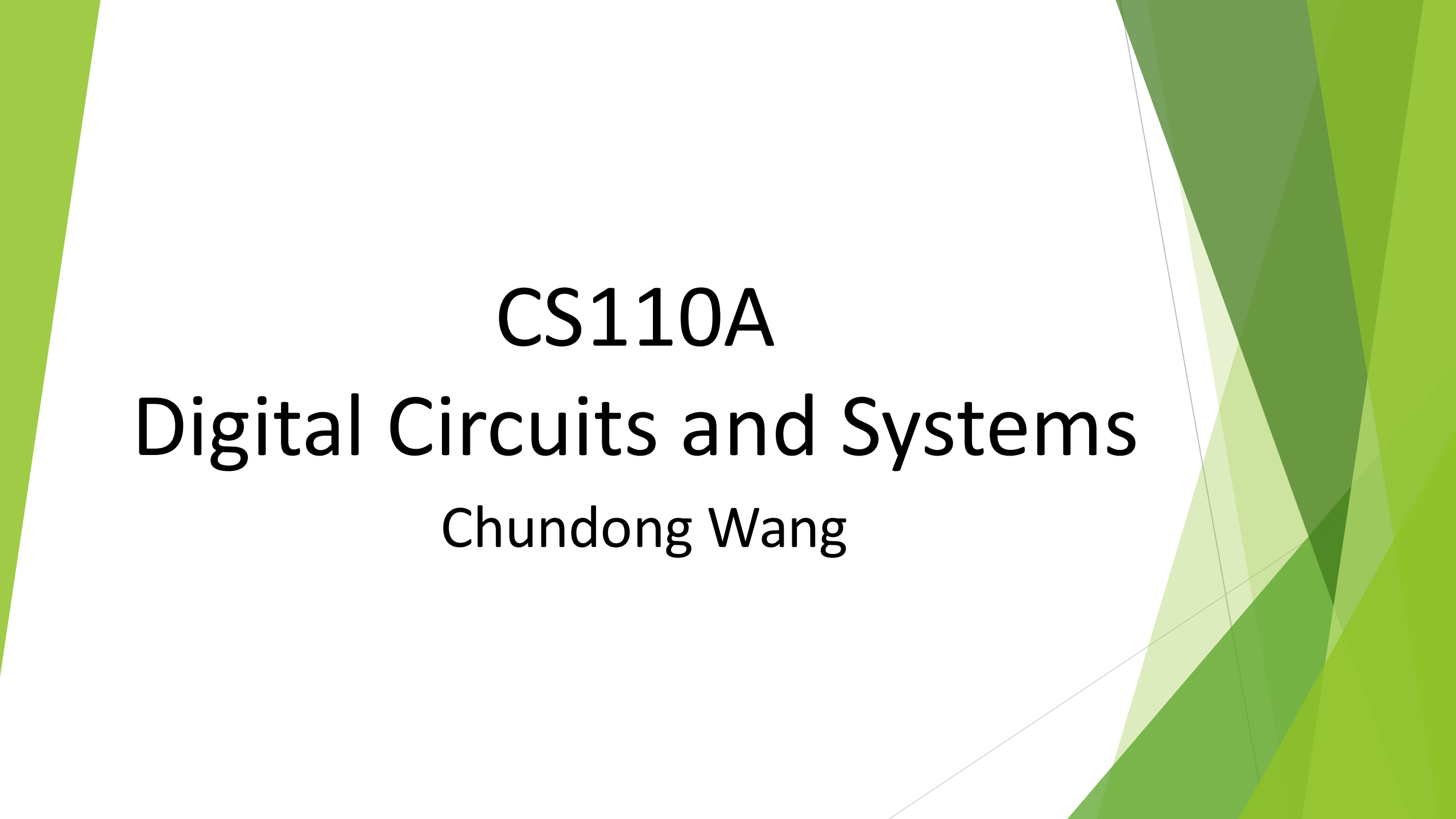


The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look.

# CS110A

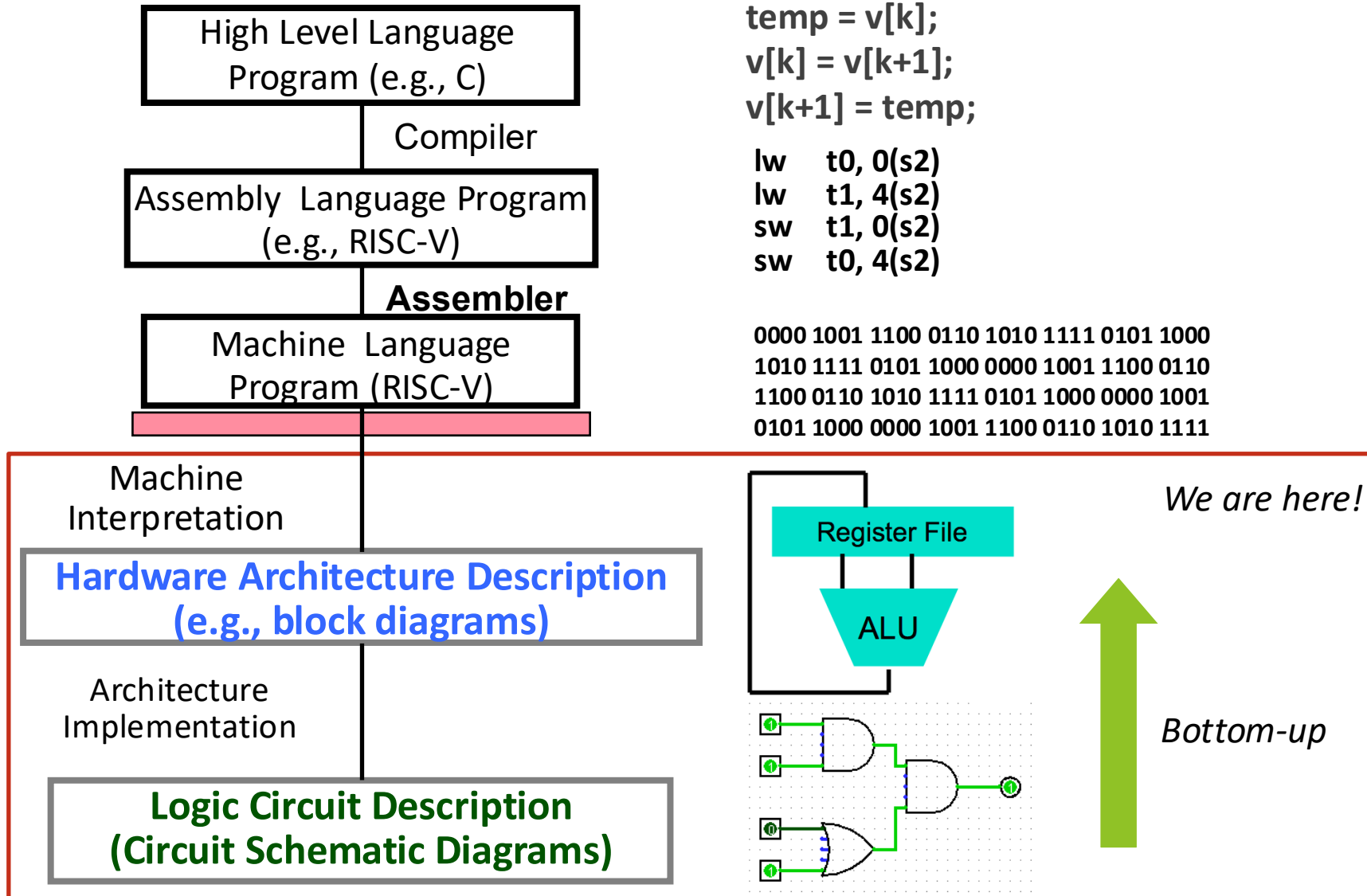
# Digital Circuits and Systems

Chundong Wang

# Outline

- ▶ Digital system
- ▶ Combinational logics
  - ▣ From transistors to basic logic gates
  - ▣ From logic gates to combinational circuits
    - ▣ Boolean algebra
    - ▣ Boolean expression
    - ▣ Truth table
- ▶ State elements
- ▶ Useful building blocks

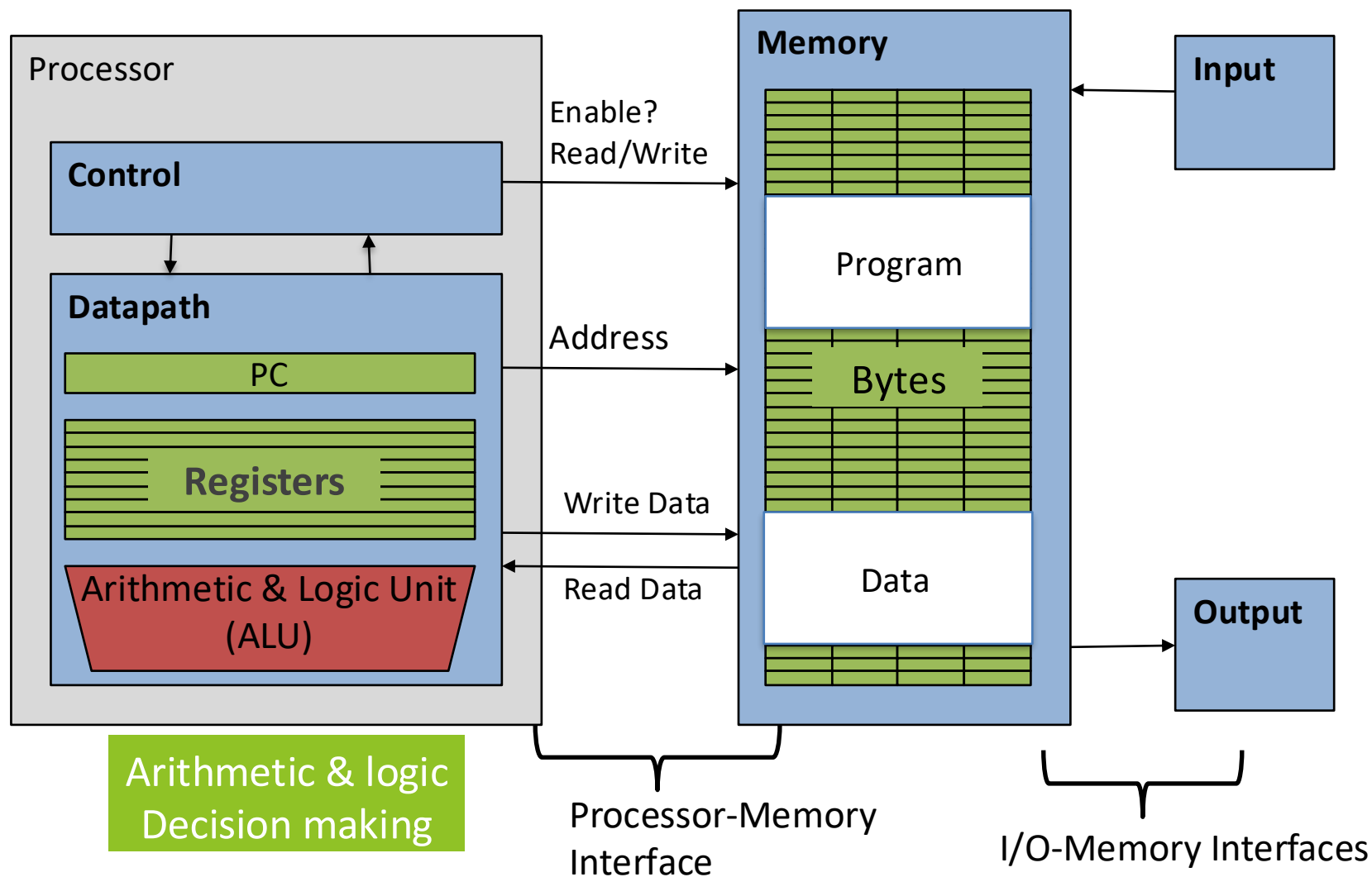
# Where are we?



