COUNTERs

Digital Logic Design (CC131)

BSCS 2nd Semester Spring-2026

By

Syed Zulfiqar Ali Shah,

Associate Professor (CS) / HoD

Govt. College of Management Sciences, Kohat

www.linkedin.com/in/syedzulfiqaralishah

COUNTER

- A Counter is a Sequential Logic Circuit that changes its output in a fixed sequence every time a clock signal arrives.

- A counter is built using Flip-flops. Each flip-flop stores 1 bit, and together they represent a binary number.

For example, a 3-bit binary counter counts like this:

Clock Pulse	Output
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111
Next	000 again

The state is changing at each clock pulse.

FINITE STATE MACHINE

- A Counter is an example of a state machine.

- A State Machine is a Sequential Logic Circuit having a limited (finite) number of states and these states occur in a predefined order.

- The number of states is called the Modulus.

TYPES OF STATE MACHINEs

Two basic types of State Machines are;

1. **Moore State Machine**

   Output depends only on the internal present state.

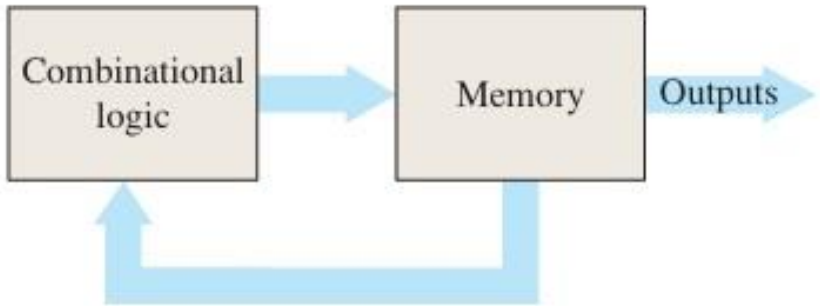
2. **Mealy State Machine**

   Output depends both on the internal present state and on the inputs.

1. MOORE STATE MACHINE

- A Moore State Machine consists of combinational logic that determines the sequence and memory.

- The combinational logic is a gate array with outputs that determine the next state of the flip-flops in the memory.



(a) Moore machine

EXAMPLE: A Moore State Machine for Filling Bottles with 25 Tablets

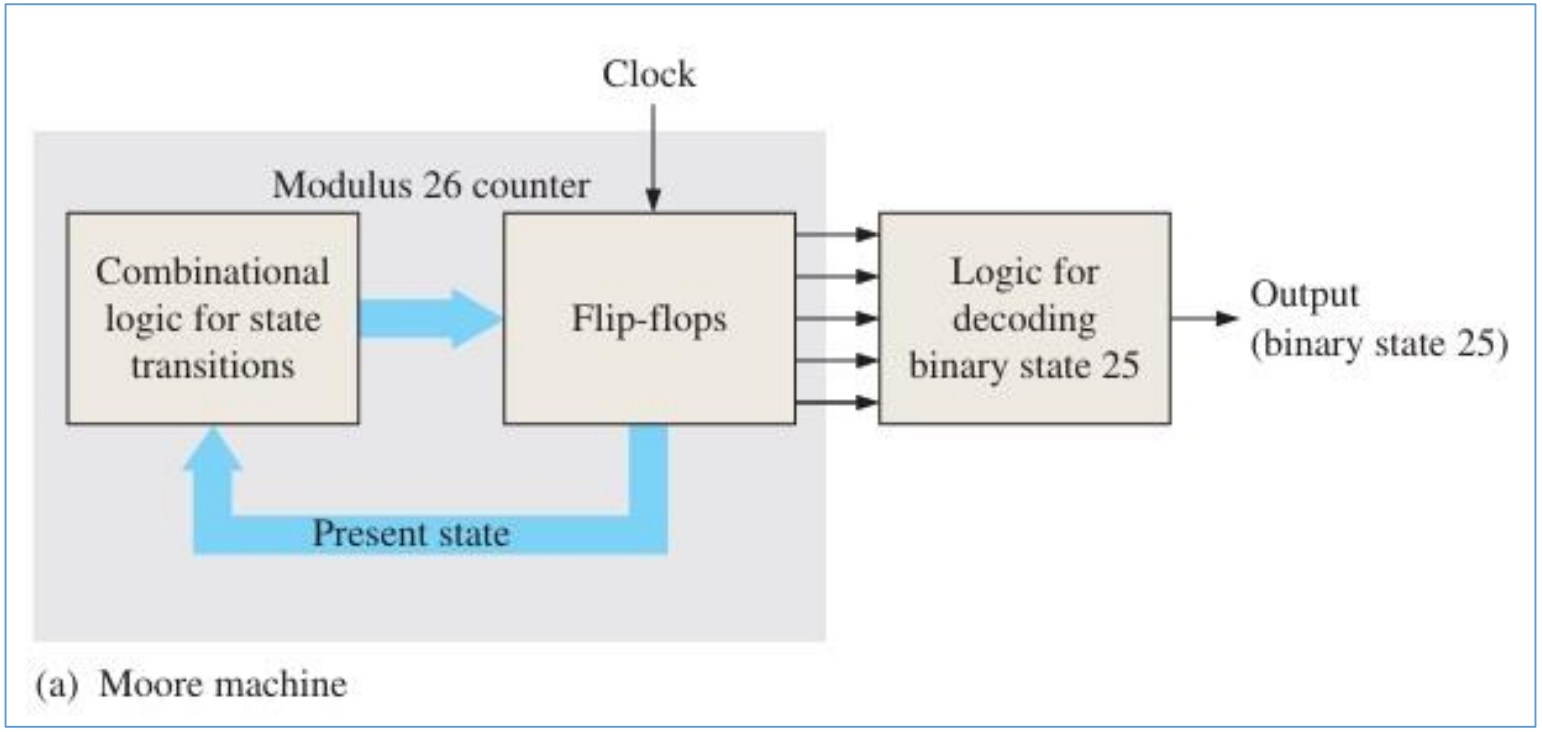
Goal of the System

- The machine must:
- Put exactly 25 tablets into each bottle
- Stop automatically after 25 tablets
- Wait until the next empty bottle arrives
- Repeat the process again

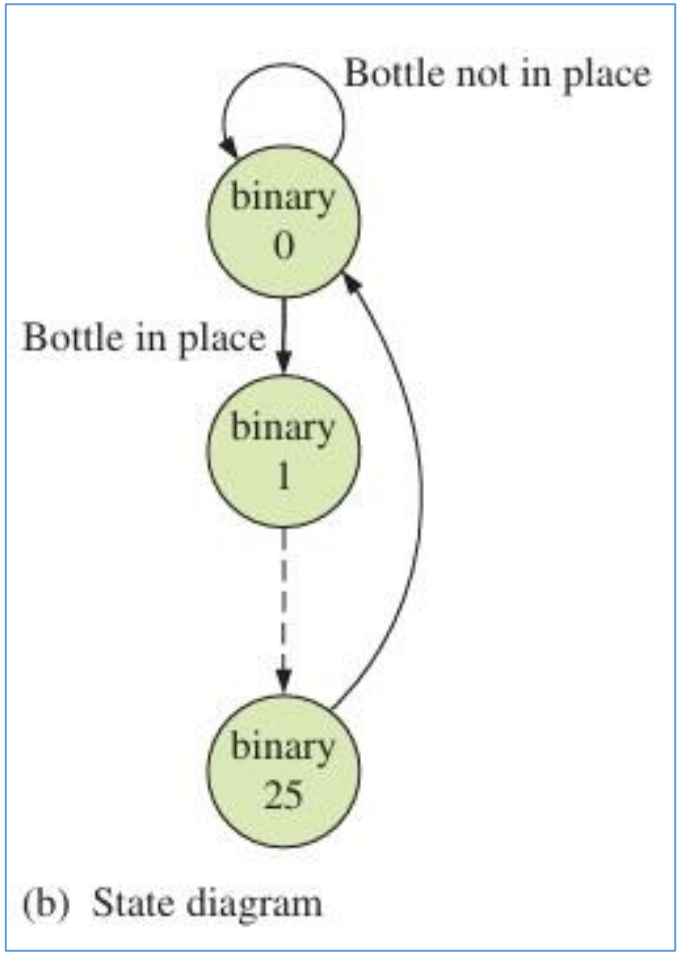
The system uses:

- **Flip-flops** → to store the current count/state
- **Combinational logic** → to decide the next state
- **Clock pulses** → each pulse inserts one tablet
- **Moore Machine** → output depends only on the present state

EXAMPLE of a Moore State Machine



(a) Moore machine



(b) State diagram

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Understanding the States

The counter has **26 states**:

0,1,2,3,…,25

So it is called a **Modulus-26 counter**.

- **State 0** → waiting/reset state
- **State 1** → 1 tablet inserted
- **State 2** → 2 tablets inserted
- . . . . . .
- **State 25** → 25 tablets inserted

After state 25, the system goes back to state 0.

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step-by-Step Working

Step 1 — No Bottle Present

The machine starts in:

STATE 0

At this state:

- No tablets are entering
- Clock is stopped
- Machine is waiting for a bottle

This is the **rest state**.

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step 2 — Bottle Arrives

A sensor detects a bottle.

The signal becomes:

$$\text{BOTTLE IN PLACE} = 1$$

Now:

- Clock is enabled
- Counter starts running

At the next clock pulse:

$$0 \rightarrow 1$$

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step 3 — First Tablet Inserted

When the machine enters:

STATE 1

the first tablet is dropped into the bottle.

Important idea:
In a Moore machine, the output depends only on the **current state**.
So:

- Being in state 1 means:
- "1 tablet has been inserted."

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step 4 — Counter Continues Counting

Each clock pulse:

- Inserts one more tablet
- Advances the counter by one state

Sequence:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow \cdots \rightarrow 25$$

Examples:

- State 5 → 5 tablets inserted
- State 12 → 12 tablets inserted
- State 20 → 20 tablets inserted

The flip-flops continuously store the present count.

