Lecture 10
Finite State Machine

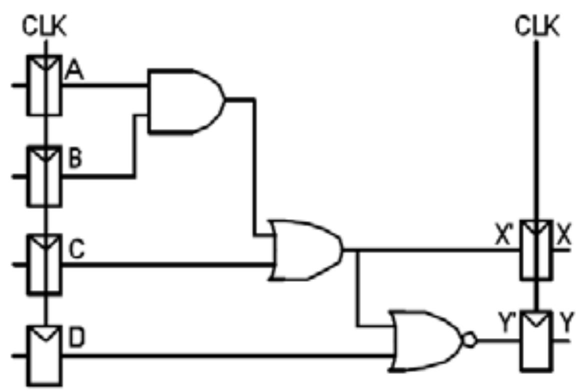
Outline

- Basics of Finite State Machine (***)
- State Reduction (***)
- Textbook chapters: 9.1-9.4

Finite State Machines

- A mathematical representation which can describe sequential logic circuits
- Computation can be considered as a sequence of steps, each step is a state, and the computation is a sequence of steps (states) which the circuit executes (visits)

- At any given point in time we can freeze the execution and associate the 'state' of the circuit with the content of the memory elements
  - [A,B,C,D,X,Y] can be thought of as a binary code (of 6 bits) that uniquely identifies a state
  - This circuit can possible exhibit one of 2^6 states, i.e., a finite number of states can be encoded



Can have 64 steps in the calculation max.

Building Finite State Machines

- We will first review an example

Example: Sequence Detector

Assert output whenever a sequence of [1 0] is detected in the input stream
Otherwise output is 0

Input	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
Output	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**

Process for creating a circuit for FSM

1. Analyze the verbal problem statement

2. Create example input/output sequences that represent typical scenarios

3. Going through these sequences,

a. Either create a new state and branch out for the corresponding input case

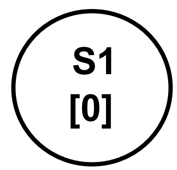
b. Revisit an existing state that covers this input behavior

4. For each state, transitions for all possible input combinations must be accounted for

5. For each possible transition, the output response must be attached

Process for creating a circuit for FSM

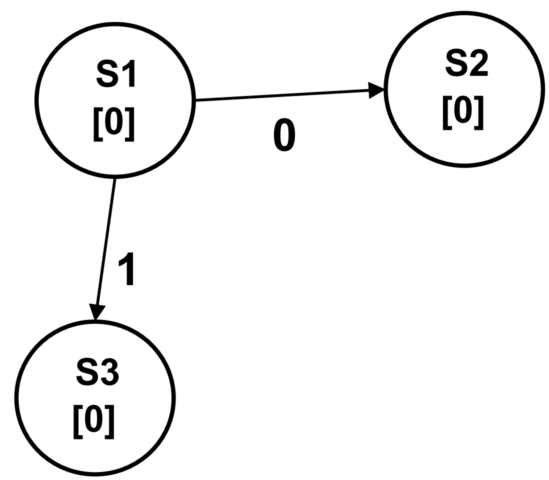
- Always start with the 'initial'/'reset' state
  - How will the circuit react for all input cases starting at this state?



Process for creating a circuit for FSM

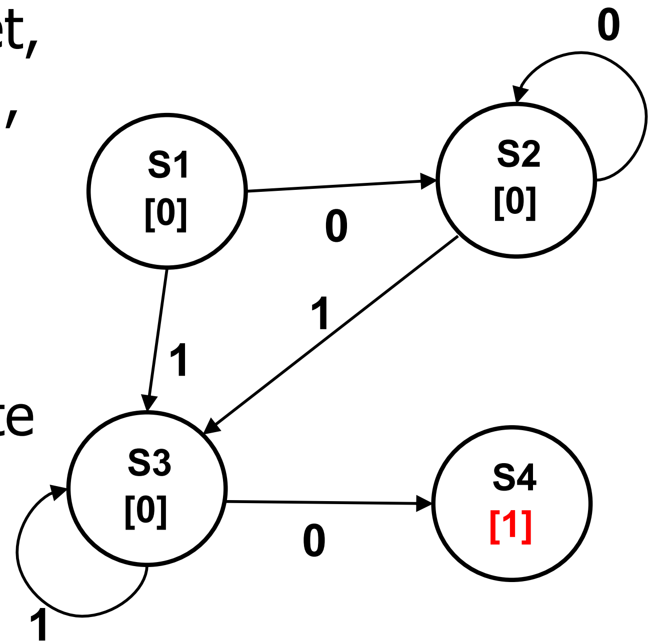
- How will the circuit react for all input cases starting at this state
  - If input is '0' that will be the case of the FSM 'having spotted a 0': a new state S2
  - If input is '1' that will be the case of the FSM having spotted a 1': different that spotting a '0', a new state S3
  - In both of these new states output is still '0' no '10' pattern detected

