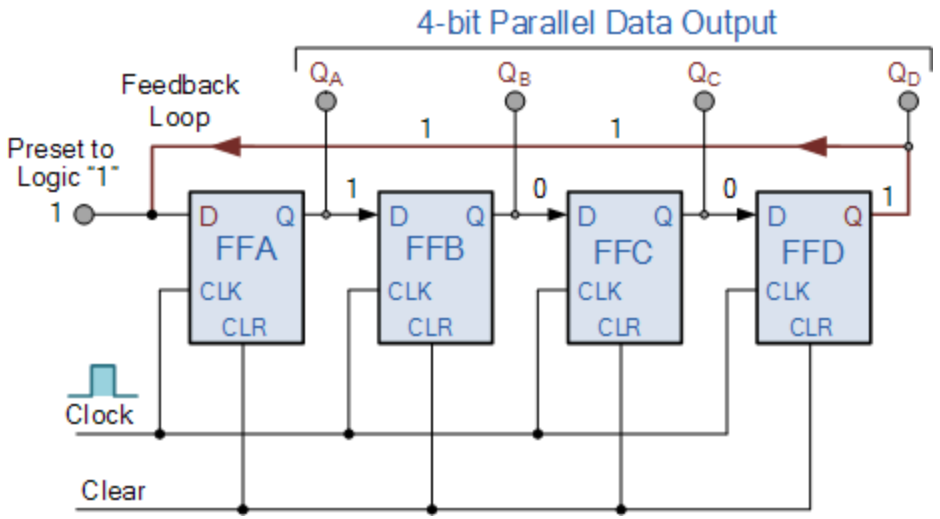
EE2000 Logic Circuit Design

Lecture 10 – Sequential Functional Blocks: Registers and Counters



Example (Counter)

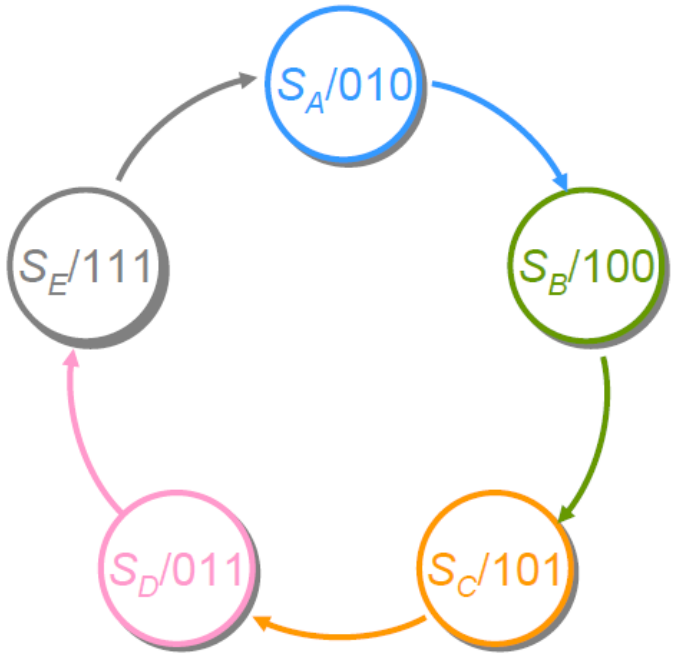
Design a free-running synchronous counter with the counting sequence of 2, 4, 5, 3, 7, 2, 4, 5, … (repetitive) using D flip-flops

Step 1:

- Input: Free-running → no input
- States: 5 states → 3 flip-flops
- Largest number 7 → $(111)_2$ → 3 outputs
- Behavior: 010 → 100 → 101 → 011 → 111...

Example (Counter)

STEP 2: Work out the State Diagram and Table



PS	NS	Output $Z_0 Z_1 Z_2$
S_A	S_B	0 1 0
S_B	S_C	1 0 0
S_C	S_D	1 0 1
S_D	S_E	0 1 1
S_E	S_A	1 1 1

- S_A = 010, S_B = 100, S_C = 101, S_D = 011, S_E = 111

Example (Counter)

STEP 3: Assign FFs

5 states → 3 FFs

PS	NS	Output $Z_0 Z_1 Z_2$
S_A	S_B	0 1 0
S_B	S_C	1 0 0
S_C	S_D	1 0 1
S_D	S_E	0 1 1
S_E	S_A	1 1 1

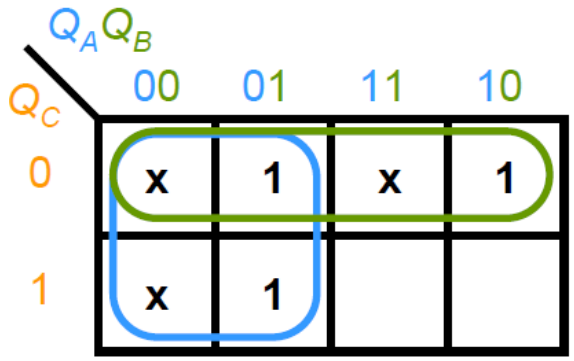
PS			NS			Output
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	$Z_0 Z_1 Z_2$
0	0	0	x	x	x	x x x
0	0	1	x	x	x	x x x
0	1	0	1	0	0	0 1 0
0	1	1	1	1	1	0 1 1
1	0	0	1	0	1	1 0 0
1	0	1	0	1	1	1 0 1
1	1	0	x	x	x	x x x
1	1	1	0	1	0	1 1 1

- $S_A = 010$, $S_B = 100$, $S_C = 101$, $S_D = 011$, $S_E = 111$

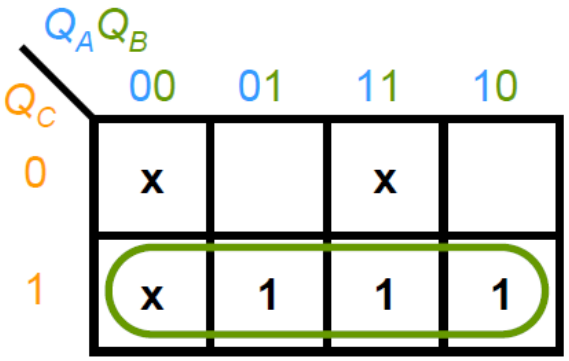
Example (Counter)

STEP 4: Work out D_A, D_B, D_C

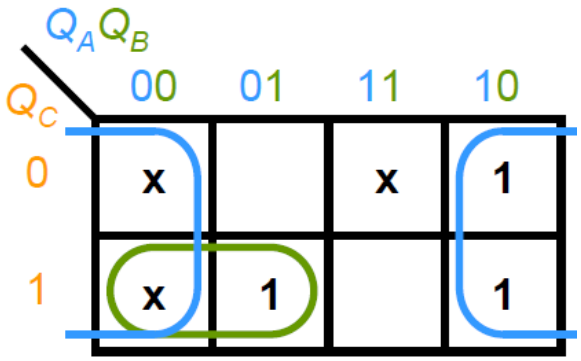
PS			NS			Output
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	$Z_0\ Z_1\ Z_2$
0	0	0	x	x	x	x x x
0	0	1	x	x	x	x x x
0	1	0	1	0	0	0 1 0
0	1	1	1	1	1	0 1 1
1	0	0	1	0	1	1 0 0
1	0	1	0	1	1	1 0 1
1	1	0	x	x	x	x x x
1	1	1	0	1	0	1 1 1



$D_A = Q_A' + Q_C'$



$D_B = Q_C$



$D_C = Q_B' + Q_A'Q_C$

Example (Counter)

STEP 5: Work out z

PS			NS			Output
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	$Z_0 Z_1 Z_2$
0	0	0	x	x	x	x x x
0	0	1	x	x	x	x x x
0	1	0	1	0	0	0 1 0
0	1	1	1	1	1	0 1 1
1	0	0	1	0	1	1 0 0
1	0	1	0	1	1	1 0 1
1	1	0	x	x	x	x x x
1	1	1	0	1	0	1 1 1

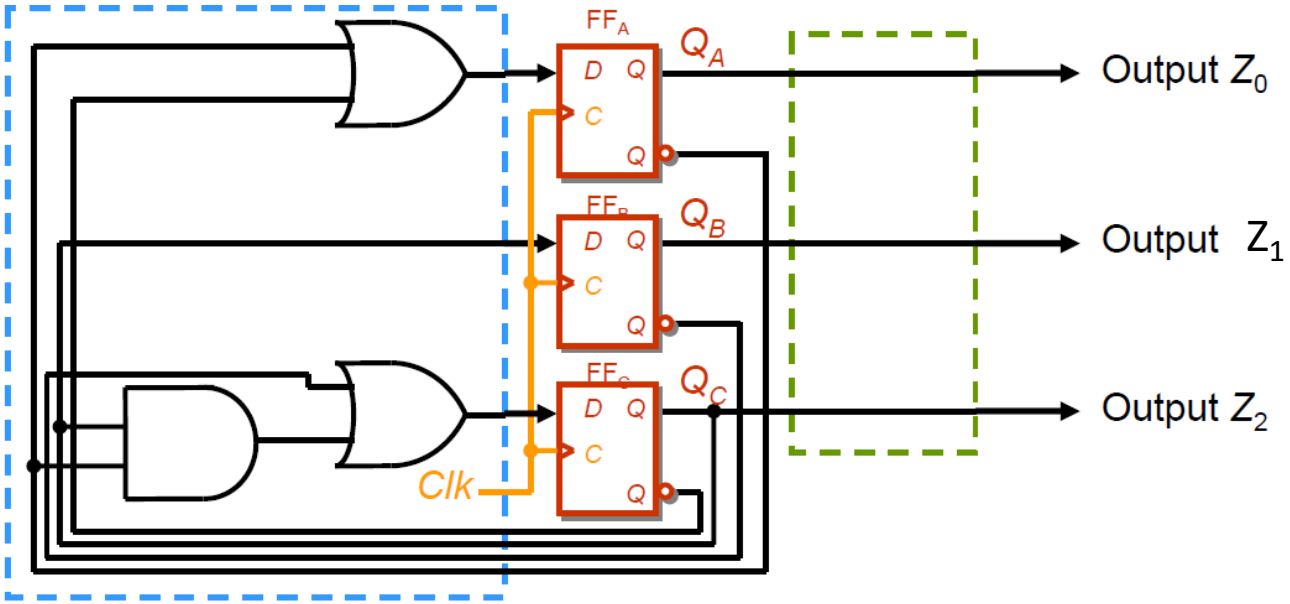
$Z_0 = Q_A$

$Z_1 = Q_B$

$Z_2 = Q_C$

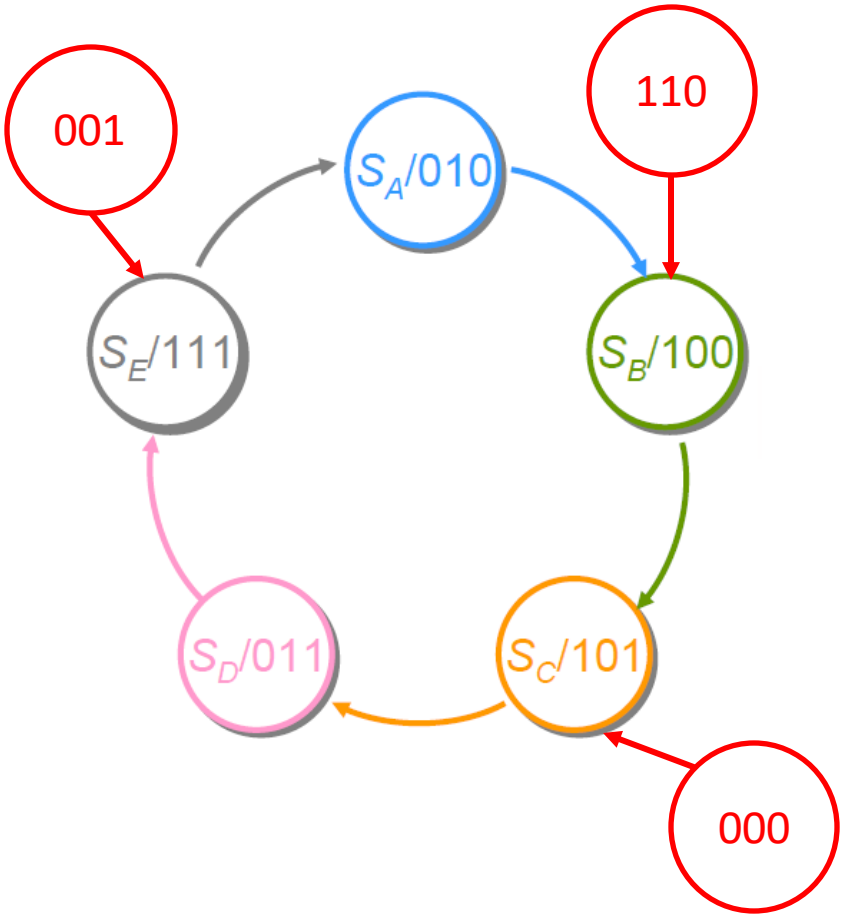
Example (Counter)

$$D_A(Q_A, Q_B, Q_C) = Q_A' + Q_C'$$
$$D_B(Q_A, Q_B, Q_C) = Q_C$$
$$D_C(Q_A, Q_B, Q_C) = Q_B' + Q_A' Q_C$$



Is It Self Starting?

Self starting: The counter will enter the main loop regardless on the initial state



$D_A = Q_A' + Q_C'$

$D_B = Q_C$

$D_C = Q_B' + Q_A'Q_C$

Present State	Next State
000	101
001	111
110	100

Exercise

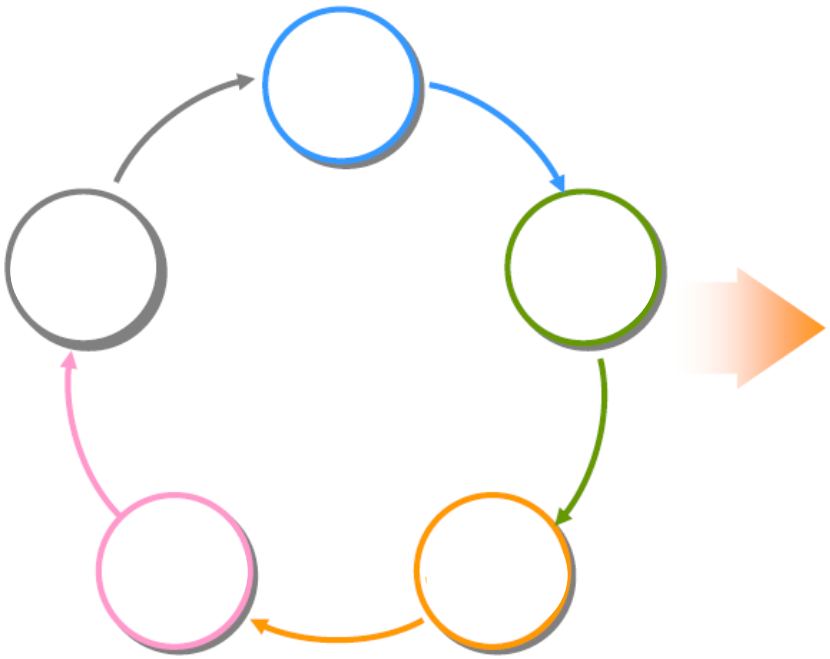
Use T FFs and logic gates to design a synchronous counter with counting sequence 0, 4, 2, 1, 6…

Step 1:

- States
- Largest number
- Behavior

Exercise

STEP 2: Work out the State Diagram and Table



Present State			Next State		
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Exercise

STEP 3: Work out T_A, T_B, T_C

Present State			Next State			Flip-Flop		
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	T_A	T_B	T_C
0	0	0	1	0	0			
0	0	1	1	1	0			
0	1	0	0	0	1			
0	1	1	x	x	x			
1	0	0	0	1	0			
1	0	1	x	x	x			
1	1	0	0	0	0			
1	1	1	x	x	x			

T_A Q_AQ_B $T_A =$

Q_C	00	01	11	10
0				
1				

T_B Q_AQ_B $T_B =$

Q_C	00	01	11	10
0				
1				

T_C Q_AQ_B $T_C =$

Q_C	00	01	11	10
0				
1				

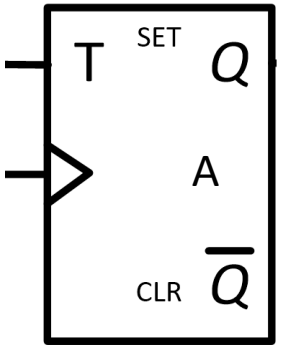
Exercise

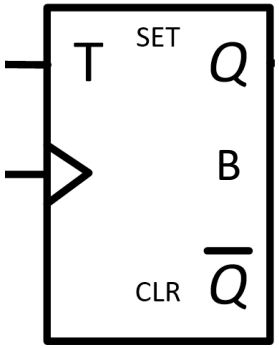
$$T_A = Q'_B + Q_A$$

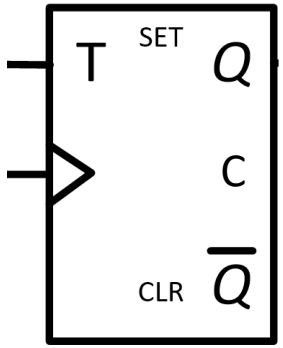
$$T_B = Q_A + Q_B + Q_C$$

STEP 4: Draw the Circuit

$$T_C = Q'_A Q_B + Q_C = (Q_A + Q'_B)' + Q_C$$



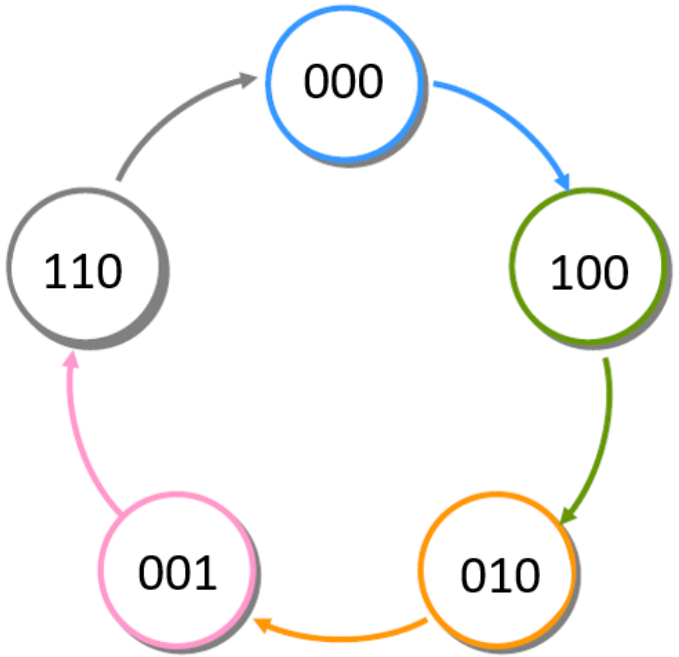




Exercise

Determine the state transition of remaining states

Present State			Next State			Flip-Flop		
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	T_A	T_B	T_C
0	0	0	1	0	0	1	0	0
0	0	1	1	1	0	1	1	1
0	1	0	0	0	1	0	1	1
0	1	1						
1	0	0	0	1	0	1	1	0
1	0	1						
1	1	0	0	0	0	1	1	0
1	1	1						



$$T_A = Q'_B + Q_A$$

$$T_B = Q_A + Q_B + Q_C$$

$$T_C = Q'_A Q_B + Q_C$$

What will you learn?

10.1 Learn various Registers

- Parallel load
- Bidirectional shift
- Example: Serial adder

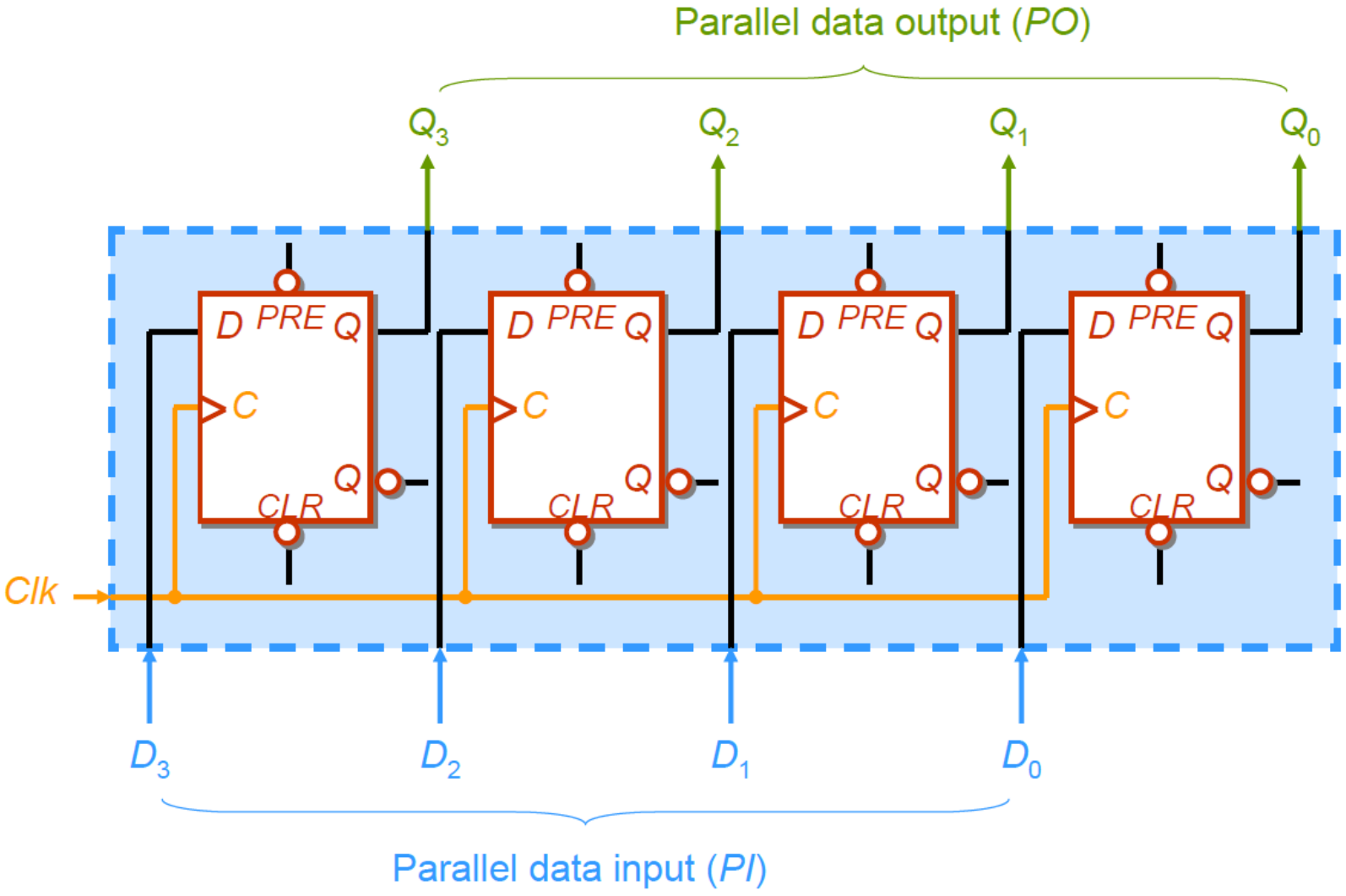
10.2 Learn various Counters

- Asynchronous: Ripple
- Synchronous: Up-down, Modulo, Ring

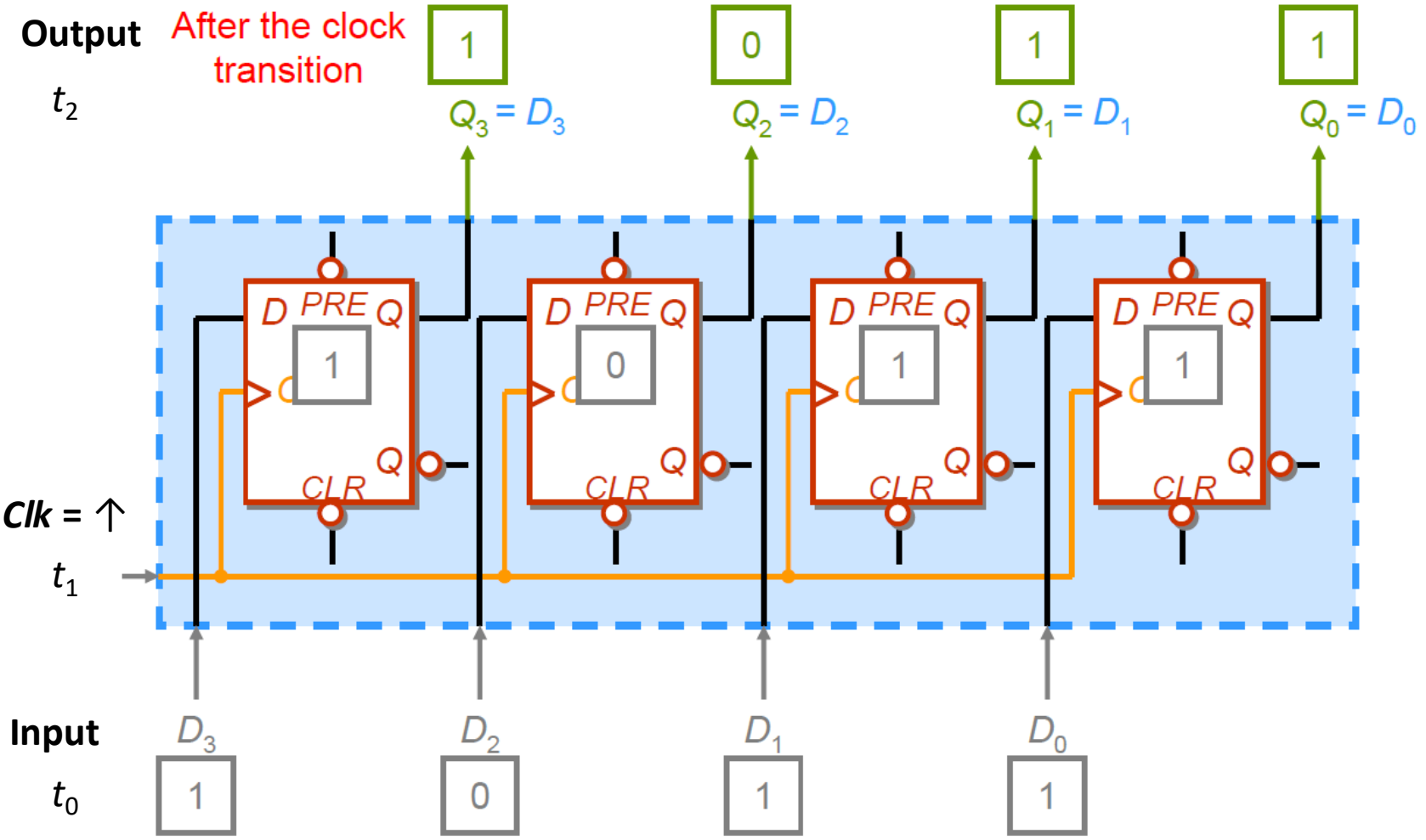
10.1 Registers

- Memory used to store binary data
- A collection of flip-flops: To store ***n***-bit data, a register requires ***n*** flip-flops
- ***n***-bit register
  - ***n*** edge-triggered flip-flops
  - Clock pulse as the enable signal
- Parallel or serial design