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

State Transition Behavior

The next state depends only on the present state:

$$\text{Next State} = \text{Present State} + 1$$

until state 25 is reached.

This is why it is called a **Moore machine**:

- Output depends only on state
- Not directly on external inputs

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step 5 — Reaching State 25

When the counter reaches:

$$25 = 11001_2$$

The decoding logic recognizes this binary pattern.
The output logic now generates a control signal.
This signal:

- Stops tablet flow
- Stops the clock
- Declares the bottle full

So:

State 25 $\Rightarrow$ Bottle Filled

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step 6 — Counter Resets

After state 25:

$$25 \rightarrow 0$$

The machine returns to the waiting state.

Now:

- Current bottle is removed
- Next empty bottle is awaited

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Step 7 — Waiting for Next Bottle

If no bottle is present:

- System remains in state 0
- No counting occurs

When another bottle arrives:

$$0 \rightarrow 1$$

and the entire cycle repeats.

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Why This is a Moore Machine

A Moore machine's output depends only on the present state.
Examples here:

- State 0 → waiting
- State 25 → stop filling

The output is determined entirely by the state stored in the flip-flops.

The input ("bottle present") only helps the machine leave state 0; it does not directly generate the output.

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Simple Real-Life Interpretation

Think of the states as labels:
The machine simply moves through these labels one by one with every clock pulse.

State	Meaning
0	Waiting for bottle
1	1 tablet inserted
2	2 tablets inserted
…	…
25	Bottle full

EXAMPLE of a Moore State Machine for Filling Bottles with 25 Tablets

Key Educational Insight

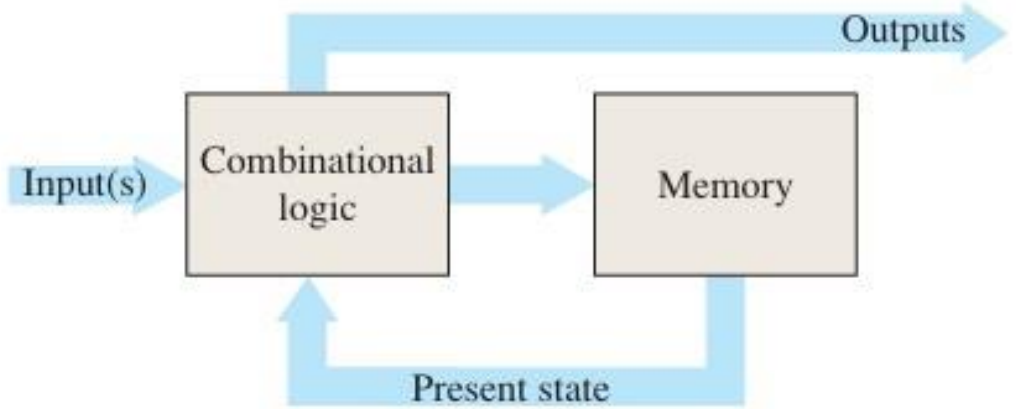
This example shows that:

- A **counter** can behave as a **finite state machine**
- Each count value represents a **state**
- The machine changes states synchronously with the clock
- Outputs are generated from states

So a counter is actually one practical implementation of a Moore FSM.

2. MEALY STATE MACHINE

- A Mealy State Machine also consists of combinational logic and memory. But in addition, there are inputs to the combinational logic.
- The outputs come directly from the combinational logic and not from the memory.
- Thus the output depends on both inputs and the present state.



(b) Mealy machine

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Imagine a factory machine that drops tablets into bottles one by one.

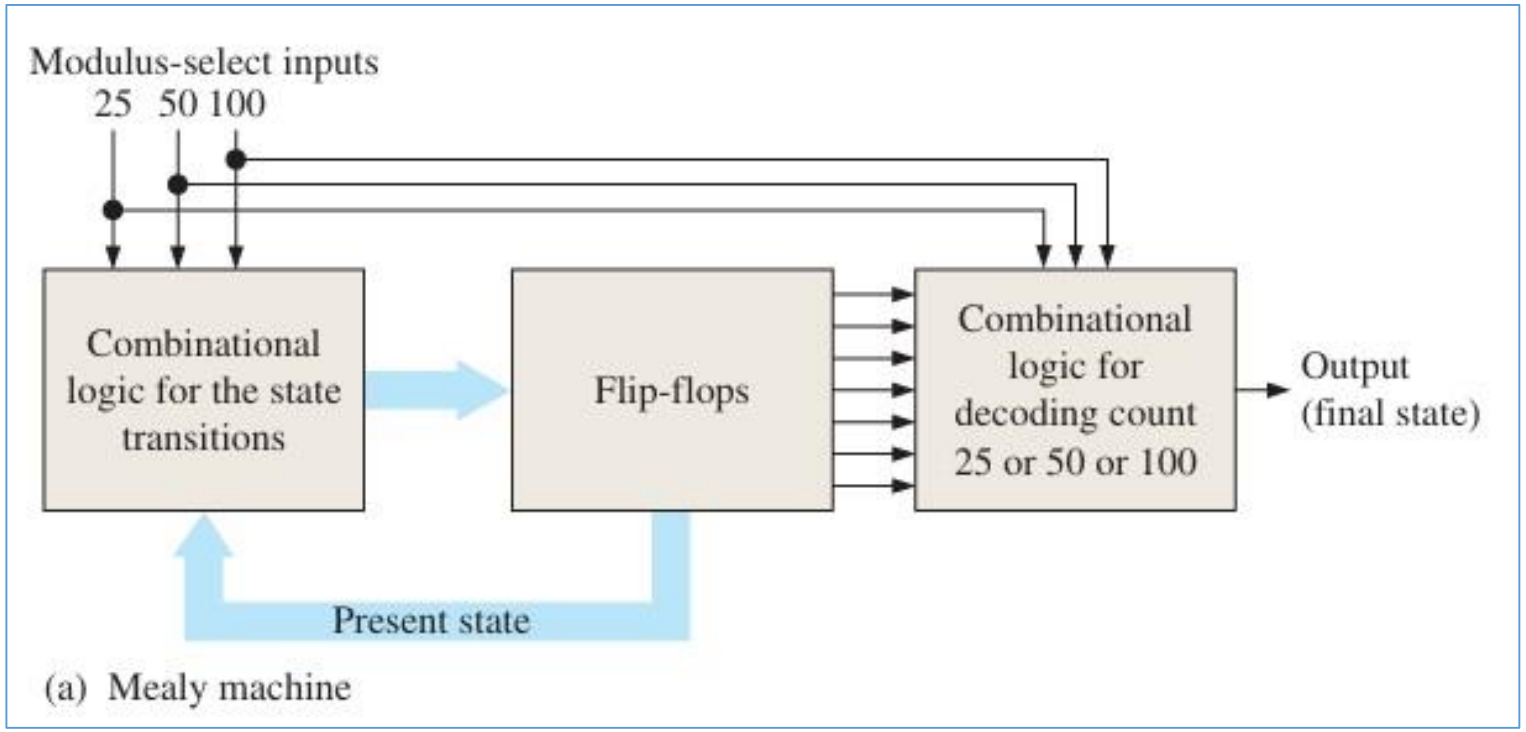
The machine must support **three bottle sizes**:

- 25-tablet bottle
- 50-tablet bottle
- 100-tablet bottle

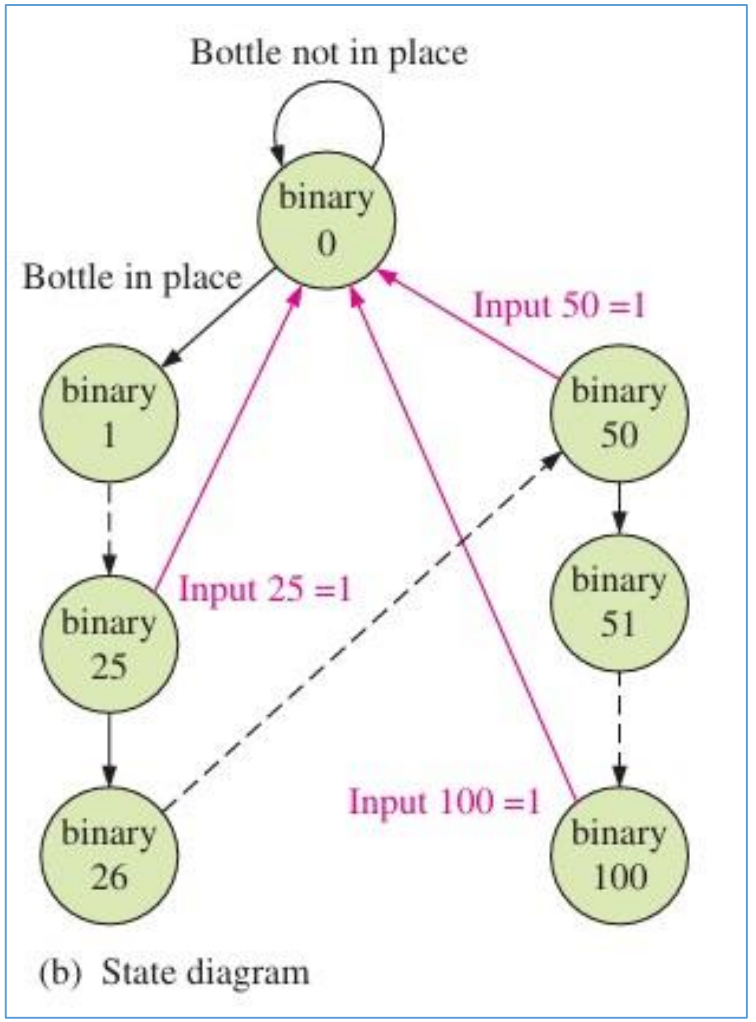
The job of the state machine is:

1. Count the tablets entering the bottle
2. Stop the flow when the required number is reached
3. Reset for the next bottle

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets



(a) Mealy machine



(b) State diagram

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Step-by-Step Operation

Step 1 — Select the Bottle Size

Before filling starts, the operator selects the bottle type:

- Select 25
- Select 50
- Select 100

These selection lines are called **modulus-select inputs** because they decide the modulus (maximum count) of the counter.

