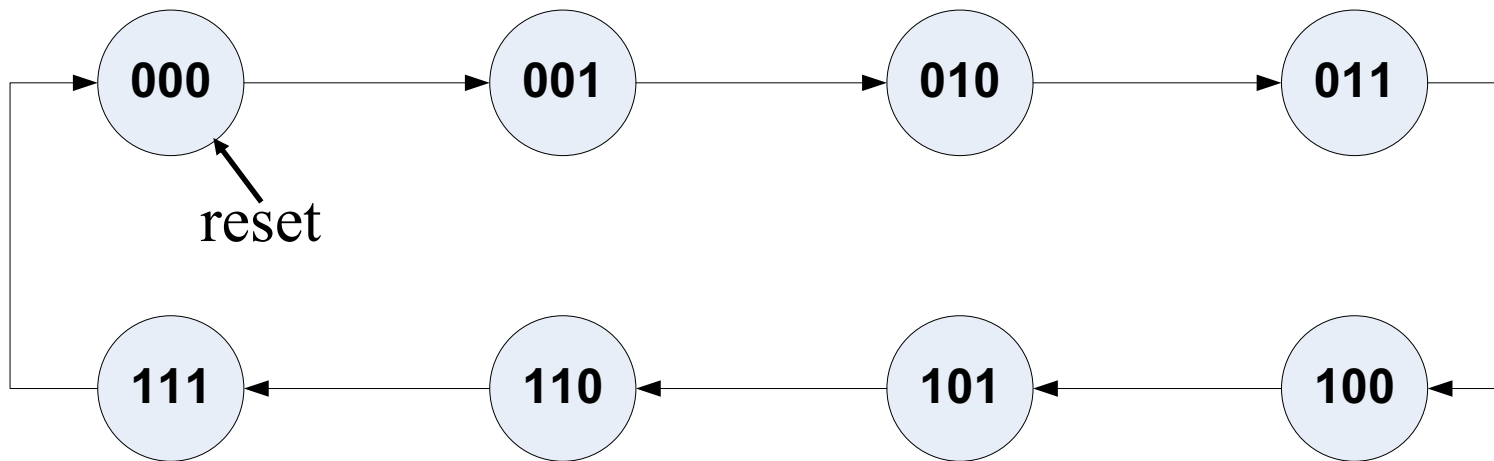# Topic 10

Finite State Machine

Recall Synchronous Binary Counter

- A circuit that changes its value (state) at every clock edge
- The counting sequence describes the behavior of the circuit

000 → 001 → 010 → 011 → 100 → 101 → 110 → 111 → (back to 000)

reset → 000

Describing Behavior of Sequential Circuit

- Finite-State Machine (FSM)
  - A way to describe **desired behavior** of a sequential circuit
  - Consists of a set of states, transitions between states, and maybe inputs and outputs
    - *present state*: currently happening
    - *next state*: next to happen
  - Example: Toggle output x every clock cycle
    - Two states: "Off" and "On"
    - Corresponding outputs: x=0 or 1
    - No input
    - Transition from Off to On, or On to Off, on rising clock edge
    - Arrow with no starting state points to initial state

Outputs: x

State Diagram


3
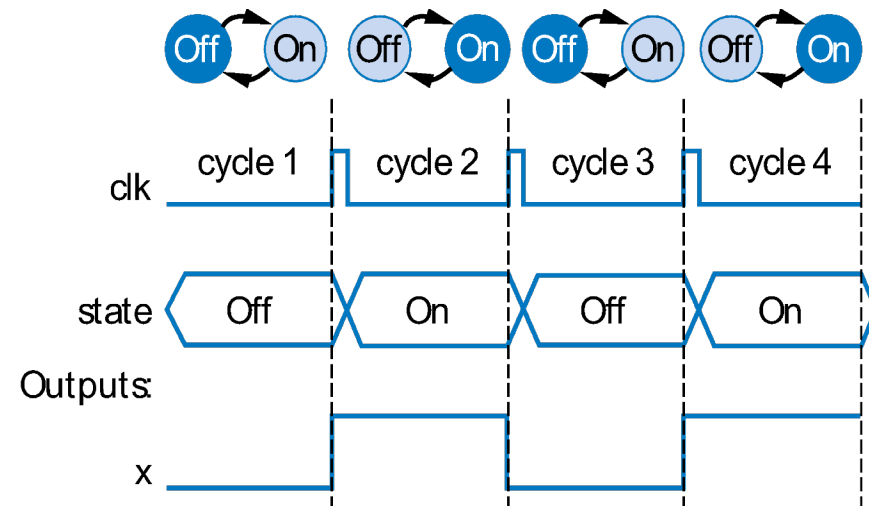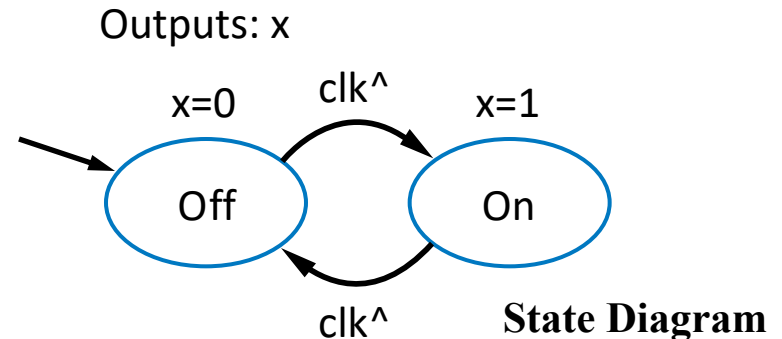
Example: Output Special Pattern

- Want a circuit to output
  0, 1, 1, 1, 0, 1, 1, 1, ...
  - One bit at a time
  - Each bit for one clock cycle
- Can be described as FSM
  - Four states
    - Each state corresponds to an output, 0, 1, 1, 1
    - Then repeat
  - Transition on rising clock edge to next state

Outputs: x

Where the circuit starts

4
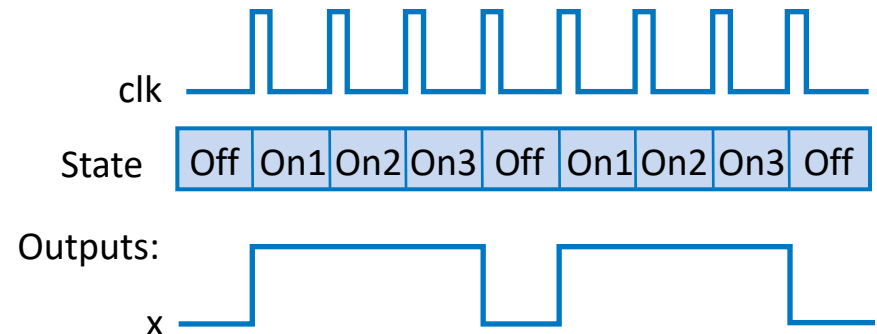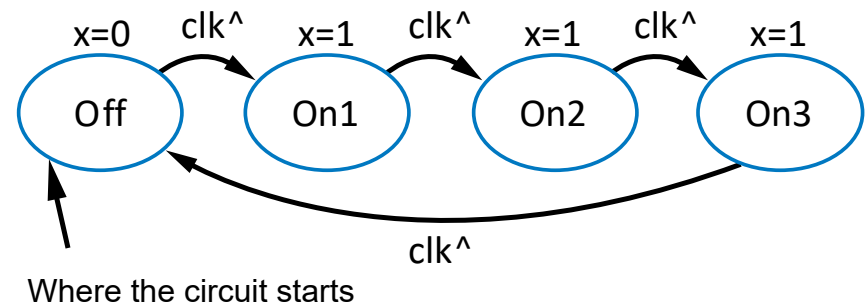
Example: FSM with Input

- b is a push button, output x to stay on for exactly 3 clock cycles no matter how long b is pushed

- Wait in "Off" state while b is 0 (b')

- When b is 1 (and rising clock edge), transition to On1
  - Sets x=1
  - On next two clock edges, transition to On2, then On3, which also set x=1

- So x=1 for three cycles after button pressed

- Potential issue: if button b stays on, what will happen?

Inputs: b; Outputs: x


5
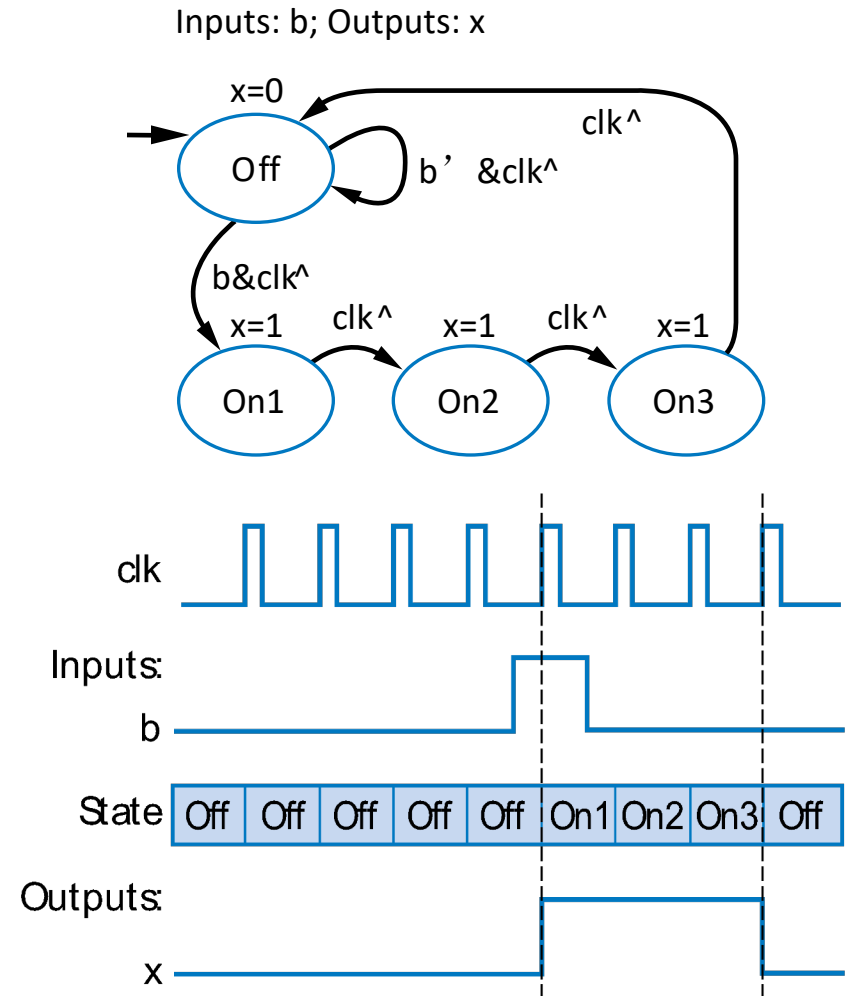
State Diagram Simplification: Clock Implicit

