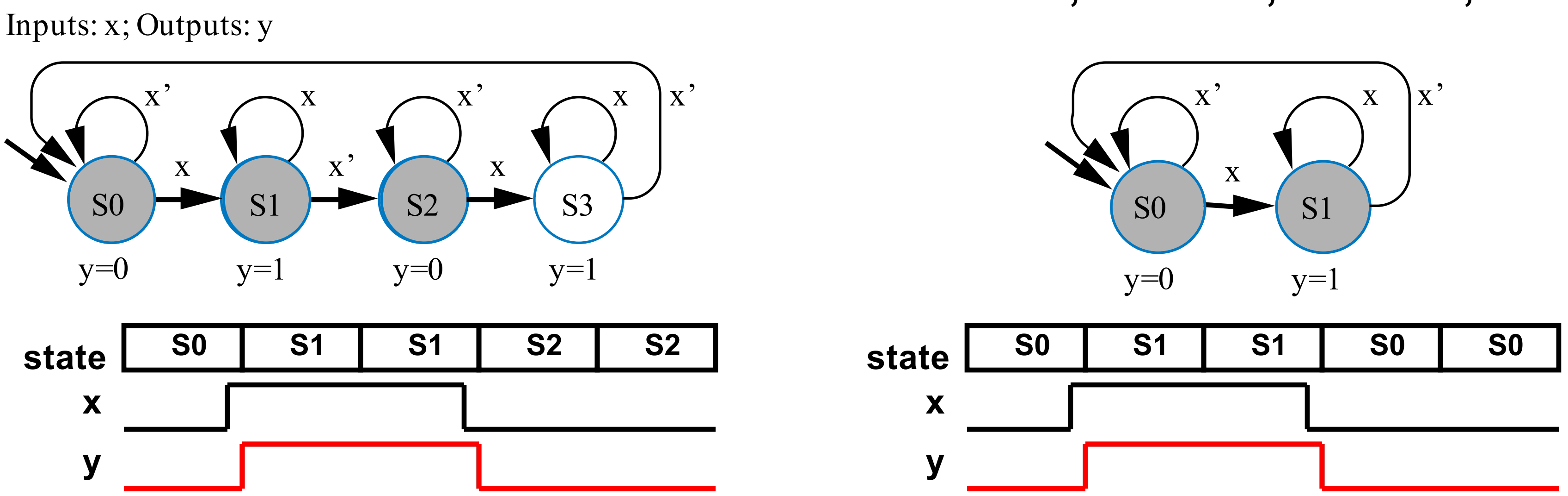
Topic 11

FSM Optimizations

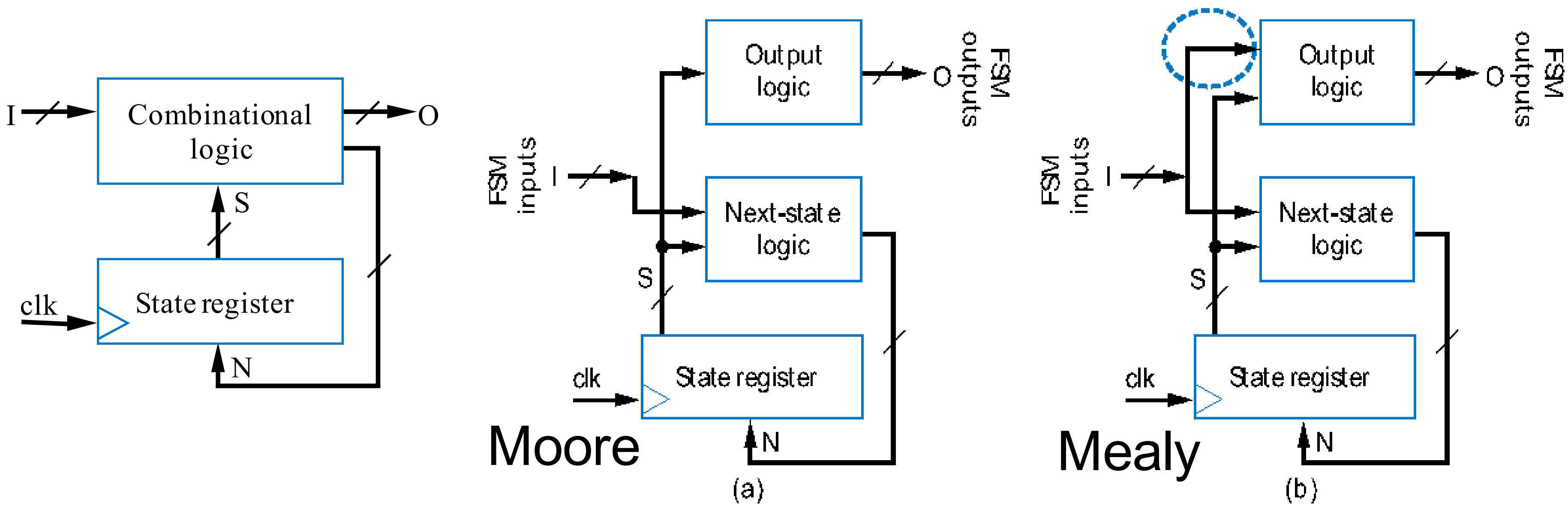
Optimization by State Reduction

- **Goal**: **Reduce number of states in FSM *without* changing behavior**
  - Fewer states potentially reduce size of state register
- Consider the two FSMs below with *x*=1, then 1, then 0, 0



For the same sequence of inputs,
the output of the two FSMs is the same

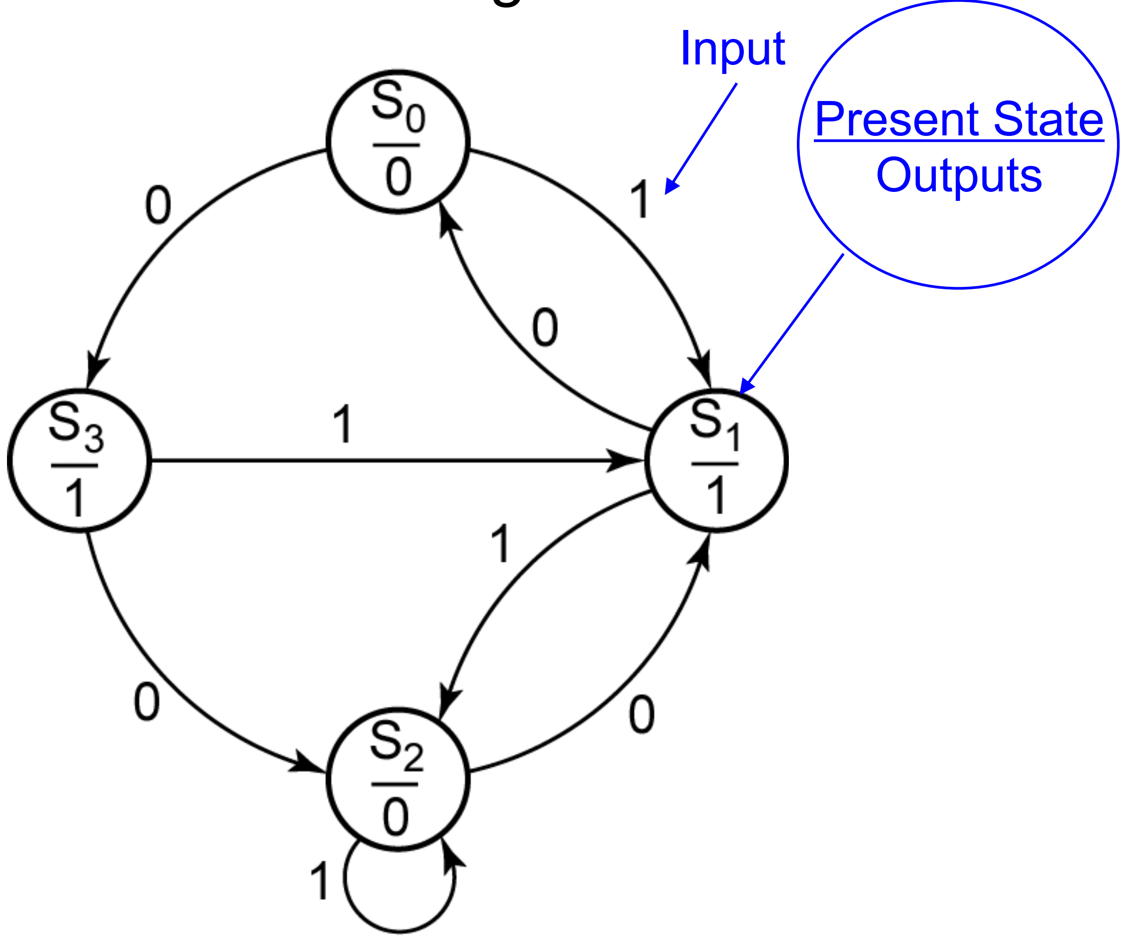
Moore vs. Mealy FSMs



- FSM implementation architecture
  - Next state logic – function of present state and FSM inputs
  - Output logic
    - Depends on present state only – ***Moore FSM***
    - Depends on present state and FSM inputs – ***Mealy FSM***

Moore FSM Representation

State Diagram

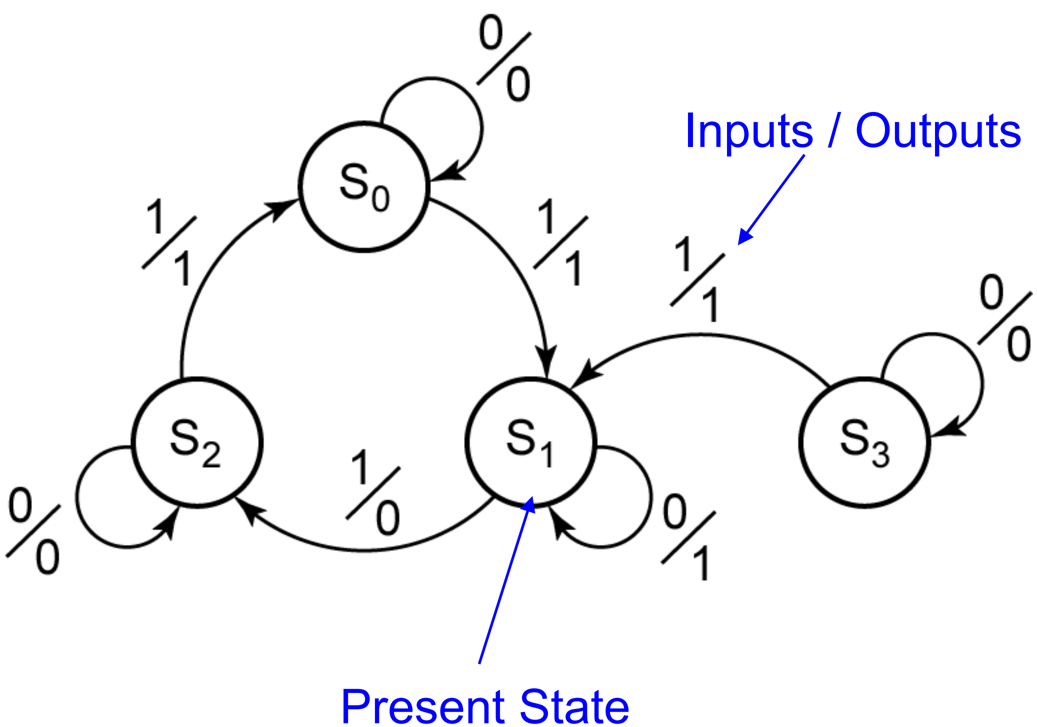


State Table

In	P.S.	N.S.	Out
0	S0	S3	0
1	S0	S1	
0	S1	S0	1
1	S1	S2	
0	S2	S1	0
1	S2	S2	
0	S3	S2	1
1	S3	S1	

Mealy FSM Representation

State Diagram

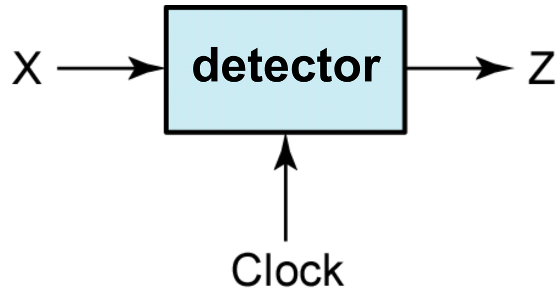


State Table

In	P.S.	N.S.	Out
0	S0	S0	0
1	S0	S1	1
0	S1	S1	1
1	S1	S2	0
0	S2	S2	0
1	S2	S0	1
0	S3	S3	0
1	S3	S1	1

Design of an FSM - Mealy

- Example: design a non-overlapping sequence detector as Mealy FSM



- Z is determined every three bits, Z = 1, as soon as an input sequence 101 is detected