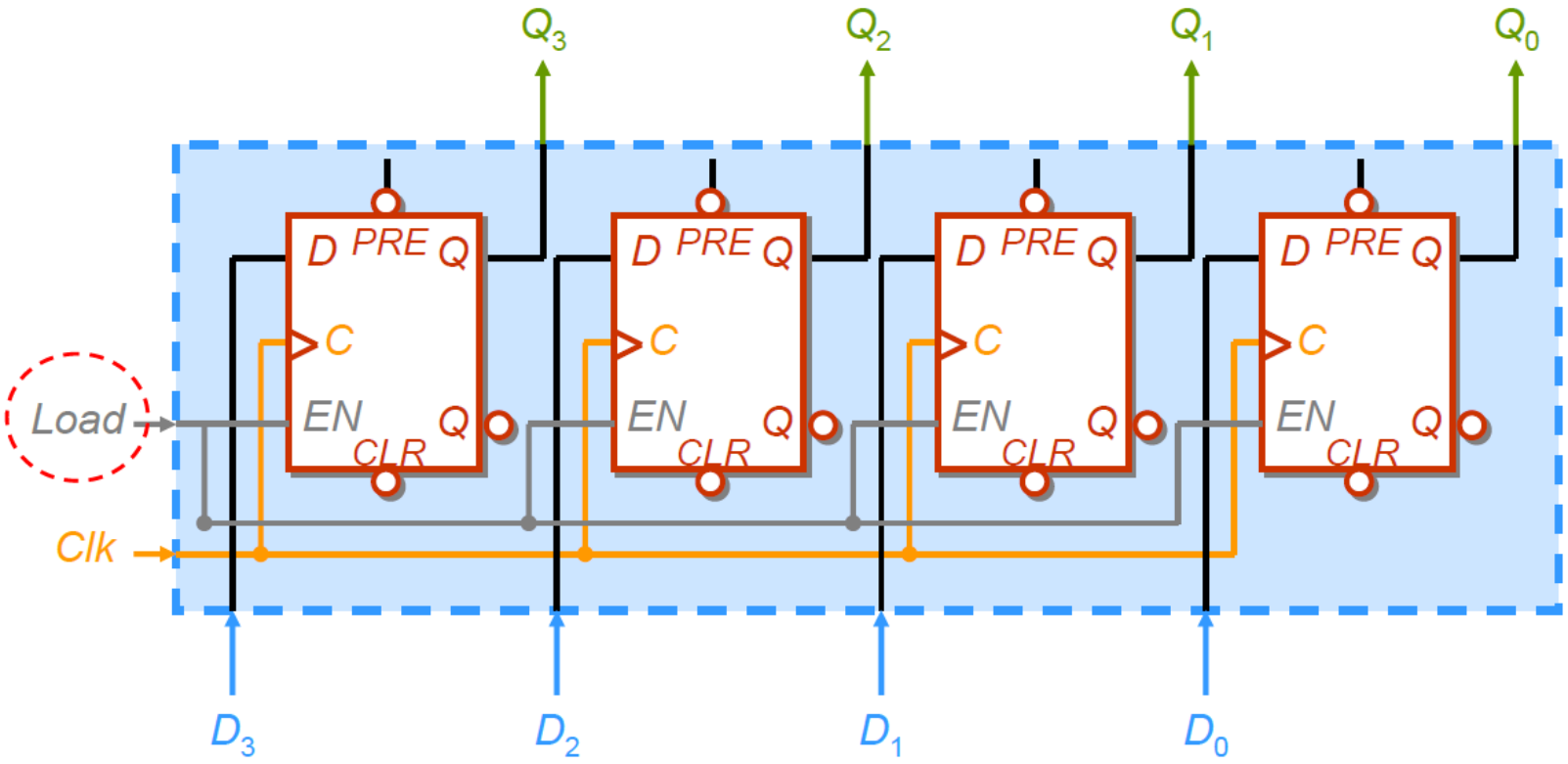
Parallel-in Parallel-out Register



Example

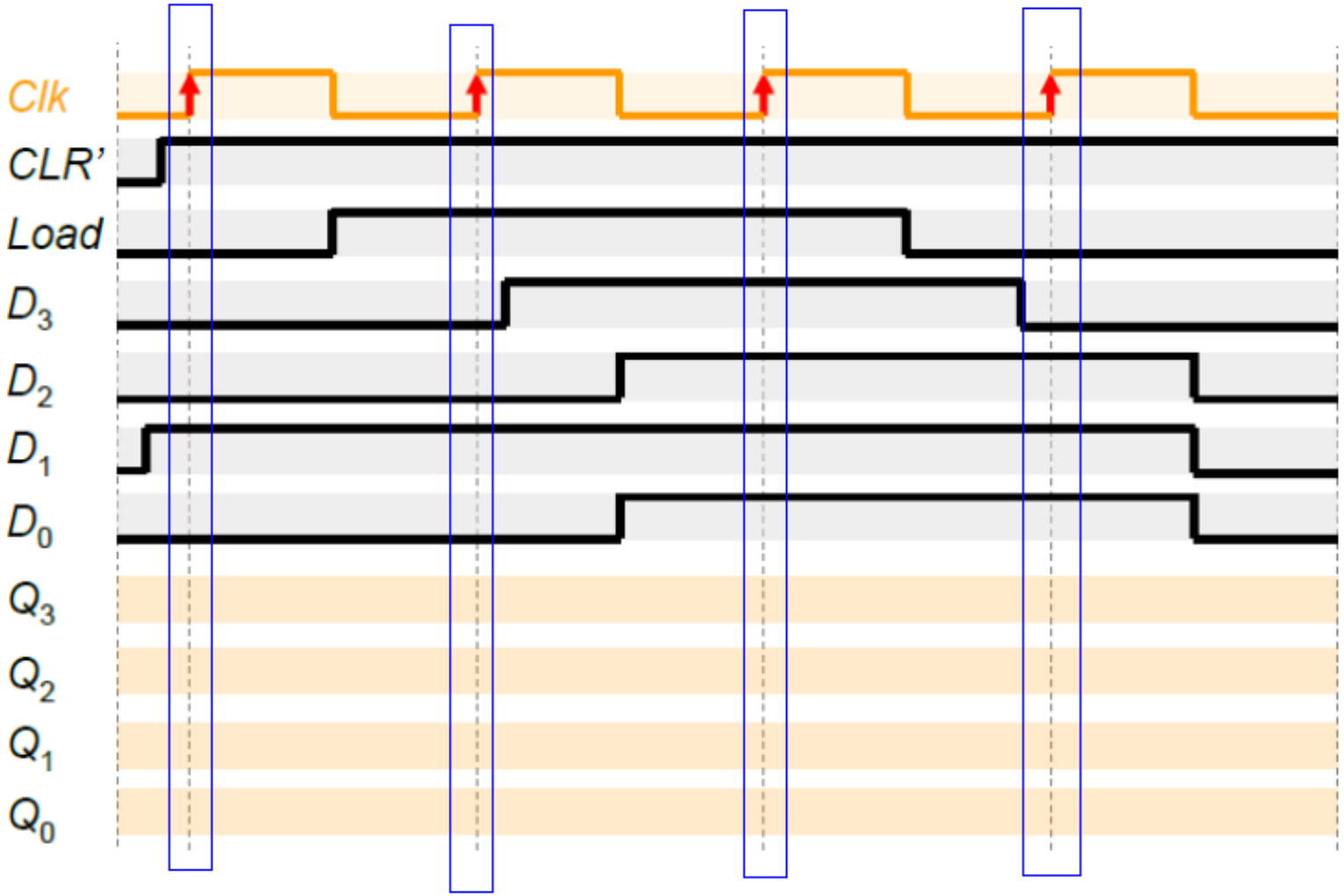

Output
t_2
After the clock transition
1
0
1
1
$Q_3 = D_3$
$Q_2 = D_2$
$Q_1 = D_1$
$Q_0 = D_0$
D PRE Q
1
CLR Q
D PRE Q
0
CLR Q
D PRE Q
1
CLR Q
D PRE Q
1
CLR Q
Clk = ↑
t_1
Input
t_0
D_3
1
D_2
0
D_1
1
D_0
1

Register with Parallel Load

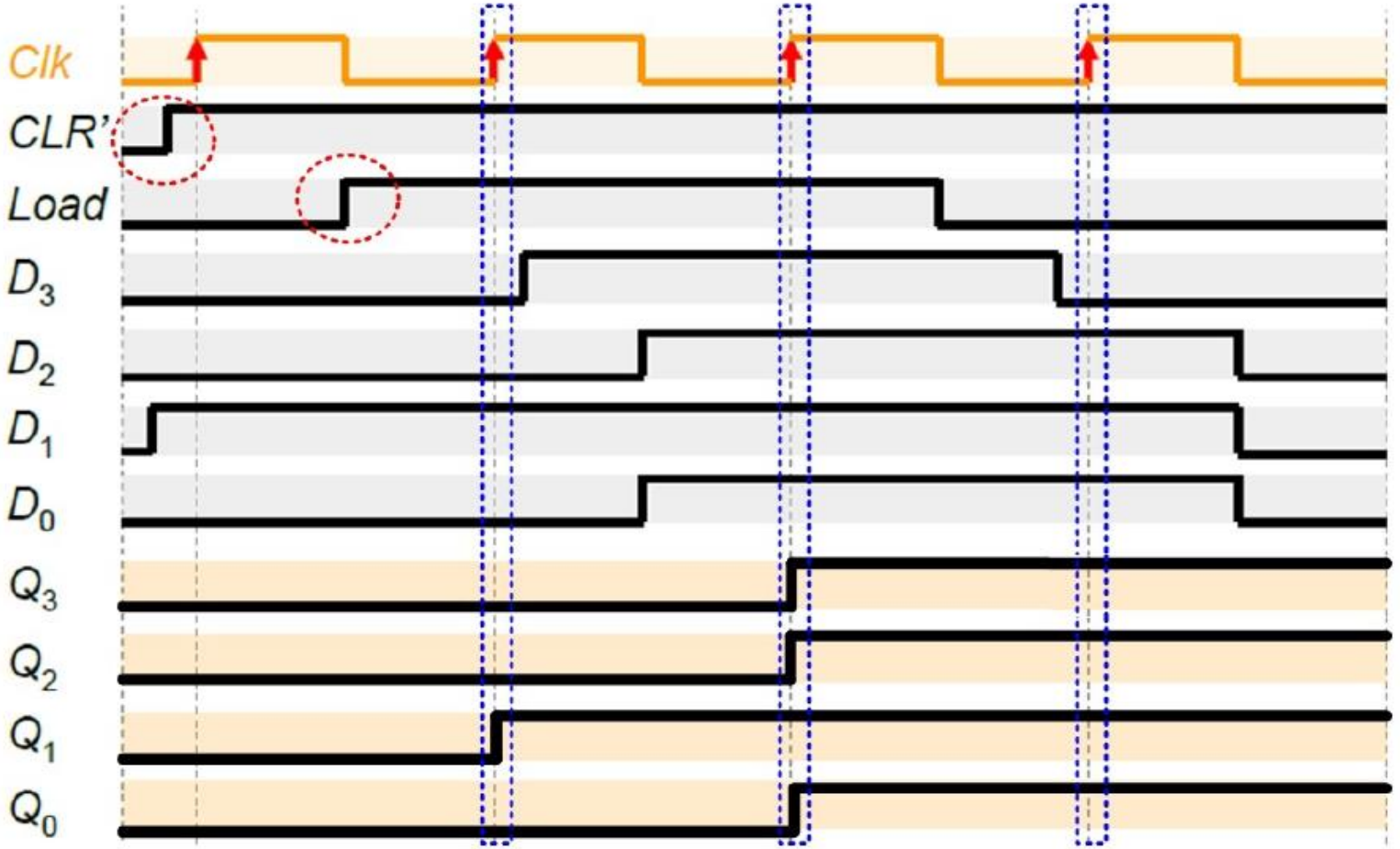


The register will load the inputs into the flip-flops only when *Clk* is at the rising edge AND *Load* = 1

Timing Diagram

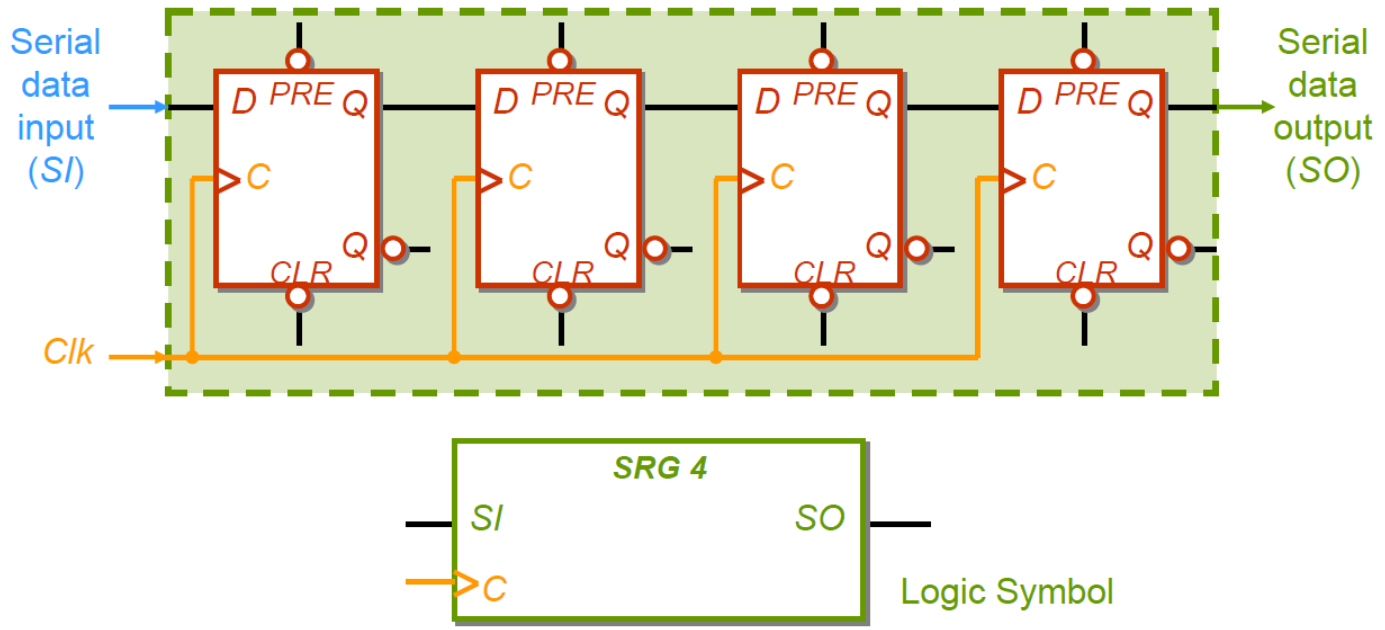


Timing Diagram

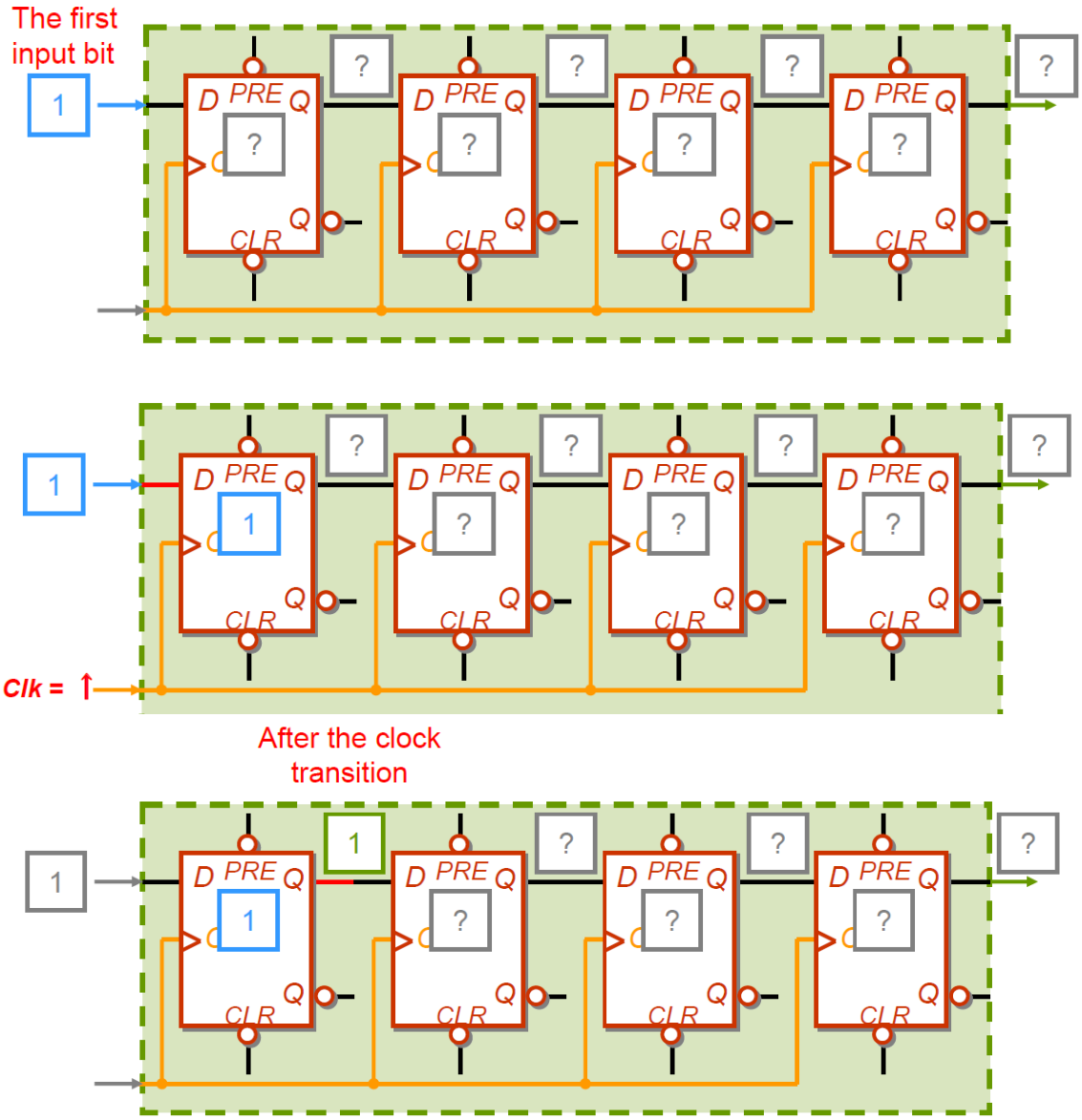


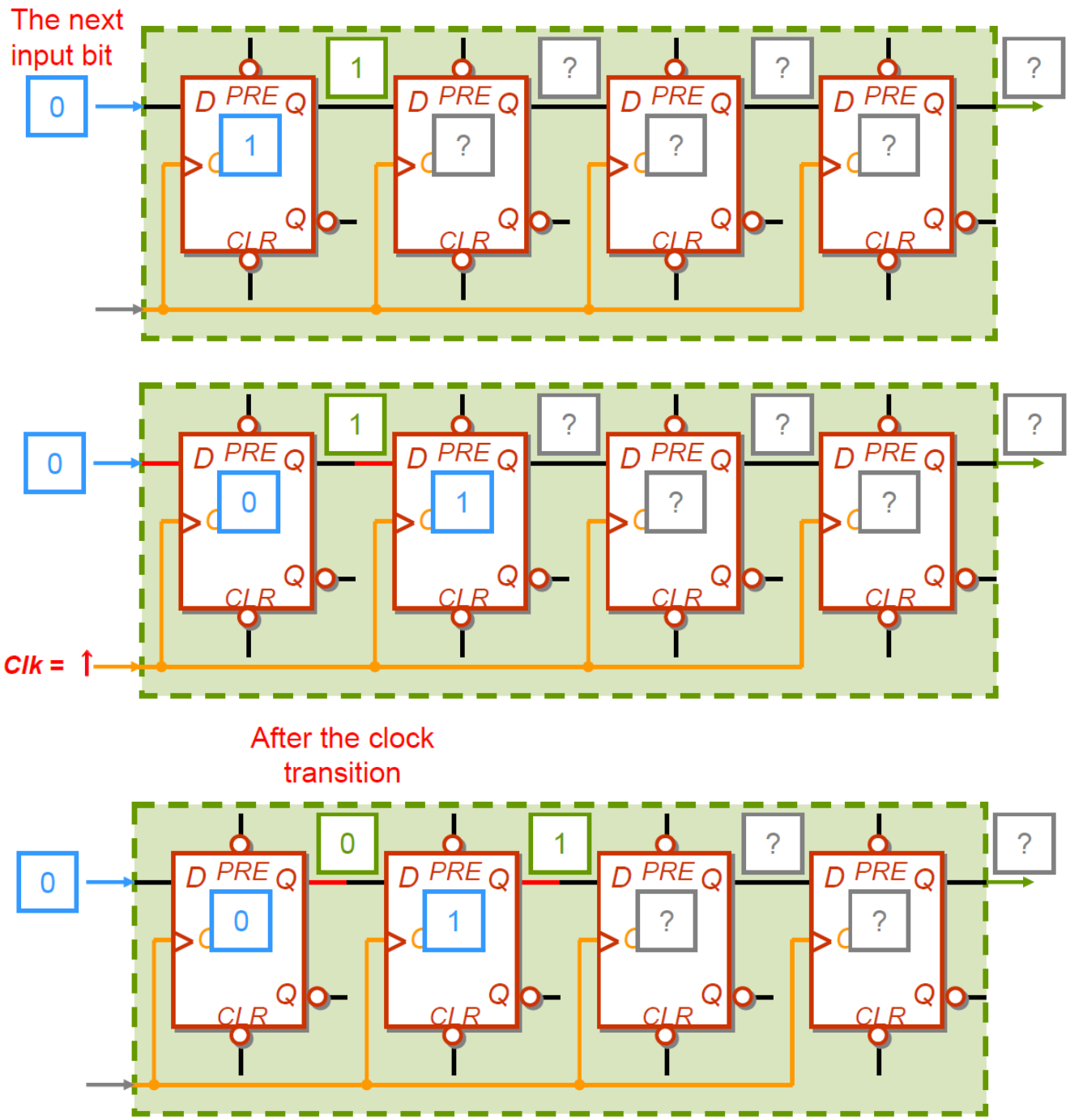
Shift Register

- Serial-in serial-out
- Data move (shift) to the next flip-flop on each clock



How Does It Work?