- ***Synchronous*** FSMs – FSM behaviors synchronized to active edge of clock
  - Asynchronous FSMs -- less common
- Make implicit – all state transitions are triggered by rising edge of clock
- Make implicit – Unlabeled path means transition is triggered only by clock, inputs don't matter

Inputs: b; Outputs: x
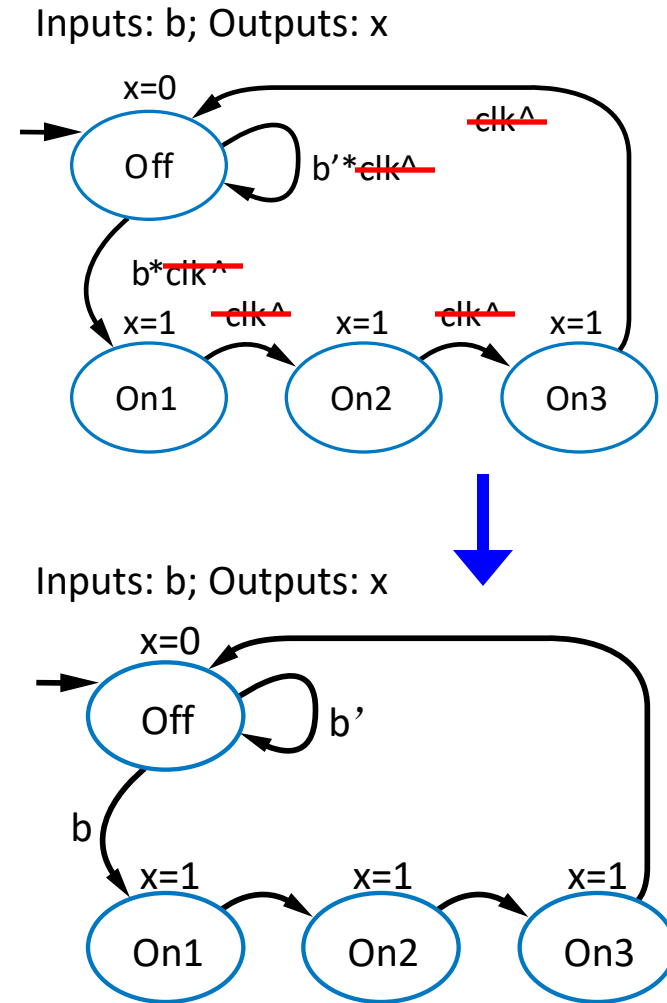
Inputs: b; Outputs: x

FSM Definition

- FSM consists of
  - Set of states
    - Ex: {Off, On1, On2, On3}
  - Set of inputs, set of outputs
    - Ex: Inputs: {b}, Outputs: {x}
  - Initial state
    - Ex: "Off"
  - Set of transitions
    - Describes next states
    - Ex: Has 5 transitions
  - Set of actions (outputs)
    - Sets outputs while in states
    - Ex: x=0, x=1, x=1, and x=1

Inputs: b; Outputs: x
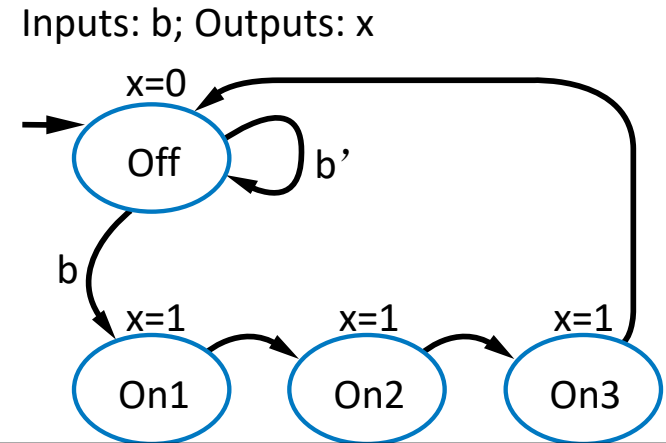
FSM can be represented graphically, known as *state diagram*

	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
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

FSM can also be represented in tabular form, known as *state table*

State Diagram and State Table

State Diagram

Input

Present State
Outputs

Or
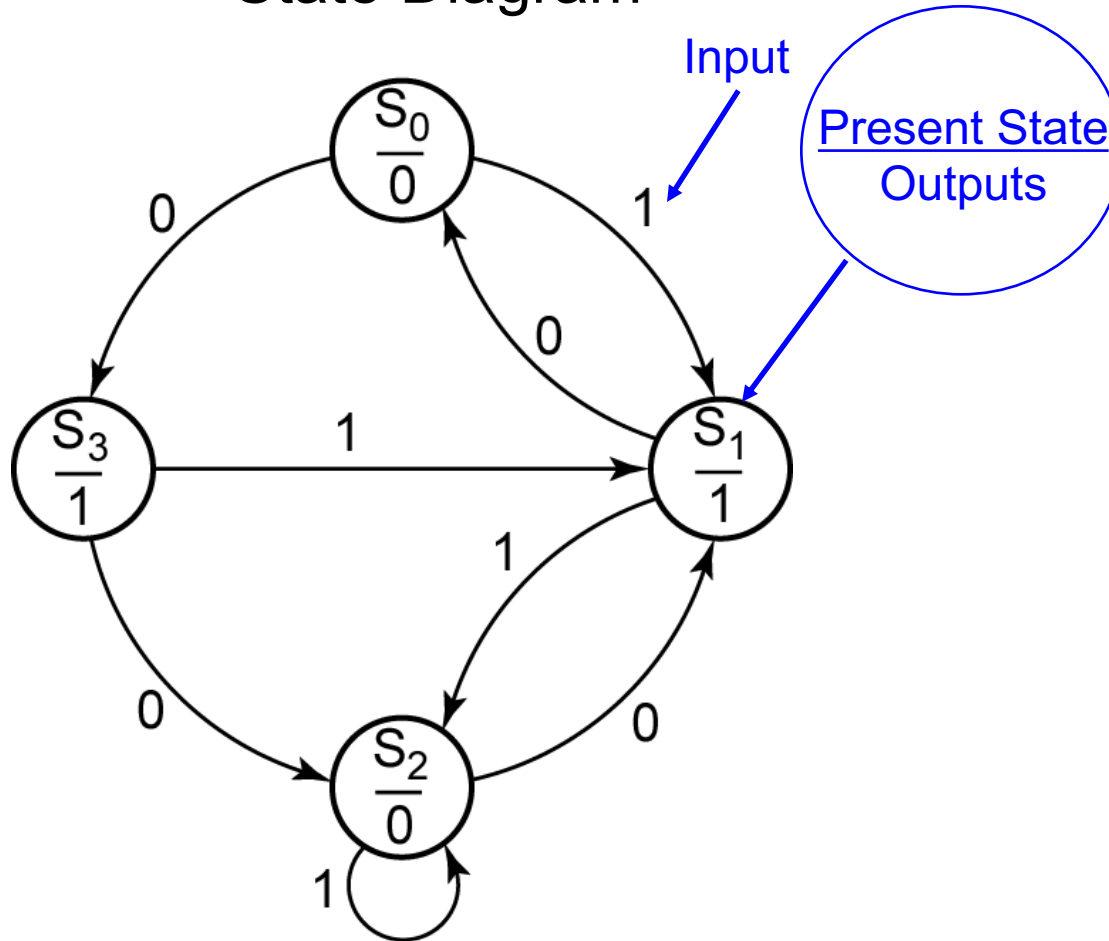
Outputs

Present State

State Table

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

Example: Secure Car Key

- All new car keys contain a tiny computer chip
  - When car starts, car's computer (under engine hood) requests identifier from key
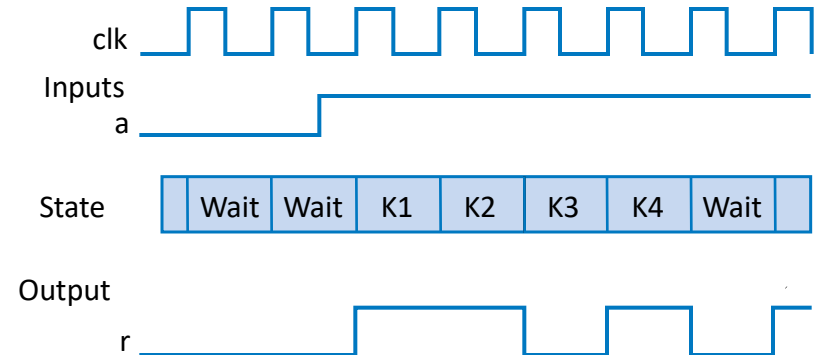 - Key transmits identifier
    - If not, computer shuts off car
- FSM
  - Wait until computer requests ID (a=1)
  - Transmit ID (in this case, 1101)

Inputs: a; Outputs: r
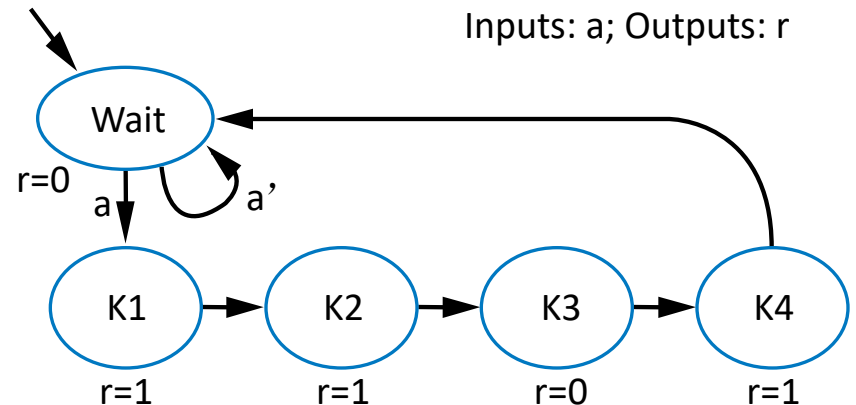
FSM Example: Secure Car Key (cont.)

