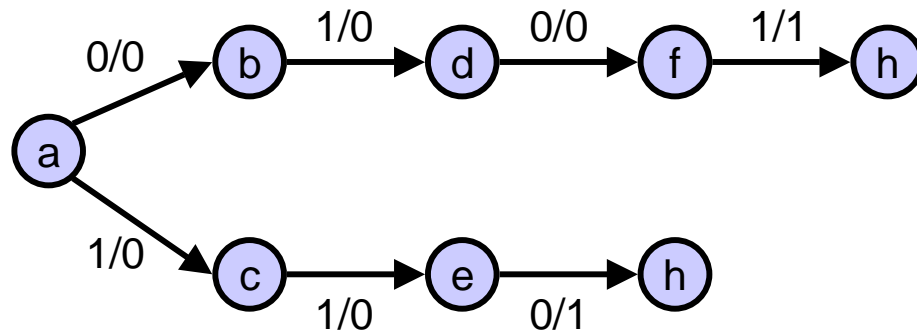
- Unknown/nonunique starting state, nonunique successor states, incomplete specification of machine

# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a deterministic machine

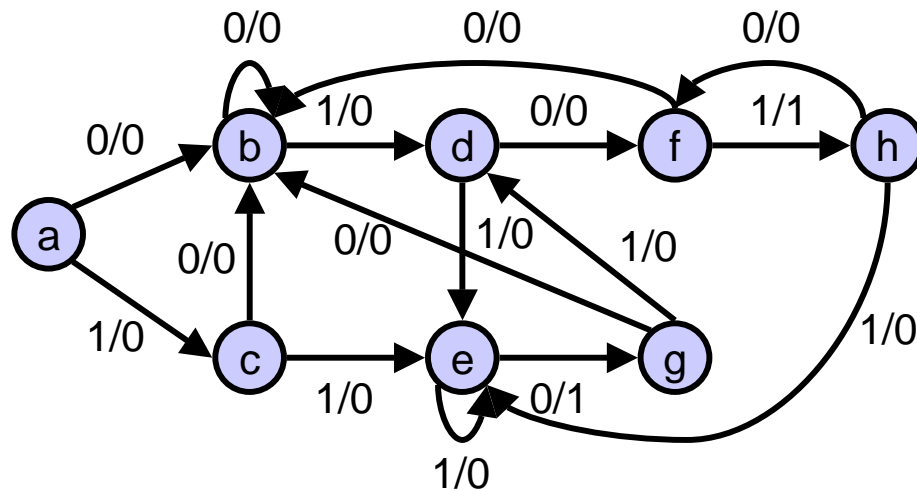
# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a deterministic machine
- Generate basic sequences to be recognized – easy to do



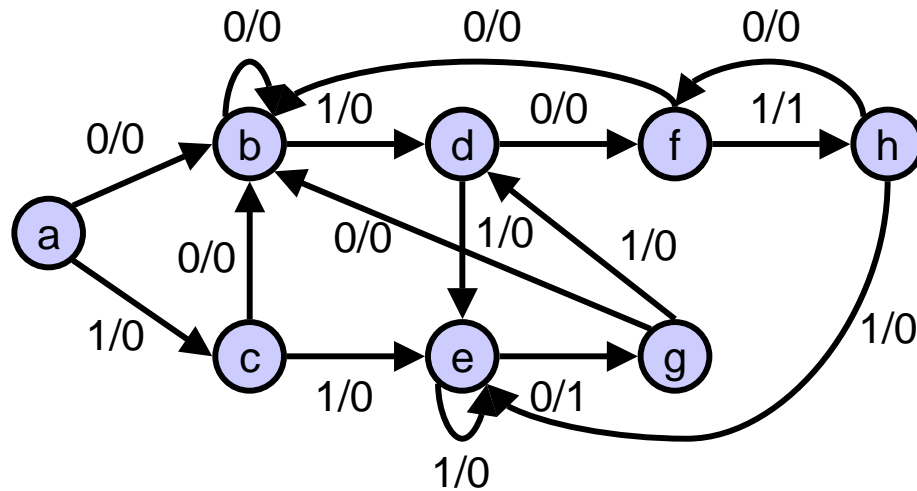
# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a deterministic machine
- Generate basic sequences to be recognized – easy to do
- Add all other transitions – not so simple