Circle = output response
Branches = input



Process for creating a circuit for FSM

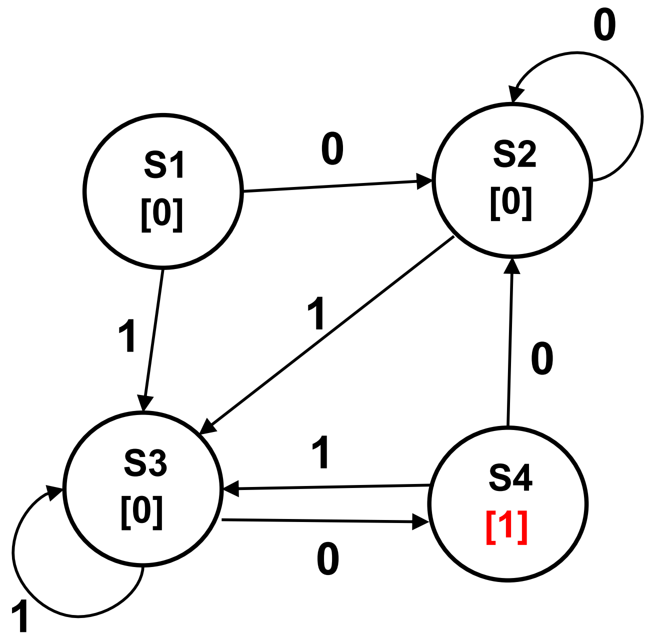
- Cover all outgoing transitions from each new state added
  - @S2:
    - If input is '0' the status of the FSM is no different, self loop back to S2
    - If input is '1' that is the first '1' spotted yet, which is the same situation as being in S3, transition to S3
  - @S3:
    - If input is '0' that is a '10' pattern DETECTED, this is a new event, NEW State S3, with output of new state = '1'
    - If input is '1' the status of the FSM is no different, self loop back to S3



Process for creating a circuit for FSM

- Cover all outgoing transitions from each new state added
  - @S4:
    - If input is '0', this is no different than having seen 'just any sequence' 0s, transition to S2
    - If input is '1', this is no different than having seen 'just any sequence' 1s, transition to S3

State 1: this is a safe way to do initialization



Finite State Machine Basics

- State updated at rising edge of clock
- With n bits, we can realize 2^n states in total
- Two types of FSM: Mealy or Moore State Machine

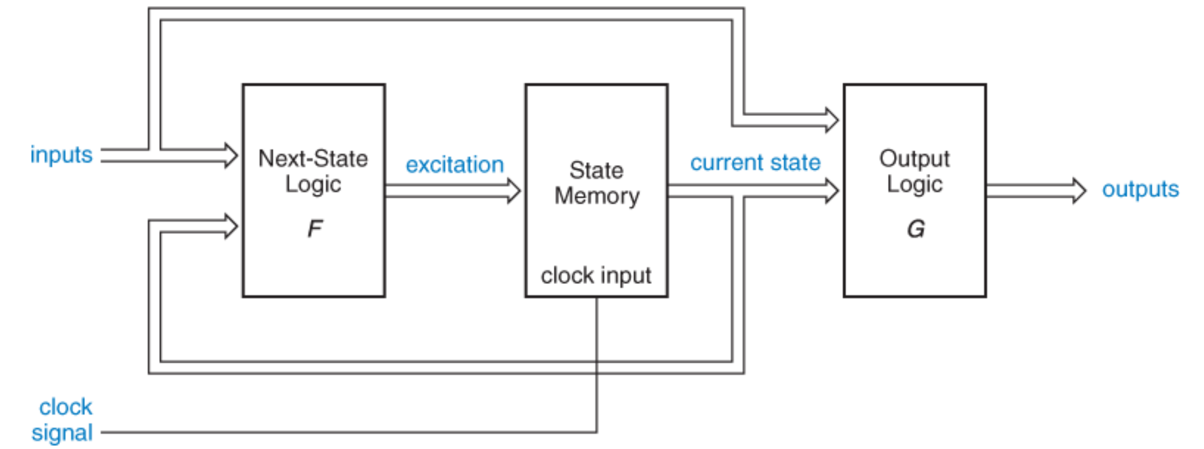
Mealy Machine:

Next state = F (current state , input)
Output = G (current state , input)

Moore Machine:

Next state = F (current state , input)
Output=G(current state)