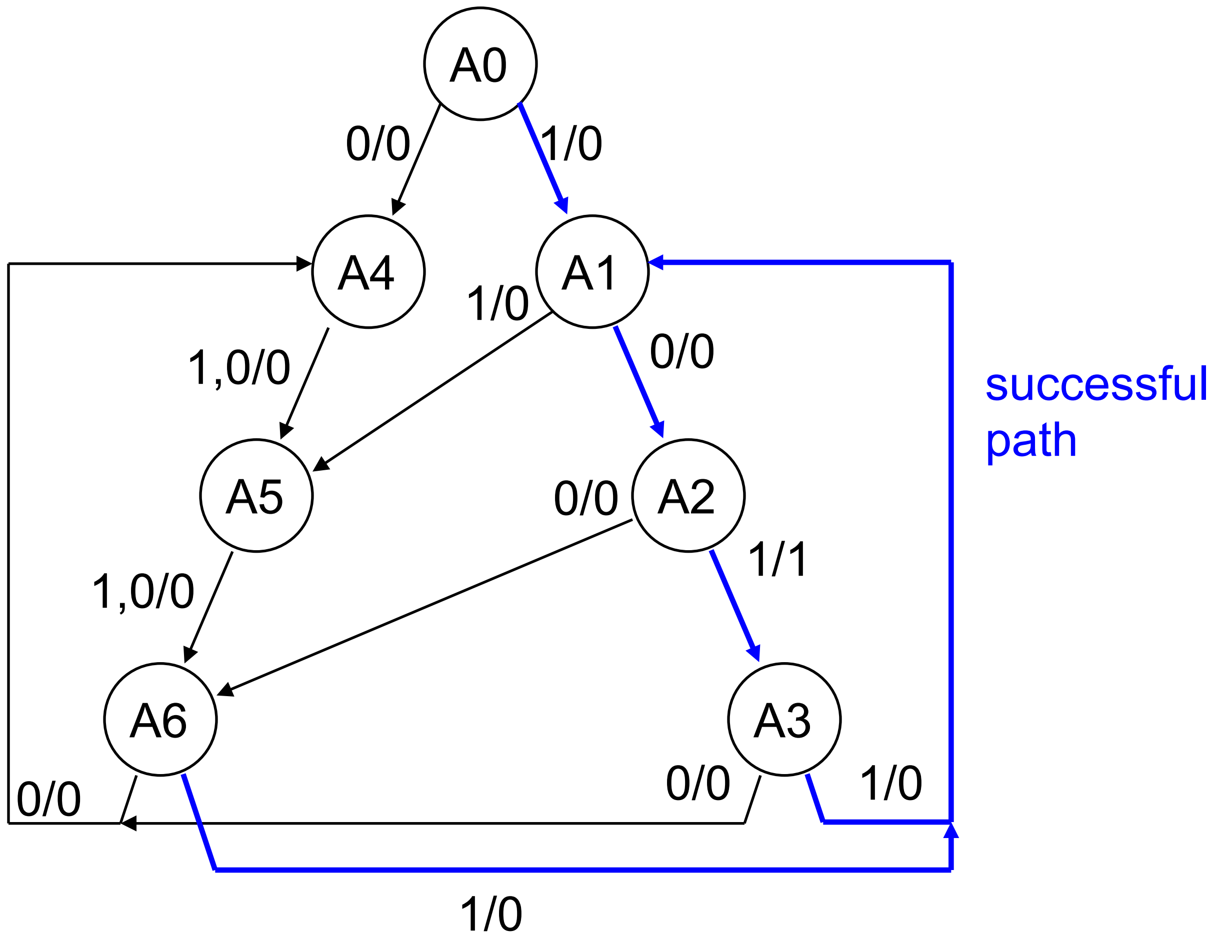
X =	0	0	1	1	0	1	1	0	0	1	0	1	0	1	0
Z =	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

one input: X

one output: Z

Development of State Diagram

- Drawing the state diagram



FSM Optimization – State Reduction

- Two states are equivalent iff both their next states and outputs are identical

Present State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
A0	A4	A1	0	0
A1	A2	A5	0	0
A2	**A0**	**A0**	0	1
~~A3~~	~~A4~~	~~A1~~	~~0~~	~~0~~
A4	A5	A5	0	0
A5	**A0**	**A0**	0	0
~~A6~~	~~A4~~	~~A1~~	~~0~~	~~0~~

Alternative representation of state table
- Easier for state reduction
- Harder for truth table

equivalent states

Reduced State Table

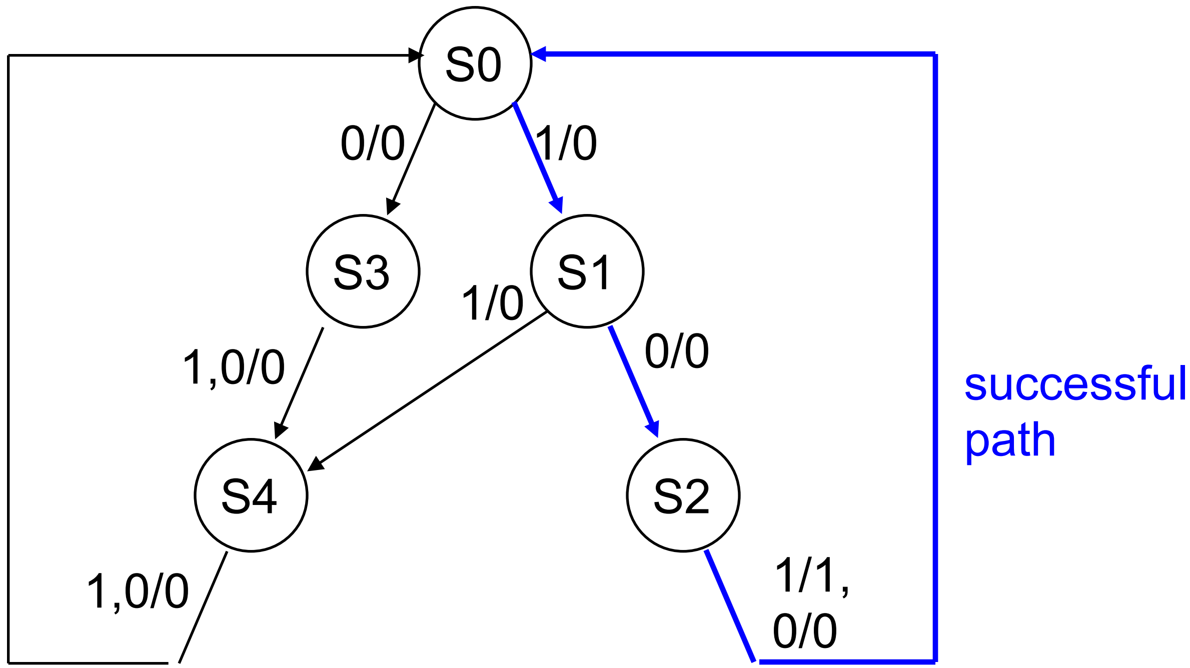
- Reduced state table

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
A0	A4	A1	0	0
A1	A2	A5	0	0
A2	A0	A0	0	1
~~A3~~	~~A4~~	~~A1~~	~~0~~	~~0~~
A4	A5	A5	0	0
A5	A0	A0	0	0
~~A6~~	~~A4~~	~~A1~~	~~0~~	~~0~~

A0 → S0
A1 → S1
A2 → S2
A4 → S3
A5 → S4

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S0	S3	S1	0	0
S1	S2	S4	0	0
S2	S0	S0	0	1
S3	S4	S4	0	0
S4	S0	S0	0	0

Reduced State Diagram



State Assignment

- Number of bits of binary number should be enough to represent all the states

S0	S1	S2	S3	S4
000	001	010	011	100

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
000	011	001	0	0
001	010	100	0	0
010	000	000	0	1
011	100	100	0	0
100	000	000	0	0

In	Present State			Next State			Out
X	P2	P1	P0	n2	n1	n0	Z
0	0	0	0	0	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	0	0	0
……				x			
1	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	1
1	0	1	1	1	0	0	0
1	1	0	0	0	0	0	0
……				x			

State and Output Equations

In	Present State			Next State			Out
X	P2	P1	P0	n2	n1	n0	Z
0	0	0	0	0	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	0	0	0
……				x			
1	0	0	0	0	0	1	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	1
1	0	1	1	1	0	0	0
1	1	0	0	0	0	0	0
……				x			

n2 (X p2 \ p1p0)	00	01	11	10
00	0	0	1	0
01	0	X	X	X
11	0	X	X	X
10	0	1	1	0

n2 = p0 X + p1p0

n1 (X p2 \ p1p0)	00	01	11	10
00	1	1	0	0
01	0	X	X	X
11	0	X	X	X
10	0	0	0	0

n1 = p2’p1’X’

n0 (X p2 \ p1p0)	00	01	11	10
00	1	0	0	0
01	0	X	X	X
11	0	X	X	X
10	1	0	0	0

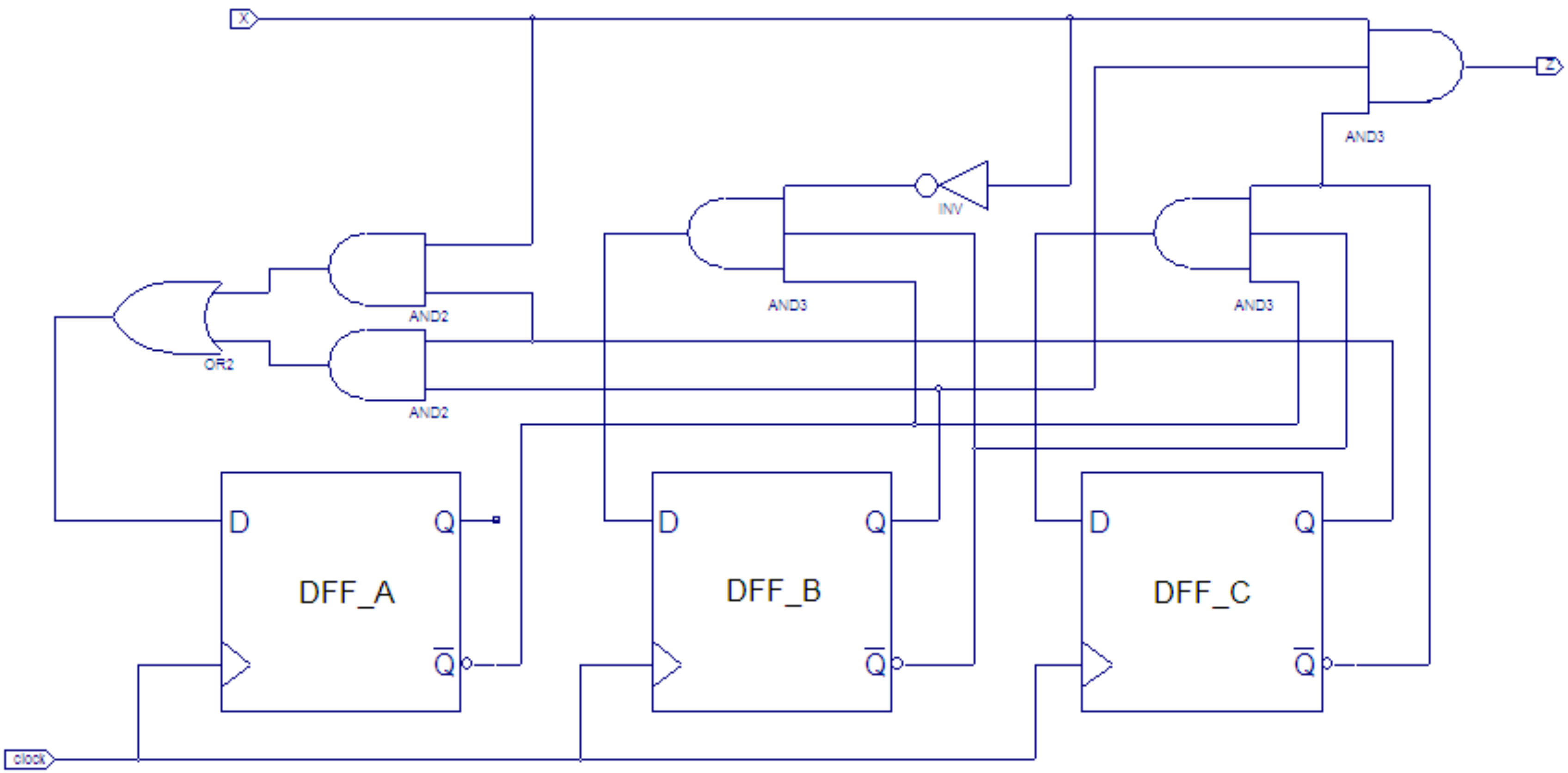
n0 = p2’p1’p0’