# Use for Nondeterministic Machines

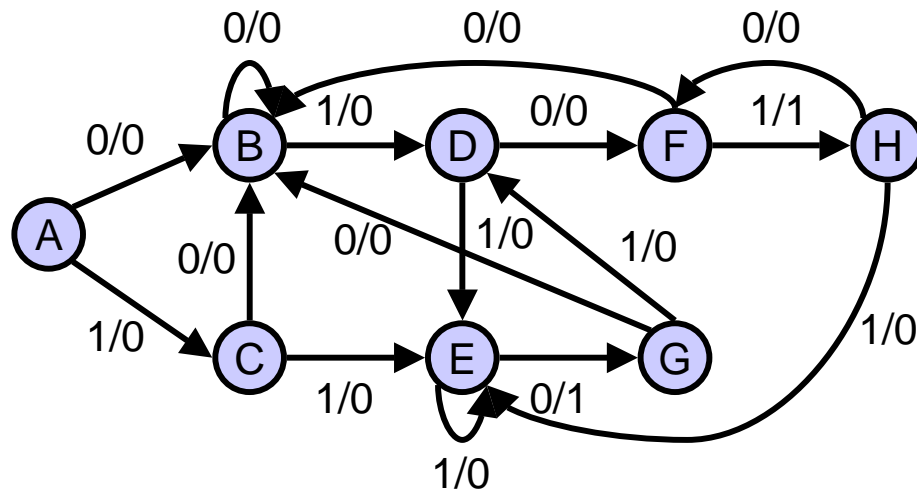
- Recognize a sequence that ends with 0101 or 110 using a deterministic machine
- Generate basic sequences to be recognized – easy to do
- Add all other transitions – not so simple
- Derive state table – straightforward



State	0	1
a	b 0	c 0
b	b 0	d 0
c	b 0	e 0
d	f 0	e 1
e	g 1	e 0
f	b 0	h 1
g	b 0	d 0
h	f 0	e 0

# Use for Nondeterministic Machines

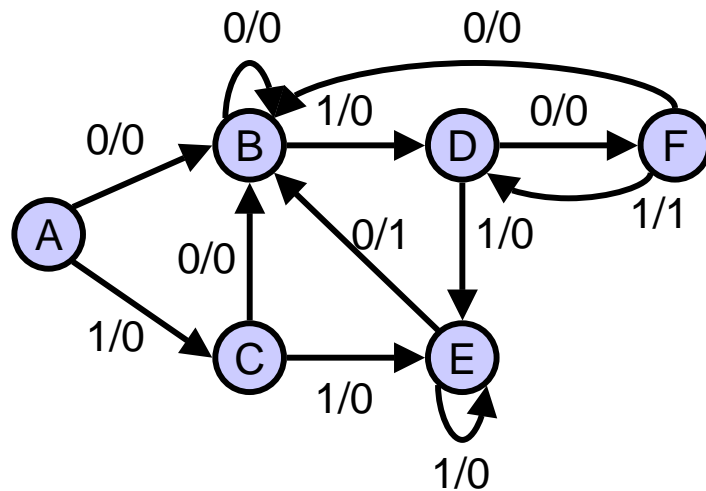
- Recognize a sequence that ends with 0101 or 110 using a deterministic machine
- Generate basic sequences to be recognized – easy to do
- Add all other transitions – not so simple
- Derive state table – straightforward
- Eliminate equivalent states



State	0	1
a	b 0	c 0
b	b 0	d 0
c	b 0	e 0
d	f 0	e 1
e	g 1	e 0
f	b 0	h 1
g	b 0	d 0
h	f 0	e 0

# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a deterministic machine
- Generate basic sequences to be recognized – easy to do
- Add all other transitions – not so simple
- Derive state table – straightforward
- Eliminate equivalent states



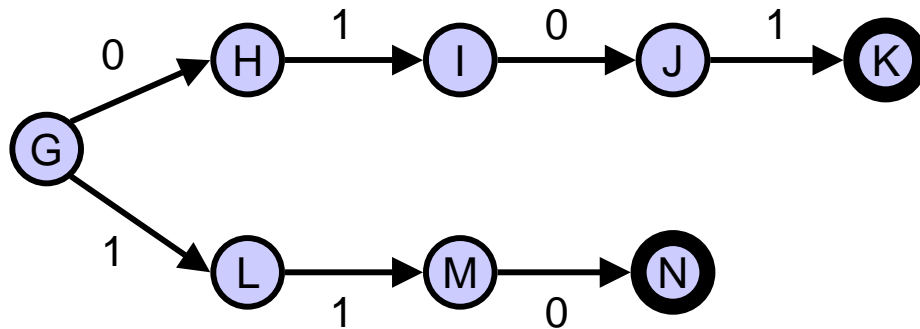
State	0	1
a	b 0	c 0
b	b 0	d 0
c	b 0	e 0
d	f 0	e 1
e	b 1	e 0
f	b 0	d 1

# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine

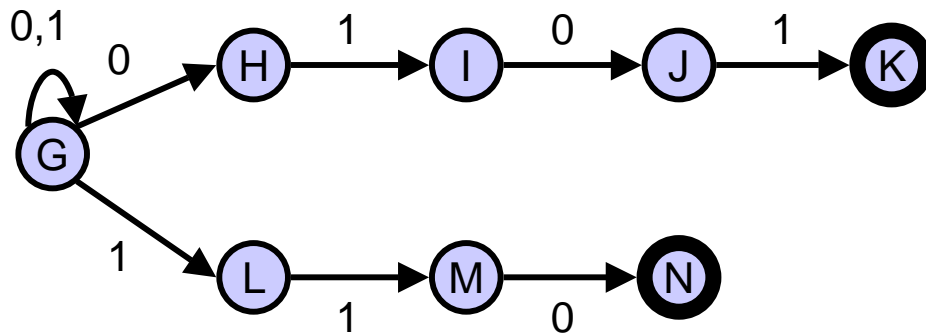
# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do



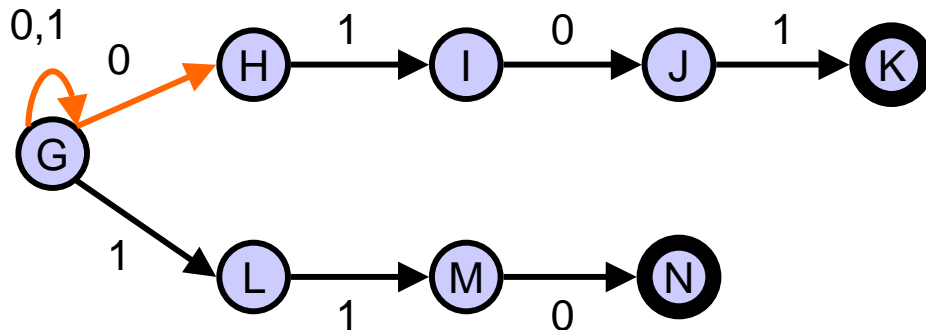
# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do



# Use for Nondeterministic Machines

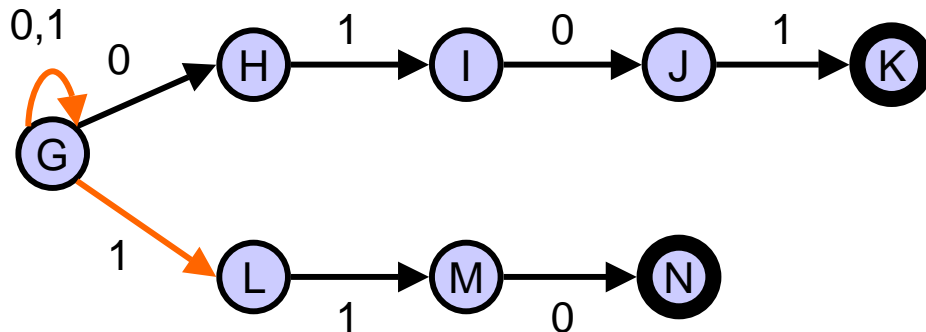
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system - mechanical



State	0	1	Out
G	<b>GH</b>		0

# Use for Nondeterministic Machines

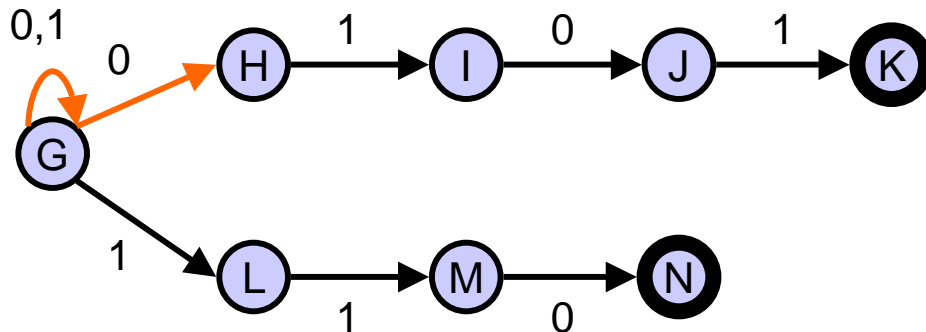
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system - mechanical