- Timing diagrams show states and output values for different input waveforms

Inputs: a; Outputs: r
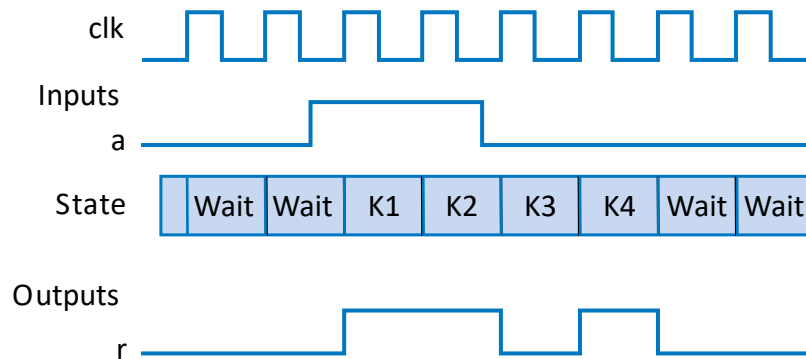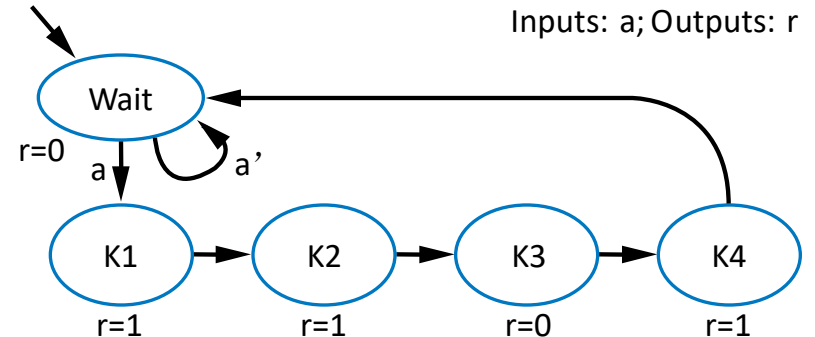
Example: Digital Lock

- Unlock door (u=1) only when buttons pressed in sequence:
  - start, then red, blue, green, red
- Input buttons: *s, r, g, b*
- Input *a* indicates that some color button pressed, *lasts for only one clock cycle with each button press*
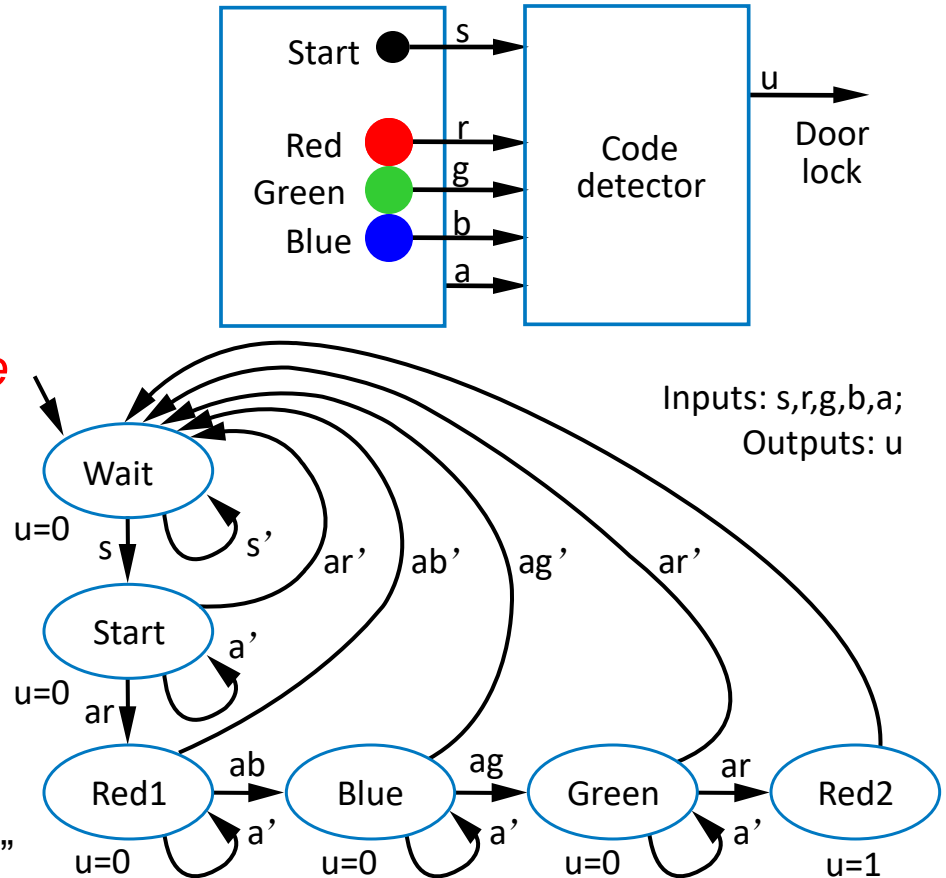- FSM
  - Wait for start (s=1) in "Wait"
  - Once started, go to "Start", then
    - If see red, go to "Red1"
    - Then, if see blue, go to "Blue"
    - Then, if see green, go to "Green"
    - Then, if see red, go to "Red2", and u=1
    - Wrong button at any step, return to "Wait", without opening door

Start ● — s → Code detector → u Door lock

Red ● — r
Green ● — g
Blue ● — b
— a

Inputs: s,r,g,b,a;
Outputs: u

Wait
u=0 — s → s'
ar' ab' ag' ar'

Start
a'
u=0 — ar →

Red1 — ab → Blue — ag → Green — ar → Red2
a' a' a'
u=0 u=0 u=0 u=1

Q: Can you trick this FSM to open the door, without knowing the code?