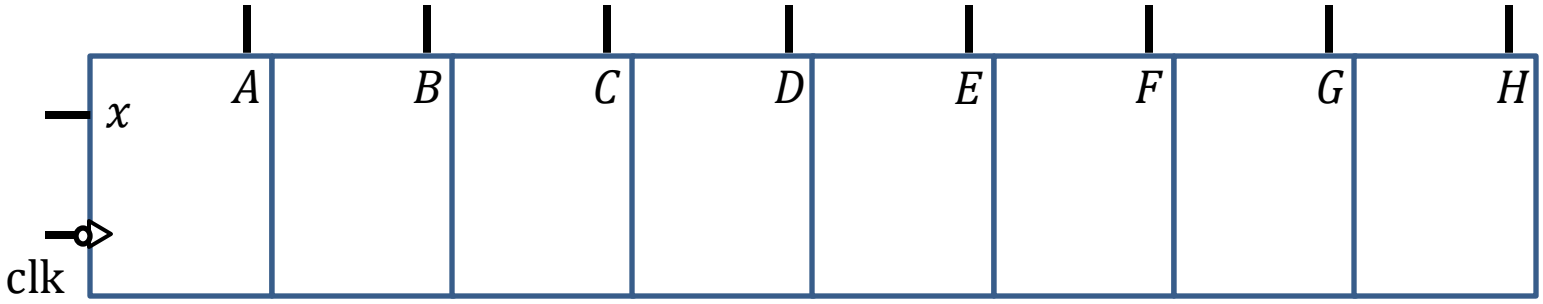



The next input bit
0
D PRE Q
1
CLR Q
1
?
Clk =
After the clock transition
0

Exercise

Given an 8-bit serial-in parallel-out shift register as shown below, design a system with one output z, which is 1 only if the input x has been alternating for seven clock times (including the present one).

z?

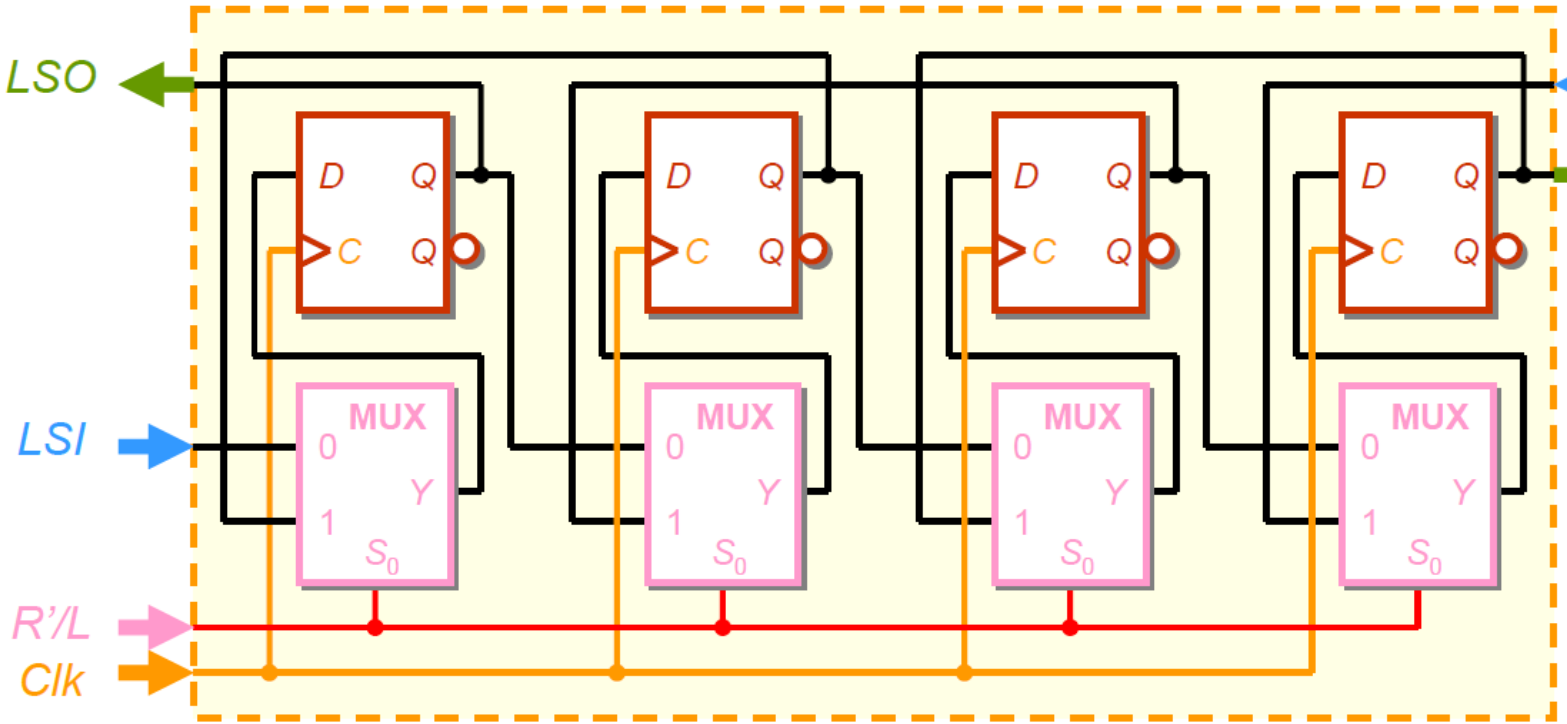


Characteristics

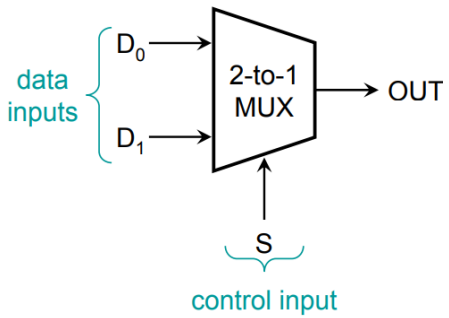
- Shift register has a chain of flip-flops
- When the FFs are triggered, the stored bits will be shifted to the next FFs
- First-in-First-out structure
- Shift register with two shifting directions?

  → **Bidirectional shift register!**

Bidirectional Shift Register

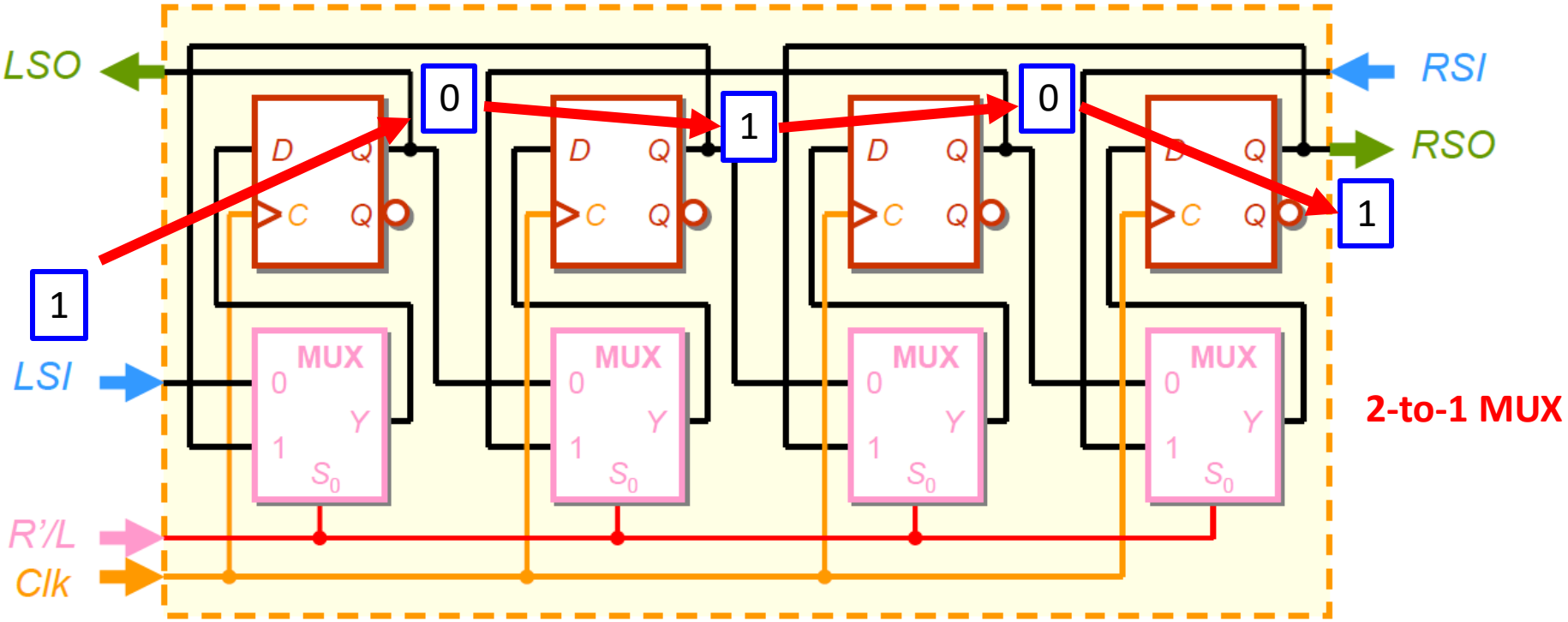


2-to-1 MUX



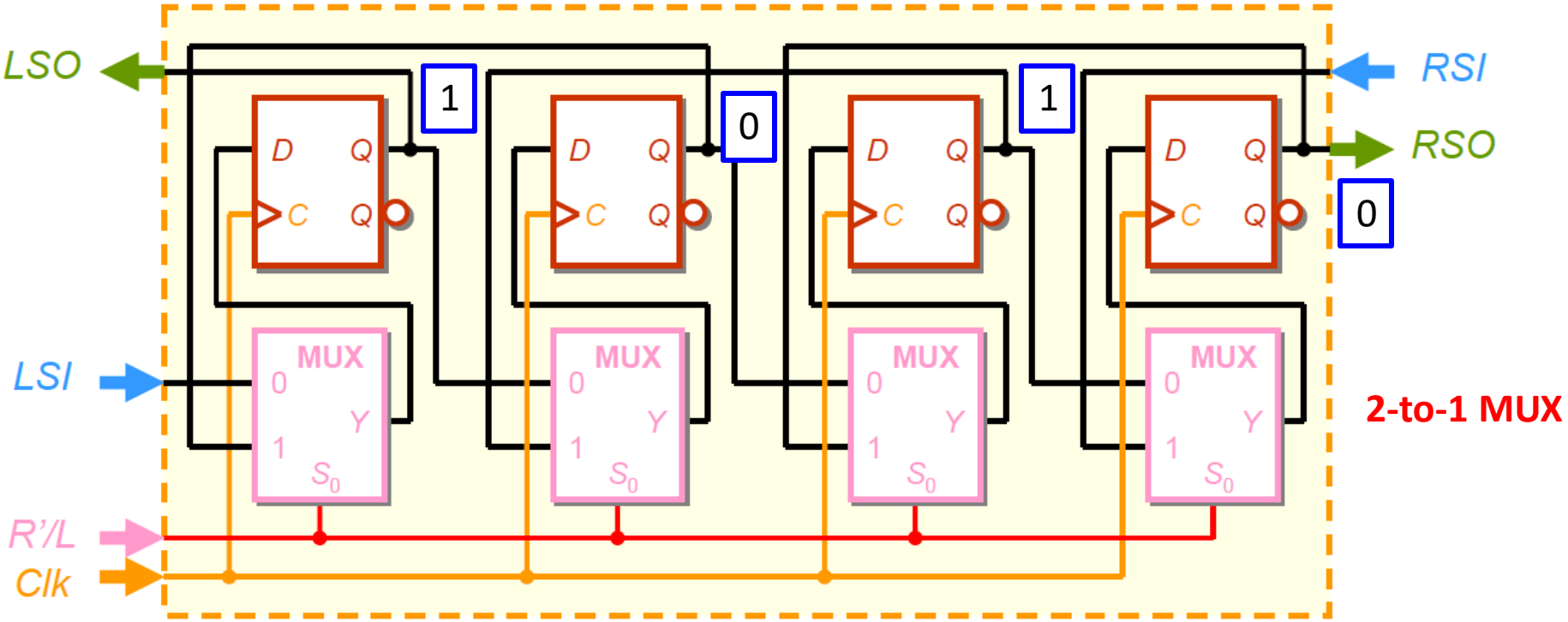
Input			Output
S	D_1	D_0	
0	x	x	D_0
1	x	x	D_1

How Does It Work?



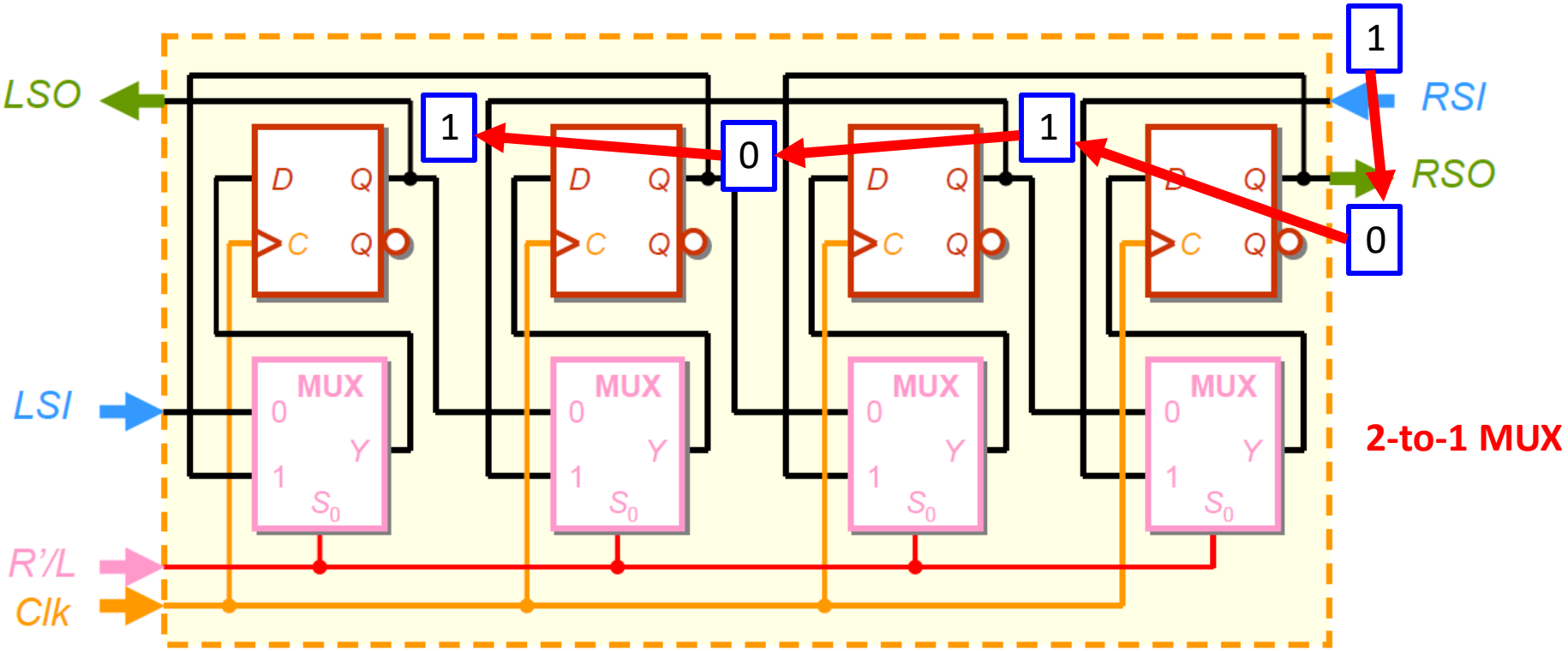
- Assume these are the initial states
- When *R*' = 0 (*L* = 0) and upon Clk ↑, stored bits shift to right!

How Does It Work?



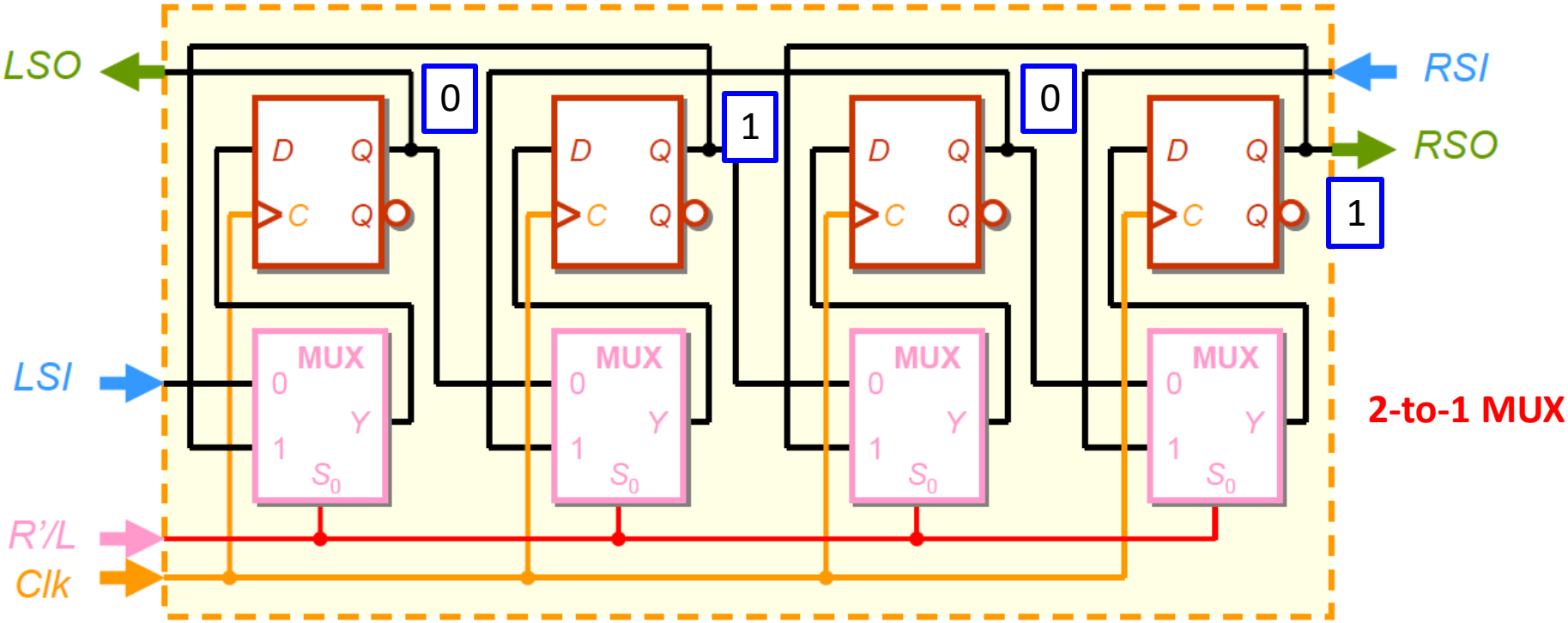
$Q_1Q_2Q_3Q_4$ => "0101" becomes "1010"

How Does It Work?



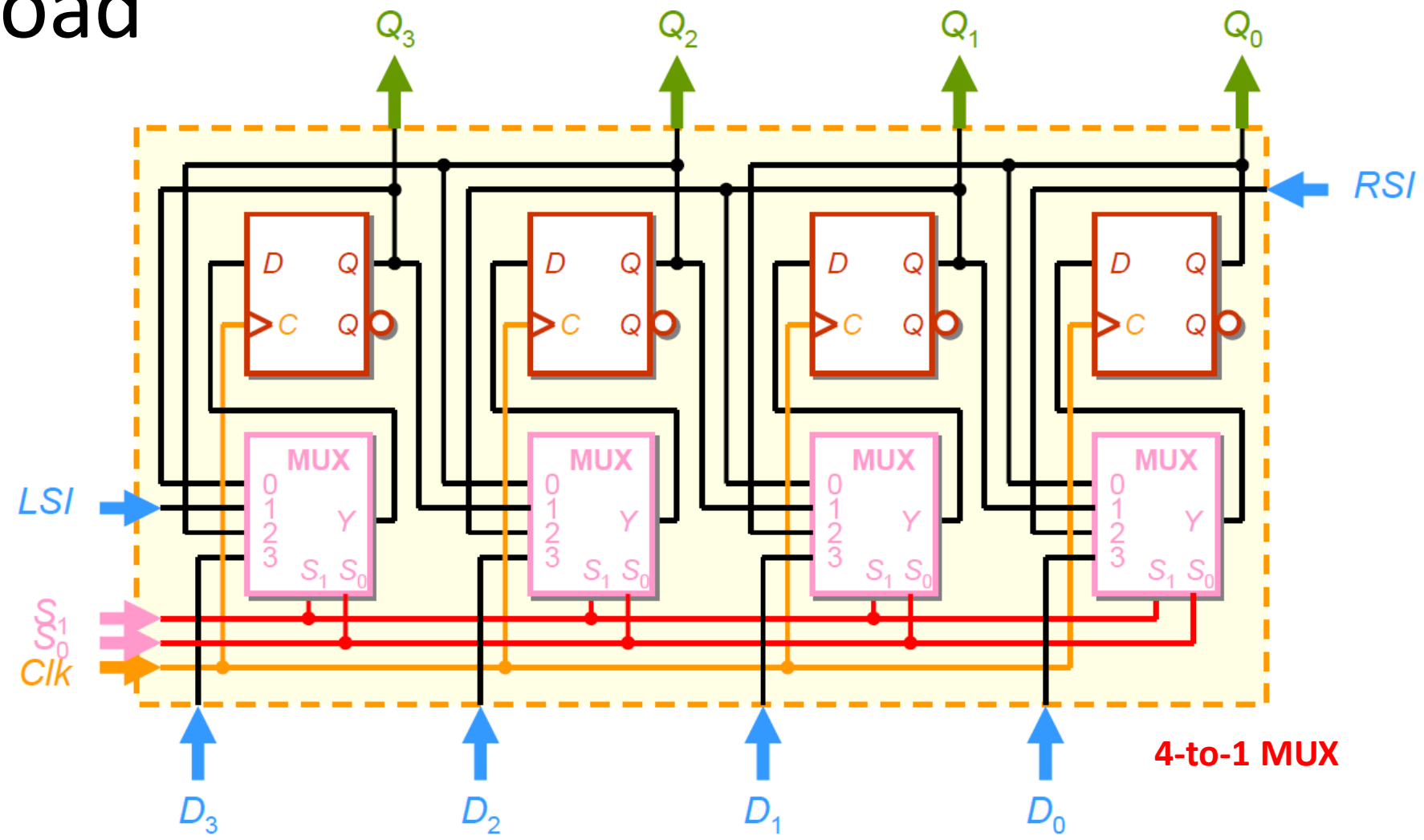
- Assume these are the initial states
- When *R*' = 1 (*L* = 1) and upon Clk ↑, stored bits shift to left!