State	0	1	Out
G	GH	GL	0

# Use for Nondeterministic Machines

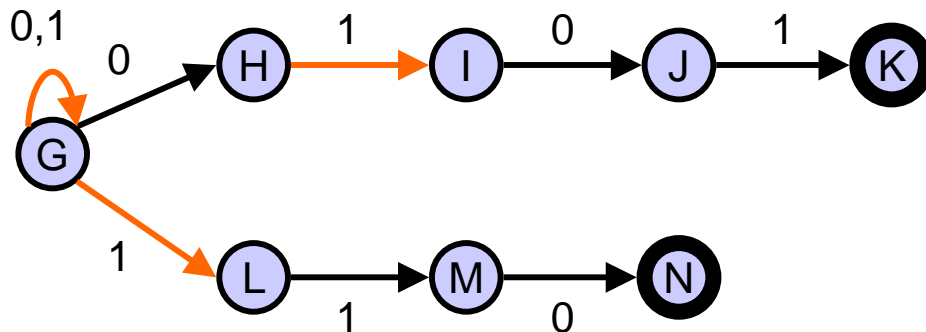
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system - mechanical



State	0	1	Out
G	GH	GL	0
GH	<b>GH</b>		0

# Use for Nondeterministic Machines

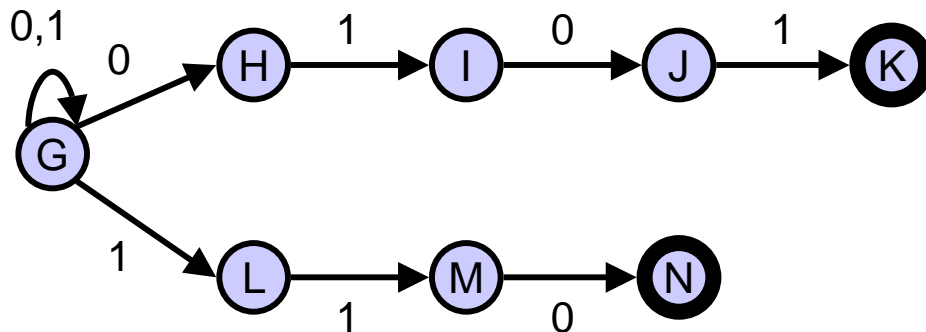
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system - mechanical



State	0	1	Out
G	GH	GL	0
GH	GH	<b>GIL</b>	0

# Use for Nondeterministic Machines

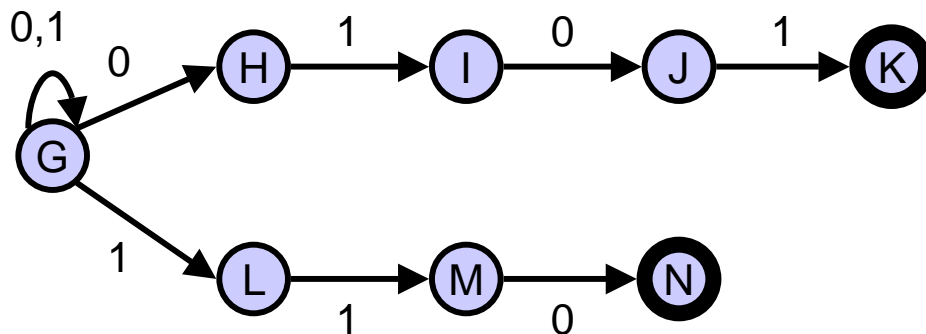
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system - mechanical



State	0	1	Out
G	GH	GL	0
GH	GH	GIL	0
GL	GH	GLM	0
GIL	GHJ	GLM	0
GLM	GHN	GLM	0
GHJ	GH	GILK	0
GHN	GH	GIL	1
GILK	GHJ	GLM	1

# Use for Nondeterministic Machines

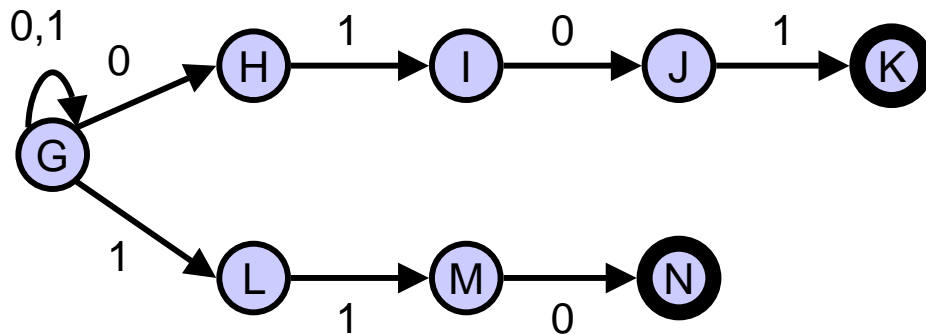
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system – mechanical
- Eliminate redundant states – mechanical