A: Yes, hold all buttons simultaneously

11

Improve FSM for Code Detector

Inputs: s,r,g,b,a;
Outputs: u
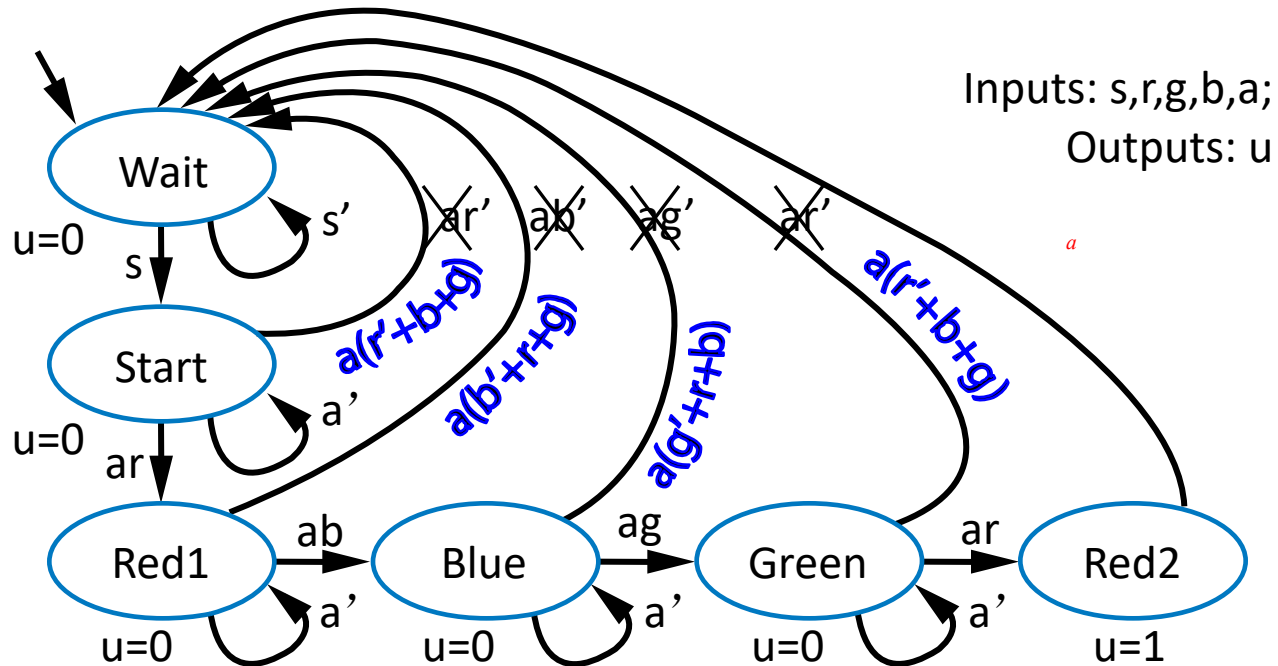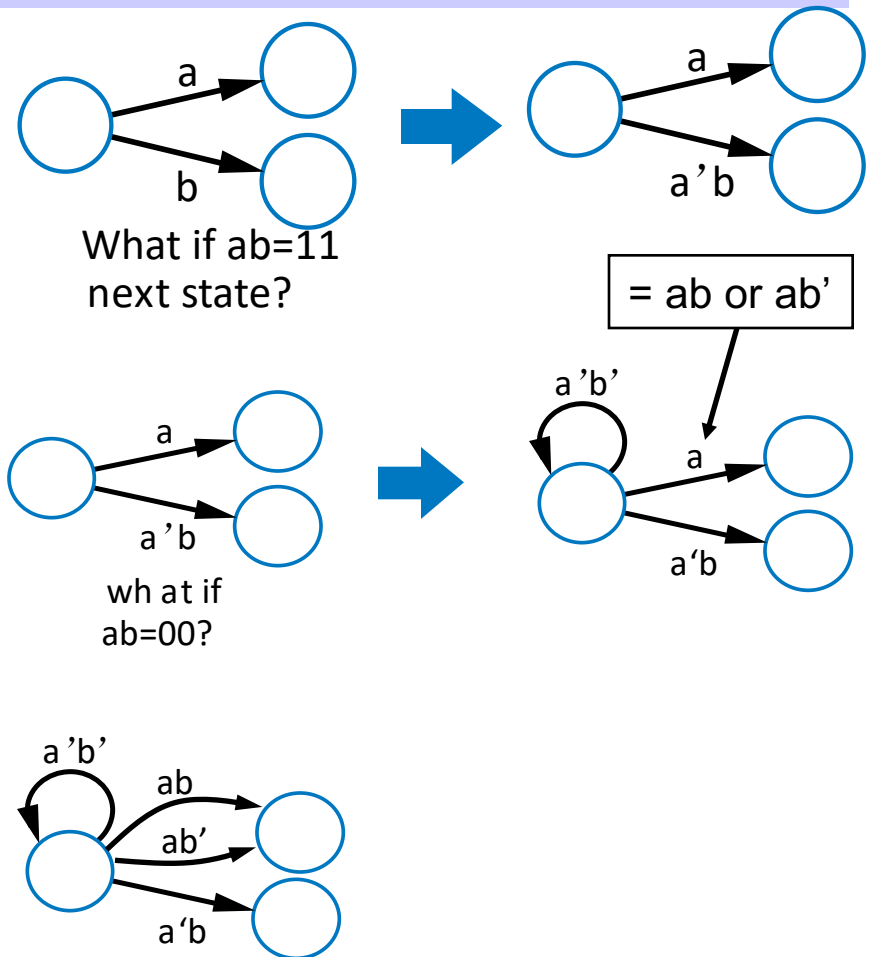
- **New transition conditions** detect if wrong button pressed, returns to "Wait"

Common State Transition Property

- *Only* one condition should be true, among all transitions leaving a state

- *One* condition must be true
  - For any input combination

- All conditions must be considered when leaving a state

What if ab=11 next state?

= ab or ab'

wh at if ab=00?

Pitfall is Common

- Recall code detector FSM
  - Do the transitions obey the two required transition properties?

    NO!

  - How would it go wrong?

    E.g. arbg = 1111
    How to solve?

Answer: **ar** should be **arb'g'**
(likewise for ab, ag, ar)

14
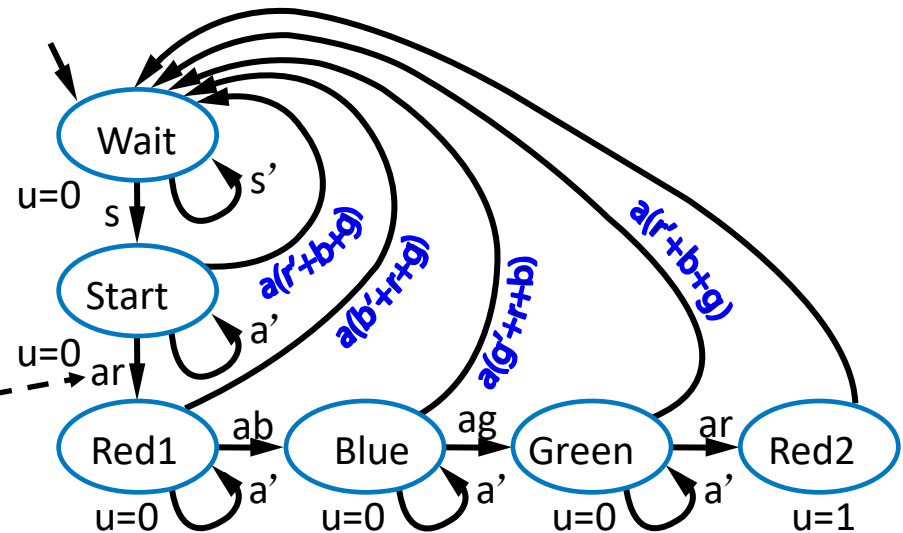
Implementation of FSM
Example: Synchronous Binary Counter

- FSM that counts binary numbers – counter
- An n-bit binary counter can count in binary from 0 up to 2^n-1 and repeat
- An n-bit binary counter consists of n flip-flops
- All the flip-flops are synchronized to the same clock – synchronous counter
- May be implemented by different type of flip-flops
- Example: a 3-bit binary counter can count through this sequence

```
000 → 001 → 010 → 011
                    ↓
111 ← 110 ← 101 ← 100
```

reset
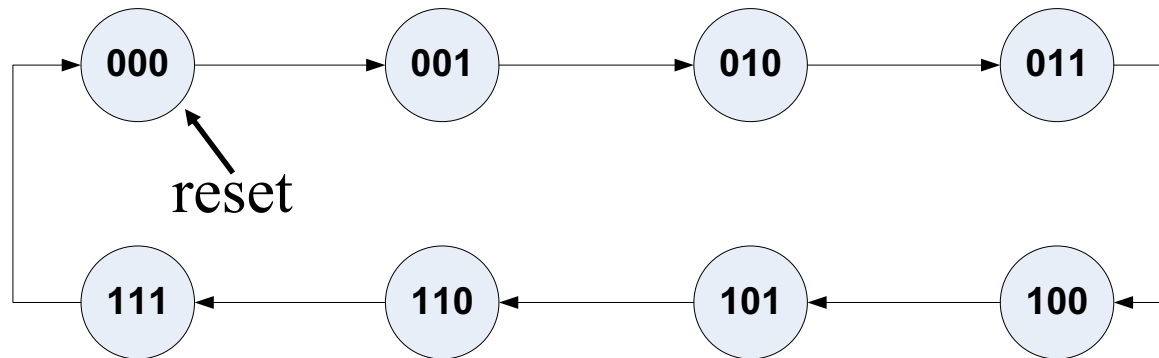
Synchronous Binary Counter Design

- An FSM (without external inputs)
  - State Table

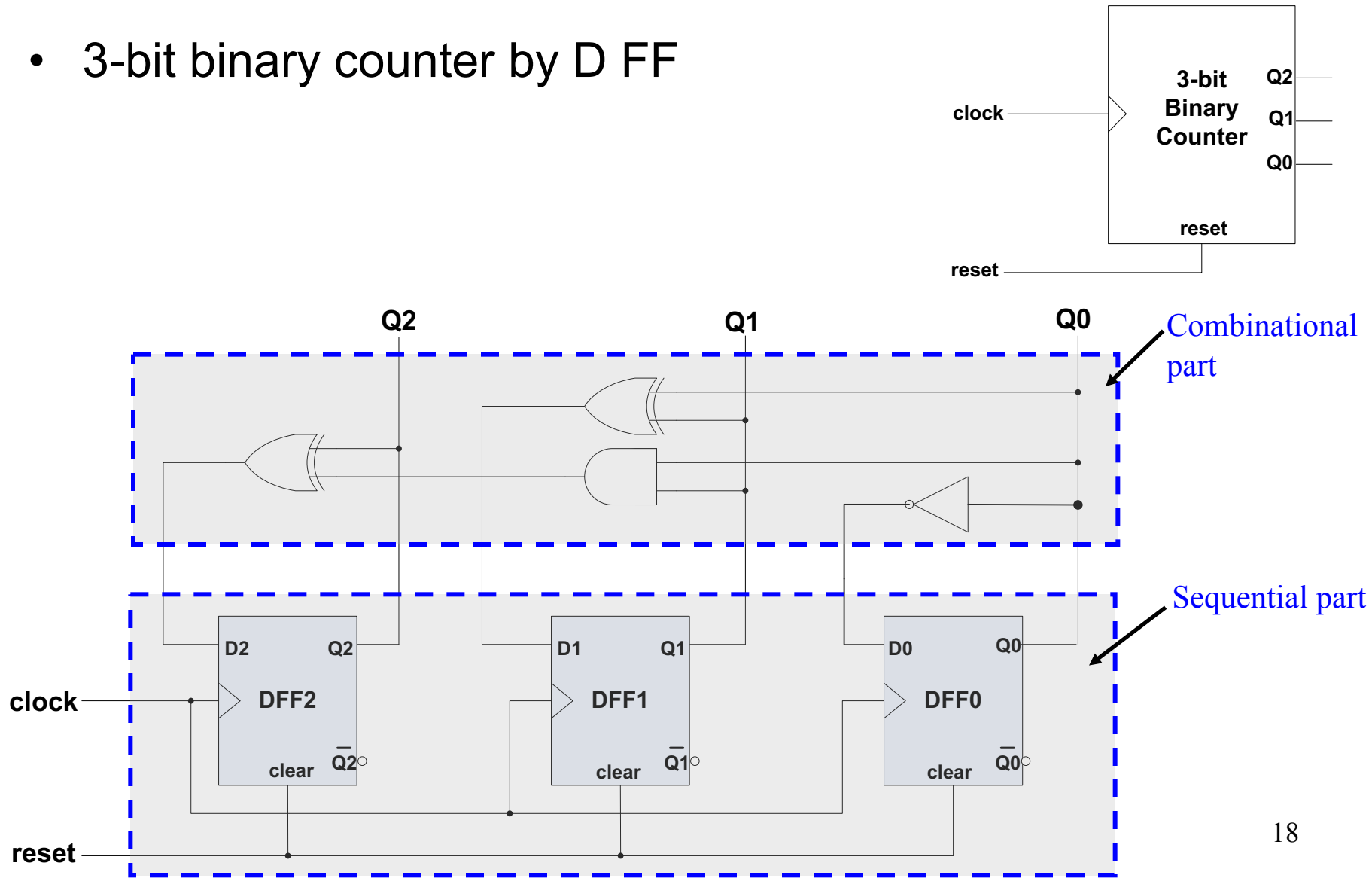|	Present State			Next State	
$Q2$	$Q1$	$Q0$	$Q2^+$	$Q1^+$	$Q0^+$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Counter Implemented with D Flip-Flop