Z (X p2 \ p1p0)	00	01	11	10
00	0	0	0	0
01	0	X	X	X
11	0	X	X	X
10	0	0	0	1

Z = p1p0’X

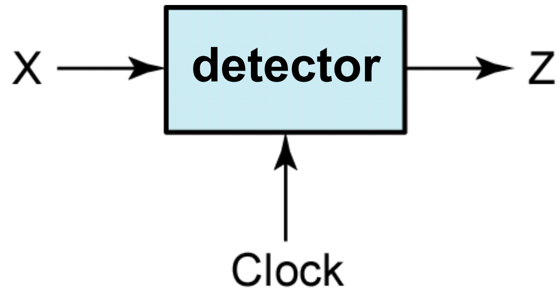
Completed Logic Circuit

- Circuit Implementation of FSM
  - Using D FFs



Alternative Design of the FSM – Moore

- Example: design a non-overlapping sequence detector as Moore FSM



- Z is determined every three bits, Z = 1 at the next edge after desired sequence is detected

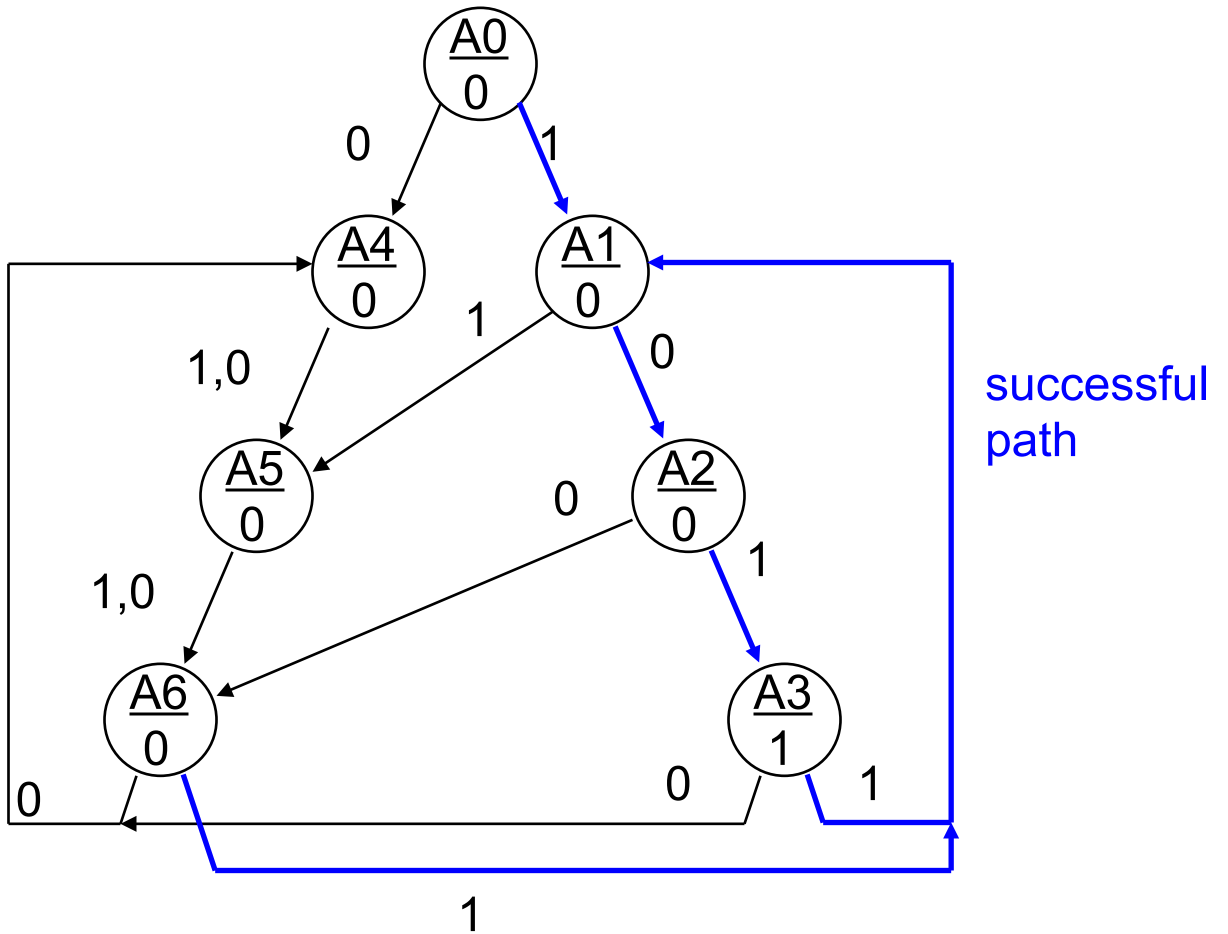
X =	0	0	1	1	0	1	1	0	0	1	0	1	0	1	0
Z =	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

one input: X

one output: Z

Development of State Diagram

- Drawing the state diagram



FSM Optimization – State Reduction

- Two states are equivalent iff their next states and outputs are identical

Present State	Next State		Output
	X = 0	X = 1	
A0	A4	A1	0
A1	A2	A5	0
A2	**A0**	A3	0
A3	A4	A1	1
A4	A5	A5	0
A5	**A0**	**A0**	0
~~A6~~	~~A4~~	~~A1~~	~~0~~

equivalent states

Reduced State Table

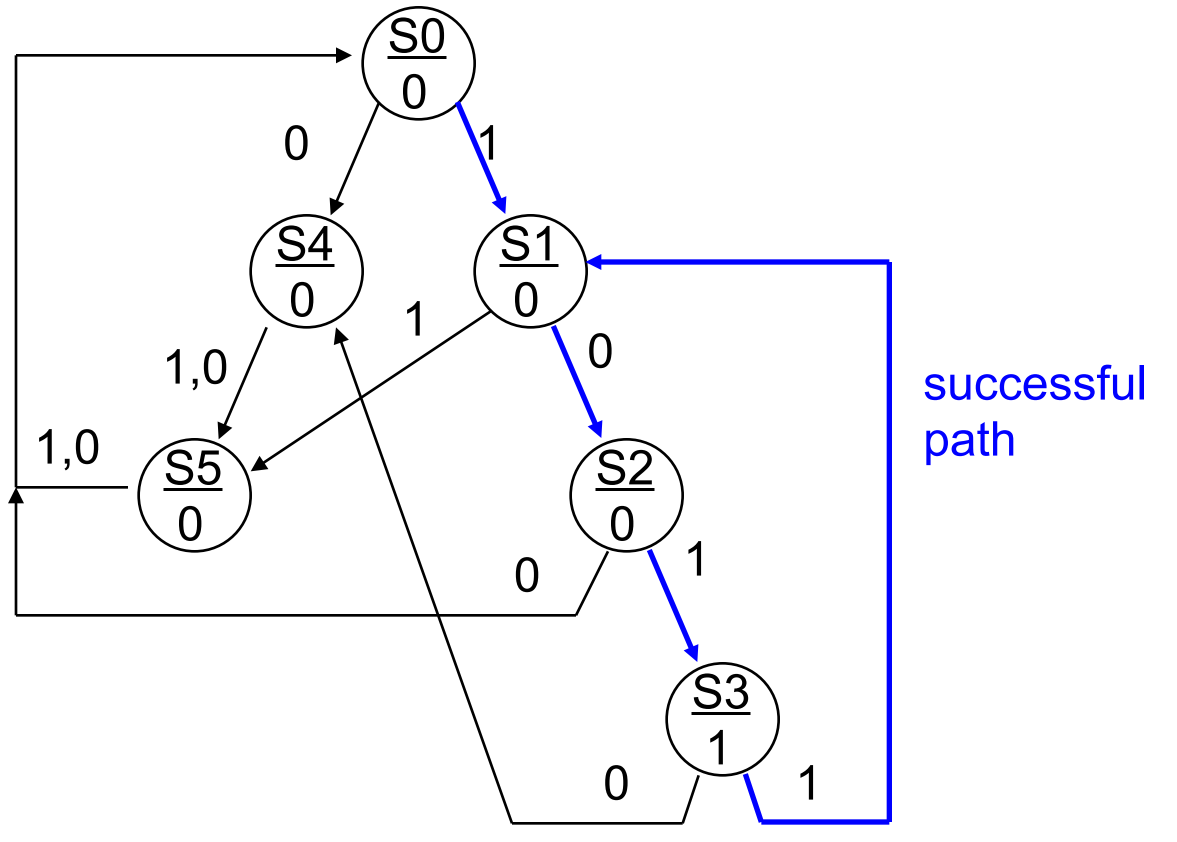
- Reduced state table

Present State	Next State		Output
	X = 0	X = 1	
A0	A4	A1	0
A1	A2	A5	0
A2	A0	A3	0
A3	A4	A1	1
A4	A5	A5	0
A5	A0	A0	0
~~A6~~	~~A4~~	~~A1~~	~~0~~

A0 → S0
A1 → S1
A2 → S2
A3 → S3
A4 → S4
A5 → S5

Present State	Next State		Output
	X=0	X=1	
S0	S4	S1	0
S1	S2	S5	0
S2	S0	S3	0
S3	S4	S1	1
S4	S5	S5	0
S5	S0	S0	0

Reduced State Diagram



State Assignment

- Number of bits of binary number should be enough to represent all the states

S0	S1	S2	S3	S4	S5
000	001	010	011	100	101

Present State	Next State		Output
	X=0	X=1	
000	100	001	0
001	010	101	0
010	000	011	0
011	100	001	1
100	101	101	0
101	000	000	0

In	Present State			Next State			Out
X	p2	p1	p0	n2	n1	n0	Z
0	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0
……				x			
1	0	0	0	0	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	0	0	1	1
1	1	0	0	1	0	1	0
1	1	0	1	0	0	0	0
……				x			

State and Output Equations

In	Present State			Next State			Out
X	p2	p1	p0	n2	n1	n0	Z
0	0	0	0	1	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0
……				x			
1	0	0	0	0	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	0	0	1	1
1	1	0	0	1	0	1	0
1	1	0	1	0	0	0	0
……				x			

n2 Xp2 \ p1p0	00	01	11	10
00	**1**	**0**	**1**	**0**
01	**1**	**0**	**X**	**X**
11	**1**	**0**	**X**	**X**
10	**0**	**1**	**0**	**0**

n2 = p2p0’+p1’p0’X’+
p1p0X’+p2’p1’p0X

n1 Xp2 \ p1p0	00	01	11	10
00	**0**	**1**	**0**	**0**
01	**0**	**0**	**X**	**X**
11	**0**	**0**	**X**	**X**
10	**0**	**0**	**0**	**1**

n1 = p1p0’X+
p2’p1’p0X’

n0 Xp2 \ p1p0	00	01	11	10
00	**0**	**0**	**0**	**0**
01	**1**	**0**	**X**	**X**
11	**1**	**0**	**X**	**X**
10	**1**	**1**	**1**	**1**

