

Topic 8

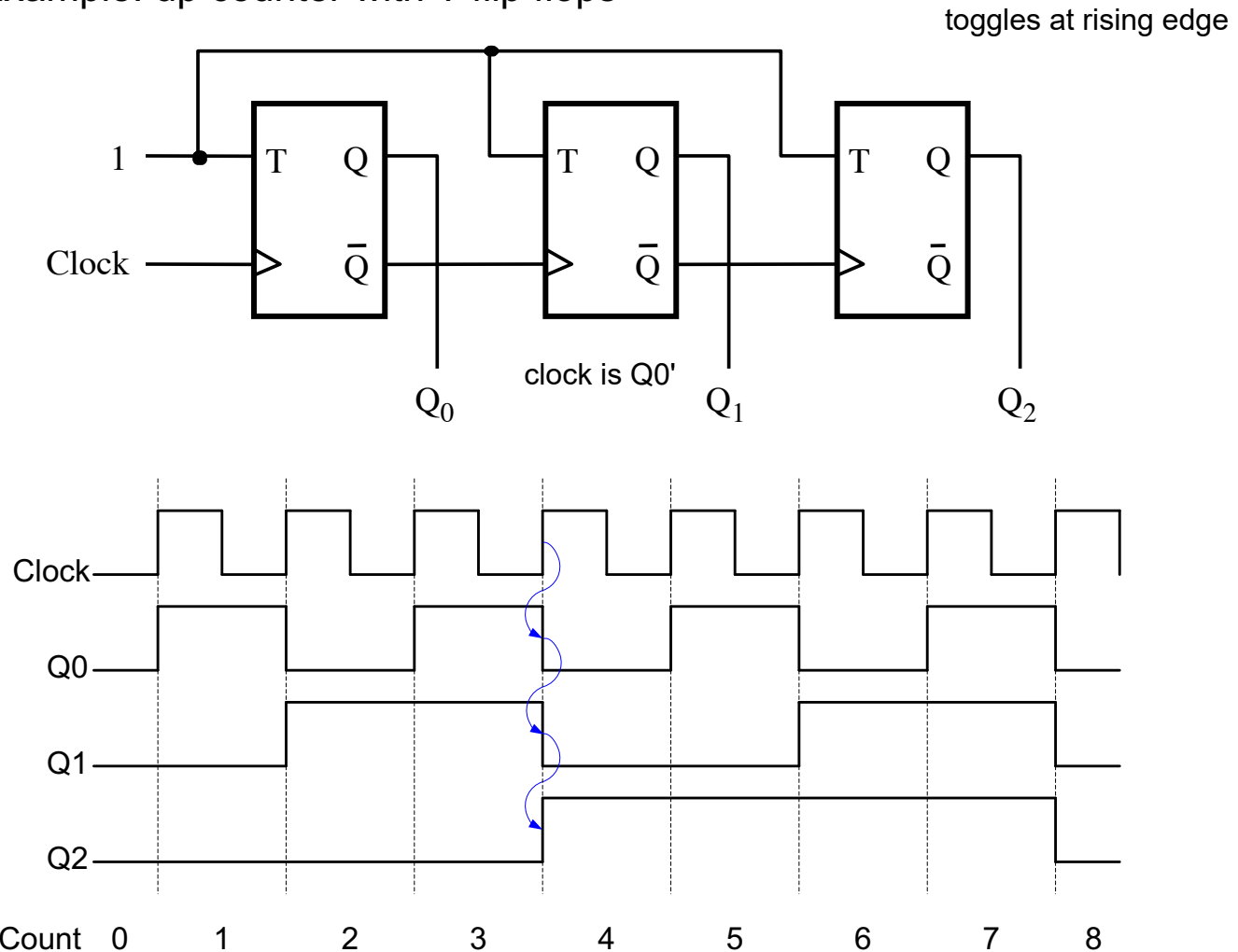
Counters

Counters

- A digital circuit that counts binary numbers
- Count in different format: binary, decimal, one-hot, ...
- An n -bit binary counter can count in binary from 0 up to 2^n-1 and repeat, or in reversed order
- Count up or count down, increment or decrement once per clock cycle – counting number of clock cycles
- An n -bit binary counter consists of n flip-flops
- Counters:
 - Asynchronous counters
 - Synchronous counters

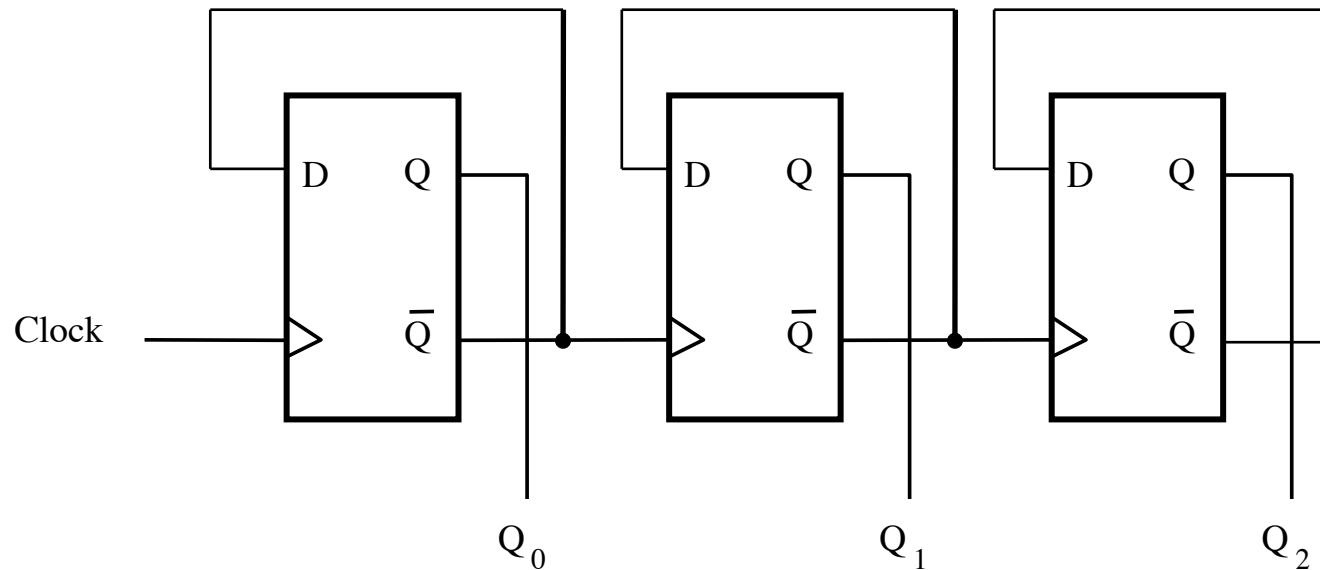
Asynchronous Binary Counter – Ripple Counter

- Flip flops are not triggered by a global clock signal
 - Example: up counter with T flip flops



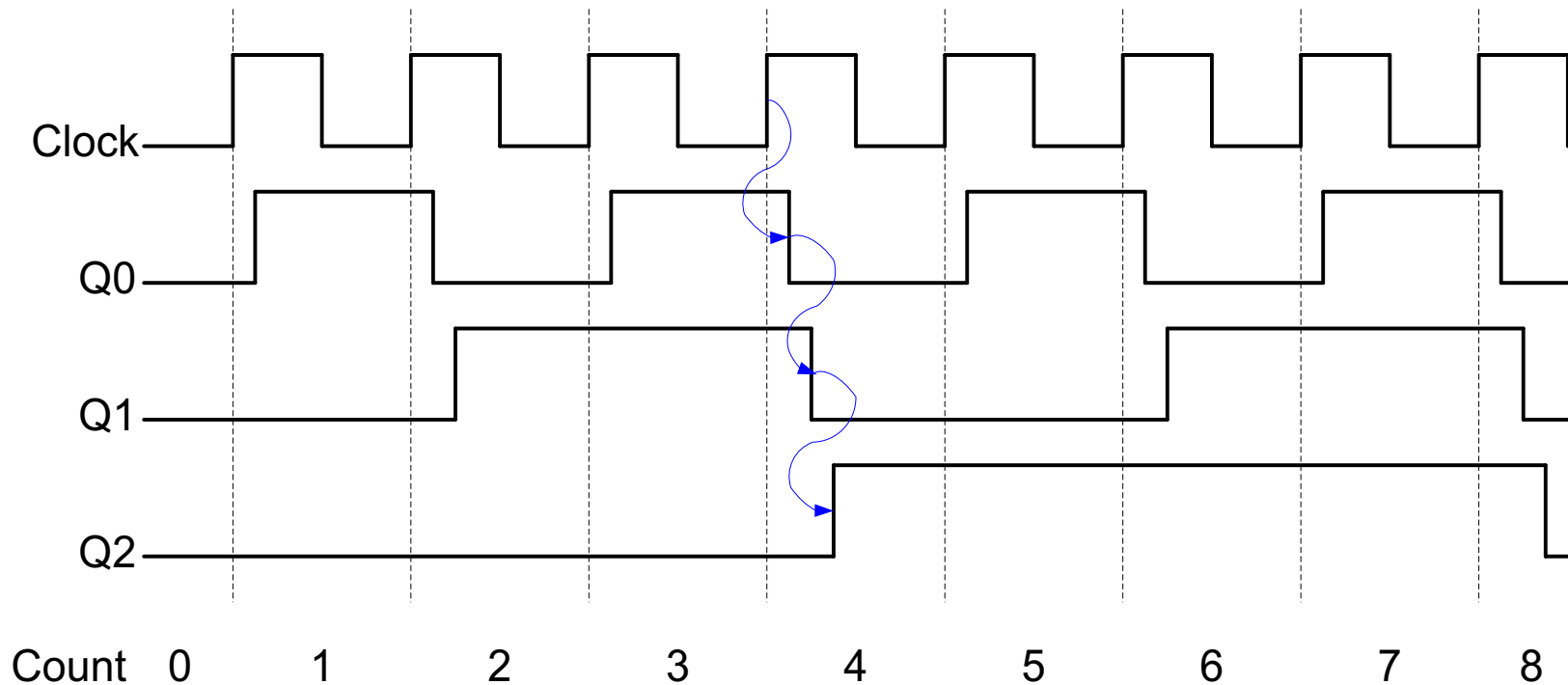
Asynchronous Binary Counter – Ripple Counter

