Registers and Counters

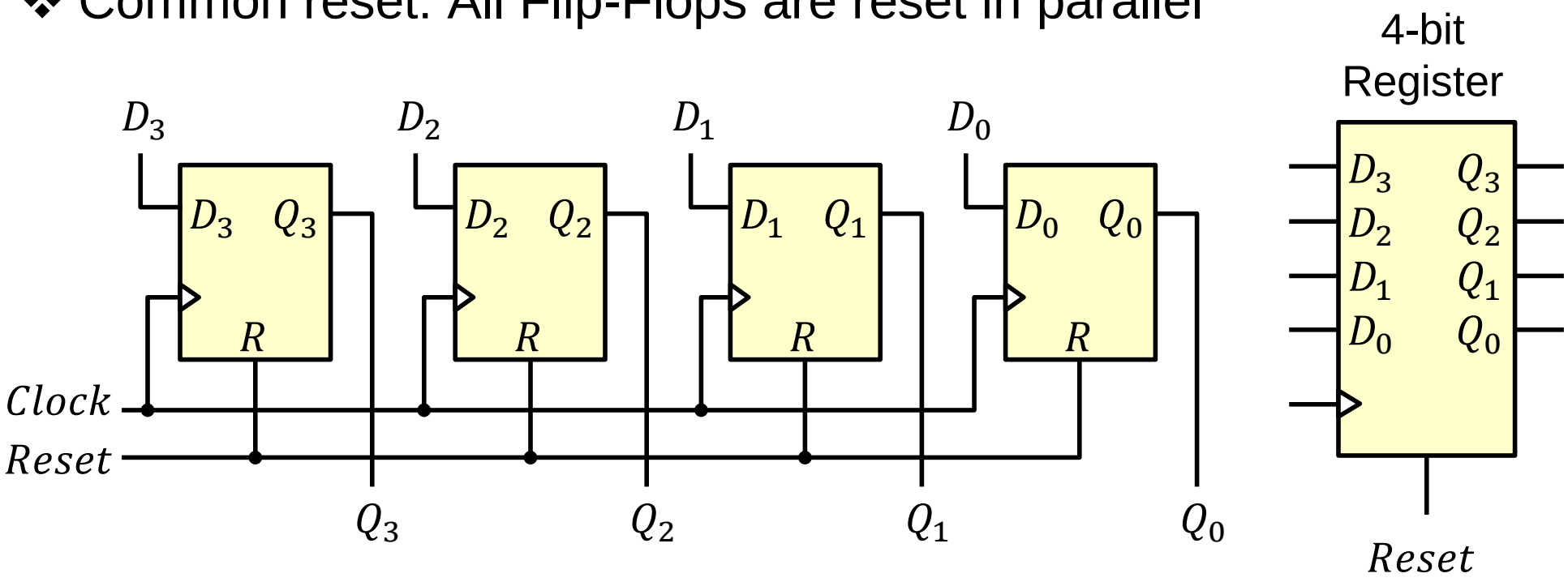
Aziz Qaroush

Presentation Outline

- Registers
- Shift Registers and their Applications
- Ripple Counters
- Synchronous Counters
- BCD Counters

Register

- A register is a circuit capable of storing data
- An n-bit register consists of n Flip-Flops and stores n bits
- Common clock: data is loaded in parallel at the same clock edge
- Common reset: All Flip-Flops are reset in parallel

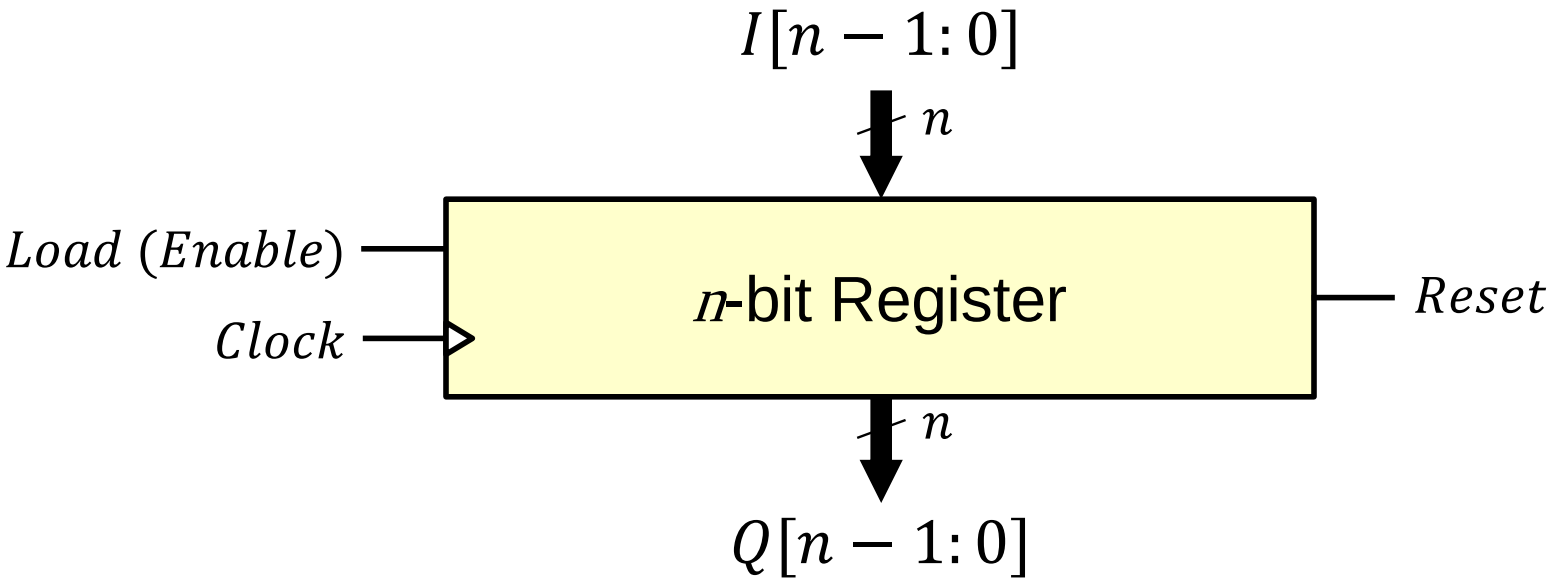


Register Load (or Enable)

- **Question:** How to control the loading of data into a register?
- **Solution:** Introduce a register Load (or Enable) signal

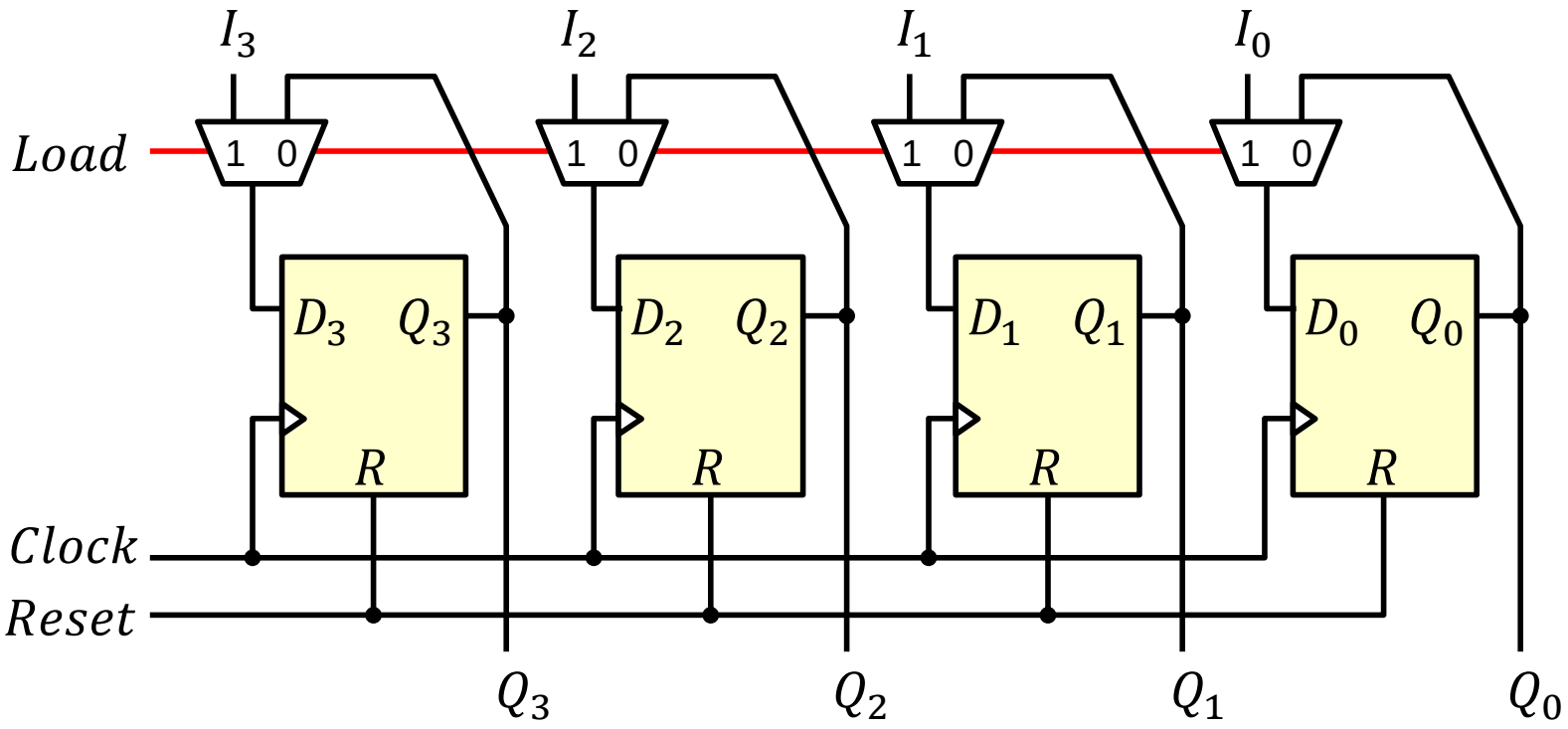
  If the register is enabled, load the data into the register

  Otherwise, do not change the value of the register
- **Question:** How to implement register Load?



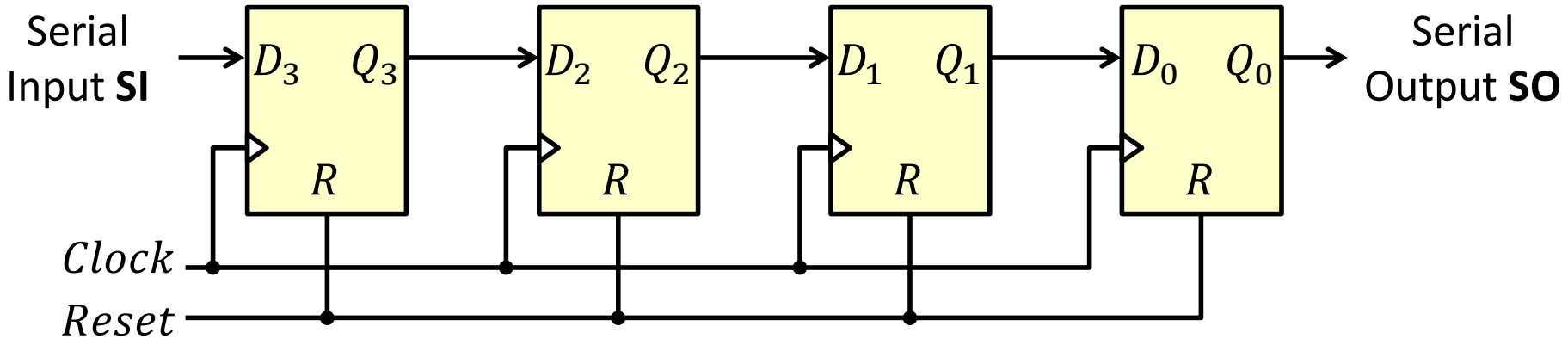
Register with Parallel Load

- **Solution:** Add a Mux at the D input of the register
- $D_i = Load \cdot I_i + \overline{Load} \cdot Q_i$
- If $Load$ is **1** then $D_i = I_i$ If $Load$ is **0** then $D_i = Q_i$



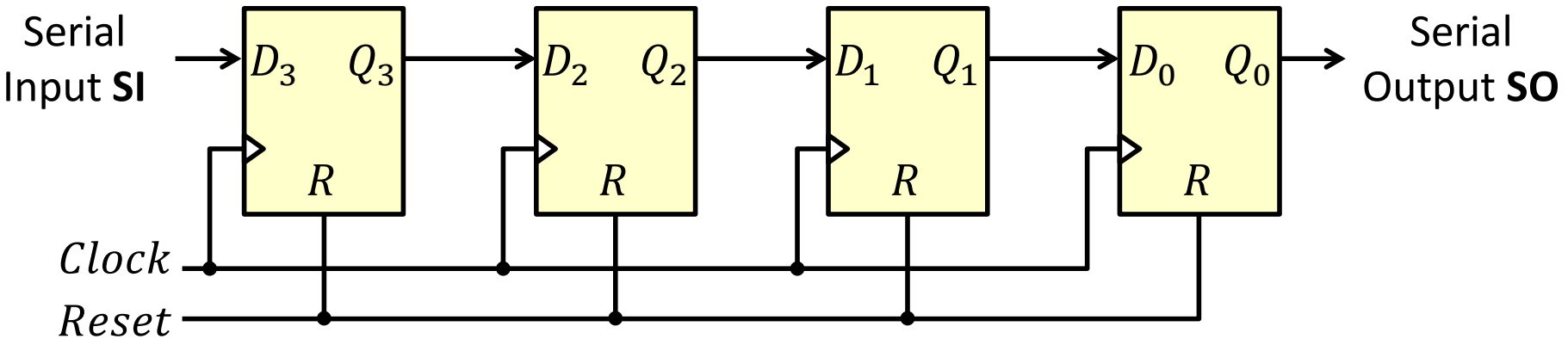
Shift Registers

- A shift register is a cascade of flip flops sharing the same clock
- Allows the data to be shifted from each flip-flop to its neighbor
- The output of a flip-flop is connected to the input of its neighbor
- Shifting can be done in either direction
- All bits are shifted simultaneously at the active edge of the clock



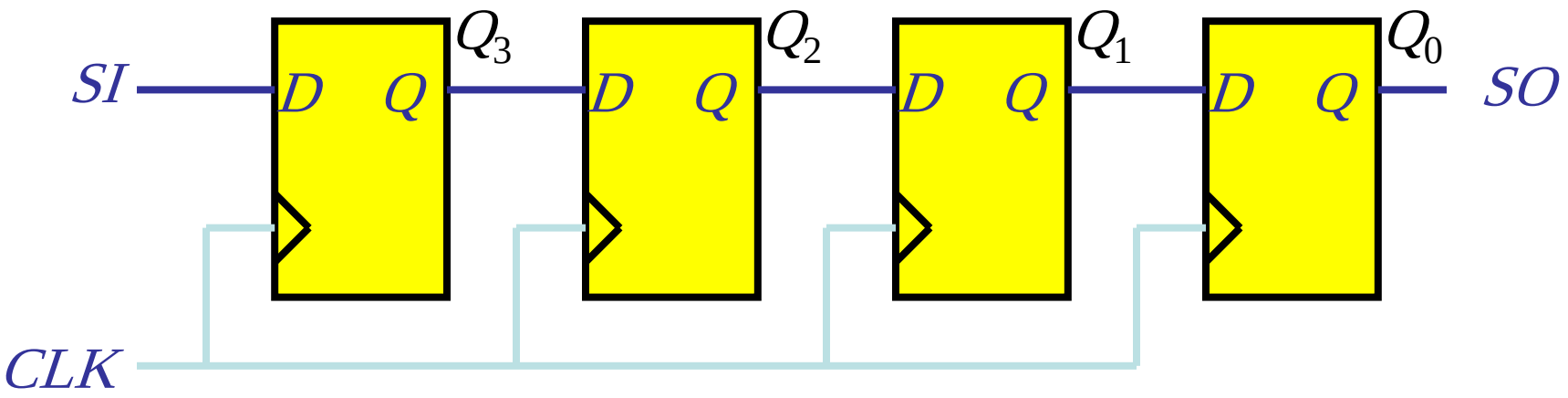
Right Shift Register

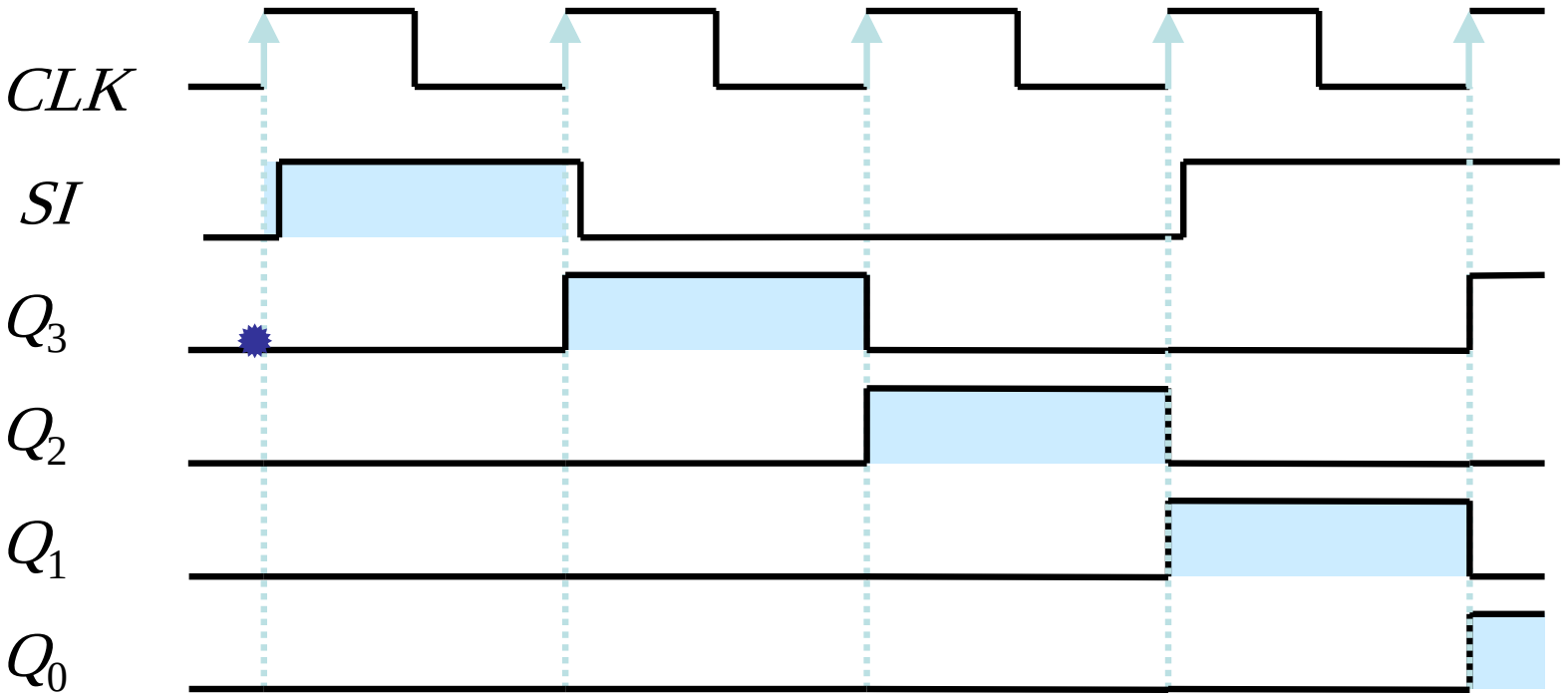
Timing of a Shift Register



Cycle	SI	Q3	Q2	Q1	Q0 = SO
T0	1	1	0	1	0
T1	0	1	1	0	1
T2	1	0	1	1	0
T3	1	1	0	1	1
T4	0	1	1	0	1
T5	1	0	1	1	0
T6	0	1	0	1	1