Example:

- If 25 is selected → counter must count from 0 to 24
- If 50 is selected → counter must count from 0 to 49
- If 100 is selected → counter must count from 0 to 99

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Step 2 — Counter Starts from State 0

When an empty bottle arrives:

- The counter is reset to 0
- Tablet flow begins
- Each clock pulse represents one tablet entering the bottle

So the counter changes state like this:

$$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \ldots$$

Each state represents:

"How many tablets have entered so far."

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Step 3 — Counter Continues Counting

As tablets enter:

- Every clock pulse increments the counter
- The machine continuously checks:
  - current count (present state)
  - selected bottle size (modulus-select input)

Example for a 25-tablet bottle:

$$0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \rightarrow 24$$

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Step 4 — Machine Detects the Terminal Count

The combinational logic compares:

- Present state
- Selected modulus

Examples:

- If 25-tablet mode is selected:
  - detect count 24
- If 50-tablet mode is selected:
  - detect count 49
- If 100-tablet mode is selected:
  - detect count 99

At that moment, the next incoming tablet would complete the bottle.

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Step 5 — Output Signal is Generated

When the terminal count is detected:

- The machine generates an output signal
- This signal:
  - stops tablet flow
  - stops the clock
  - resets the counter for the next bottle

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Why This is a Mealy Machine

In this system, the output depends on:

1. **Present state** (current count)
2. **External input** (selected bottle size)

For example:

- State 24 may or may not stop the machine:
  - stop if 25-mode selected
  - continue if 50-mode selected

So the same state can produce different outputs depending on the input. That is exactly the definition of a Mealy machine.

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Important Observation

The counter hardware itself may remain the same.

What changes is:

- the terminal-count detection logic

The modulus-select inputs tell the logic:

"At which count should the machine stop?"

EXAMPLE: A Mealy State Machine for Filling Bottles with 25, 50, 100 Tablets

Concrete Example

Case: 50-tablet bottle selected

Initial State

Count=0

Counting

$$0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \rightarrow 48 \rightarrow 49$$

Detection

Logic checks:

- Present state = 49
- Selected mode = 50

Condition satisfied.

Action

Machine:

- stops tablet flow
- resets counter
- waits for next bottle

Types of Counters

1. Asynchronous (Ripple) Counter

2. Synchronous Counter

Types of Counters

1. Asynchronous (Ripple) Counter

- Flip-flops do **not** receive the clock at the same time.
- Output of one flip-flop drives the next.
- Simpler but slower.

2. Synchronous Counter

- All flip-flops receive the same clock pulse together.
- Faster and more reliable.
- JK flip-flops are used to illustrate synchronous counters
- D flip-flops can also be used but require more logic.

2-Bit Asynchronous Binary Counter

Setup

- Two D flip-flops (FF0, FF1) connected in series
- Clock (CLK) feeds **only FF0** (the LSB)
- FF0's complemented output ($\bar{Q}_0$) triggers FF1
- Both start RESET (Q = 0)

2-Bit Asynchronous Binary Counter

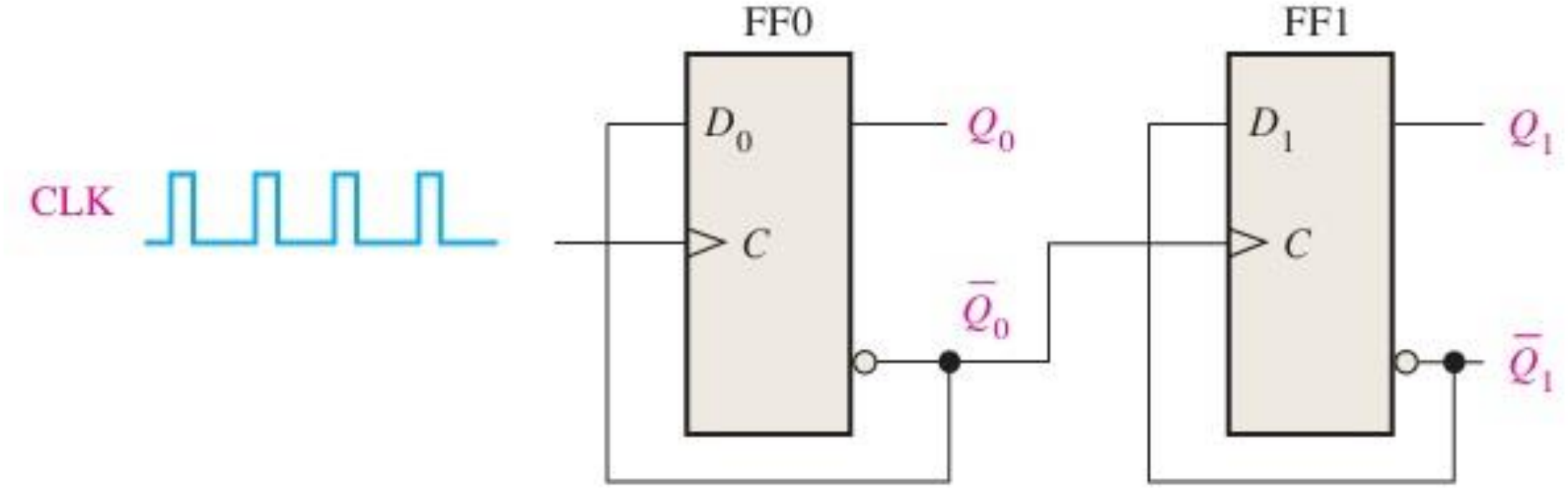


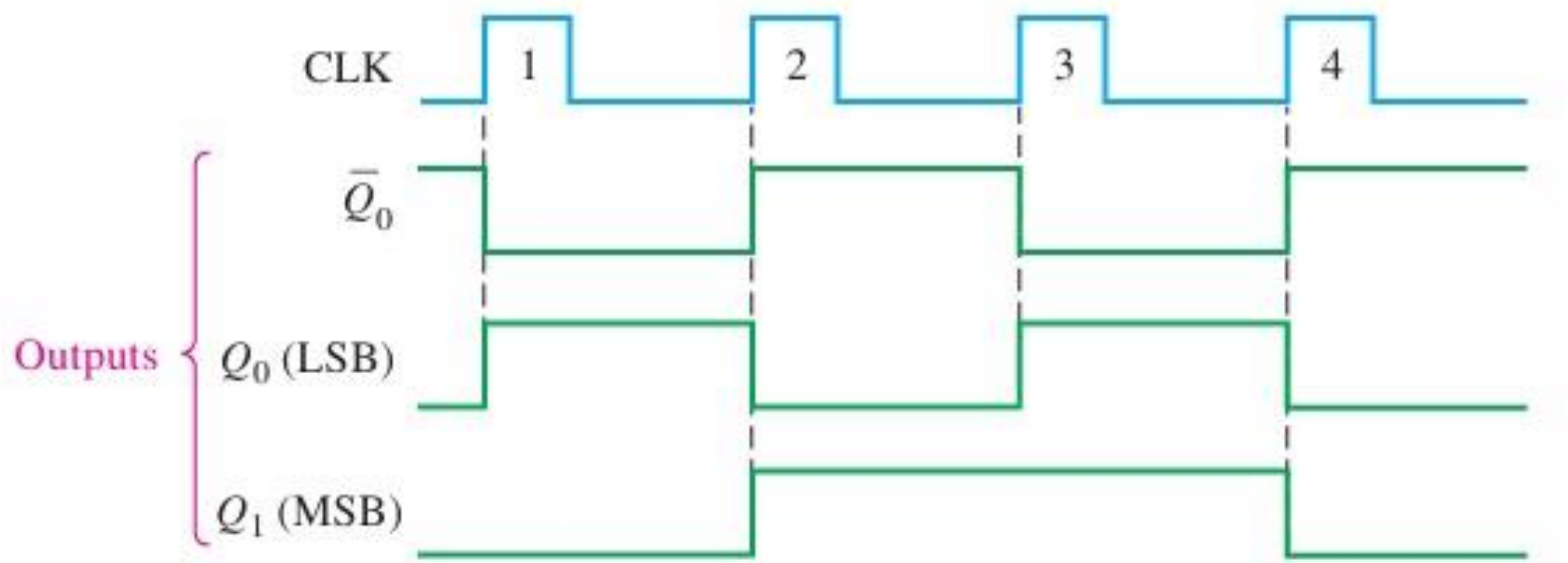
Figure 9-4

2-Bit Asynchronous Binary Counter

Step-by-Step Operation

- **CLK1 hits** → FF0 toggles → Q_0 goes HIGH (1)
- **CLK2 hits** → FF0 toggles → Q_0 goes LOW (0), so $\overline{Q_0}$ goes HIGH → **this triggers FF1** → Q_1 goes HIGH (1)
- **CLK3 hits** → FF0 toggles again → Q_0 goes HIGH (1), $\overline{Q_0}$ goes LOW → FF1 is unaffected
- **CLK4 hits** → FF0 toggles → Q_0 goes LOW, $\overline{Q_0}$ goes HIGH → **FF1 triggers again** → Q_1 resets to 0 → counter **recycles** to 00