- Example:
 - Alternative binary ripple counter, with D flip flops, up counter



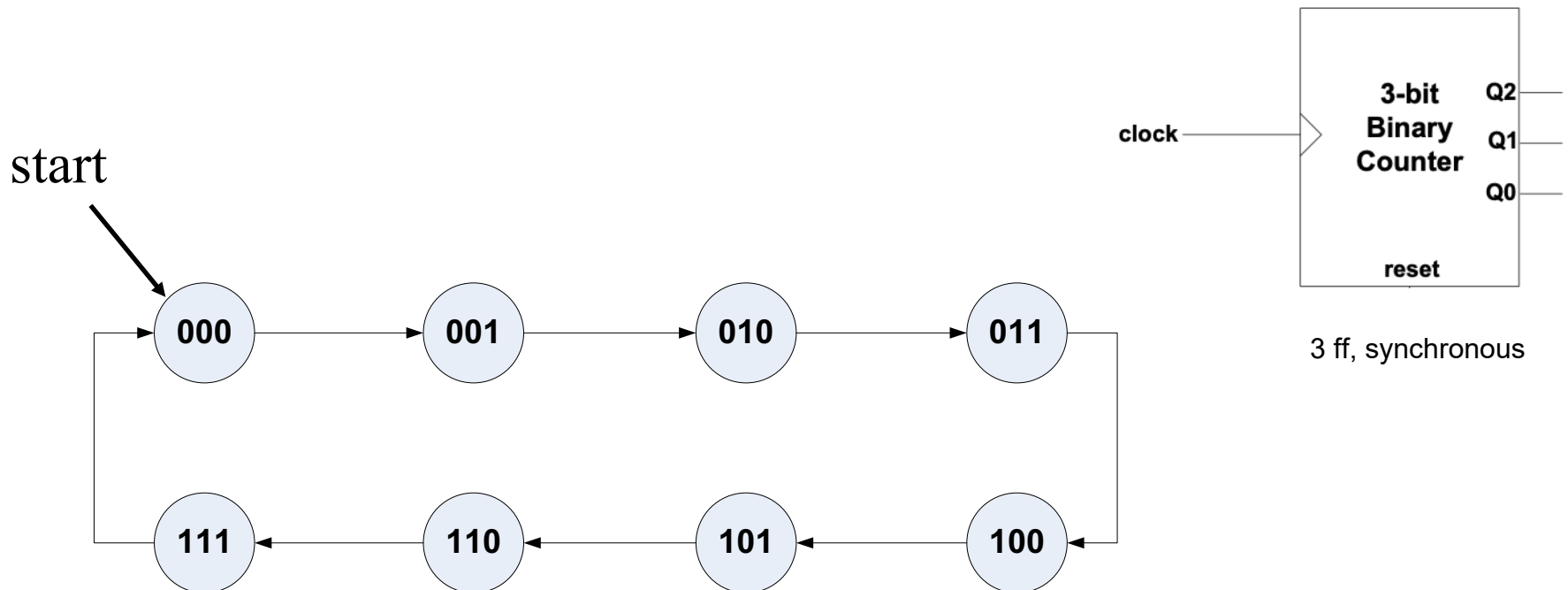
Asynchronous Binary Counter – Ripple Counter

- Problem with asynchronous counters:
 - Delays caused by each stage – timing issues



Synchronous Binary Counter

- All the flip-flops are triggered by (synchronized to) the same clock – synchronous counter
- May be implemented by different type of flip-flops
- Example: a 3-bit binary counter can count through this sequence



Synchronous Binary Counter Design


- Following the counting sequence

Current Value (state)			Next Value (state)		
Q2	Q1	Q0	Q2 ⁺	Q1 ⁺	Q0 ⁺
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Counter Implemented with D Flip-Flop

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

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

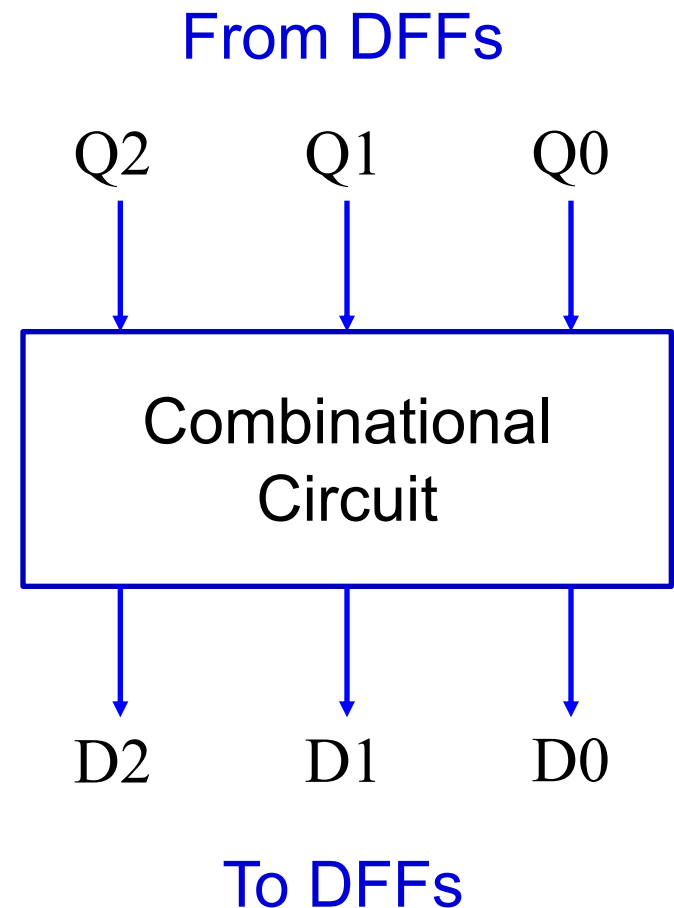


Counter Implemented with D Flip-Flop

- Drop the columns for Next State

Present State (D-FF outputs)			Next State (D-FF inputs)		
Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Truth table inputs Truth table outputs



State Registers Implemented with D Flip-Flop

Karnaugh map for D2:

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

$$\begin{aligned}
 D2 &= Q2Q1' + Q2Q0' + Q2'Q1Q0 \\
 &= Q2(Q1' + Q0') + Q2'Q1Q0 \\
 &= Q2(Q1Q0)' + Q2'(Q1Q0) \\
 &= Q2 \oplus (Q1Q0)
 \end{aligned}$$

Karnaugh map for D1:

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

$$\begin{aligned}
 D1 &= Q1'Q0 + Q1Q0' \\
 &= Q1 \oplus Q0
 \end{aligned}$$

Karnaugh map for D0:

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

$$D0 = Q0'$$

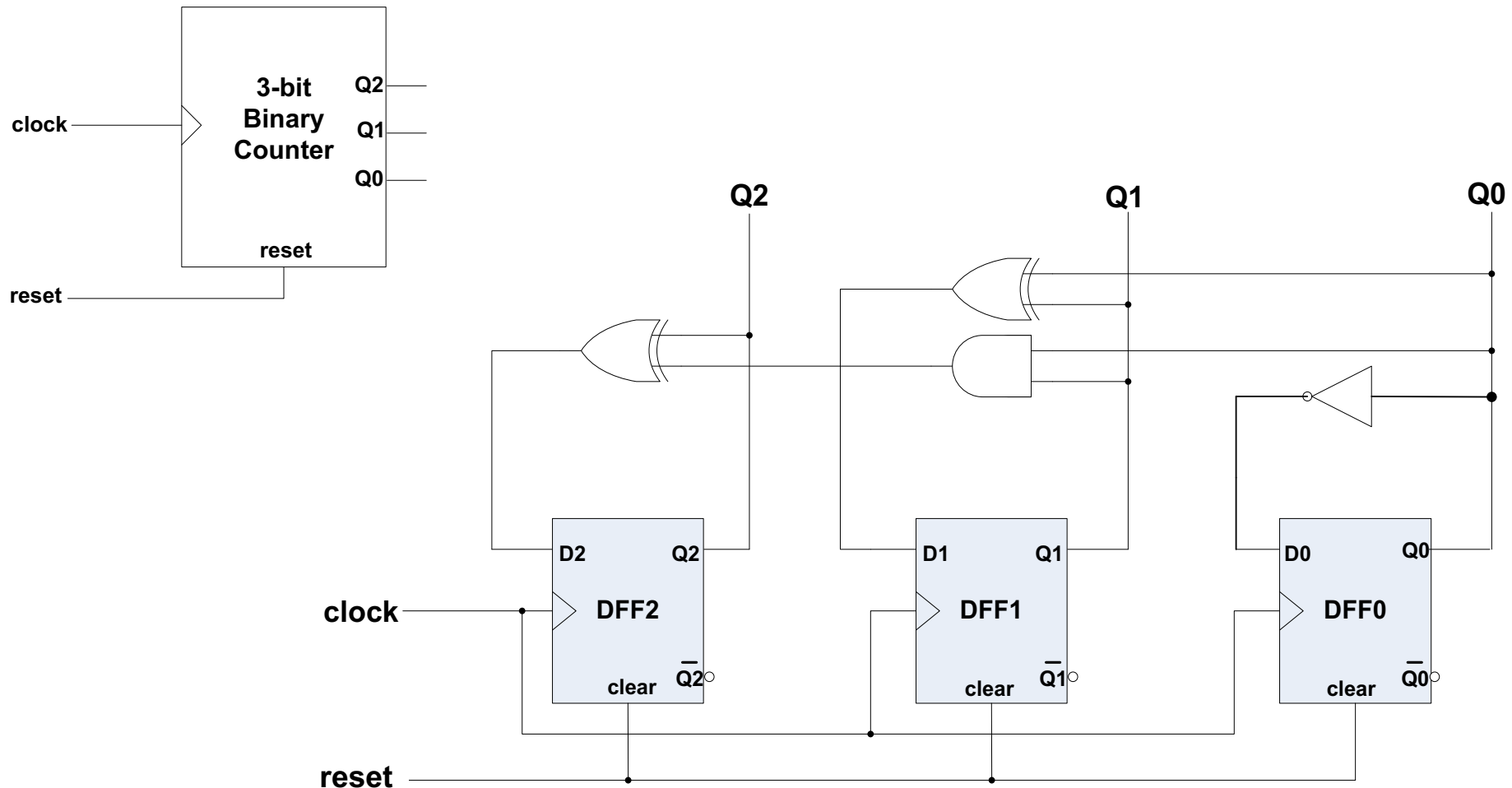
- The input equation can be generalized as

$$Dn = Qn \oplus (Qn-1 \dots Q1Q0)$$

Present State			D flip flop input		
Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

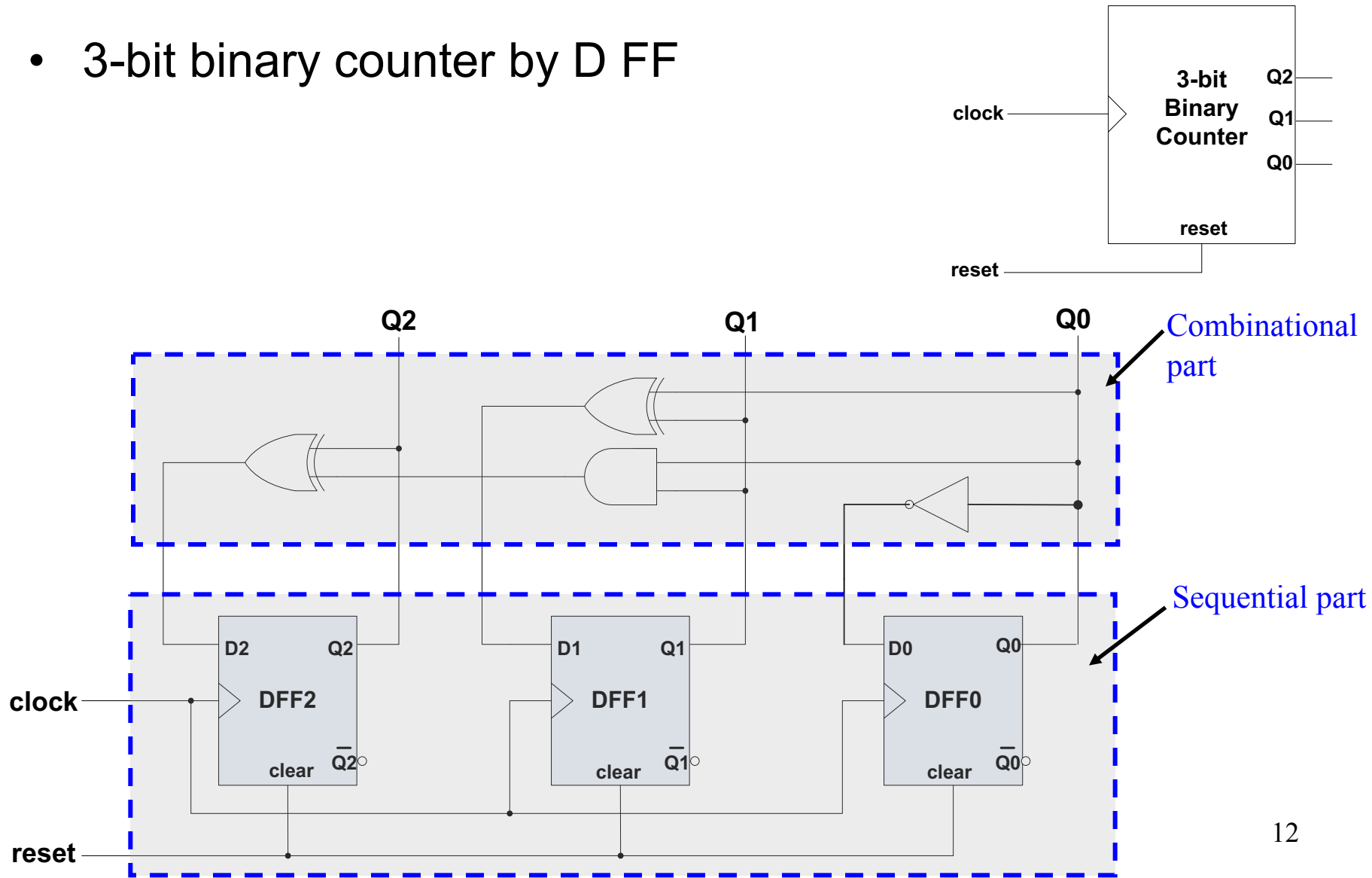
Synchronous Binary Counter with D Flip-Flop

- 3-bit binary counter by D FF



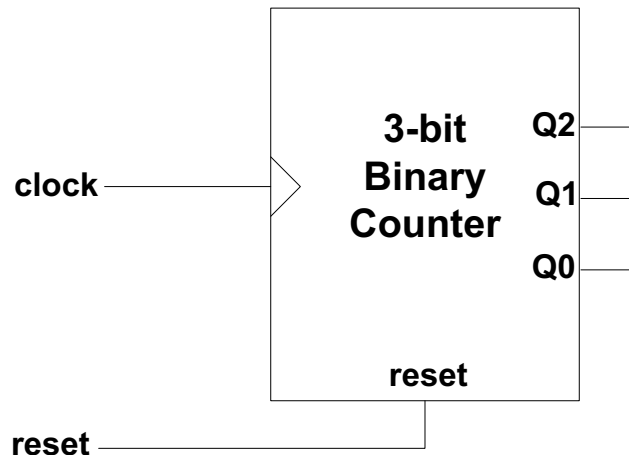
Synchronous Binary Counter with D Flip-Flop

- 3-bit binary counter by D FF



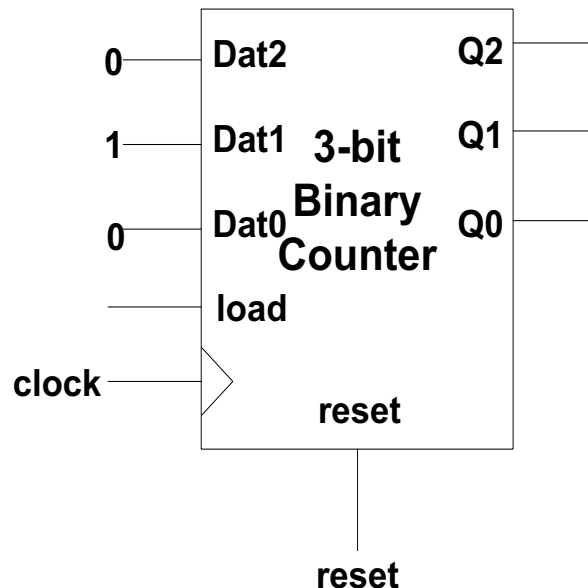
Verilog Model: Synchronous Binary Counter

```
module counter_N_bit (clock, reset, Q);  
  parameter N = 3; ← Defines a constant N  
  input      clock, reset;  
  output     [N-1:0]    Q;  
  reg        [N-1:0]    Q;  
  always @ (posedge reset or posedge clock)  
    if (reset == 1'b1) Q <= 0;  
    else                Q <= Q + 1;  
endmodule
```



Synchronous Counter with Control Input

- Control inputs may be added to the flip flops in a binary counter to control the behavior of the counter
- Load (Parallel Load):** integrate new function: use MUX
 - Numbers can be loaded into the counter anytime when the load input is high, thus the count sequence can be customized