# Hardware (HW) Design

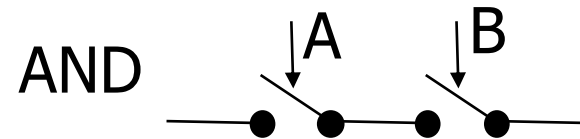
- ▶ How a modern processor is built, starting with basic elements (transistors) as building blocks
- ▶ Why study hardware design?
  - ▶ Understand capabilities and limitations of HW in general and processors in particular
  - ▶ What processors can do fast and what they can't do fast (avoid slow things if you want your code to run fast!)
  - ▶ Background for more in-depth HW courses (Digital circuit/VLSI/AI computing system, etc.)
  - ▶ There is only so much you can do with standard processors: you may need to design own custom HW for extra performance
    - ✓ Even some commercial processors today have customizable hardware!
    - ✓ E.g. Google Tensor Processing Unit (TPU)

# A High-level View of Computer

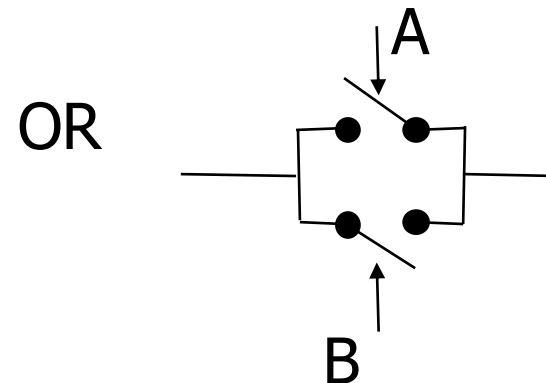


# Switches

- Compose switches into more complex ones (Boolean functions):



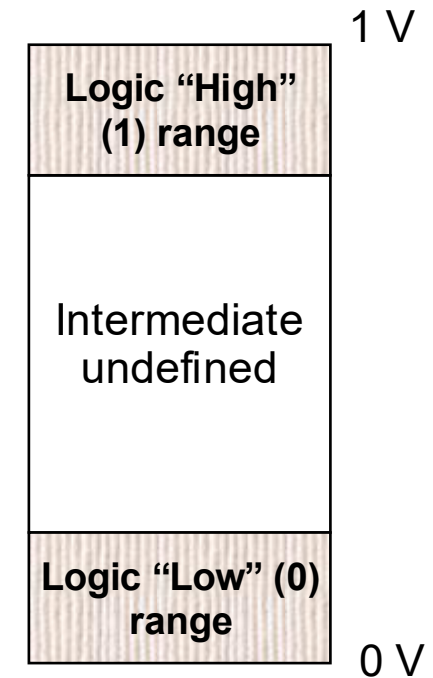
$$Z \equiv A \text{ and } B$$



$$Z \equiv A \text{ or } B$$

# Binary Representation of Signals

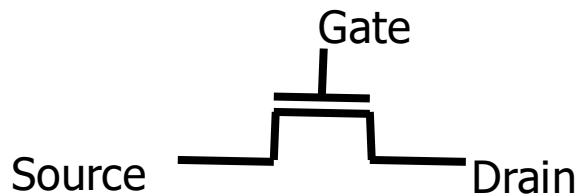
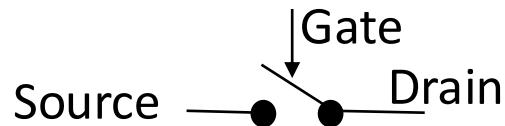
- ▶ High voltage ( $V_{dd}$ ) represents 1, or true
  - ▶ In modern microprocessors, core  $V_{dd} \sim 1.0$  Volt
- ▶ Low voltage (0 Volt or Ground) represents 0, or false
- ▶ **Digital**: discretize signal/voltage to a 0 or a 1
  - ▶ This removes noise as signals propagate – a big advantage of digital systems over **analog** systems
  - ▶ Circuits to discriminate between two possible inputs are simple to implement and have scaled well with Moore's Law.
- ▶ If one switch can control another switch with digital signal, we can build a computer!
- ▶ Our switches: CMOS transistors



# NMOS & PMOS Transistors

- ▶ Three terminals: source, gate, and drain

- ▶ Basic model



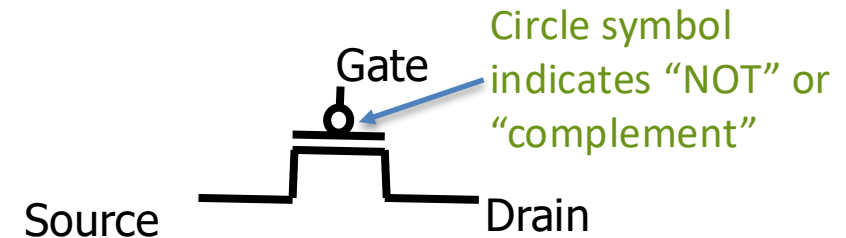
n-channel transistor

off when voltage at Gate is low

on when:

voltage (Gate) > voltage (Threshold)

(**High** resistance when gate voltage **Low**,  
**Low resistance when gate voltage High**)



Circle symbol  
indicates "NOT" or  
"complement"

p-channel transistor

on when voltage at Gate is low

off when:

voltage (Gate) > voltage (Threshold)

(**Low** resistance when gate voltage **Low**,  
**High resistance when gate voltage High**)

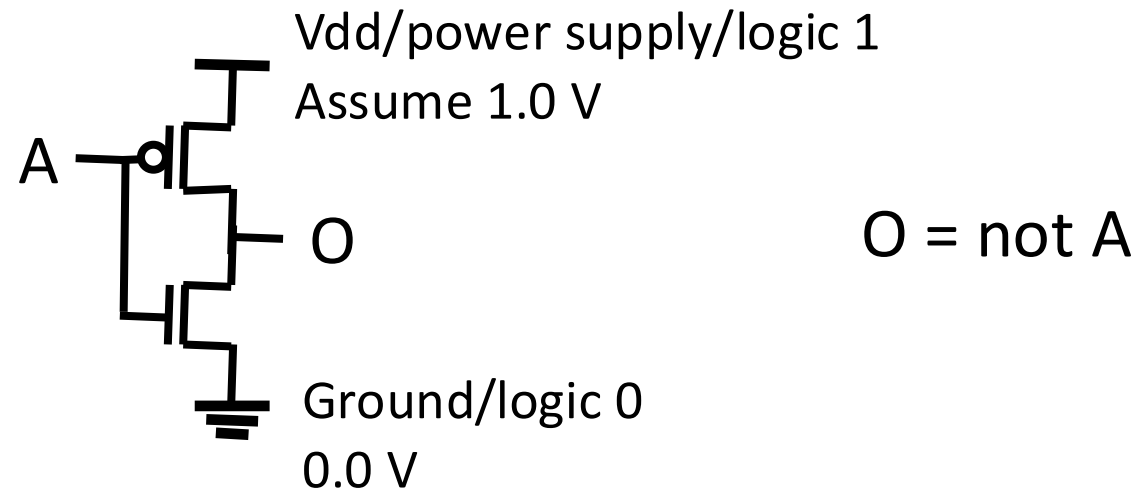
# Synchronous Digital System (SDS)

- ▶ A system that processes digital signals (0s and 1s)
- ▶ Synchronous digital systems consist of two basic types of circuits.
  - ▶ Combinational logic circuits
    - ▶ The outputs solely depend on the input
    - ▶ No way to store information
  - ▶ State Elements (next lecture)
    - ▶ Circuits that store information
    - ▶ E.g., registers and memory
- ▶ CPU cores are SDS's

- Our Goal: Implement a RISC-V processor as a synchronous digital system.
- This SDS should have the capabilities to execute RISC-V instructions.

# The Simplest CMOS Circuits

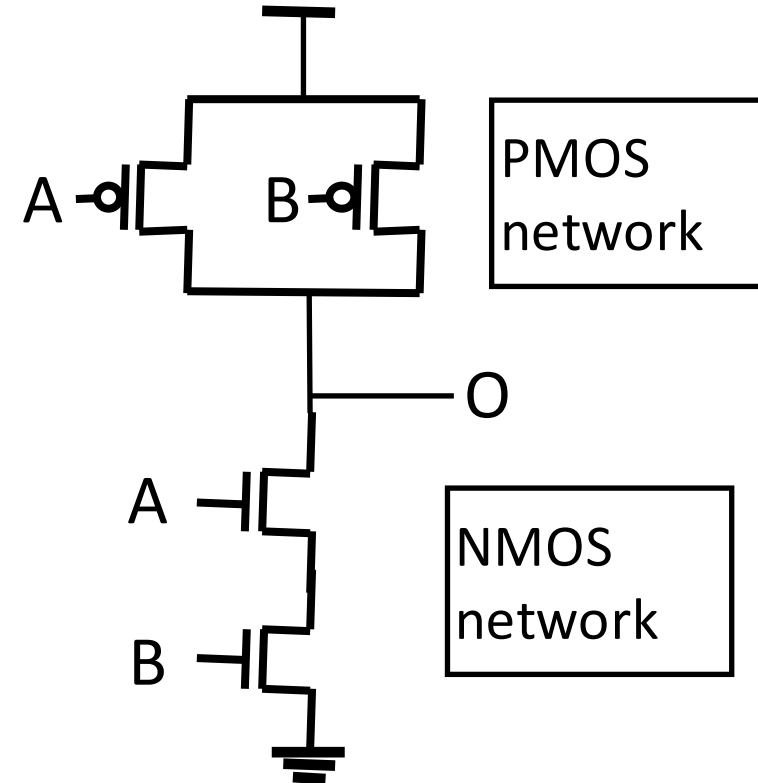
## ► Inverter/Not gate



# NAND Gate

Truth table

| A | B | O |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

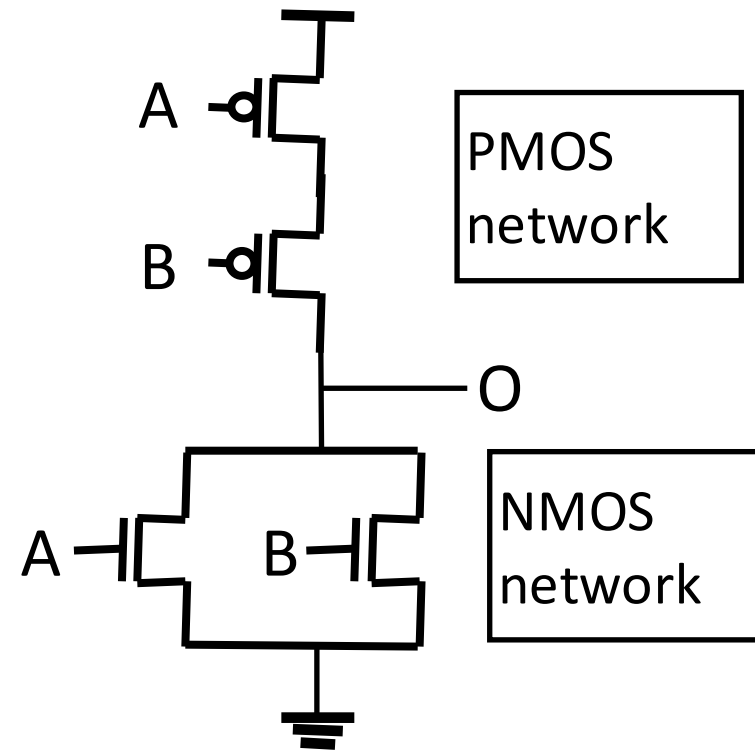


What about 3-input NAND?  
What about 2-input AND?