Timing of a Shift Register





Classification of Shift Register

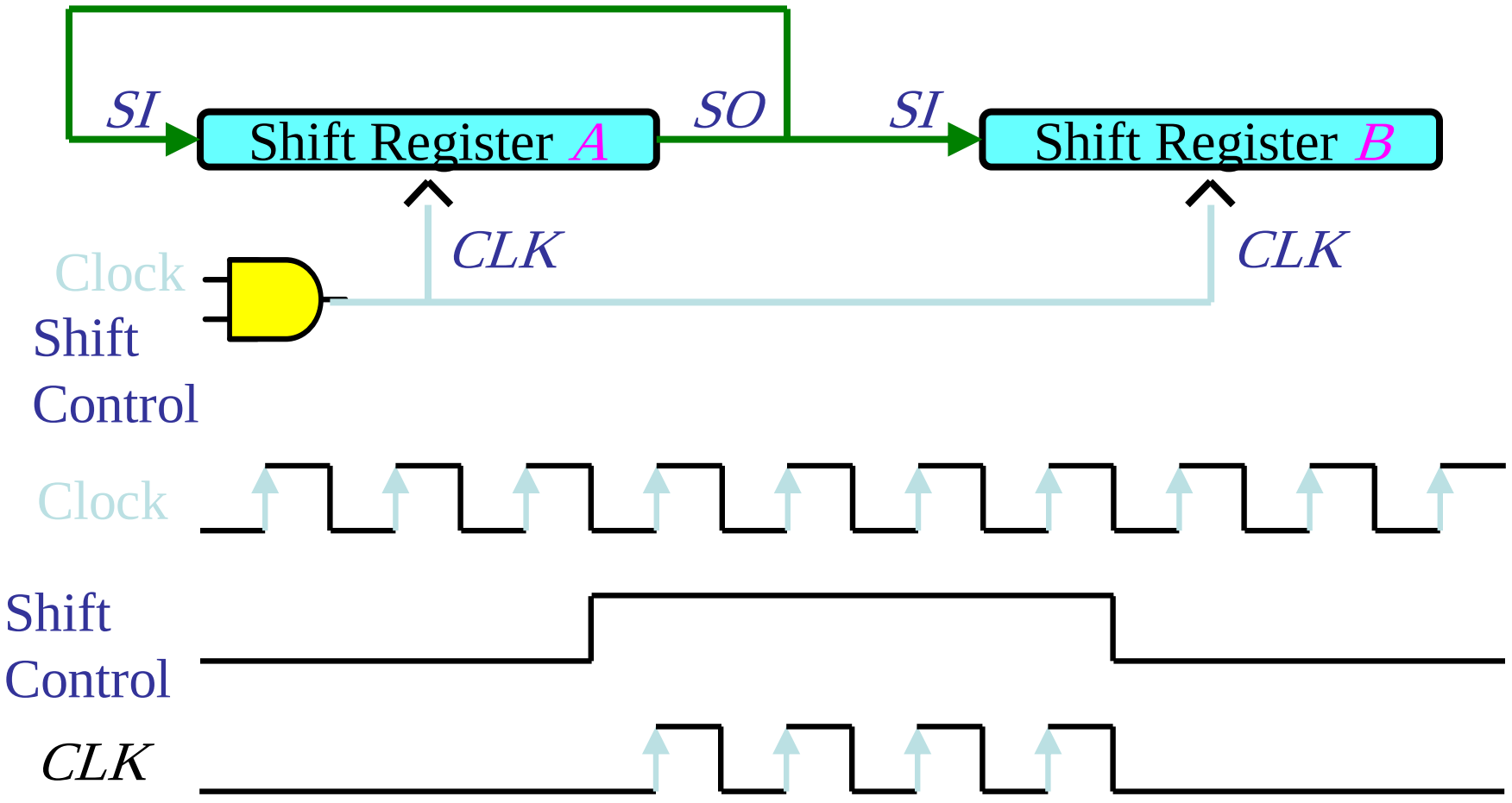
- Based on Direction
  - Unidirectional Shift Register: A register capable of shifting in one direction
  - Bidirectional Shift Register: A register can shift in both directions
  - Universal Shift Register: Bidirectional shift register + parallel load capability

- Based on Transferred Data
  - Parallel-in Parallel-out
  - Serial-in Serial-out
  - Serial-in Parallel-out
  - Parallel-in Serial-out

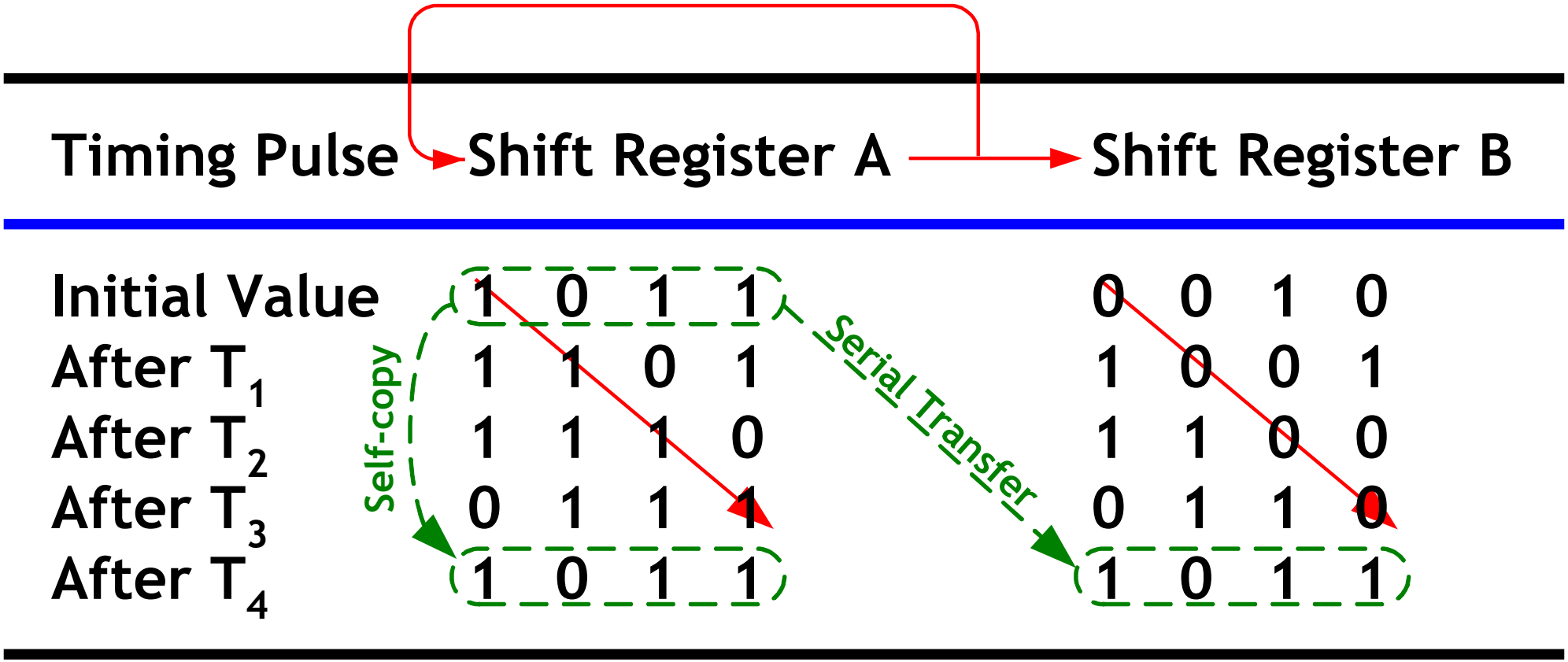
Serial Transfer vs. Parallel Transfer

1. **Serial Mode of Digital Systems: Manipulation one bit at a time.**
2. **Serial Transfer: Transferring one bit at a time by shifting the bits out of the source register into the destination register.**
3. **Parallel Transfer: All the bits of the register are transferred at the same time.**

Serial Transfer



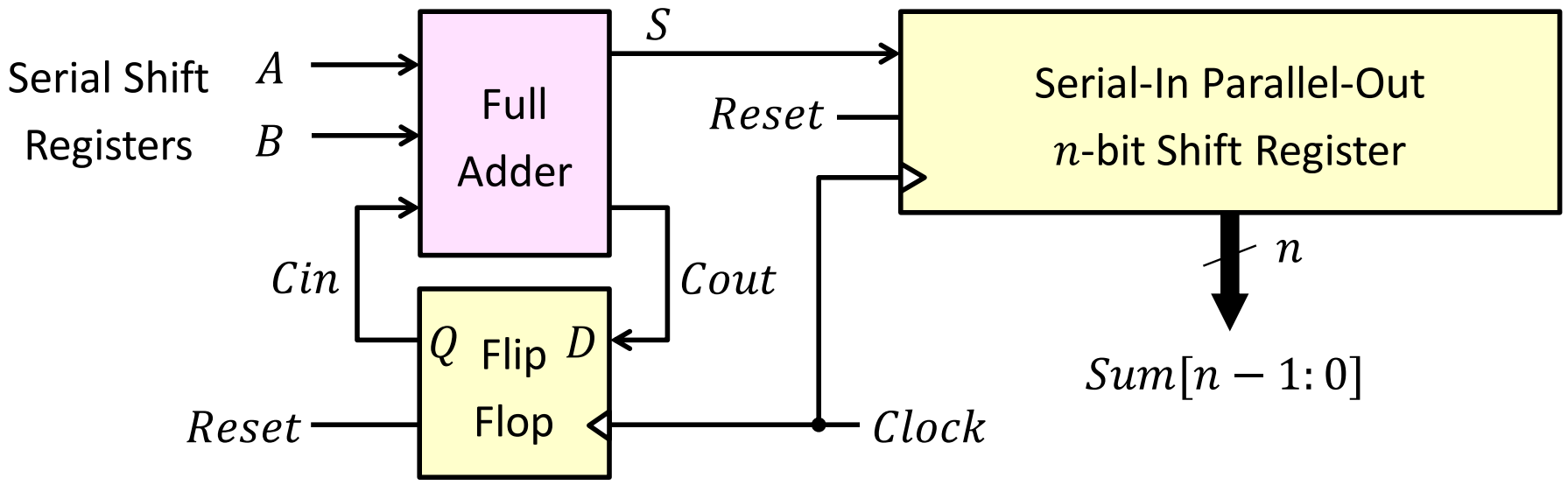
Timing of Serial Transfer Example



Bit Serial Adder

- Adding two n-bit numbers A and B serially over n clock cycles
- A bit-serial adder can be implemented using
  1. **A Full Adder**
  2. **A Flip-Flop to store the carry-out**
  3. **A Shift Register to store the n-bit sum**

Serial Addition Starts at the Least-significant bit



Serial Addition

- **Second Form of Serial Adder:** Implemented with JK FF

State Table for Serial Adder

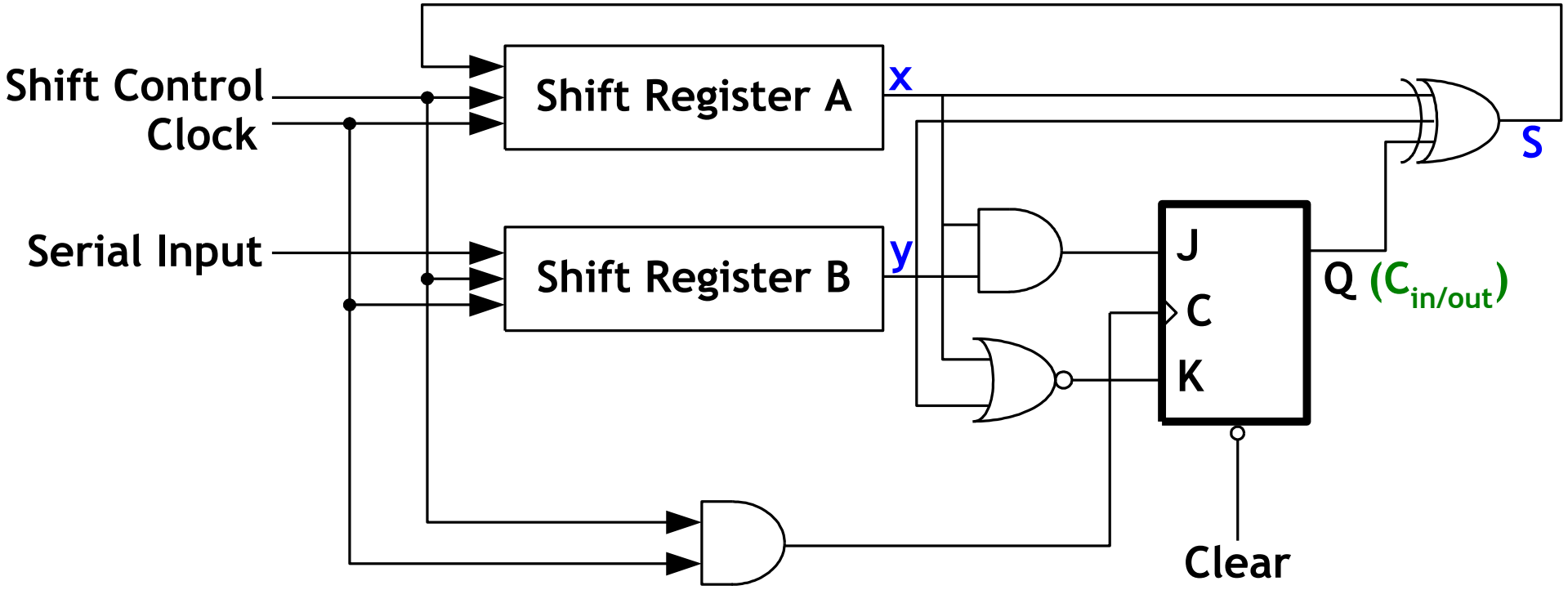
Present State $Q(C_{in})$	Inputs x y	Next State $Q(C_{out})$	Output S	Flip-Flop Input J_Q K_Q
0	0 0	0	0	0 X
0	0 1	0	1	0 X
0	1 0	0	1	0 X
0	1 1	1	0	1 X
1	0 0	0	1	X 1
1	0 1	1	0	X 0
1	1 0	1	0	X 0
1	1 1	1	1	X 0

Simplified Equation

$$\begin{cases} J_Q = xy \\ K_Q = x'y' = (x+y)' \\ S = x \oplus y \oplus Q \end{cases}$$

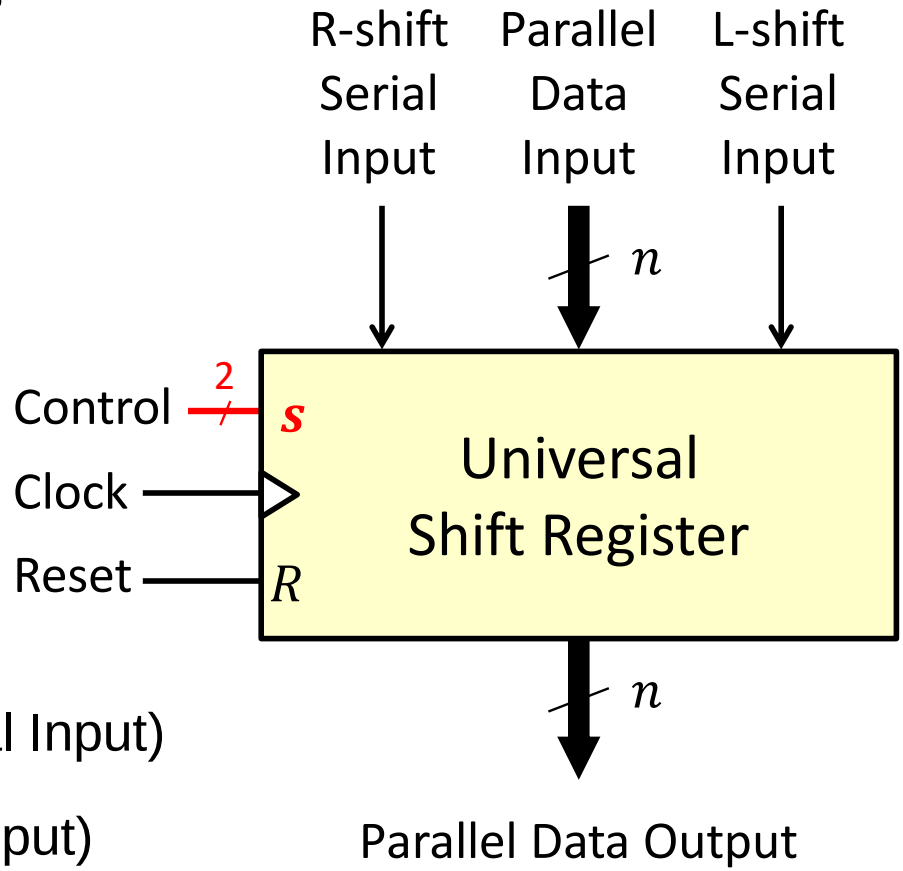
Serial Addition

- **Second Form of Serial Adder:** Circuit Diagram

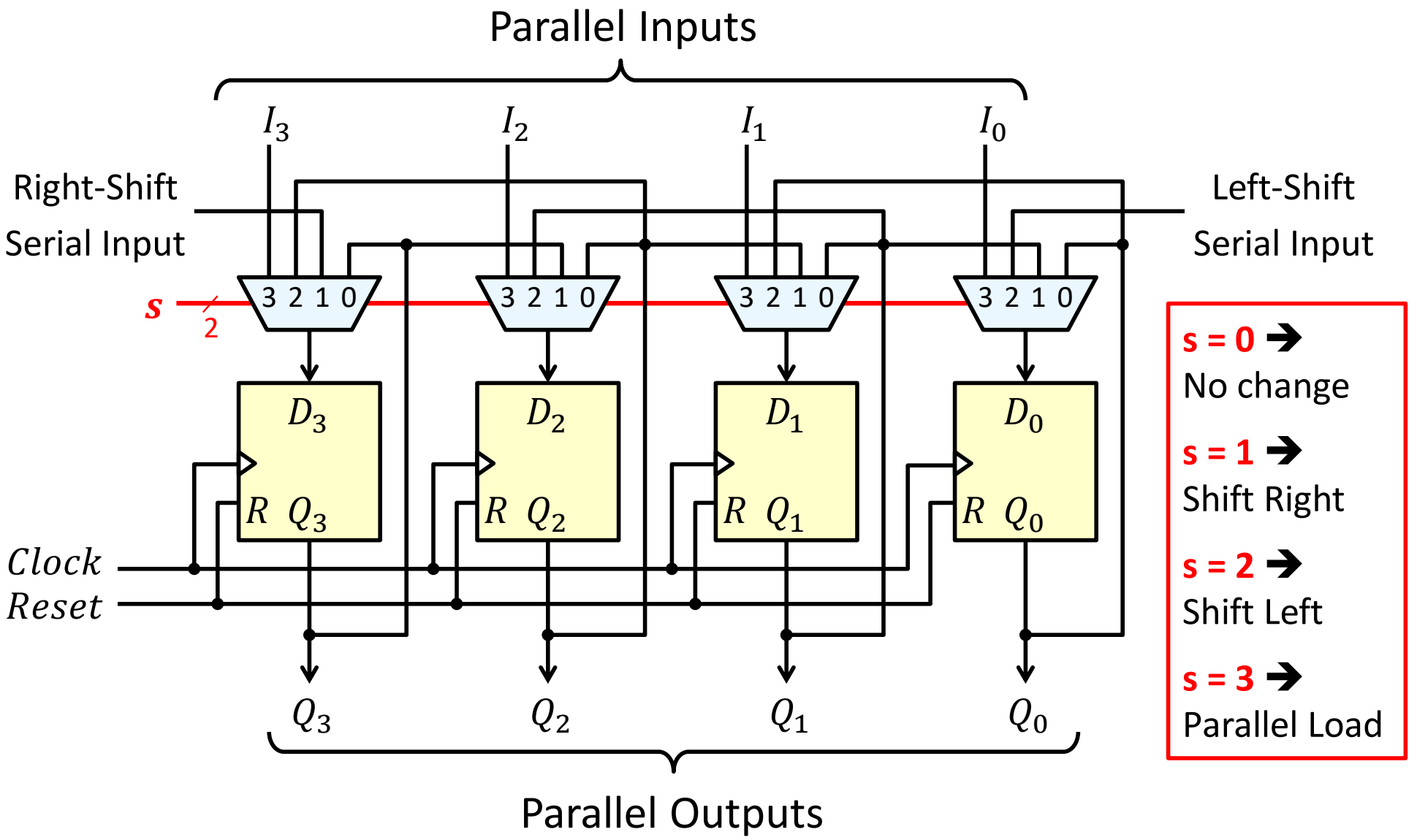


Universal Shift Register

- A Universal Shift Register has the following specification:
  - n parallel data input and n output lines
  - Right-shift and Left-shift Serial Inputs
  - Two control input lines ***s***
  - Clock input
  - Reset input
- Four control functions:
  - ***s* = 00** ➔ No change in value
  - ***s* = 01** ➔ Shift Right (Right-Shift Serial Input)
  - ***s* = 10** ➔ Shift Left (Left-Shift Serial Input)
  - ***s* = 11** ➔ Parallel Load n input bits