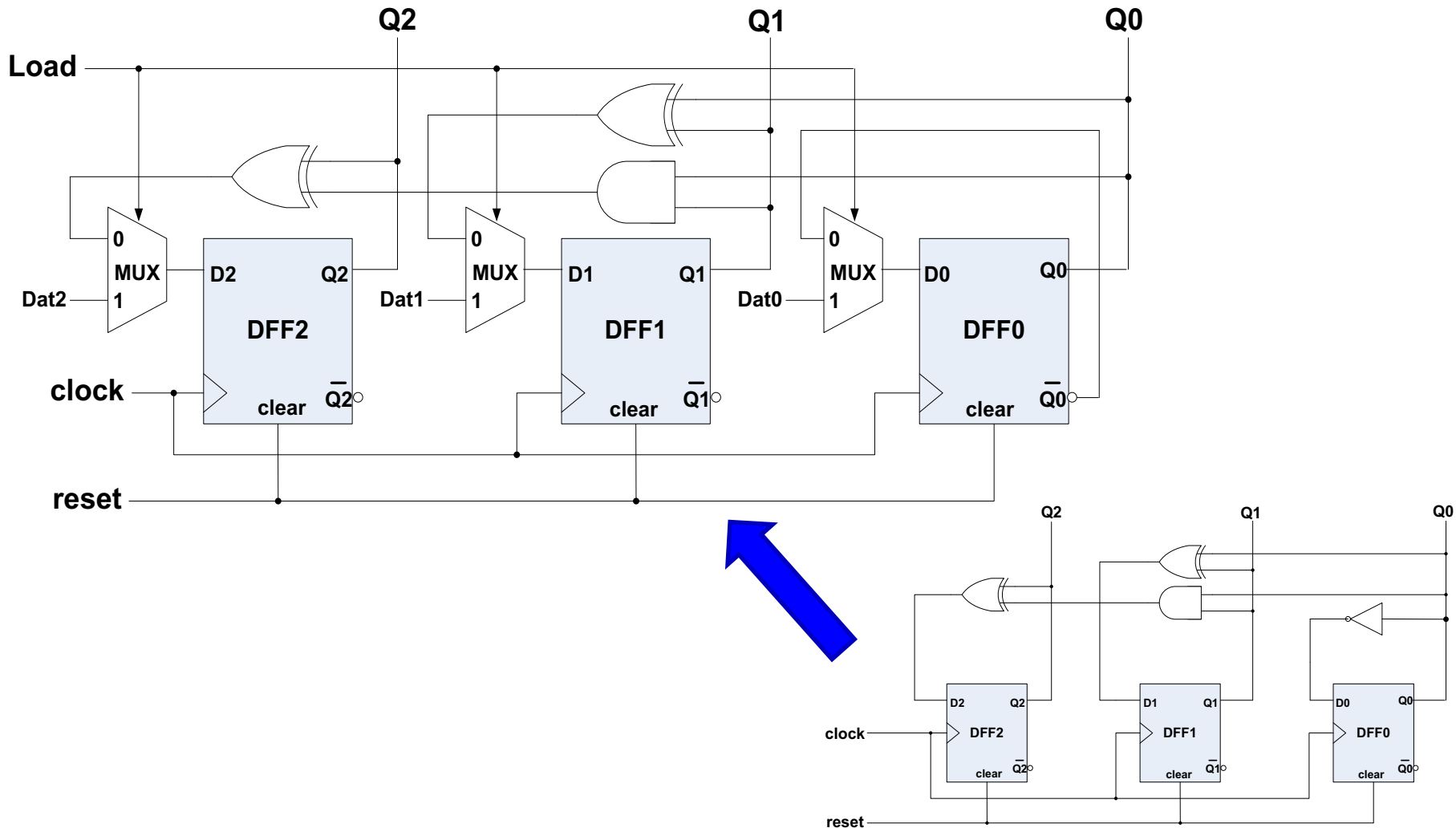


Counting sequence:

010 → ... → 110 → 111 → 000 → 001 ...

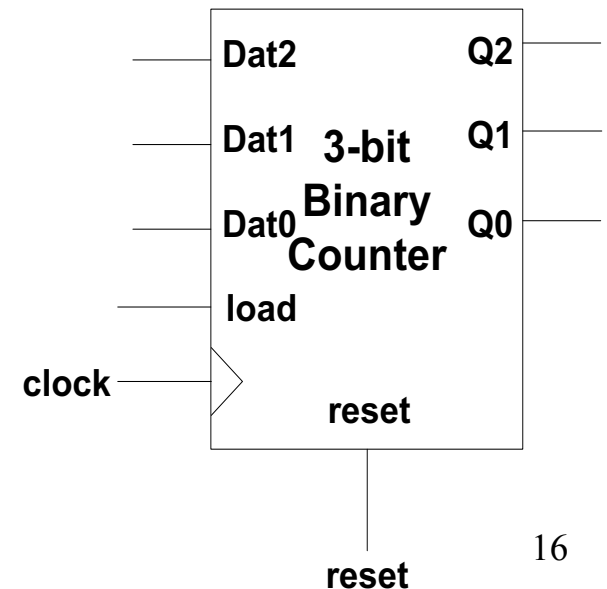
Implementation of Parallel Load

- Using MUXes for the D inputs of DFFs



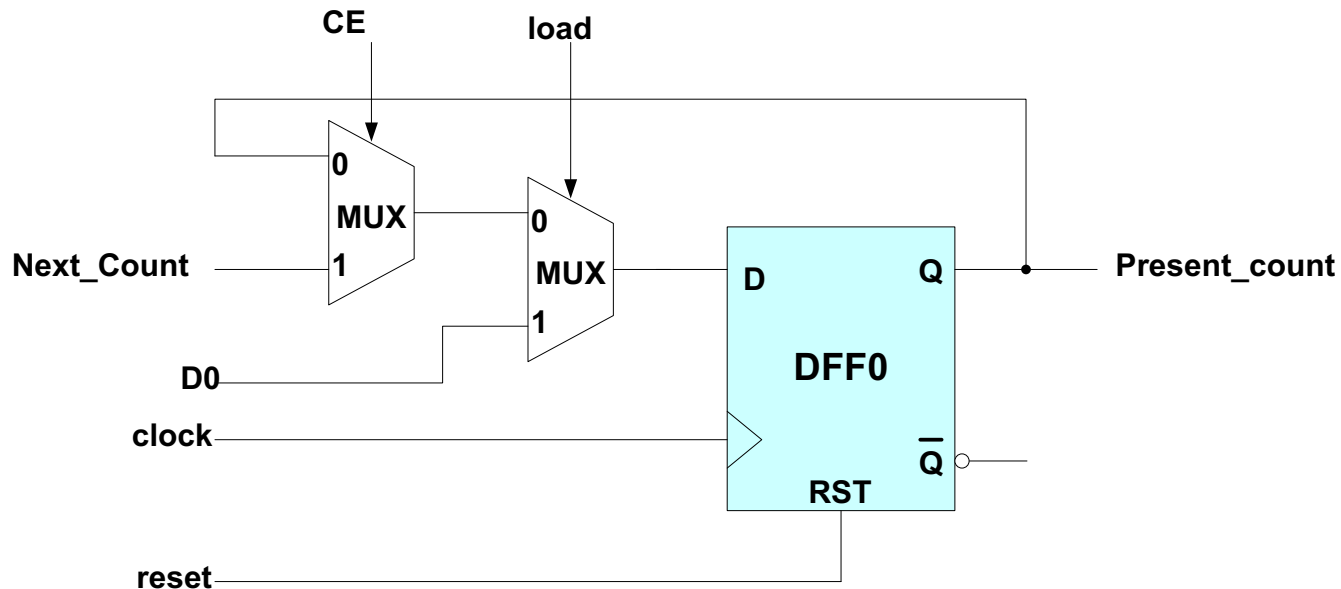
Verilog Model: Counter with Parallel Load

```
module counter_N_bit (clock, reset, load, Dat, Q);  
  parameter N = 3;  
  input      clock, reset, load;  
  input      [N-1:0]    Dat;  
  output     [N-1:0]    Q;  
  reg       [N-1:0]    Q;  
  always @ (posedge reset or posedge clock)  
    if (reset == 1'b1) Q <= 0;  
    else if (load == 1'b1) Q <= Dat;  
    else Q <= Q + 1;  
endmodule
```