Moore versus Mealy machines

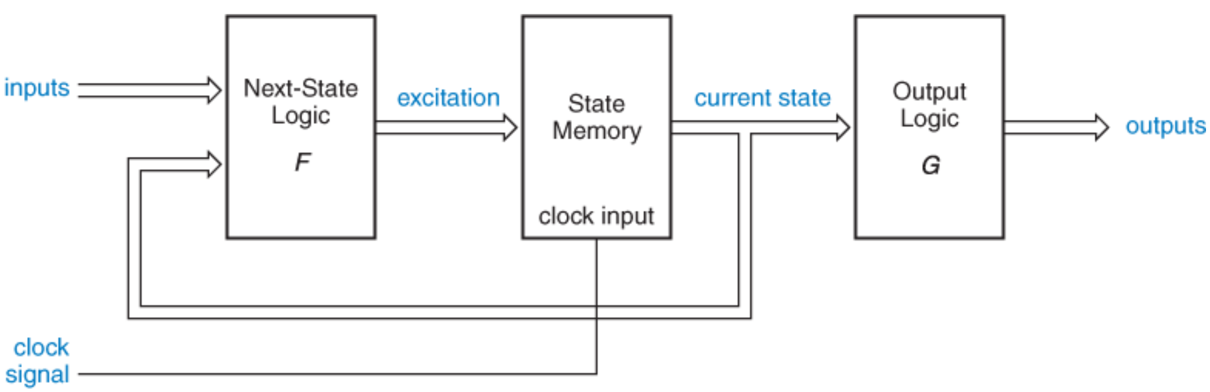


Mealy machine

Outputs depend on state and on inputs

Input changes can cause immediate output changes (**asynchronous**)

Not necessarily follow the clock



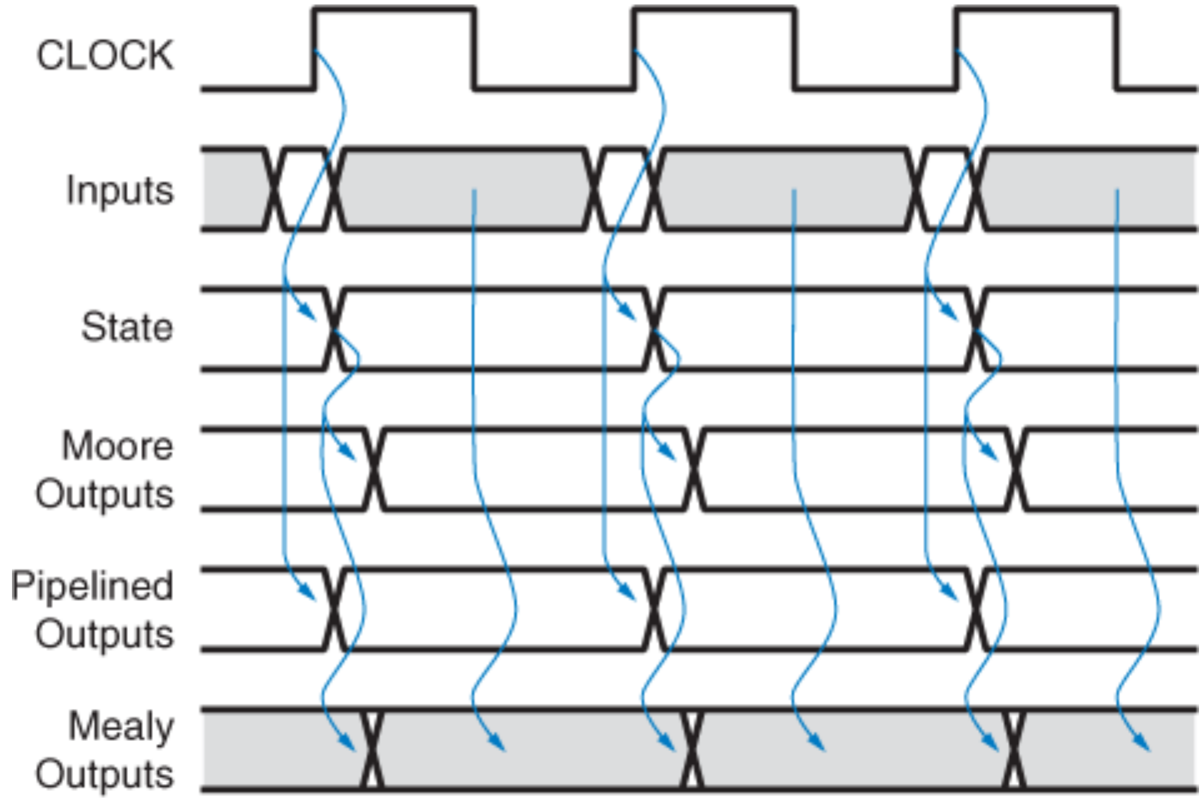
Moore machine

Outputs are a function of current state

Outputs change synchronously with state changes

Let 9 last example is a Moore machine - the result does not explicitly depend on the inputs

