COL215 Digital Logic and System Design

- State Minimization in FSMs,
- Introduction to Asynchronous FSMs

07 November 2017

Cubical Technique for Minimization

Star (*) operation

$A_i \backslash B_i$	0	1	x
0	0	ø	0
1	ø	1	1
x	0	1	x

Possible results of operation

C = A * B on cubes A and B

- if A and B are far apart, no new cube is generated
- if A, B overlap, the result is intersection of A and B
- if A, B differ in exactly one literal, new cube generated is union of a part of A and a part of B

If $A_i * B_i = ø$ for more than one i,
then C = ø
Otherwise,
if $A_i * B_i \neq ø$ then
$C_i = A_i * B_i$
else
$C_i = x$

Sharp (#) operation

$A_i \backslash B_i$	0	1	x
0	ε	ø	ε
1	ø	ε	ε
x	1	0	ε

Possible results of operation

C = A # B on cubes A and B

- if A and B are disjoint, the result is A
- if A is contained in B, the result is empty
- if B partly overlaps with A, the result is non-overlapping part of A

If $A_i \# B_i = ø$ for some i,
then C = A
If $A_i \# B_i = ε$ for all i,
then C = ø

State Minimization

Finite State Machine Model

Quintuple $(X, Y, S, \delta, \lambda)$

X = set of primary input patterns

Y = set of primary output patterns

S = set of states {includes an initial state}

δ = state transition function $S \times X \rightarrow S$

λ = output function

$S \times X \rightarrow Y$ for Mealy model or

$S \rightarrow Y$ for Moore model

Assumptions

- Synchronous circuits
- Single clock
- Single phase
- Edge triggered D type flip-flops/registers

Completely specified FSMs

Definition of state equivalence:

equiv (s_i , s_j) iff

output_seq (s_i, input_seq) =

output_seq (s_j, input_seq)

for all input_seq

Incompletely specified FSMs

- Definition of compatible states:

comp (s_i , s_j) iff

output_seq (s_i, input_seq) =

output_seq (s_j, input_seq)

when *both are defined*,

for all *applicable* input_seq

Incompletely specified FSMs

- Easier way to check compatibility:

comp (s_i , s_j) ≡

(s_i = s_j)

or

λ (s_i, x) = λ (s_j, x) *if both are defined* and

comp (δ (s_i, x), δ (s_j, x)),

for all *applicable* x in X

Equivalence vs compatibility

- Reflexive, symmetric and transitive
- Equivalence classes (groups of equivalent states) are disjoint
- Partition is unique

- Reflexive, symmetric but NOT transitive
- Compatibility classes (groups of compatible states) are overlapping
- Partition is not unique, each choice has “implications”

“Implications” while partitioning

- If states A and B are chosen as compatible, successor of A and successor of B *(under all applicable inputs)* should also be chosen as compatible
- Closure property: A partition has closure property if all the implications are taken care of

Example

In	PS	NS	Out
0	s_1	s_3	1
1	s_1	s_5	*
0	s_2	s_3	*
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

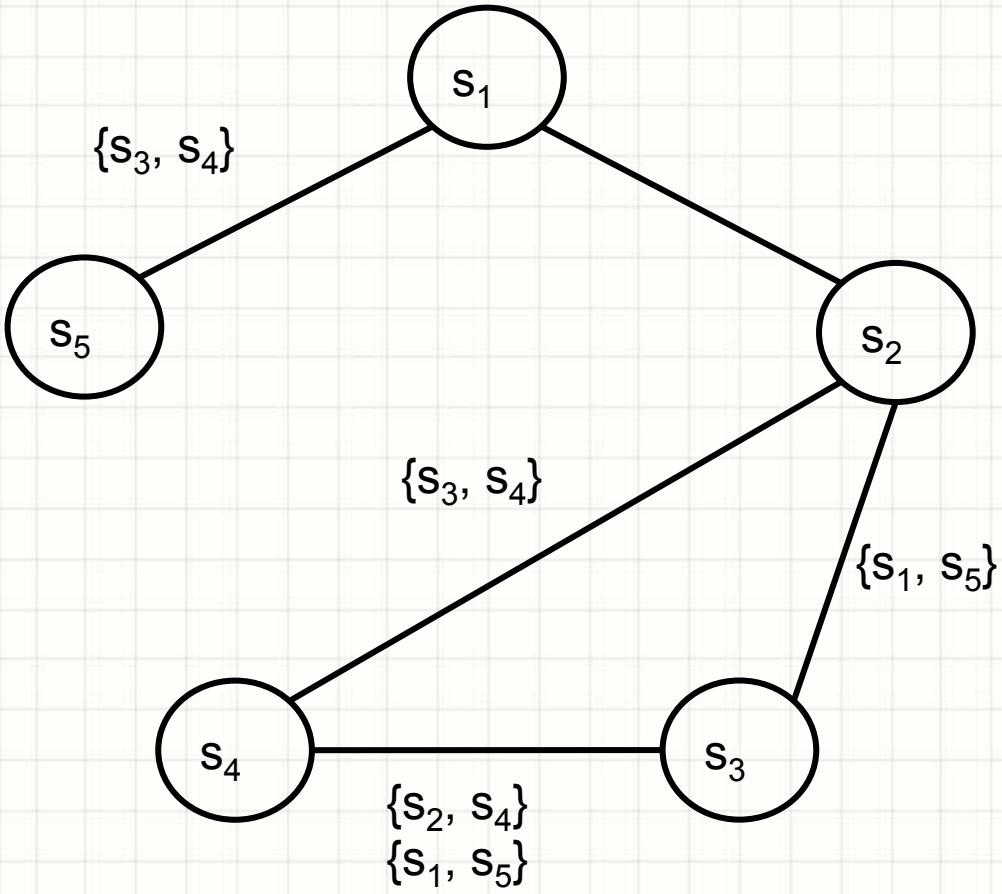
There is an exponential number of completely specified FSMs corresponding to an incompletely specified FSM