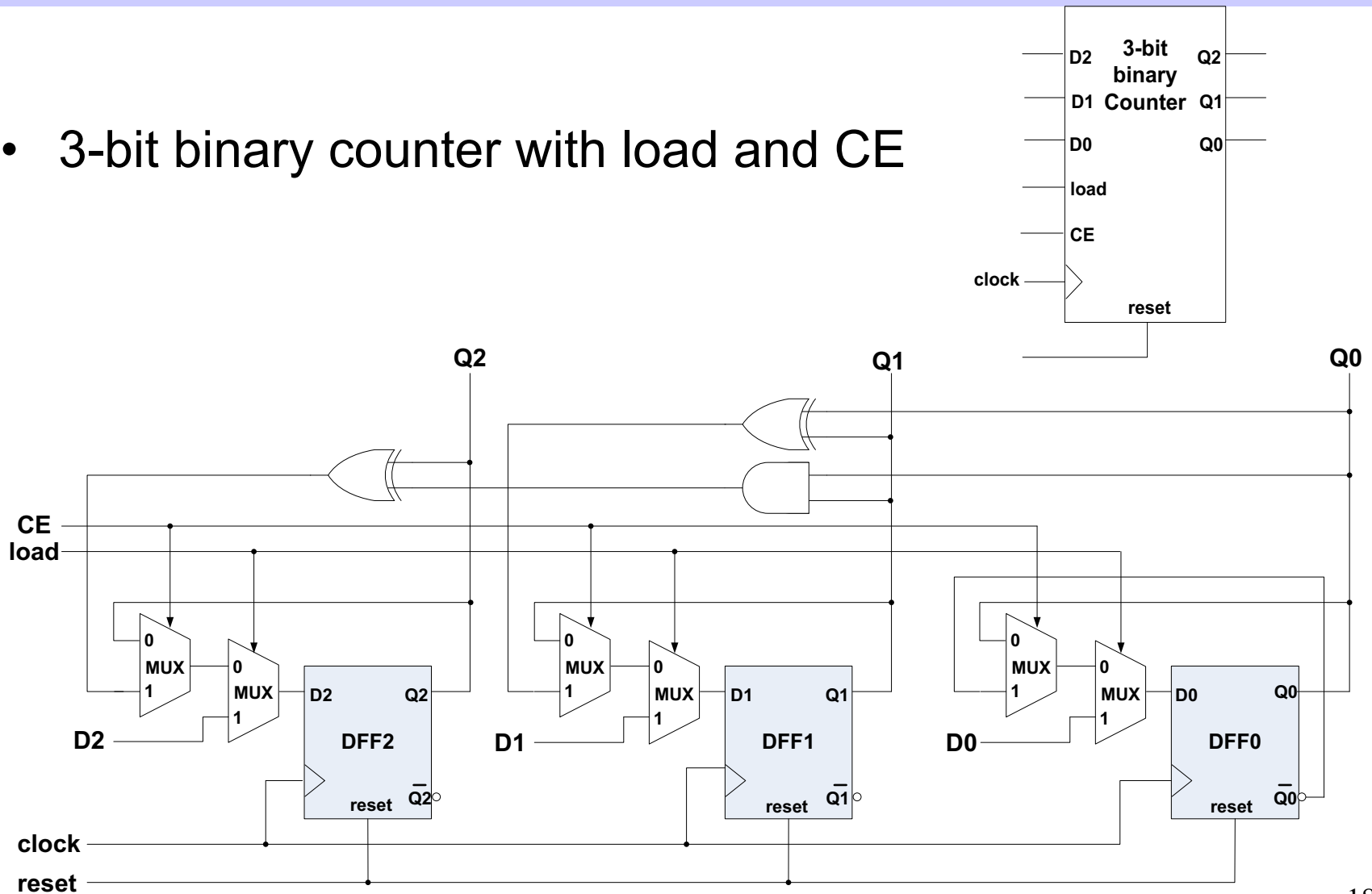
Binary Counter with Count Enable

- **Count Enable (CE) :**
 - when $CE = 1$, counter counts
 - when $CE = 0$, counter holds the values
- Used to hold the counter to
 - Wait certain acknowledge signal coming from other devices
 - Concatenate small counters into bigger ones
- Implementation of 1-bit of a counter with both CE and load:





Binary Counter with CE and Load

- 3-bit binary counter with load and CE



Priority of External Control Signals

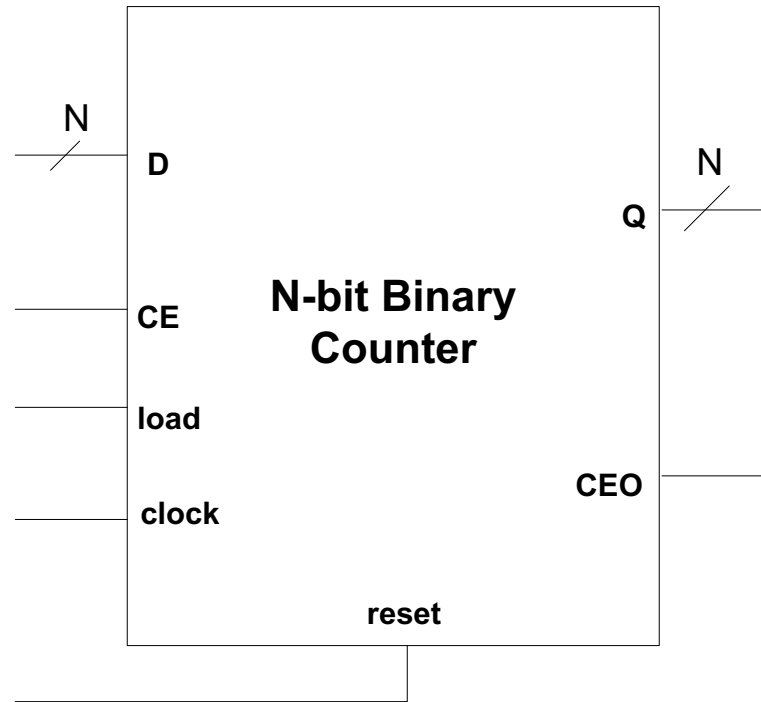
- Function table of a counter with external controls
 - a prioritized hierarchical control structure

reset 	load 	CE	Action on the rising clock edge
1	X	X	Clear ($Q_n \leq 0$)
0	1	X	Load ($Q_n \leq D_n$)
0	0	1	Count
0	0	0	Hold (No Change)

N-bit Binary Counter

- Count Enable Output (CEO)**

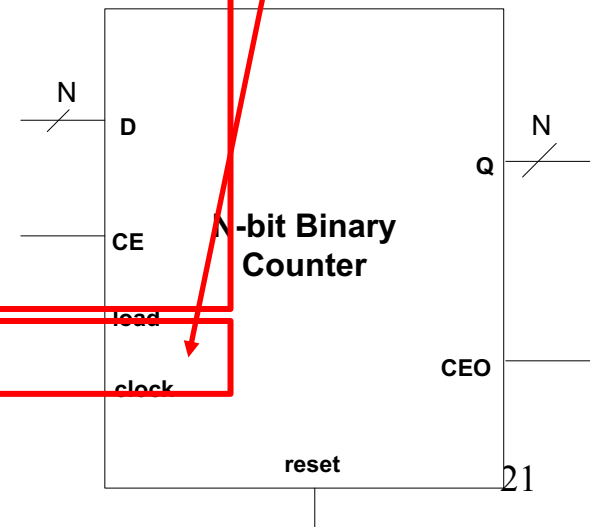
$$\text{CEO} = \text{CE} \cdot Q_{N-1} \cdot Q_{N-2} \cdot \dots \cdot Q_0$$



Verilog Model: Counter with Count Enable

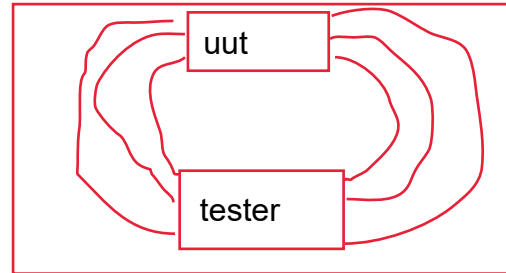
```
module counter_N_bit (clock, reset, load, CE, D, Q, CEO);  
    parameter N = 3;  
    input      clock, reset, load, CE;  
    input      [N-1:0] D;  
    output     [N-1:0] Q;  
    output     CEO;  
    reg        [N-1:0] Q;  
  
    always @ (posedge reset or posedge clock)  
        if (reset == 1'b1) Q <= 0;  
        else if (load == 1'b1) Q <= D;  
        else if (CE == 1'b1) Q <= Q + 1;  
        else Q <= Q;  
  
    assign CEO = CE & (&Q); & all the bits of Q  
endmodule
```

Executed
concurrently



Testbench Written in Verilog

```
module Test_Banch;  
    parameter half_period = 50;  
    parameter counter_size = 4;  
  
    wire [counter_size-1:0] Q;  
    reg [counter_size-1:0] Din;  
    reg clock, load, reset, CE;
```



test bench (can also be tester)

UUT:UNIT UNDER TEST

All executed
concurrently

```
counter_N_bit #(counter_size) UUT (clock,reset,load,CE,Din,Q,CEO);
```

```
initial begin cannot use initial in real verilog files
```

在零时刻

```
#0 clock = 0; Din = 0; load = 0; CE = 1; reset = 1;
```

等100

```
#100 reset = 0;
```

再等200

```
#200 Din = 8; load = 1;
```

```
#100 load = 0;
```

```
#300 CE = 0;
```

```
#200 CE = 1;
```