- Use D flip flops to hold values： **Q^+ = D upon active edge**
- Next state equations

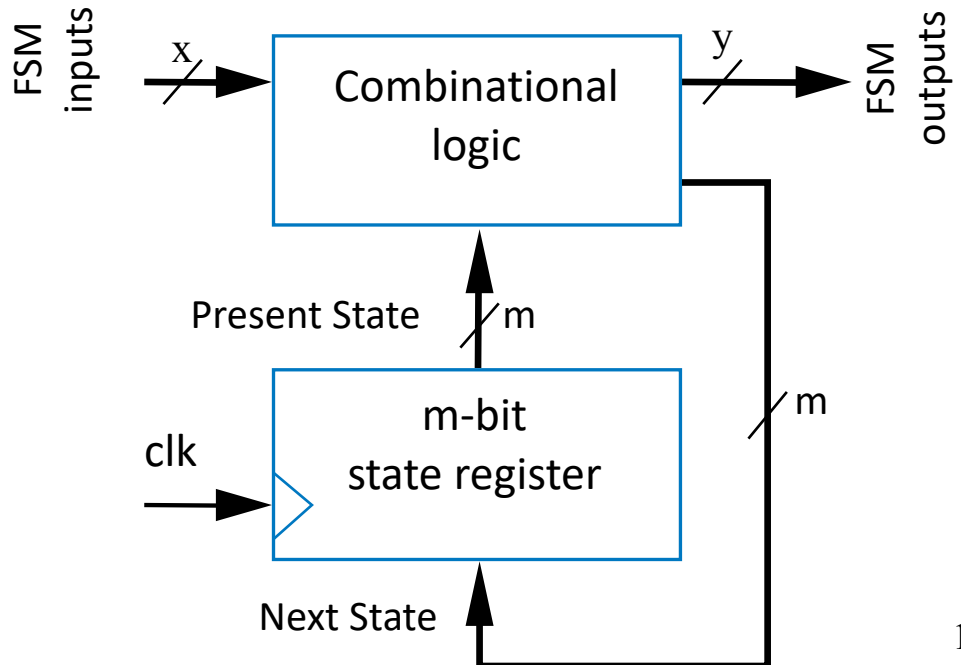| Present State			Next State			D flip flop input		
Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$	D2	D1	D0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

Synchronous Binary Counter with D Flip-Flop

- 3-bit binary counter by D FF

18

Standard FSM Architecture

- How to design sequential circuit?
  - Design as FSM
  - Use standard architecture
    - **State register** -- to store the present state
    - **Combinational logic** -- to compute outputs, and next state
 - Known as *controller*

19

FSM (Controller) Design

- Five step FSM design process

	Step	Description
Step 1	*Capture the FSM*	Create an FSM that describes the desired behavior of the controller.
Step 2	*Create the architecture*	Create the standard architecture by using a state register of appropriate width, and combinational logic with inputs being the state register bits and the FSM inputs and outputs being the next state bits and the FSM outputs.
Step 3	*Encode the states*	Assign a unique binary number to each state. Each binary number representing a state is known as an **encoding.** Any encoding will do as long as each state has a unique encoding.
Step 4	*Create the state table*	Create a truth table for the combinational logic such that the logic will generate the correct FSM outputs and next state signals. Ordering the inputs with state bits first makes this truth table describe the state behavior, so the table is a state table.
Step 5	*Implement the combinational logic*	Implement the combinational logic using any method.

FSM Design Example: Push Button

- **Step 1:** Capture the FSM
  - Already done

- **Step 2:** Create architecture
  - 2-bit state register (for 4 states)
  - Input b, output x
  - Present state signals (s1, s0)
  - Next state signals (n1, n0)

- **Step 3:** Encode the states
  - Any encoding with unique representation for each state will work

Inputs: b; Outputs: x
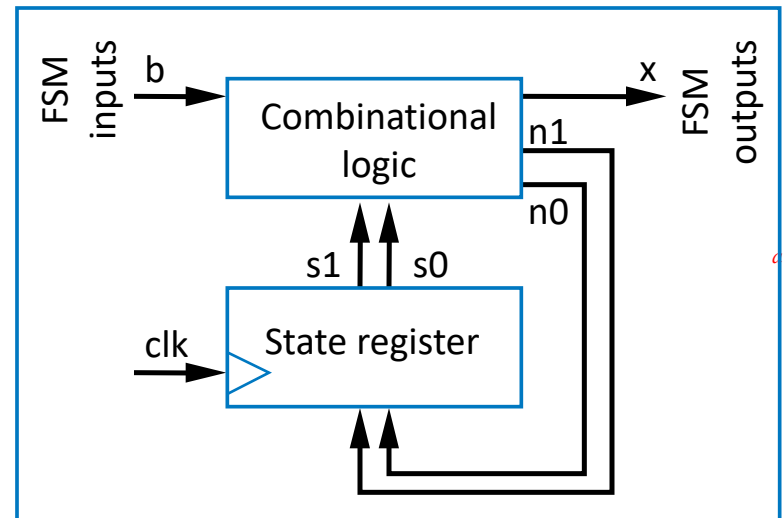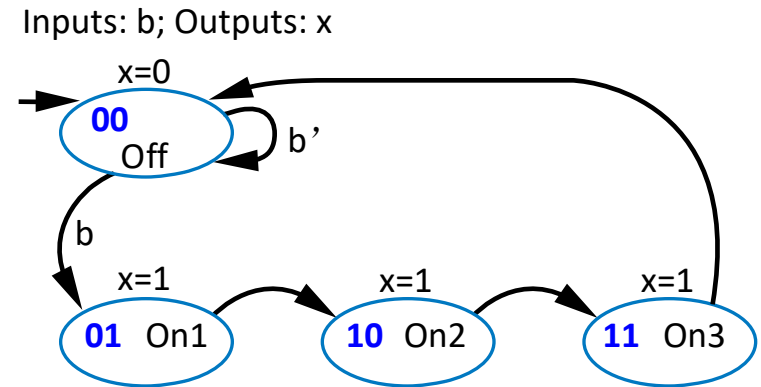
FSM Design Example: Push Button (cont.)

- Step 4: Create state table

Inputs: b; Outputs: x

Inputs			Outputs		
Present state		b	x	**Next state**	
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

Combinational Logic Inputs   **Combinational Logic Outputs**

FSM Design Example: Push Button (cont.)

- Step 5: Implement combinational logic

$x = s1 + s0$

$n1 = s1's0 + s1s0'$

S_0

$n0 = s0'b + s1s0'$

$s1s0'b$

	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
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