Timing Diagram of FSM



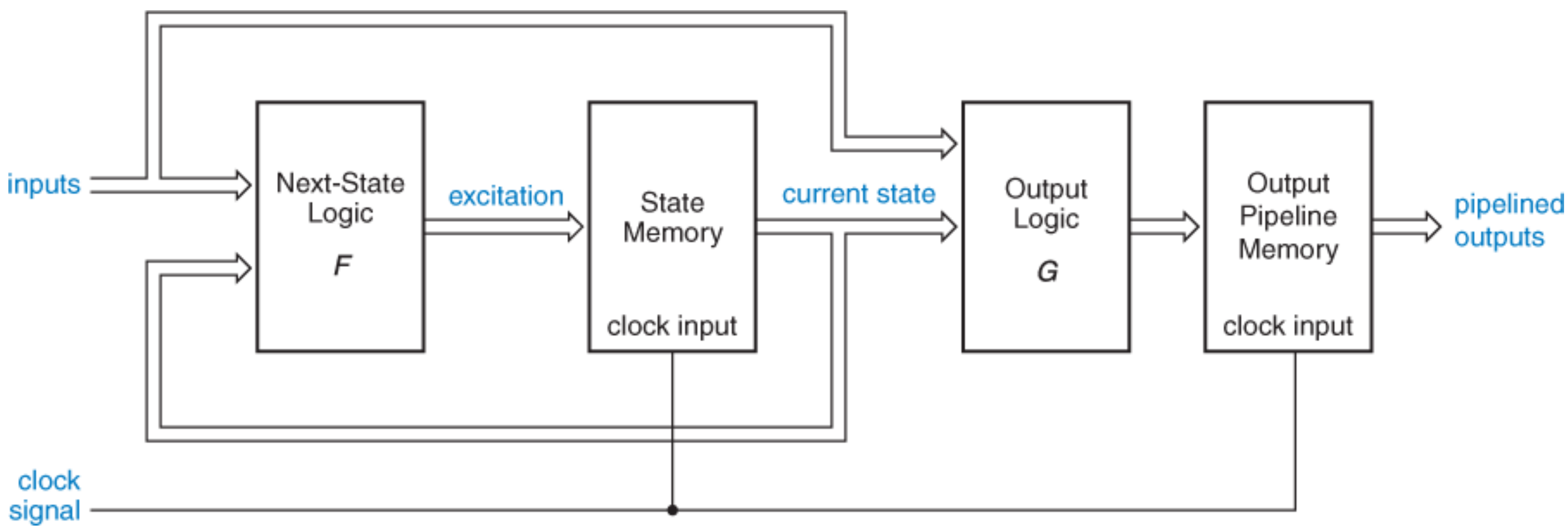
- Note that the outputs of Moore's machine are synchronous while Mealy's are not

Comparing Moore and Mealy machines

- Mealy state machines
  - + Typically have fewer states
  - + React faster to inputs — don't wait for clock
  - – Asynchronous outputs can be dangerous
- Moore state machines
  - + Safer to use because outputs change at clock edge
  - – May take additional logic to decode state into outputs
- Alternatively, design synchronous Mealy machines
  - – Design a Mealy machine
  - – Then register the outputs

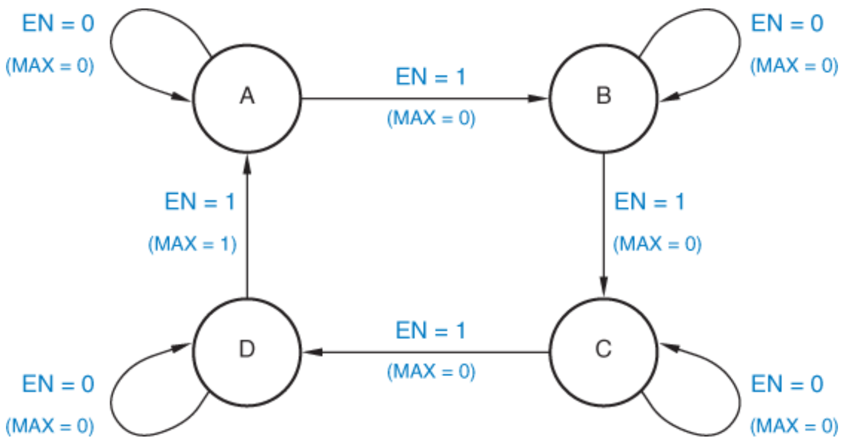
Mealy Machine with Pipelined Outputs

- Registered state and registered outputs
  - No glitches on outputs
  - No race conditions between communicating machines