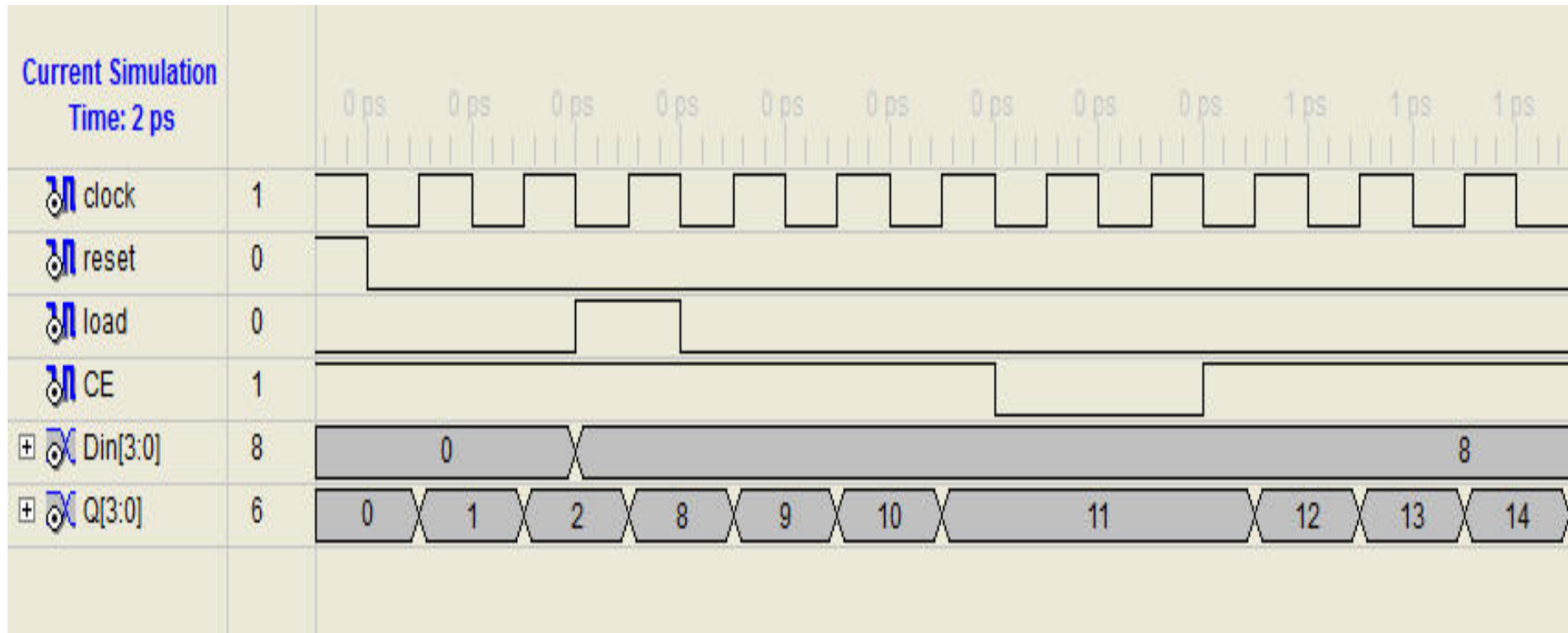
```
end
```

```
always #half_period clock = ~clock;
```

```
initial #2000 $stop; $: system call
```

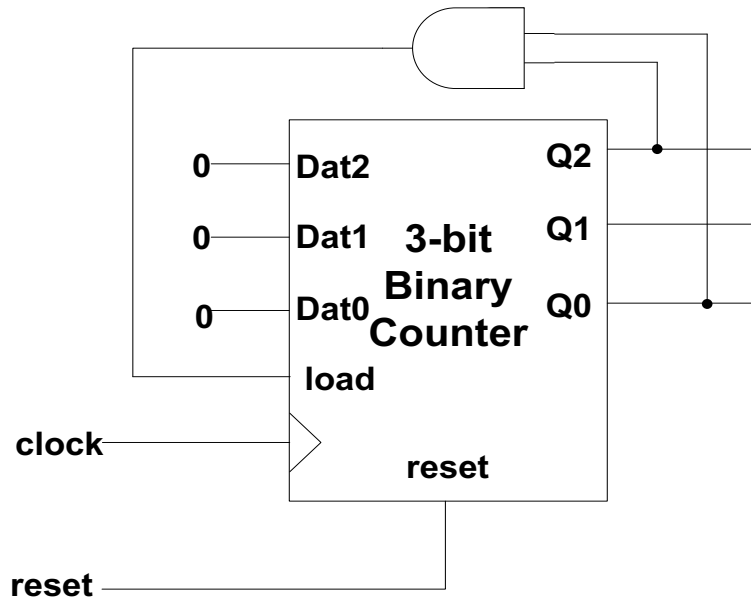
```
endmodule
```

Simulation Result



Customize Counting Sequence

- What does this circuit do?

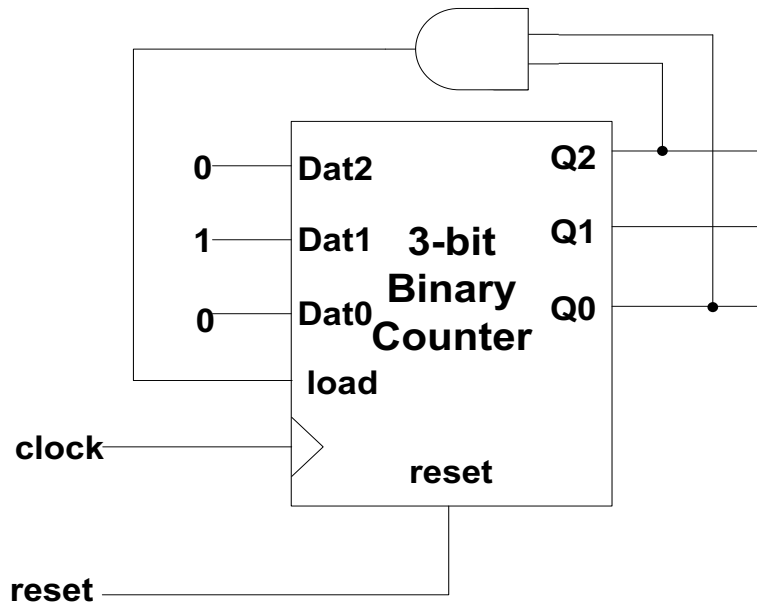


Q2	Q1	Q0	Q2 ⁺	Q1 ⁺	Q0 ⁺
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	X		
1	1	1			

- load = Q2Q0, initial number = 000
- The count sequence now becomes
000 → 001 → ... → 101 → 000 (modulo-6 counter)

Customize Counting Sequence

- What does this do?



Q2	Q1	Q0	Q2 ⁺	Q1 ⁺	Q0 ⁺
0	0	0	X		
0	0	1			
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	X		
1	1	1			

- load = Q2Q0, initial number = 010
- The count sequence is

$000 \rightarrow \dots \rightarrow 101 \rightarrow 010 \rightarrow 011 \rightarrow \dots \rightarrow 101 \rightarrow 010 \rightarrow \dots$

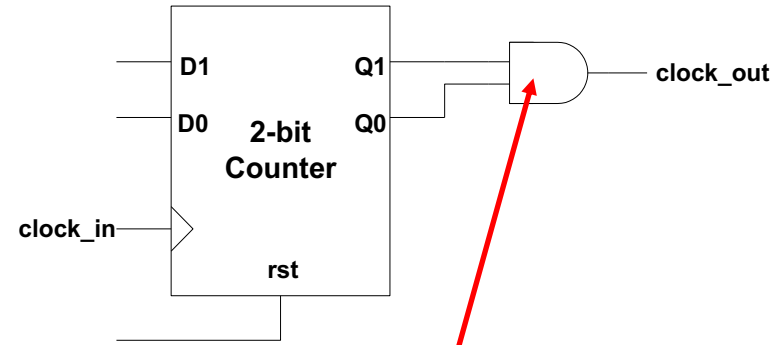
 In first cycle

An Application of Counter: Clock Divider

- Slows a clock down by reducing its frequency
 - Generates clock signal with bigger clock cycle
 - Input clock can be slowed by odd or even number of times
 - For slower devices and lower power consumption
- Generates slower pulses
 - It counts the number of active edges of the input clock signal until the desired number is reached
 - Then it produces output
 - Toggles the output once, or
 - Generates a pulse
 - Starts counting all over again

Example: Divide by 4

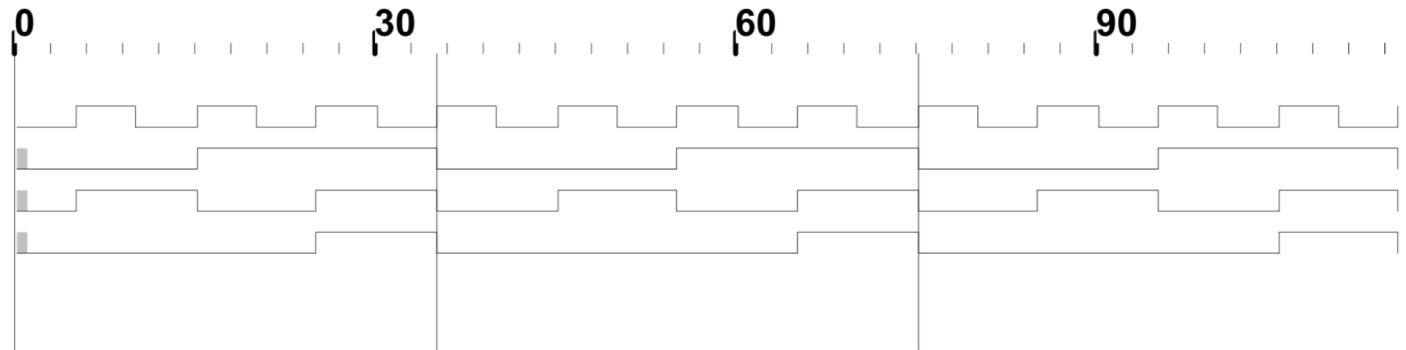
- 4-fold clock divider: 2-bit binary counter
- The clock_out generates one pulse for every four input clock cycles.
 - So the frequency of the clock_in is reduced by 4 times
- Q1 serves the same purpose with 50% duty cycle