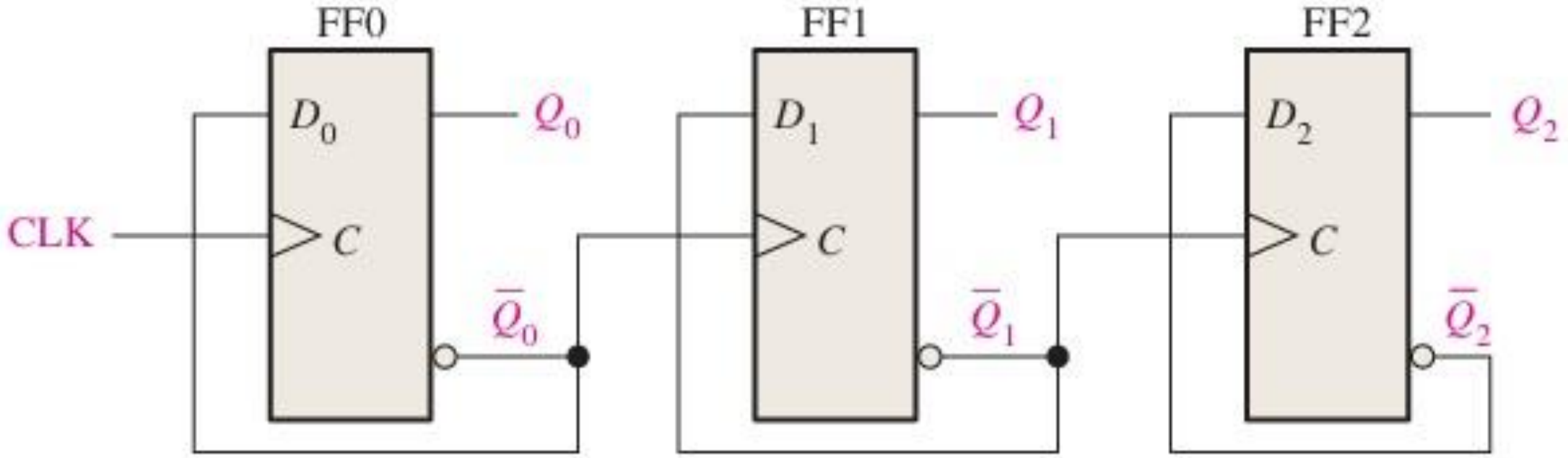
2-Bit Asynchronous Binary Counter



How It Counts

CLK Pulse	Q_1 (MSB)	Q_0 (LSB)	Binary
Initial	0	0	00
1	0	1	01
2	1	0	10
3	1	1	11
4 (recycle)	0	0	00

3-Bit Asynchronous Binary Counter



3-Bit Asynchronous Binary Counter

Step 1 — The building blocks

The counter uses 3 flip-flops (FF0, FF1, FF2), each storing one bit:

Q_0 (LSB), Q_1, and Q_2 (MSB).

Together they count from 000 to 111 in binary (0 to 7 in decimal).

Step 2 — How each flip-flop triggers

FF0 is directly clocked by the external CLK signal.

But FF1 is clocked by $\overline{Q_0}$ (the inverted output of FF0), and FF2 is clocked by $\overline{Q_1}$.

This "chain reaction" is what makes it *asynchronous* — the flip-flops don't all switch at the same time.

3-Bit Asynchronous Binary Counter

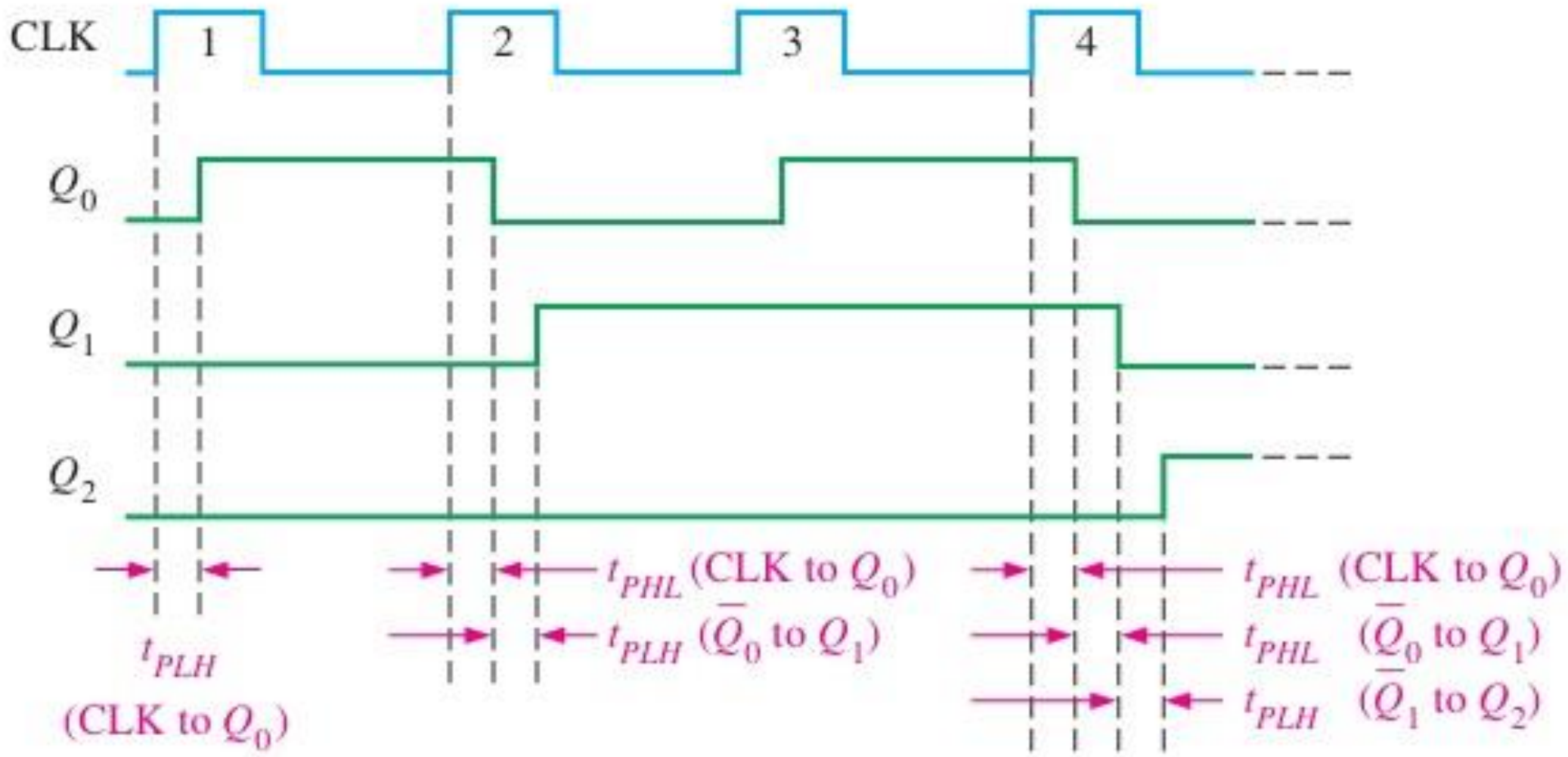
Step 3 — Ripple effect
Each clock pulse triggers FF0 first → its output change triggers FF1 → which then triggers FF2.
The signal "ripples" through the chain with a small delay at each stage, hence the nickname *ripple counter*.

Step 4 — The count sequence
Starting at 000, each clock pulse increments the binary count by 1, all the way to 111 (7), then it recycles back to 000 on the 8th pulse.

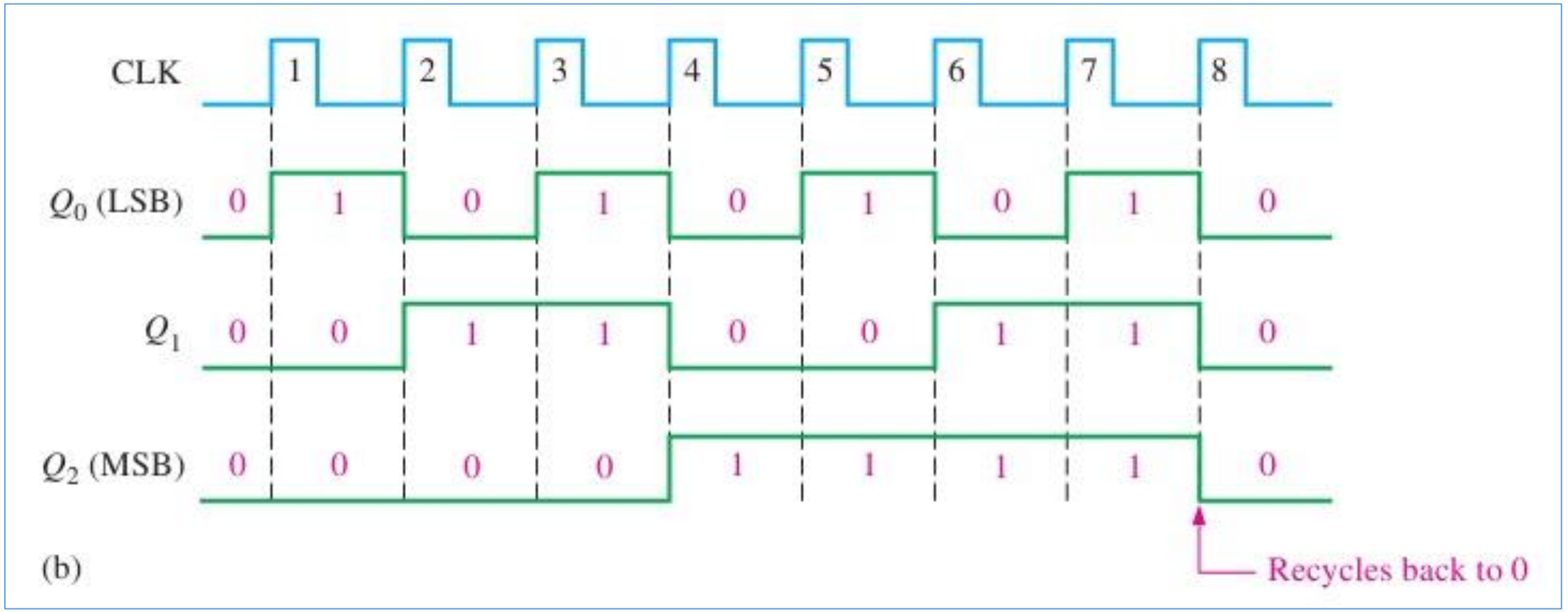
Step 5 — Propagation delay (the key limitation)
Because the flip-flops trigger one after another, Q_0 changes first, Q_1 changes slightly later, and Q_2 changes last.
By clock pulse 4, it takes *three* full propagation delays for the count to fully settle — this limits how fast the counter can be clocked.