# What Gate?

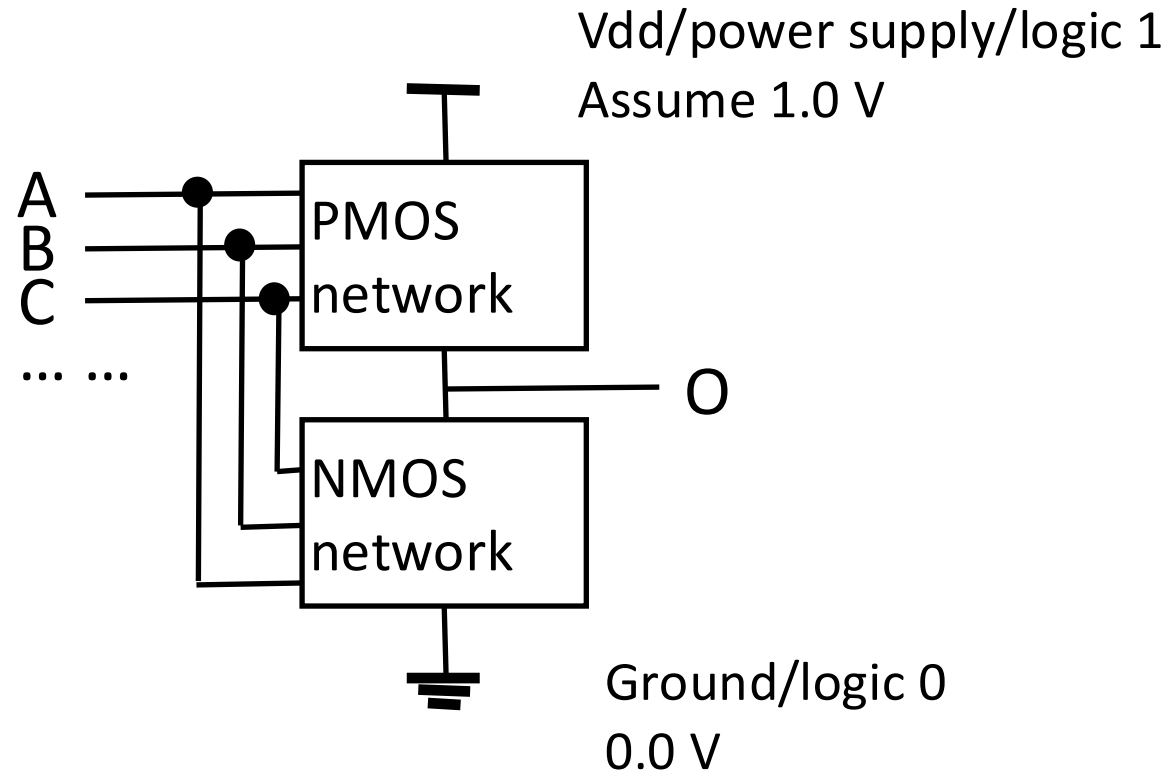
Truth table

| A | B | O |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



What about 3-input NOR?

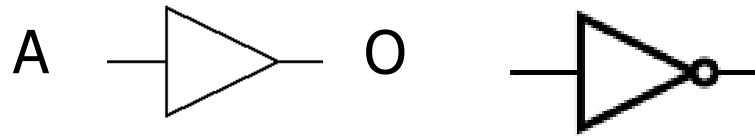
# General CMOS Logic Gates



# Basic Symbols

## ► Standard symbols for logic gates

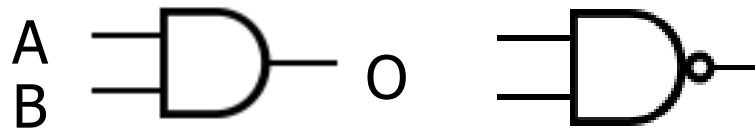
### ► Buffer, NOT



- Universal sets
  - NOT, AND, OR

Can be combined to implement any logics

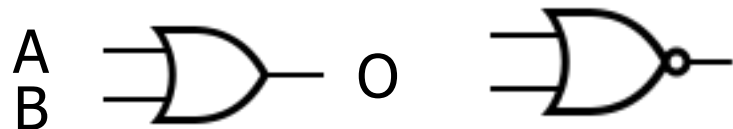
### ► AND, NAND



– NAND

– NOR

### ► OR, NOR



Through Boolean algebra!

# Boolean Algebra

- ▶ Use plus “+” for OR
  - ▶ “logical sum”  $1+0=0+1=1$  (True);  $1+1=2$  (True);  $0+0=0$  (False)
- ▶ Use product for AND ( $a \bullet b$  or implied via  $ab$ )
  - ▶ “logical product”  $0 \bullet 0 = 0 \bullet 1 = 1 \bullet 0 = 0$  (False);  $1 \bullet 1 = 1$  (True)
- ▶ “Bar” to mean complement (NOT)
- ▶ Thus
$$\begin{aligned} & ab + a + \bar{c} \\ = & a \bullet b + a + \bar{c} \\ = & (a \text{ AND } b) \text{ OR } a \text{ OR } (\text{NOT } c) \end{aligned}$$



# Build Combinational Circuits with Basic Logic Gates

Combinational circuits: the ones that the output of the digital circuits depends solely on its inputs; usually built with logic gates without feedback

**Step 1:** Write down truth table of the desired logic

For example build an XOR  
with AND/OR/NOT

| A | B | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Build Combinational Circuits with Basic Logic Gates

Combinational circuits: the ones that the output of the digital circuits depends solely on its inputs; usually built with logic gates without feedback

**Step 2:** Pick the lines with 1 as the output; write them down in *Sum of Minterms (Product)* form;

For example build an XOR with AND/OR/NOT

| A | B | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Minterms*

|                            |       |
|----------------------------|-------|
| $\overline{A}\overline{B}$ | $m_0$ |
| $\overline{A}B$            | $m_1$ |
| $A\overline{B}$            | $m_2$ |
| $AB$                       | $m_3$ |

# Build Combinational Circuits with Basic Logic Gates

Combinational circuits: the ones that the output of the digital circuits depends solely on its inputs; usually built with logic gates without feedback

**Step 3:** Simplify using Laws of Boolean algebra;

For example build an XOR  
with AND/OR/NOT

| A | B | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$O = m_1 + m_2$$

*Minterms*

|                            |       |
|----------------------------|-------|
| $\overline{A}\overline{B}$ | $m_0$ |
| $\overline{A}B$            | $m_1$ |
| $A\overline{B}$            | $m_2$ |
| $AB$                       | $m_3$ |

# Laws of Boolean Algebra

## AND form

$$\overline{XX} = 0$$

$$X0 = 0$$

$$X1 = X$$

$$XX = X$$

$$XY = YX$$

$$(XY)Z = X(YZ)$$

$$X(Y+Z) = XY+XZ$$

$$XY+X = X$$

$$\overline{XY} = \overline{X+Y}$$

## OR form

$$\overline{X+X} = 1$$

$$X+1 = 1$$

$$X+0 = X$$

$$X+X = X$$

$$X+Y = Y+X$$

$$(X+Y)+Z = X+(Y+Z)$$

$$X+YZ = (X+Y)(X+Z)$$

$$(X+Y)X = X$$

$$\overline{X+Y} = \overline{X} \overline{Y}$$

Complementarity

Laws of 0's and 1's

Identities

Idempotent Laws

Commutativity

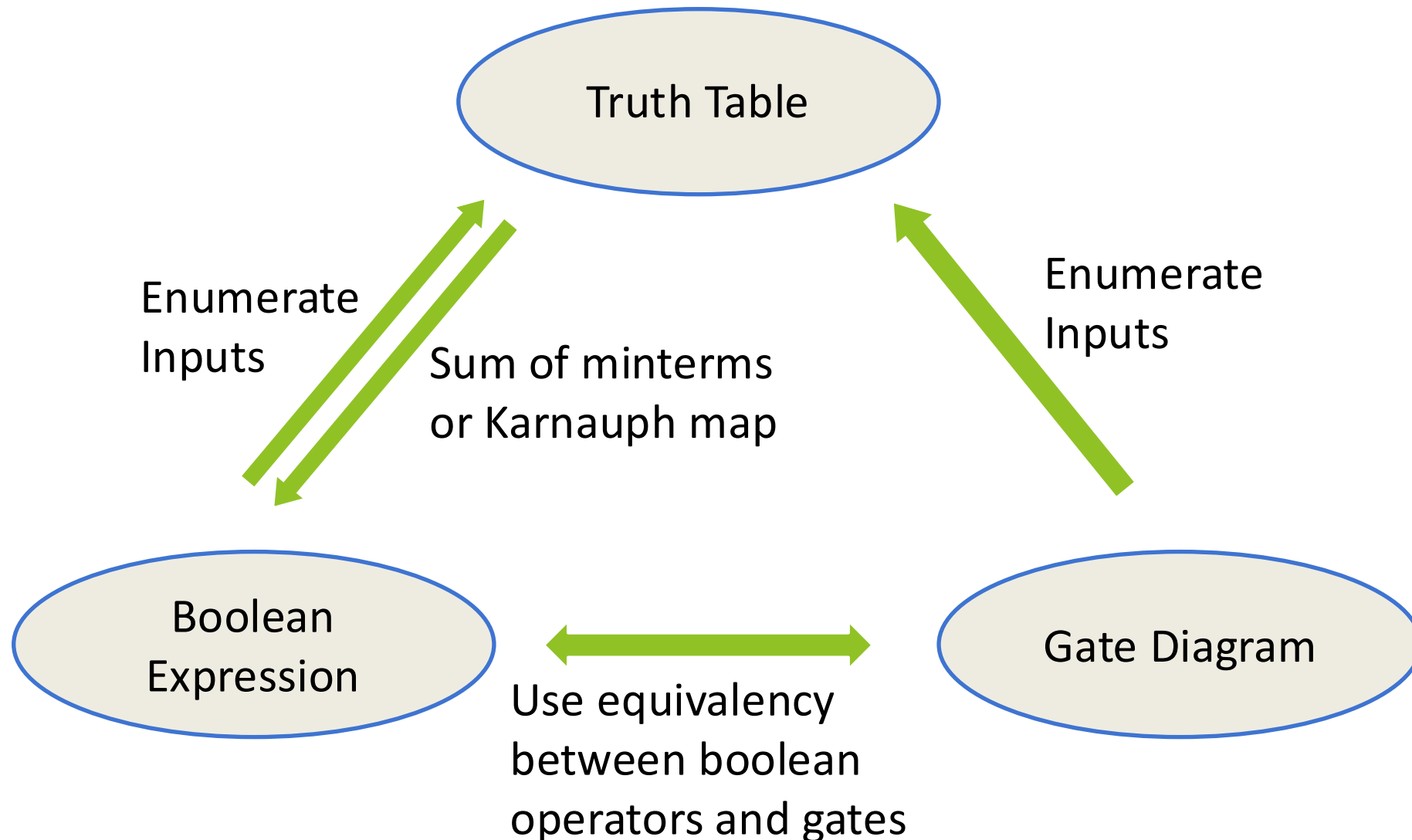
Associativity

Distribution

Absorption

DeMorgan's Law

# Representations of Combinational Logic



# How to build an Adder?

- ▶ Half?
- ▶ Full with carry bit?