Compatible pairs

	Pairs	Depends on
Compatible	$\{s_1, s_2\}$	
Compatible	$\{s_1, s_5\}$	$\{s_3, s_4\}$
Compatible	$\{s_2, s_4\}$	$\{s_3, s_4\}$
Compatible	$\{s_2, s_3\}$	$\{s_1, s_5\}$
Compatible	$\{s_3, s_4\}$	$\{s_2, s_4\}$, $\{s_1, s_5\}$
Incompatible	$\{s_1, s_3\}$	
Incompatible	$\{s_1, s_4\}$	
Incompatible	$\{s_2, s_5\}$	
Incompatible	$\{s_3, s_5\}$	
Incompatible	$\{s_4, s_5\}$	

In	PS	NS	Out
0	s_1	s_3	1
1	s_1	s_5	*
0	s_2	s_3	*
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

Compatibility classes



Classes	Implied classes
$\{s_1, s_2\}$	
$\{s_1, s_5\}$	$\{s_3, s_4\}$
$\{s_2, s_3, s_4\}$	$\{s_1, s_5\}$

Selected classes may not always be maximal

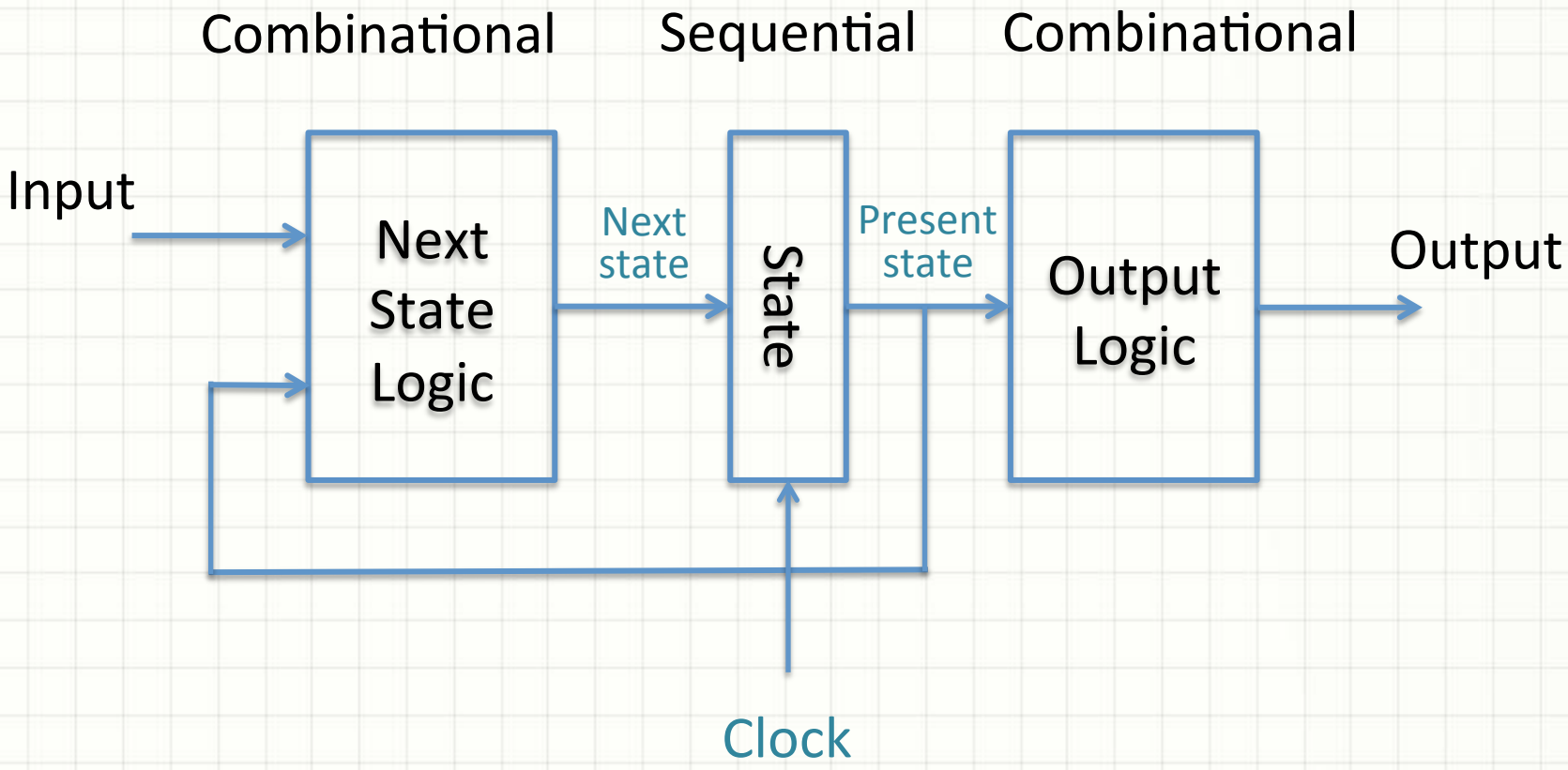
Minimized FSM Table

In	PS	NS	Out
0	s_1	s_3	1
1	s_1	s_5	*
0	s_2	s_3	*
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

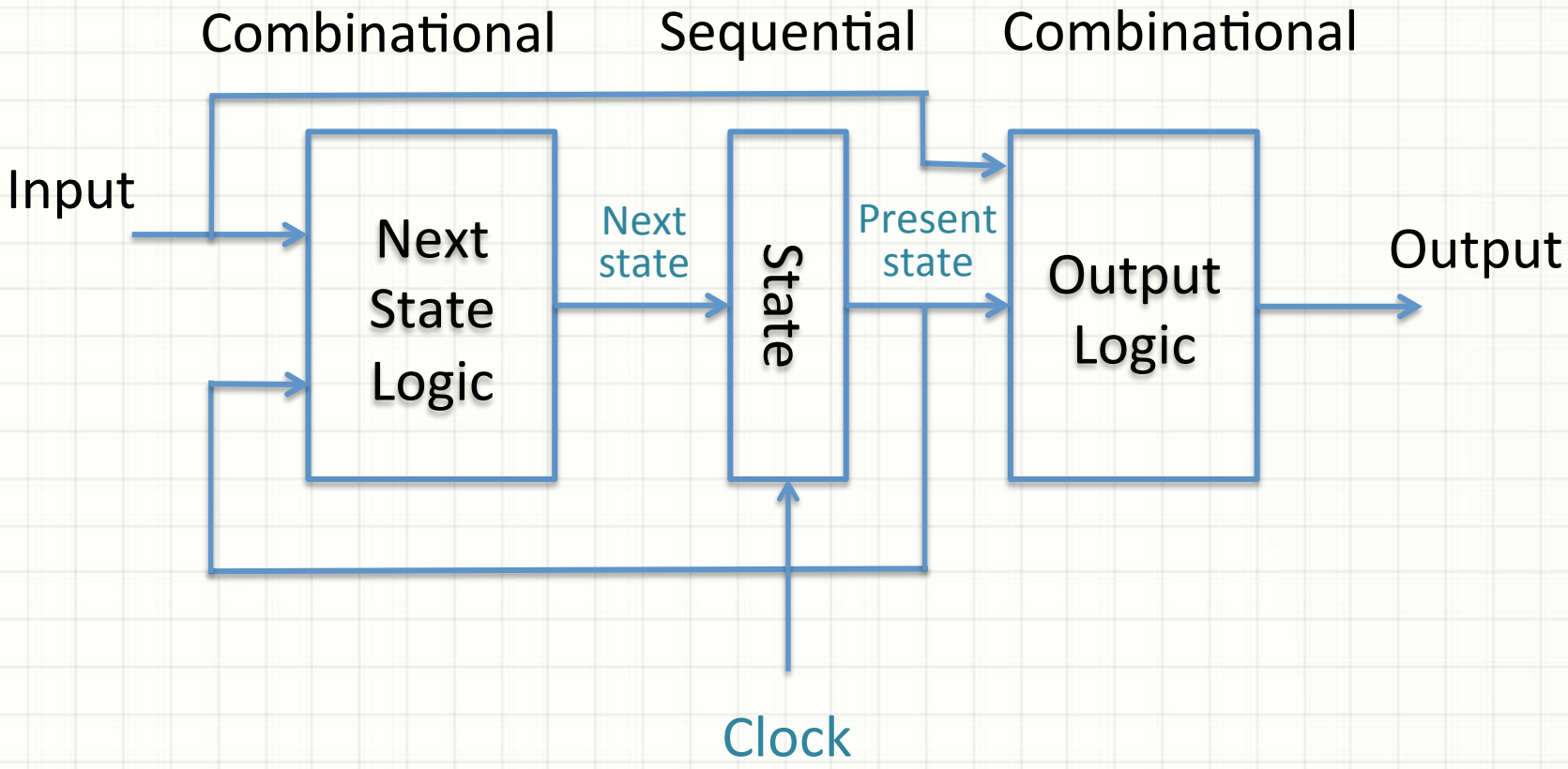
In	PS	NS	Out
0	s_{15}	s_{234}	1
1	s_{15}	s_{15}	0
0	s_{234}	s_{234}	0
1	s_{234}	s_{15}	1