n0 = p2p0’+p2’X

Z p2 \ p1p0	00	01	11	10
0	**0**	**0**	**1**	**0**
1	**0**	**0**	**X**	**X**

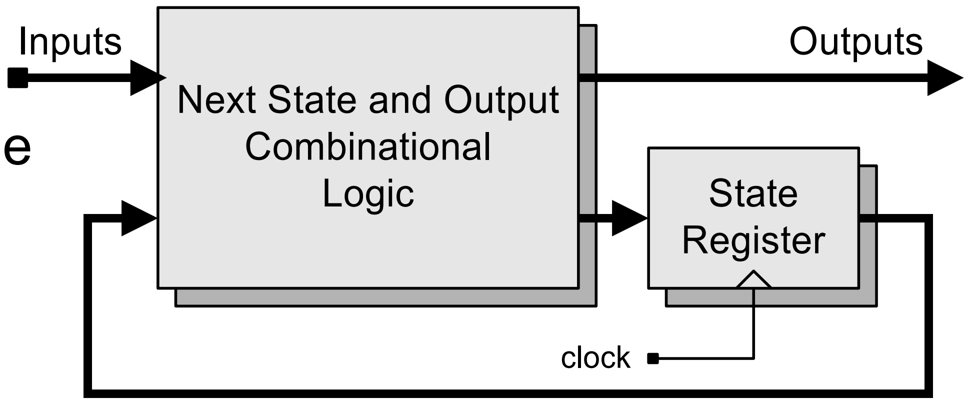
Z = p1p0

Mealy FSM vs. Moore FSM

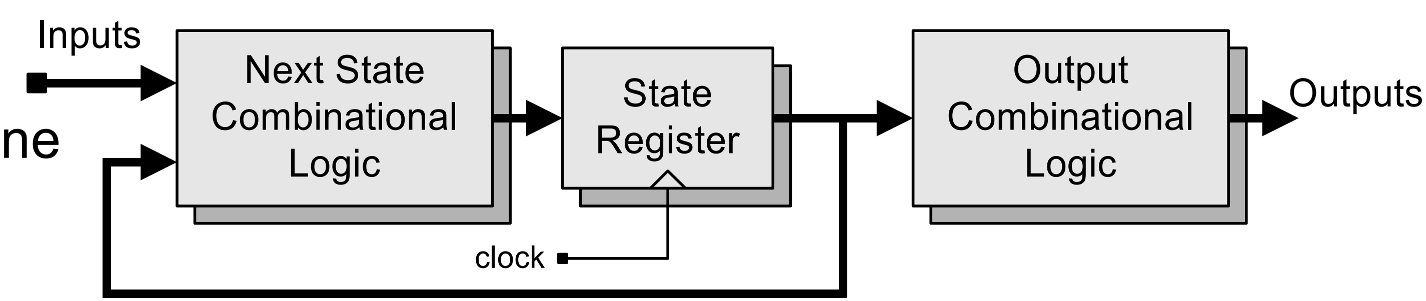
- Output
  - Mealy: depends on both inputs and presents
  - Moore: doesn't depend on inputs
- State Diagram
  - Mealy: less states -> potentially less number of flip-flops
  - Moore: more states than Mealy -> possibly bigger circuit
- Speed of output response to the inputs
  - Mealy: quick, as soon as input changes
  - Moore: as long as one clock cycle delay
- TIMING ISSUE
  - Mealy: asynchronous, may cause serious problem
  - Moore: synchronous, more stable

HDL Modeling of FSM

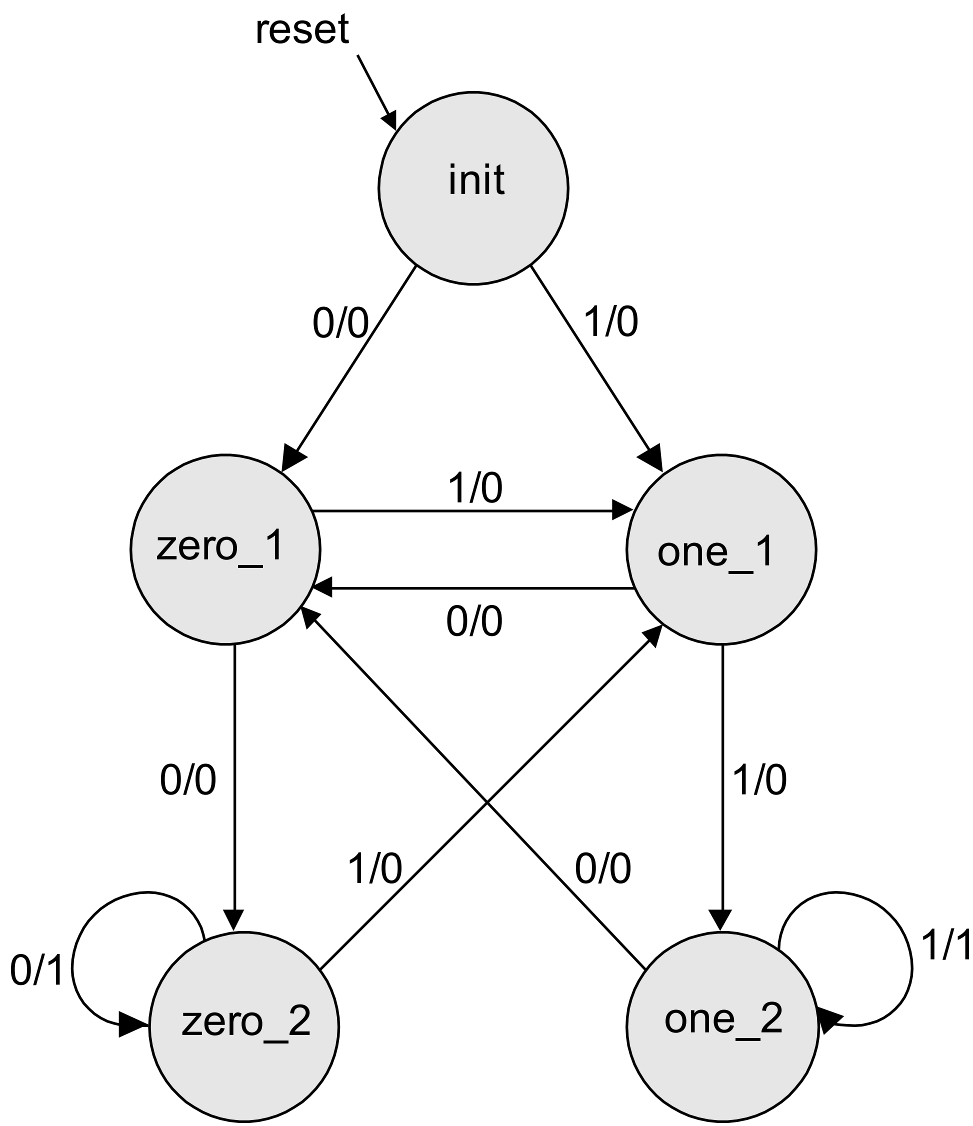
Mealy Machine



Moore Machine



Modeling of FSM – Mealy



```
module seq_det_mealy_1exp (clock, reset, in_bit, out_bit);
  input       clock, reset, in_bit;
  output      out_bit;

  reg [2:0]   curr_state, next_state;

  parameter   init   = 3'b000;
  parameter   zero_1 = 3'b001;
  parameter   one_1  = 3'b010;
  parameter   zero_2 = 3'b011;
  parameter   one_2  = 3'b100;

  always @ (posedge clock or posedge reset)
    if (reset == 1) curr_state <= init;
    else            curr_state <= next_state;

  always @ (curr_state or in_bit)
    case (curr_state)
      init: if (in_bit == 0) next_state <= zero_1; else
            if (in_bit == 1) next_state <= one_1;  else
                             next_state <= init;
```

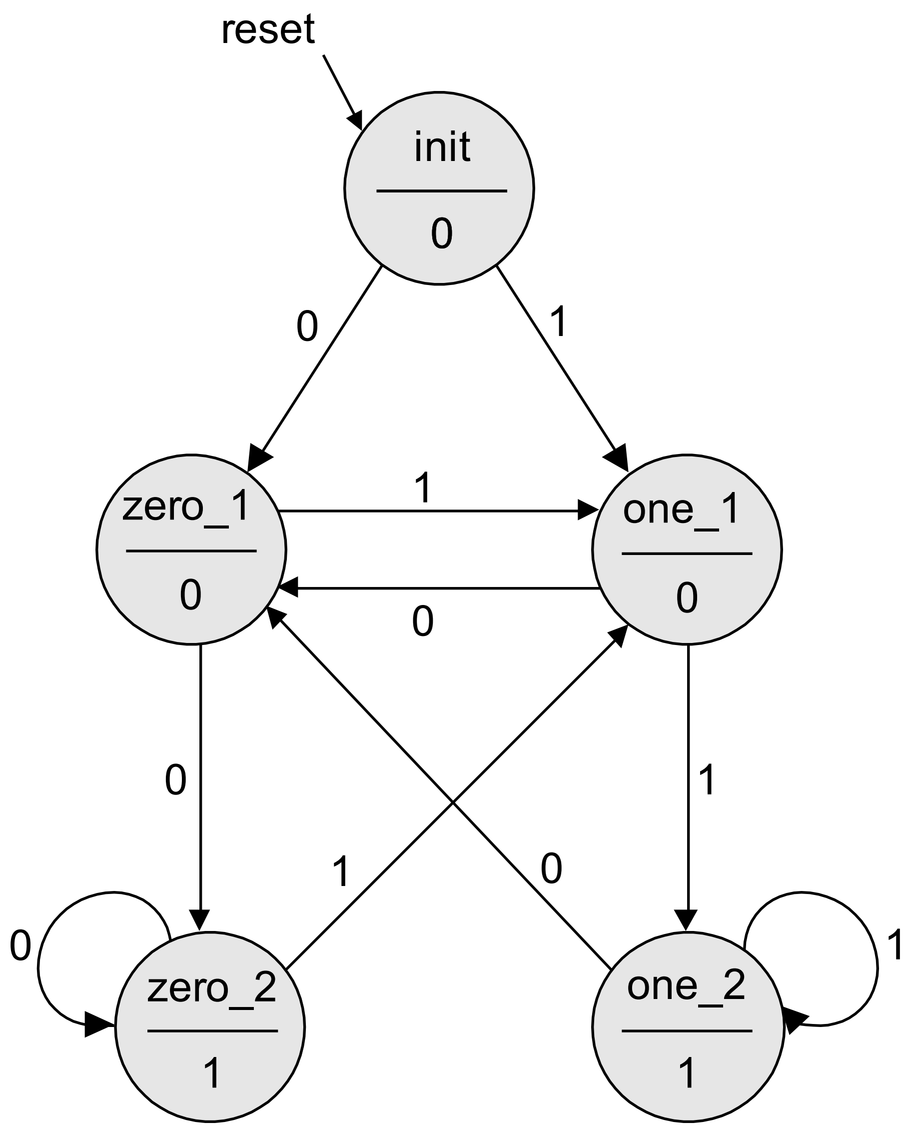
State register

Combinational logic for next state

```
    zero_1: if (in_bit == 0) next_state <= zero_2; else
            if (in_bit == 1) next_state <= one_1;  else
                             next_state <= init;
    zero_2: if (in_bit == 0) next_state <= zero_2; else
            if (in_bit == 1) next_state <= one_1;  else
                             next_state <= init;
    one_1:  if (in_bit == 0) next_state <= zero_1; else
            if (in_bit == 1) next_state <= one_2;  else
                             next_state <= init;
    one_2:  if (in_bit == 0) next_state <= zero_1; else
            if (in_bit == 1) next_state <= one_2;  else
                             next_state <= init;
    default:                 next_state <= init;
  endcase

 assign out_bit = (((curr_state==zero_2)&&(in_bit==0))||
                   ((curr_state==one_2)&&(in_bit==1))) ? 1 : 0;
endmodule
```