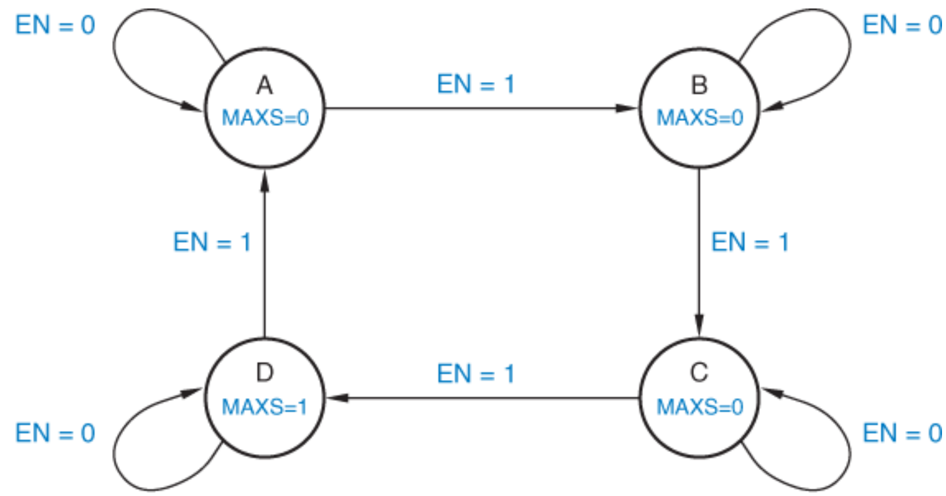


State Diagram

MAX/MAXS is output



Mealy Machine



Moore Machine

- Use graphical format to represent state transition table

FSM Example

This table tells that this is a Mealy machine

Transition Table

Q1Q0	Q1*Q0*	
Q1Q0	EN=0	EN=1
00	00	01
01	01	10
10	10	11
11	11	00

State Table

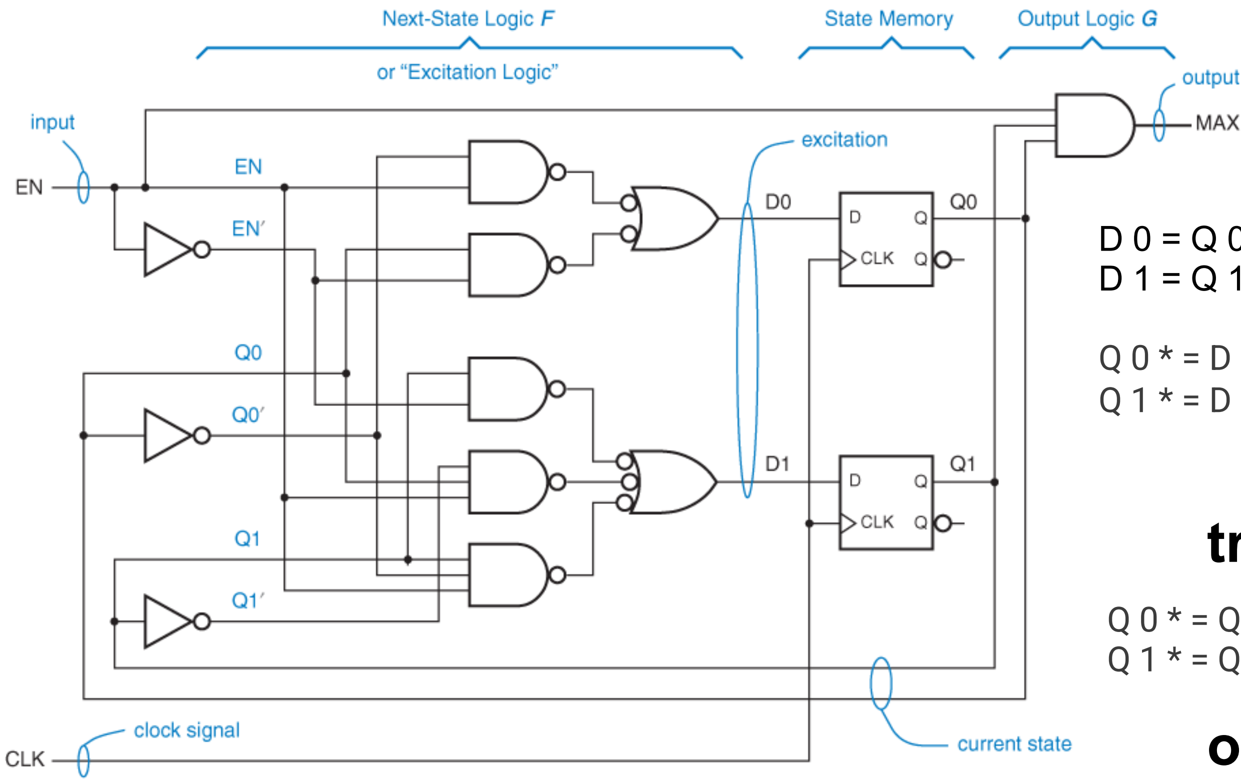
S	S*	
	EN=0	EN=1
A	A	B
B	B	C
C	C	D
D	D	A

State/Output Table

S	S*, MAX	
	EN=0	EN=1
A	A, 0	B, 0
B	B, 0	C, 0
C	C, 0	D, 0
D	D, 0	A, 1

- Use tables to represent transition relationships
- Can use alphanumeric state names , e.g. A, B, S0, S1 to replace “00”, “01” etc.

FSM Example



$D0 = Q0 \cdot EN' + Q0' \cdot EN$
$D1 = Q1 \cdot EN' + Q1' \cdot Q0 \cdot EN + Q1 \cdot Q0' \cdot EN$

$Q0^* = D0$
$Q1^* = D1$

transition equation

$Q0^* = Q0 \cdot EN' + Q0' \cdot EN$
$Q1^* = Q1 \cdot EN' + Q1' \cdot Q0 \cdot EN + Q1 \cdot Q0' \cdot EN$

output equation

$MAX = Q1 \cdot Q0 \cdot EN$

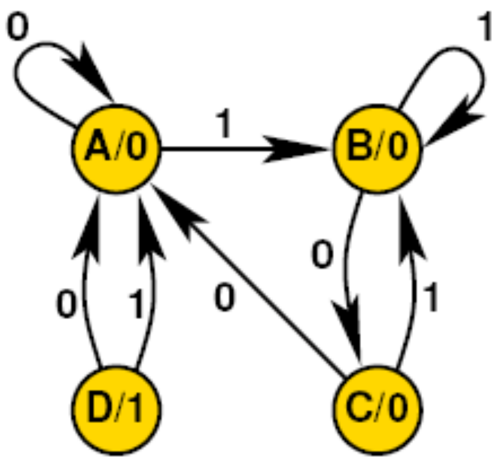
- Circuits are built to realize transition table/equations