Introduction to Asynchronous FSMs

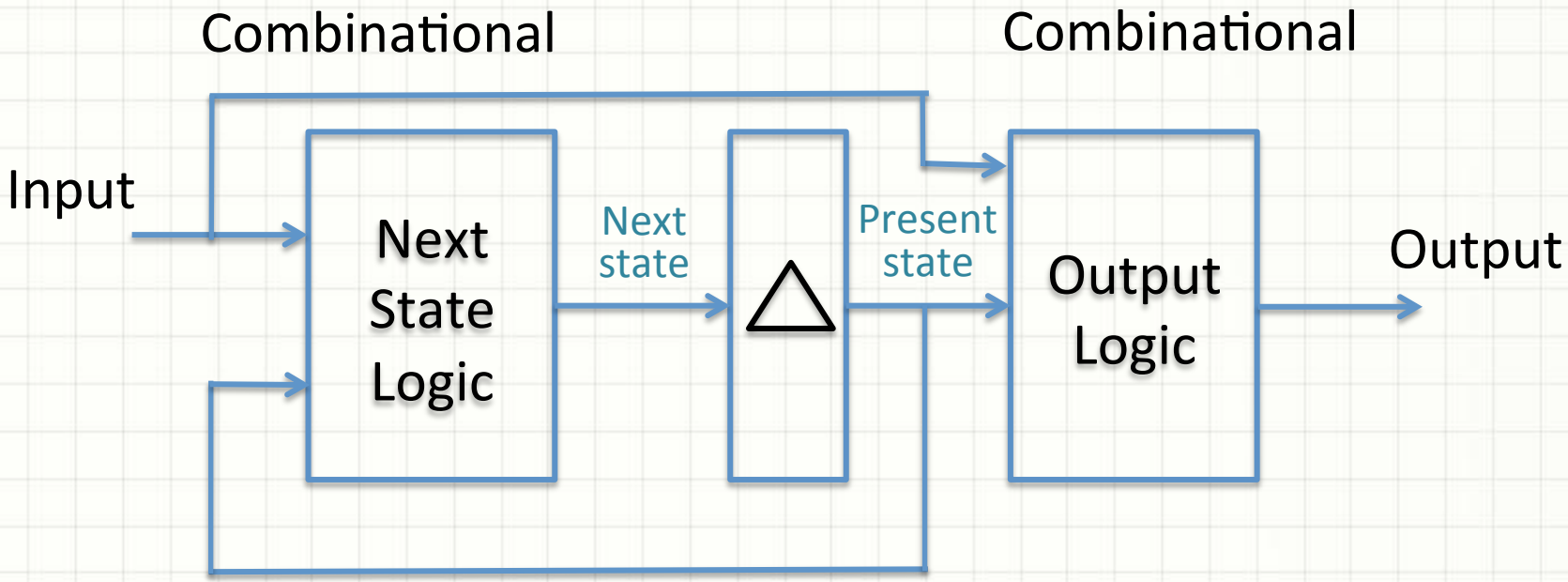
Synchronous FSM (Moore)