State	0	1	Out
G	GH	GL	0
GH	GH	GIL	0
GL	GH	GLM	0
GIL	GHJ	GLM	0
GLM	GHN	GLM	0
GHJ	GH	GILK	0
GHN	GH	GIL	1
GILK	GHJ	GLM	1

# Use for Nondeterministic Machines

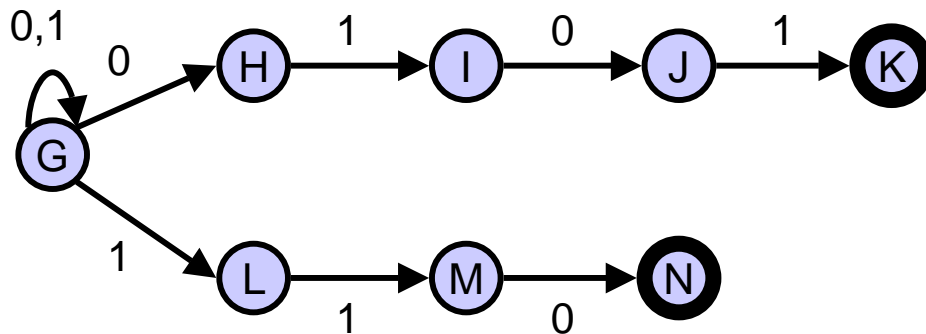
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system – mechanical
- Eliminate redundant states – mechanical



State	0	1	Out
G	GH	GL	0
GH	GH	GIL	0
GL	GH	GLM	0
GIL	GHJ	GLM	0
GLM	<b>GH</b>	GLM	0
GHJ	GH	<b>GIL</b>	0

# Use for Nondeterministic Machines

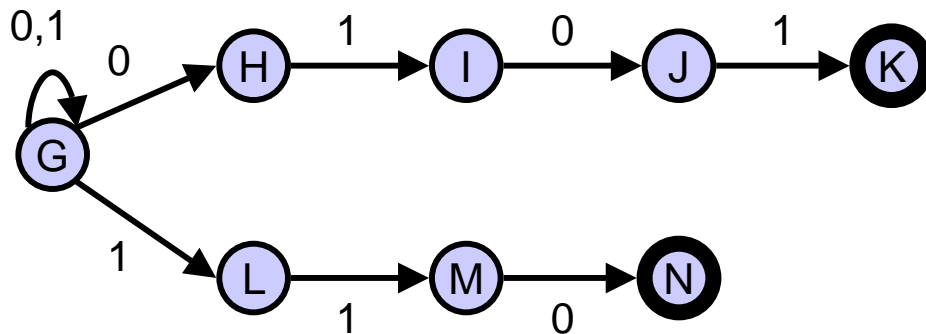
- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system – mechanical
- Eliminate redundant states – mechanical
- Convert from Moore to Mealy machine, rename states



State	0	1
A (G)	B 0	C 0
B (GH, <b>GHN</b> )	B 0	D 0
C (GL)	B 0	E 0
D ( <b>GILK</b> , GIL)	F 0	E 0
E (GLM)	<b>B 1</b>	E 0
F (GHJ)	B 0	<b>D 1</b>

# Use for Nondeterministic Machines

- Recognize a sequence that ends with 0101 or 110 using a nondeterministic machine
- Generate basic sequences to be recognized – easy to do
- Add nondeterministic transitions – easy to do
- Convert to deterministic system – mechanical
- Eliminate redundant states – mechanical
- Convert from Moore to Mealy machine, rename states



- Results in equivalent machine with less effort

State	0	1
A (G)	B 0	C 0
B (GH, <b>GHN</b> )	B 0	D 0
C (GL)	B 0	E 0
D ( <b>GILK</b> , GIL)	F 0	E 0
E (GLM)	<b>B 1</b>	E 0
F (GHJ)	B 0	<b>D 1</b>

State	0	1
a	b 0	c 0
b	b 0	d 0
c	b 0	e 0
d	f 0	e 1
e	b 1	e 0
f	b 0	d 1

# Experiments on FSMs

## Synchronizing Sequences

- Consider the machine with the state table:

State	0	1
A	B 0	C 0
B	B 1	D 0
C	A 0	E 1
D	C 0	E 0
E	C 1	C 0

- What is the final state if the input is 000?