State Minimization

Implement FSM with fewest possible states

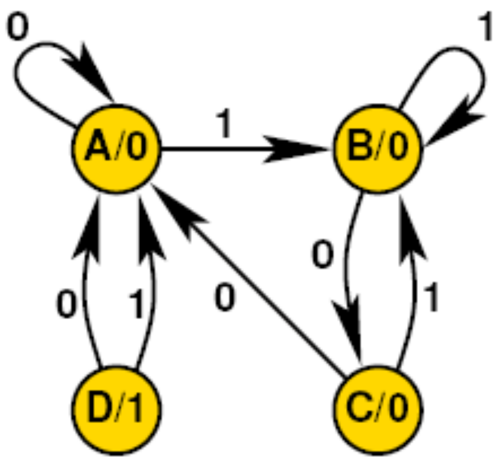
- Least number of flip-flops
  Number of distinct states to be encoded correlates with the number of flip flops that will store the codes for those states
- Boundaries are power of two number of states
- Reduce the number of gates needed for implementation of the combinational logic that fires up the state code contents of the memory elements

State Reduction



	Next state		
Present State	I= 0	I = 1	Output
A	A	B	0
B	C	B	0
C	A	B	0
D	A	A	1

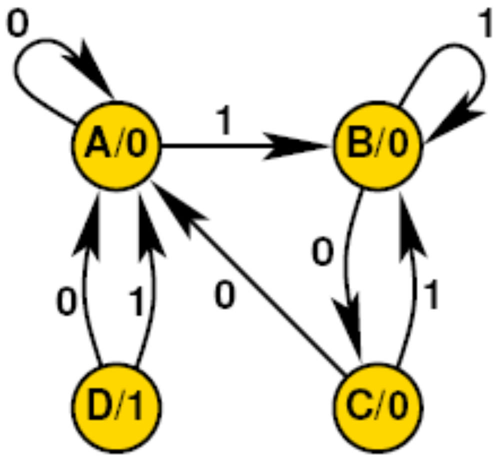
State Reduction



	Next state		
Present State	I= 0	I = 1	Output
A	A	B	0
B	C	B	0
C	A	B	0
D	A	A	1

Same next state for same input AND same output

State Reduction

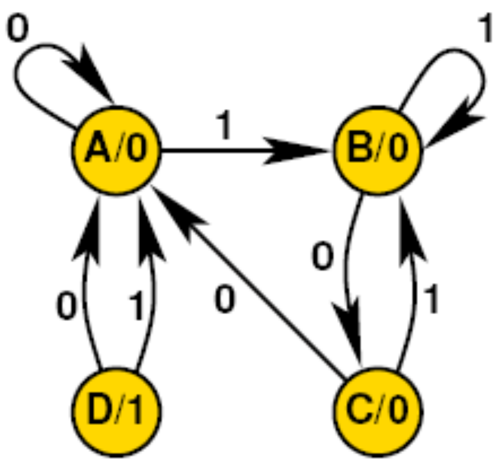


	Next state		
Present State	I= 0	I = 1	Output
AC	AC	B	0
B	AC	B	0
D	AC	AC	1

A and C are equivalent
Give a common new name (AC) and let one of them also represent both (remove one state from the table)

Update all states named A or C as AC

State Reduction



	Next state		
Present State	I= 0	I = 1	Output
ABC	ABC	ABC	0
D	ABC	ABC	1

AC and B are equivalent
Give a common new name (ABC) and let one of them also represent both (remove one state from the table)

Update all states named AC or B as ABC

State Reduction

Identify and combine states that have equivalent behavior

Equivalent States: for all input combinations, states transition to the same or equivalent states, as well as same output behavior

State Reduction

Algorithmic Approach!!!

- Start with state transition table
- Identify states with same output behavior
- If such states transition to the same next state, they are equivalent
- Combine into a single new renamed state
- Repeat until no new states are combined

But How??

Systematic Approach to State Reduction

- Sometimes equivalency is not so obvious
- Implication Chart

Implication Chart

	Next state		
Current state	I=0	I=1	output
A	B	E	0
B	C	D	0
C	D	D	0
D	A	A	1
E	D	D	0

It is not obvious whether A and B are equivalent
If A and B were to be equivalent, then

B and C have to be equivalent
E and D have to be equivalent

In this case, we see that E and D are not (their output behavior does not match), hence this implies that A and B cannot be equivalent

Implication Chart

1. Construct implication chart, one square for each combination of states taken two at a time

B				
C				
D				
E				
	A	B	C	D

Implication Chart

1. Construct implication chart, one square for each combination of states taken two at a time
2. Square labeled (Si, Sj)
   - if outputs differ than square gets X
   - Otherwise write down implied state pairs for all input combinations
   - If the equivalency is directly observable the square gets 1