Synchronous FSM (Mealy)



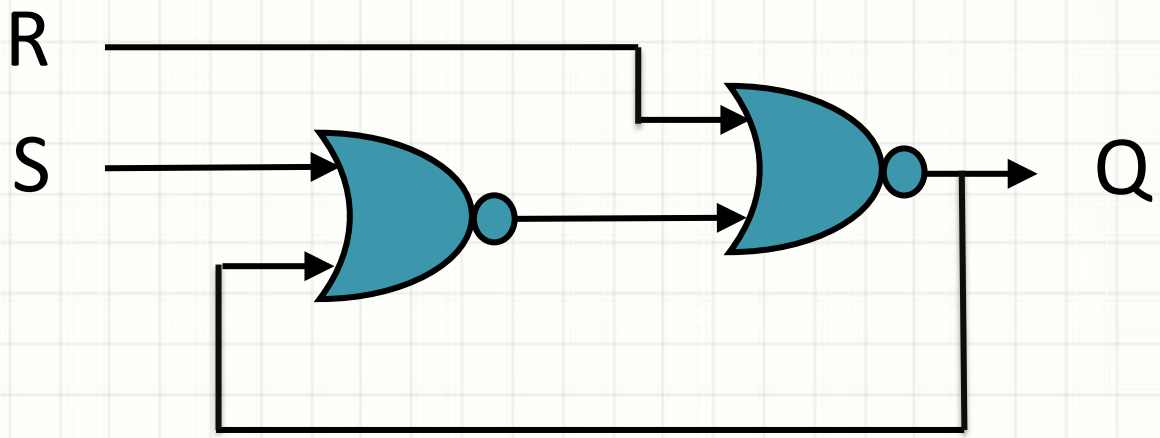
Asynchronous FSM (Mealy)