Combinational logic for FSM outputs

Modeling of FSM – Moore

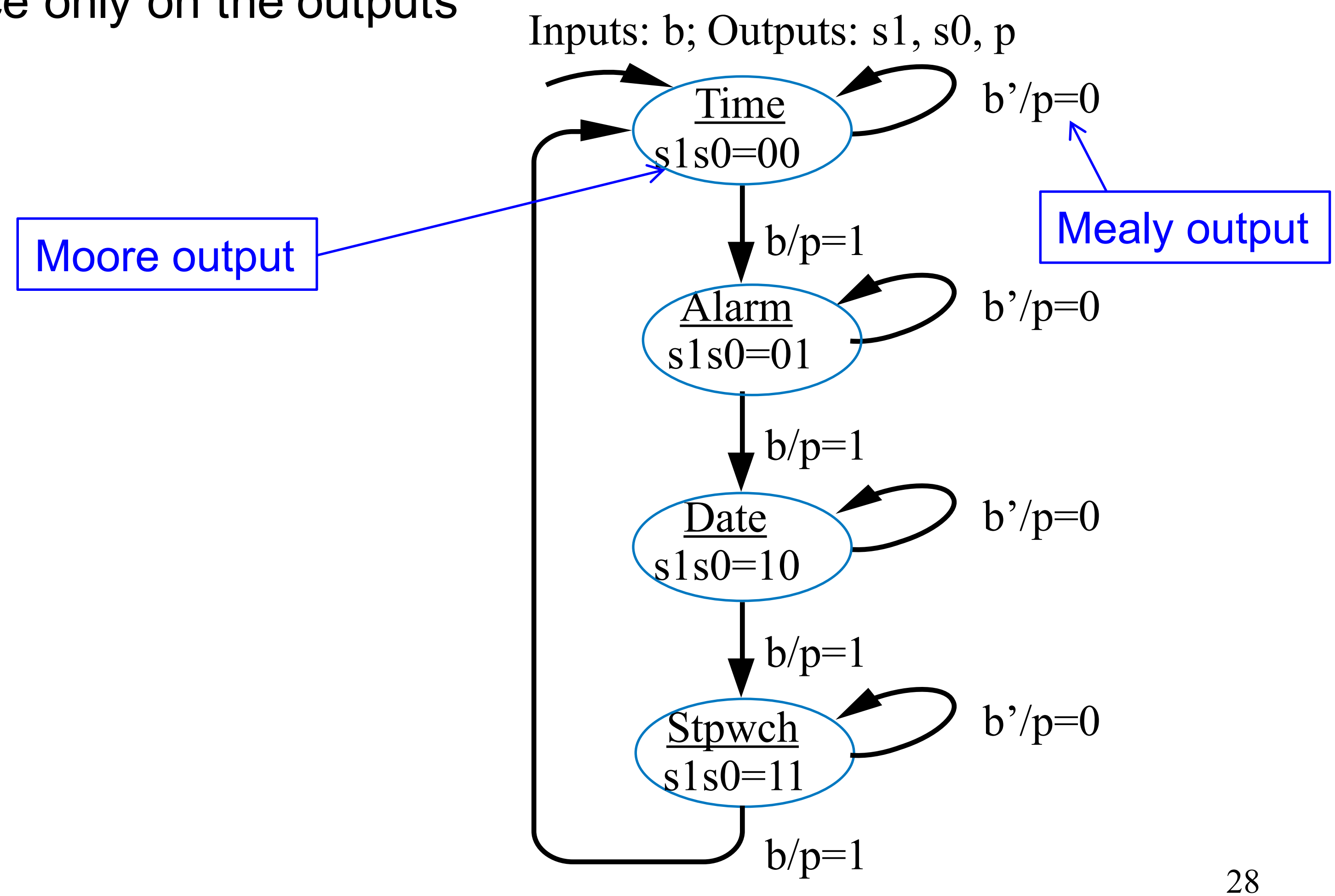


```
... // the same as Mealy

zero_1: if (in_bit == 0) next_state <= zero_2; else
        if (in_bit == 1) next_state <= one_1;  else
                         next_state <= init;
zero_2: if (in_bit == 0) next_state <= zero_2; else
        if (in_bit == 1) next_state <= one_1;  else
                         next_state <= init;
one_1:  if (in_bit == 0) next_state <= zero_1; else
        if (in_bit == 1) next_state <= one_2;  else
                         next_state <= init;
one_2:  if (in_bit == 0) next_state <= zero_1; else
        if (in_bit == 1) next_state <= one_2;  else
                         next_state <= init;
default:                 next_state <= init;
endcase

assign out_bit = ((curr_state==zero_2)||(curr_state==one_2))
                 ? 1 : 0;

endmodule
```

Combinational logic for FSM outputs

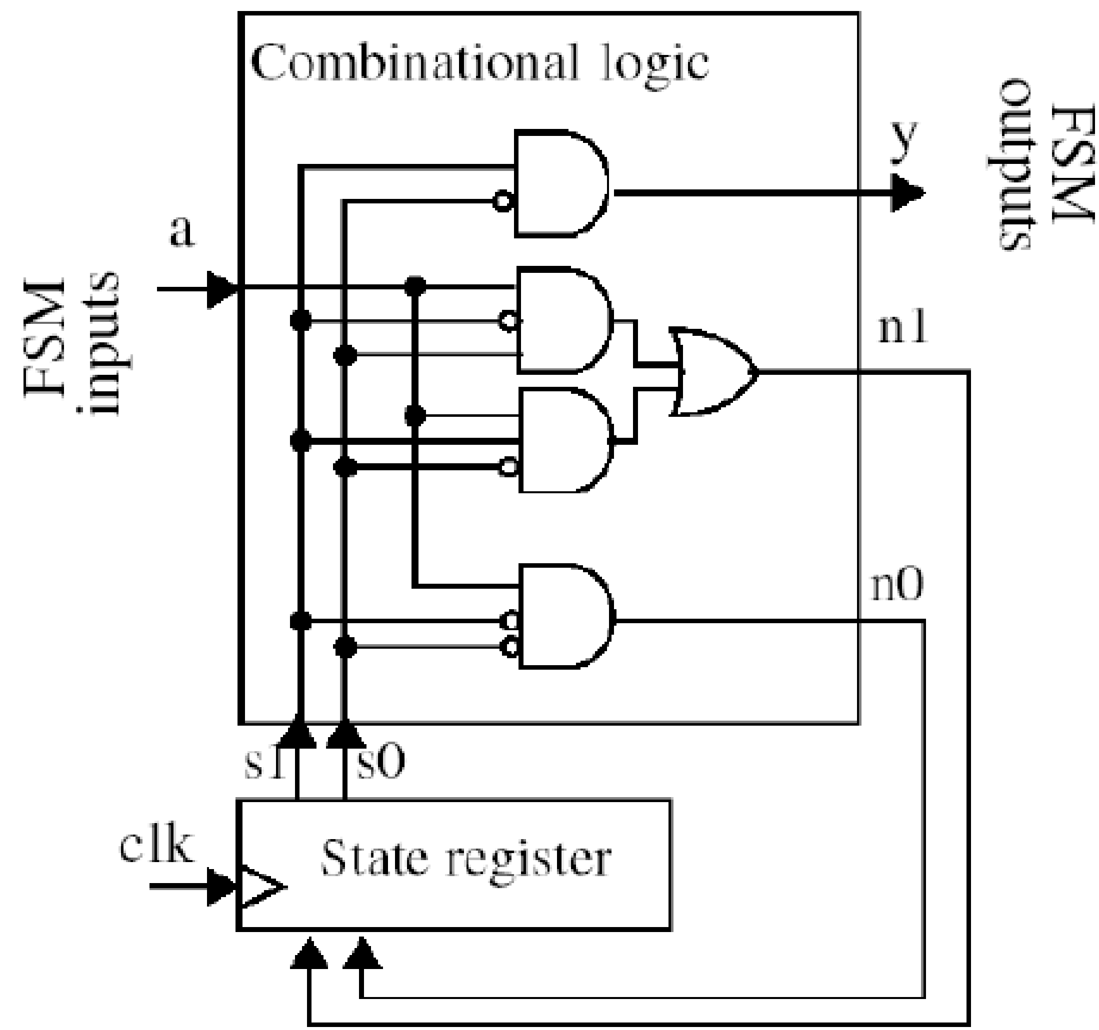
Mealy and Moore can be Combined

- May be combined in same FSM
  - Difference only on the outputs



FSM Reverse Engineering

- Given a circuit of FSM, figure out the behavior
  - Mealy or Moore?
  - How many states?
  - Logic for next state?
  - State table?
  - State diagram?

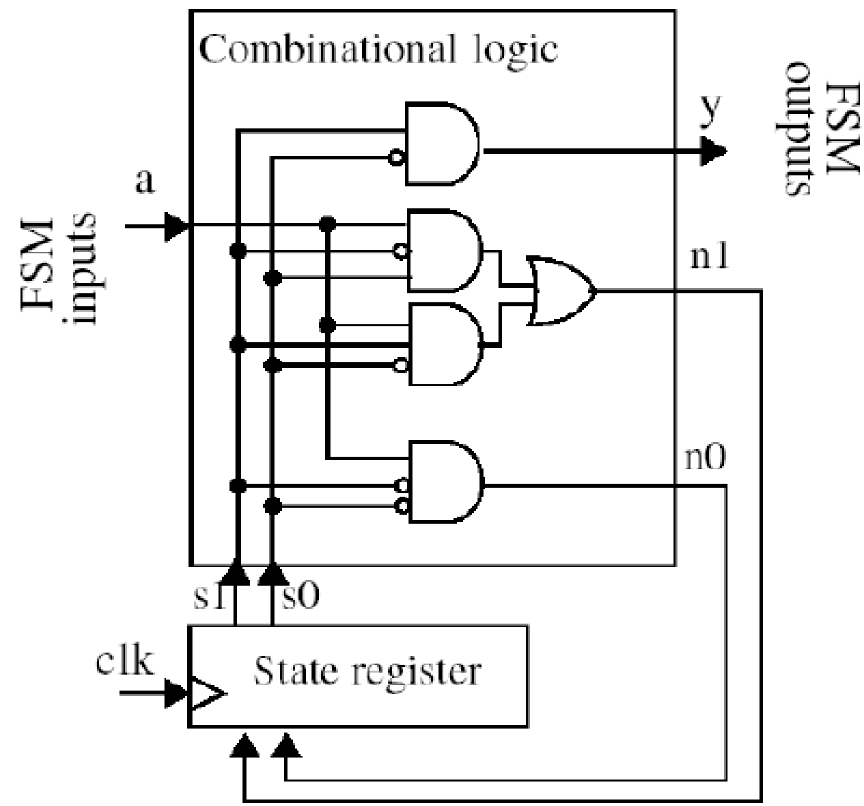


FSM Reverse Engineering

- Given a circuit of FSM, figure out the behavior
  - $y = s1 \cdot s0'$, Moore!
  - 2 bit state register, 4 states
  - Logic for next state:

    $n1 = a \cdot s1' \cdot s0 + a \cdot s1 \cdot s0'$

    $n0 = a \cdot s1' \cdot s0'$
  - State table?
  - State diagram?



FSM Reverse Engineering

- Given a circuit of FSM, figure out the behavior
  - $y = s1 \cdot s0'$, Moore!
  - 2 bit state register, 4 states
  - Logic for next state:

    $n1 = a \cdot s1' \cdot s0 + a \cdot s1 \cdot s0'$

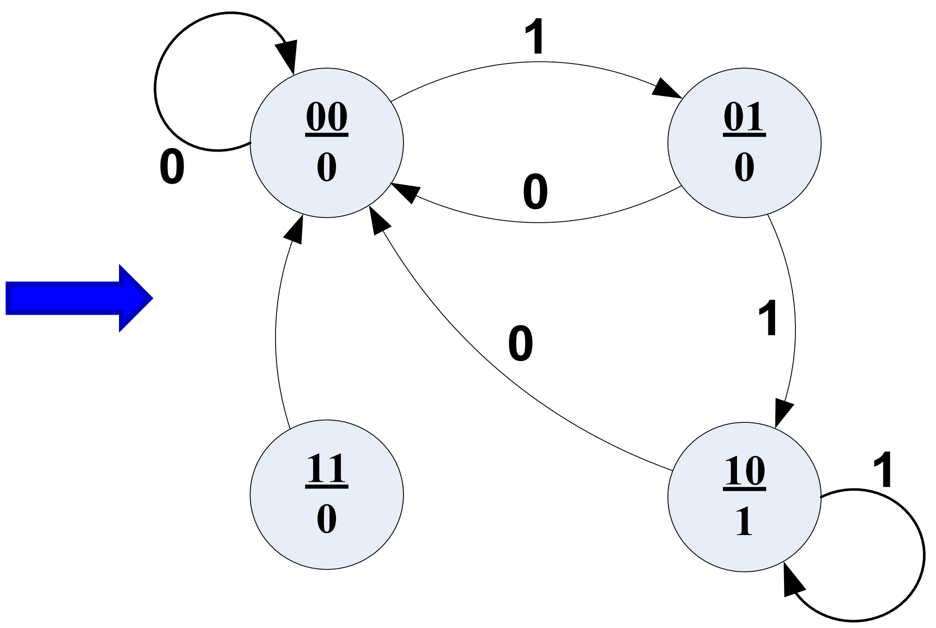
    $n0 = a \cdot s1' \cdot s0'$
  - State table:
  - State diagram?