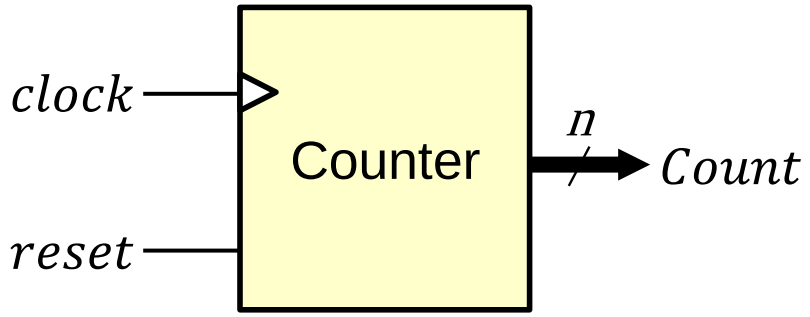


Universal Shift Register Design



Counter

- Sequential circuit that goes through a specific sequence of states
- Output of the counter is the **count value**
- Modulo-N counter: goes through 0, 1, 2, …, (N – 1)
- Modulo-8 binary counter: goes through 0, 1, 2, …, 7
- Modulo-10 (BCD) counter: goes through 0, 1, 2, …, 9
- Counting can be up or down
- Some Applications:
  - Timers
  - Event Counting
  - Frequency Division



Implementing Counters

Two Basic Approaches:

1. Ripple Counters

- ✧ The system clock is connected to the clock input of the first flip-flop (LSB)
- ✧ Each flip-flop output connects to the clock input of the next flip-flop
- ✧ Advantage: simple circuit and low power consumption
- ✧ Disadvantage: The counter is not truly synchronous
- ✧ No common clock to all flip-flops
- ✧ Ripple propagation delay as the clock signal propagates to the MSB

2. Synchronous Counters

- ✧ The system clock is connected to the clock input of ALL flip-flops
- ✧ Combinational logic is used to implement the desired state sequence

Ripple Counters

- A binary ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the *C* input of the next higher order flip-flop.

- The flip-flop holding the least significant bit receives the incoming count pulses.

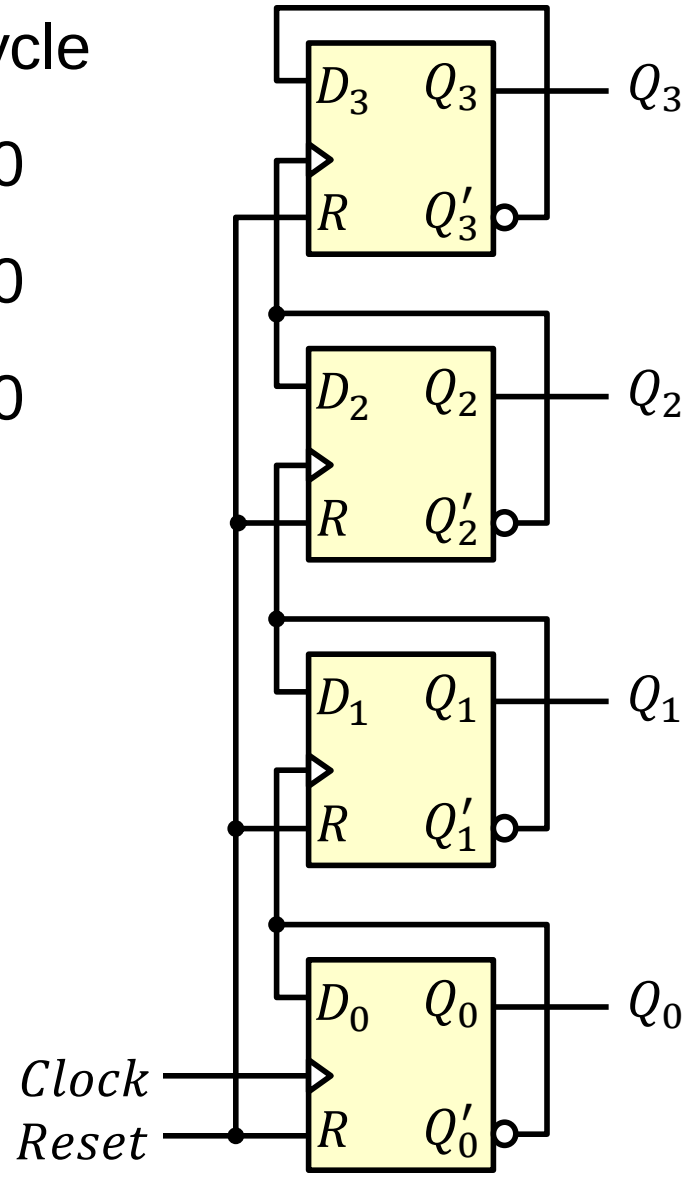
- A complementing flip-flop can be obtained from a *JK* flip-flop with the *J* and *K* inputs tied together or from a *T* flip-flop. A third possibility is to use a *D* flip-flop with the complement output connected to the *D* input

Ripple Counter

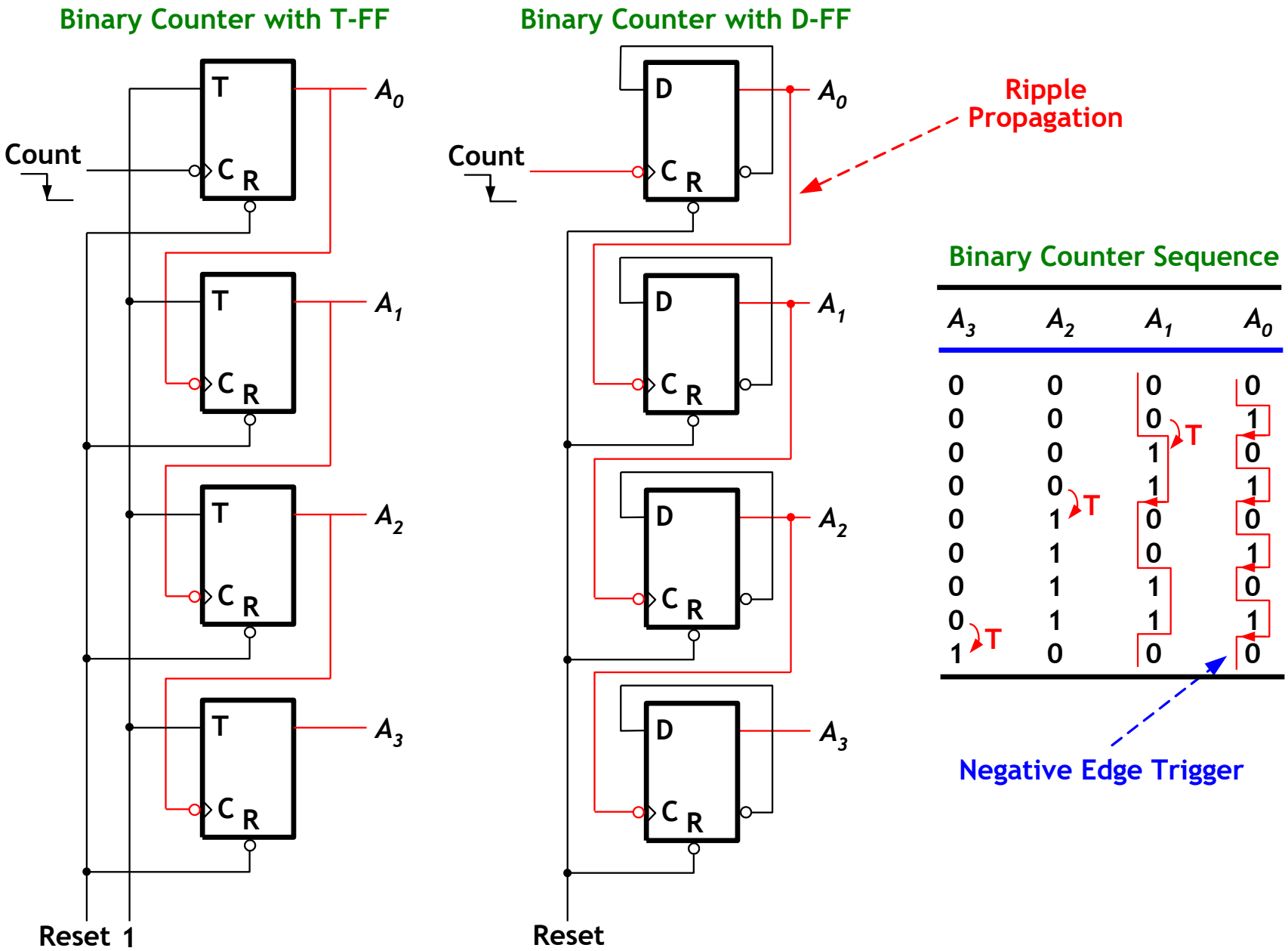
- Q_0 toggles at the positive edge of every cycle
- Q_1 toggles when Q_0 goes from 1 down to 0
- Q_2 toggles when Q_1 goes from 1 down to 0
- Q_3 toggles when Q_2 goes from 1 down to 0

Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0

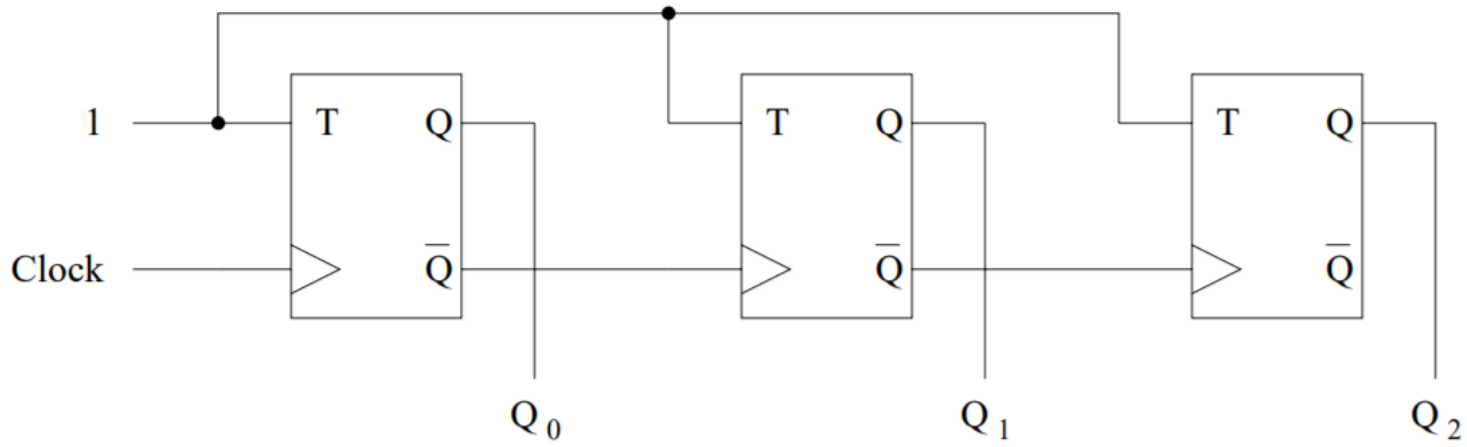
Counts Up
from 0 to 15
then back to 0

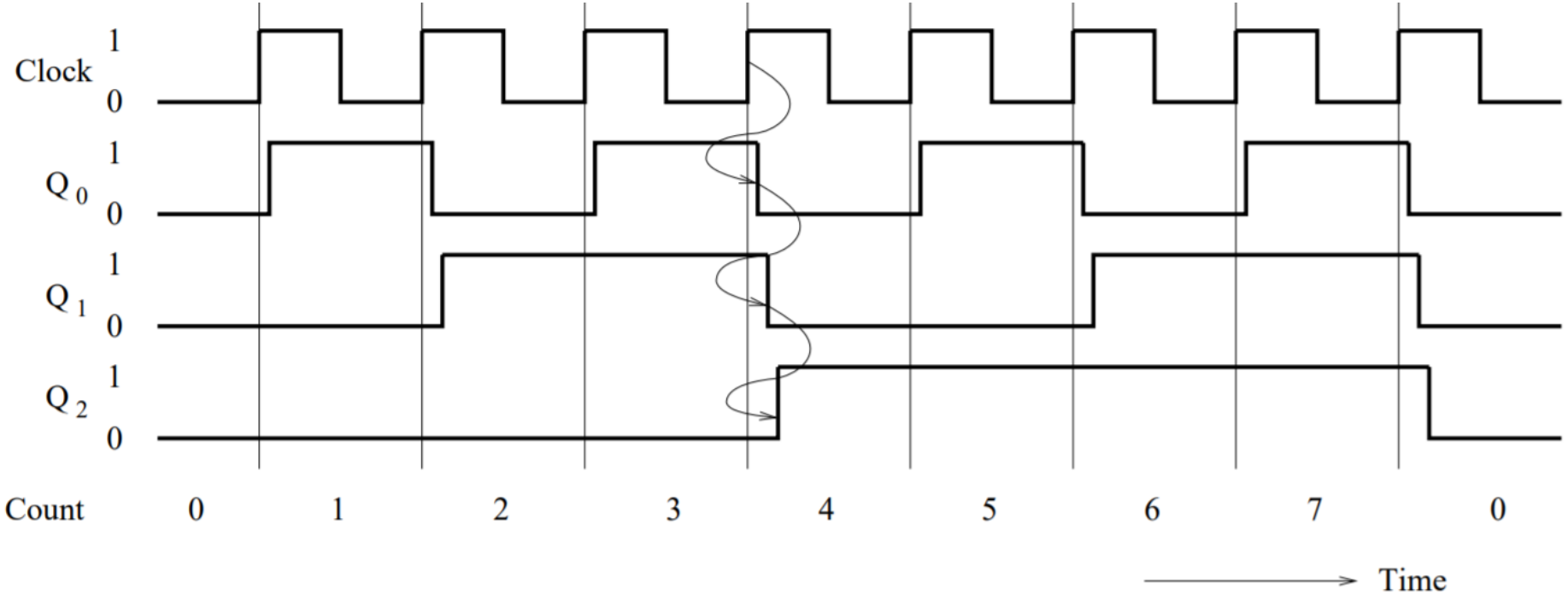


4-Bit Binary Ripple Up Counter

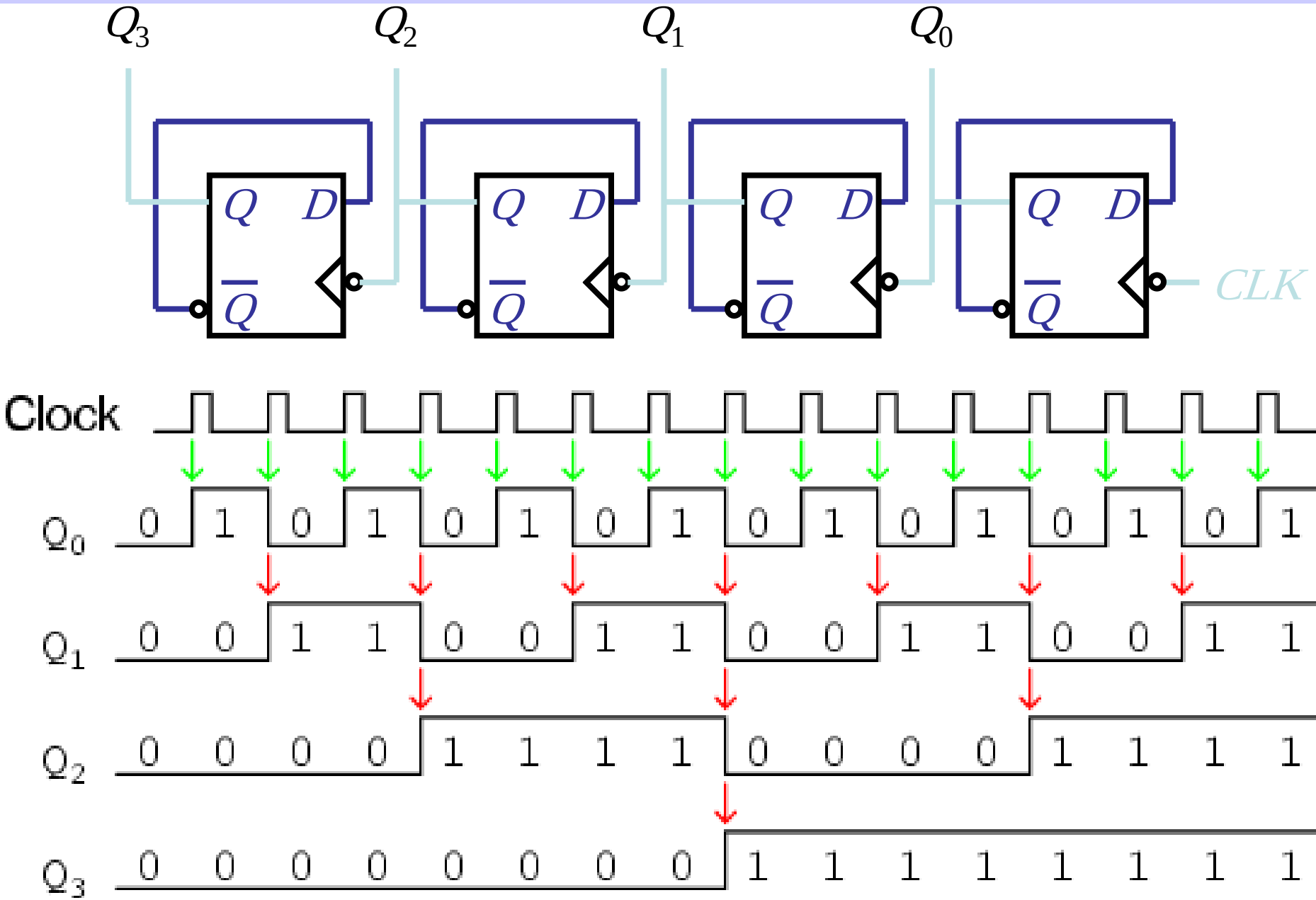


3-Bit Binary Ripple Up Counter - Timing Diagram

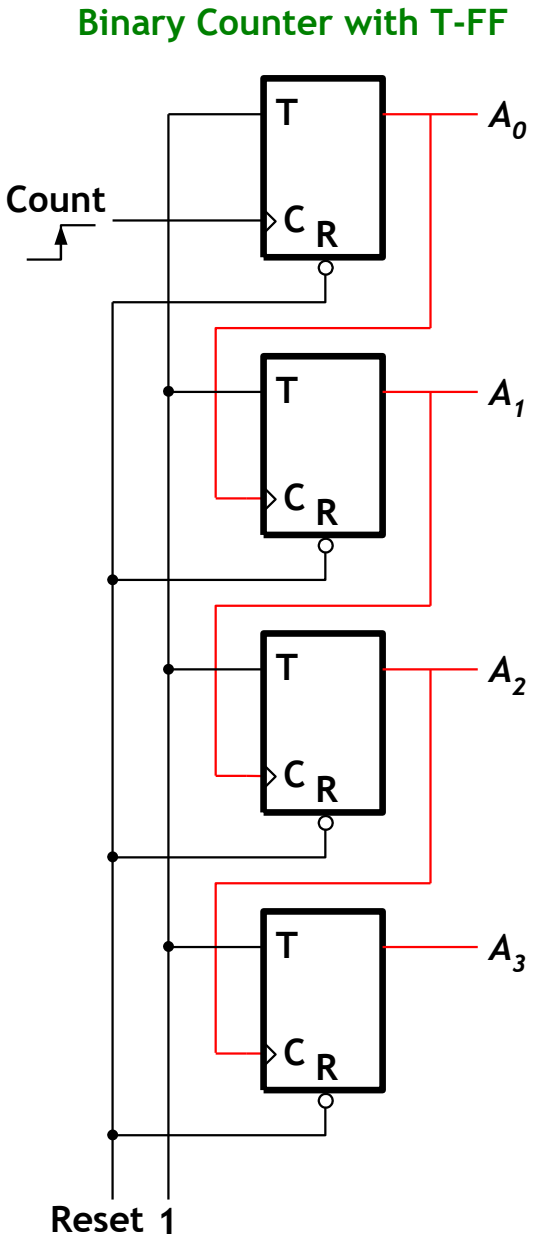


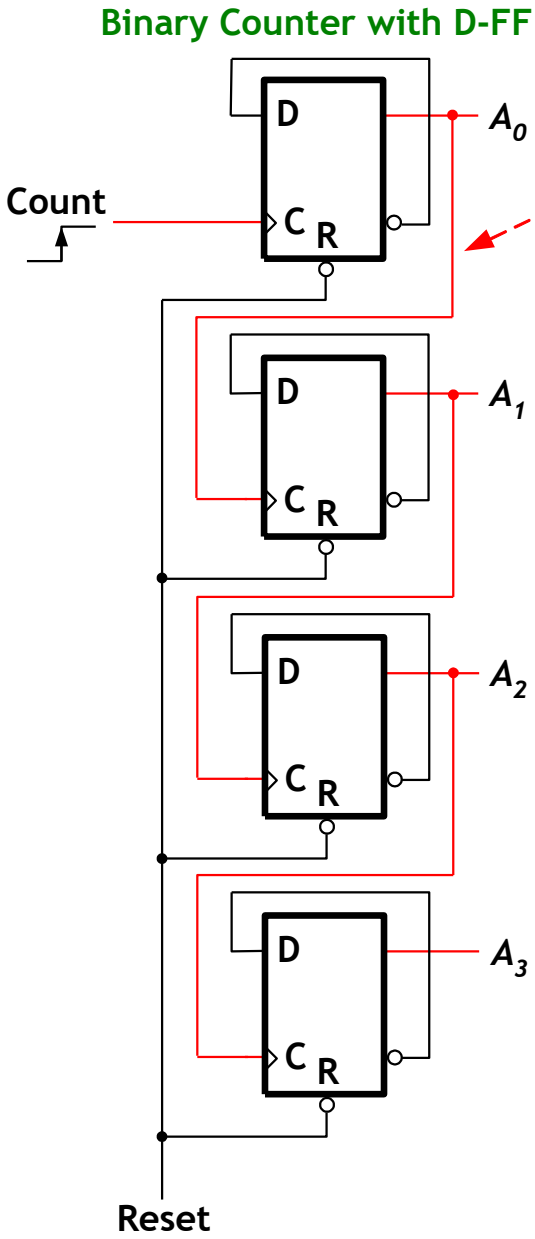


4-Bit Binary Ripple Up Counter



4-Bit Binary Ripple Down Counter





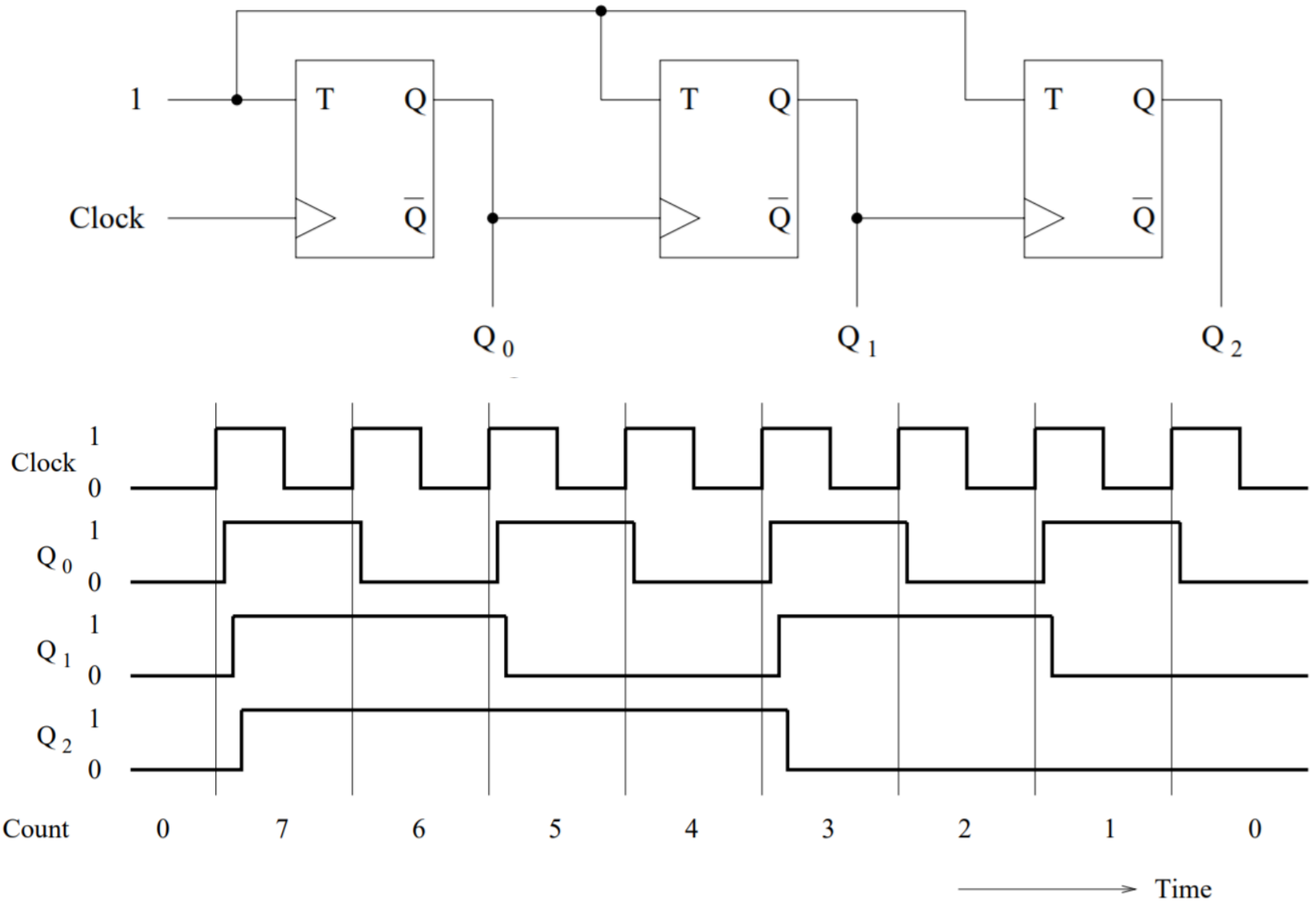
Binary Counter Sequence

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0 T	1
0	0	1	0
0	0 T	1 T	1
0	1	0	0
0	1	0 T	1
0	1	1	0
0 T	1 T	1 T	1
1	0	0	0

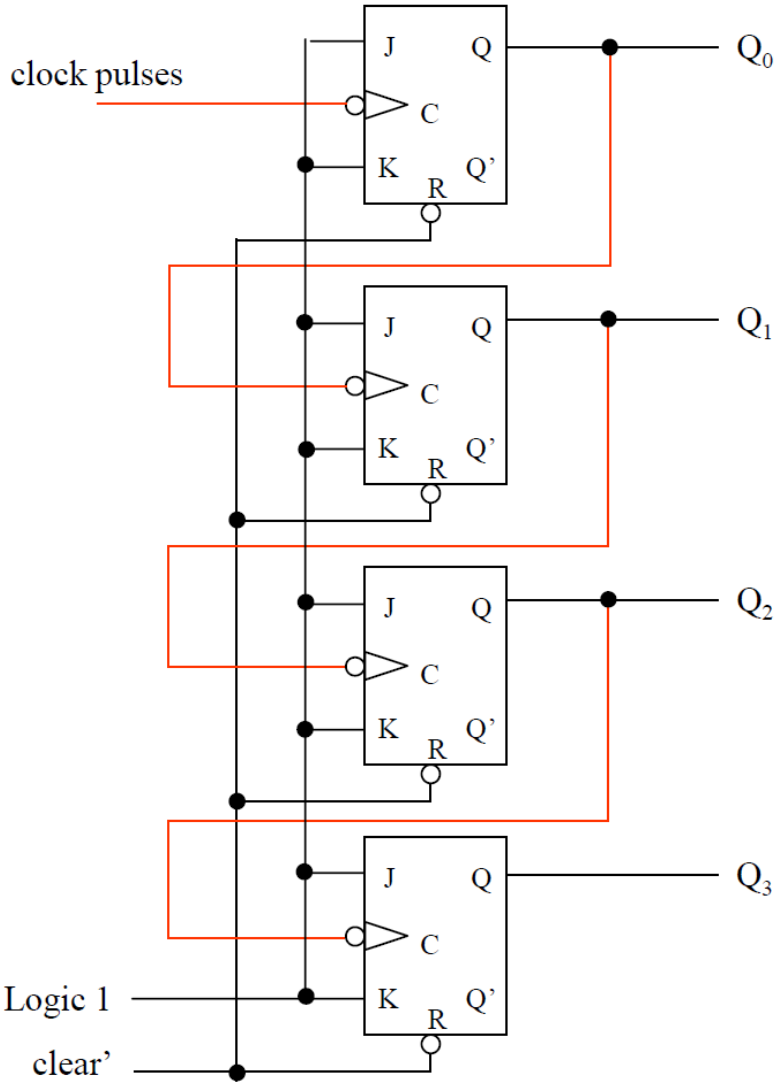
Count Sequence

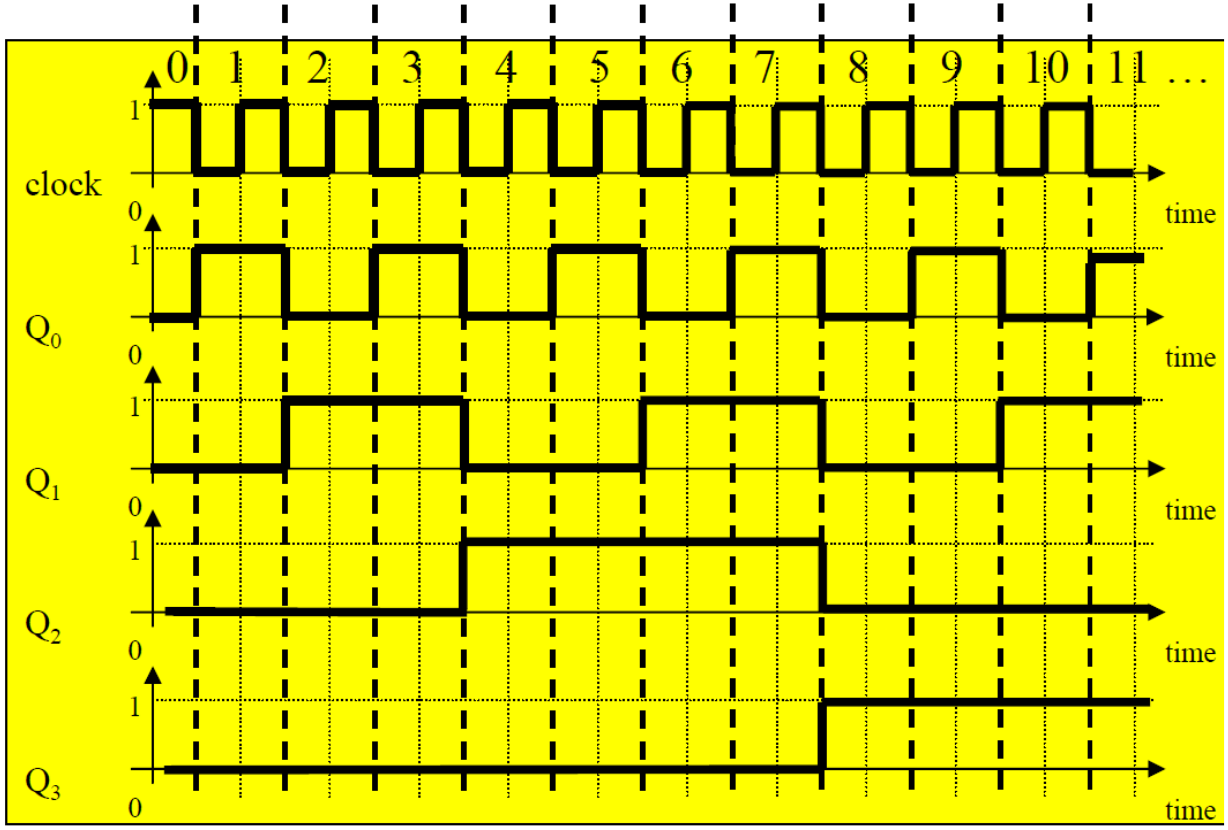
Positive Edge Trigger

3-Bit Binary Ripple Down Counter – Timing Diagram



4-bit Ripple Counter Using JK FF



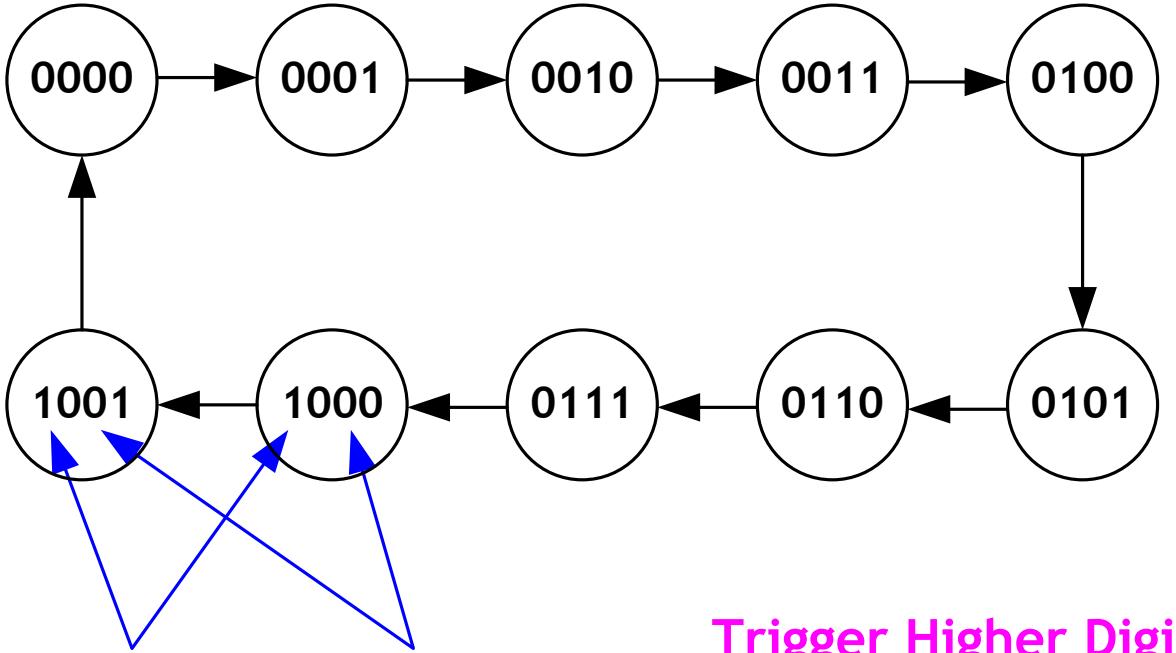


How to modify this circuit to count downwards:

1. Use Q' as input to C
2. Use +ve edge triggered f/f

BCD Ripple Counter

State Diagram of a Decimal BCD Counter



If Q_8 = 1 then Q_2 = 0

2

Trigger Higher Digit

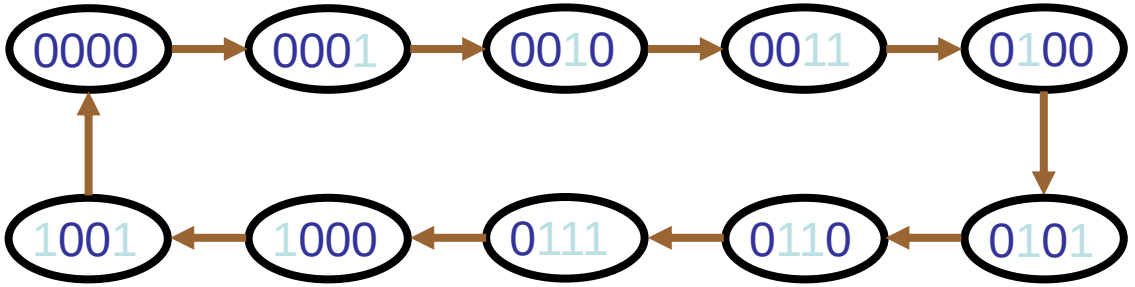
BCD Counter Sequence

Q_8	Q_4	Q_2	Q_1
0	0	0	0
0	0	0 T	1
0	0	1	0
0	0 T	1 T	1
0	1	0	0
0	1 T	0 T	1
0	1	1	0
0 T	1 T	1 T	1
1	0 T	0 T	0
1 T	0	0	1

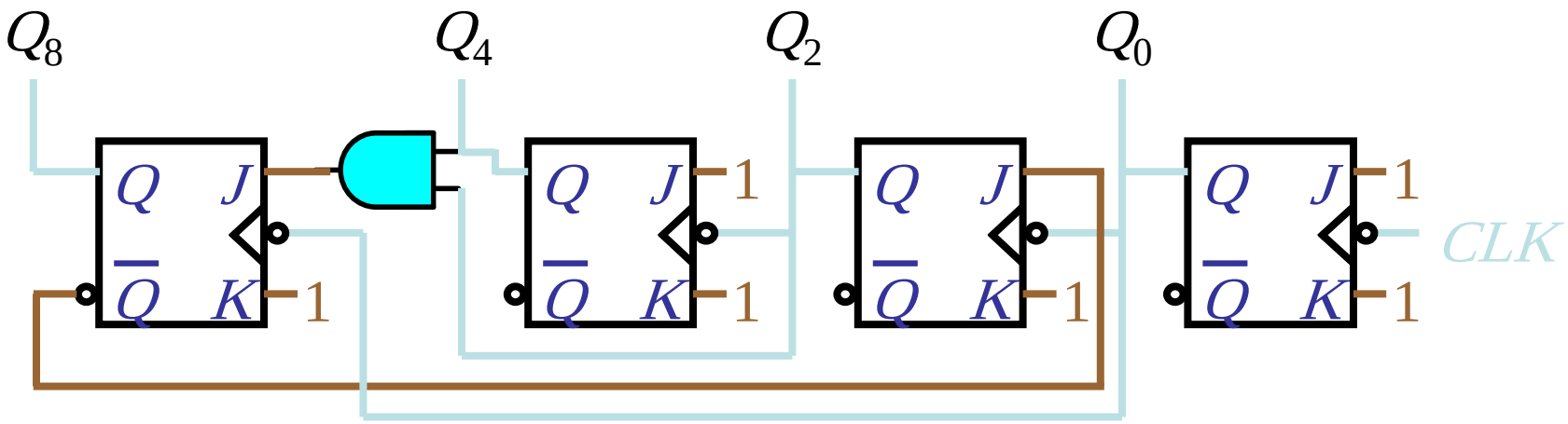
3 1

4 If (Q_2 or Q_4 = 0) then Q_8 Remain at 0
If (Q_2 and Q_4 = 1 and Q_1↓) then Q_8 Complement

BCD Ripple Counter

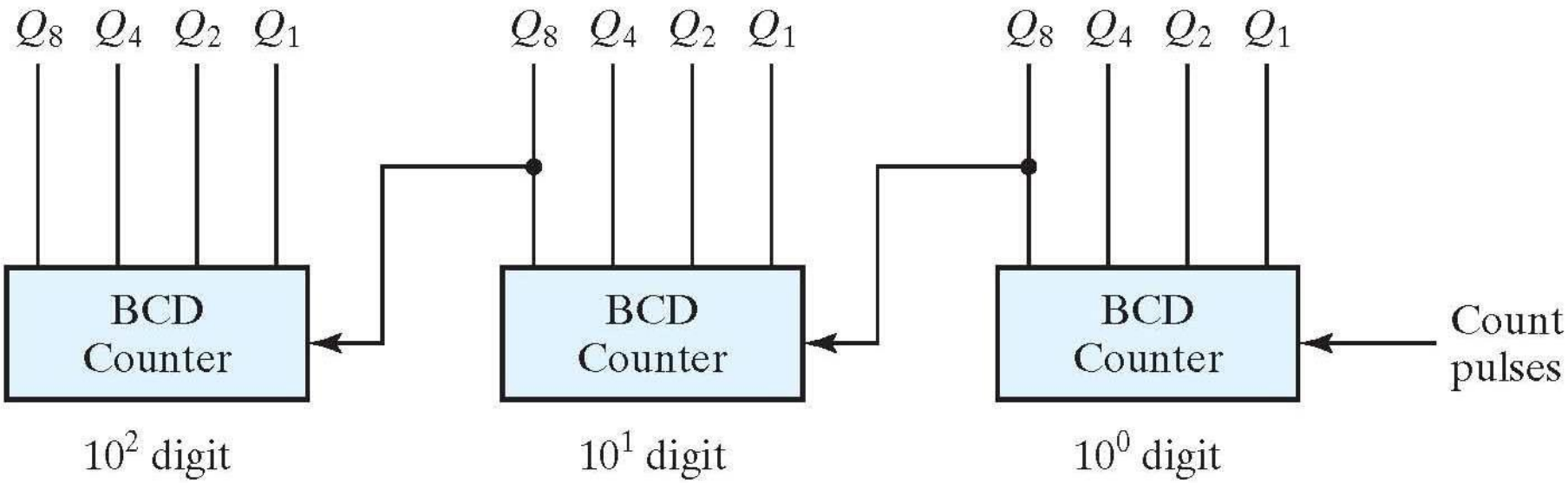


Toggle
Q_1 : CLK(1→0)
Q_2 : Q_8=0 and Q_1(1→0)
Q_4 : Q_2(1→0) Q_8 : Q_4Q_2=11 and Q_1 (1 →0)



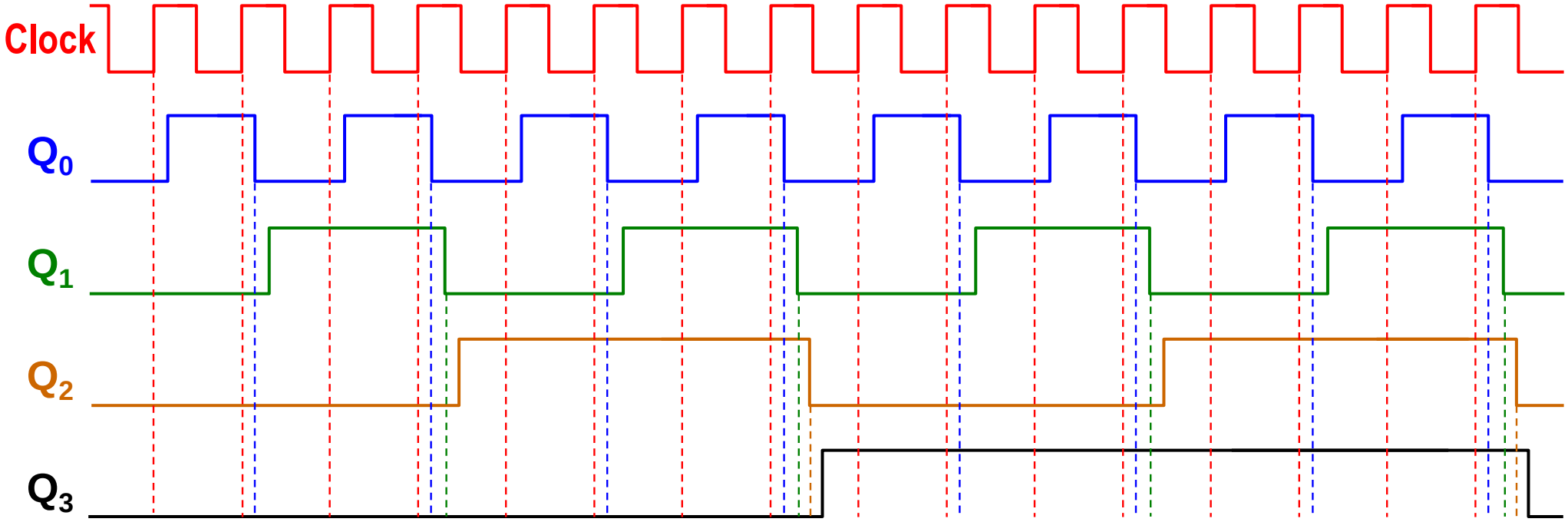
- Q1 complements every count
- Q2 comp when Q1 goes 1➔0 while Q8=0 when Q8 becomes 1, Q2 remains at 0
- Q4 comp when Q2 Goes from 1➔0
- Q8 =0 as long as Q2 or Q4=0, and Complemented when both are 1 And Q1 goes 1➔0

Decades Counter



- The inputs to the second and third decades come from Q_8 of the previous decade.
- When Q_8 in one decade goes from 1 to 0, it triggers the count for the next higher order decade while its own decade goes from 9 to 0.

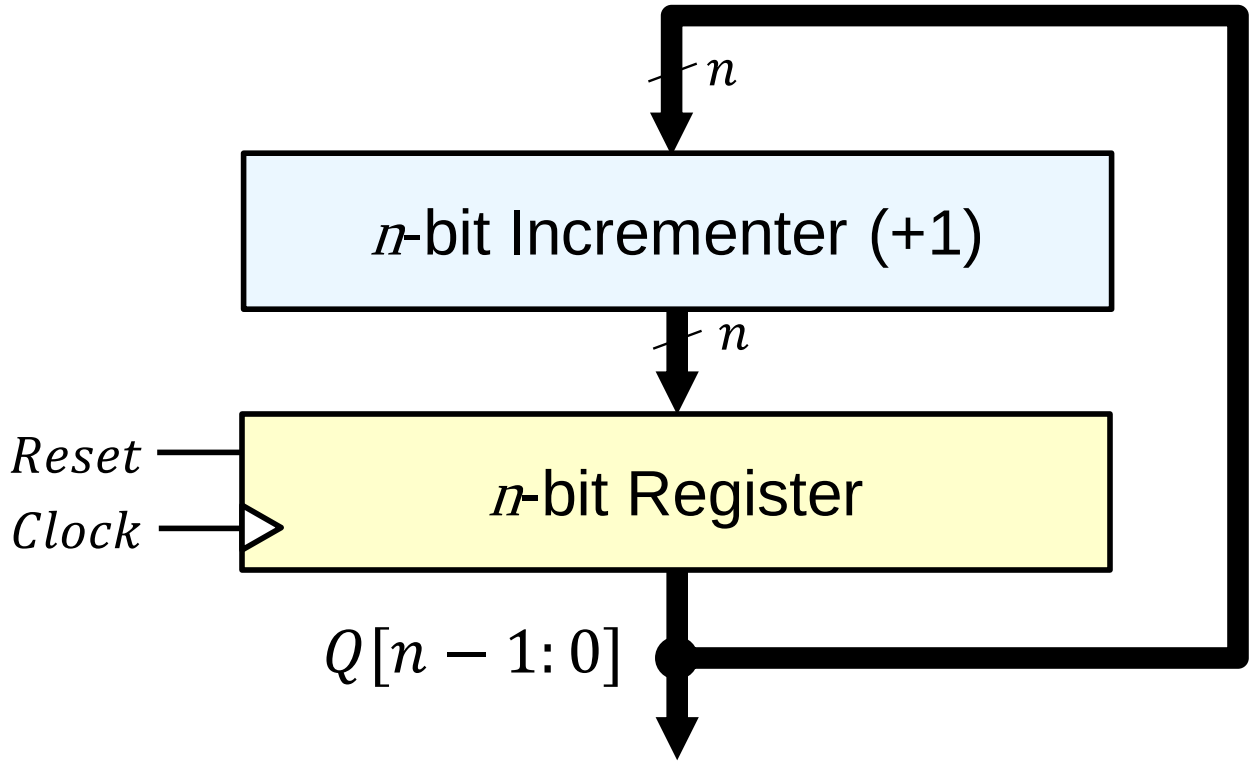
Drawback of ripple counter



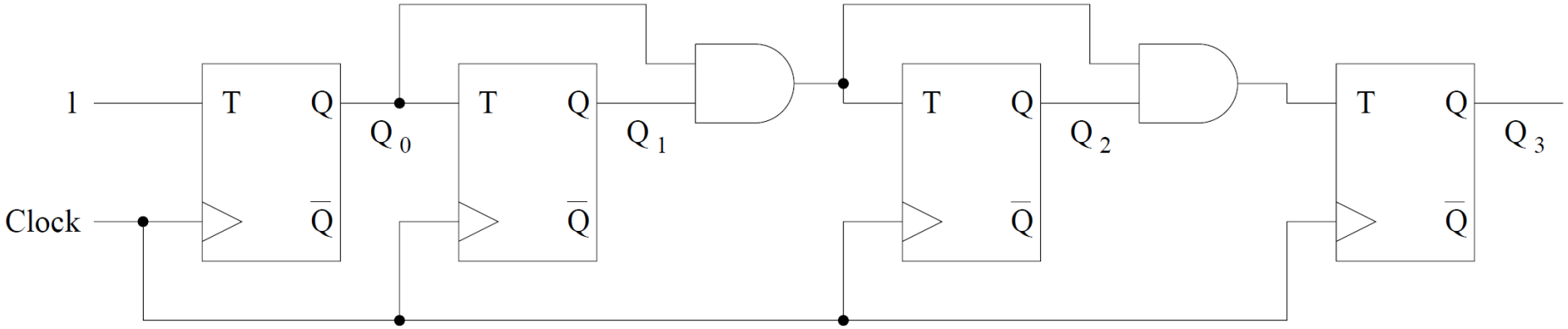
- Flip-flops are NOT driven by the same clock (Not Synchronous)
- Q delay increases as we go from Q_0 to Q_3
- Given Δ = flip-flop delay ➔ Delay of Q_0, Q_1, Q_2, Q_3 = Δ, 2Δ, 3Δ, 4Δ

Synchronous Counter

- Avoid clock rippling
- n-bit Register with a **common clock** for all flip-flops
- n-bit Incrementer to generate next state (Up-Counter)



Synchronous Up-Counter with T Flip-Flops



Clock Cycle	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Q_1 changes

Q_2 changes

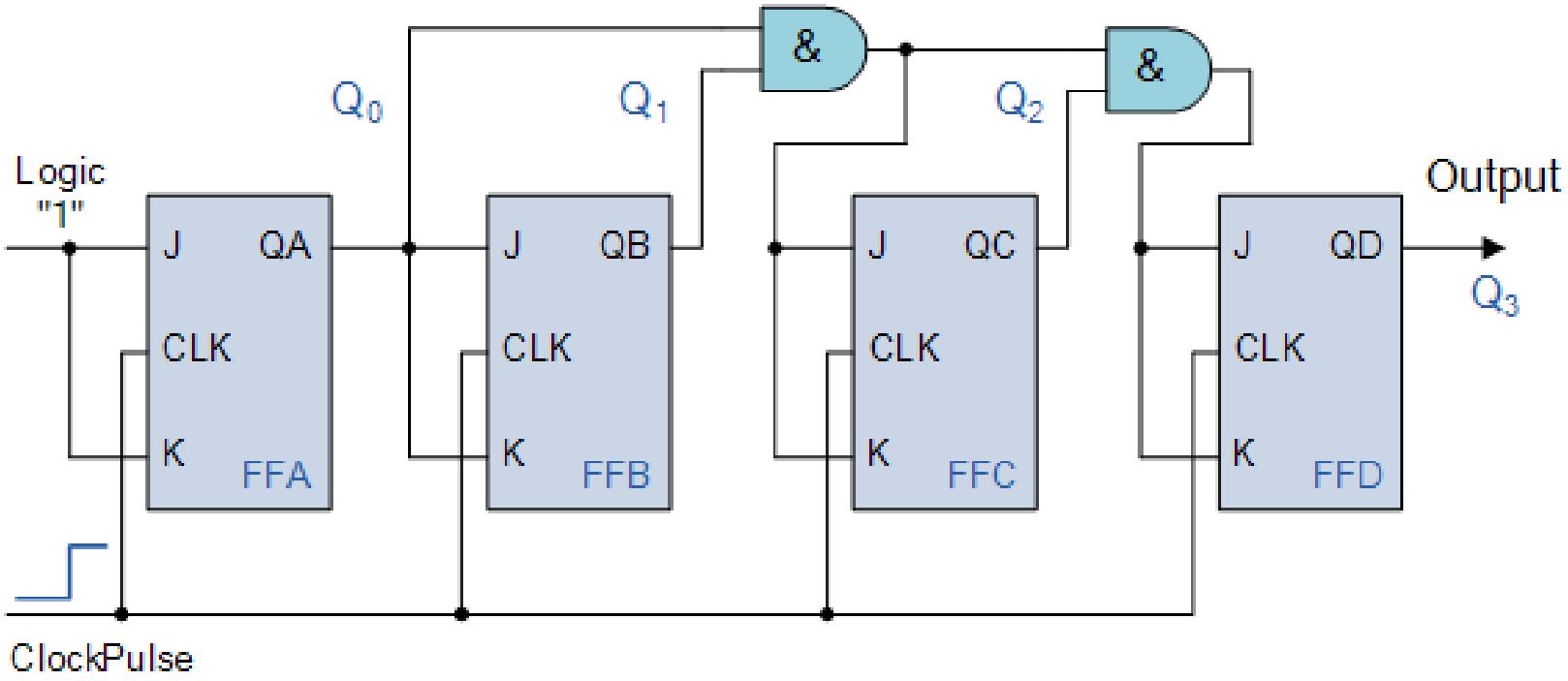
Q_3 changes

In general, for an *n*-bit up-counter, a give flip-flop changes its state only when all the preceding flip-flops are in the state $Q = 1$.