How Does It Work?

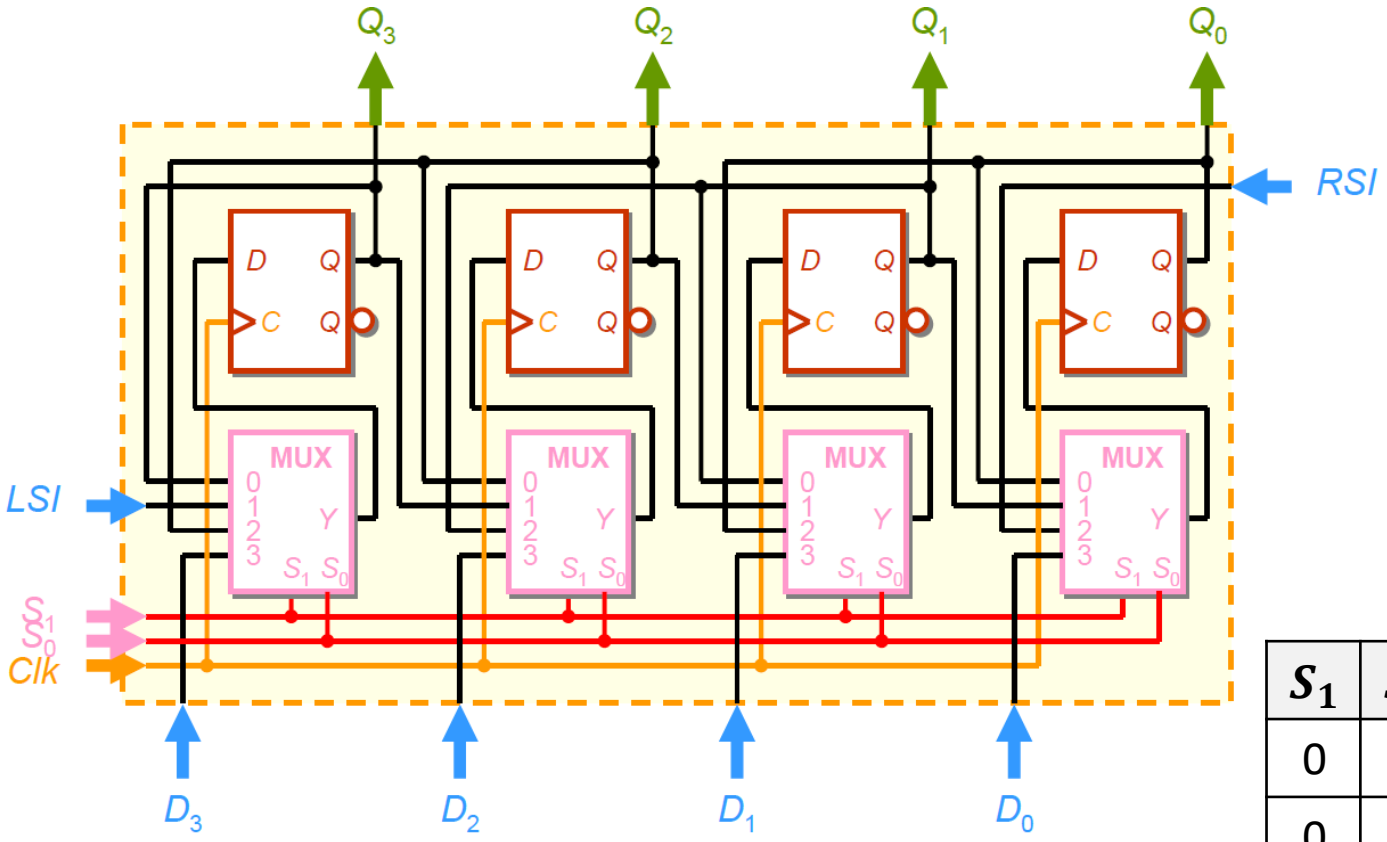


$Q_1Q_2Q_3Q_4$ =>“1010” becomes “0101”

Bidirectional Shift Register + Parallel Load



How Does It Work?

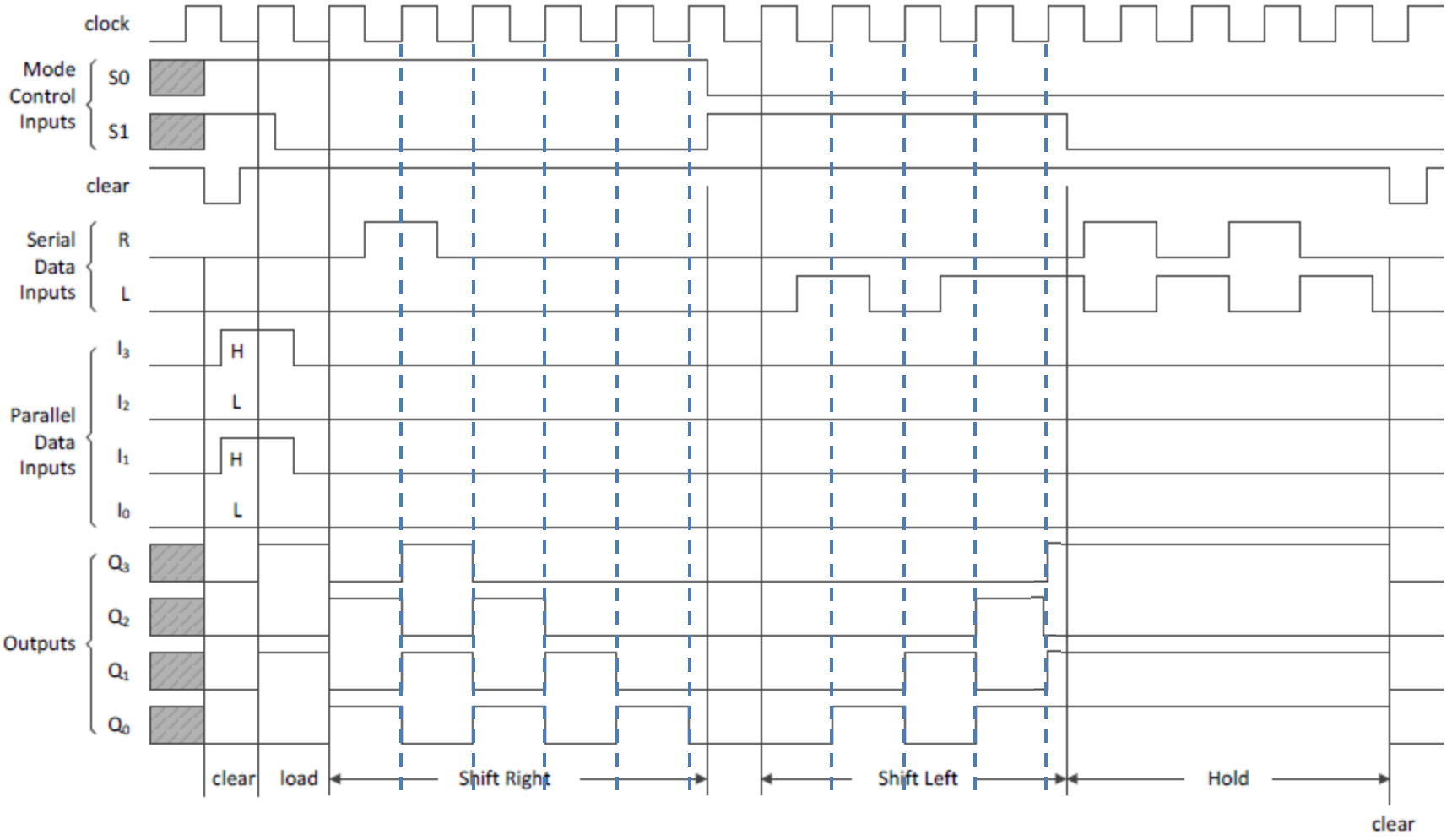


S_1	S_0	**Operation**
0	0	HOLD
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Timing Diagram

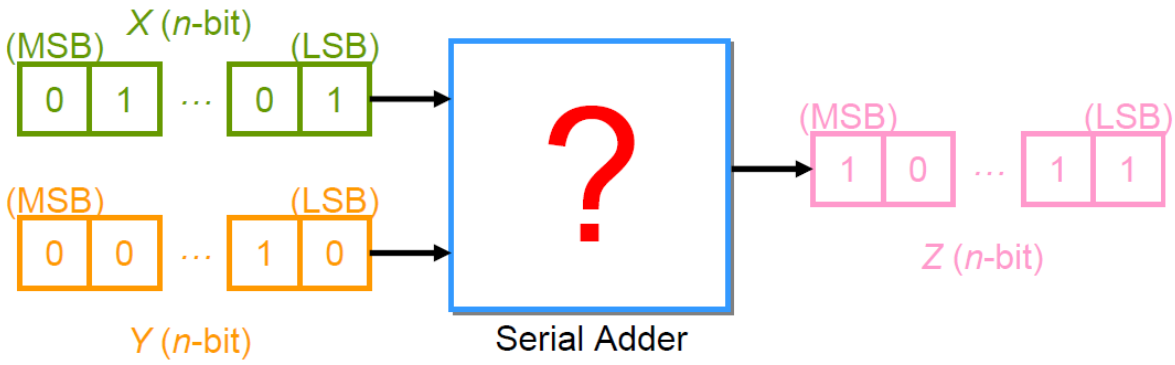
Shift right: $RQ_3Q_2Q_1$
Shift left: $Q_2Q_1Q_0L$

S_1	S_0	Operation
0	0	HOLD
0	1	Shift right
1	0	Shift left
1	1	Parallel load



Serial Binary Adder

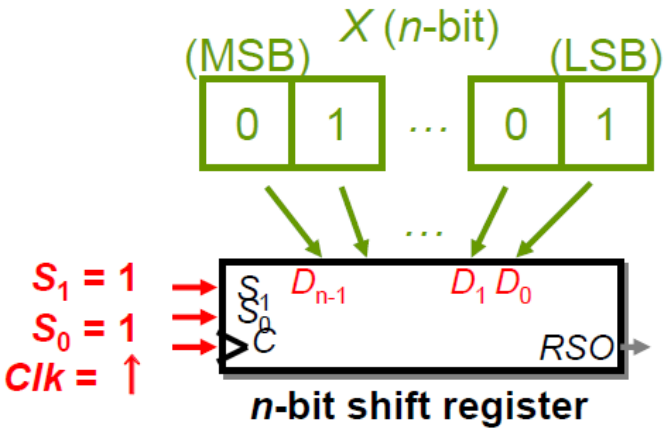
- A system that performs the addition of two binary numbers in serial form (bit by bit).

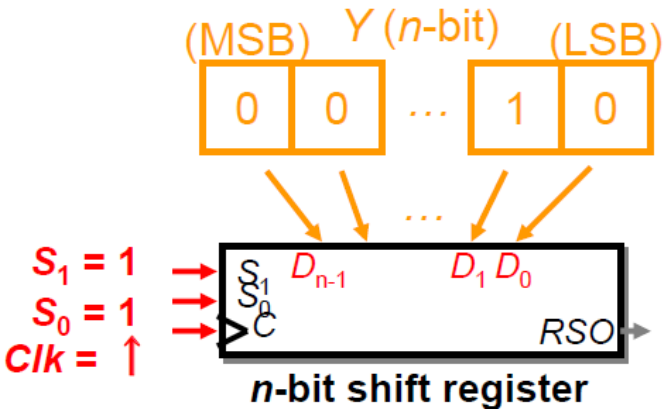


- What do we need?
  - 2 inputs: 2 shift registers
  - Perform addition: a Full Adder
  - Store Carry-out bit: a flip-flop
  - Store output: a shift register

How Does It Work?

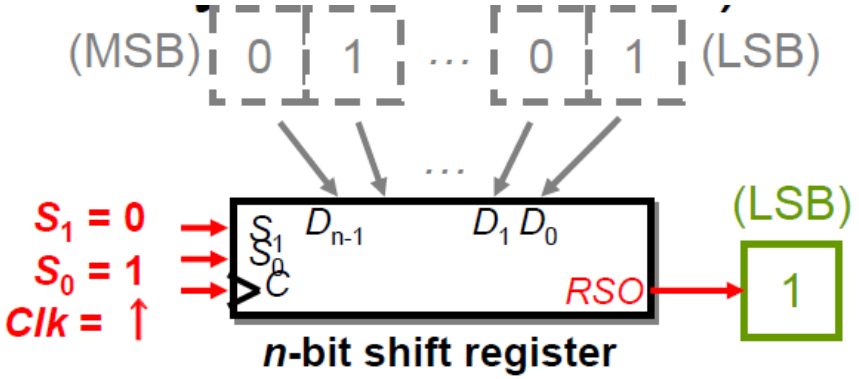
STEP 1: Parallel loading of the two binary number *X* and *Y* (***n***-bit numbers into ***n***-bit shift register)

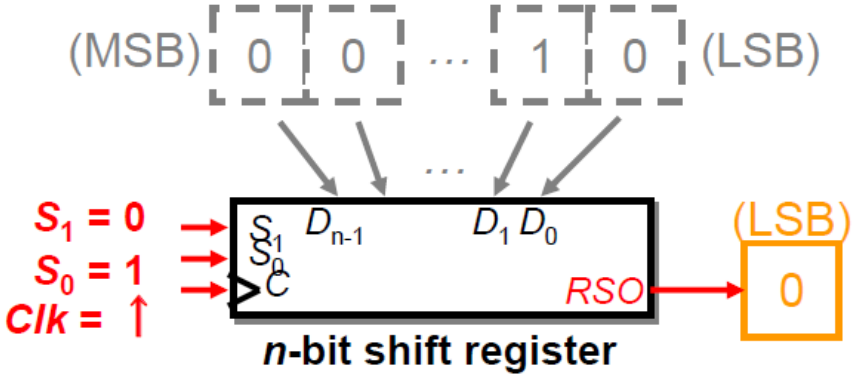




How Does It Work?

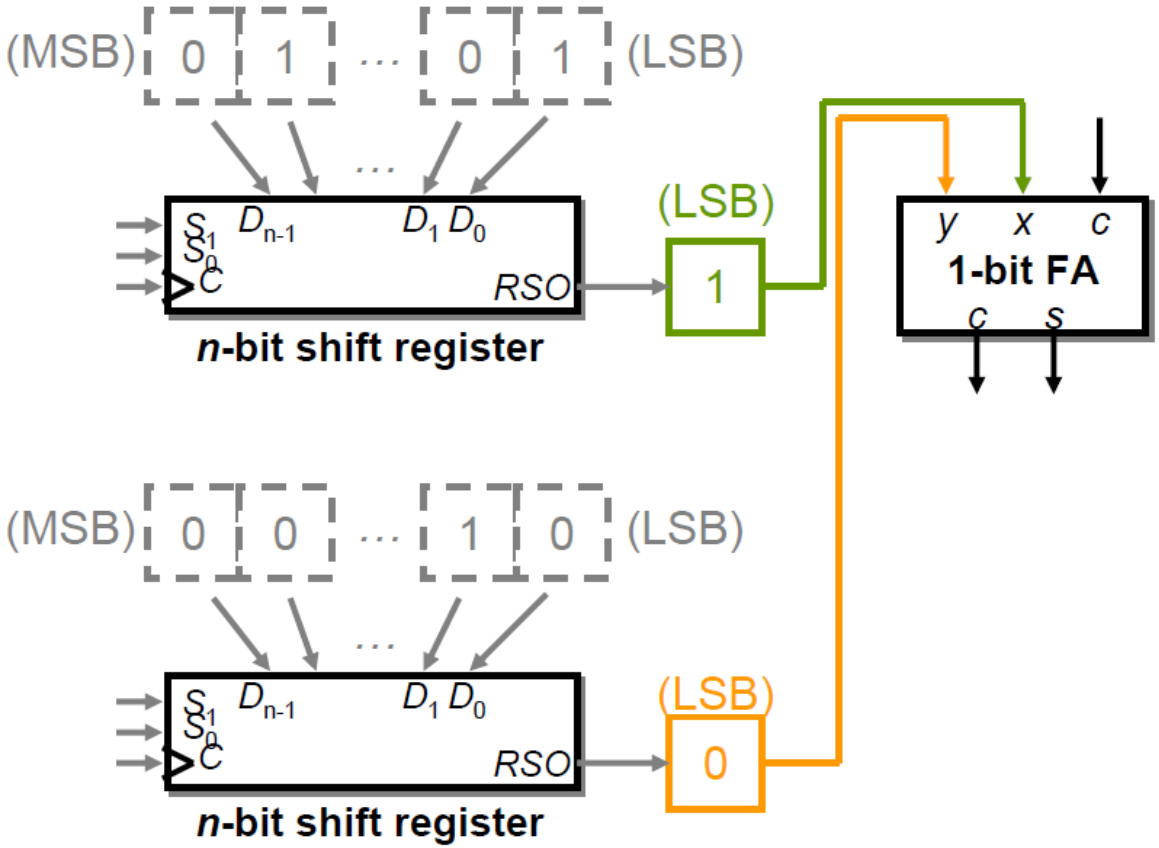
STEP 2: Shift out the binary number bit-by-bit, starting from LSB first





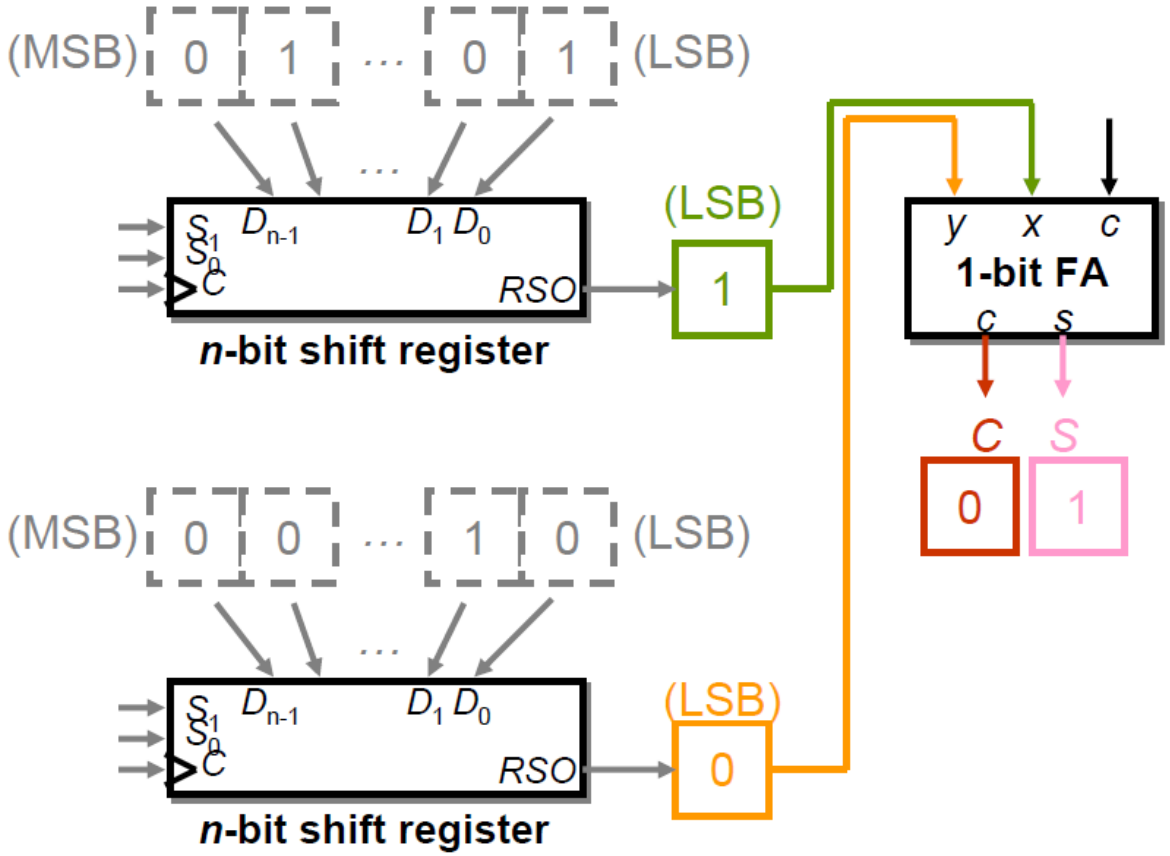
How Does It Work?

STEP 3: Forward the bits to full adder



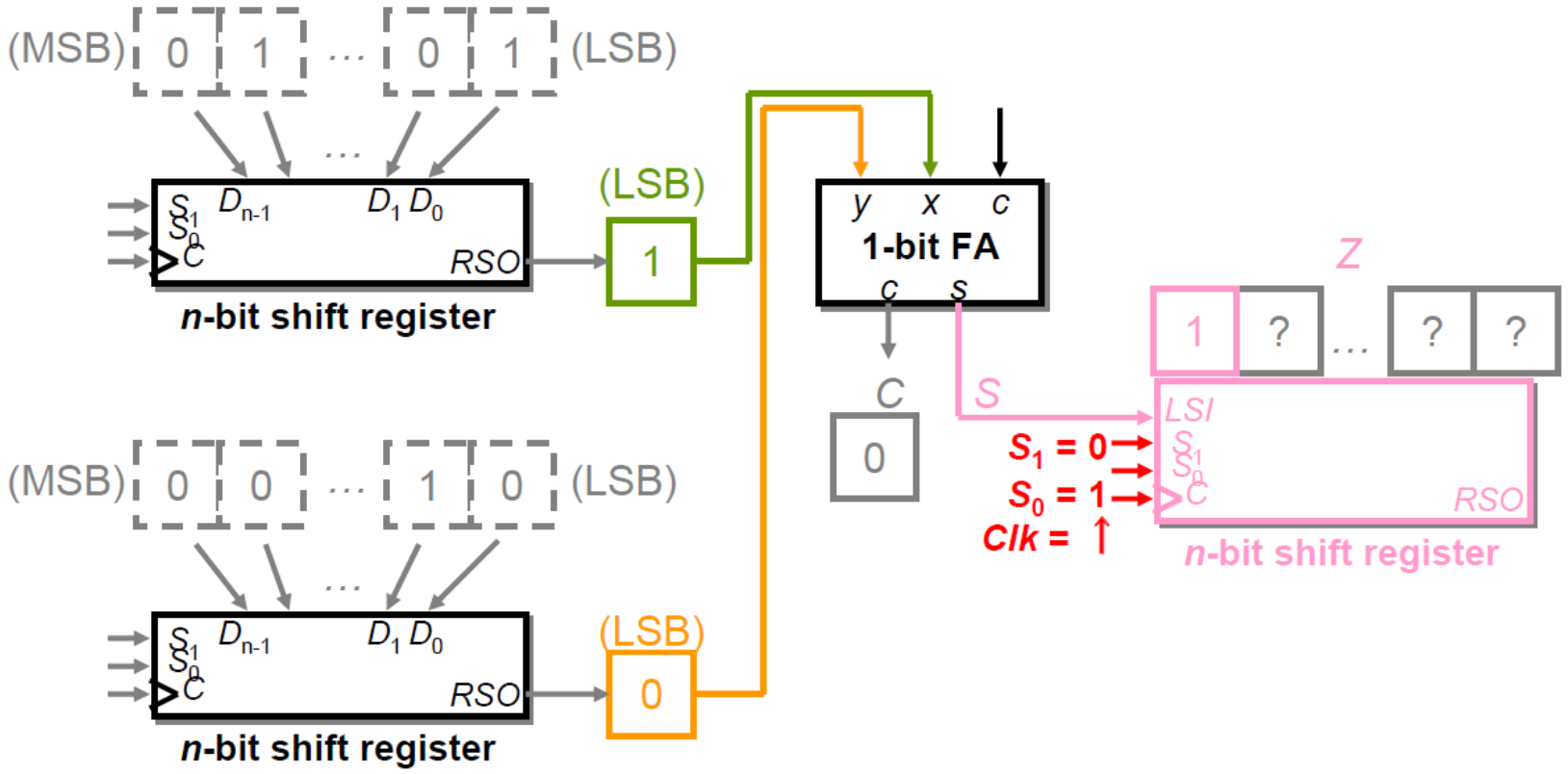
How Does It Work?