B	B=C D=E			
C	B=D D=E	C=D		
D	X	X	X	
E	B=D D=E	C=D	1	X
	A	B	C	D

	Next state		
Current state	I=0	I=1	output
A	B	E	0
B	C	D	0
C	D	D	0
D	A	A	1
E	D	D	0

Implication Chart

2. Square labeled (*Si*, *Sj)*
   - if outputs differ then square gets *X*
   - Otherwise write down implied state pairs for all input combinations
   - If the equivalency is directly observable the square gets 1
3. Advance through chart top-to-bottom and left-to-right
   - If square (*Si*, *Sj)* contains next state pair *Sm*, *Sn* and that pair has a square already labeled *X*, then *Si*, *Sj* is also labeled *X*

B	B=C D=E			
C	B=D D=E	C=D		
D	X	X	X	
E	B=D D=E	C=D	1	X
	A	B	C	D

B	**X**			
C	**X**	C=D		
D	X	X	X	
E	**X**	C=D	1	X
	A	B	C	D

Implication Chart

2. Square labeled (*Si*, *Sj)*
   - if outputs differ than square gets *X*
   - Otherwise write down implied state pairs for all input combinations
   - If the equivalency is directly observable the square gets 1
3. Advance through chart top-to-bottom and left-to-right
   - If square (*Si*, *Sj)* contains next state pair *Sm*, *Sn* and that pair has a square already labeled *X*, then *Si*, *Sj* is also labeled *X*

B	X			
C	X	C=D		
D	X	X	X	
E	X	C=D	1	X
	A	B	C	D

⇒

B	**X**			
C	**X**	X		
D	X	X	X	
E	**X**	X	1	X
	A	B	C	D

Implication Chart

2. Square labeled (*Si*, *Sj)*
   – if outputs differ than square gets *X*
   – Otherwise write down implied state pairs for all input combinations
   – If the equivalency is directly observable the square gets 1
3. Advance through chart top-to-bottom and left-to-right
   – If square (*Si*, *Sj)* contains next state pair *Sm*, *Sn* and that pair has a square already labeled *X*, then *Si*, *Sj* is also labeled *X*
4. Continue executing Step 3 until no new squares are marked with *X*
5. For each remaining unmarked square *Si*, *Sj*, then conclude *Si* and *Sj* are equivalent

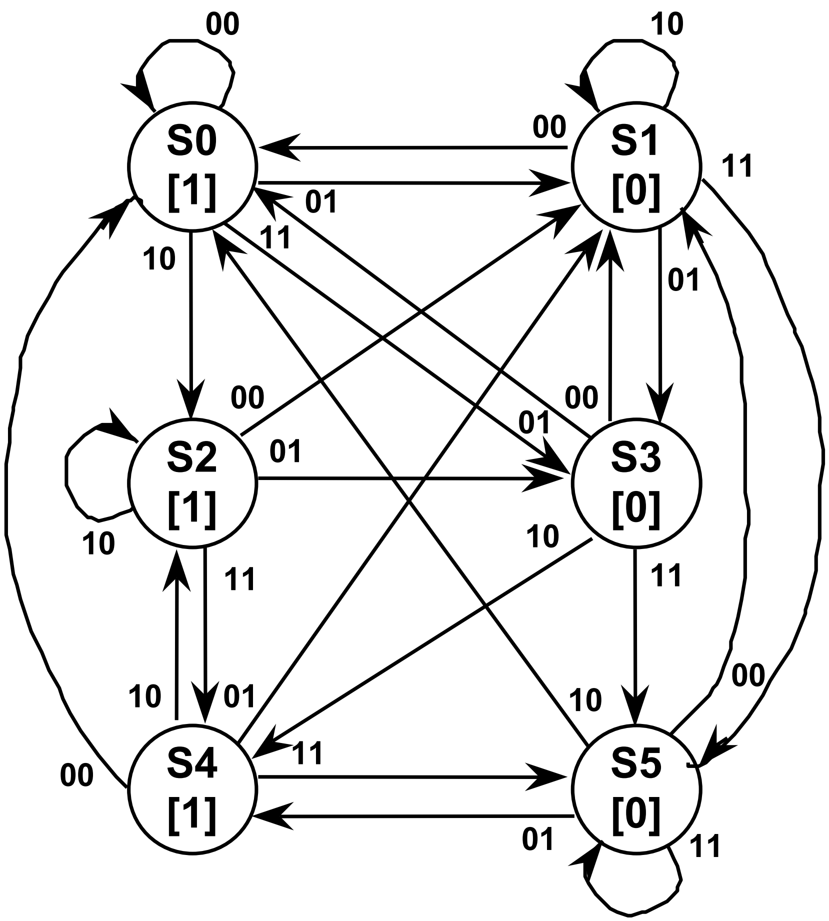
B	**X**			
C	**X**	**X**		
D	X	X	X	
E	**X**	**X**	1	X
	A	B	C	D