Outline

- SR Latch
- Gated D Latch
- Master-Slave D Flip-flop

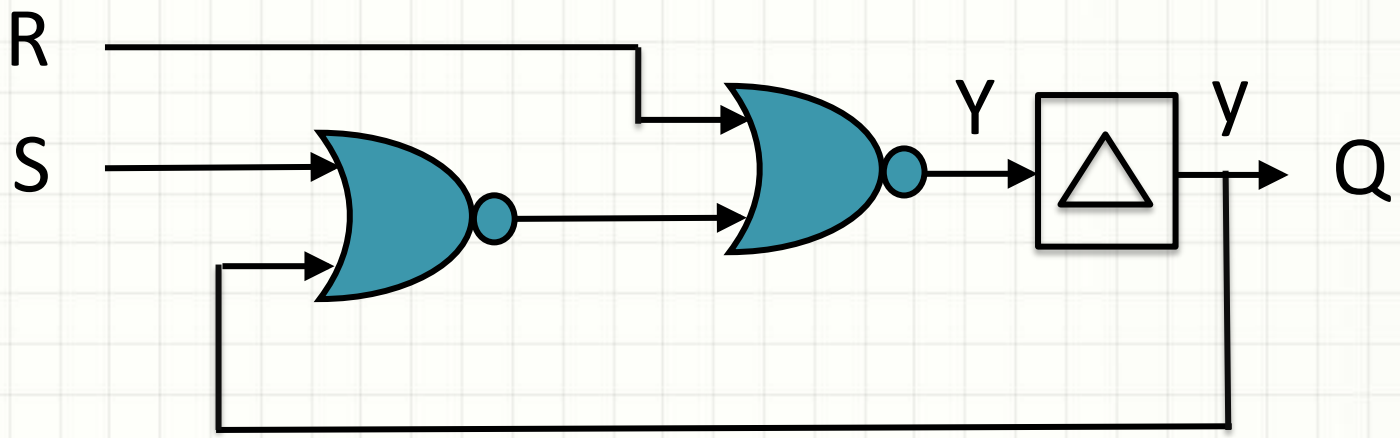
Set-Reset (SR) Latch



R : Reset

S : Set

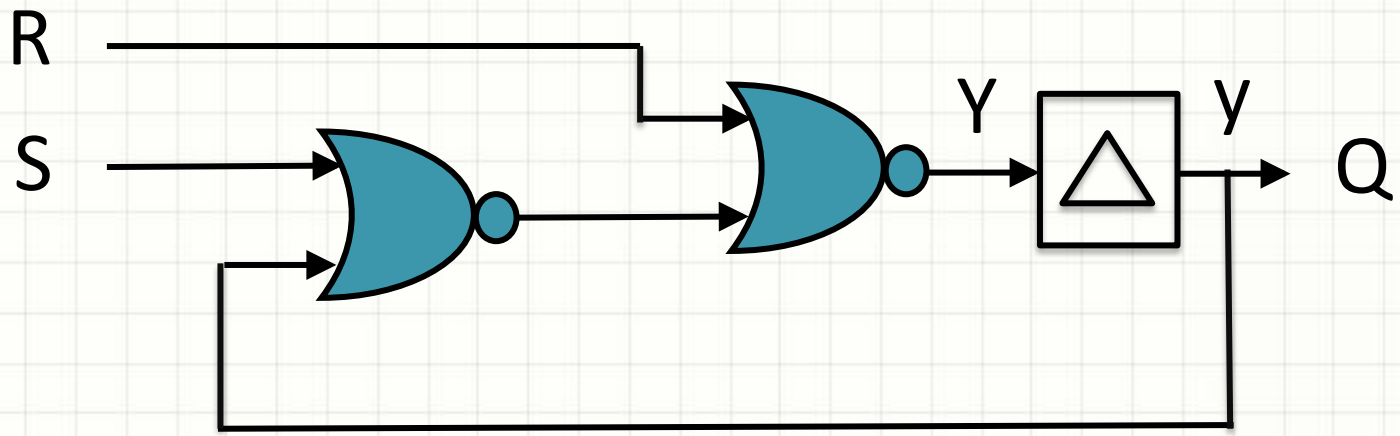
Set-Reset (SR) Latch



R : Reset

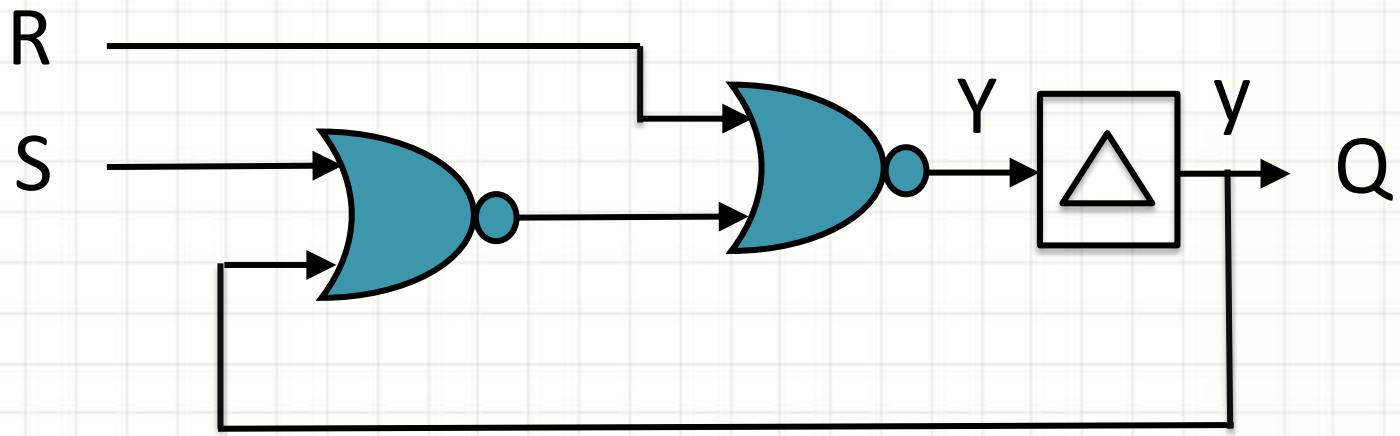
S : Set

State assigned table



$$Y = ((S + y)' + R)' = (S + y).R' = S.R' + y.R'$$