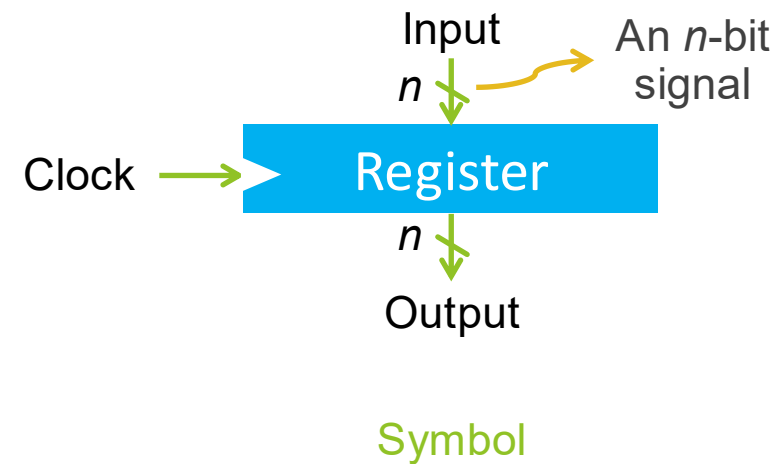
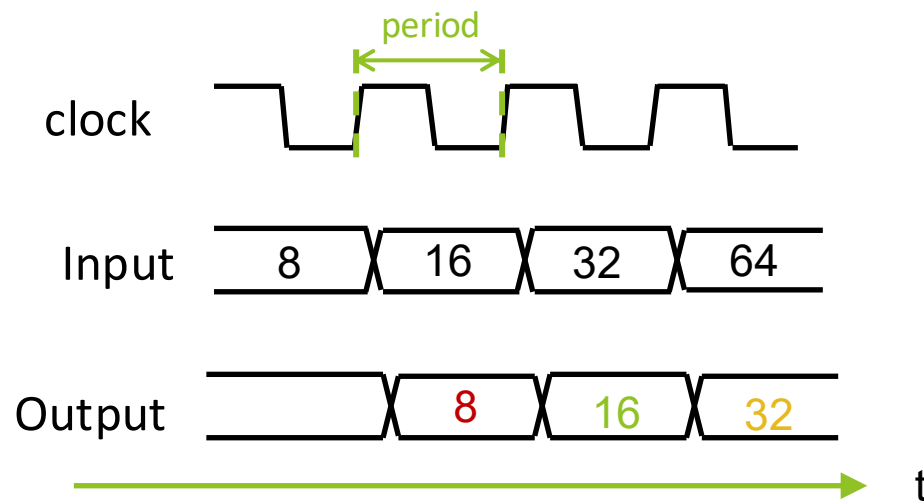
# Synchronous Digital System (SDS)

- ▶ A system that processes digital signals (0s and 1s)
- ▶ Synchronous digital systems consist of two basic types of circuits.
  - ▶ Combinational logic circuits
    - ▶ The outputs solely depend on the input
    - ▶ No way to store information
  - ▶ State Elements
    - ▶ Circuits that store information
    - ▶ E.g., **registers** and memory
    - ▶ Finite state machine
    - ▶ Timing constraints

- Our Goal: Implement a RISC-V processor as a synchronous digital system.
- This SDS should have the capabilities to execute RISC-V instructions.

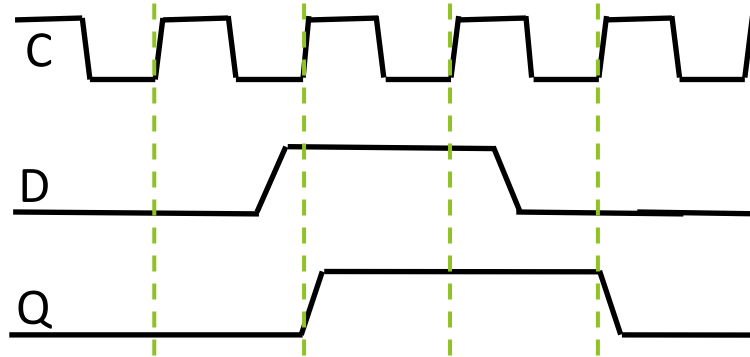
# Registers

- ▶ A digital element that can store and hold state
- ▶ State change governed by a special signal called “clock” (or trigger)
- ▶ Sample input on trigger (can be the positive or negative edge of the clock)
- ▶ At all other times, ignore the input
- ▶ Output follows the value stored
- ▶ **Timing diagram** describes its behavior

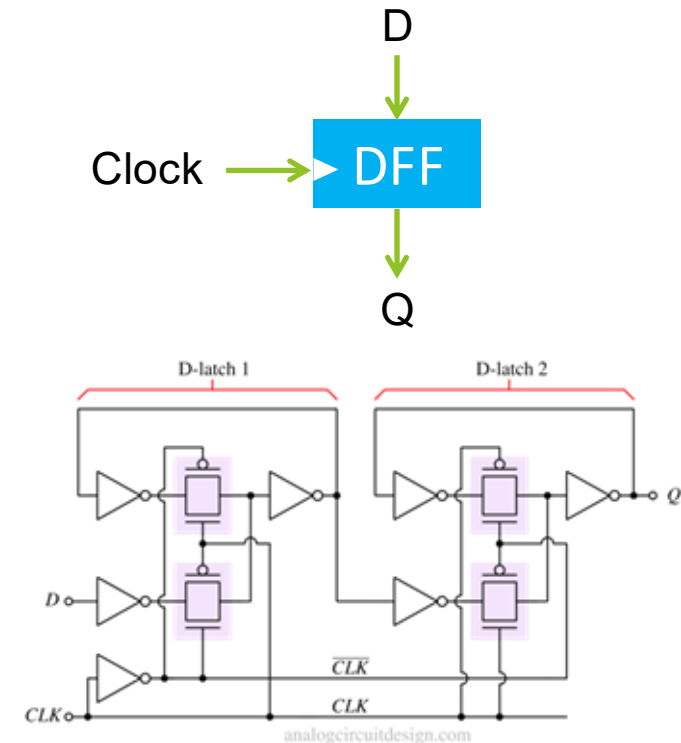
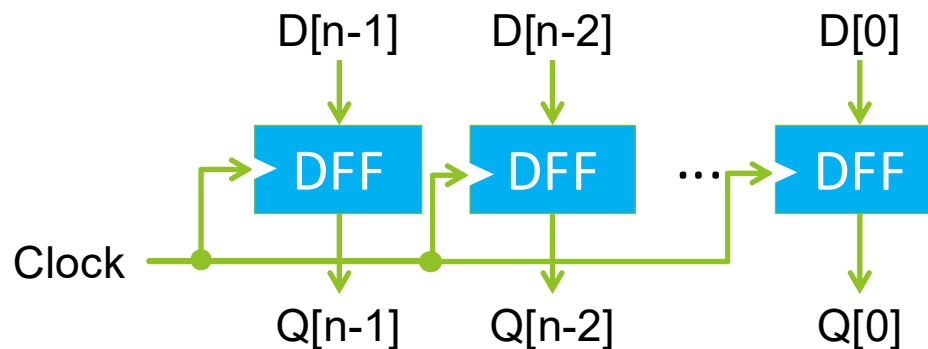


# Inside a Register

- ▶ A register can be implemented by multiple D flip-flops (DFFs)
- ▶ Each DFF can store 1 bit (the same as a 1-bit register)
- ▶ Timing diagram



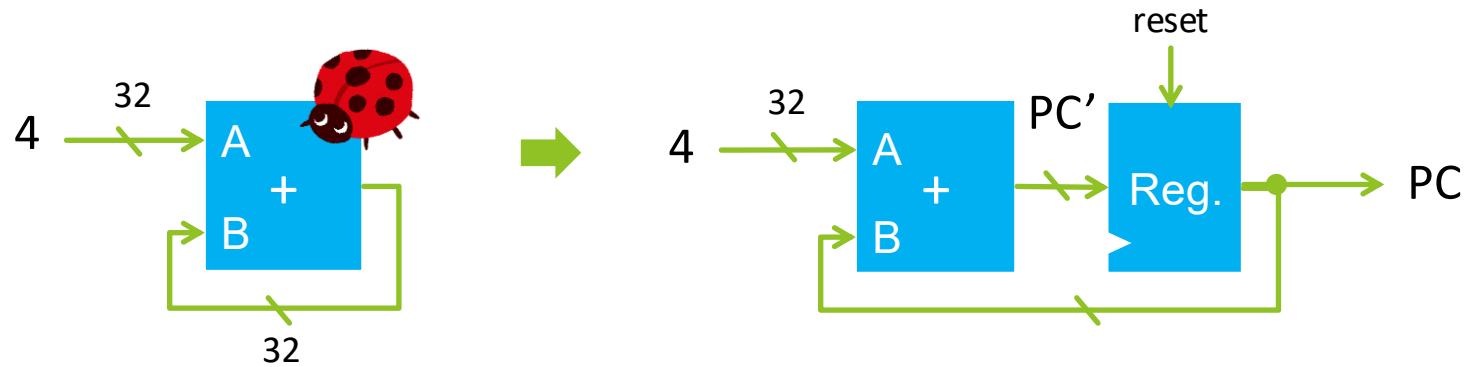
- The DFFs share one clock signal to **synchronize**  
n-bit register (a possible implementation)



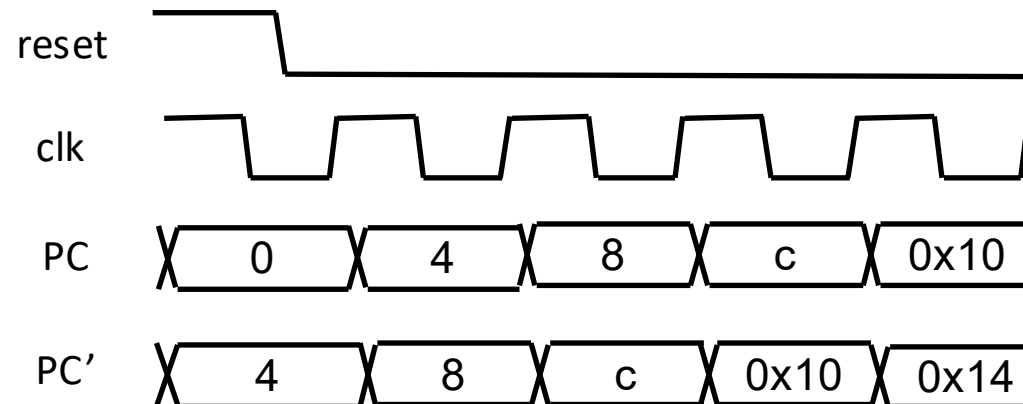
A possible transistor-level implementation of a DFF (optional)