3-Bit Asynchronous Binary Counter

Propagation Delay adds up from LSB to MSB



3-Bit Asynchronous Binary Counter



3-Bit Asynchronous Binary Counter

TABLE 9–2

State sequence for a 3-bit binary counter.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

Types of Counters

1. Asynchronous (Ripple) Counter

- Flip-flops do **not** receive the clock at the same time.
- Output of one flip-flop drives the next.
- Simpler but slower.

2. Synchronous Counter

- All flip-flops receive the same clock pulse together.
- Faster and more reliable.
- JK flip-flops are used to illustrate synchronous counters
- D flip-flops can also be used but require more logic.

2-Bit Synchronous Binary Counter

- A **2-bit synchronous binary counter** counts from **0 to 3** (binary: 00 → 01 → 10 → 11 → 00...) and repeats.

- "Synchronous" means **all flip-flops share the same clock** — they all trigger at the same time.

The Building Blocks

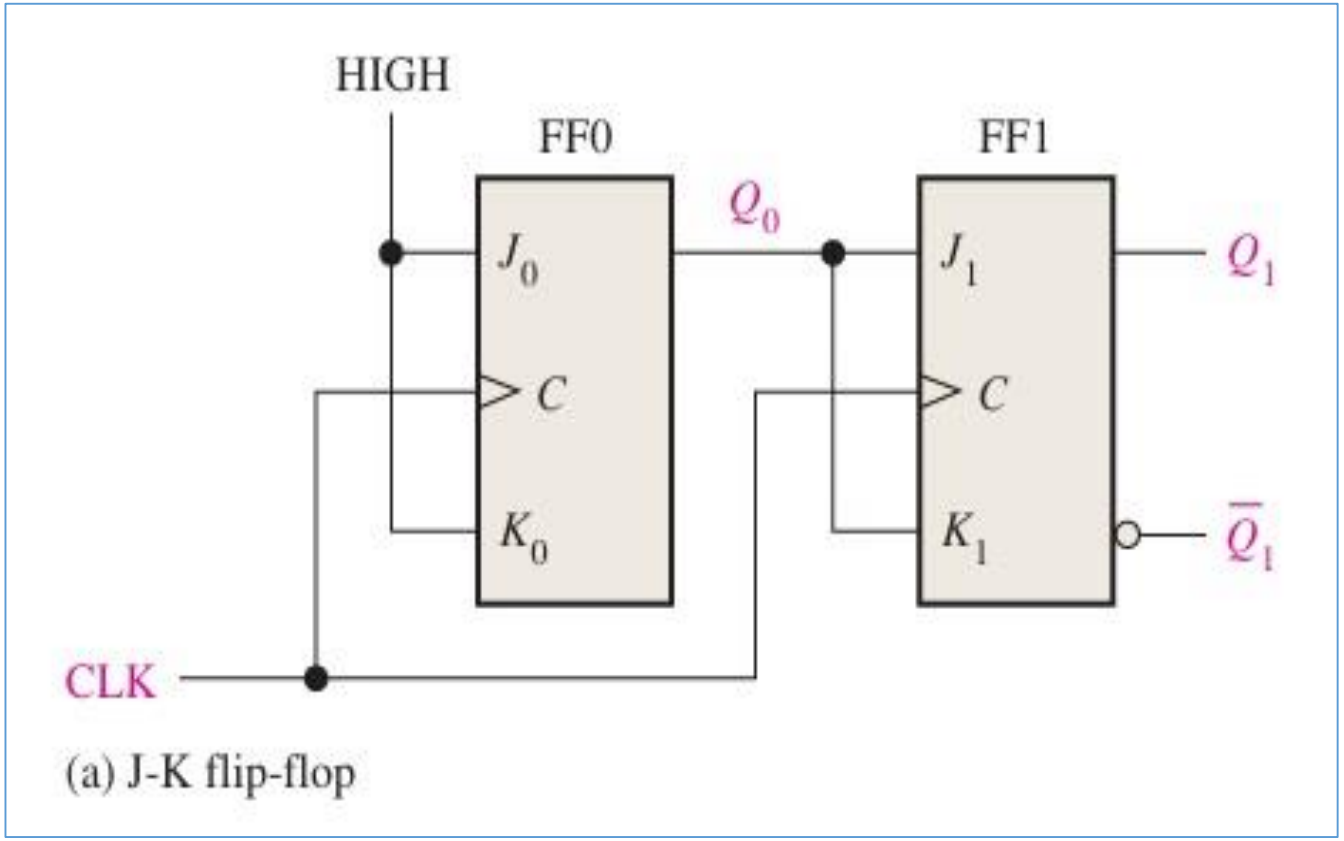
It uses **2 flip-flops**:

Flip-Flop	Output	Represents
FF0	Q_0	LSB (Least Significant Bit)
FF1	Q_1	MSB (Most Significant Bit)

How the Circuit is Wired

JK Flip-Flop Version:

- **FF0:** J_0 and K_0 are both tied to **HIGH** → always toggles on every clock
- **FF1:** J_1 and K_1 are connected to $\mathbf{Q_0}$ → only toggles when $Q_0 = 1$

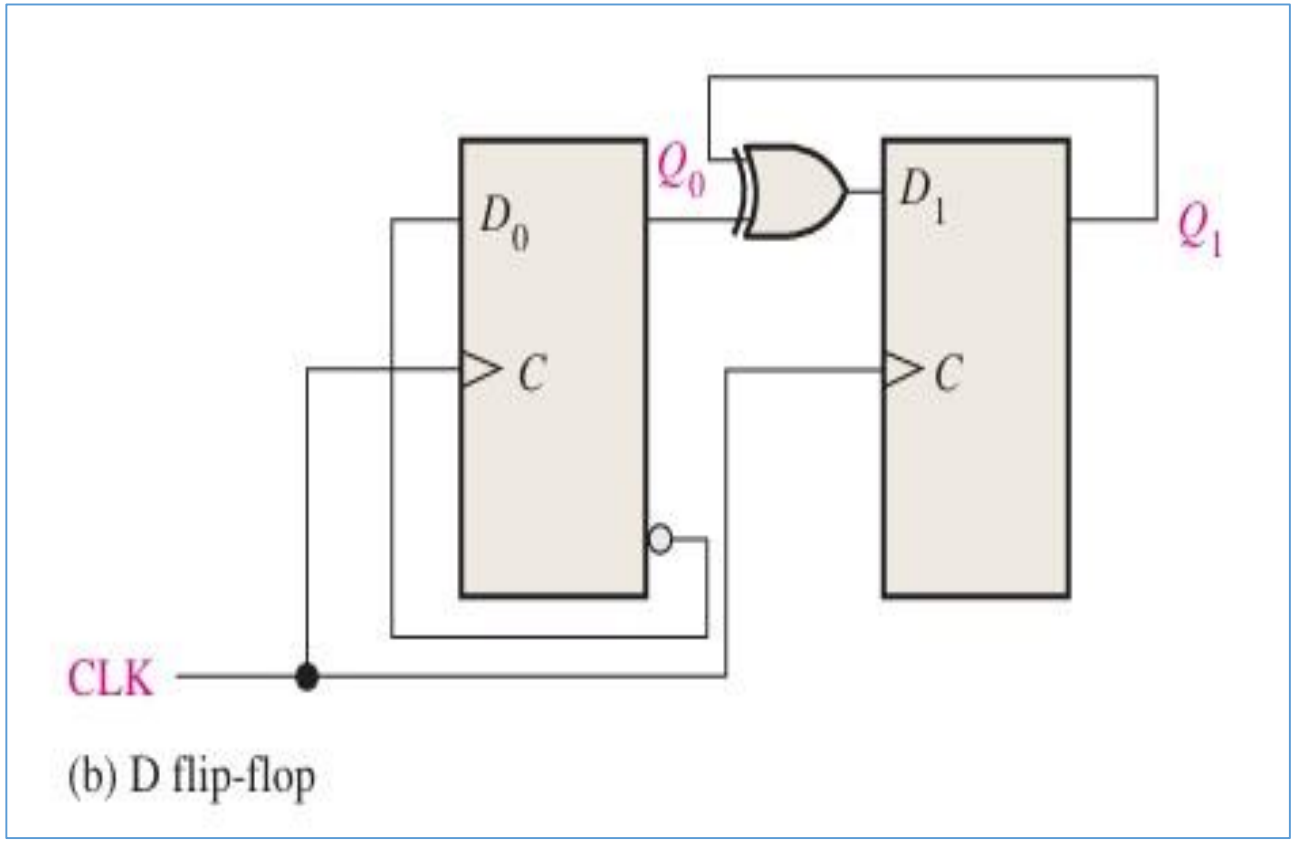


(a) J-K flip-flop

How the Circuit is Wired

D Flip-Flop Version:

- Uses an **AND gate + inverter** to achieve the same logic
- $D_0 = \overline{Q_0}$ (complement), $D_1 = Q_0$ XOR Q_1