STEP 3: Perform one bit addition



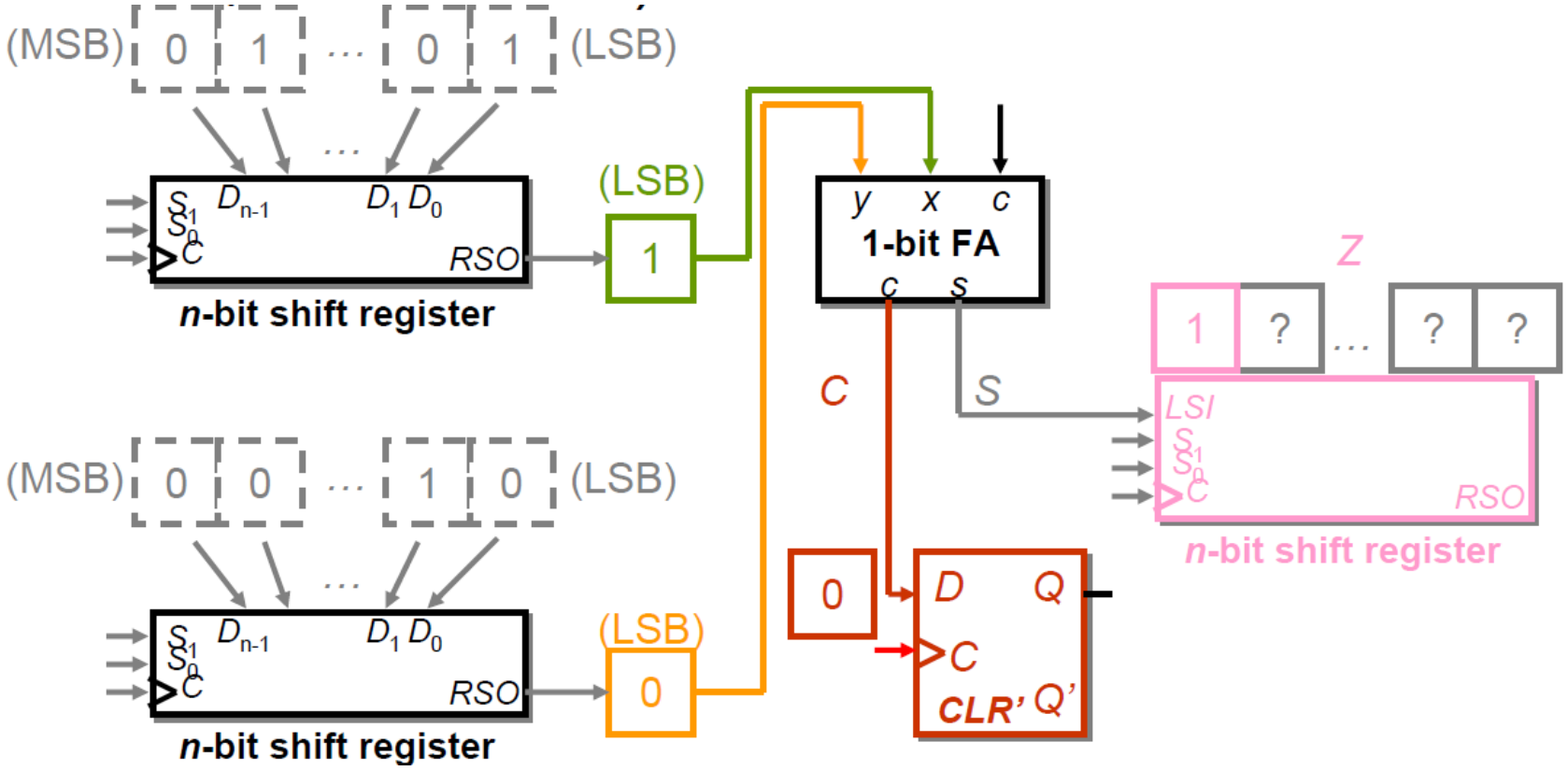
How Does It Work?

STEP 4: Feed the 1-bit output sum (*S*) to the output shift register (*Z*)



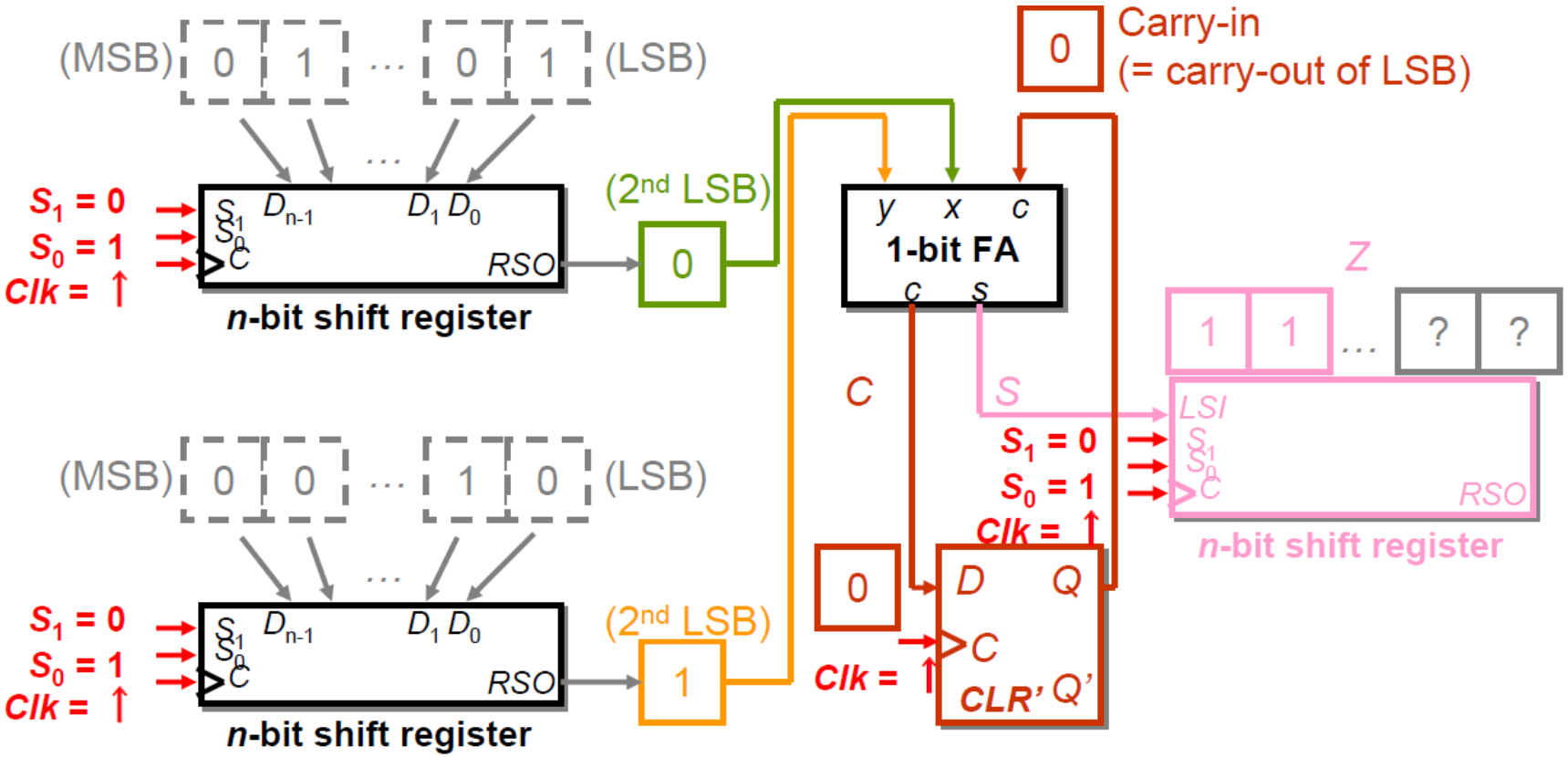
How Does It Work?

STEP 5: Store the 1-bit Carry-out (*C*) to a flip-flop for later use



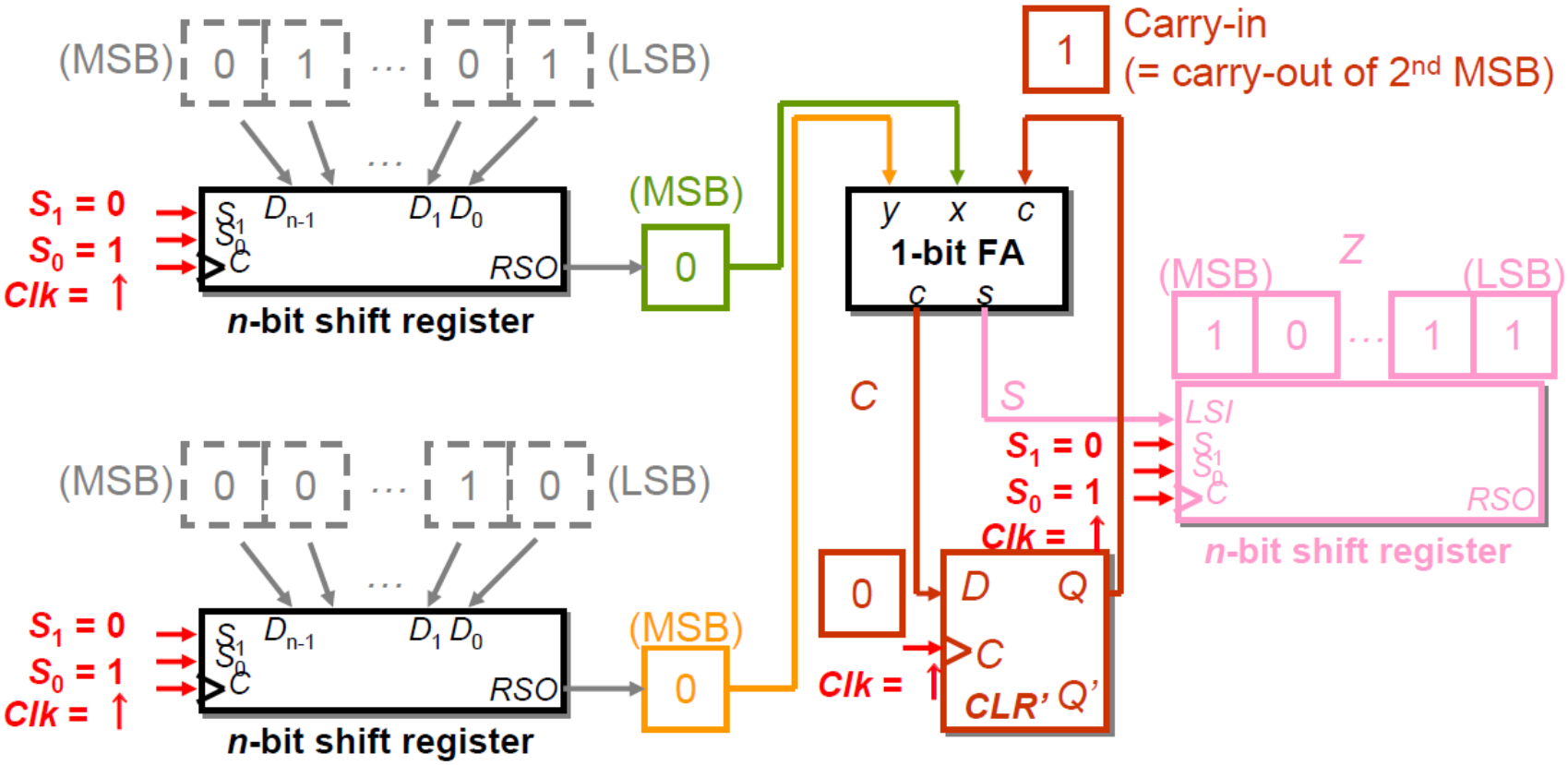
How Does It Work?

STEP 6: Calculate the next bit

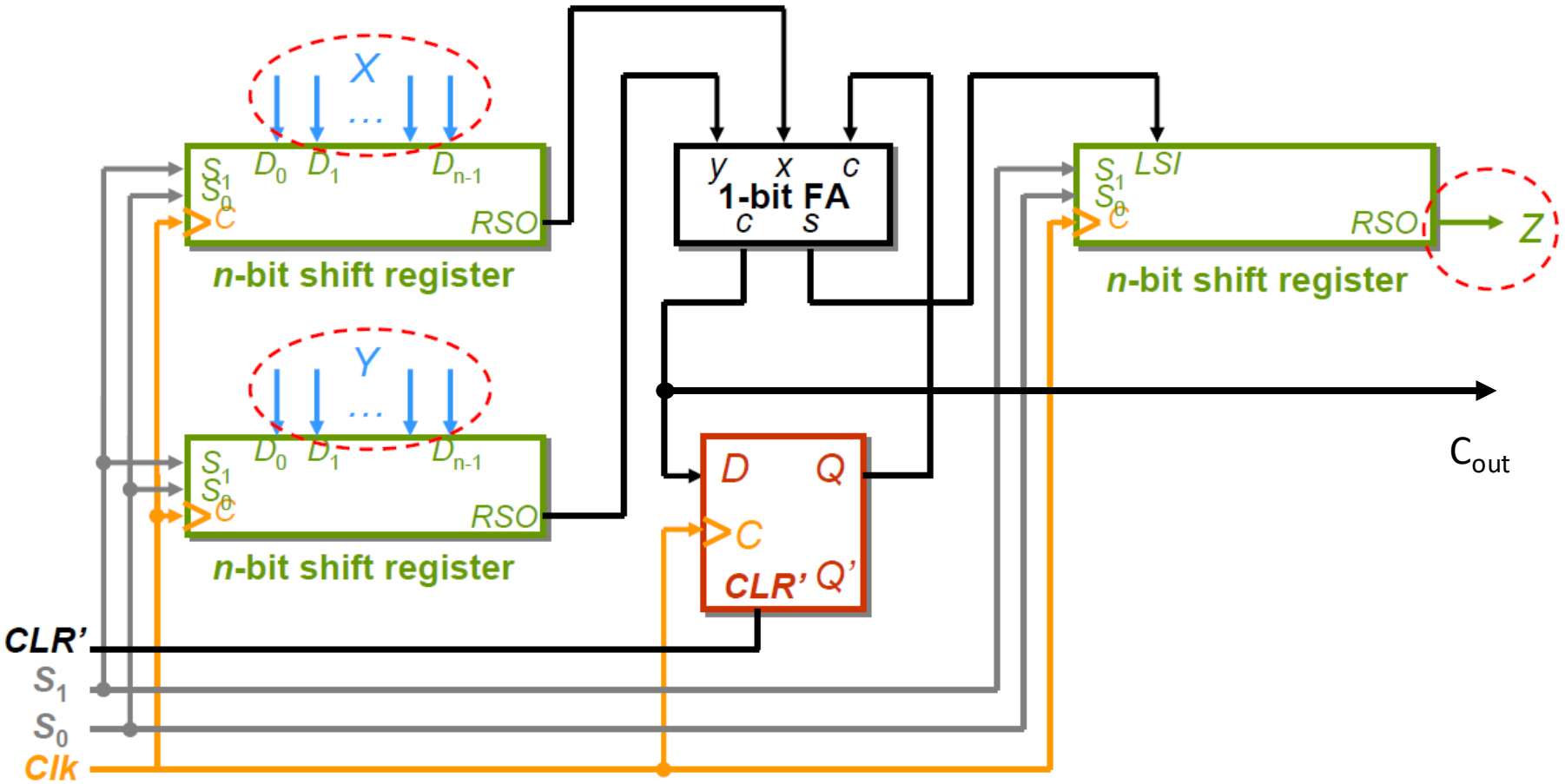


How Does It Work?

Last STEP: Calculate the MSB



Serial Adder (Full Circuit)



Input: X, Y (n-bit parallel load)

Output: Z (n-bit serial out, LSB first)

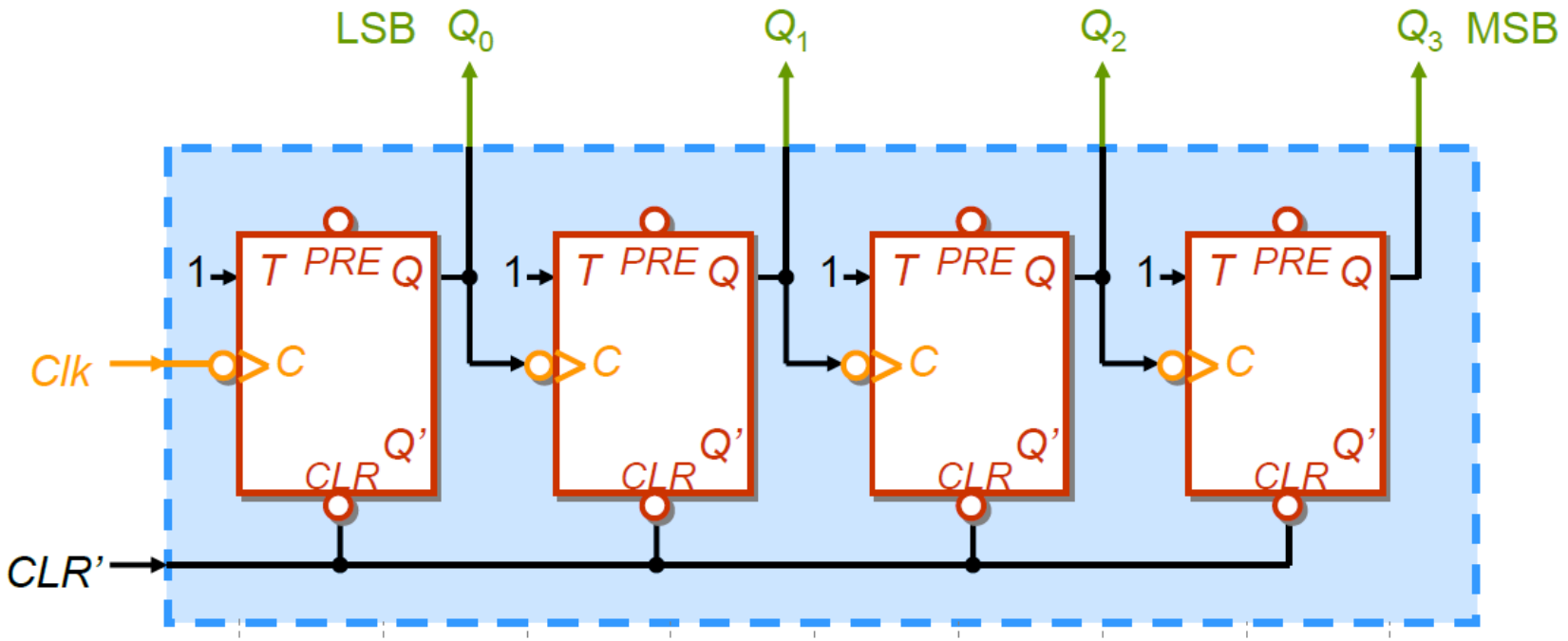
Adder: add the LSB first

$Z = X + Y$

10.2 Counters

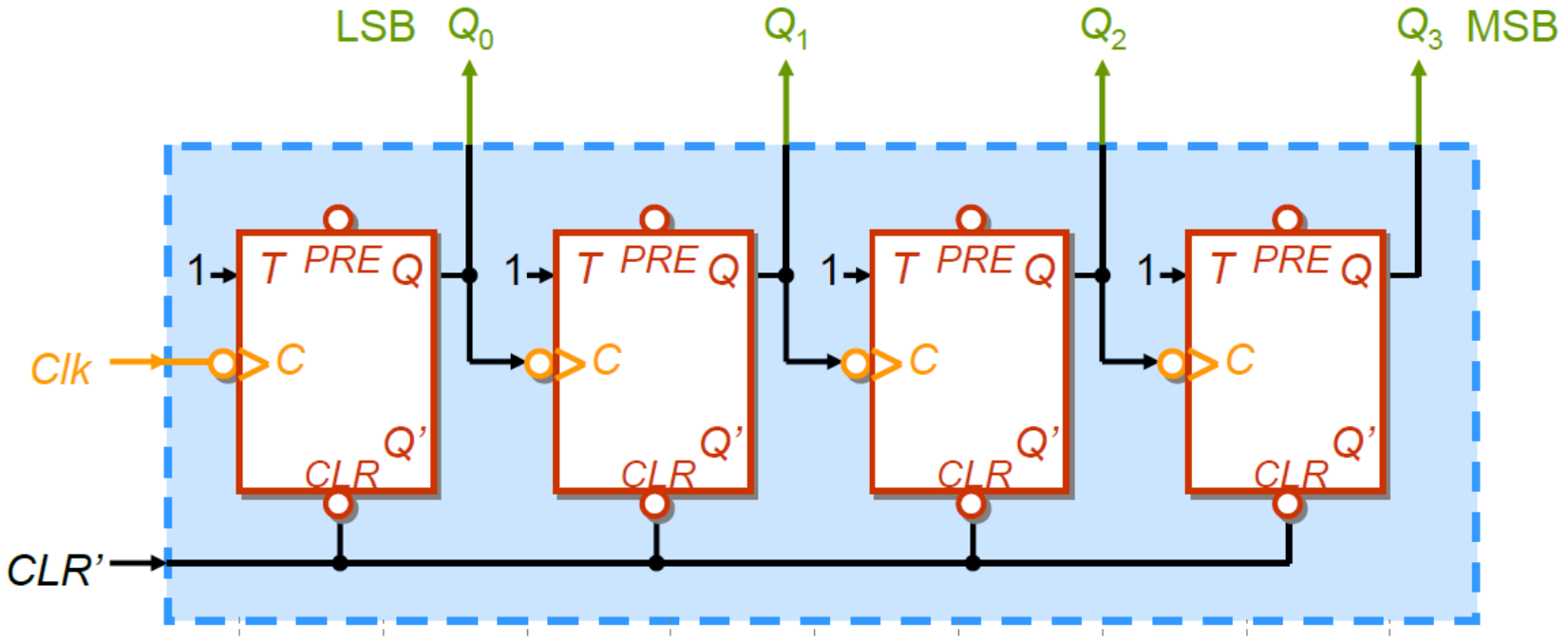
- A special register that goes through a predefined sequence of states
- Used to control operations in a circuit
- Asynchronous counter such as ripple counter (FFs are triggered by the transition of other FFs)
- Synchronous counter: FFs are triggered by a common clock pulse signal

Ripple Counter (Asynchronous)



- Input signals all HIGH ‘1’: Toggle mode
- First FF is triggered with Clk signal
- Subsequent FFs are triggered by the output transition of previous FFs.

How Does It Work?