# Combine Combinational Circuits and DFFs

- Synchronous digital circuit can have feedback, e.g., iterative accumulator
  - e.g.  $PC = PC + 4$  without considering branch or jump

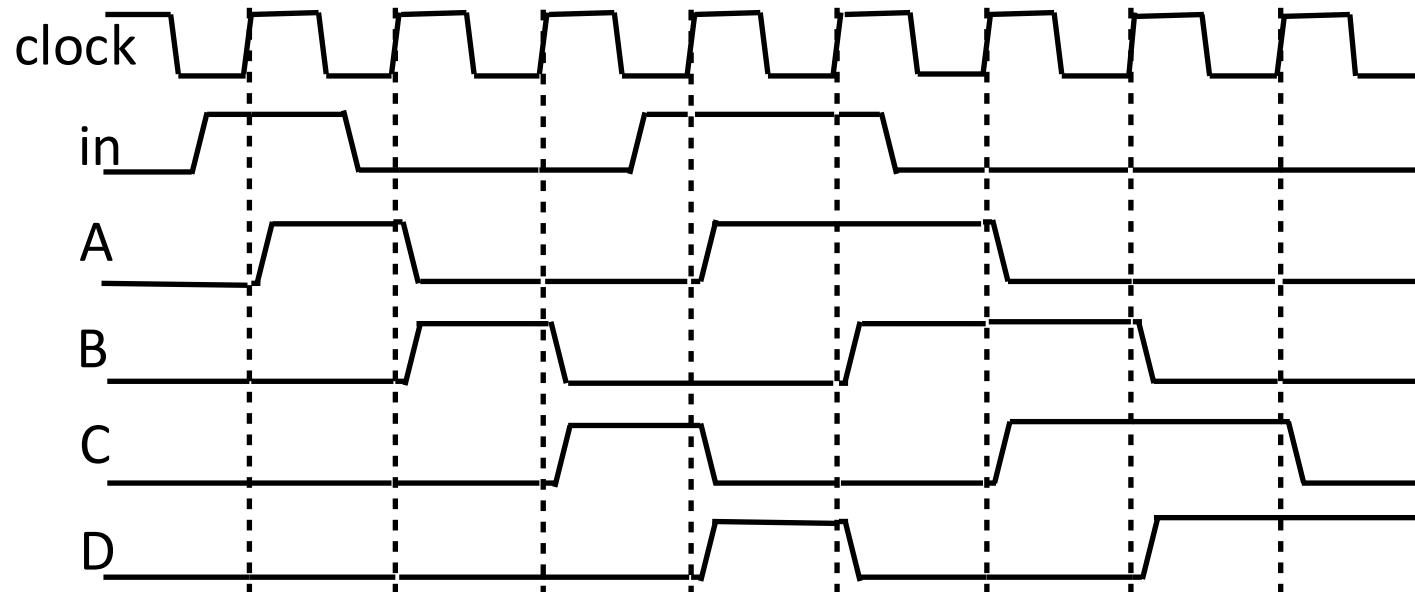
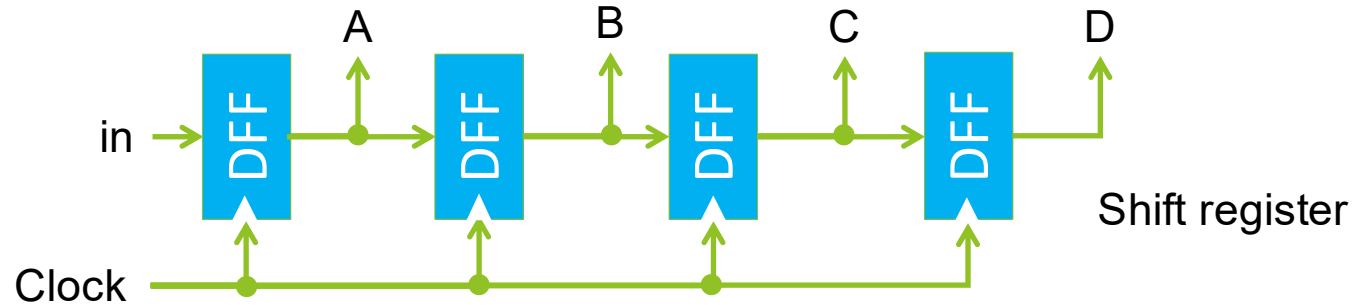


- Timing diagram



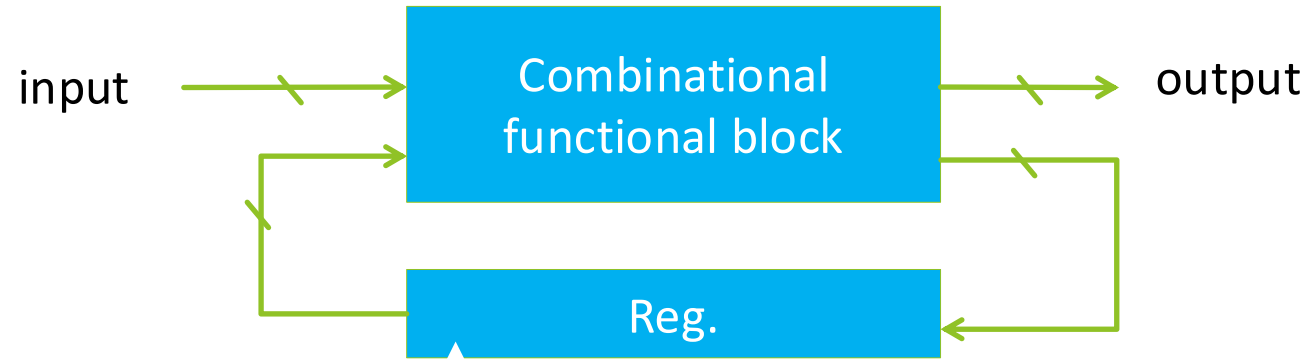
# Combine Combinational Circuits and DFFs

- Synchronous digital circuit that can have multiple stages of DFFs or registers, and **all the registers share one clock**

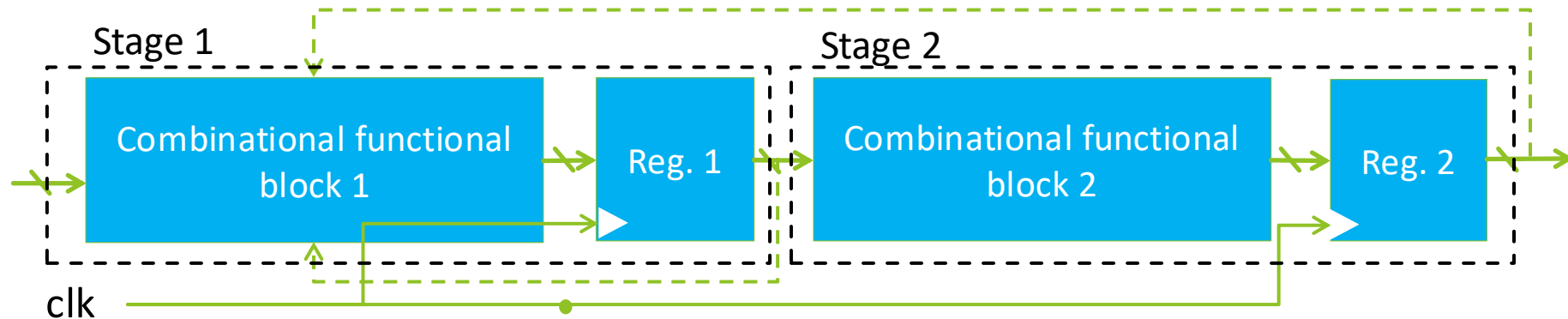


# General form of synchronous circuits

- State element can work together with combinational logic (can have feedbacks)

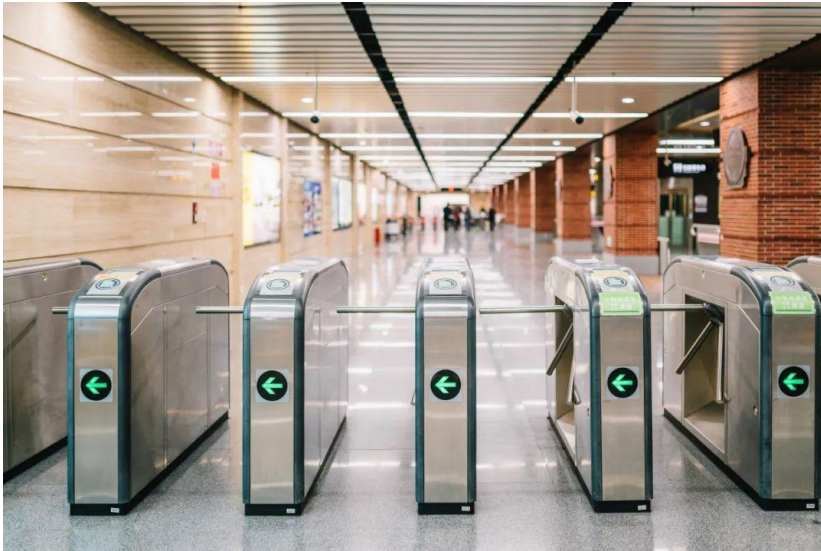


- We can cascade multiple stages of the above synchronous circuit (can have feedbacks across stages)



# Model of synchronous circuits, FSM

- Finite state machine (FSM): a mathematical model of computation
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.
- An FSM is defined by a list of its states, its initial state (entrance), the inputs that trigger each transition, and optional outputs



A turnstile

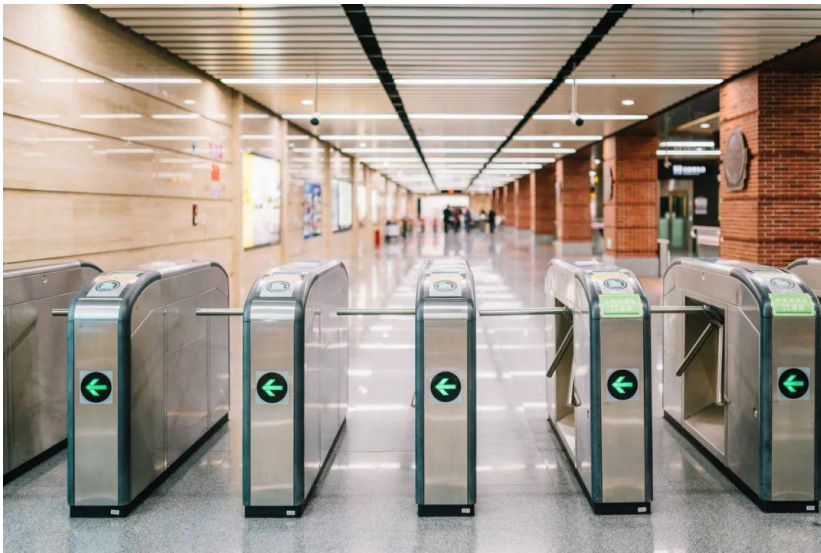
States

Locked

Unlock

# Model of synchronous circuits, FSM

- Finite state machine (FSM): a mathematical model of computation
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.
- An FSM is defined by a list of its states, its initial state (entrance), the inputs that trigger each transition, and optional outputs