Ideal case, delay ignored

T1 35 T2 75 Tdelta 40

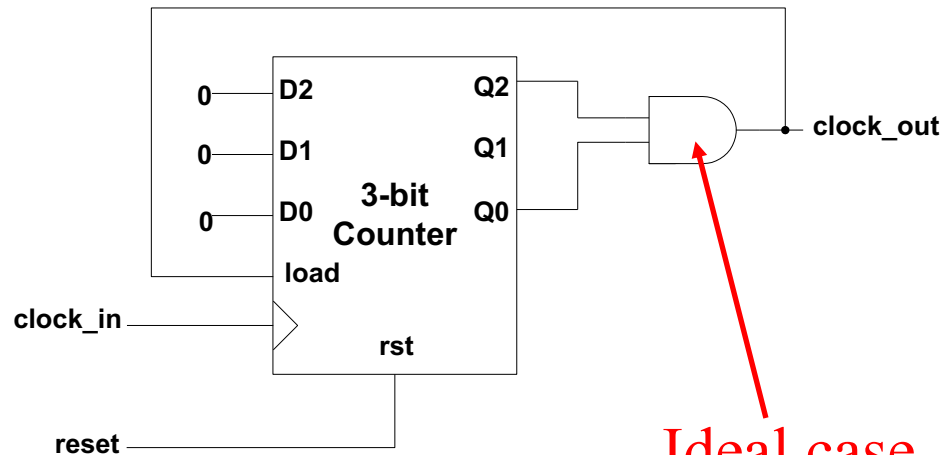
Name
clock_divider
clock_in
Q1
Q0
clock_out



Both have the same slower frequency

Example: Divide by 6

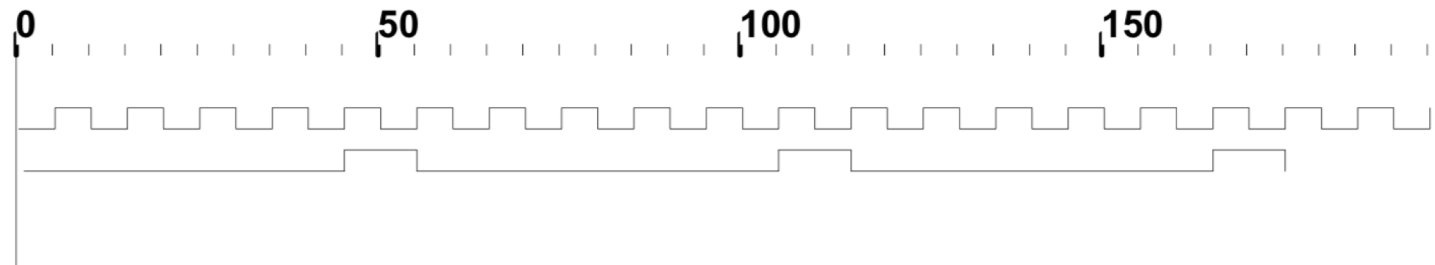
- $2^2 < 6 < 2^3$, at least a 3-bit counter
- Counting sequence: $000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 000$
- The output should be a logic “1” whenever Q2 and Q0 are high
 - $\text{clock_out} = \text{load} = Q2 \ \& \ Q0$



Ideal case, delay ignored

T1 T2 Tdelta

Name
clock_divider
clock_in
clock_out



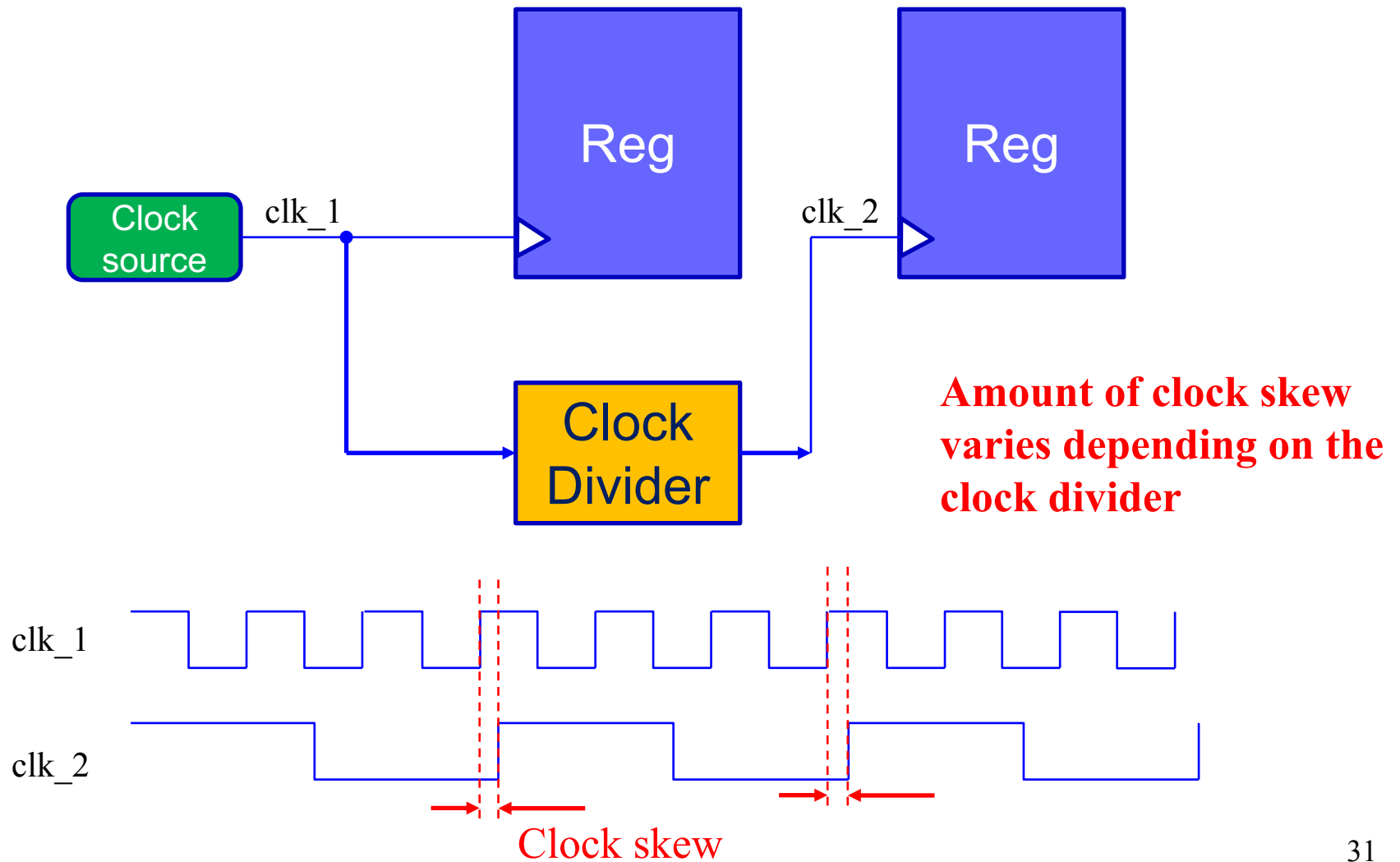
Divide by N

- In general, for N-fold Clock Divider
 - $2^{n-1} < N < 2^n$, where n is the size of counter
 - $N-1$ should be the upper bound of the counter's counting sequence, i.e.
 $0 \rightarrow 1 \rightarrow \dots \rightarrow N-1 \rightarrow 0$
 - Thus the counter should be force back to 0 using load input when it reaches the number $N-1$
 - If B_{N-1} is the binary equivalent of decimal number $N-1$, both load and output clock can be formed by ANDing the bits of '1's in B_{N-1}