# Experiments on FSMs

## Synchronizing Sequences

- Consider the machine with the state table:

State	0	1
A	B 0	C 0
B	B 1	D 0
C	A 0	E 1
D	C 0	E 0
E	C 1	C 0

- What is the final state if the input is 000 for each initial state?
  - A: A – B – B

# Experiments on FSMs

## Synchronizing Sequences

- Consider the machine with the state table:

State	0	1
A	B 0	C 0
B	B 1	D 0
C	A 0	E 1
D	C 0	E 0
E	C 1	C 0

- What is the final state if the input is 000 for each initial state?
  - A: A  $\rightarrow$  B  $\rightarrow$  B  $\rightarrow$  B
  - B: B  $\rightarrow$  B  $\rightarrow$  B  $\rightarrow$  B
  - C: C  $\rightarrow$  A  $\rightarrow$  B  $\rightarrow$  B
  - D: D  $\rightarrow$  C  $\rightarrow$  A  $\rightarrow$  B
  - E: E  $\rightarrow$  C  $\rightarrow$  A  $\rightarrow$  B

# Experiments on FSMs

## Synchronizing Sequences

- Consider the machine with the state table:

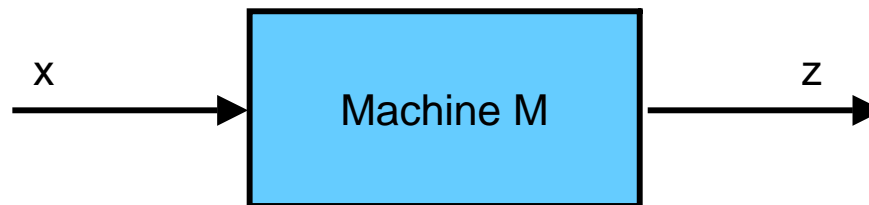
State	0	1
A	B 0	C 0
B	B 1	D 0
C	A 0	E 1
D	C 0	E 0
E	C 1	C 0

- What is the final state if the input is 000 for each initial state?
  - A: A → B → B → B
  - B: B → B → B → B
  - C: C → A → B → B
  - D: D → C → A → B
  - E: E → C → A → B
- 000 is a synchronizing sequence for this machine, always leading it to state B

# Experiments on FSMs

## Homing Sequences

- Given a user selected input sequence  $X_i$ , observing only  $Z_i$ , can the *final* state of  $M$  be determined?
- If  $X_i$  exists, it is a homing sequence

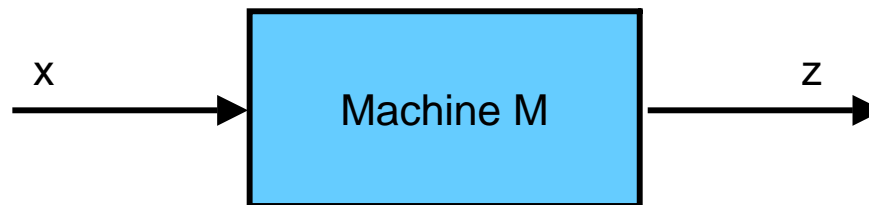


- Homing sequences can be used to test the behavior of  $M$

# Experiments on FSMs

## Distinguishing Sequences

- Given a user selected input sequence  $X_i$ , observing only  $Z_i$ , can the *initial* state of  $M$  be determined?
- If  $X_i$  exists, it is a distinguishing sequence