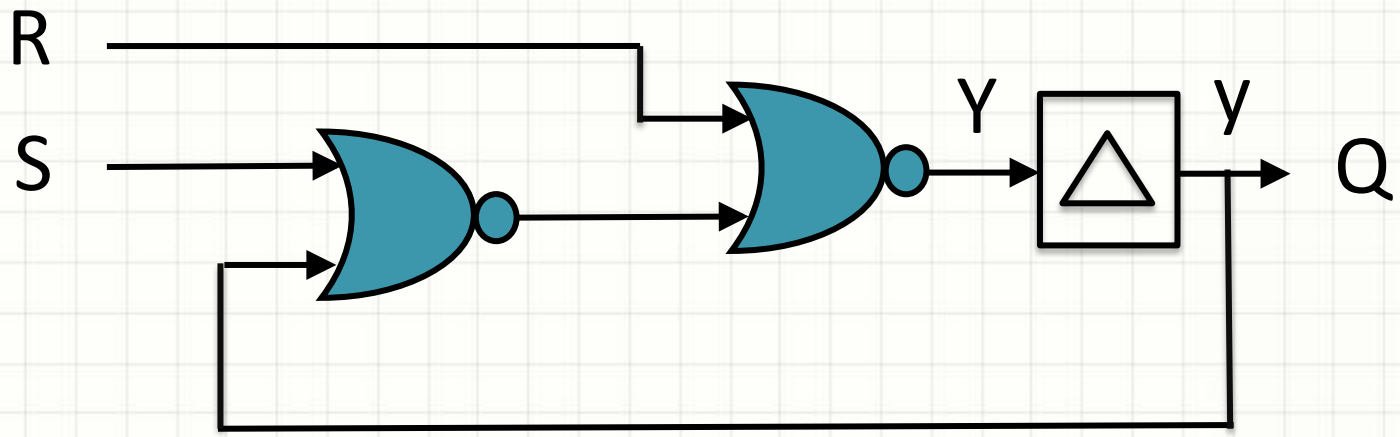
State assigned table



Present state y	Next state Y				Output Q
	SR = 00	SR = 01	SR = 10	SR = 11	
0	0	0	1	0	0
1	1	0	1	0	1

$$Y = ((S + y)' + R)' = (S + y).R' = S.R' + y.R'$$

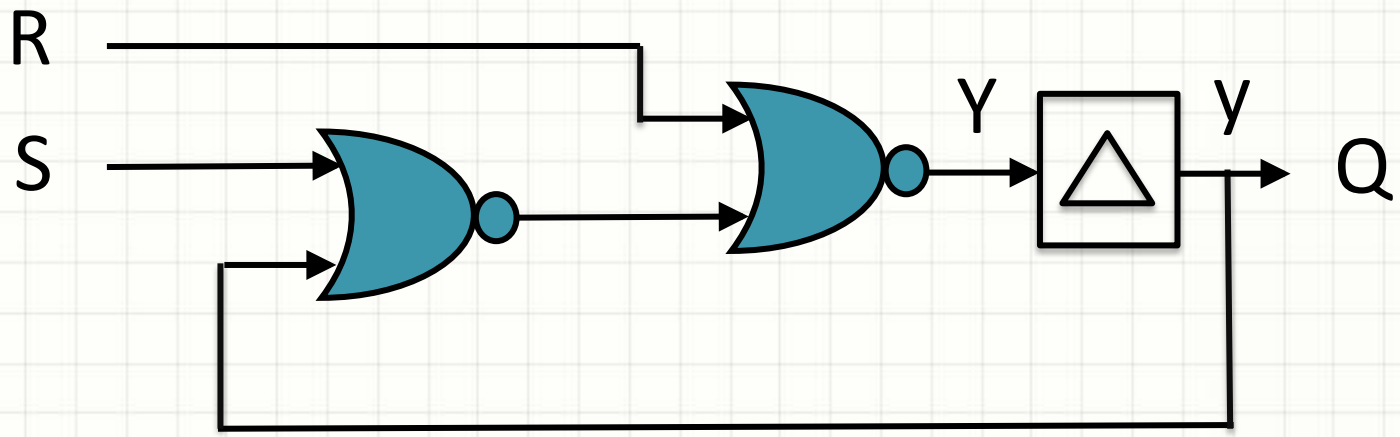
State assigned table



Present state y	Next state Y				Output Q
	SR = 00	SR = 01	SR = 10	SR = 11	
0	0	0	1	0	0
1	1	0	1	0	1