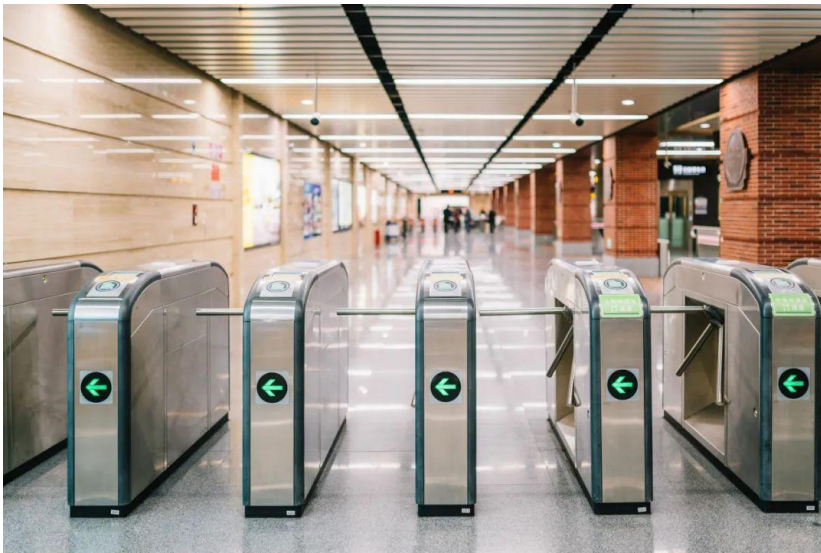


A turnstile

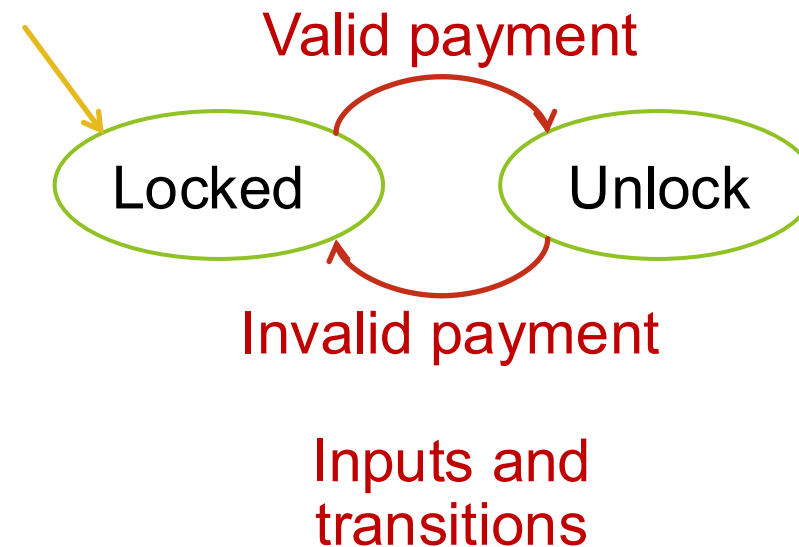


# Model of synchronous circuits, FSM

- Finite state machine (FSM): a mathematical model of computation
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.
- An FSM is defined by a list of its states, its initial state (entrance), the inputs that trigger each transition, and optional outputs

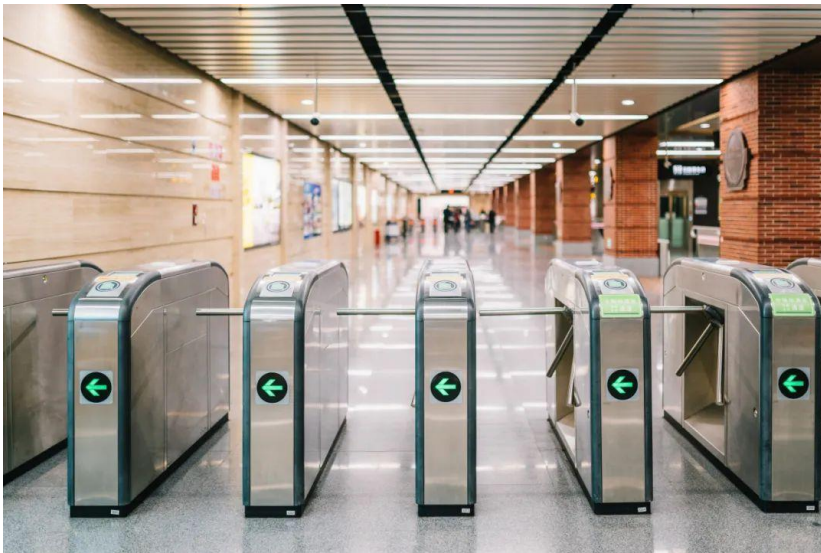


A turnstile

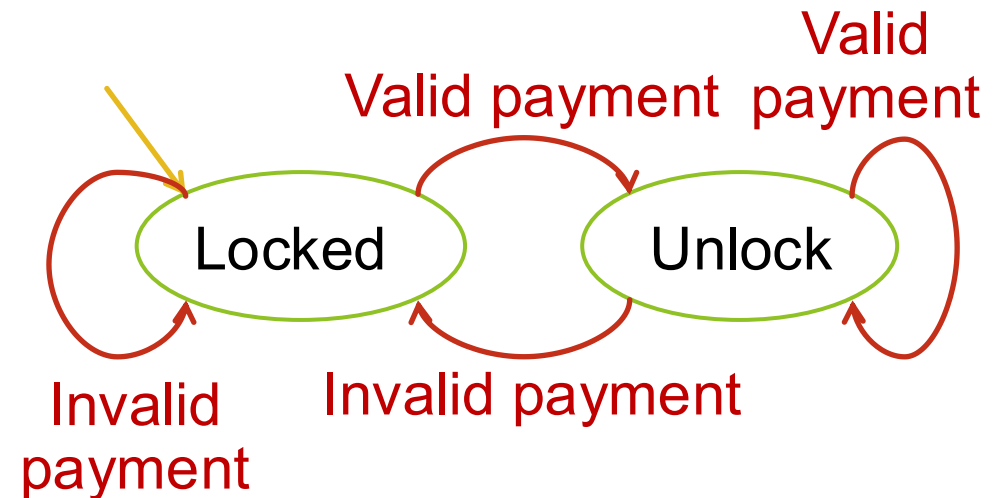


# Model of synchronous circuits, FSM

- Finite state machine (FSM): a mathematical model of computation
- It is an abstract machine that can be in exactly one of a finite number of states at any given time.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.
- An FSM is defined by a list of its states, its initial state (entrance), the inputs that trigger each transition, and optional outputs

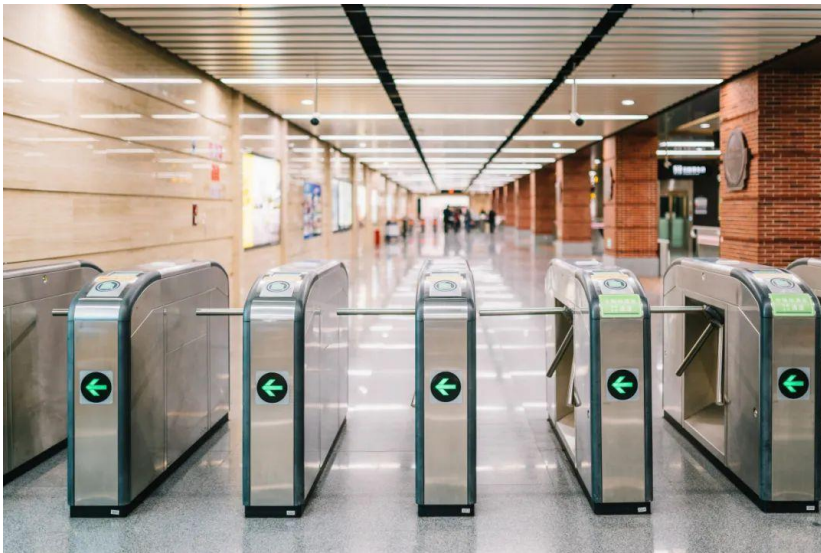


A turnstile

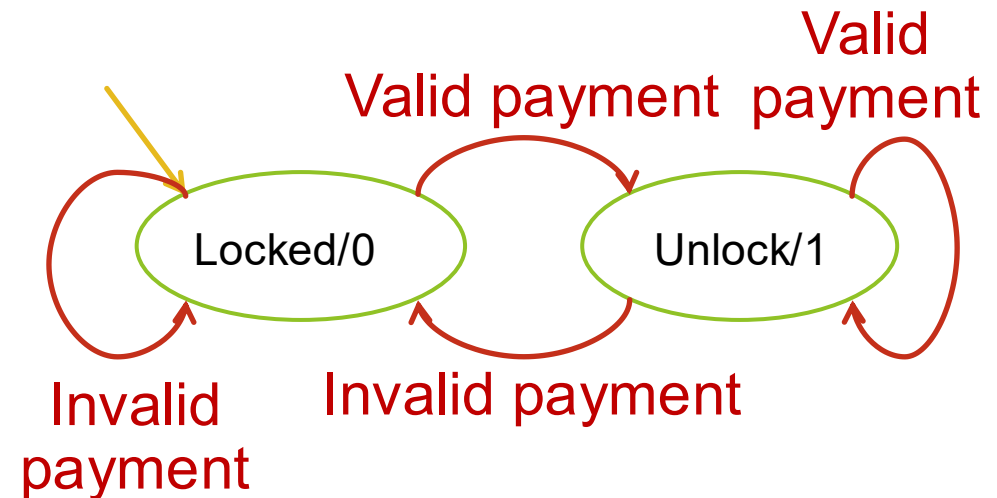


# Model of synchronous circuits, FSM

- Define: State “Locked/hold the bar” as ‘0’, “Unlock/release the bar” as ‘1’; use register/DFF to store the state
- Define: “valid payment” as input ‘1’, “invalid payment” as input ‘0’
- Build a synchronous circuit that implements automatic control of the turnstile



A turnstile

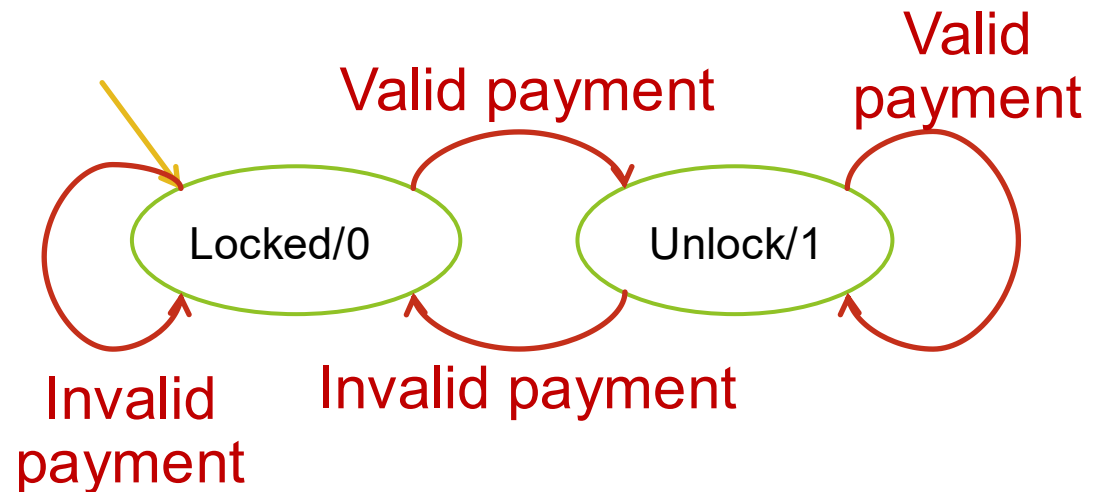


# Model of synchronous circuits, FSM

- Define: State “Locked/hold the bar” as ‘0’, “Unlock/release the bar” as ‘1’; use register/DFF to store the state
- Define: “valid payment” as input ‘1’, “invalid payment” as input ‘0’
- Build a synchronous circuit that implements automatic control of the turnstile
- Truth table of this synchronous circuit

| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

$Q_{k-1}$ : state at the (k-1)th clock cycle  
 $Q_k$ : state at the (k)th clock cycle