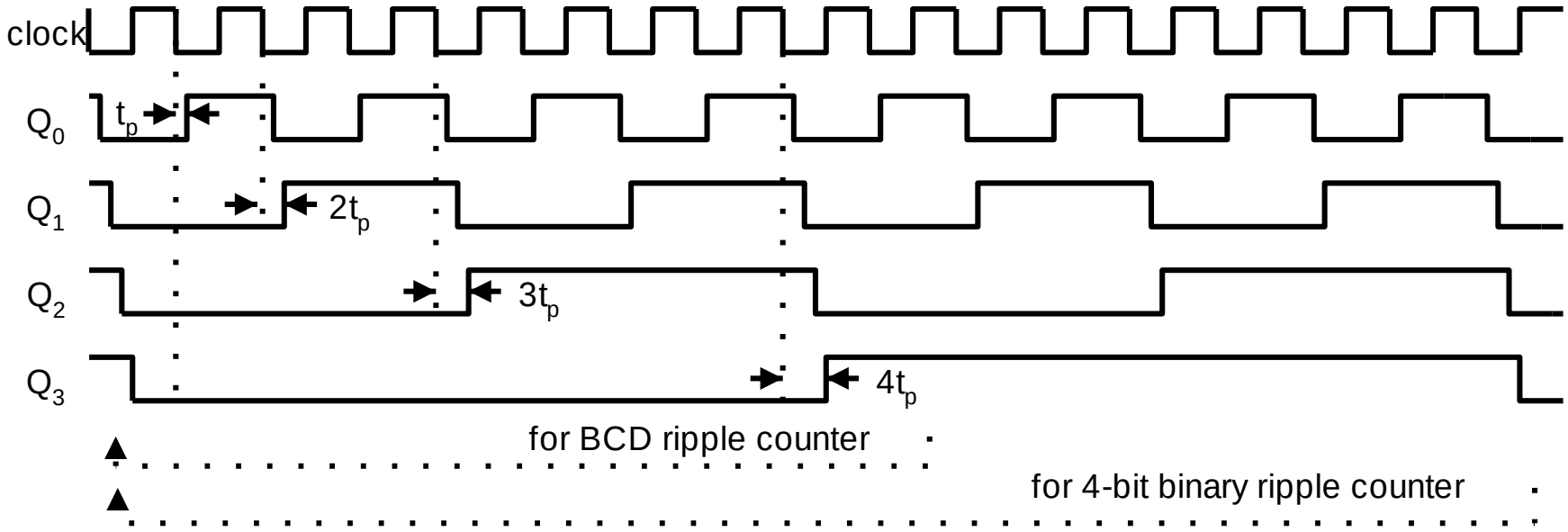
Clk ↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Q_0	0	1 →	0	1 →	0	1 →	0	1 →	0	1 →	0	1 →	0	1 →	0	1
Q_1	0	0	1	1 →	0	0	1	1 →	0	0	1	1 →	0	0	1	1
Q_2	0	0	0	0	1	1	1	1 →	0	0	0	0	1	1	1	1
Q_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Characteristics

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

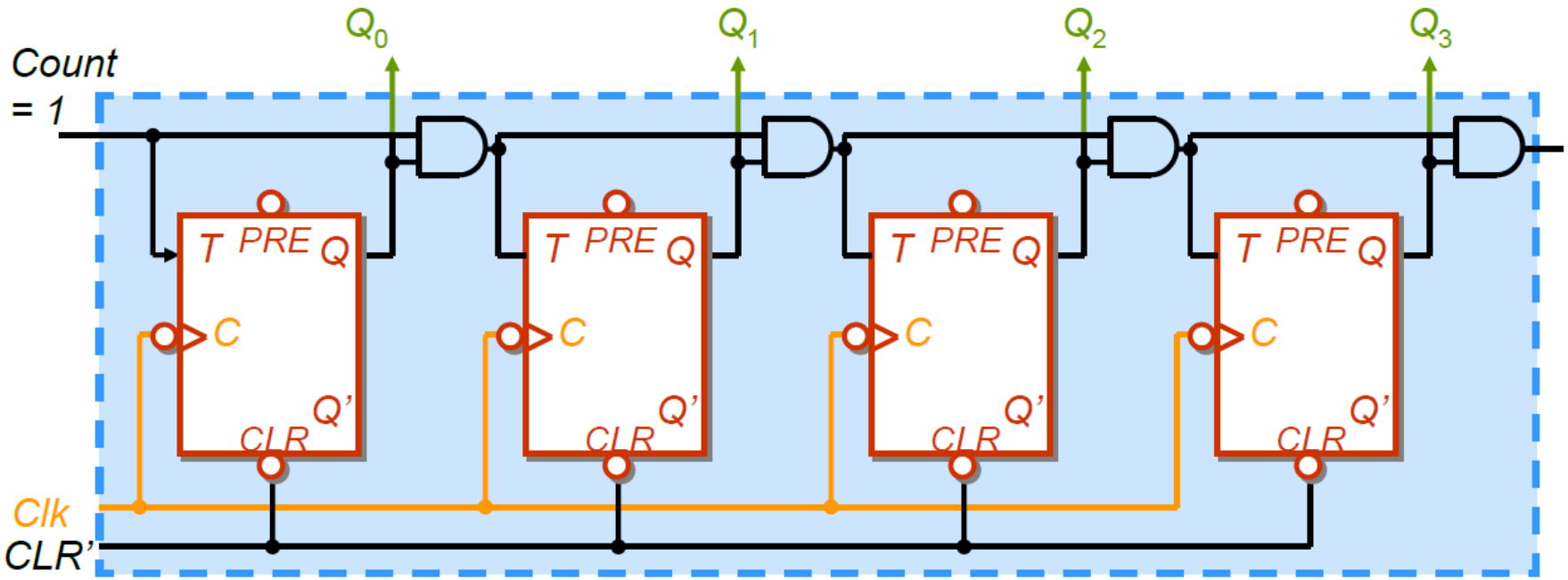
- Up Counter: from 0000 to 1111
- Asynchronous: Triggered by other FFs
- Problem?

Propagation Delay



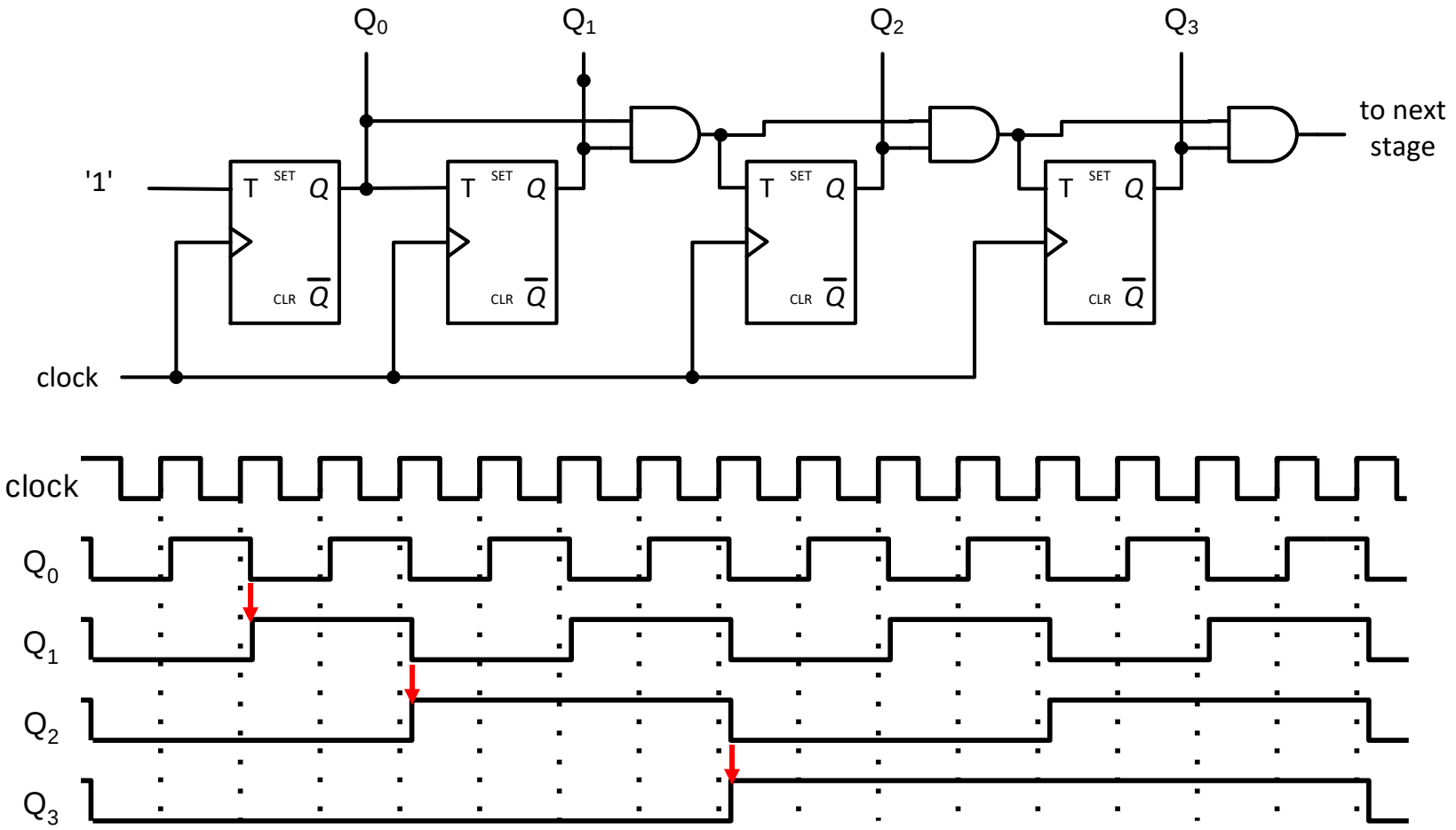
- Propagation delay will rippled through the “clock input” of FFs
- Last flip-flop has the largest delay $4t_p$ for 4-bit ripple counter

Synchronous Counter



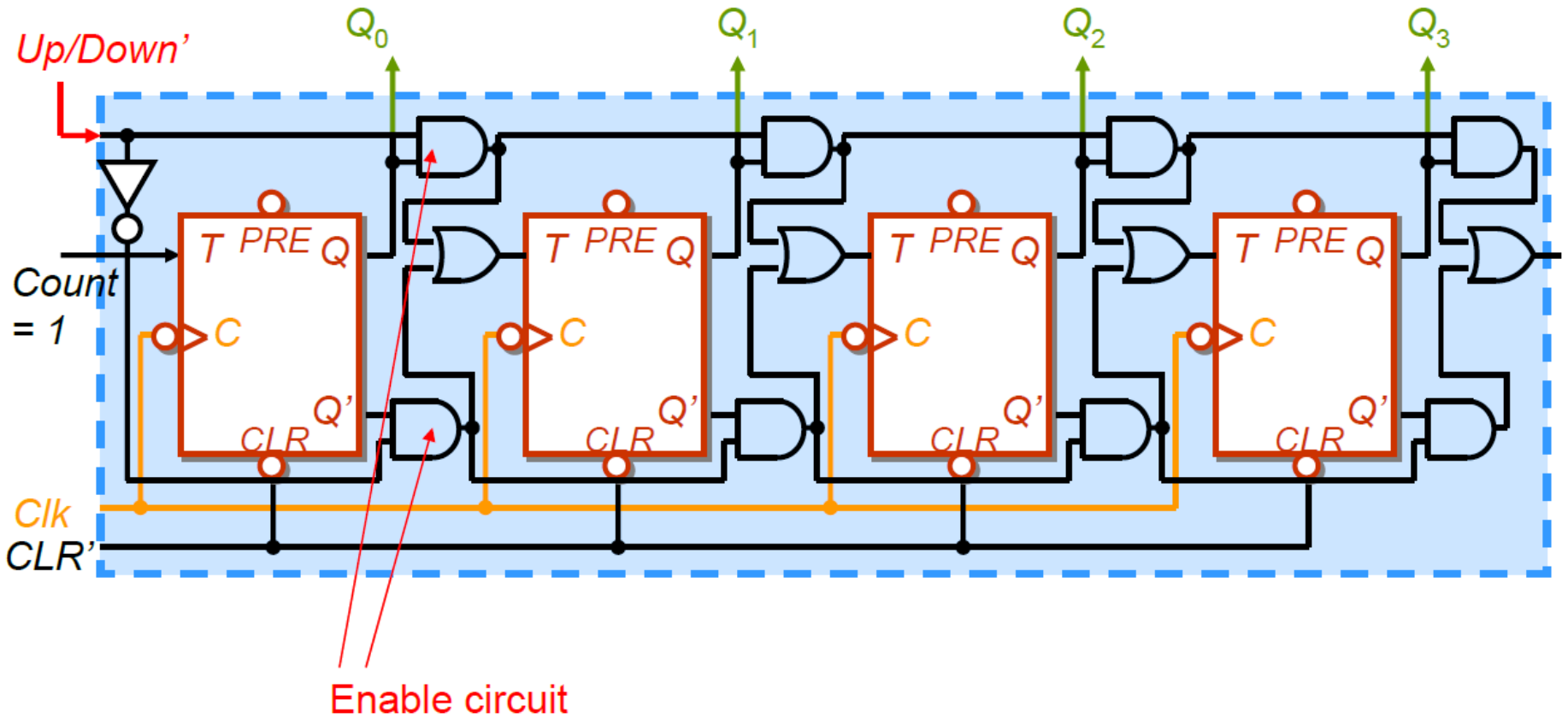
All FFs are driven by a common clock, thus all the change of states will occur at the same time without further delay

Synchronous Counter



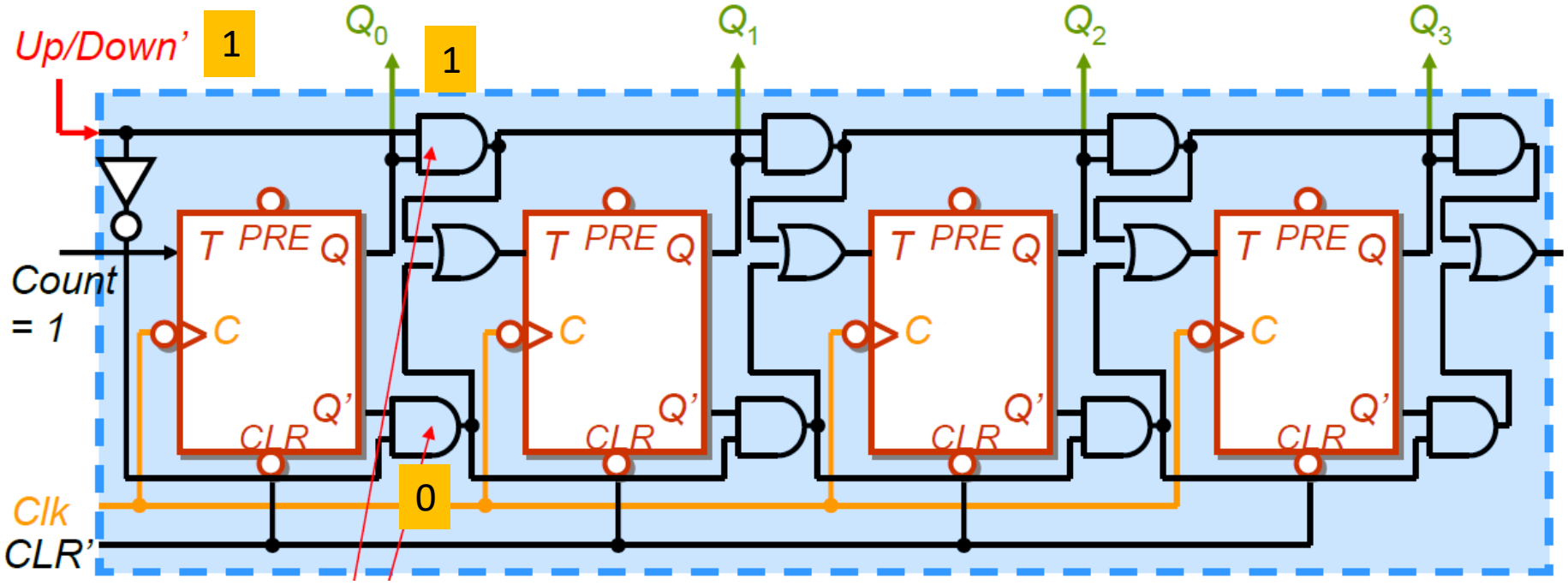
Delays of all FFs are their own propagation delays! No ripple!

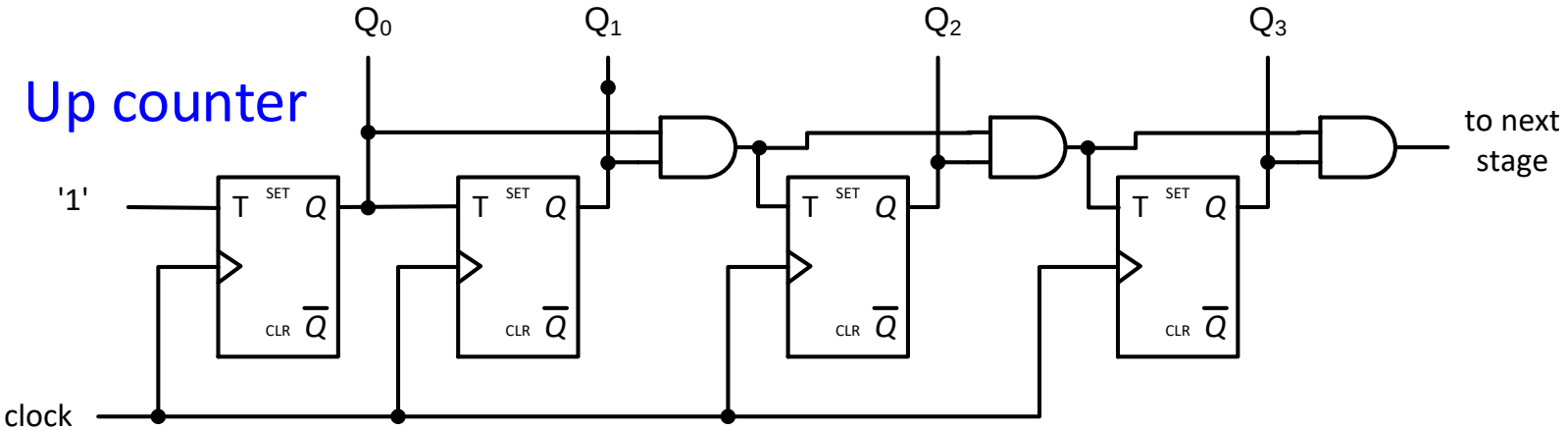
Bidirectional (Up-Down) Counter



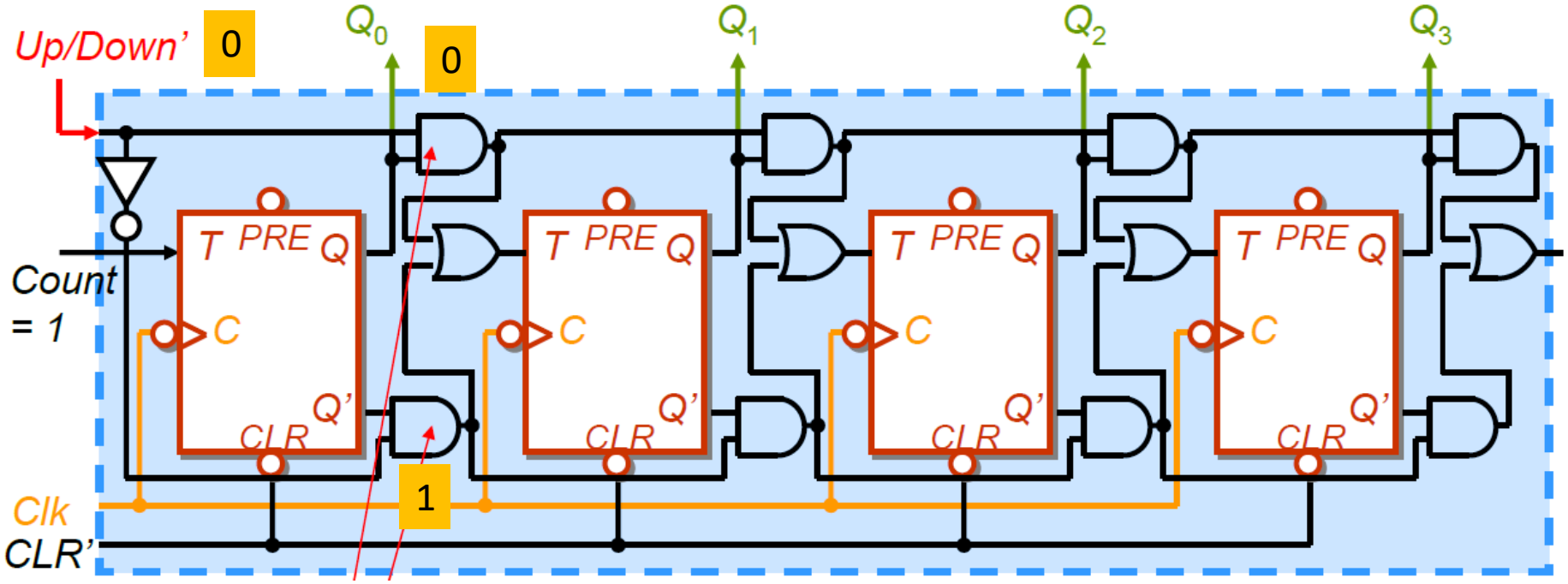
Additional input ***Up/Down'*** to control the counting direction by selecting feeding either ***Q*** or ***Q'*** to next ***T***

How Does It Work?

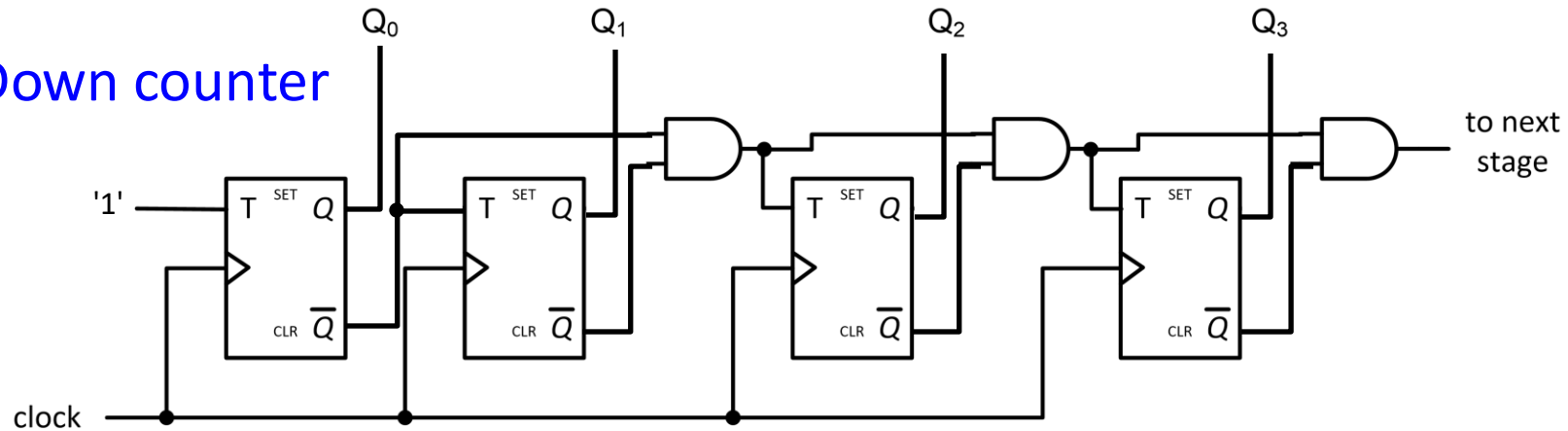




How Does It Work?