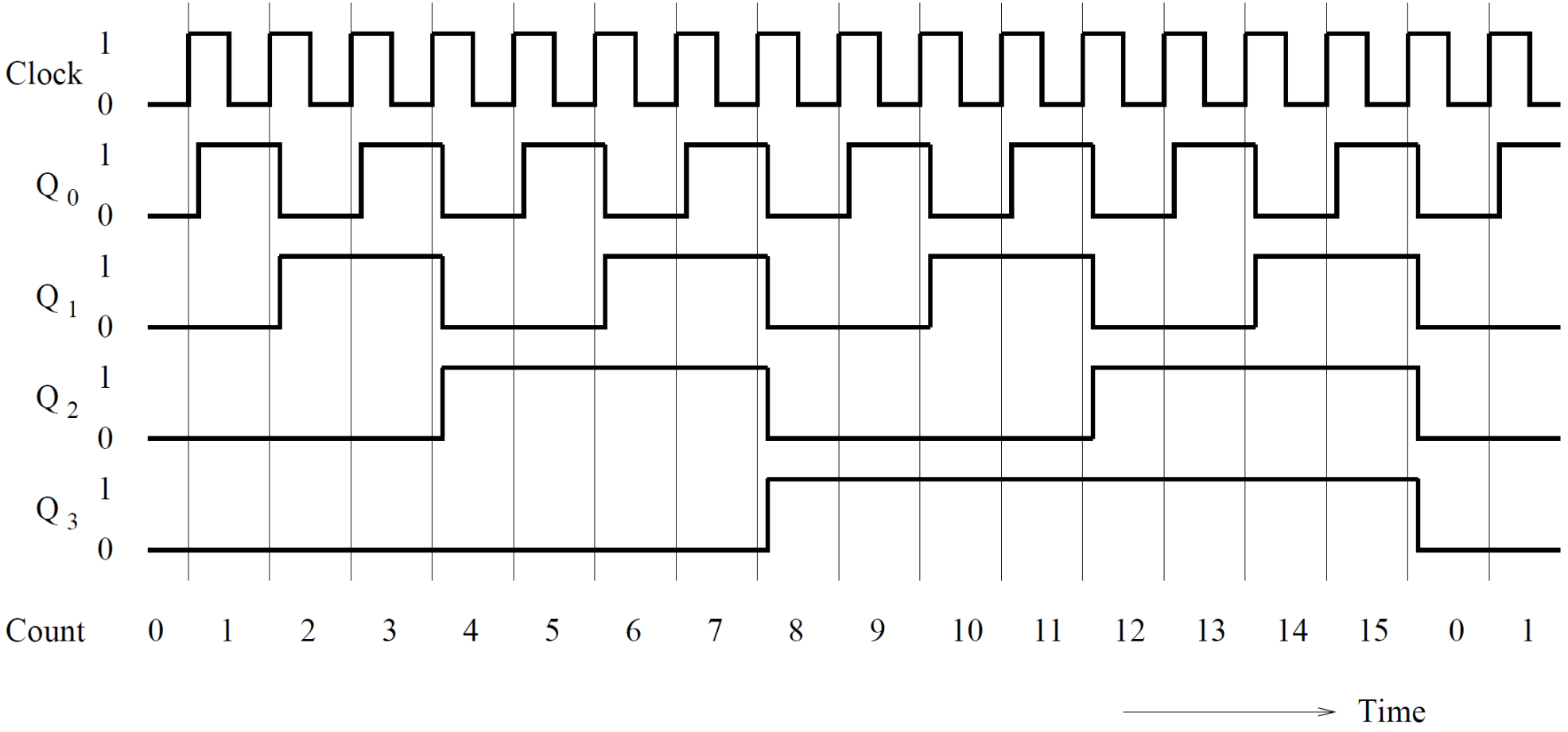
Therefore, if we use T flip-flops to realize the 4-bit counter, then the T inputs should be defined as

$$
\begin{aligned}
T_0 &= 1 \\
T_1 &= Q_0 \\
T_2 &= Q_0 Q_1 \\
T_3 &= Q_0 Q_1 Q_2
\end{aligned}
$$

Synchronous Up-Counter with JK Flip-Flops

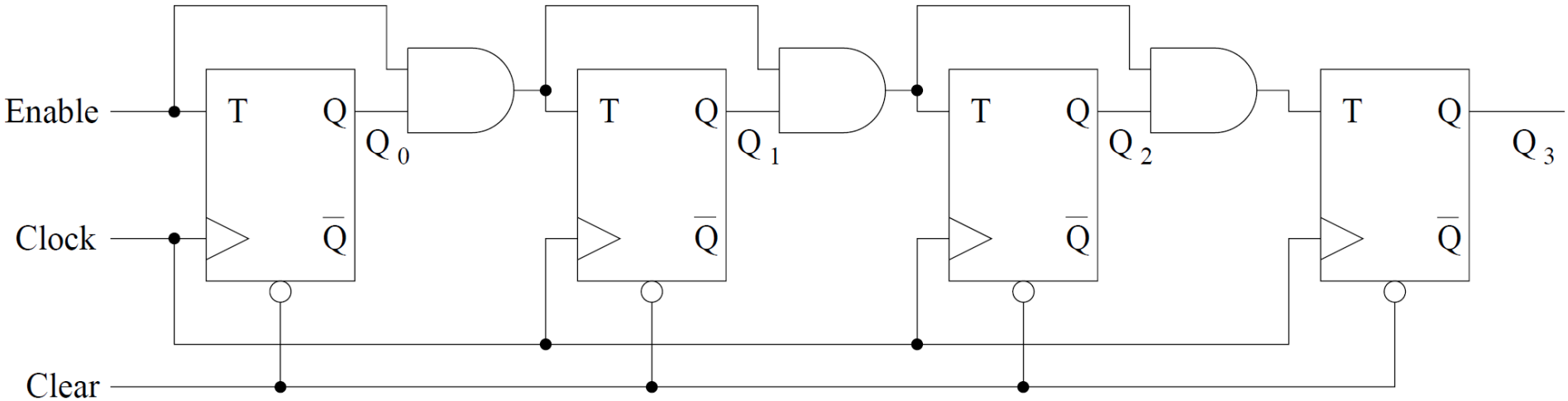


Timing diagram for a 4-bit synchronous up-counter



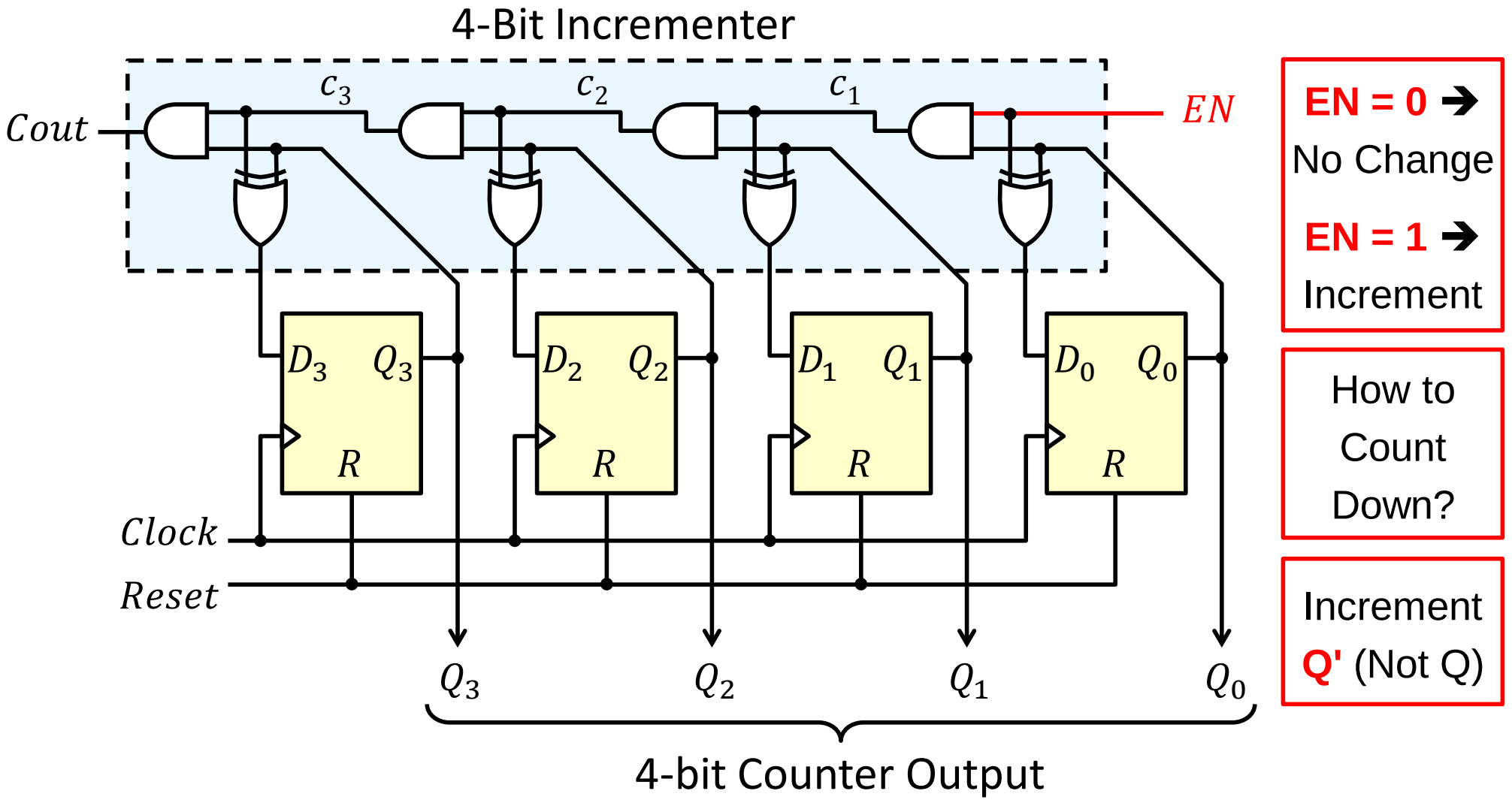
Enable and Clear Capability

- if *Enable* = 0, then all *T* inputs will be equal to 0. If *Enable* = 1, then the counter operates as explained previously
- In many applications, it is also necessary to start with the count equal to zero. This is easily achieved if the flip-flops can be cleared

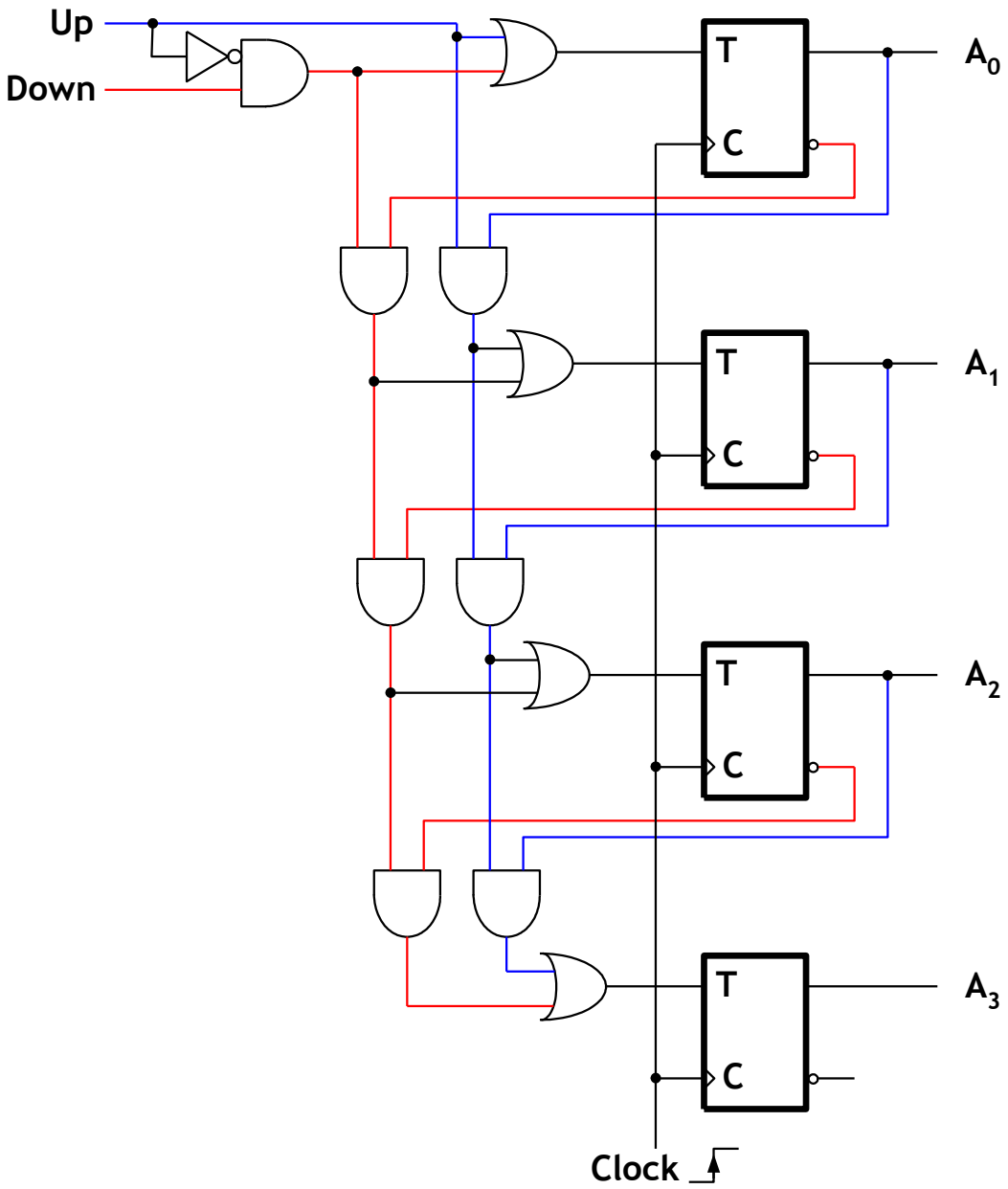


4-Bit Synchronous Counter with Enable

- An incrementer is a reduced (contracted) form of an adder



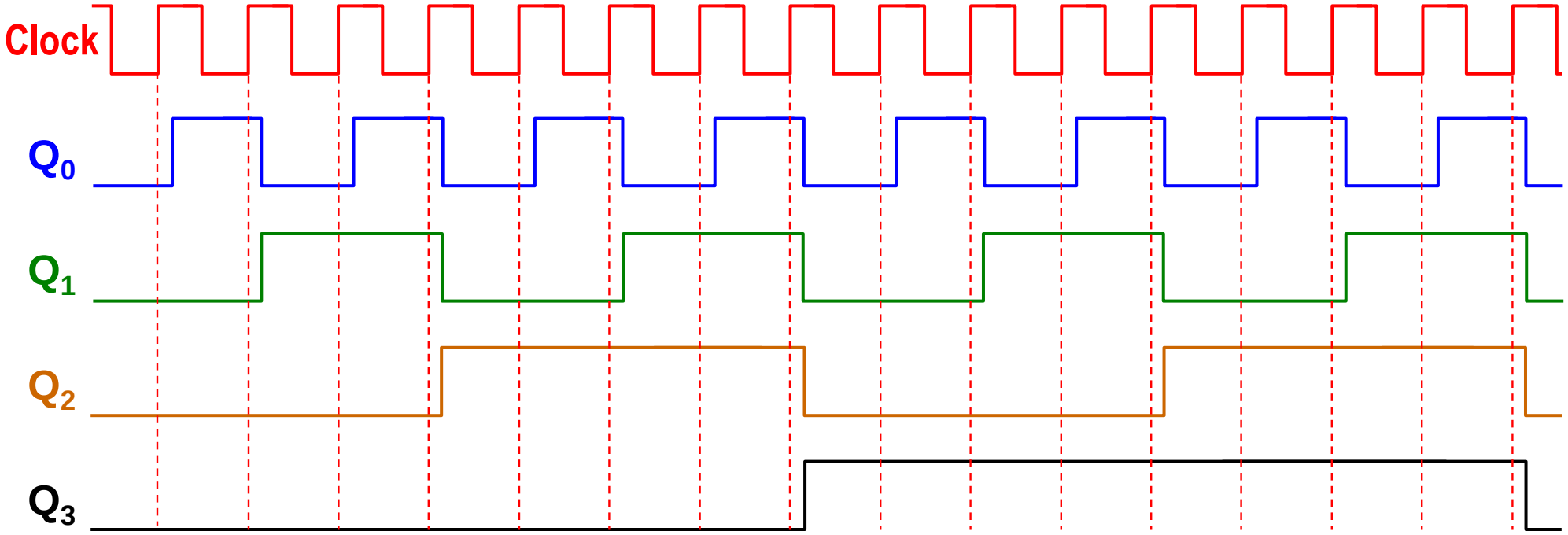
4-bit Binary Up-Down Counters



Function Table

Up	Down	Function
0	0	No Change
0	1	Down Count
1	0	Up Count
1	1	Up Count

Timing of a Synchronous Counter



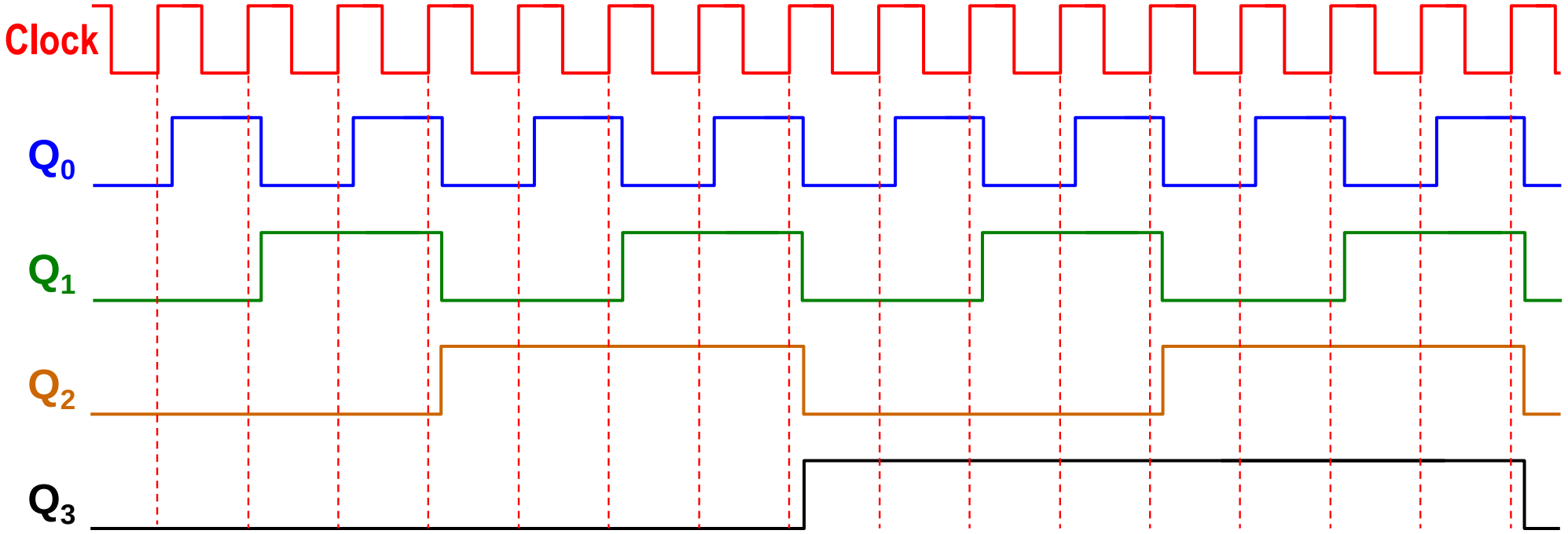
❖ **Advantage of Synchronous counter:**

ALL Flip-flops are driven by the **same clock**

Delay of all outputs is identical ➔ Delay of $Q_0 = Q_1 = Q_2 = Q_3 = \Delta$

Frequency Division

- A counter can be used as a frequency divider
- Counter is driven by a **Clock** with **frequency F**
- Output Q_0 Frequency = **F/2**, Output Q_1 Frequency = **F/4**
- Output Q_2 Frequency = **F/8**, Output Q_3 Frequency = **F/16**



BCD Counter

- A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000.
- Because of the return to 0 after a count of 9, a BCD counter does not have a regular pattern, unlike a straight binary count.
- To derive the circuit of a BCD synchronous counter, it is necessary to go through a sequential circuit design procedure.

Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

BCD Counter

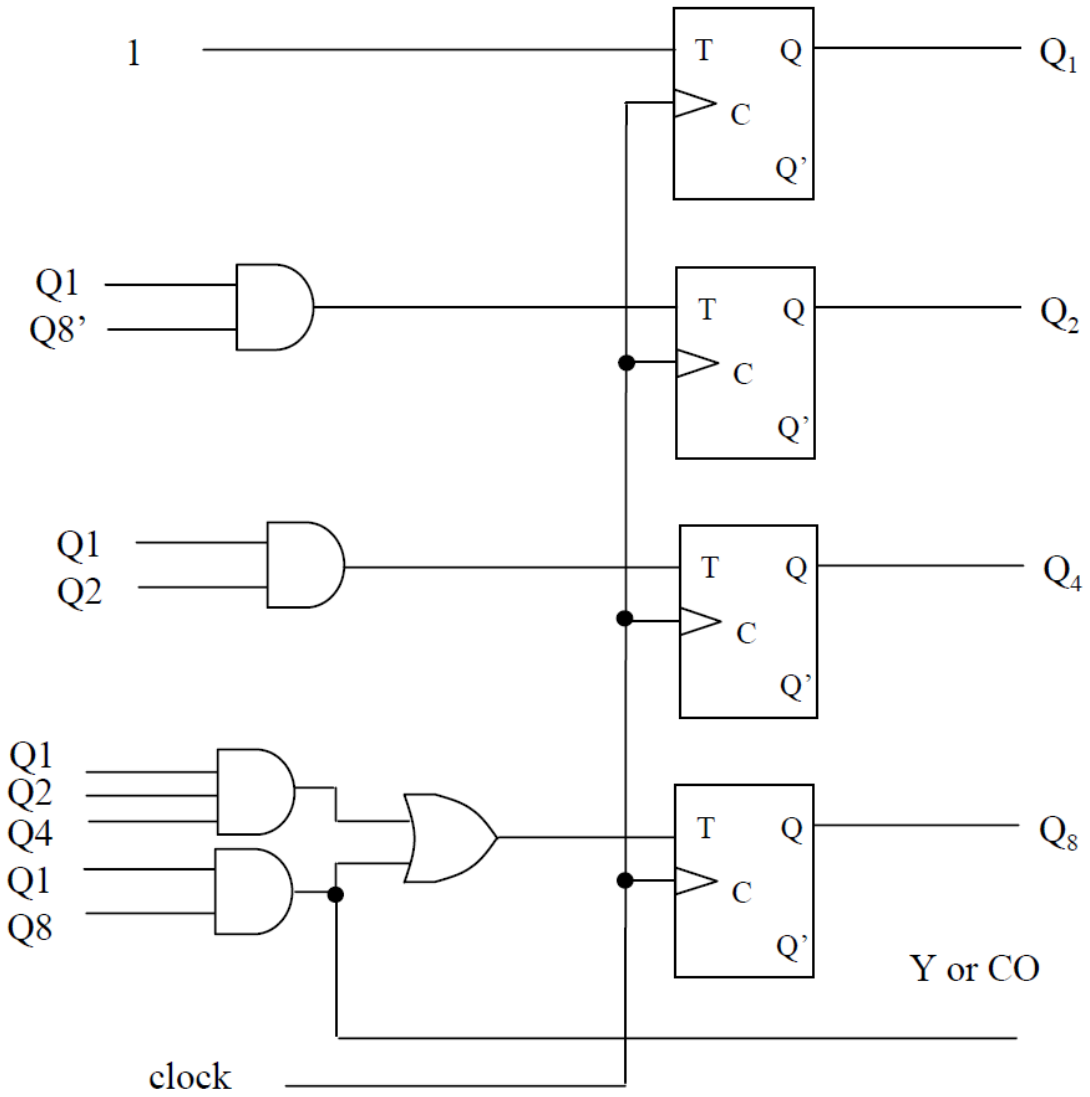
$T_{Q1} = 1$

$T_{Q2} = Q_1 Q_8'$

$T_{Q4} = Q_1 Q_2$

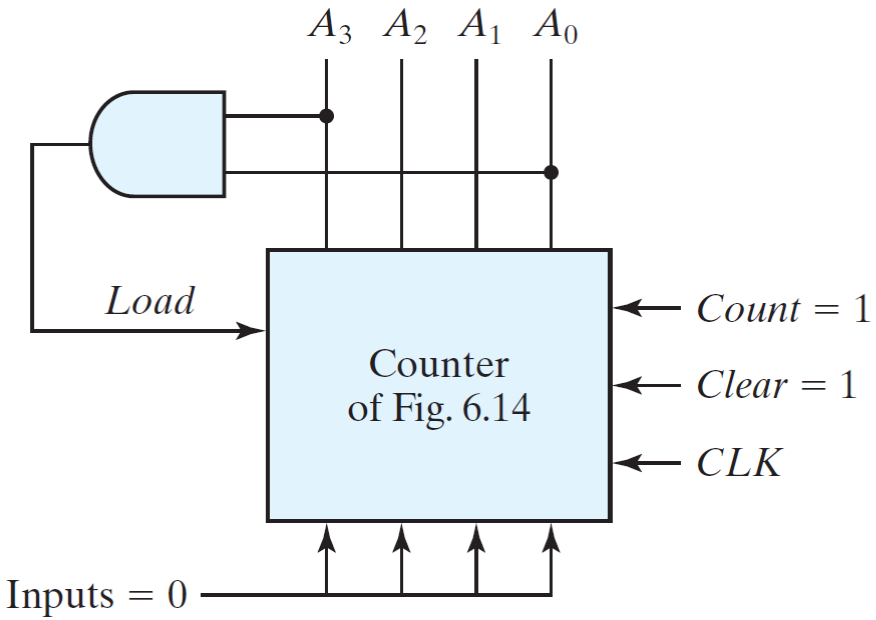
$T_{Q8} = Q_1 Q_8 + Q_1 Q_2 Q_4$

$Y = Q_8 Q_1$

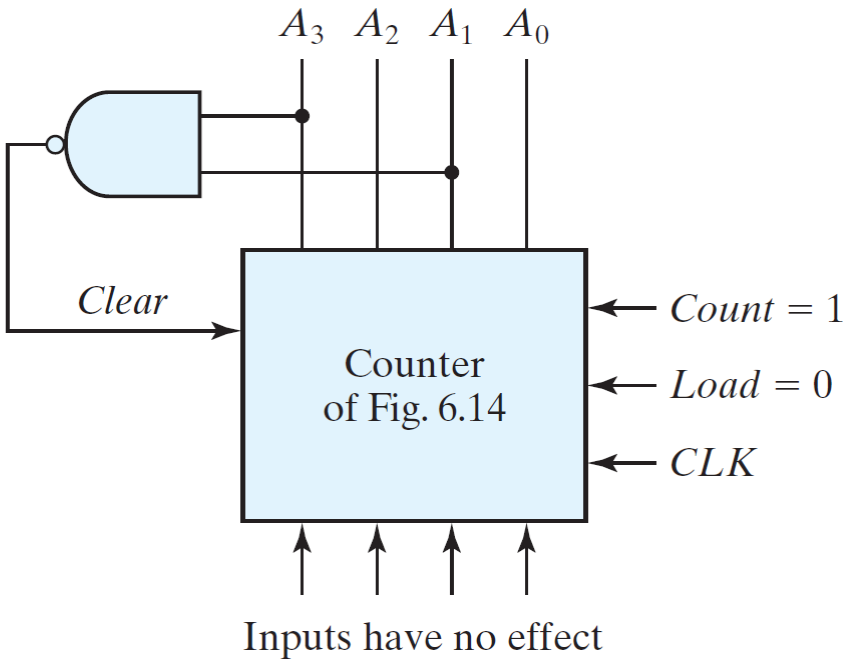


BCD Counter – Alternative Design

- **Problem:** Convert a 4-bit binary counter into a BCD counter
- **Solution:** When output reaches **9** then reset back to **0**
- **Asynchronous Reset:** Count to **10** and reset immediately



(a) Using the load input

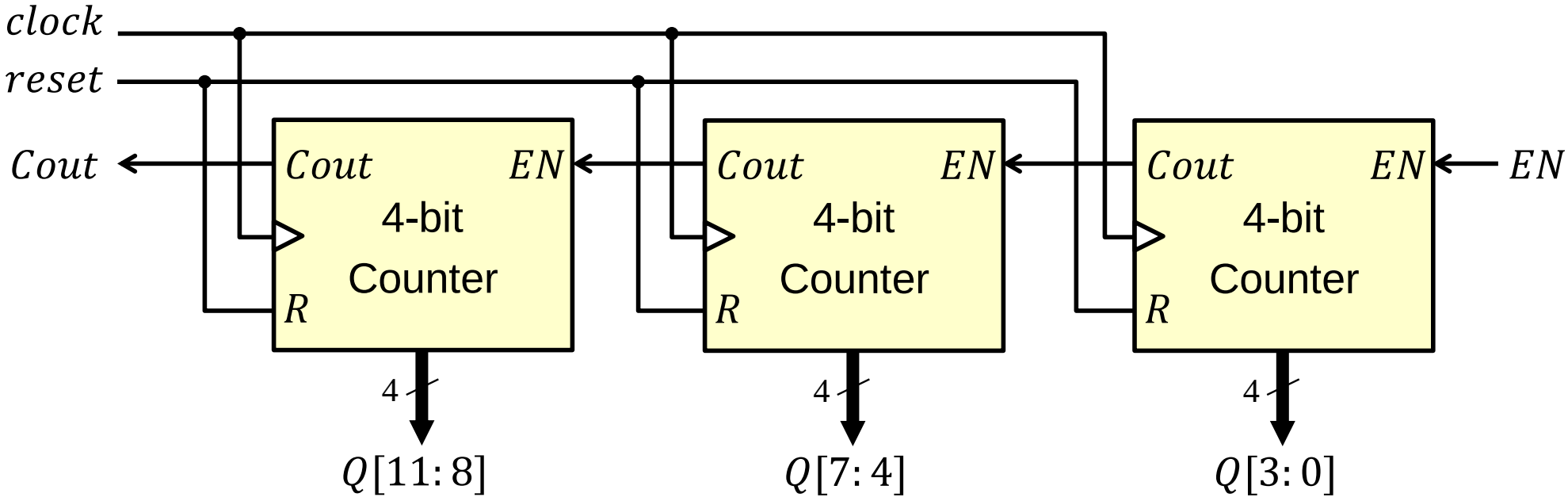


(b) Using the clear input

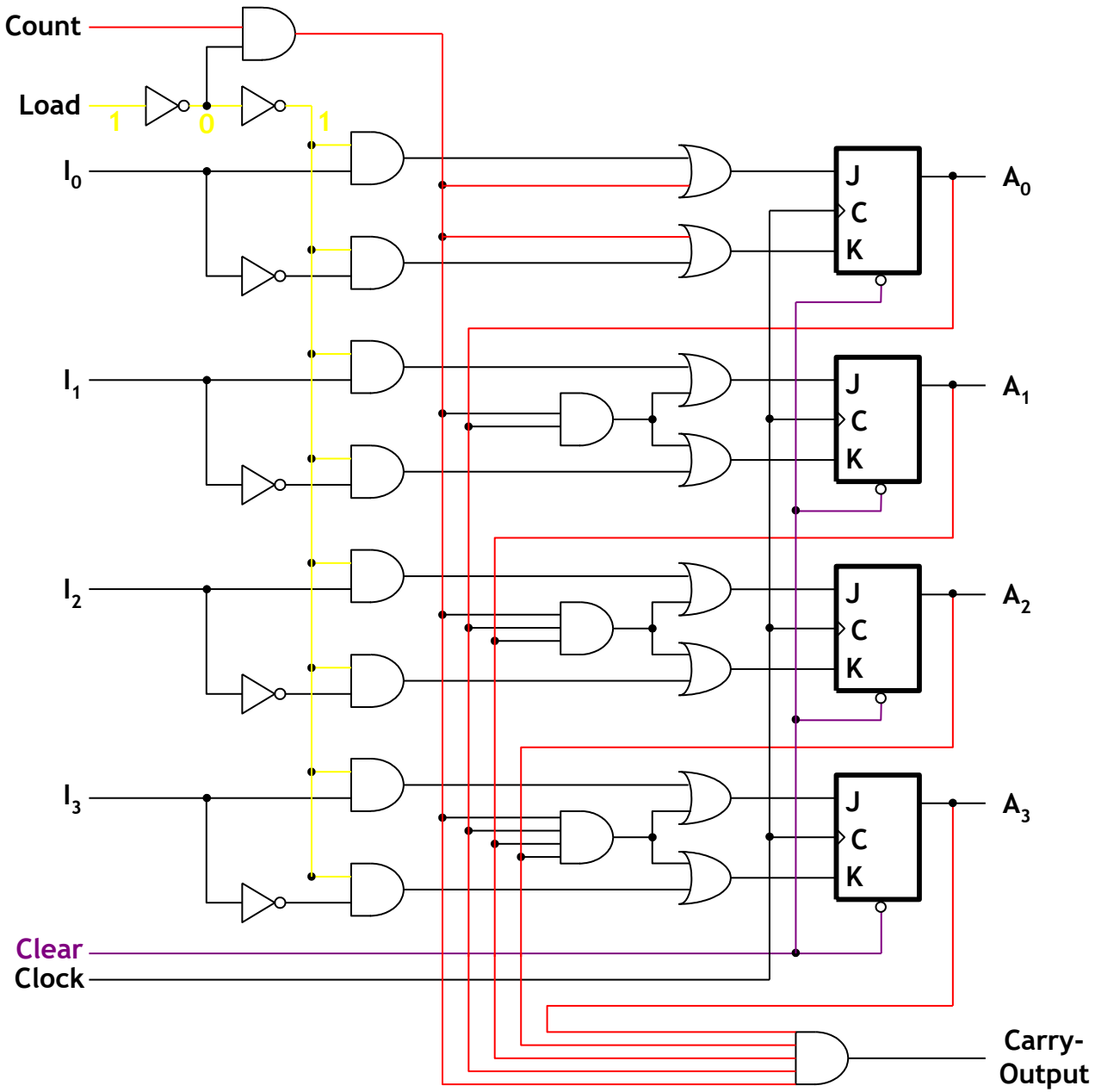
Building Larger Synchronous Counters

- Smaller counters can be used to build a larger counter
- Example: 12-bit counter designed using three 4-bit counters

  Counts from **0** to **4095** ($2^{12} - 1$), then back to **0**
- The $Cout$ of a 4-bit counter is used to enable the next counter



Synchronous Counter with Parallel Load

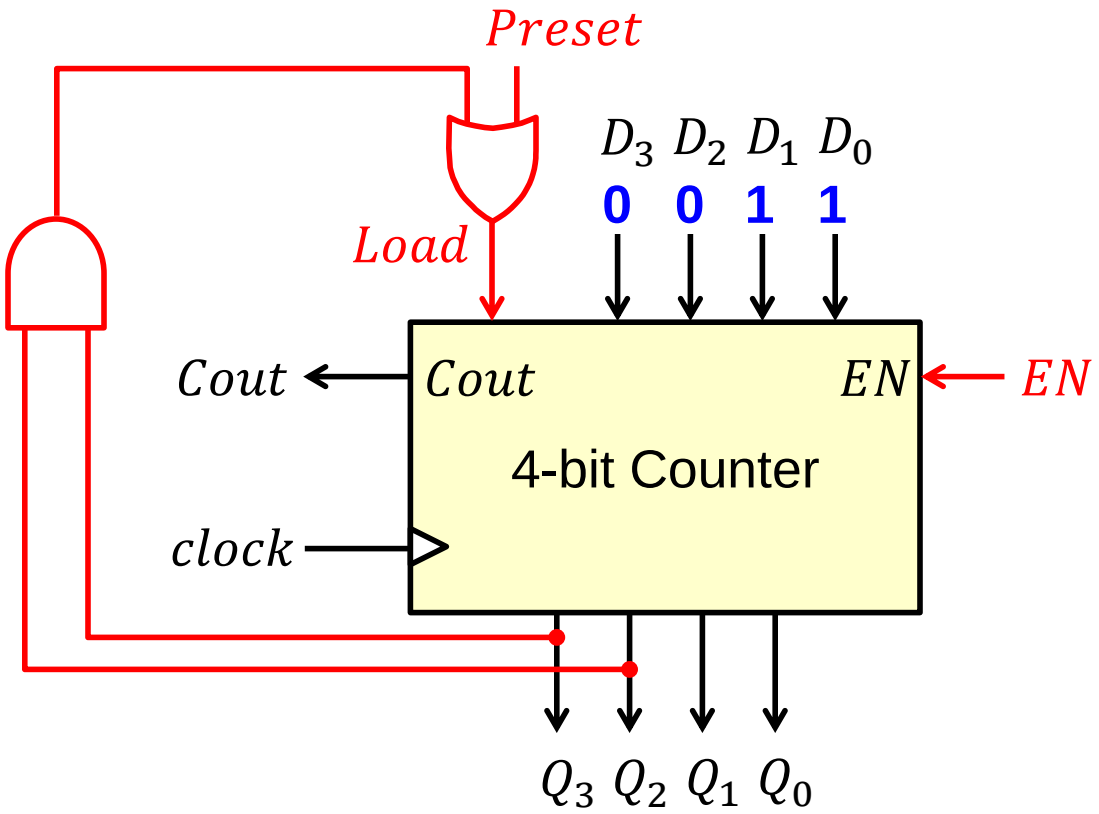


Function Table

Clear	Clock	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	load Inputs
1	↑	0	1	Up Count
1	↑	0	0	No Change

3-to-12 Binary Counter

- Convert a 4-bit binary counter with load into 3-to-12 counter
- **Solution:** Detect binary count **12** and then load **3**
- **Detect 12:** Binary count with $Q_3 = Q_2 = 1$