x s0 b

s1	00	01	11	10
0	**0**	**0**	**1**	**1**
1	**1**	**1**	**1**	**1**

n1 s0 b

s1	00	01	11	10
0	**0**	**0**	**1**	**1**
1	**1**	**1**	**0**	**0**

n0 s0 b

s1	00	01	11	10
0	**0**	**1**	**0**	**0**
1	**1**	**1**	**0**	**0**

FSM Design Example: Push Button (cont.)
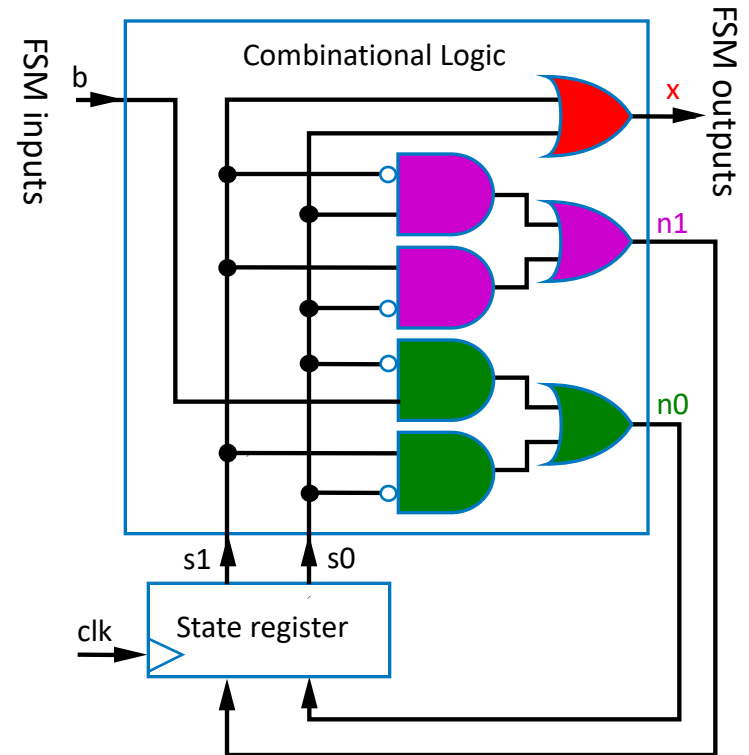
- Step 5: Implement combinational logic (cont)
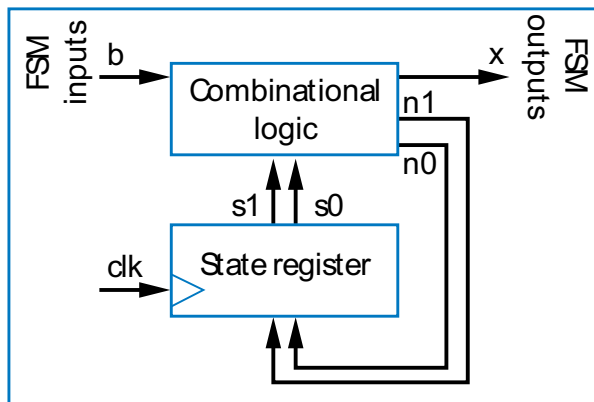
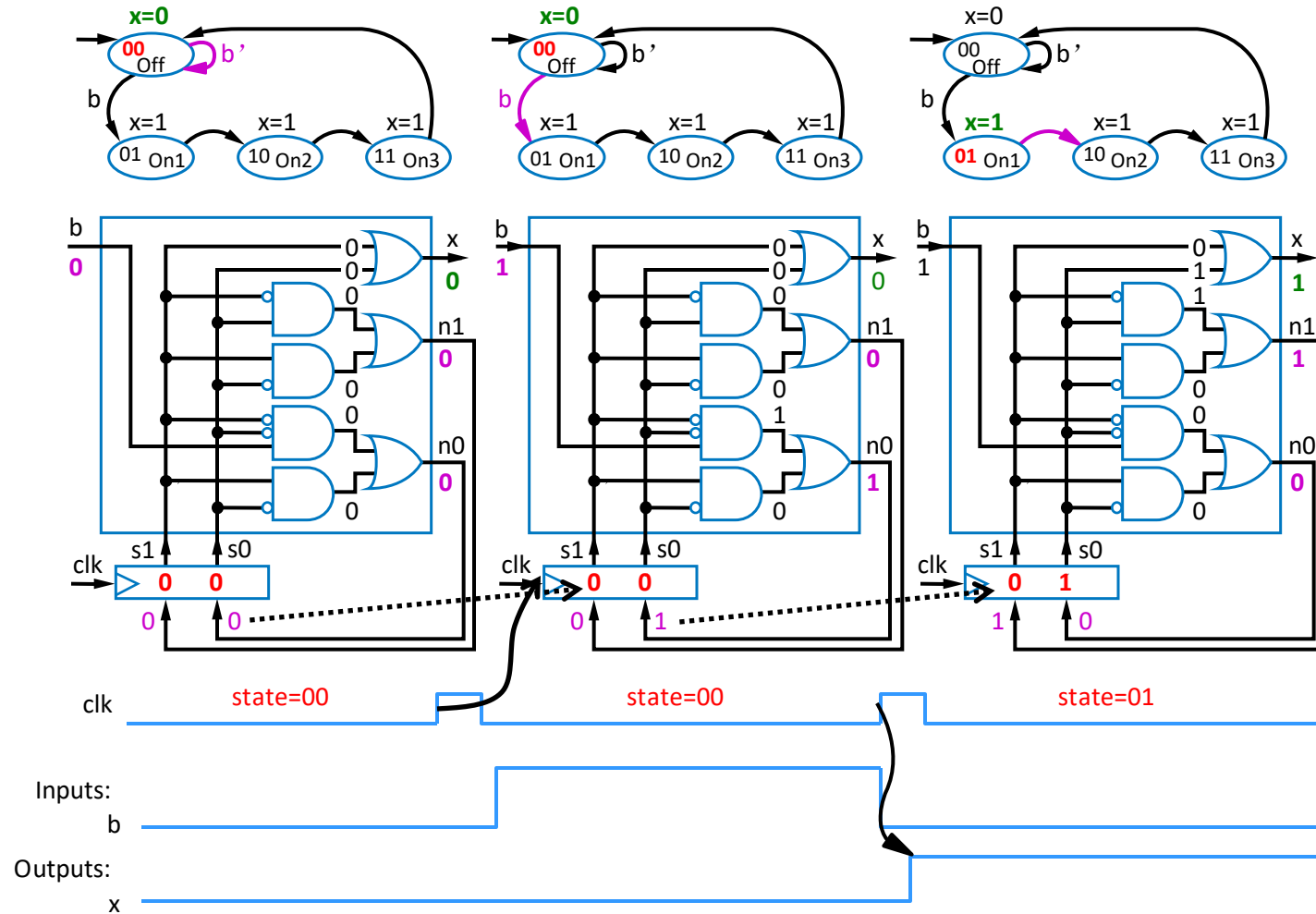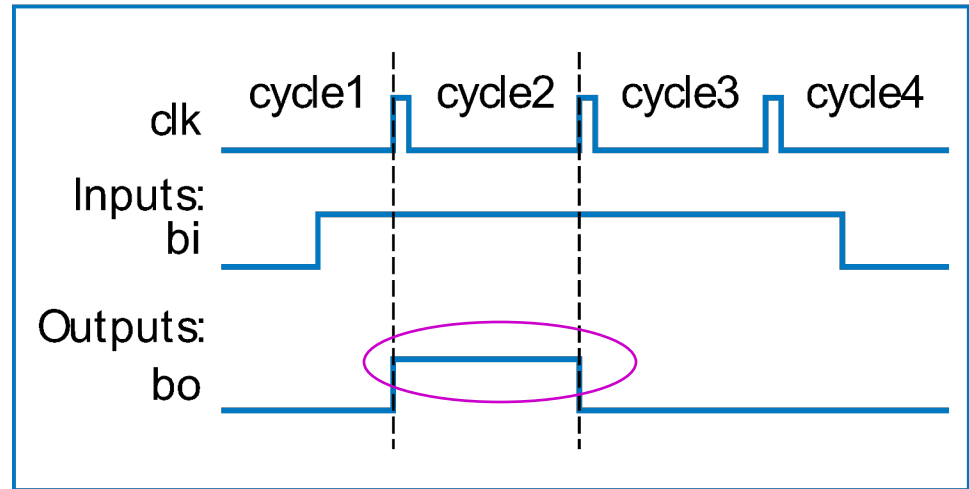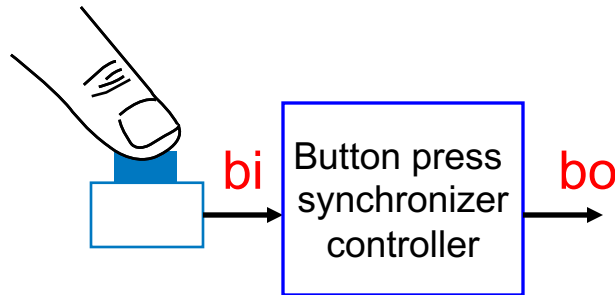$x = s1 + s0$

$n1 = s1's0 + s1s0'$

$n0 = s0'b + s1s0'$

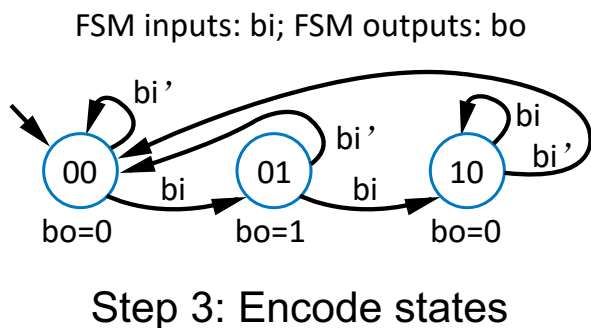# Understanding the Controller's Behavior

25

FSM Design Example:
Button Press Synchronizer

- Want simple sequential circuit that converts button press to single clock cycle duration, regardless of length of time that button actually pressed
- Producing a signal like 'a' in the secure door example

FSM Design Example:
Button Press Synchronizer (cont.)

FSM inputs: bi; FSM outputs: bo

Step 1: FSM

Step 2: Create architecture

$n1 = s1's0bi + s1s0bi$
$n0 = s1's0'bi$
$bo = s1's0bi' + s1's0bi = s1s0$

FSM inputs: bi; FSM outputs: bo

Step 3: Encode states

Combinational logic					
Inputs			Outputs		
s1	s0	bi	n1	n0	bo
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

A, B, C, unused

Step 4: State table

may be 'x'

Step 5: Create combinational circuit

27
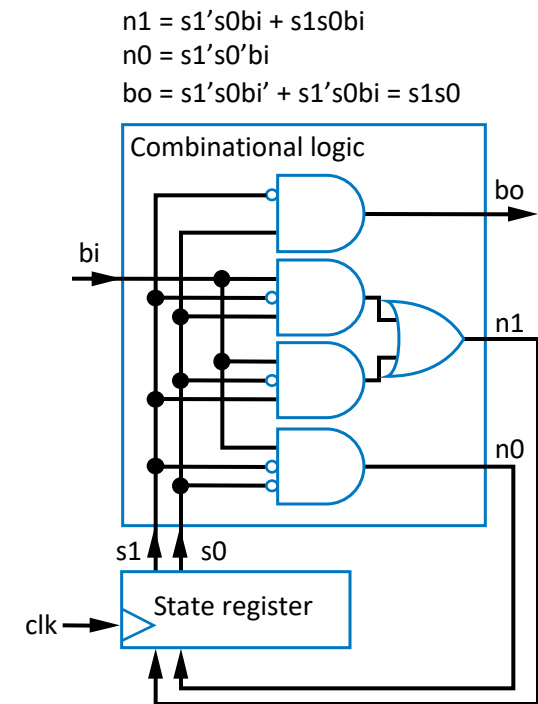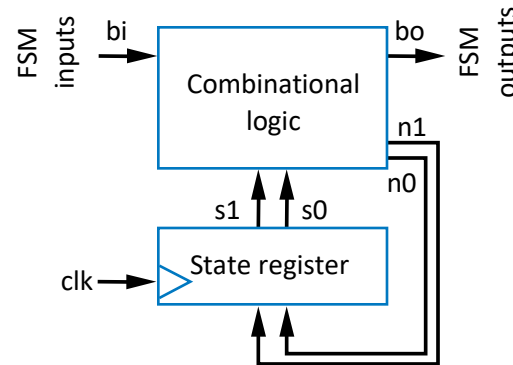
FSM Example: Sequence Generator