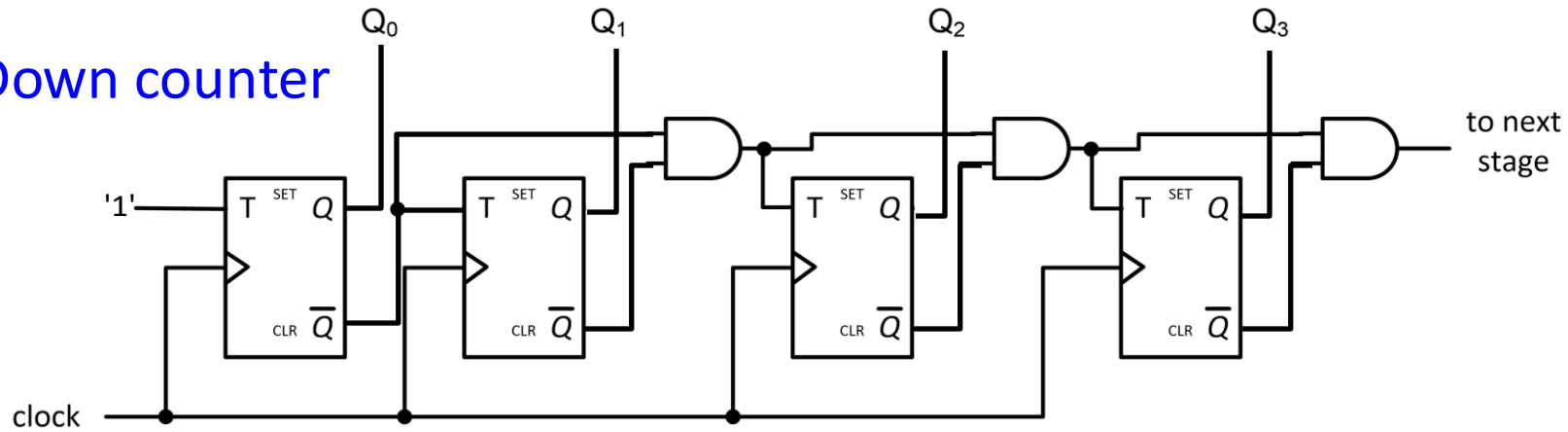


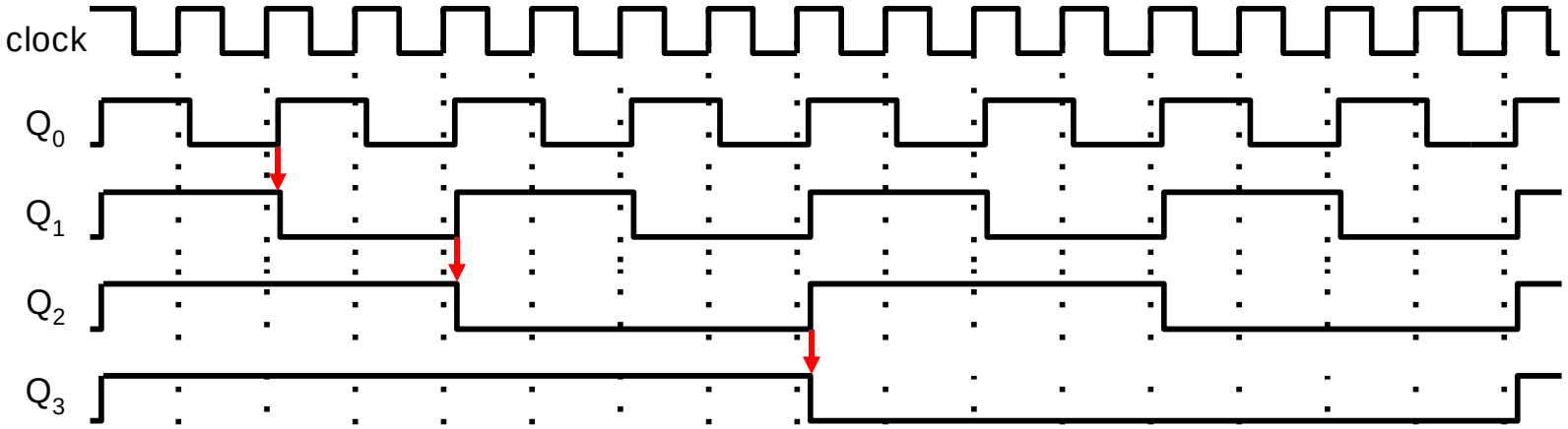
Down counter



How Does It Work?

Down counter



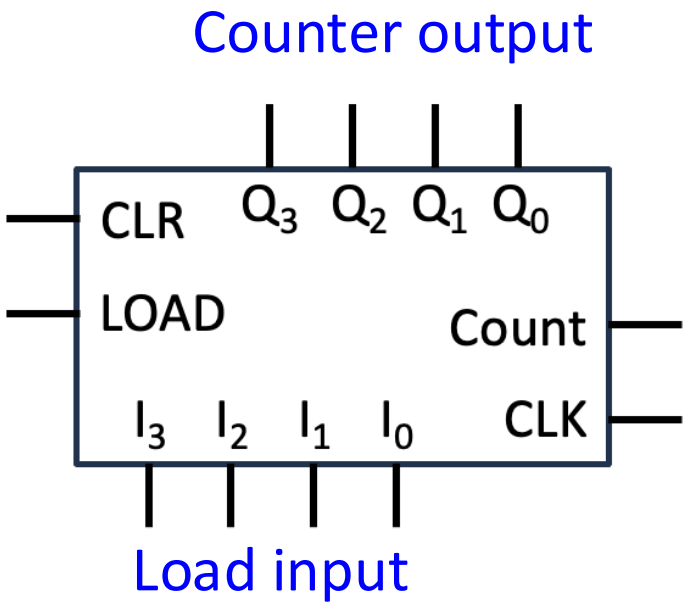


Synchronous Up/Down Counter

Upward Counting Sequence			
Q_3	Q_2	Q_1	Q_0
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
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Downward Counting Sequence			
Q_3	Q_2	Q_1	Q_0
1	1	1	1
1	1	1	0
1	1	0	1
1	1	0	0
1	0	1	1
1	0	1	0
1	0	0	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1
0	0	0	0

Binary Counter

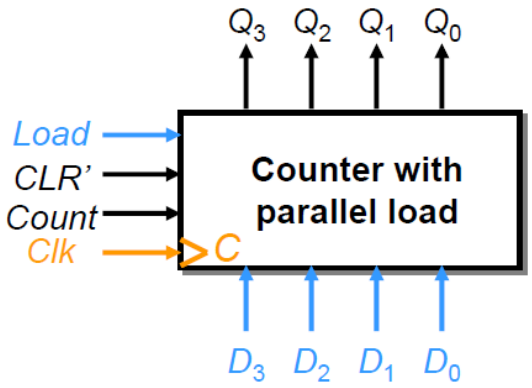


- **LOAD**: to load inputs ($I_3I_2I_1I_0$) to outputs ($Q_3Q_2Q_1Q_0$) with signal '1'
- **CLR**: to clear the counter ($Q_3Q_2Q_1Q_0$ = "0000") with signal '1'
- **Count**: +1 with signal '1' (e.g., from "0010" to "0011")
- **LOAD/CLR**: can be active high/low and async/sync

Modulo Counter

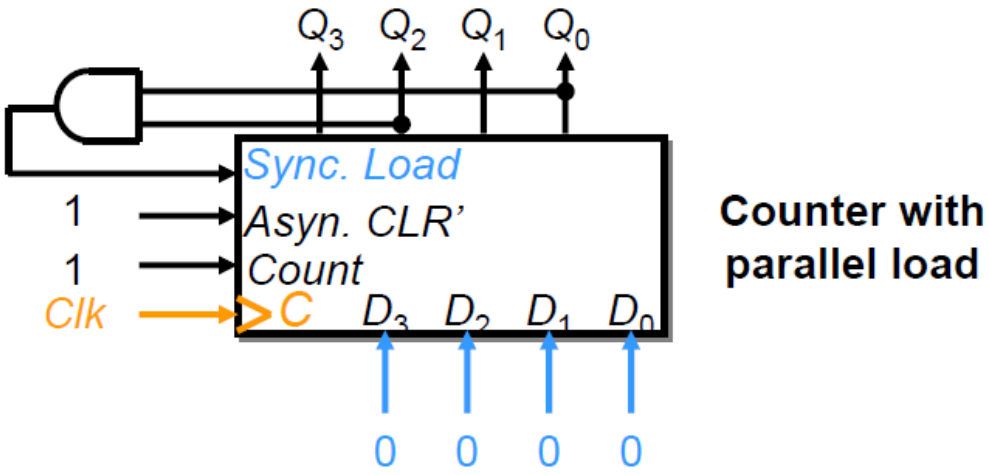
- A modulo-***N*** (abbreviated mod-***N***) counter is a counter that goes through a repeated sequence of ***N*** counts.
  - 0, 1, 2, 3, 0, 1… → mod-4 counter

- Counter + Parallel Load



Parallel load the data into the counter when *Load* = 1 and *Clk* is in active transition

Modulo Counter (Example)

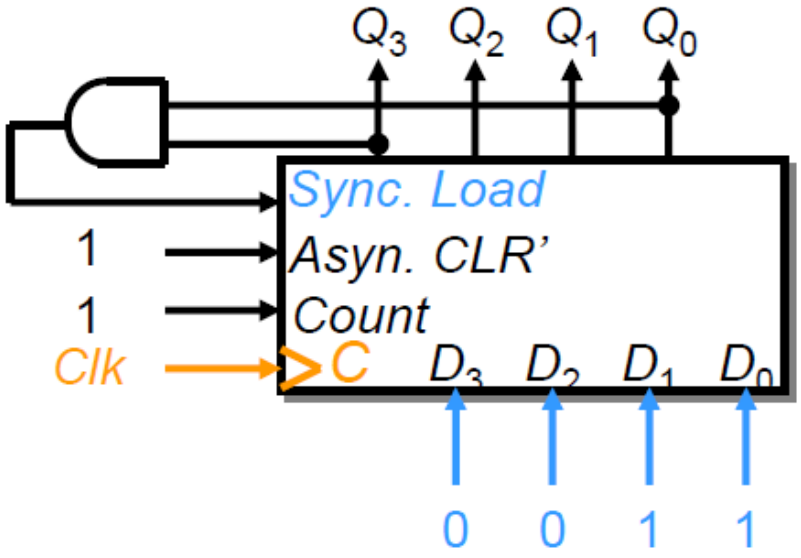


- Initial input: 0000
- Synchronous Load Bit: Q_2Q_0

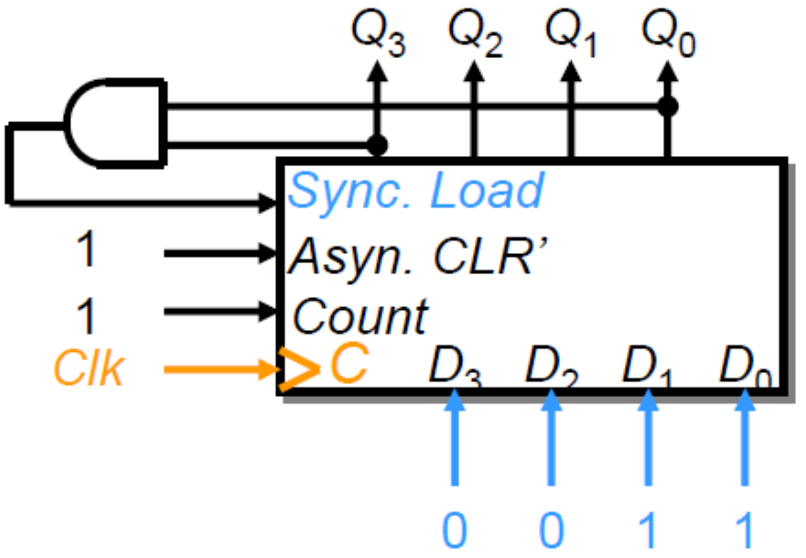
  → Reset to 0000 after x1x1
- Counter: 0000 → 0001 → 0010 → 0011 → 0100 → 0101 → 0000
- Mod-6 counter

Exercise

Determine the number sequence generated by the counter.



Exercise

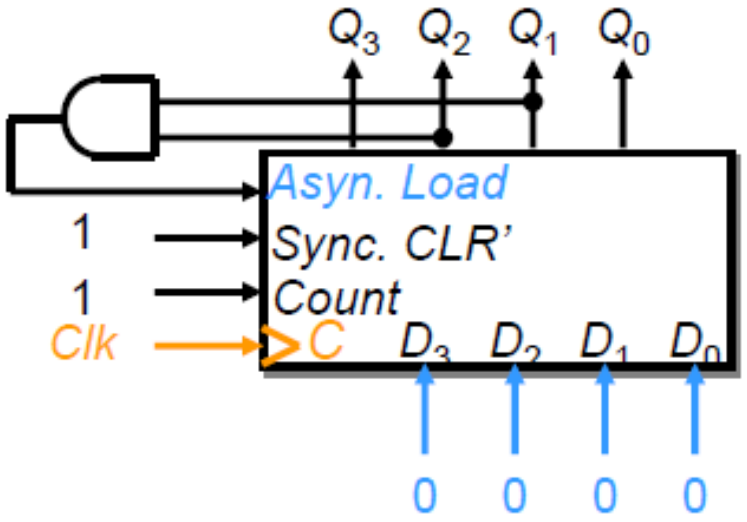


- Initial input:
- Synchronous Load Bit:

  → Reset to after
- Counter:

- Mod- counter

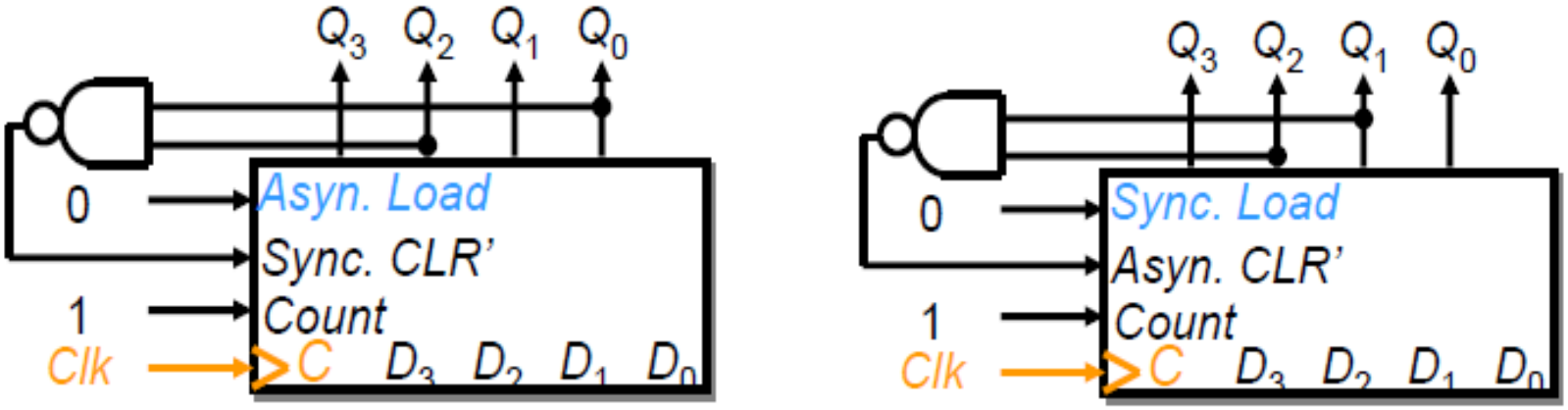
Asynchronous Load



- Initial input: 0000
- Asynchronous Load Bit: Q_2Q_1

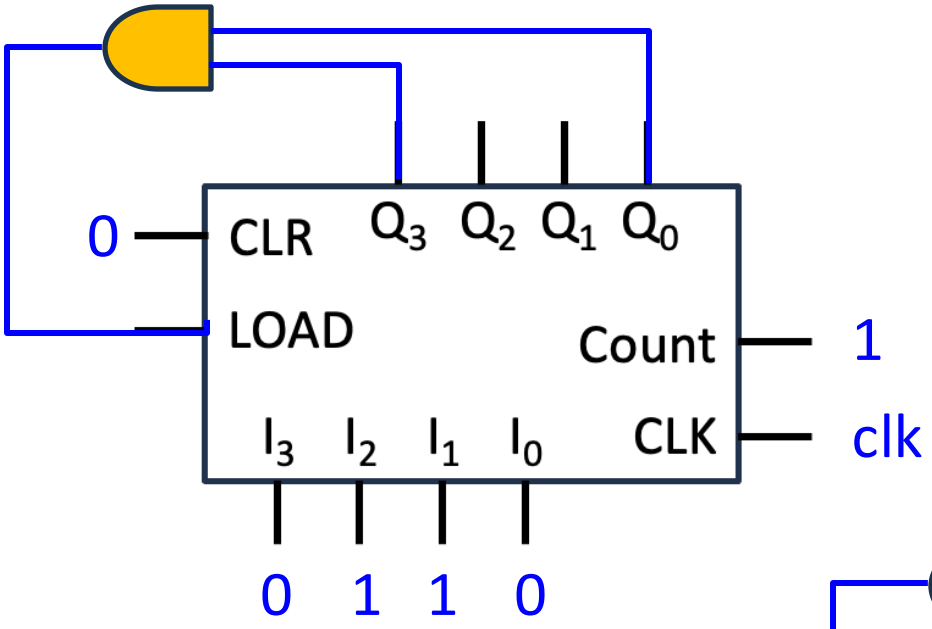
  → Reset to 0000 once x11x is detected
- Counter: 0000 → 0001 → 0010 → 0011 → 0100 → 0101 → 0110/0000
- 0110 only appear for a negligible period and the counter is reset

Use CLEAR without Input

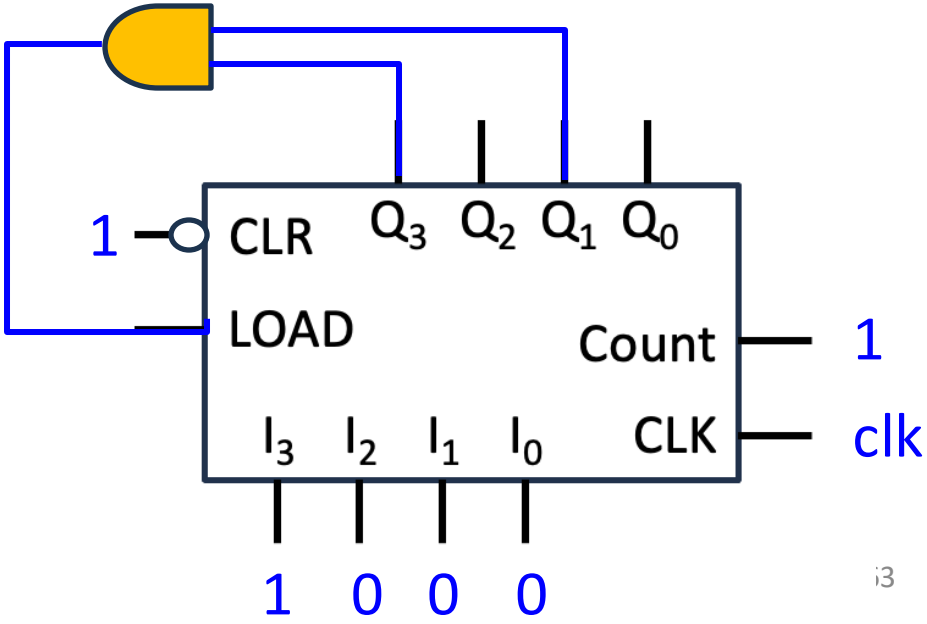


- Simpler design without inputs (Start with "0000")
- Synchronous CLR: Counter reset at the next Clk (0101)
- Asynchronous CLR: Counter reset instantly when 0110 is detected

Examples



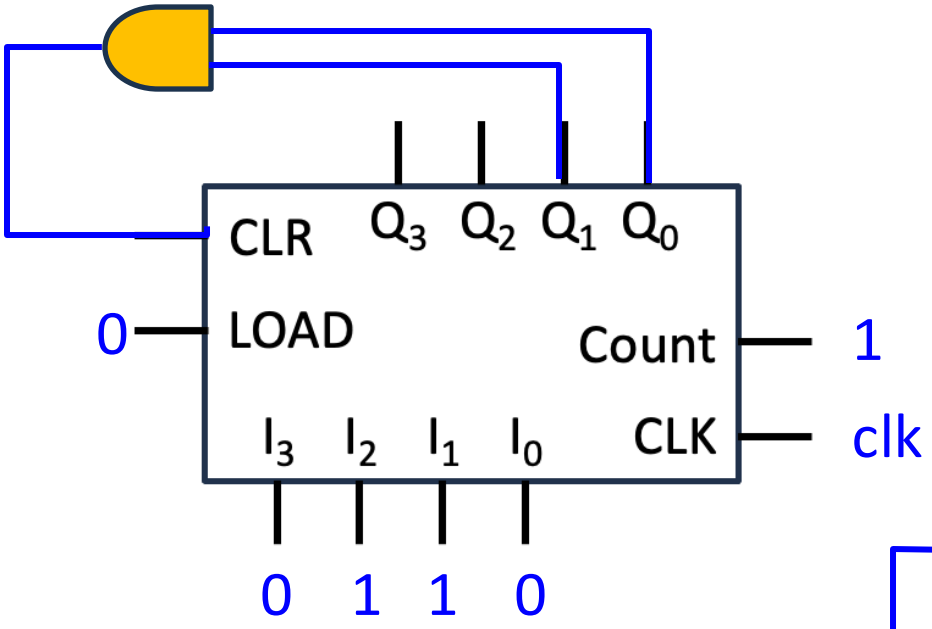
Counter: 0110 → 0111 → 1000 → 1001 → 0110



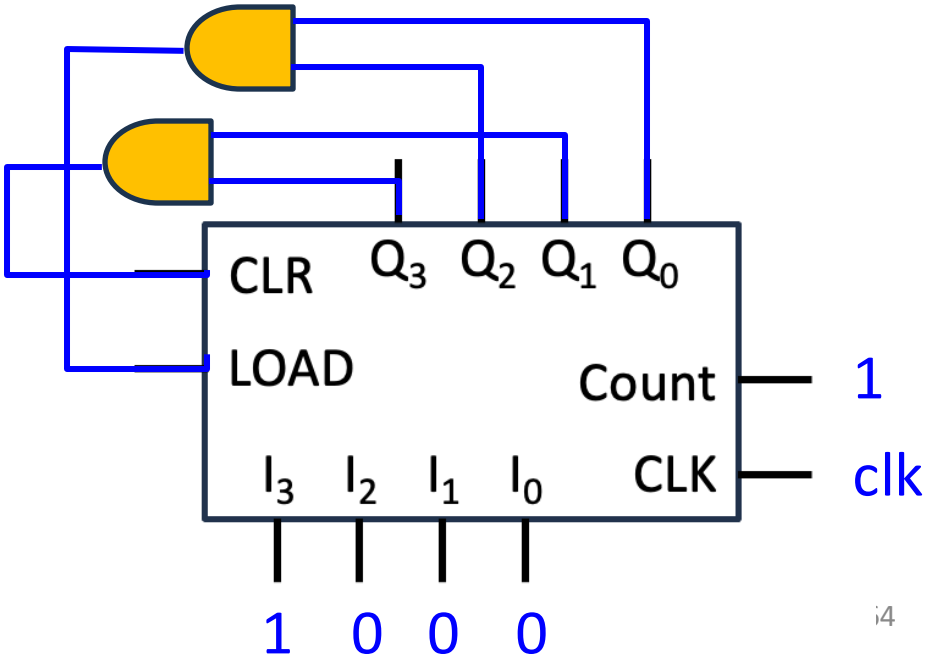
Counter: 1000 → 1001 → 1010 → 1000

Exercises

Counter:

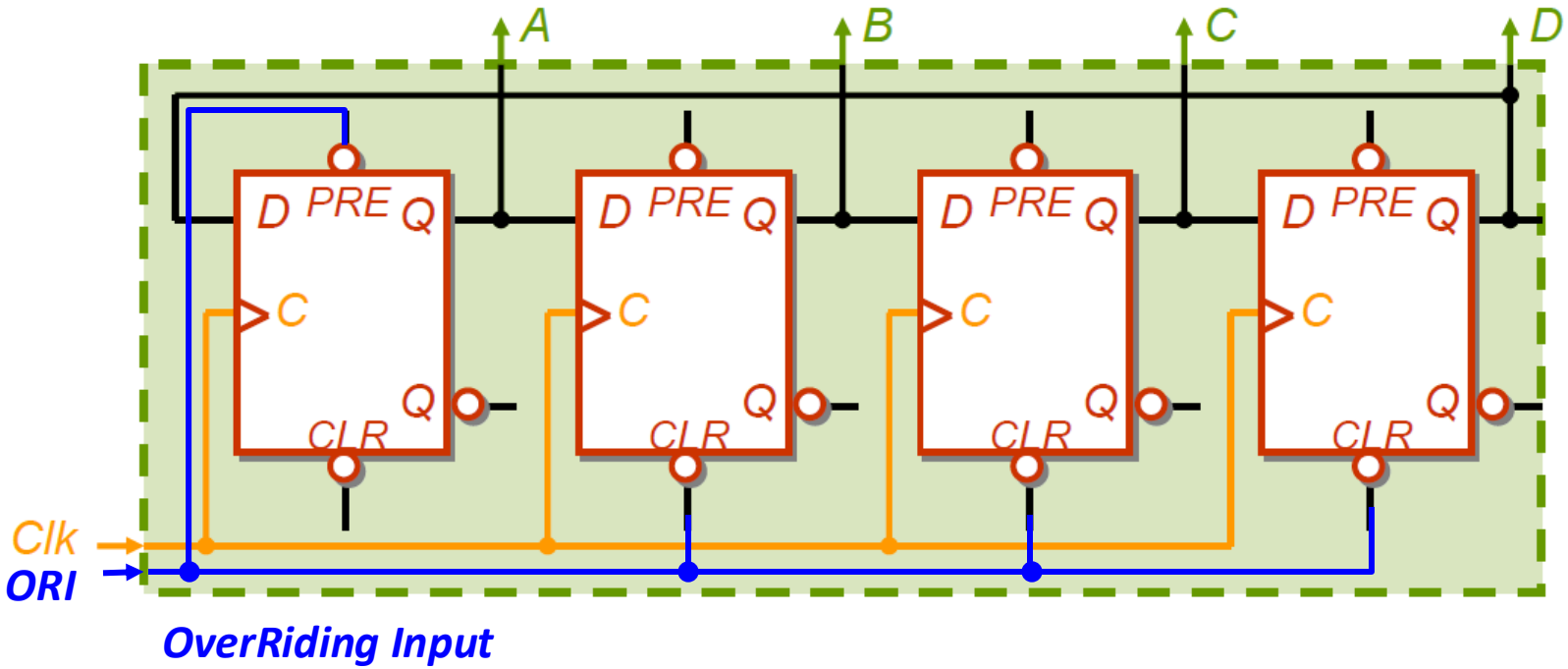


Counter:

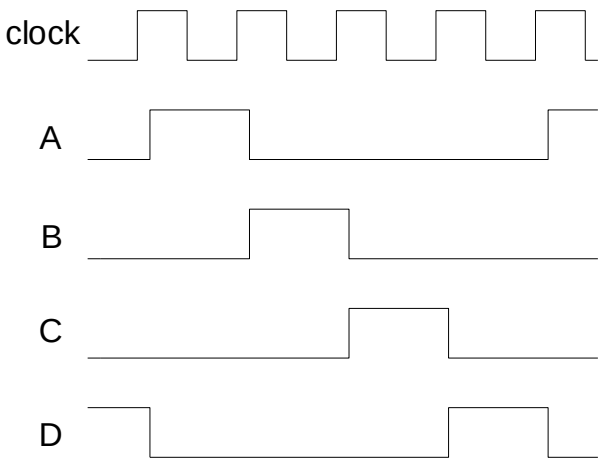


Ring Counter

- Circular shift counter: Output of last stage (FF) is connected to input of first stage (FF)
- Initial state: A is ‘1’, others are ‘0’
- Only one FF is set to ‘1’ at all time (One hot counter)



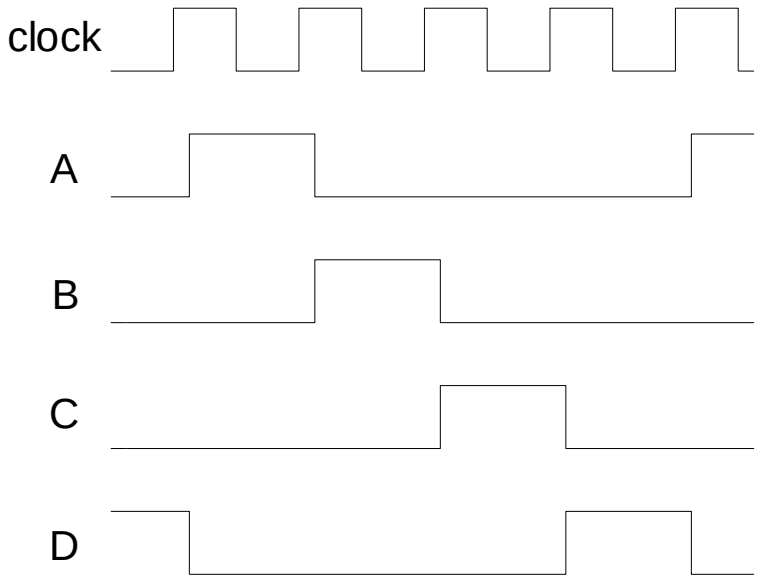
Ring Counter



Time	A	B	C	D	Decoded Output
1st clock period	1	0	0	0	A
2nd clock period	0	1	0	0	B
3rd clock period	0	0	1	0	C
4th clock period	0	0	0	1	D
5th clock period	1	0	0	0	A
…	…	…	…	…	…

- The ‘1’ will be rotated through the registers
- ***N***-stage counter = ***N*** FFs = ***N*** states
- Useful in time division multiplexing applications
  - Enable ONLY one output/device/user at a time

VHDL Codes for Ring Counter



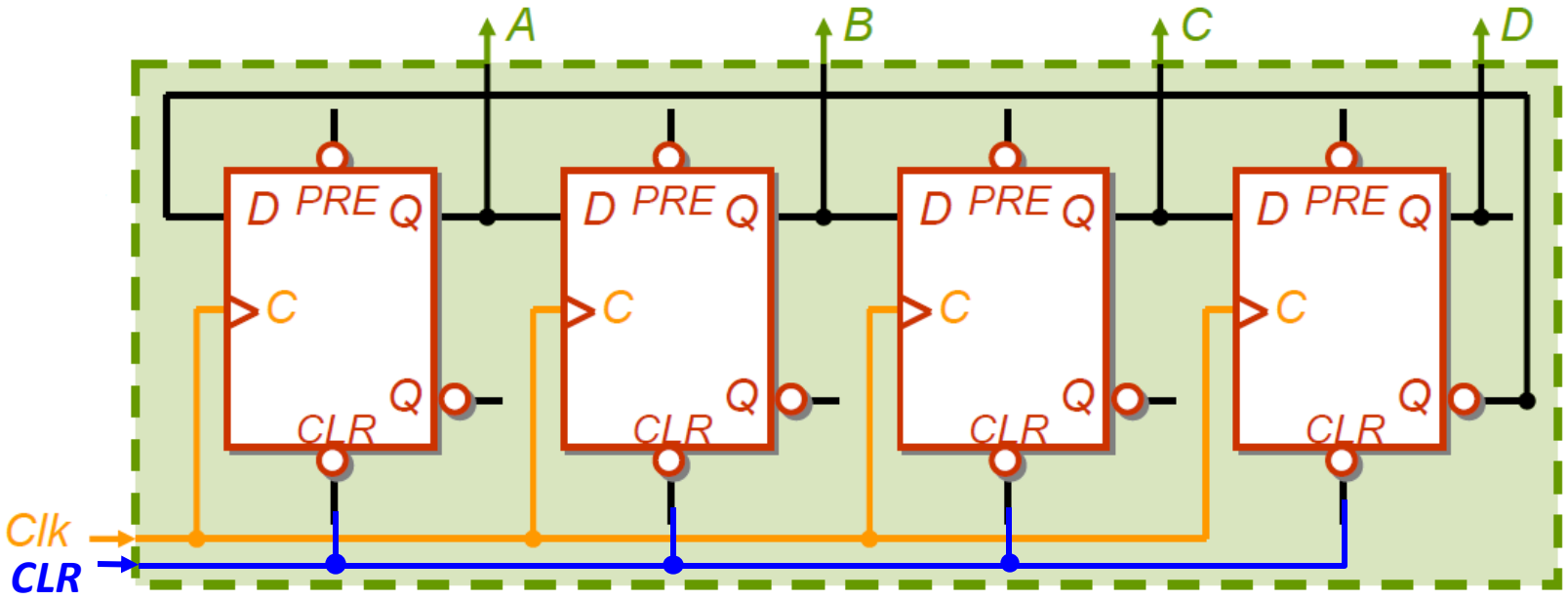
A	B	C	D	Decoded Output
1	0	0	0	A
0	1	0	0	B
0	0	1	0	C
0	0	0	1	D

VHDL Code for 4 bit Ring Counter

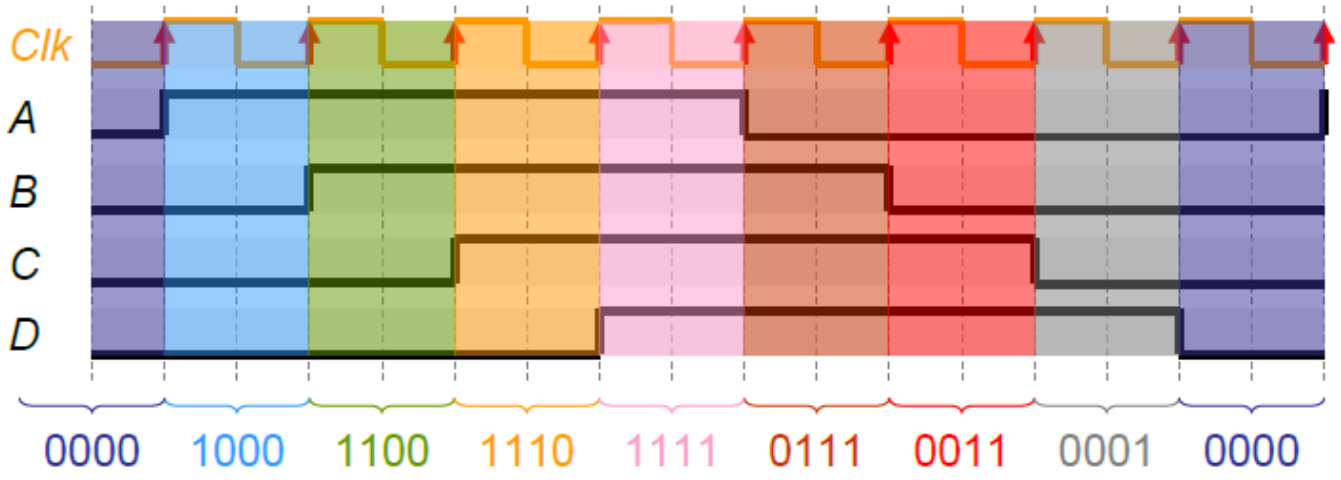
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Ring_counter is
    Port ( CLOCK : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end Ring_counter;

architecture Behavioral of Ring_counter is
signal q_tmp: std_logic_vector(3 downto 0):= "0000";
begin
process(CLOCK,RESET)
begin
if RESET = '1' then
    q_tmp <= “1000” ;
elsif Rising_edge(CLOCK) then
     q_tmp(3) <= q_tmp(0);
     q_tmp(2) <= q_tmp(3);
     q_tmp(1) <= q_tmp(2);
     q_tmp(0) <= q_tmp(1);
end if;
end process;
Q <= q_tmp;
end Behavioral;
```

Twisted Ring Counter (Johnson)

- Circular shift counter: Complement output of last stage (FF) is connected to input of first stage
- Initial state: All are ‘0’



Johnson Ring Counter

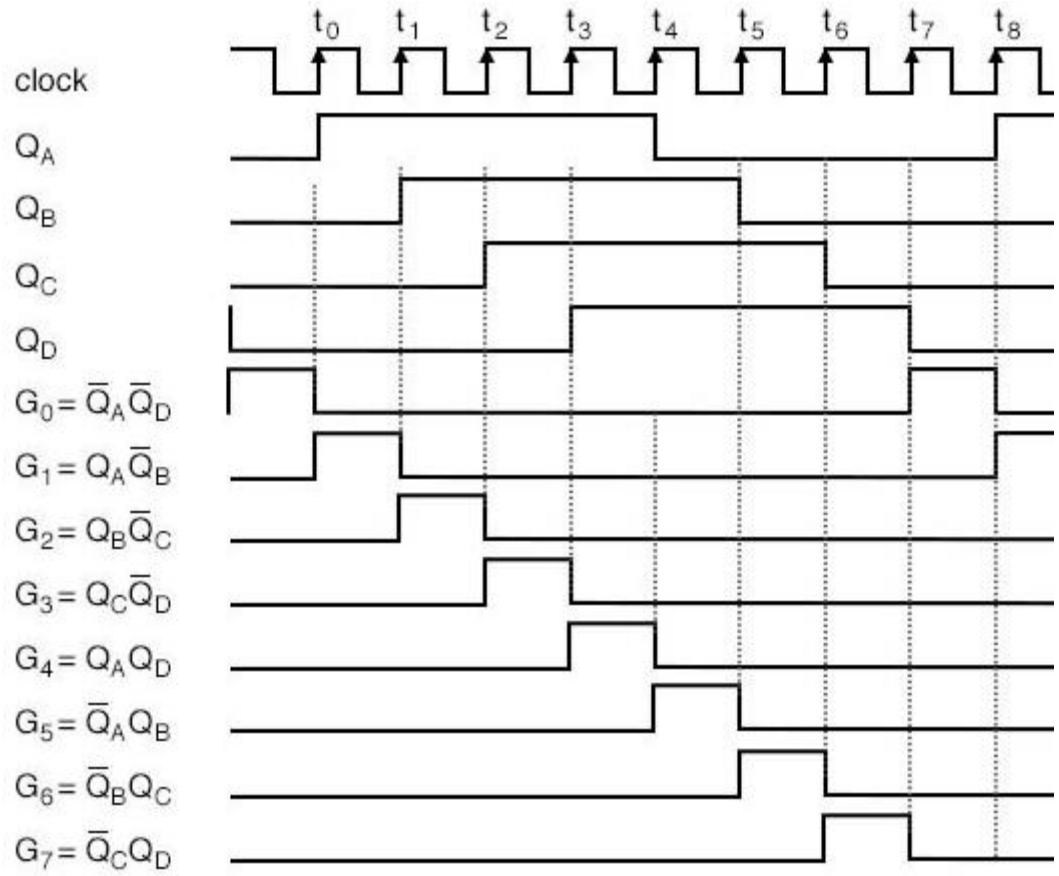


A	B	C	D
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1

- Count UP and then DOWN
- Adjacent States differ only by 1 bit, as in Gray Code
- ***N***-stage counter = ***N*** FFs = 2***N*** states

Johnson Ring Counter

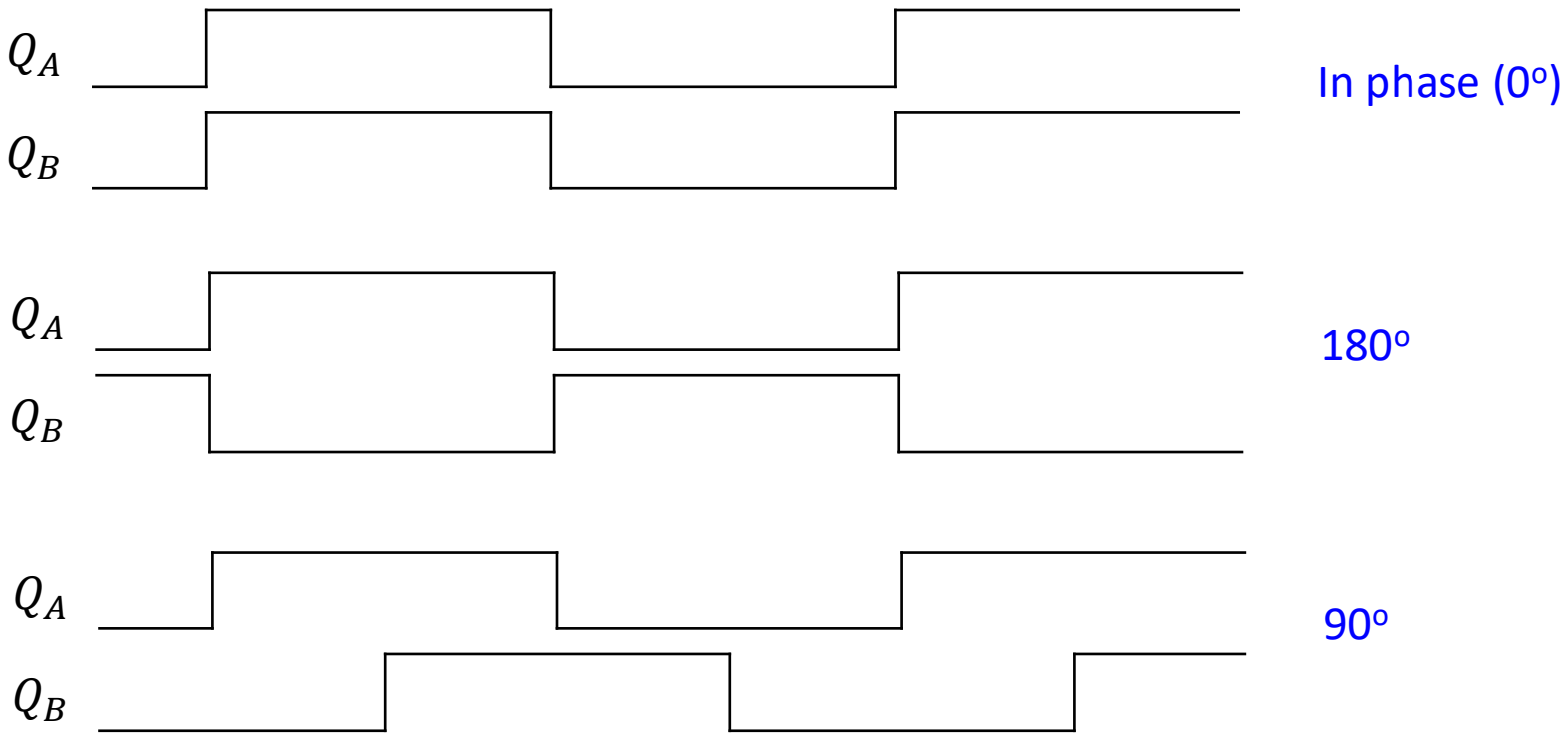
A	B	C	D	Decoded Output
0	0	0	0	$\overline{A}\,\overline{D}$
1	0	0	0	$A\,\overline{B}$
1	1	0	0	$B\,\overline{C}$
1	1	1	0	$C\,\overline{D}$
1	1	1	1	$A\,D$
0	1	1	1	$\overline{A}\,B$
0	0	1	1	$\overline{B}\,C$
0	0	0	1	$\overline{C}\,D$



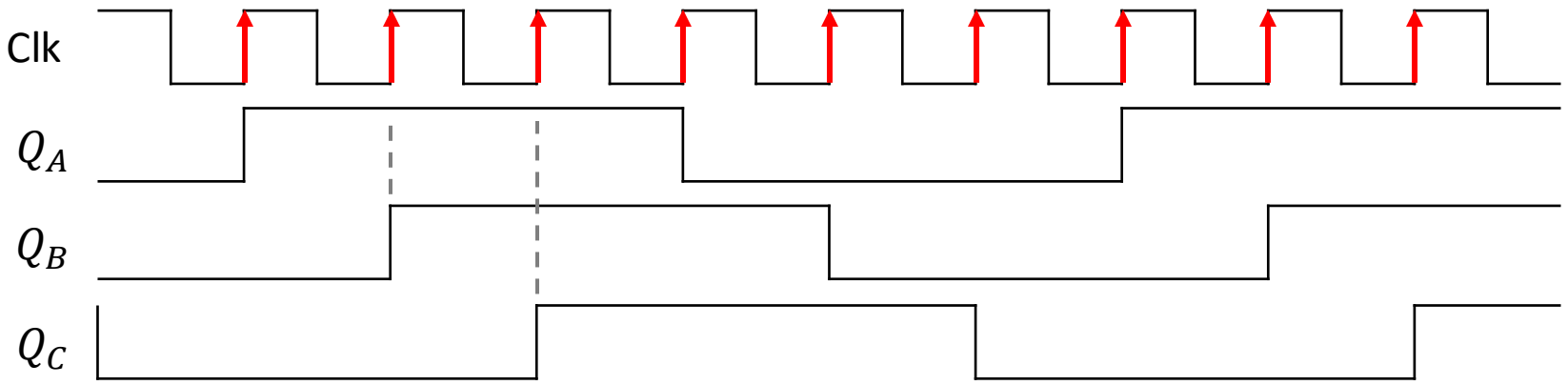
- Only two input gates are needed to decode each state for any N-stage Johnson ring counter

Example (3-phase wave generator)

Phase difference: Fraction of a period difference between the signals expressed in deg

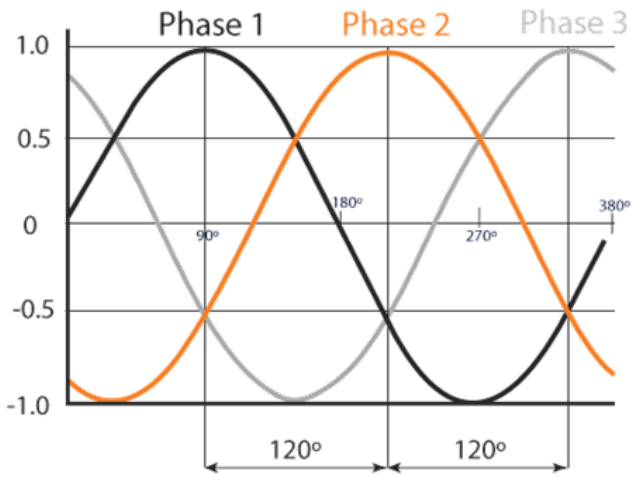


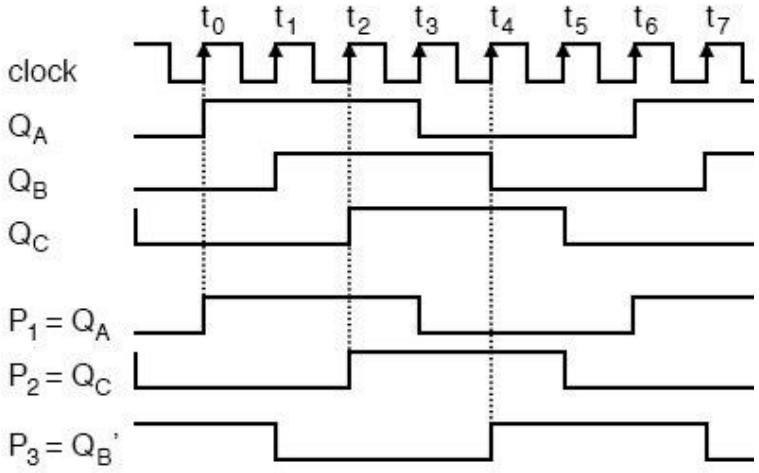
Example (3-phase wave generator)

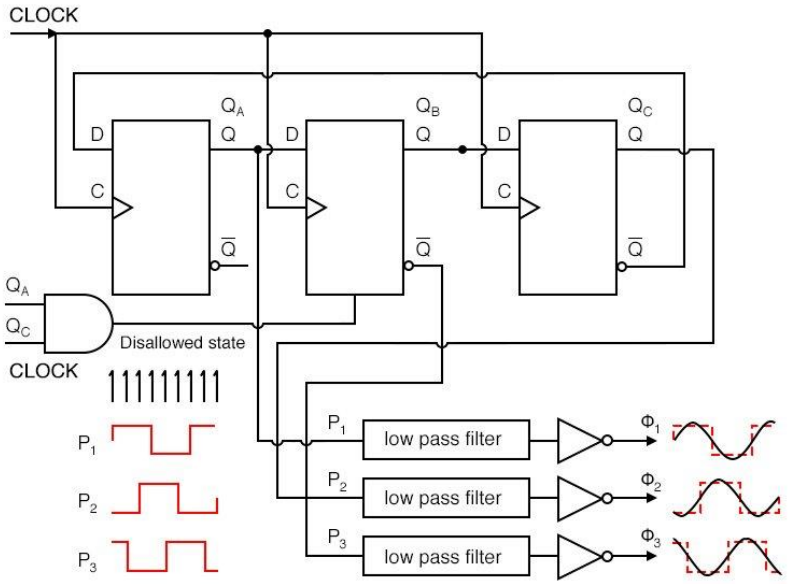


- 3-stage Johnson Ring Counter
- Phase difference
  - Q_A and Q_B: 60°
  - Q_B and Q_C: 60°
  - Q_A and Q_C: 120°

Example (3-phase wave generator)







- 3-phase alternating power supply
- 120^{o} phase difference
- Low-pass filter → sine waves