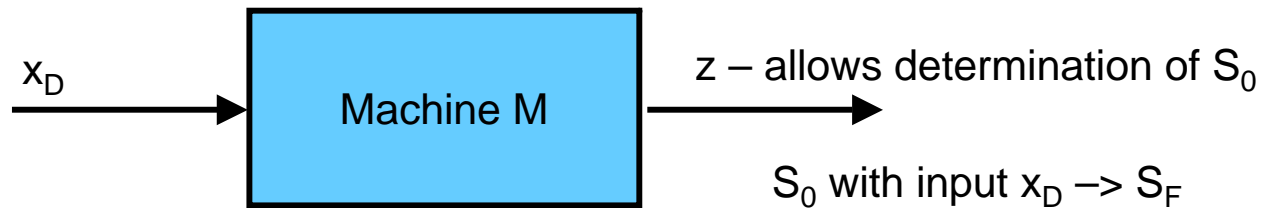


- Distinguishing sequences can be used to test the behavior of  $M$



# Distinguishing vs. Homing Sequences

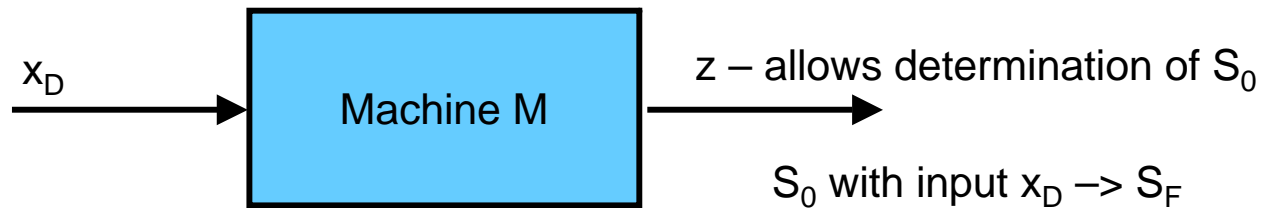
- Every distinguishing sequence is a homing sequence



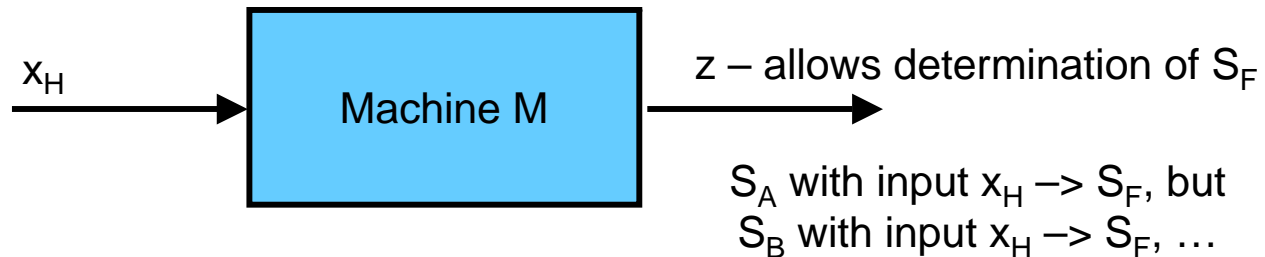
- Not all homing sequences are distinguishing sequences

# Distinguishing vs. Homing Sequences

- Every distinguishing sequence is a homing sequence



- Not all homing sequences are distinguishing sequences



# Distinguishing vs. Homing Sequences

- Consider:

State	0	1
A	B 0	C 0
B	D 0	E 1
C	D 0	A 0
D	E 0	B 0
E	E 1	C 1

# Distinguishing vs. Homing Sequences

- Consider:

State	0	1
A	B 0	C 0
B	D 0	E 1
C	D 0	A 0
D	E 0	B 0
E	E 1	C 1

Initial State	Output for input 11001	Final State
A	0 0 0 0 0	B
B	1 1 0 0 1	C
C	0 0 0 0 1	C
D	0 1 1 1 1	C
E	1 0 0 0 0	B

Each output is different,  
depending on the initial state.  
11001 is a distinguishing sequence.  
It is also a homing sequence – final state is known

# Distinguishing vs. Homing Sequences

- Consider:

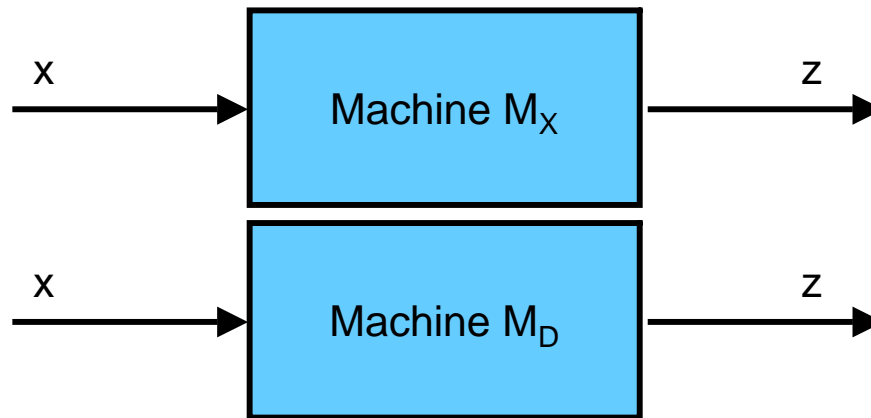
State	0	1
A	B 0	C 0
B	D 0	E 1
C	D 0	A 0
D	E 0	B 0
E	E 1	C 1