In	P. State		N. State		Out
a	s1	s0	n1	n0	y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	0	0	0

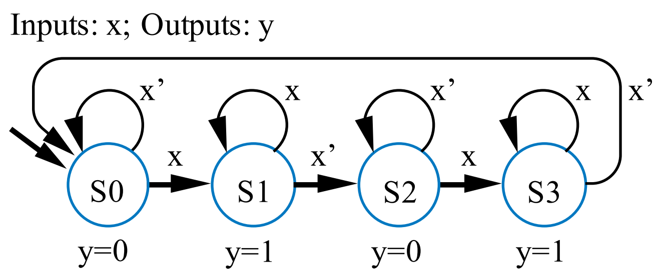
FSM Reverse Engineering

- Given a circuit of FSM, figure out the behavior
  - State diagram

In	P. State		N. State		Out
a	s1	s0	n1	n0	y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	0	0	0



Another Method for State Reduction



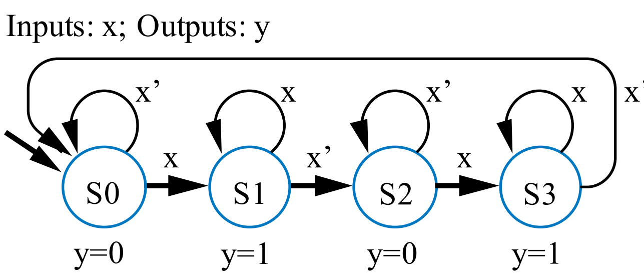
Example on the first slide

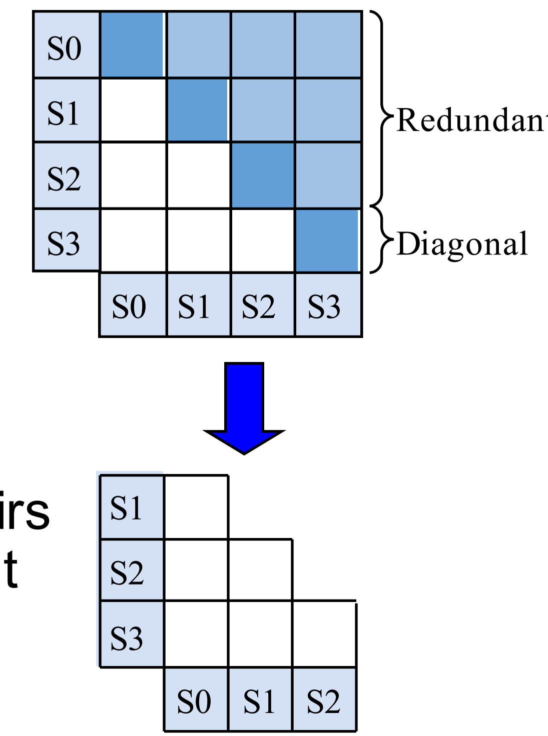
X	P.S.	N.S.	Z
0	S0	S0	0
1	S0	S1	0
0	S1	S2	1
1	S1	S1	1
0	S2	S2	0
1	S2	S3	0
0	S3	S0	1
1	S3	S3	1

Can't reduce more,
No equivalent states

State Reduction with Implication Tables

- State reduction through state table inspection isn't optimal
- A more methodical approach – **Implication Tables**
- Example:



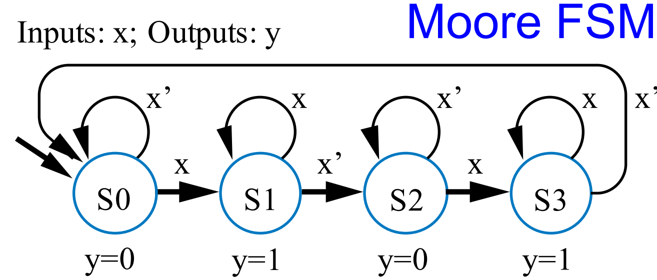


  - To compare every pair of states, construct a table of *state pairs*
  - Remove redundant state pairs, and state pairs along the diagonal since a state is equivalent to itself

State Reduction with Implication Tables

- Mark (with an X) state pairs with different outputs as non-equivalent:
  - (**S1**,**S0**): At **S1**, *y*=1 and at **S0**, *y*=0. So **S1** and **S0** are non-equivalent.
  - (**S2**, **S0**): At **S2**, *y*=0 and at **S0**, *y*=0. So we don't mark **S2** and **S0** now.
  - (**S2**, **S1**): Non-equivalent
  - (**S3**, **S0**): Non-equivalent
  - (**S3**, **S1**): Don't mark
  - (**S3**, **S2**): Non-equivalent
- Unmarked pairs (**S2, S0**) and (**S3, S1**) might be equivalent, but only if their next states are equivalent

Moore FSM



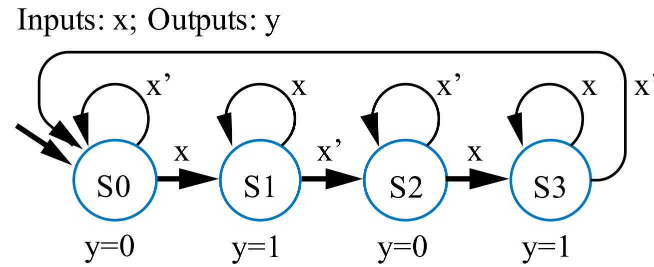
S1	X		
S2		X	
S3	X		X
	S0	**S1**	**S2**

State Reduction with Implication Tables

- List next states of unmarked state pair's corresponding to every combination of inputs

  – (**S2**, **S0**)
    - From **S2**, when *x*=1 go to **S3**
      From **S0**, when *x*=1 go to **S1**
      So add (**S3**, **S1**) as a next state pair
    - From **S2**, when *x*=0 go to **S2**
      From **S0**, when *x*=0 go to **S0**
      So add (**S2**, **S0**) as a next state pair

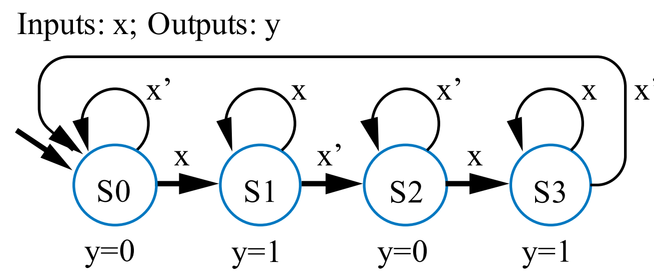
  – (**S3**, **S1**)
    - By a similar process, add the next state pairs (**S3**, **S1**) and (**S0**, **S2**)



S1			
S2	(S3, S1) (S2, S0)		
S3		(S3, S1) (S0, S2)	
	S0	**S1**	**S2**

State Reduction with Implication Tables

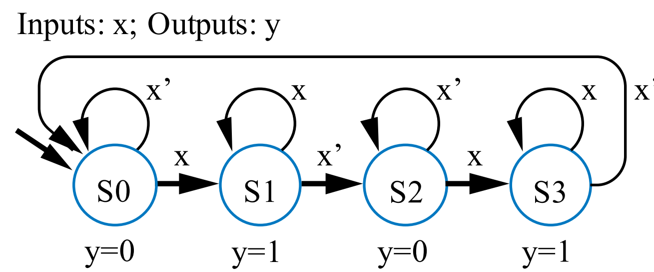
- Mark (with X) the state pair if one of its next state pairs is marked (non-equivalent)
  - (**S2**, **S0**)
    - Next state pair (**S3**, **S1**) is not marked
    - Next state pair (**S2**, **S0**) is not marked
    - So we do nothing and move on
  - (**S3**, **S1**)
    - Next state pair (**S3**, **S1**) is not marked
    - Next state pair (**S0**, **S2**) is not marked
    - So we do nothing and move on

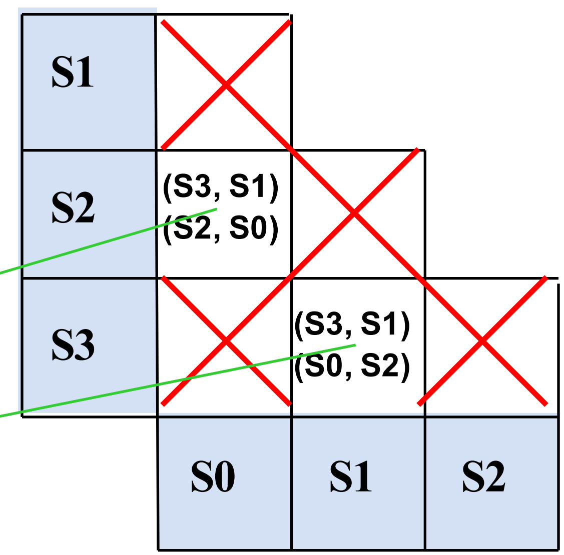


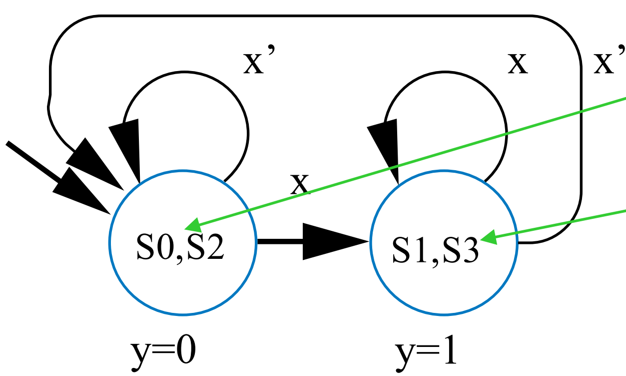
S1	X		
S2	(**S3**, **S1**) (**S2**, **S0**)	X	
S3	X	(**S3**, **S1**) (**S0**, **S2**)	X
	S0	**S1**	**S2**

State Reduction with Implication Tables

- Made a *pass* through the entire implication table
- Make additional passes until no change occurs
- Implied by the table, unmarked state pairs are equivalent





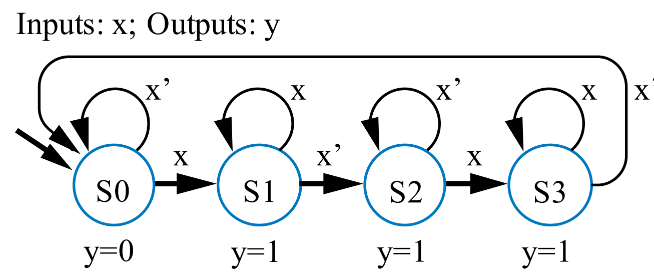


State Reduction with Implication Tables

	Step	Description
1	*Mark state pairs having different outputs as nonequivalent*	States having different outputs obviously cannot be equivalent.
2	*For each unmarked state pair, write the next state pairs for the same input values*	
3	*For each unmarked state pair, mark state pairs having nonequivalent next-state pairs as nonequivalent. Repeat this step until no change occurs, or until all states are marked.*	States with nonequivalent next states for the same input values can't be equivalent. Each time through this step is called a ***pass***.
4	*Merge remaining state pairs*	Remaining state pairs must be equivalent.