Synchronous

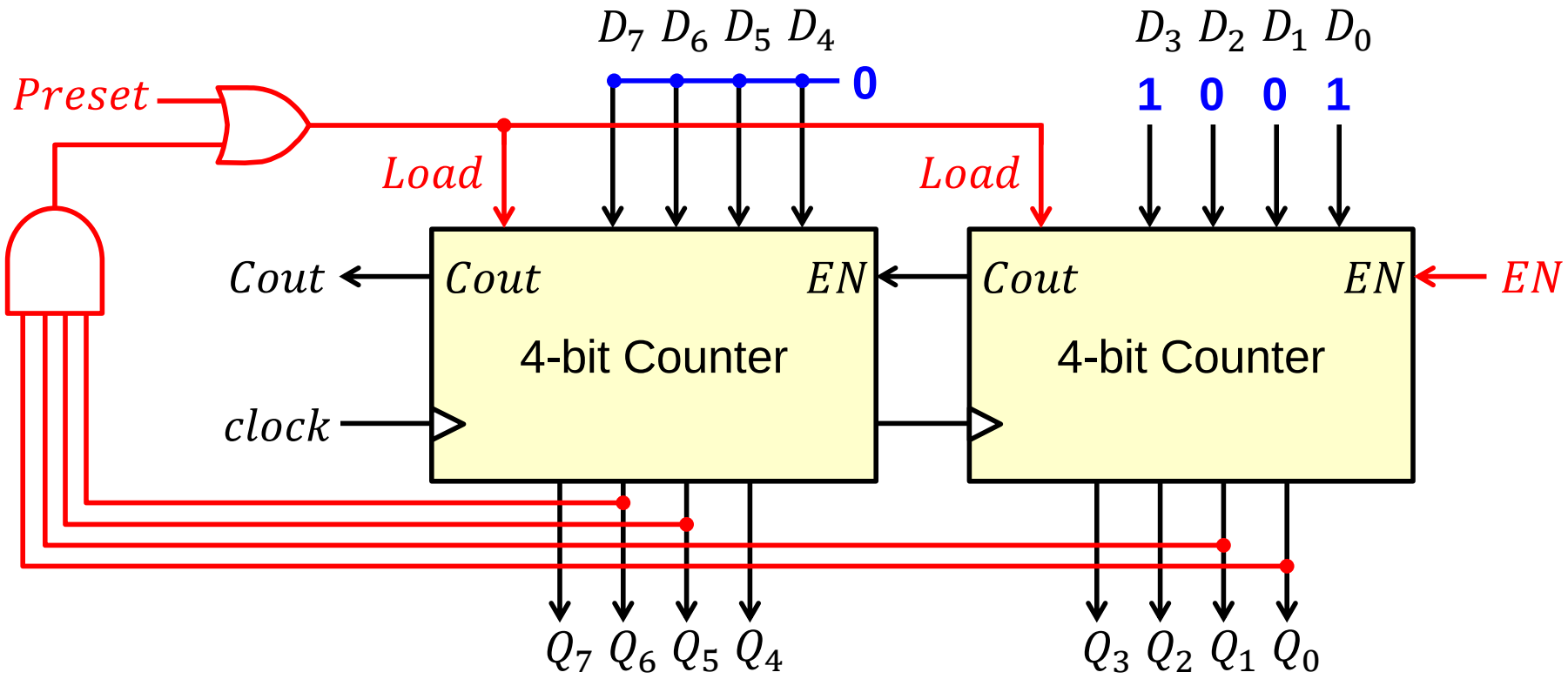
Preset

Load: 0011

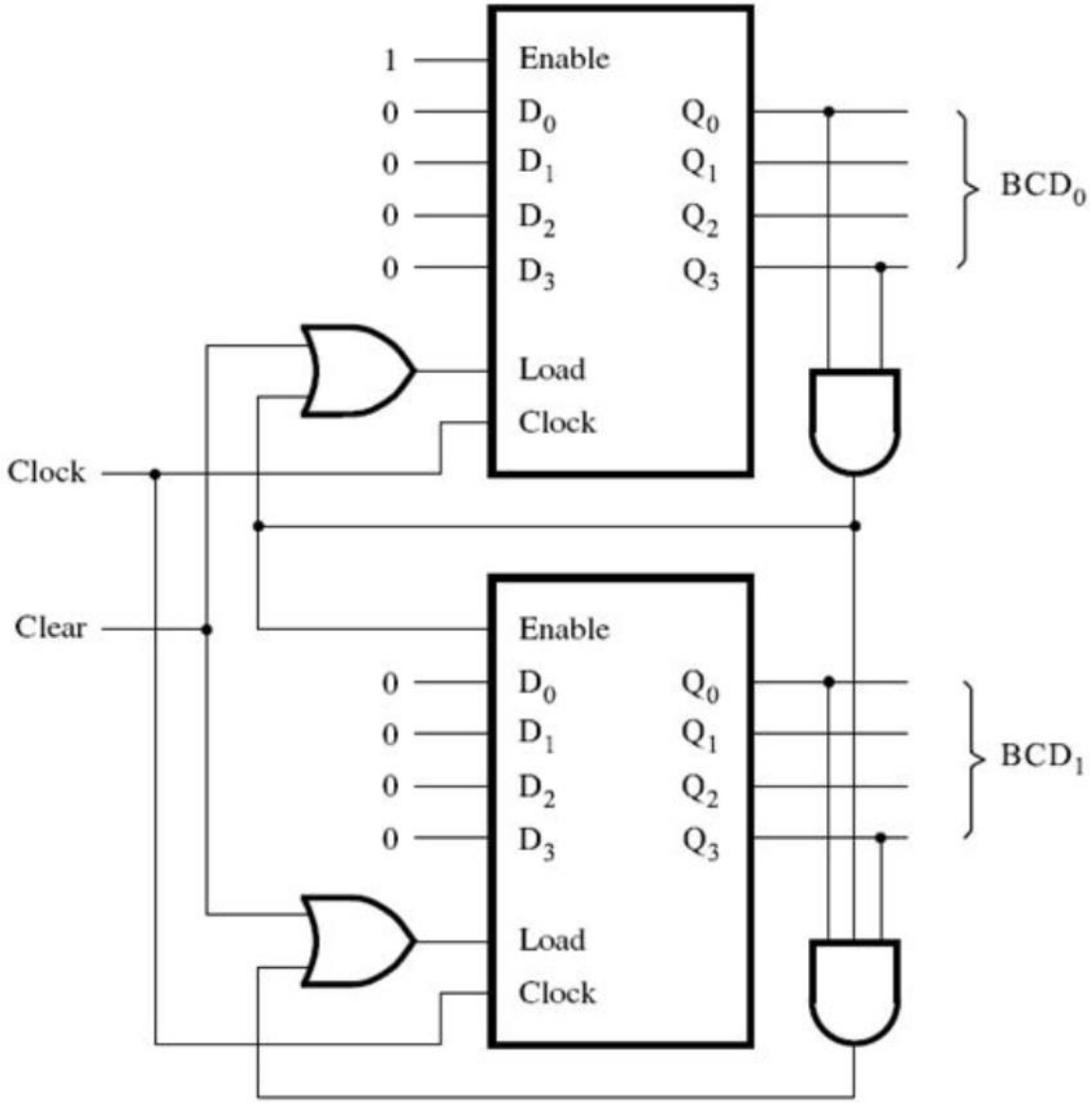
9-to-99 Binary Counter

Problem: Use two 4-bit binary counters with parallel load and logic gates to build a counter that counts from **9** to **99 = 'b01100011**

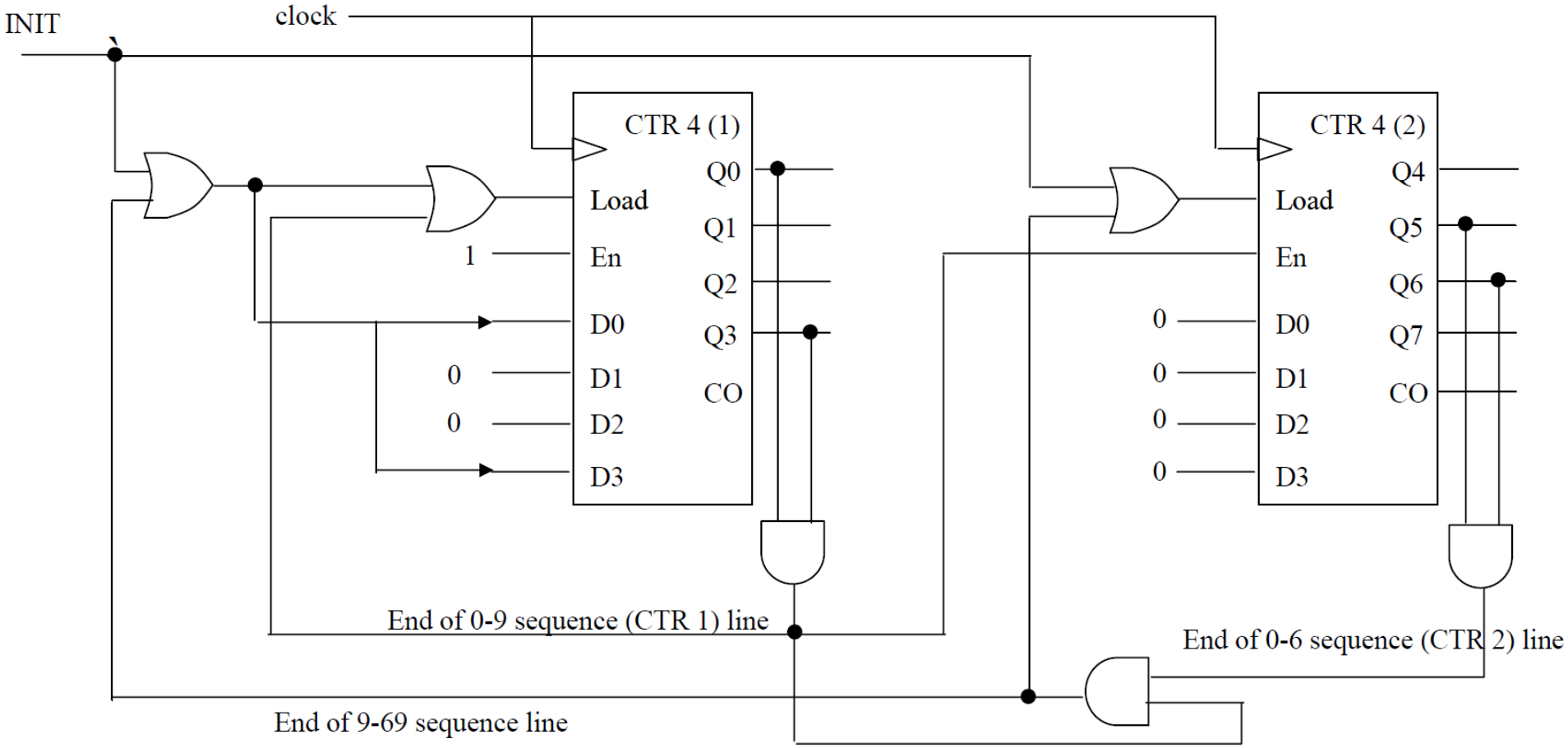
Add a synchronous **Preset** input to initialize the counter to value **9**



Two digits BCD counter (from 00 to 99)

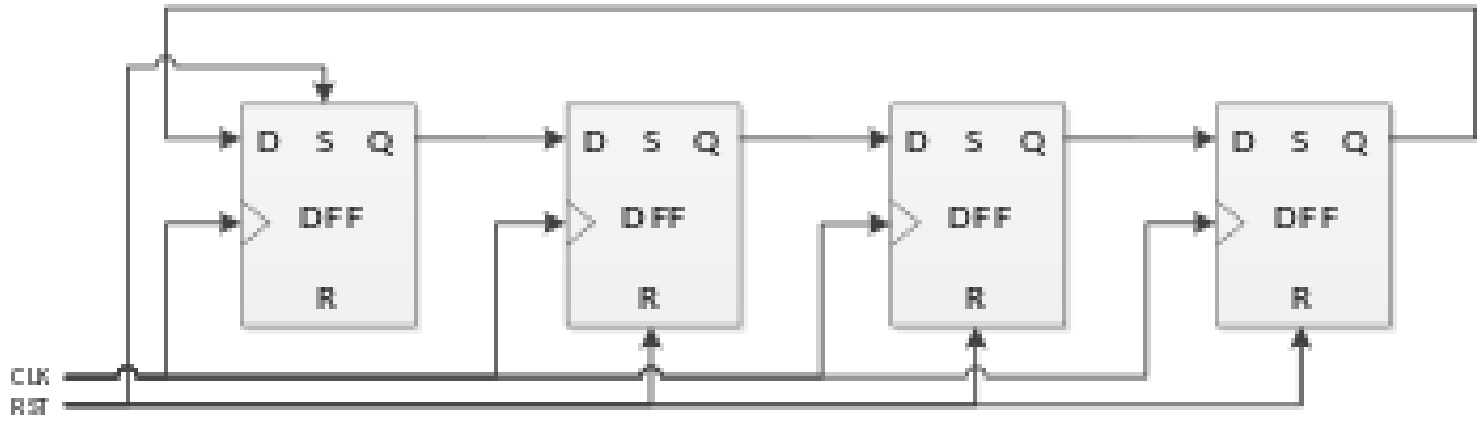


Two digits BCD counter (from 09 to 69)

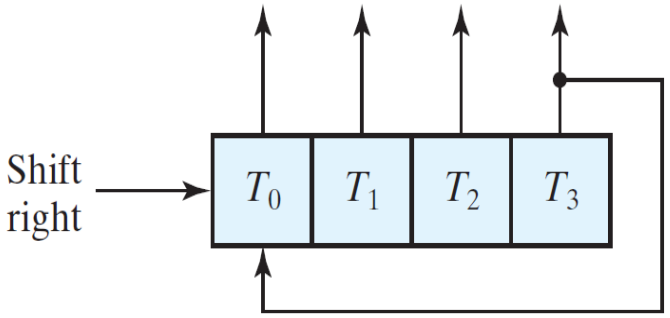


Ring counter

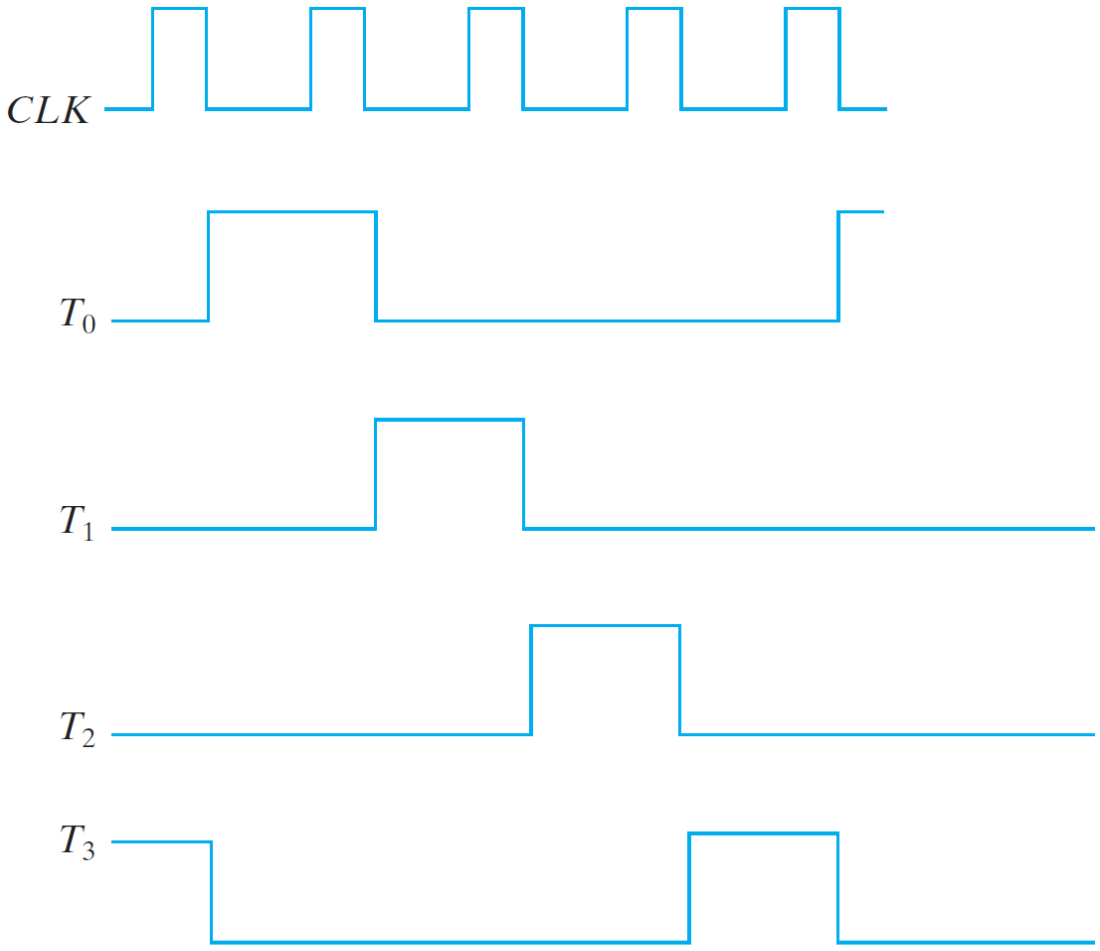
- Timing signals that control the sequence of operations in a digital system can be generated by a shift register or by a counter with a decoder
- A *ring counter* is a circular shift register with:
  - Only one flip-flop being set at any particular time; all others are cleared.
  - The output of the last flip-flop fed to the input of the first, making a "circular" or "**ring**" structure.
  - The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals.



Ring Counter



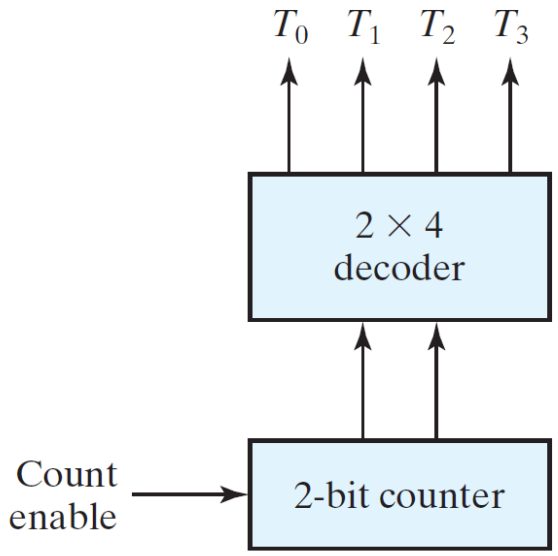
(a) Ring-counter (initial value = 1000)



(b) Sequence of four timing signals

Alternative design

- For an alternative design, the timing signals can be generated by a two-bit counter that goes through four distinct states.
- To generate 2^n timing signals, we need either a shift register with 2^n flip-flops or an n-bit binary counter together with an n-to-2^n-line decoder.



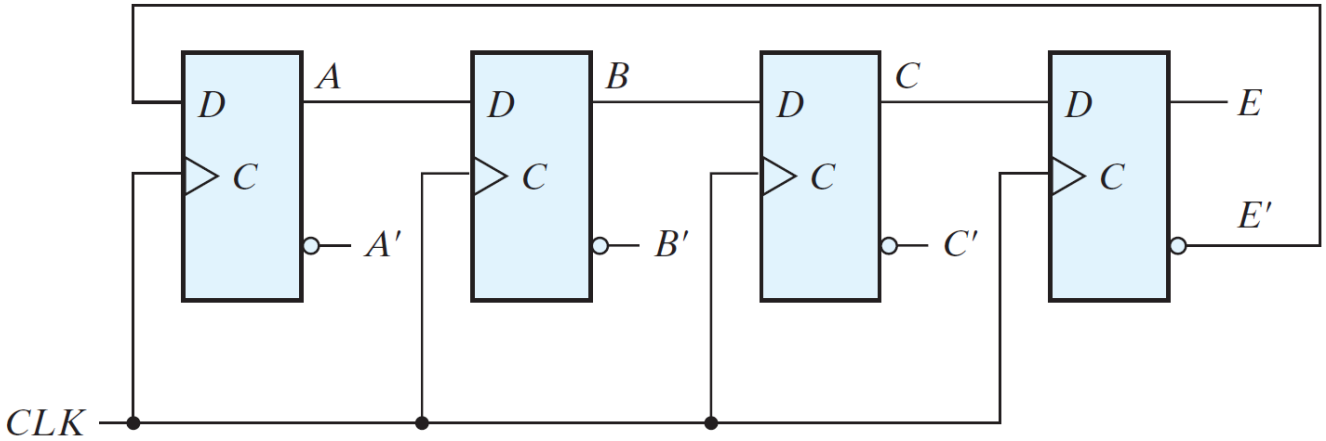
(c) Counter and decoder

Johnson Counter

- A *k* -bit ring counter circulates a single bit among the flip-flops to provide *k* distinguishable states.
- The number of states can be doubled if the shift register is connected as a *switch-tail* ring counter.
- A switch-tail ring counter is a circular shift register with the complemented output of the last flip-flop connected to the input of the first flip-flop.
- The circular connection is made from the complemented output of the rightmost flip-flop to the input of the leftmost flip-flop.
- The register shifts its contents once to the right with every clock pulse, and at the same time, the complemented value of the *last* flip-flop is transferred into the *first* flip-flop.

Johnson Counter

- In general, a *k*-bit switch-tail ring counter will go through a sequence of 2*k* states. Starting from all 0's, each shift operation inserts 1's from the left until the register is filled with all 1's. In the next sequences, 0's are inserted from the left until the register is again filled with all 0's.



Sequence number	Flip-flop outputs			
	A	*B*	*C*	*E*
1	0	0	0	0
2	1	0	0	0
3	1	1	0	0
4	1	1	1	0
5	1	1	1	1
6	0	1	1	1
7	0	0	1	1
8	0	0	0	1