(b) D flip-flop

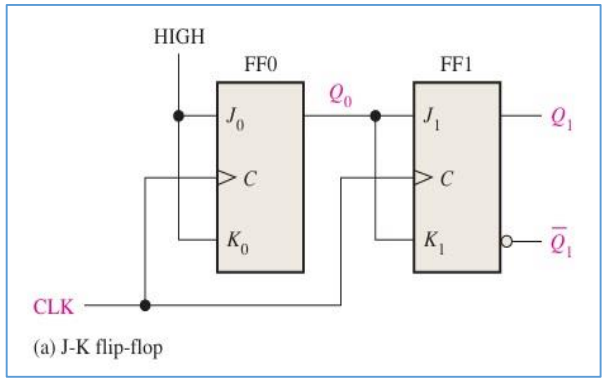
Step-by-Step Operation (Clock by Clock)

Using JK flip-flop

Initial State

$Q_1 = 0, Q_0 = 0 \rightarrow$ Binary 00
Both flip-flops are RESET

⚡ CLK1 — Rising Edge



FF0: J=1, K=1 → **Toggles** → Q_0 goes **0 → 1**
FF1: J=0, K=0 (because Q_0 was still 0 at the moment of clock edge, due to propagation delay) → **No Change** → Q_1 stays **0**

Result: $Q_1 = 0, Q_0 = 1 \rightarrow$ Binary 01

Important: Q_0 hasn't physically changed yet when FF1 reads it — this is the **propagation delay**

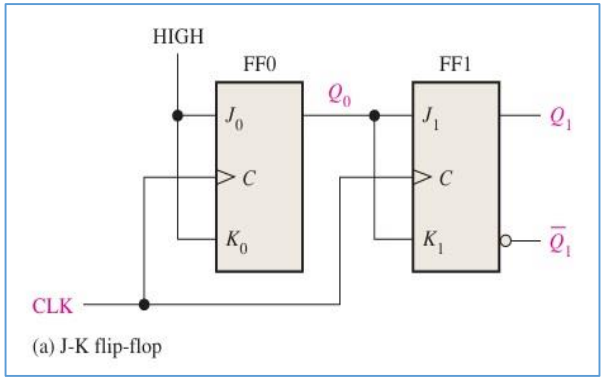
2-Bit Synchronous Binary Counter

⚡ CLK2 — Rising Edge

FF0: J=1, K=1 → Toggles → Q_0 goes 1 → 0

FF1: J=1, K=1 (because Q_0 WAS 1 at clock edge) → Toggles → Q_1 goes 0 → 1

Result: $Q_1 = 1$, $Q_0 = 0$ → Binary 10 = Decimal 2



(a) J-K flip-flop

⚡ CLK3 — Rising Edge

FF0: J=1, K=1 → Toggles → Q_0 goes 0 → 1

FF1: J=0, K=0 (Q_0 was 0 at clock edge) → No Change → Q_1 stays 1

Result: $Q_1 = 1$, $Q_0 = 1$ → Binary 11 = Decimal 3

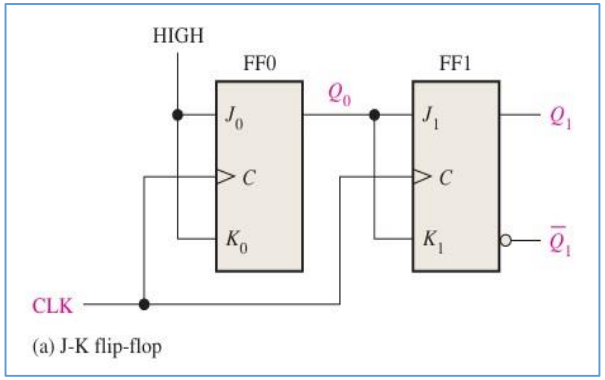
2-Bit Synchronous Binary Counter

⚡ **CLK4 — Rising Edge**

FF0: J=1, K=1 → Toggles → Q_0 goes 1 → 0

FF1: J=1, K=1 (Q_0 was 1) → Toggles → Q_1 goes 1 → 0

Result: $Q_1 = 0$, $Q_0 = 0$ → Binary 00 = Decimal 0

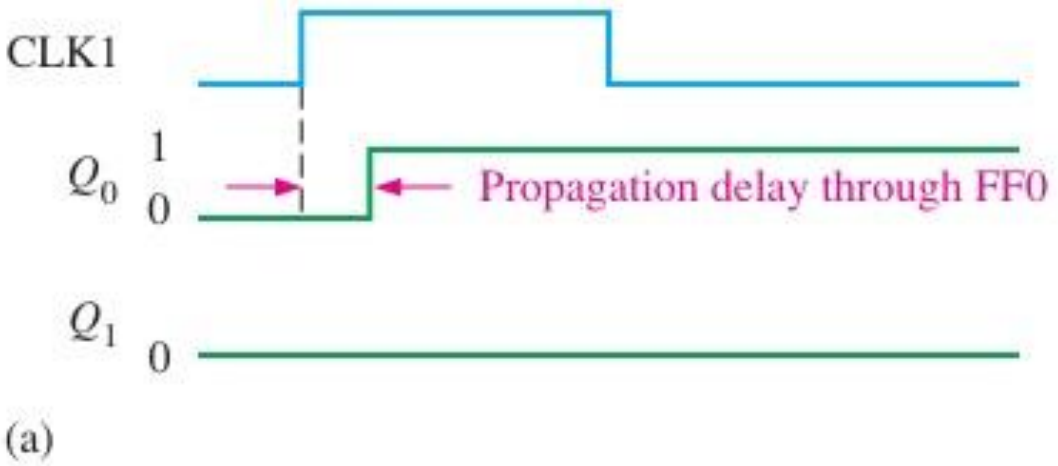


🔄 Counter **resets back to 0** and repeats!

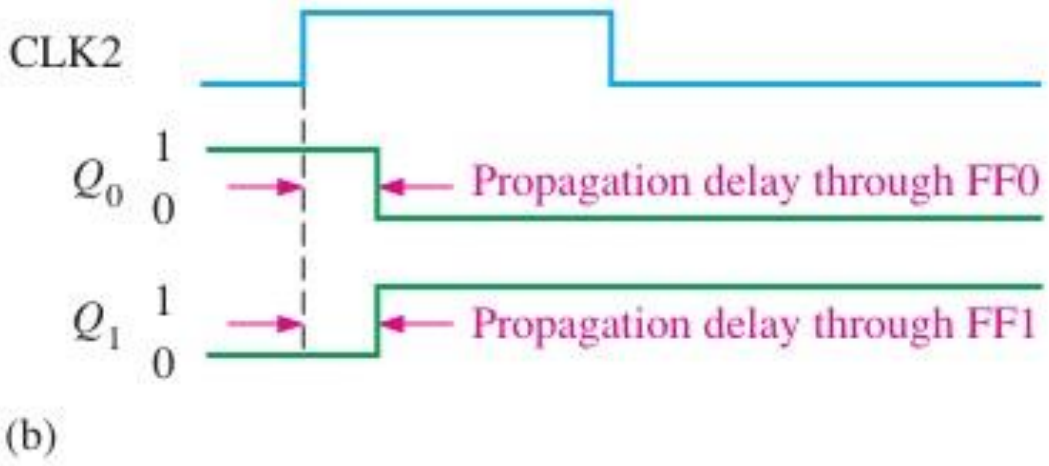
Complete Count Sequence Summary

CLK	Q_1	Q_0	Decimal	
0	0	0	0	← Start
1	0	1	1	
2	1	0	2	
3	1	1	3	
4	0	0	0	← Repeats!

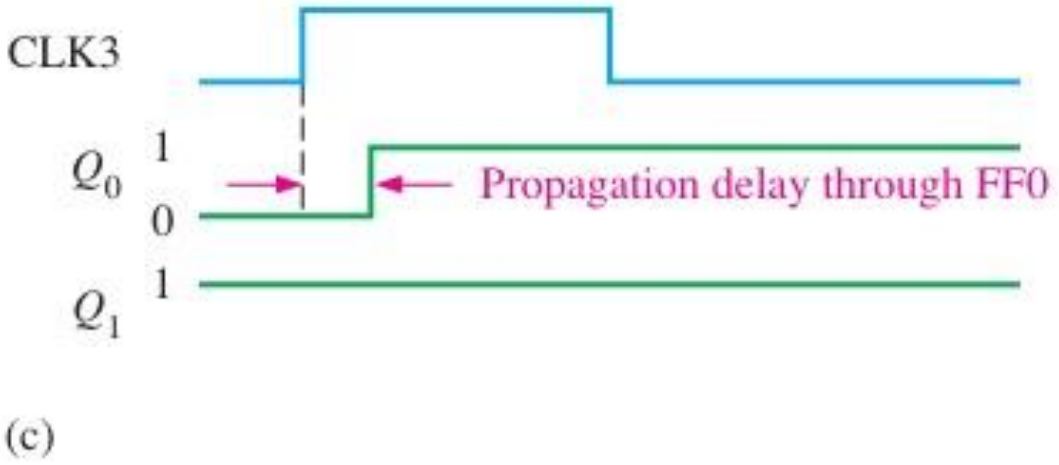
2-Bit Synchronous Binary Counter



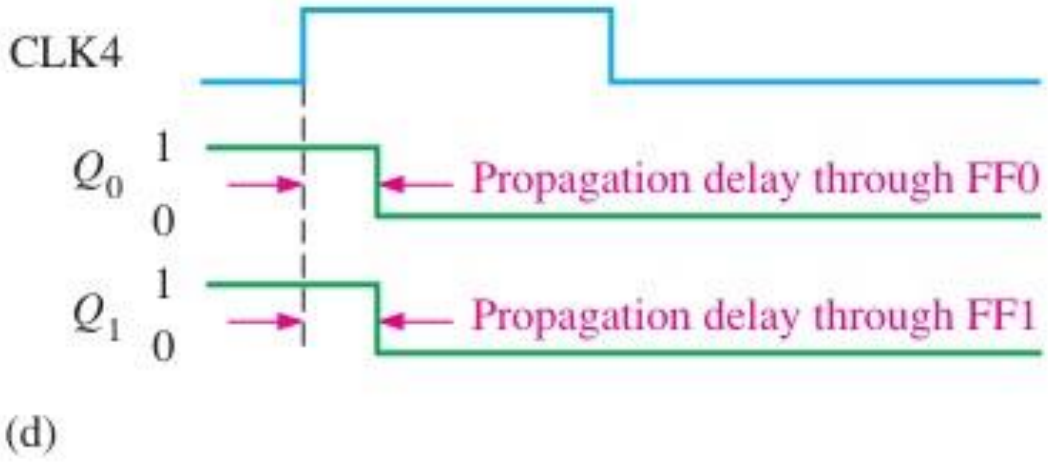
(a)