C and E are the only equivalent pair

Implication Chart-Summary

1. Construct implication chart, one square for each combination of states taken two at a time
2. Square labeled (*Si*, *Sj)*
   - if outputs differ than square gets *X*
   - Otherwise write down implied state pairs for all input combinations
3. Advance through chart top-to-bottom and left-to-right
   - If square (*Si*, *Sj)* contains next state pair *Sm*, *Sn* and that pair has a square already labeled *X*, then *Si*, *Sj* is also labeled *X*
4. Continue executing Step 3 until no new squares are marked with *X*
5. For each remaining unmarked square *Si*, *Sj*, then conclude *Si* and *Sj* are equivalent

Multiple Input State Diagram Example

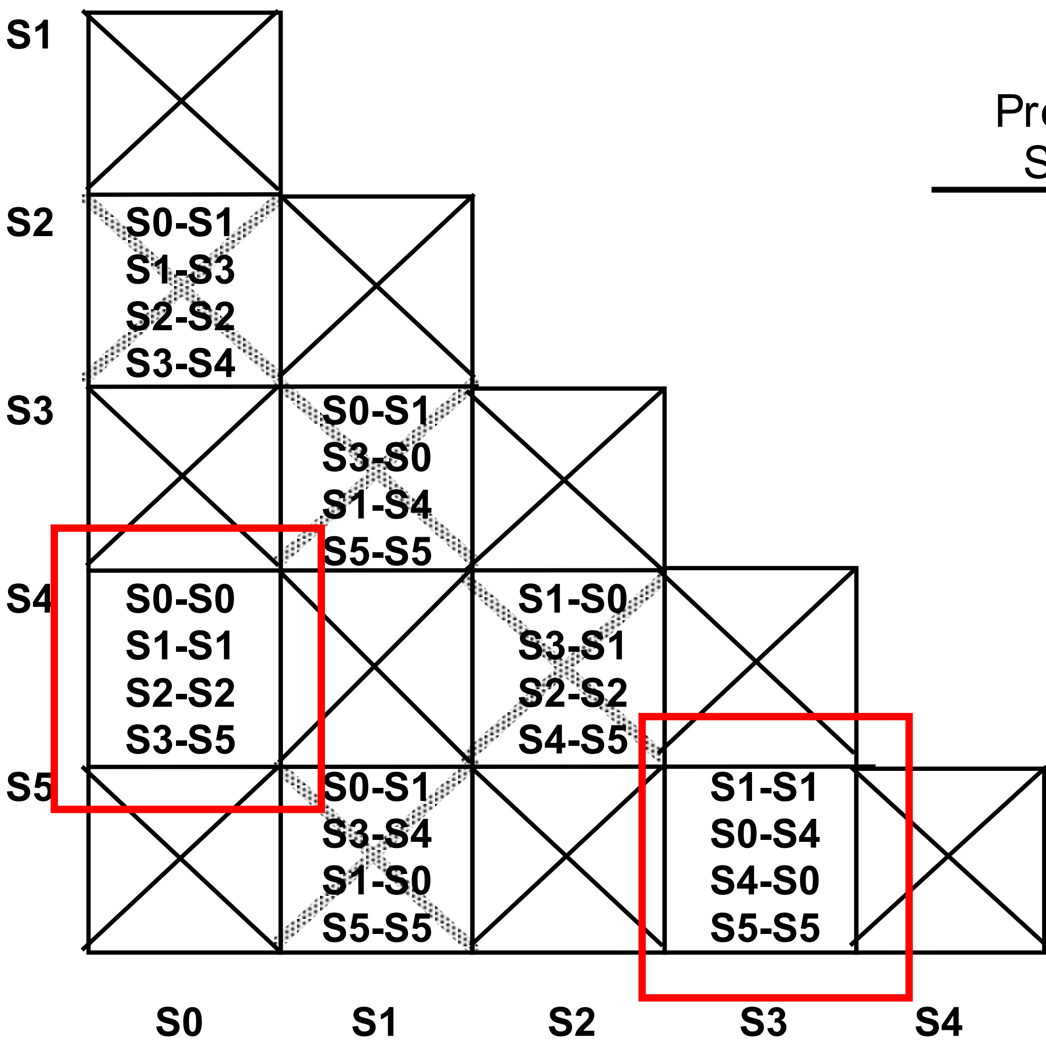


State Diagram

Present State	Next State 00	01	10	11	Output
S_0	S_0	S_1	S_2	S_3	1
S_1	S_0	S_3	S_1	S_5	0
S_2	S_1	S_3	S_2	S_4	1
S_3	S_1	S_0	S_4	S_5	0
S_4	S_0	S_1	S_2	S_5	1
S_5	S_1	S_4	S_0	S_5	0

Symbolic State Diagram

Example (contd)



Implication Chart

Present State	Next State 00	01	10	11	Output
S_0	S_0	S_1	S_2	S_3	1
S_1	S_0	S_3	S_1	S_5	0
S_2	S_1	S_3	S_2	S_4	1
S_3	S_1	S_0	S_4	S_5	0
S_4	S_0	S_1	S_2	S_5	1
S_5	S_1	S_4	S_0	S_5	0

Minimized State Table

S3 and S5
S0 and S4
are equivalent

Appendix