State Reduction Example

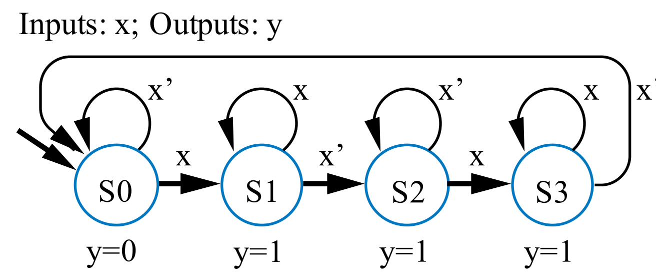
- Given FSM on the right
  - **Step 1:** Mark state pairs having different outputs as nonequivalent



S1	X		
S2	X		
S3	X		
	S0	**S1**	**S2**

State Reduction Example

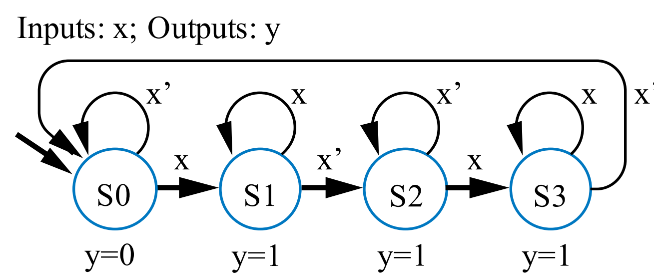
- Given FSM on the right
  - **Step 1:** Mark state pairs having different outputs as nonequivalent
  - **Step 2:** For each unmarked state pair, write the next state pairs for the same input values



S1			
S2		(S2, S2) (S3, S1)	
S3		(S0, S2) (S3, S1)	(S0, S2) (S3, S3)
	S0	**S1**	**S2**

State Reduction Example

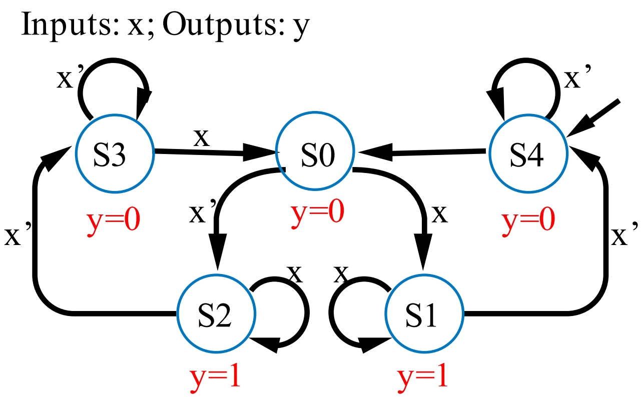
- Given FSM on the right
  - **Step 1:** Mark state pairs having different outputs as nonequivalent
  - **Step 2:** For each unmarked state pair, write the next state pairs for the same input values
  - **Step 3:** For each unmarked state pair, mark state pairs having nonequivalent next state pairs as nonequivalent.
    - Repeat this step until no change occurs, or until all states are marked.
  - **Step 4:** Merge remaining state pairs



S1			
S2		(S2, S2) (S3, S1)	
S3		(S0, S2) (S3, S1)	(S0, S2) (S3, S3)
	S0	**S1**	**S2**

All state pairs are marked – there are no equivalent state pairs to merge

A Larger State Reduction Example



S1				
S2		(S3,S4) (S2,S1)		
S3	(S3,S2) (S0,S1)			
S4	(S4,S2) (S0,S1)			(S4,S3) (S0,S0)
	S0	S1	S2	S3

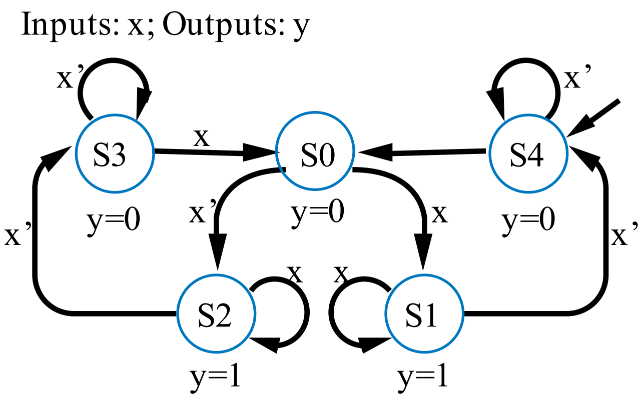
– **Step 1:** Mark state pairs having different outputs as nonequivalent

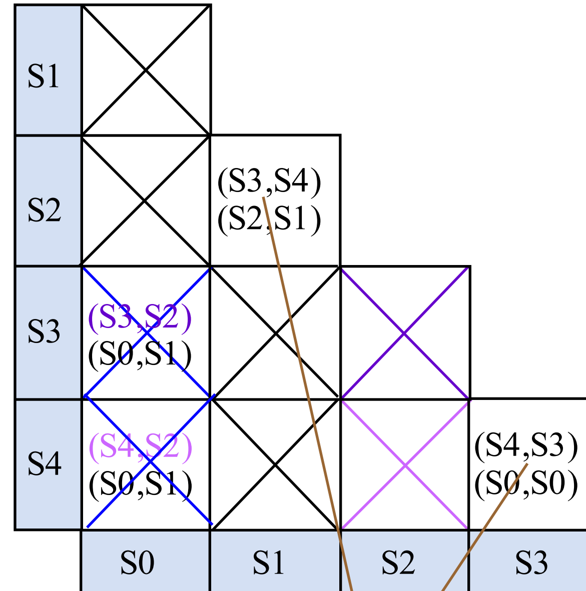
– **Step 2:** For each unmarked state pair, write the next state pairs for the same input values

– **Step 3:** For each unmarked state pair, mark state pairs having nonequivalent next state pairs as nonequivalent.
  - Repeat this step until no change occurs, or until all states are marked.

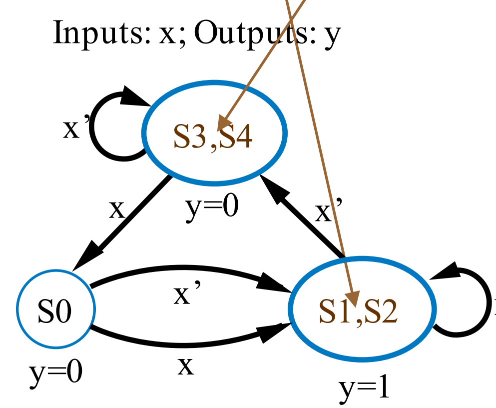
– **Step 4:** Merge remaining state pairs

A Larger State Reduction Example



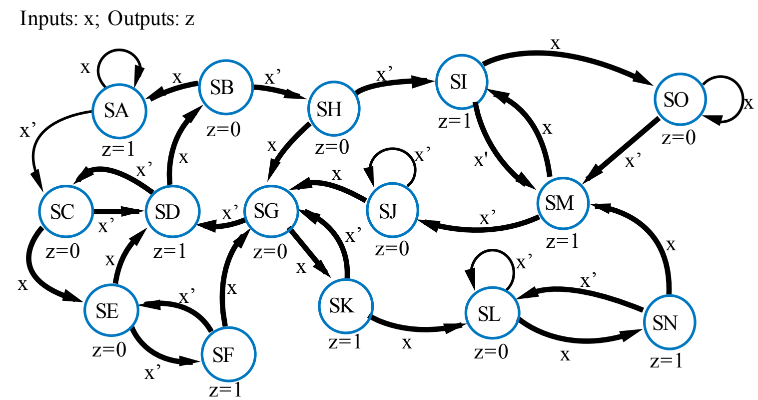


- **Step 1:** Mark state pairs having different outputs as nonequivalent
- **Step 2:** For each unmarked state pair, write the next state pairs for the same input values
- **Step 3:** For each unmarked state pair, mark state pairs having nonequivalent next state pairs as nonequivalent.
    - Repeat this step until no change occurs, or until all states are marked.
- **Step 4:** Merge remaining state pairs



Complex FSM

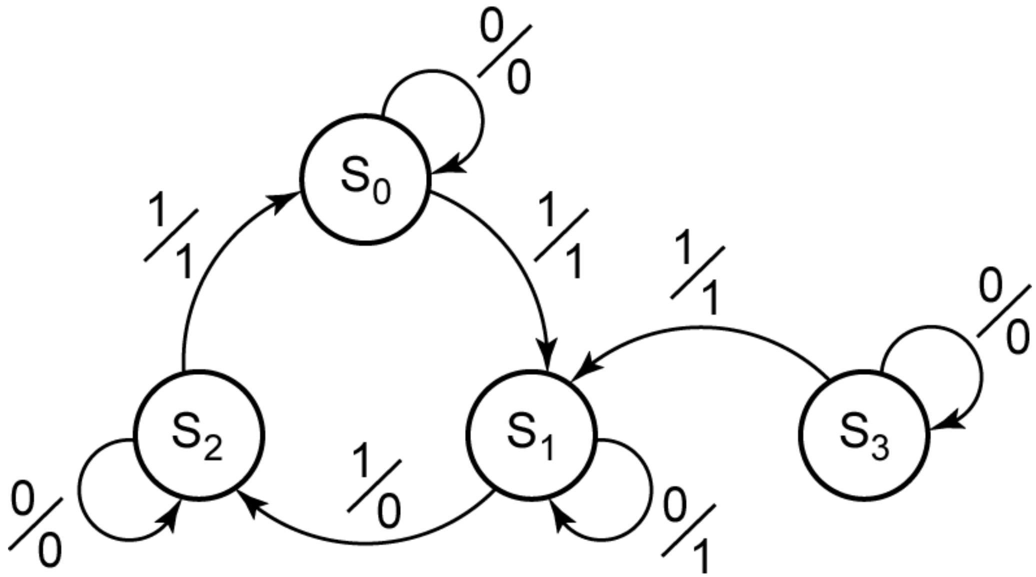
- Automation needed
  - Table for large FSM too big for humans to work with
    - *n* inputs: each state pair can have 2^n next state pairs.
    - 4 inputs → 2^4=16 next state pairs



  - 100 states would have table with 100*100=100,000 state pairs cells
  - State reduction typically automated

Mealy FSM Reduction with Implication Table

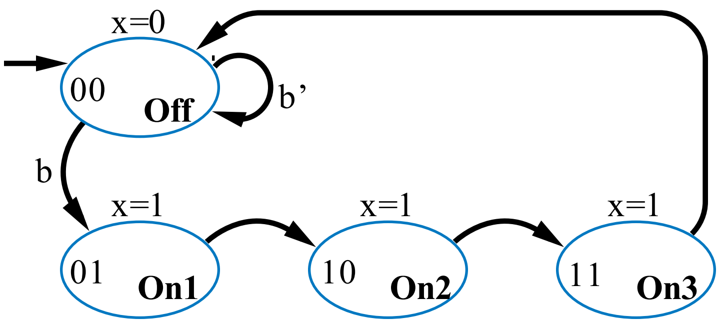
- Example:

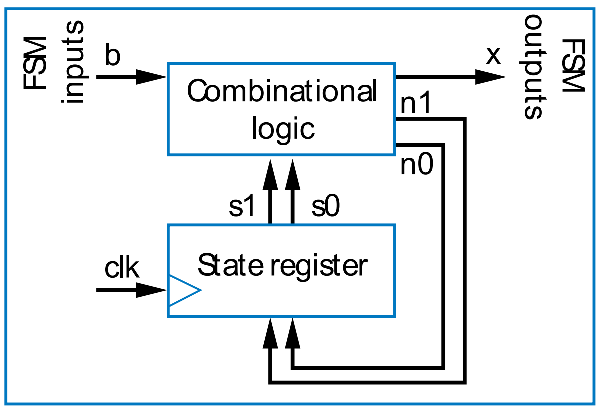


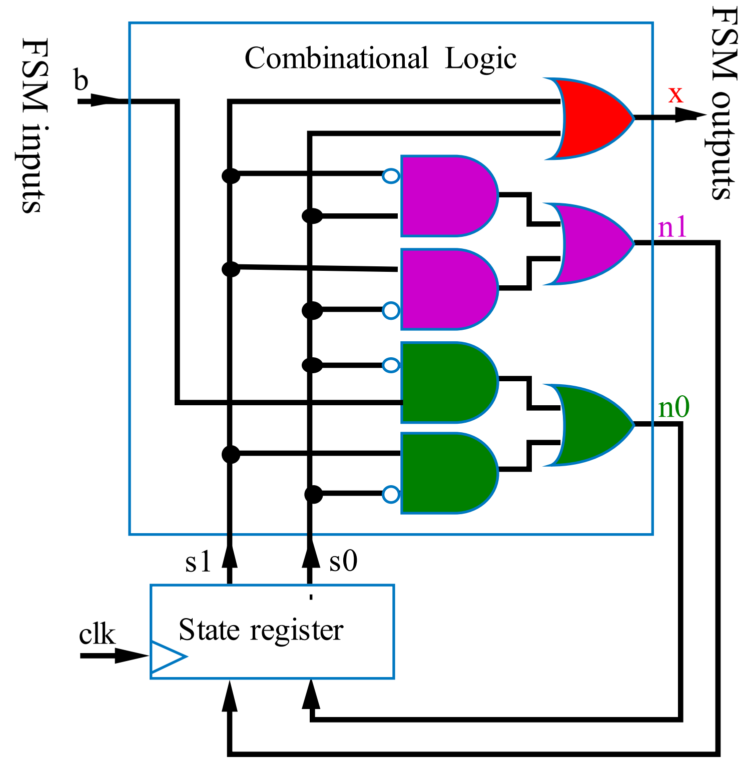
- Should have both next state pairs and output pairs in a cell for comparison