(b)

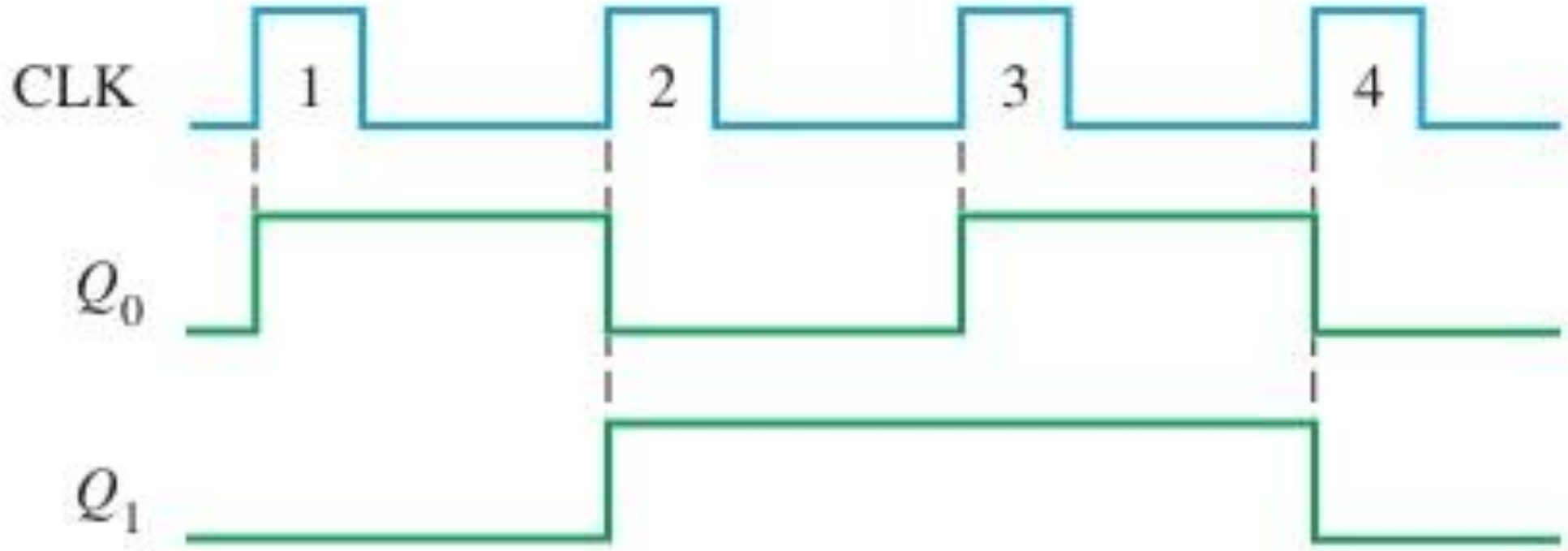


(c)



(d)

2-Bit Synchronous Binary Counter



3-Bit Synchronous Binary Counter

Overview

Counts from **0 to 7** (000 → 111) then resets.
Uses **3 flip-flops** (FF0, FF1, FF2).

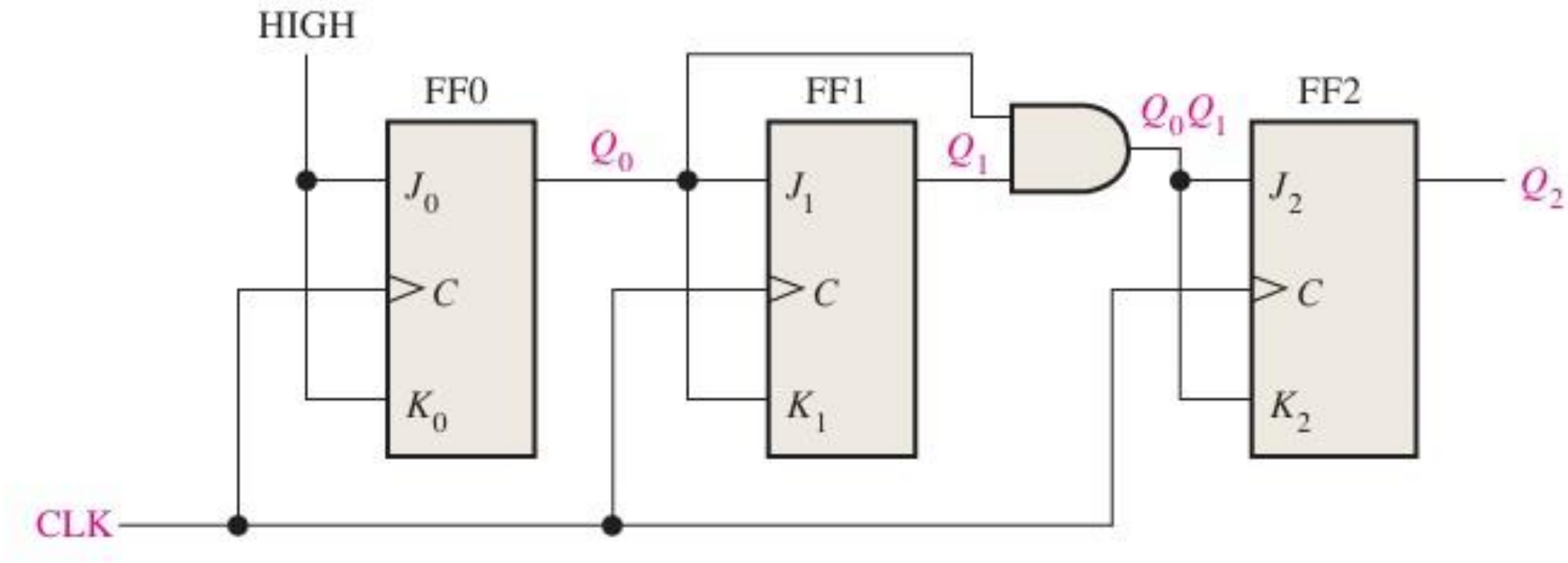
Flip-Flop	Output	Role
FF0	Q_0	LSB
FF1	Q_1	Middle bit
FF2	Q_2	MSB

3-Bit Synchronous Binary Counter

Wiring Rules (How inputs are connected)

Flip-Flop	J & K inputs	Toggle Condition
FF0	Always HIGH	**Every** clock
FF1	Connected to Q_0	When $Q_0 = 1$
FF2	Connected to AND gate ($Q_0 \cdot Q_1$)	When $Q_0=1$ **AND** $Q_1=1$

3-Bit Synchronous Binary Counter



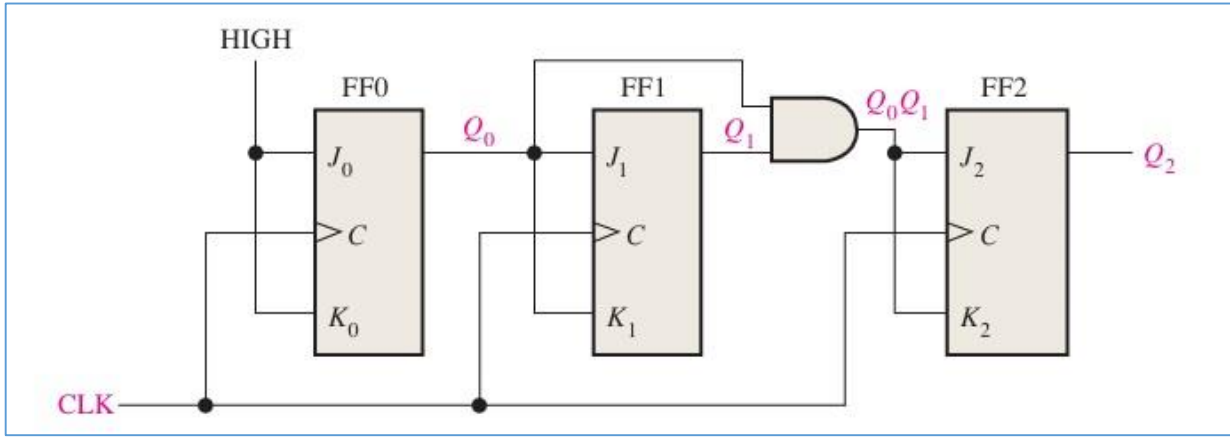
Clock-by-Clock Operation

Start: Q_2=0, Q_1=0, Q_0=0 → Decimal 0

⚡ CLK1

- FF0 toggles → Q_0: 0→**1**
- FF1: Q_0 was 0 → No change → Q_1=0
- FF2: $Q_0 \cdot Q_1$ was 0 → No change → Q_2=0