Initial State	Output for input 101	Final State
A	0 0 0	B
B	1 1 1	C
C	0 0 1	E
D	0 0 0	B
E	1 0 0	B

Each output is NOT different,  
depending on the initial state (e.g., states A and D).  
101 is a NOT distinguishing sequence.  
But it is a homing sequence – final state is known

# Machine Identification Checking Sequences

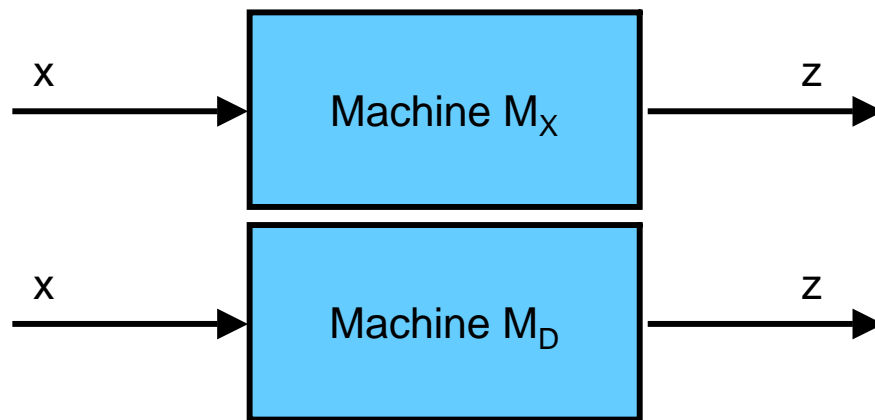
- Are  $M_X$ , a machine as observed, equivalent to  $M_D$ , a machine whose operation is known?
  - Is  $M_X$  operating as designed ( $M_D$ )?



# Machine Identification

## Checking Sequences

- Are  $M_X$ , a machine as observed, equivalent to  $M_D$ , a machine whose operation is known?
  - Is  $M_X$  operating as designed ( $M_D$ )?



- It is not always possible to distinguish two different machines
- The checking sequence can quickly become unmanageable

# Checking Sequences

• Consider:

•  $M_1$ :

State	0	1
A	A 0	B 1
B	B 0	A 0

$M_x$  is either  $M_1$  or  $M_2$ .

•  $M_2$ :

State	0	1
A	B 1	A 0
B	B 0	C 1
C	C 0	B 0

# Checking Sequences

• Consider:

•  $M_1$ :

State	0	1
A	A 0	B 1
B	B 0	A 0

•  $M_2$ :

State	0	1
A	B 1	A 0
B	B 0	C 1
C	C 0	B 0

$M_x$  is either  $M_1$  or  $M_2$ .

Starting state  
↓  
input ↙  
But:  
 $M_1(A,S) = M_2(B,S)$

# Checking Sequences

• Consider:

•  $M_1$ :

State	0	1
A	A 0	B 1
B	B 0	A 0

•  $M_2$ :

State	0	1
A	B 1	A 0
B	B 0	C 1
C	C 0	B 0

$M_x$  is either  $M_1$  or  $M_2$ .

Starting state  
↓  
input ↗  
But:  
 $M_1(A,S) = M_2(B,S)$

Without knowing starting state,  $M_1$  and  $M_2$  cannot be distinguished

# Checking Sequences

- Can the state table of a system be completely determined without internal knowledge of the system?

# Checking Sequences

- Can the state table of a system be completely determined without internal knowledge of the system?

Generally, no:

Any test sequence of length  $N$  that allows determination of a state table of machine  $M_A$  can be recognized by  $M_B$  with more states – it takes an infinitely long sequence to avoid this possibility.

# Summary

- Fundamental concepts of digital systems (Mano Chapter 1)
- Binary codes, number systems, and arithmetic (Ch 1)
- Boolean algebra (Ch 2)
- Simplification of switching equations (Ch 3)
- Digital device characteristics (e.g., TTL, CMOS)/design considerations (Ch 10)
- Combinatoric logical design including LSI implementation (Chapter 4)
- Flip-flops and state memory elements (Ch 5)
- Sequential logic analysis and design (Ch 5)
- Counters, shift register circuits (Ch 6)
- Hazards, Races, and time related issues in digital design (Ch 9)
- Synchronous vs. asynchronous design (Ch 9)
- Memory and Programmable logic (Ch 7)
- **Minimization of sequential systems**
- **Introduction to Finite Automata**

# Homework 16 – due in Class 18

- There is no class 18...