- Want generate sequence 0001, 0011, 1100, 1000, (repeat)
  - Each value for one clock cycle

Inputs: none; Outputs: w,x,y,z

wxyz=0001   wxyz=1000

A   D

wxyz=0011   wxyz=1100
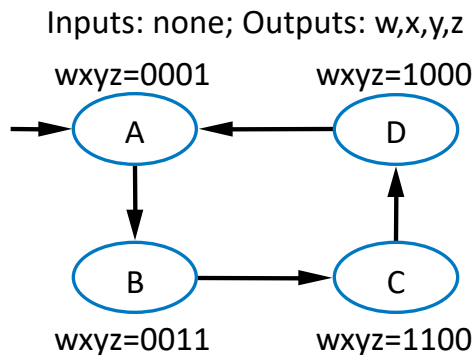
Step 1: Create FSM

Step 2: Create architecture

Inputs: none; Outputs: w,x,y,z

wxyz=0001   wxyz=1000

A   00   D   11

    01   10

B   C

wxyz=0011   wxyz=1100
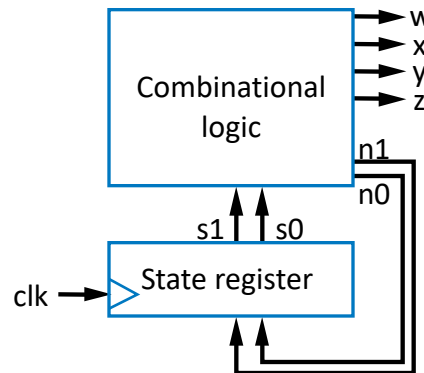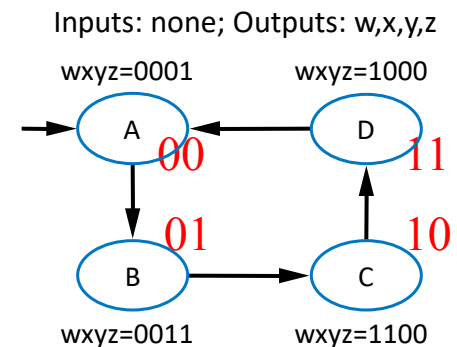
Step 3: Encode states

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

Step 4: Create state table

w = s1
x = s1s0'
y = s1's0
z = s1'
n1 = s1 xor s0
n0 = s0'

Step 5: Create combinational circuit   28

FSM Example: Secure Car Key

Step 1

Inputs: a; Outputs: r

Step 2

Step 3

Inputs: a; Outputs: r

We omit Step 5
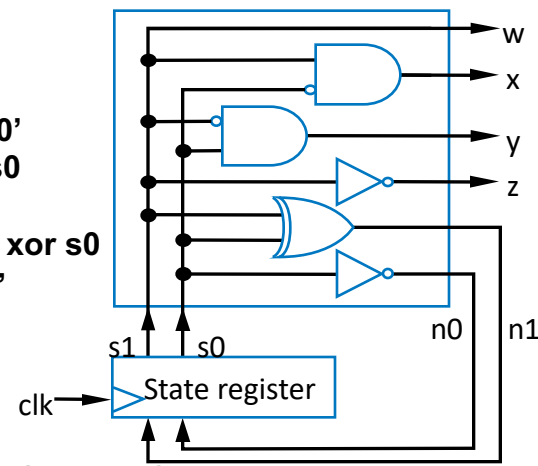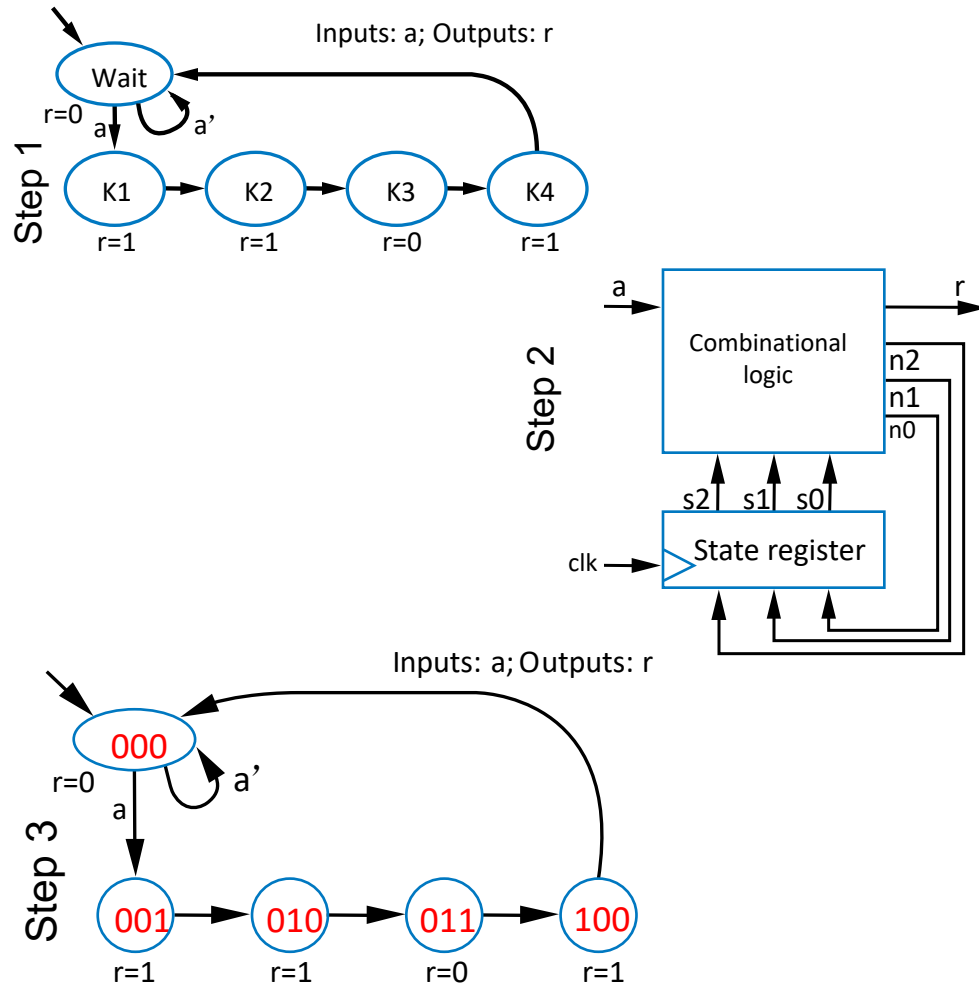
	Inputs				Outputs			
	s2	s1	s0	a	r	n2	n1	n0
Wait	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	1
K1	0	0	1	0	1	0	1	0
	0	0	1	1	1	0	1	0
K2	0	1	0	0	1	0	1	1
	0	1	0	1	1	0	1	1
K3	0	1	1	0	0	1	0	0
	0	1	1	1	0	1	0	0
K4	1	0	0	0	1	0	0	0
	1	0	0	1	1	0	0	0
Unused	1	0	1	0	0	0	0	0
	1	0	1	1	0	0	0	0
	1	1	0	0	0	0	0	0
	1	1	0	1	0	0	0	0
	1	1	1	0	0	0	0	0
	1	1	1	1	0	0	0	0

Step 4

may be 'x'