Result: 0 0 1 = Decimal 1

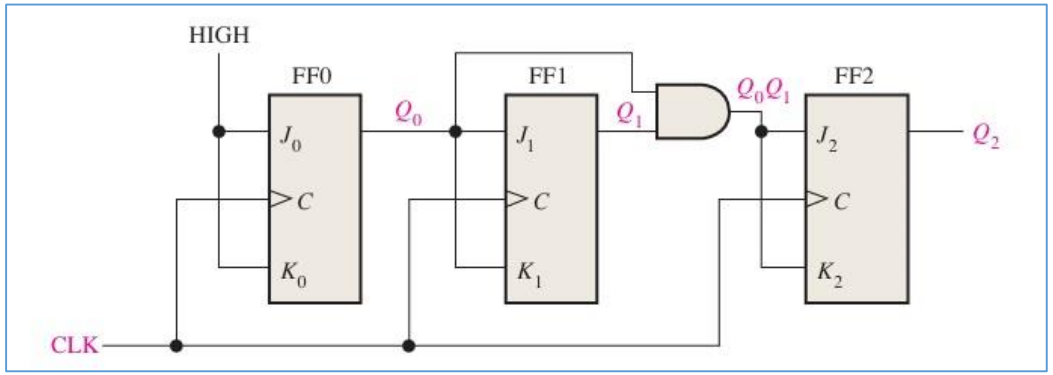


⚡ CLK2

- FF0 toggles → Q_0: 1→**0**
- FF1: Q_0 was 1 → Toggles → Q_1: 0→**1**
- FF2: $Q_0 \cdot Q_1$ = 1·0 = 0 → No change → Q_2=0

Result: 0 1 0 = Decimal 2

3-Bit Synchronous Binary Counter



⚡ CLK3

- FF0 toggles → Q_0: 0→**1**
- FF1: Q_0 was 0 → No change → Q_1=1
- FF2: $Q_0 \cdot Q_1 = 0 \cdot 1 = 0$ → No change → Q_2=0

Result: 0 1 1 = Decimal 3

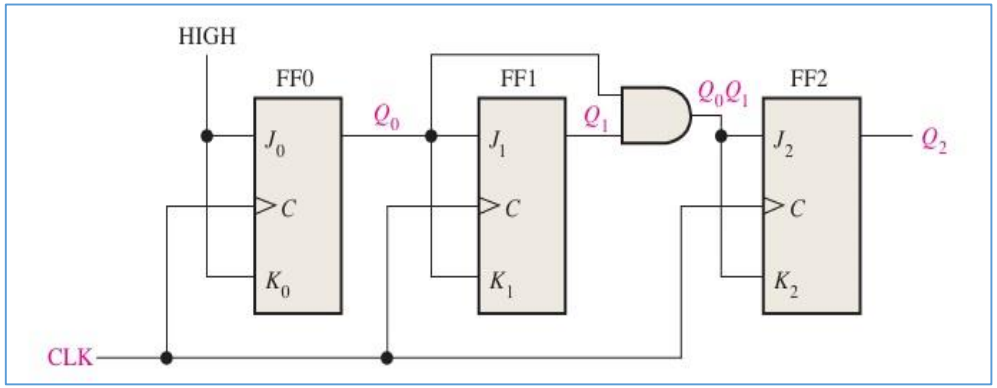
⚡ CLK4

- FF0 toggles → Q_0: 1→**0**
- FF1: Q_0 was 1 → Toggles → Q_1: 1→**0**
- FF2: $Q_0 \cdot Q_1 = 1 \cdot 1 =$ **1** → Toggles → Q_2: 0→**1**

Result: 1 0 0 = Decimal 4

AND gate fires! All three flip-flops change simultaneously

3-Bit Synchronous Binary Counter



⚡ CLK5

- FF0 toggles → Q_0: 0→**1**
- FF1: Q_0 was 0 → No change → Q_1=0
- FF2: $Q_0 \cdot Q_1 = 0 \cdot 0 = 0$ → No change → Q_2=1

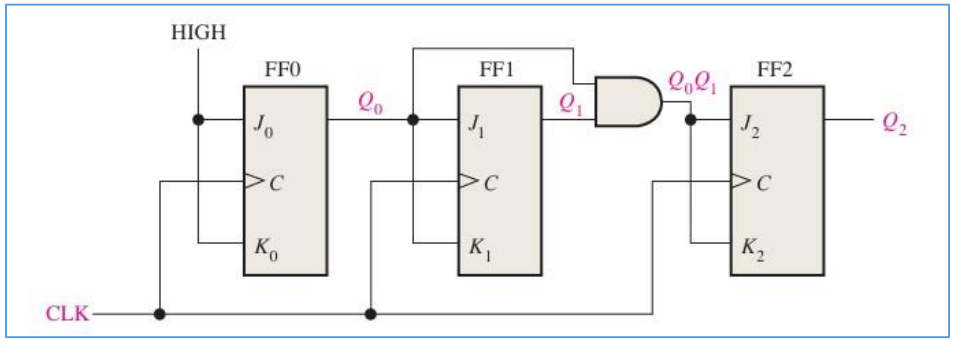
Result: 1 0 1 = Decimal 5

⚡ CLK6

- FF0 toggles → Q_0: 1→**0**
- FF1: Q_0 was 1 → Toggles → Q_1: 0→**1**
- FF2: $Q_0 \cdot Q_1 = 1 \cdot 0 = 0$ → No change → Q_2=1

Result: 1 1 0 = Decimal 6

3-Bit Synchronous Binary Counter



⚡ CLK7

- FF0 toggles → Q_0: 0→**1**
- FF1: Q_0 was 0 → No change → Q_1=1
- FF2: $Q_0 \cdot Q_1 = 0 \cdot 1 = 0$ → No change → Q_2=1

Result: 1 1 1 = Decimal 7

⚡ CLK8 — RECYCLE!

- FF0 toggles → Q_0: 1→**0**
- FF1: Q_0 was 1 → Toggles → Q_1: 1→**0**
- FF2: $Q_0 \cdot Q_1 = 1 \cdot 1 = 1$ → Toggles → Q_2: 1→**0**

Result: 0 0 0 = Decimal 0 🔄 RESET!

3-Bit Synchronous Binary Counter

TABLE 9–3

State sequence for a 3-bit binary counter.

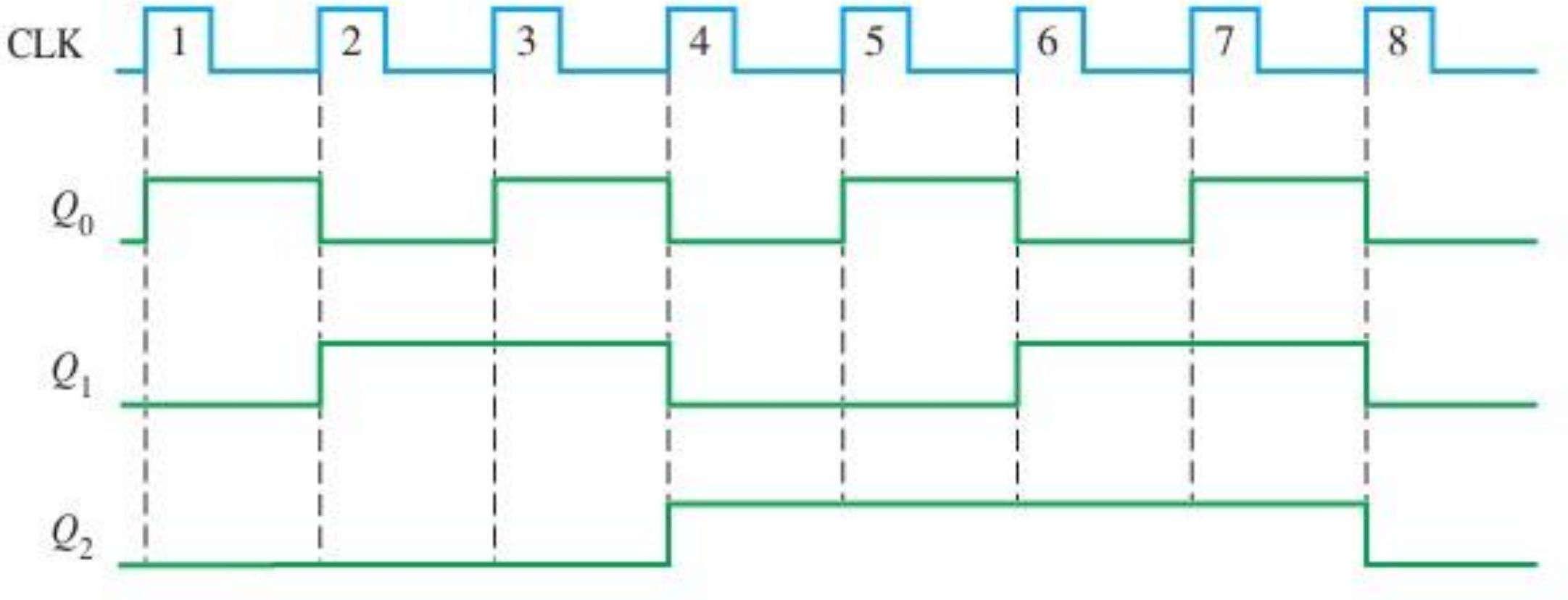
Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0

Key Takeaway

Q_0	Changes every	1 clock
Q_1	Changes every	2 clocks
Q_2	Changes every	4 clocks

The **AND gate** is the secret — it ensures FF2 only toggles when **both** Q_0 AND Q_1 are HIGH at the same time, giving correct binary counting behavior.

3-Bit Synchronous Binary Counter



3-Bit Synchronous Binary Counter

TABLE 9–4

Summary of the analysis of the counter in Figure 9–15.

Clock Pulse	Outputs			*J-K* Inputs						At the Next Clock Pulse		
	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	FF2	FF1	FF0
Initially	0	0	0	0	0	0	0	1	1	NC*	NC	Toggle
1	0	0	1	0	0	1	1	1	1	NC	Toggle	Toggle
2	0	1	0	0	0	0	0	1	1	NC	NC	Toggle
3	0	1	1	1	1	1	1	1	1	Toggle	Toggle	Toggle
4	1	0	0	0	0	0	0	1	1	NC	NC	Toggle
5	1	0	1	0	0	1	1	1	1	NC	Toggle	Toggle
6	1	1	0	0	0	0	0	1	1	NC	NC	Toggle
7	1	1	1	1	1	1	1	1	1	Toggle	Toggle	Toggle
										Counter recycles back to 000.		

*NC indicates *No Change*.

Thank You