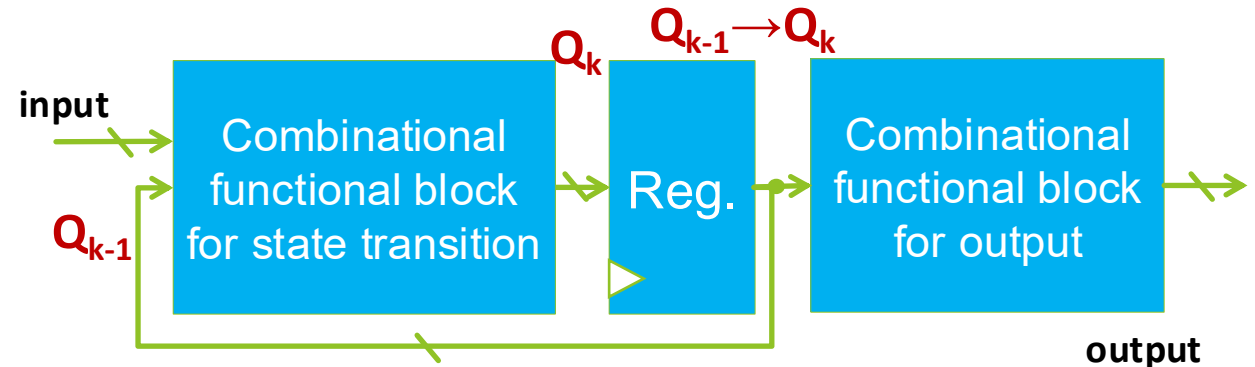
# Model of synchronous circuits, FSM

- Define: State “Locked/hold the bar” as ‘0’, “Unlock/release the bar” as ‘1’; use register/DFF to store the state
- Define: “valid payment” as input ‘1’, “invalid payment” as input ‘0’
- Build a synchronous circuit that implements automatic control of the turnstile
- Truth table of this synchronous circuit

| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

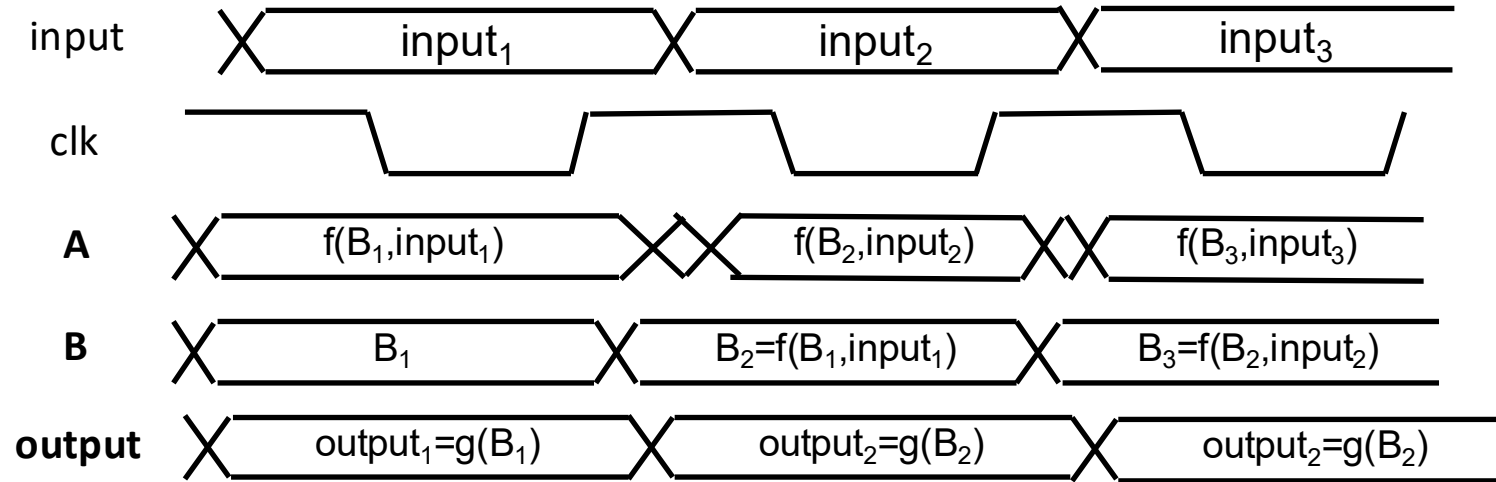
$Q_{k-1}$ : state at the (k-1)th clock cycle  
 $Q_k$ : state at the (k)th clock cycle

Template of a synchronous circuit that implements the FSM



# Model of synchronous circuits, FSM

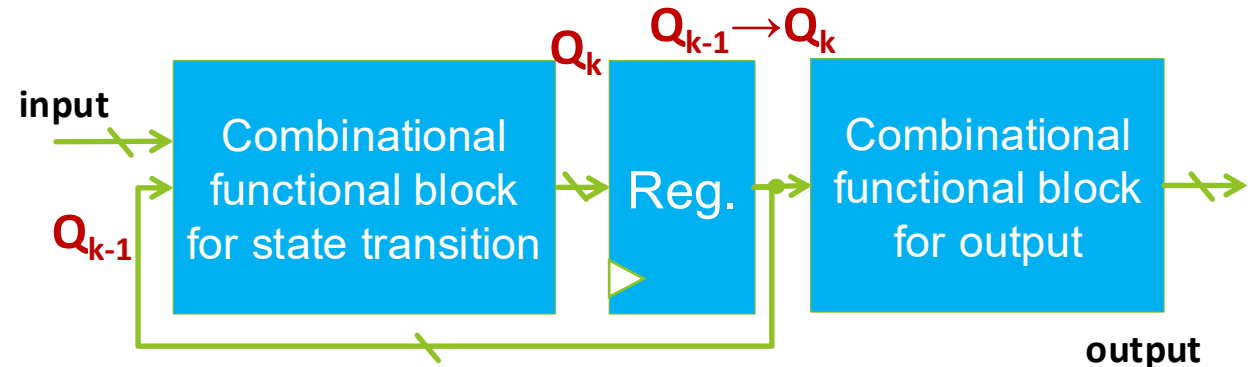
- Timing diagram (general case)



| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

$Q_{k-1}$ : state at the (k-1)th clock cycle  
 $Q_k$ : state at the (k)th clock cycle

Template of a synchronous circuit that implements the FSM



# Model of synchronous circuits, FSM

- Build the circuits

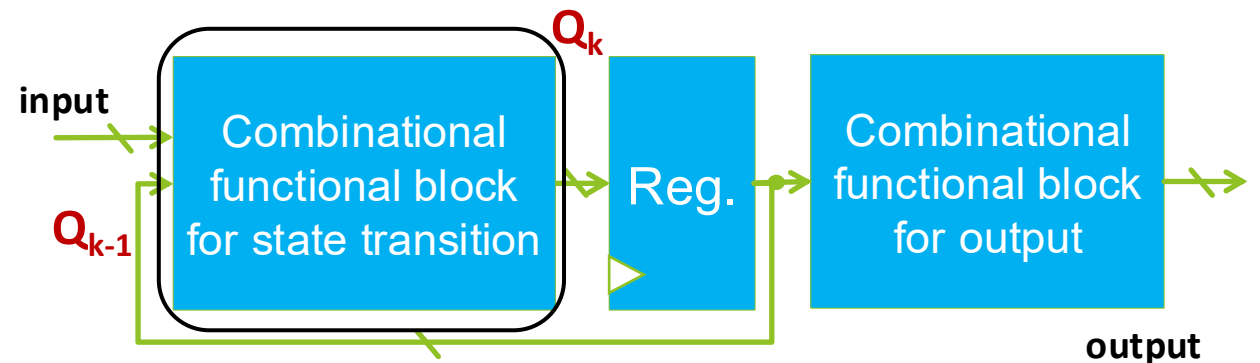
| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

State transition logic

$$Q_k = f(\text{input}, Q_{k-1})$$

↓  
 $Q_k = \text{input}$

Template of a synchronous circuit that implements the FSM



# Model of synchronous circuits, FSM

- Build the circuits

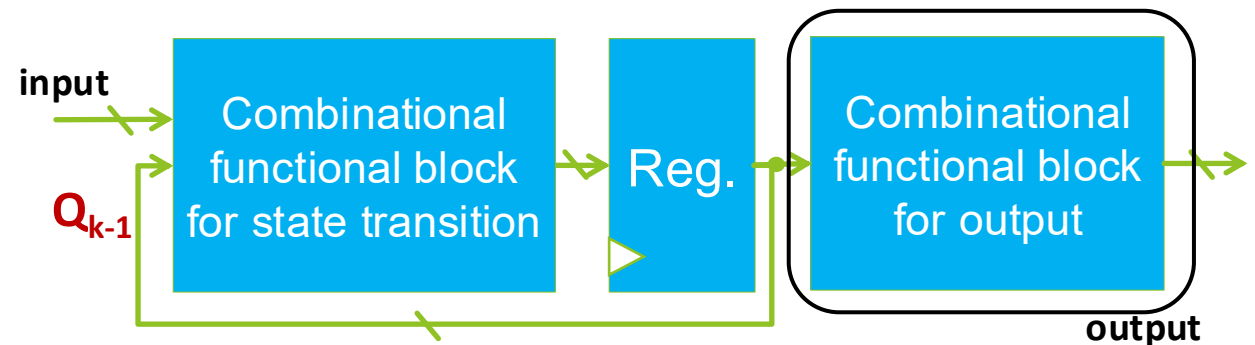
| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