29

Verilog Modeling of FSM
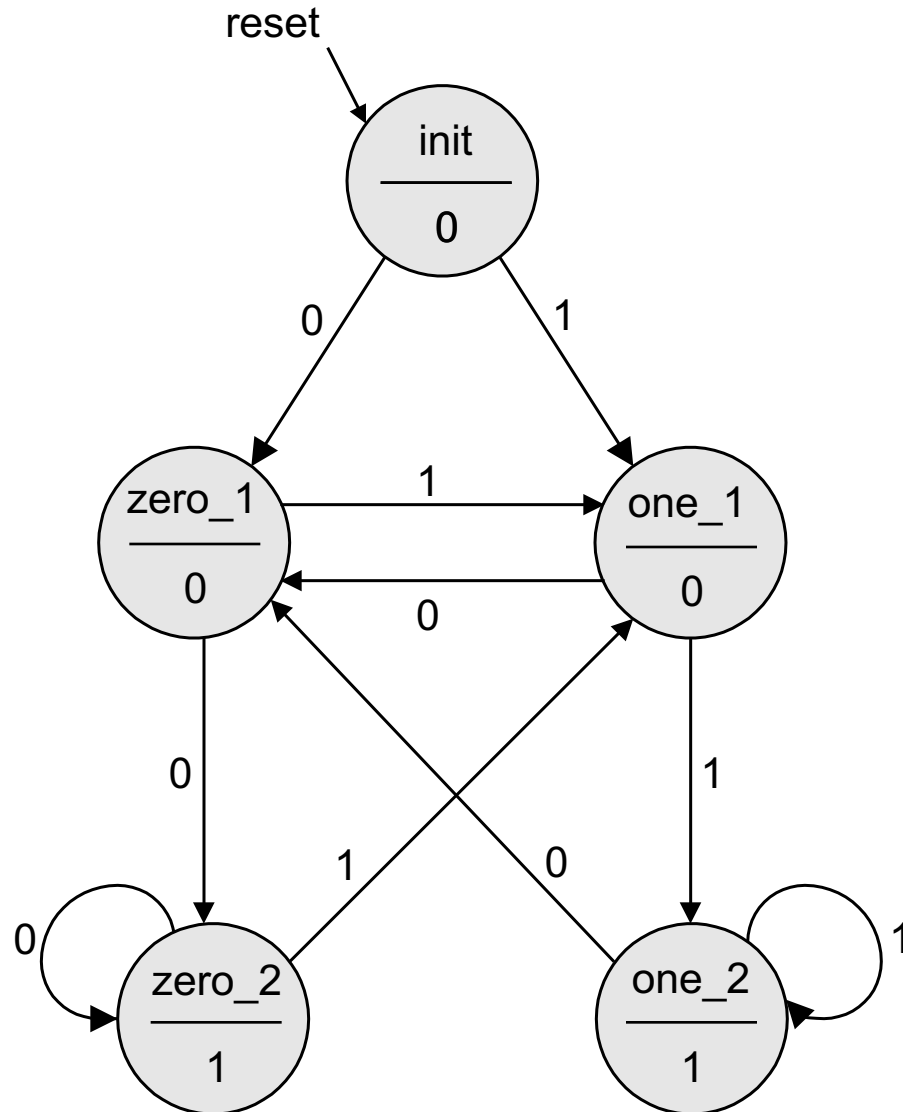
```verilog
Module example_FSM (clock, reset, in_bit, out_bit);
   input       clock, reset, in_bit;
   output      out_bit;

   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)          ← State register
     if (reset == 1) curr_state <= init;
     else            curr_state <= next_state;

   always @ (curr_state or in_bit)          ← Combinational logic
     case (curr_state)                          for next state
       init: if (in_bit == 0) next_state <= zero_1; else
             if (in_bit == 1) next_state <= one_1;  else
                              next_state <= init;
```

31

```verilog
        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

32

Revisit: Binary Counter as an FSM

- FSM that counts binary numbers – counter
- An n-bit binary counter can count in binary from 0 up to 2^n-1 and repeat
- An n-bit binary counter consists of n flip-flops
- All the flip-flops are synchronized to the same clock – synchronous counter
- May be implemented by different type of flip-flops
- Example: a 3-bit binary counter can count through this sequence
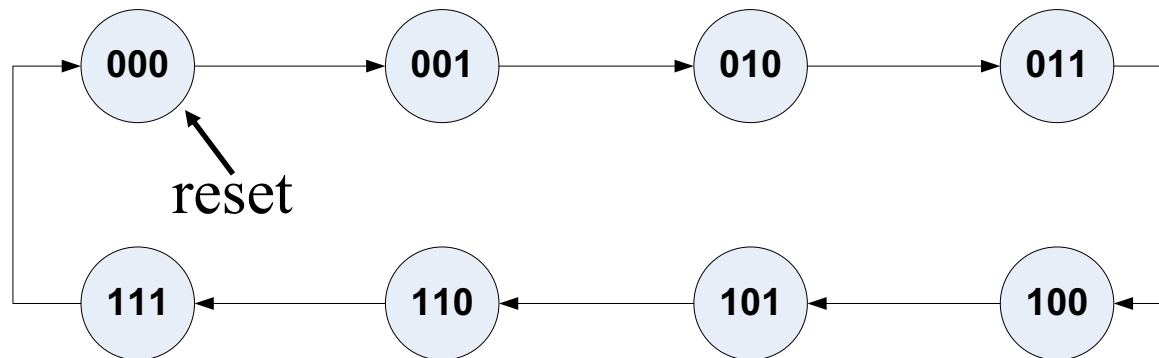
Implement FSM with T Flip-Flop

- Characteristic table and equation of T-FF

T	Q$^+$	
0	Q	No Change
1	Q'	Complement

$$Q^+ = T \oplus Q$$

- Excitation table of T-FF

Q	Q$^+$		T
0	0	No Change	0
0	1	Toggle	1
1	0	Toggle	1
1	1	No Change	0

Q	Q$^+$		T
X	Q	No Change	0
X	Q'	Toggle	1

Implement FSM with T Flip-Flop

- State Table with T input

Intermediate columns

Present State			Next State			T Input		
Q2	Q1	Q0	$Q2^+$	$Q1^+$	$Q0^+$	T2	T1	T0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Implement FSM with T Flip-Flop

- T input equations in terms of Qs can be found from the state table

T2

Q1Q0

Q2	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$$T2 = Q1Q0$$

T1

Q1Q0

Q2	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$$T1 = Q0$$

T0

Q1Q0
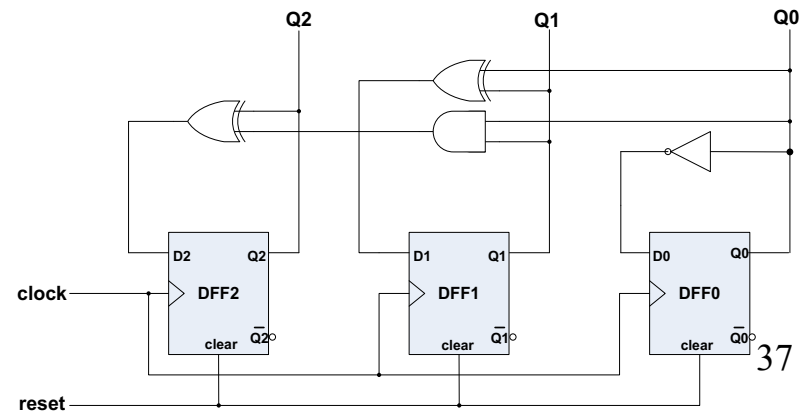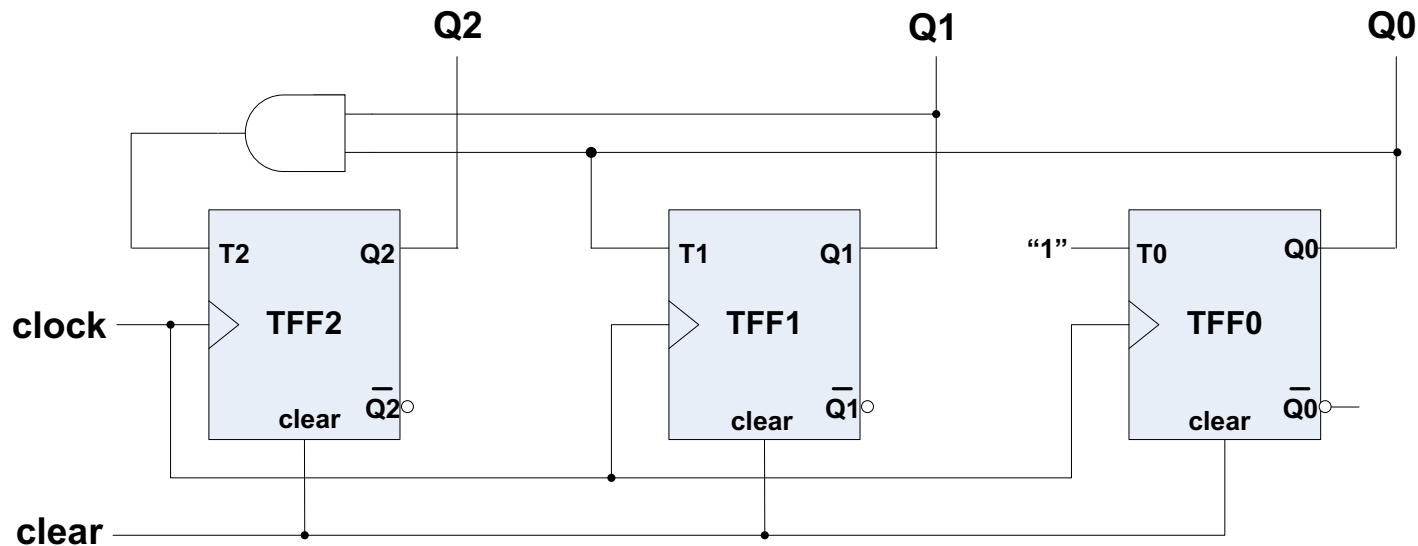
Q2	00	01	11	10
0	1	1	1	1
1	1	1	1	1

$$T0 = 1$$

Implement FSM with T Flip-Flop

- 3-bit binary counter by T FF

Implement FSM with JK Flip-Flop

- Characteristic table and equation of JK-FF

J	K	Q^+	Action
0	0	Q	Hold
0	1	0	Reset
1	0	1	Set
1	1	Q'	Toggle

$$Q^+ = JQ' + K'Q$$

- Excitation table of JK-FF

Q	Q^+	Action	J	K	J	K
0	0	Reset/Hold	0/0	1/0	0	X
0	1	Set/Toggle	1/1	0/1	1	X
1	0	Reset/Toggle	0/1	1/1	X	1
1	1	Set/Hold	1/0	0/0	X	0

Implement FSM with JK Flip-Flop

- State Table with JK input

Intermediate columns

Present State			Next State			JK Inputs					
Q2	Q1	Q0	Q2+	Q1+	Q0+	J2	K2	J1	K1	J0	K0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Implement FSM with JK Flip-Flop

- J-K Input equations

J2 = Q1Q0

J1 = Q0

J0 = 1

K2 = Q1Q0 = J2

K1=Q0 = J1

K0=1= J0

Implement FSM with JK Flip-Flop

- Circuit diagram