Optimization by State Encoding

Inputs: b; Outputs: x







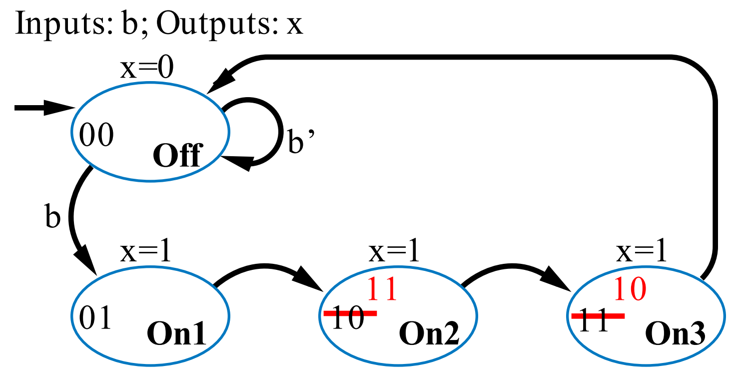
$x = s1 + s0$

$n1 = s1's0 + s1s0'$

$n0 = s0'b + s1s0'$

Optimization by State Encoding

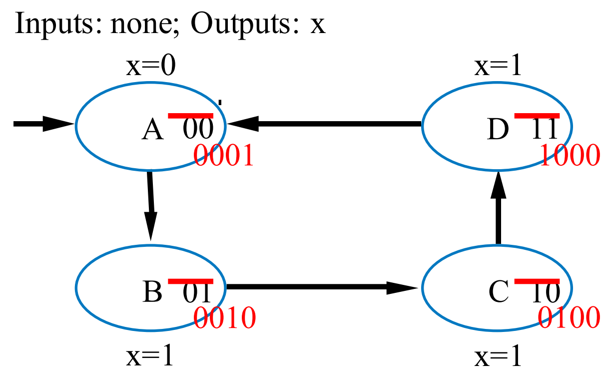
- ***Encoding***: Assigning a unique bit representation to each state
- Different encodings may optimize size, or tradeoff between size and speed
- Consider push button example
  - Regular binary encoding: **14** gate inputs
  - Try alternative encoding:
    On2= 11; On3=10
    - x = s1 + s0
    - n1 = s0
    - n0 = s1'b + s1's0
    - Only **8** gate inputs
  - Known as Gray Code



	Inputs			Outputs		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	~~0~~ 1
	0	1	1	1	1	~~0~~ 1
On2	1	~~0~~ 1	0	1	1	~~1~~ 0
	1	~~0~~ 1	1	1	1	~~1~~ 0
On3	1	~~1~~ 0	0	1	0	0
	1	~~1~~ 0	1	1	0	0

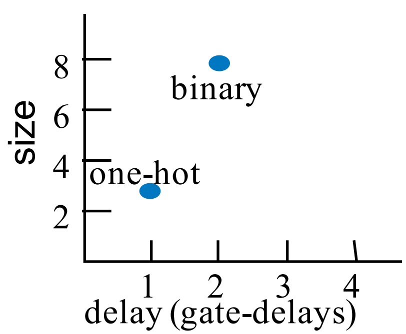
State Encoding: One-Hot Encoding

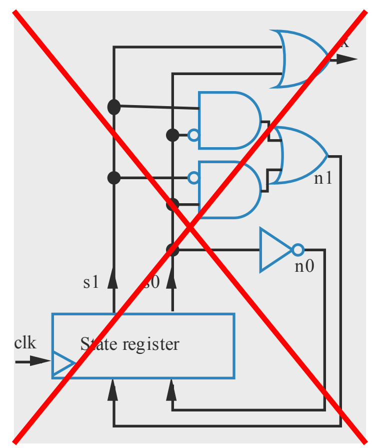
- ***One-hot encoding***
  - One bit per state – a bit being ‘1’ corresponds to a particular state
  - For A, B, C, D: A: 0001, B: 0010, C: 0100, D: 1000
- Example: FSM that outputs 0, 1, 1, 1
  - Equations if one-hot encoding:
    - n3 = s2; n2 = s1; n1 = s0; x = s3 + s2 + s1
  - Fewer gates and only one level of logic – less delay than two levels, so faster clock frequency

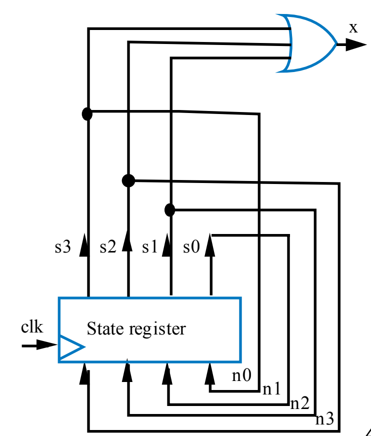


	Inputs		Outputs		
	s1	s0	n1	n0	x
A	0	0	0	1	0
B	0	1	1	0	1
C	1	0	1	1	1
D	1	1	0	0	1

	Inputs				Outputs				
	s3	s2	s1	s0	n3	n2	n1	n0	x
A	0	0	0	1	0	0	1	0	0
B	0	0	1	0	0	1	0	0	1
C	0	1	0	0	1	0	0	0	1
D	1	0	0	0	0	0	0	1	1

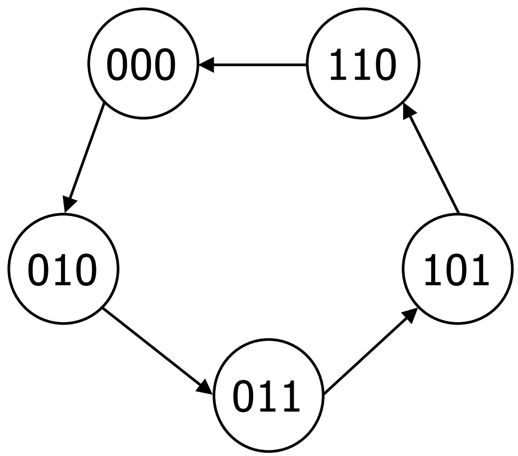






Optimization by Self-Starting FSM

- Given an FSM



Present State			Next State		
p2	p1	p0	n2	n1	n0
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	X	X	X
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	X	X	X

n2

p2 \ p1p0	00	01	11	10
0	**0**	**X**	**1**	**0**
1	**X**	**1**	**X**	**0**

$n2 = p0$

n1

p2 \ p1p0	00	01	11	10
0	**1**	**X**	**0**	**1**
1	**X**	**1**	**X**	**0**

$n1 = p1' + p2'p0'$

n0

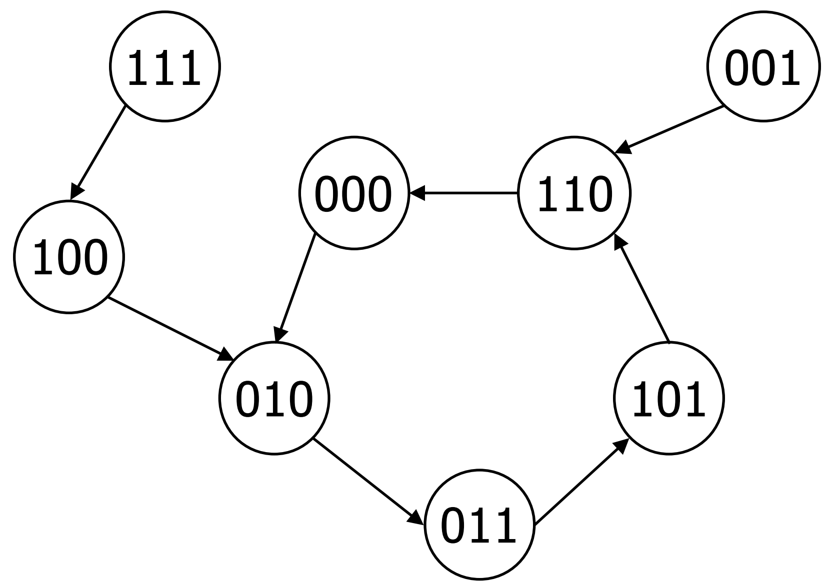
p2 \ p1p0	00	01	11	10
0	**0**	**X**	**1**	**1**
1	**X**	**0**	**X**	**0**

$n0 = p2'p1$

Self-Starting FSM

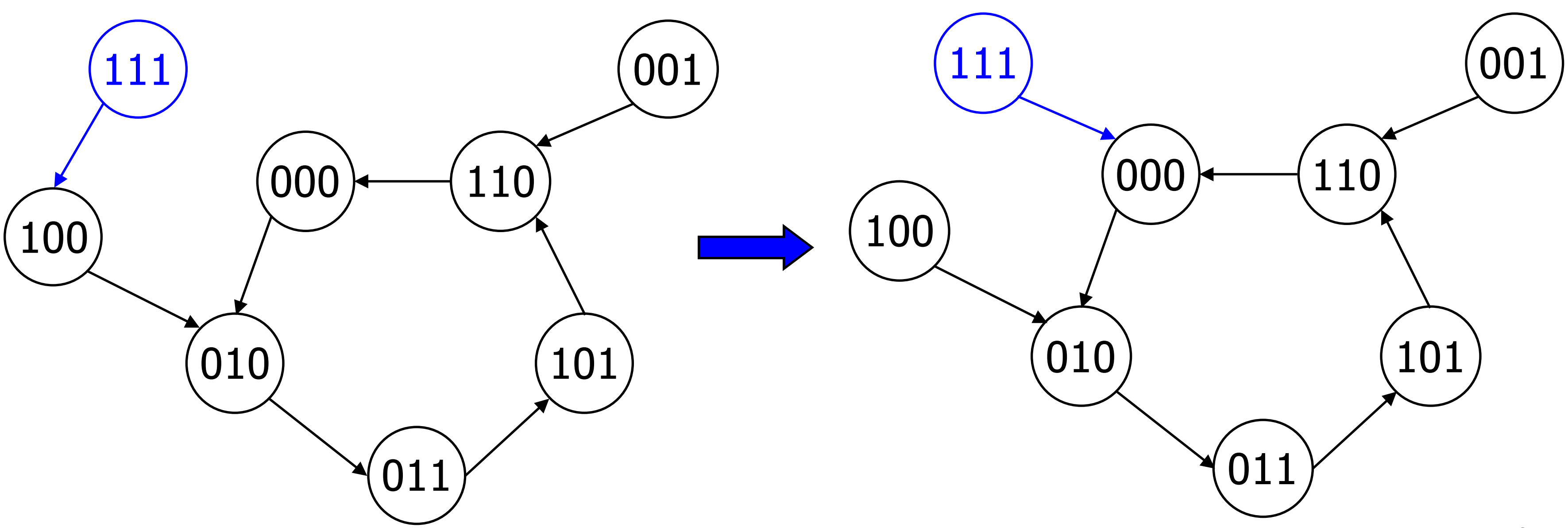
- Start-up States
  - At power-up, FSM may be in an unused or invalid state
  - Designer must guarantee it (eventually) enters a valid state
- Self-starting Solution
  - Design the FSM so that invalid states eventually go to a valid state
  - May limit exploitation of don't cares
- With current design, unused states go:
  - 001 → 110
  - 100 → 010
  - 111 → 100

n2 = p0
n1 = p1’ + p2’p0’
n0 = p2’p1



Self-Starting FSM

- If in case an unused state does not come back to the valid states by the current design
  - Designer should bring it back to a valid state
  - Update the state table to explicitly specify the next state
  - Update equations
- Example: Let the FSM recover from state 111 faster



Self-Starting FSM

- Update state table and equations

Present State p2	p1	p0	Next State n2	n1	n0
0	0	0	0	1	0
0	0	1	X	X	X
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	X	X	X
1	0	1	1	1	0
1	1	0	0	0	0
1	**1**	**1**	**0**	**0**	**0**

n2

p2 \ p1p0	00	01	11	10
0	0	X	1	0
1	X	1	0	0

n2 = p1’p0 + p2’p0

n1

p2 \ p1p0	00	01	11	10
0	1	X	0	1
1	X	1	0	0

n1 = p1’ + p2’p0’

n0

p2 \ p1p0	00	01	11	10
0	0	X	1	1
1	X	0	0	0

n0 = p2’p1

Summary: FSM Design Procedure

1. From the given problem statement, construct a state diagram (Mealy or Moore)
2. Derive a state table from the state diagram
3. Reduce the number of the states by eliminating duplicate states
4. Represent each state by state encoding (binary, one-hot, …)
5. Redraw the reduced state table (truth table)
6. Determine FSM architecture
7. Realize and simplify the next state equations and output equations
8. Check the completeness of the design, make sure the resulted FSM is a self-starting FSM
9. Bring back any unused state that does not come back to a valid state by current design and update state table and equations
10. Check your design by signal tracing, computer simulation, or hardware testing