State transition logic

Output= $g(Q_k)$

↓  
Output= $Q_k$

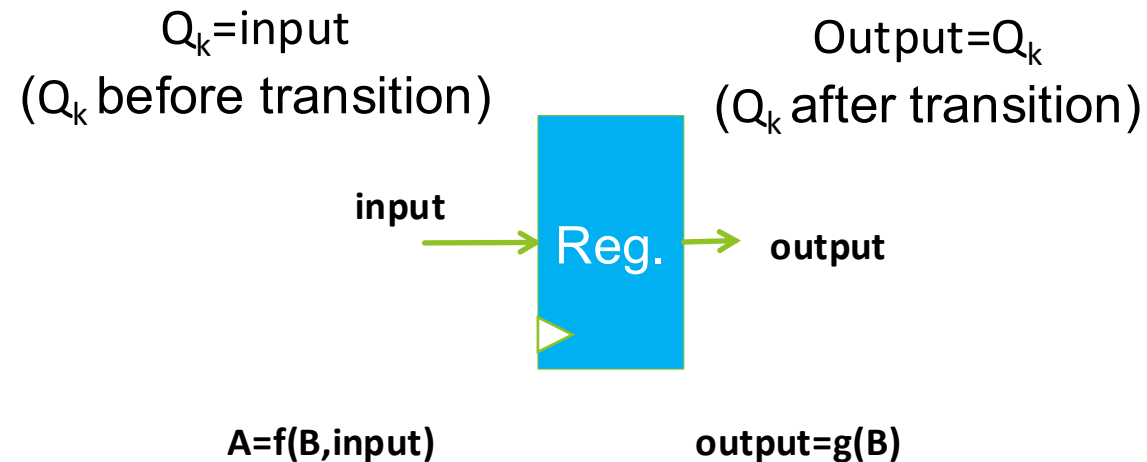
Template of a synchronous circuit that implements the FSM



# Model of synchronous circuits, FSM

- Build the circuits

| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |



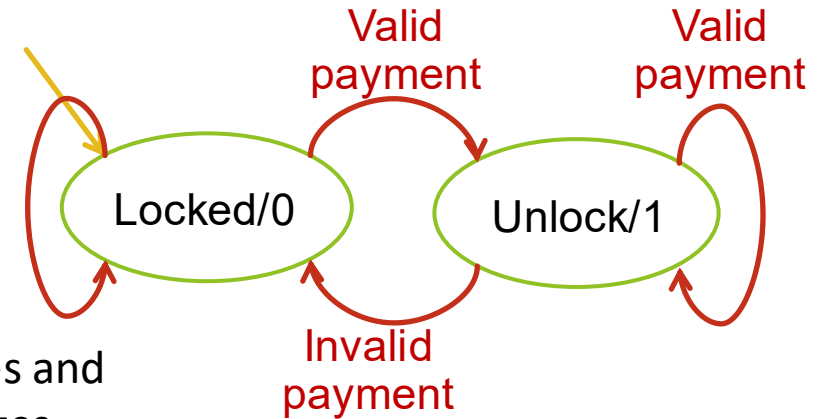
# Model of synchronous circuits, FSM

| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

Enumerate  
Inputs & states



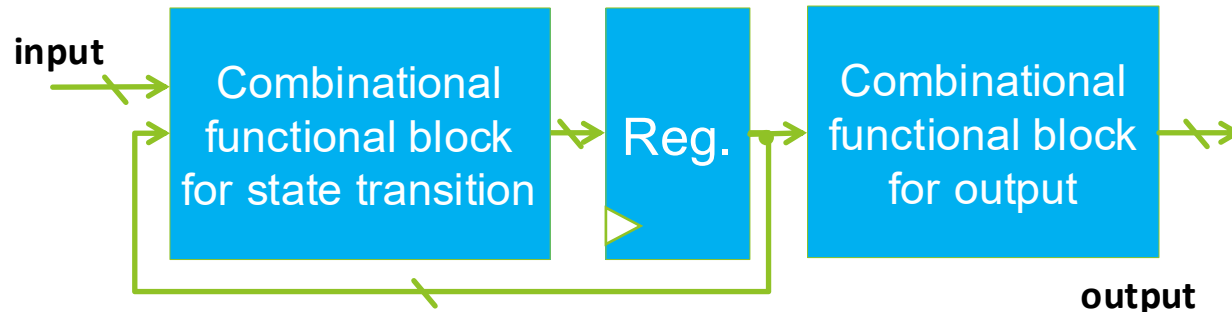
Draw states as nodes and  
transitions as edges



Enumerate  
Inputs & states

Sum of minterms  
(or Karnaugh map)

Enumerate Inputs &  
states & transitions



# Moore machine vs. Mealy machine

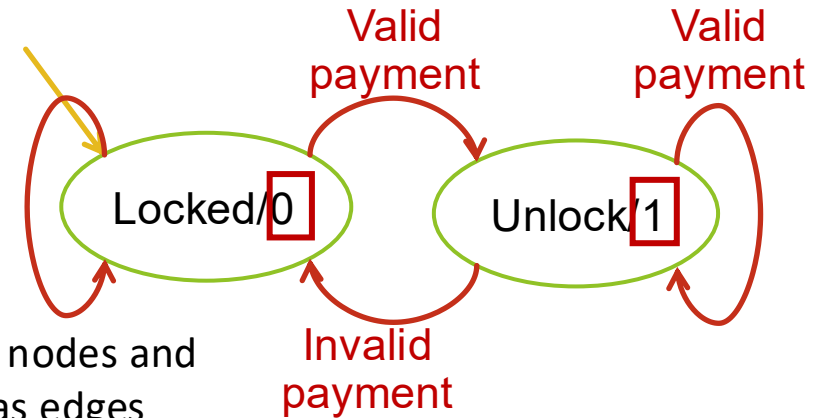
- Moore machine

| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |

Enumerate  
Inputs & states

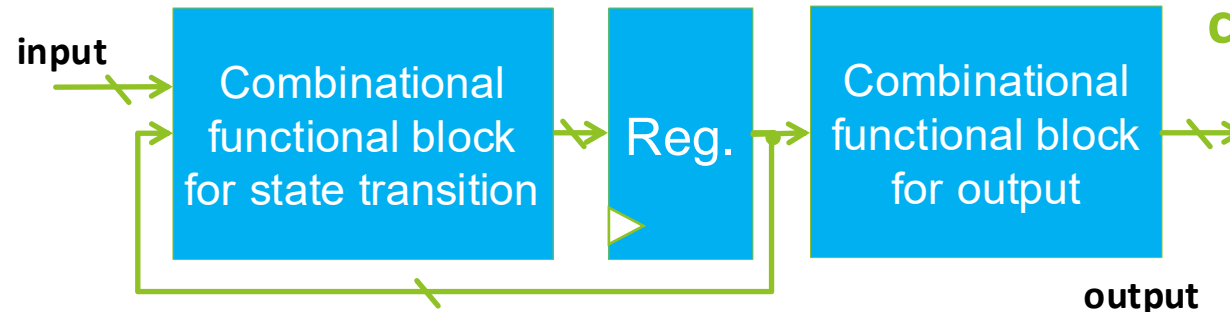


Draw states as nodes and  
transitions as edges



Sum of minterms  
(or Karnaugh map)

Enumerate  
Inputs & states

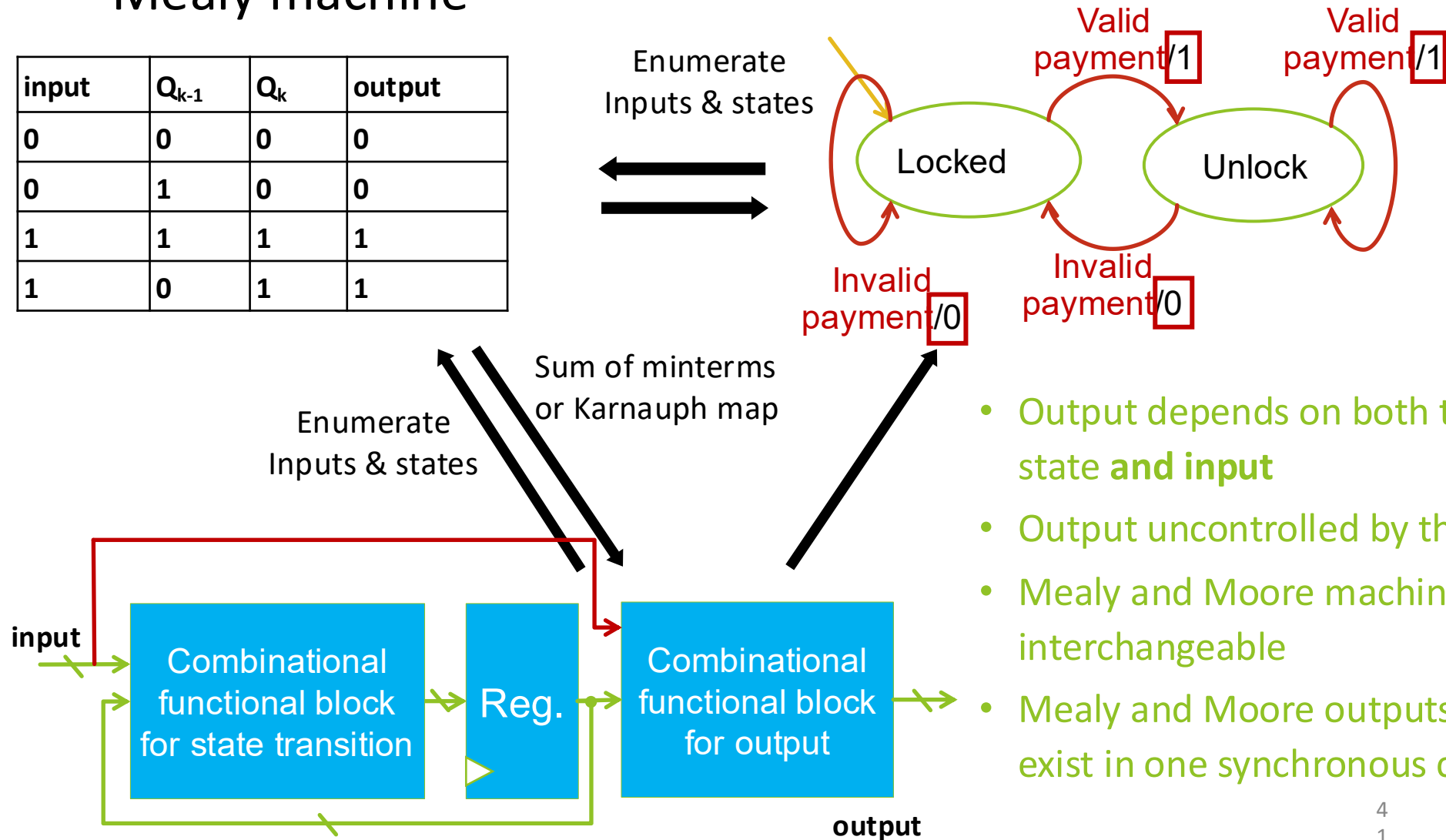


- Output depends solely on the current state
- Output controlled by the clock

# Moore machine vs. Mealy machine

- Mealy machine

| input | $Q_{k-1}$ | $Q_k$ | output |
|-------|-----------|-------|--------|
| 0     | 0         | 0     | 0      |
| 0     | 1         | 0     | 0      |
| 1     | 1         | 1     | 1      |
| 1     | 0         | 1     | 1      |



- Output depends on both the current state **and** input
- Output uncontrolled by the clock
- Mealy and Moore machines are interchangeable
- Mealy and Moore outputs can co-exist in one synchronous circuit

# Summary for building synchronous circuits

- ▶ Step 1: Draw finite state machine of the desired function (we ignore the initialization)
- ▶ Step 2: Define/assign binary numbers to represent the states, the inputs and the outputs
- ▶ Step 3: Write down the truth table (enumerate input/previous state (and current state) and their corresponding current state (and output))
- ▶ Step 4: Use template and decide the combinational block for state transition and output logic
- ▶ Otherwise, we can also use simple synchronous circuit blocks to build larger synchronous circuit like for the multi-bit adder