○ stable state

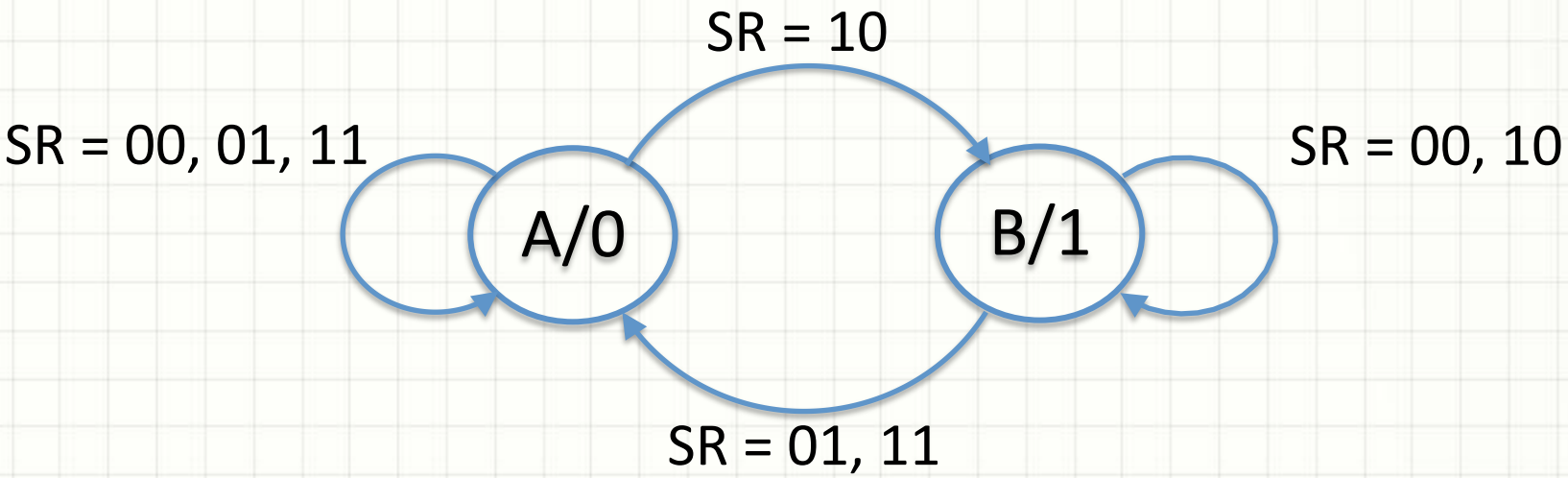
State transition table



Present state y	Next state Y				Output Q
	SR = 00	SR = 01	SR = 10	SR = 11	
A	A	A	B	A	0
B	B	A	B	A	1

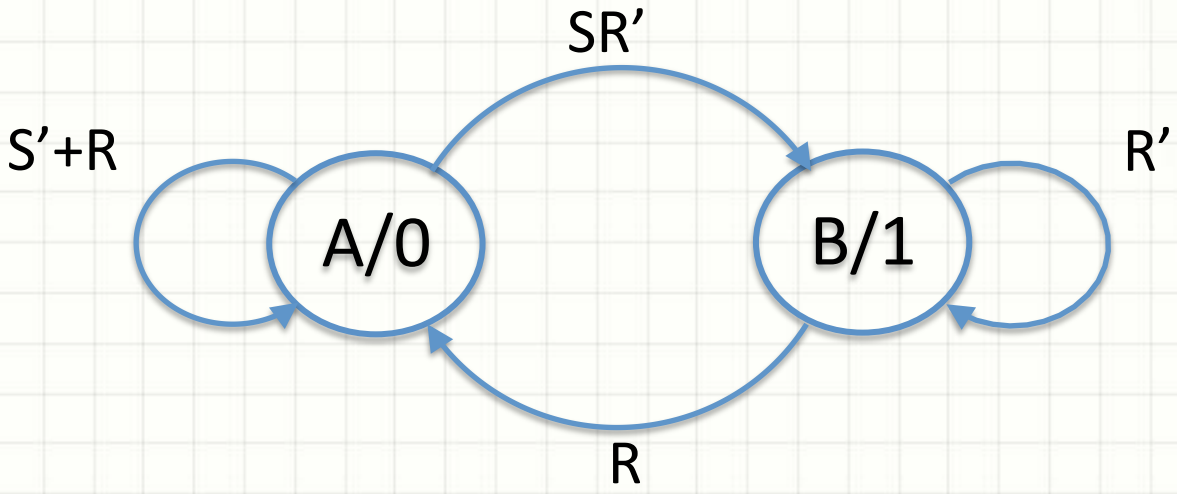
State transition diagram

Present state y	Next state Y				Output Q
	SR = 00	SR = 01	SR = 10	SR = 11	
A	A	A	B	A	0
B	B	A	B	A	1



State transition diagram

Present state y	Next state Y				Output Q
	SR = 00	SR = 01	SR = 10	SR = 11	
A	A	A	B	A	0
B	B	A	B	A	1



QUESTIONS?