Gated Clock

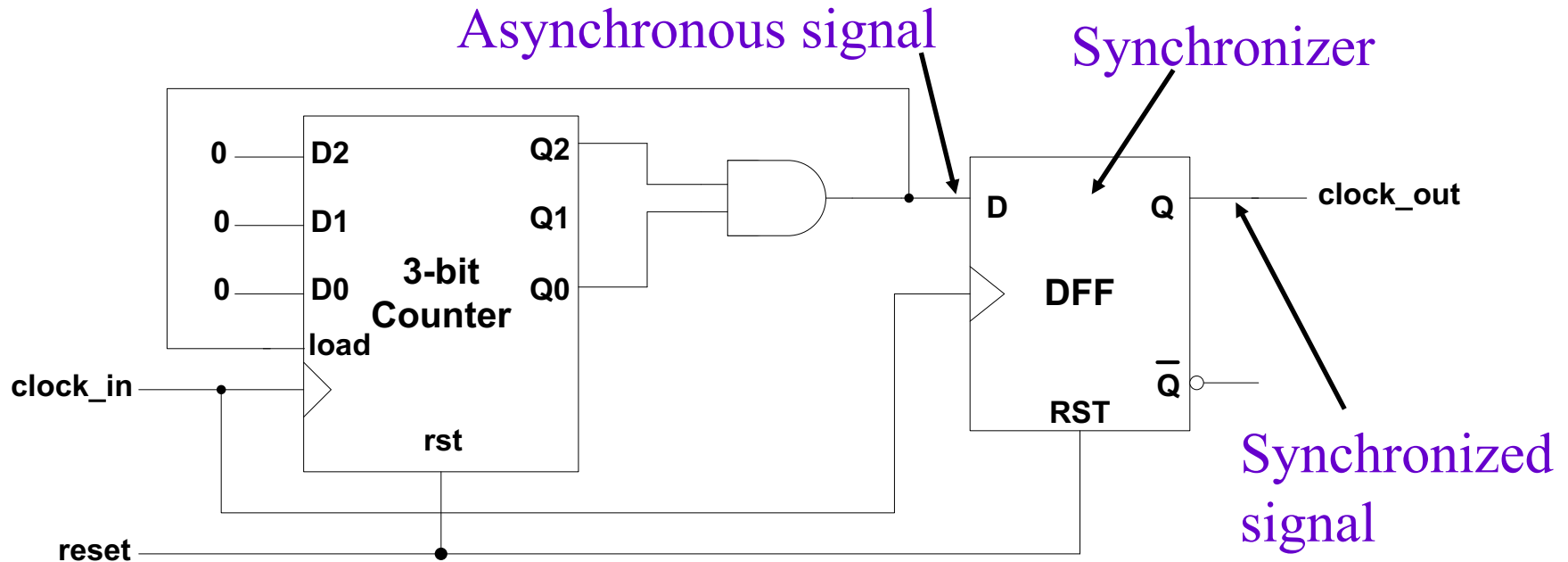
- Propagation delay of the output clock:
 - The output of the clock dividers comes from a combinational logic circuit (single AND gate or a net of gates in multiple levels)
 - The output will be delayed due to the propagation delay of the gates
 - Amount of delay changes depending on the combinational circuit
 - This kind of output is called **asynchronous output or unregistered output or gated output**
 - Asynchronous output may cause timing issues if the output is used as a synchronizing signal (i.e. clock)
 - **Clock skew: the difference in the arrival time of clock signal between *two sequentially-adjacent registers*** (wikipedia)

Gated Clock



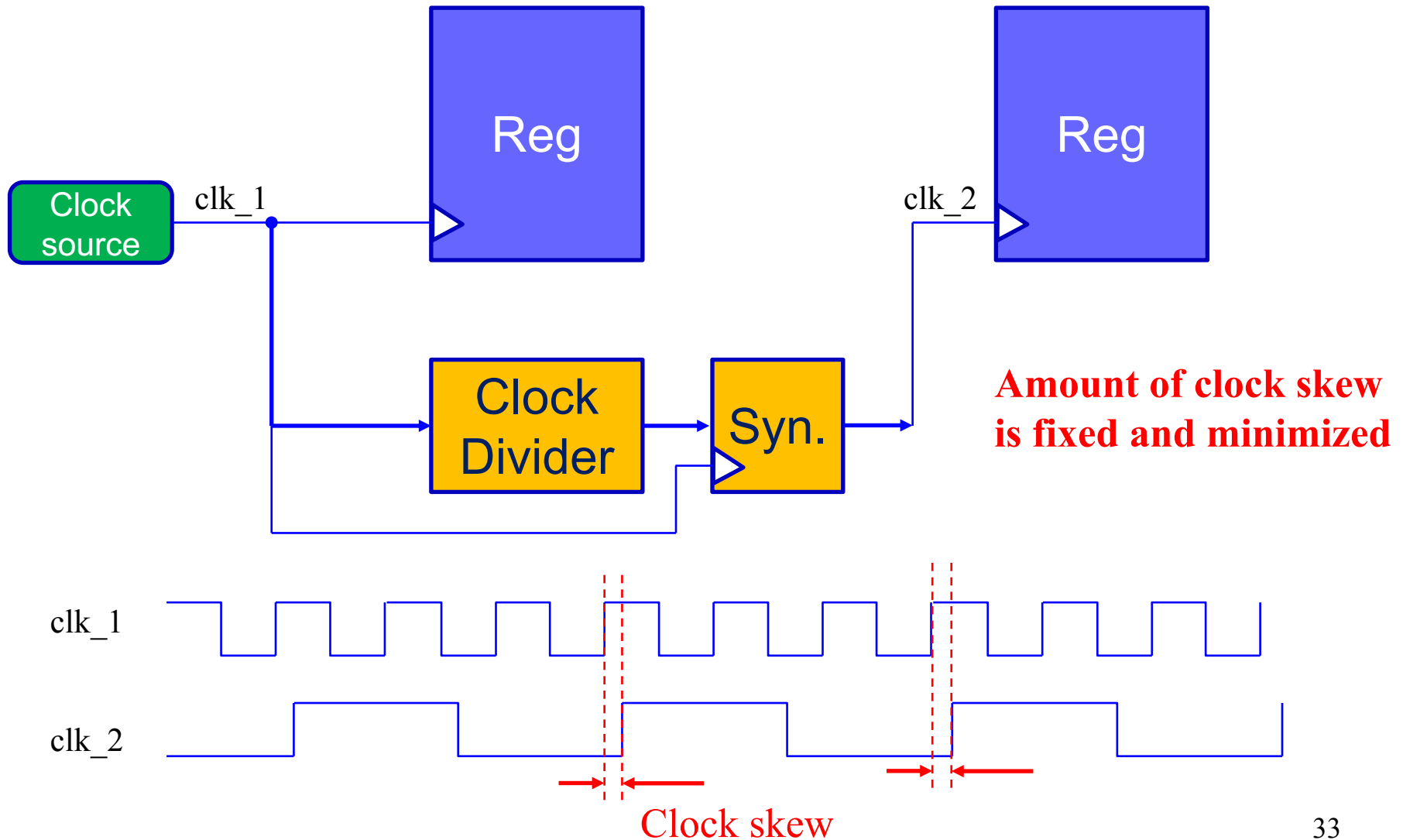
Synchronizer

- One possible solution to reduce clock skew is to synchronize the output by a D flip-flop



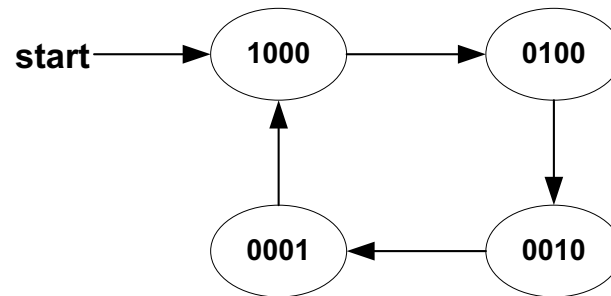
- Synchronizer **delays** the input asynchronous signal by **up to 1 clock cycle**
 - But aligns the synchronized signal with clock
 - Clock skew is not completely removed, only **reduced**

Synchronized Clock



Ring Counter

- A ring counter is a circular shifter with only one FF being set at any time



Current				Next			
1	0	0	0	0	1	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	0	0	1
0	0	0	1	1	0	0	0
all other inputs				X			

D3 = Q0

D2 = Q3

D1 = Q2

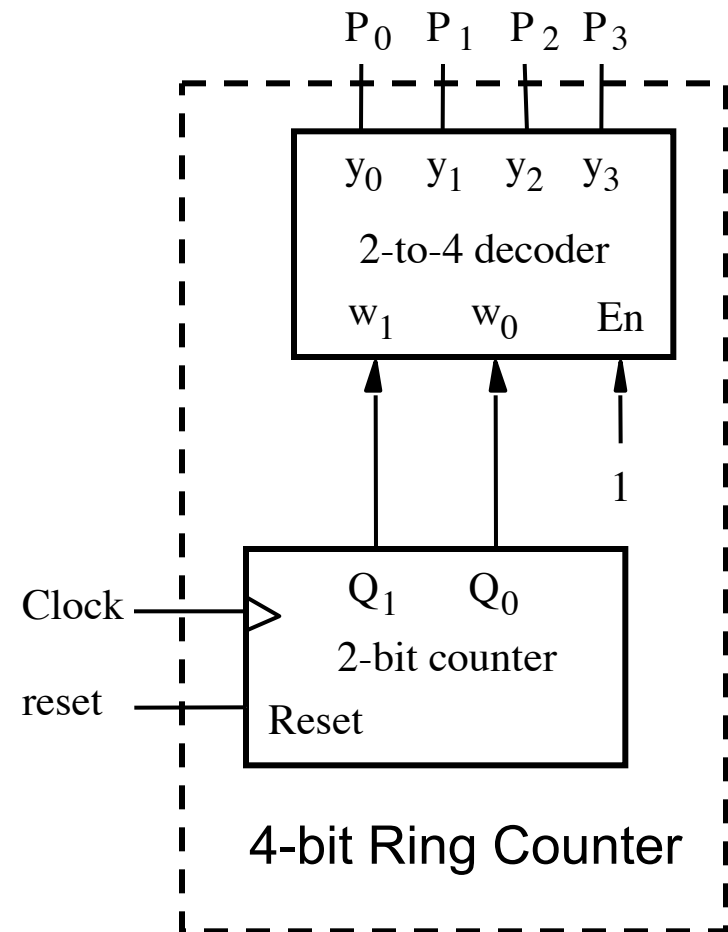
D0 = Q1

Ring Counter – Design Alternative

- A ring counter is a circular shift register with only one FF being set at any time

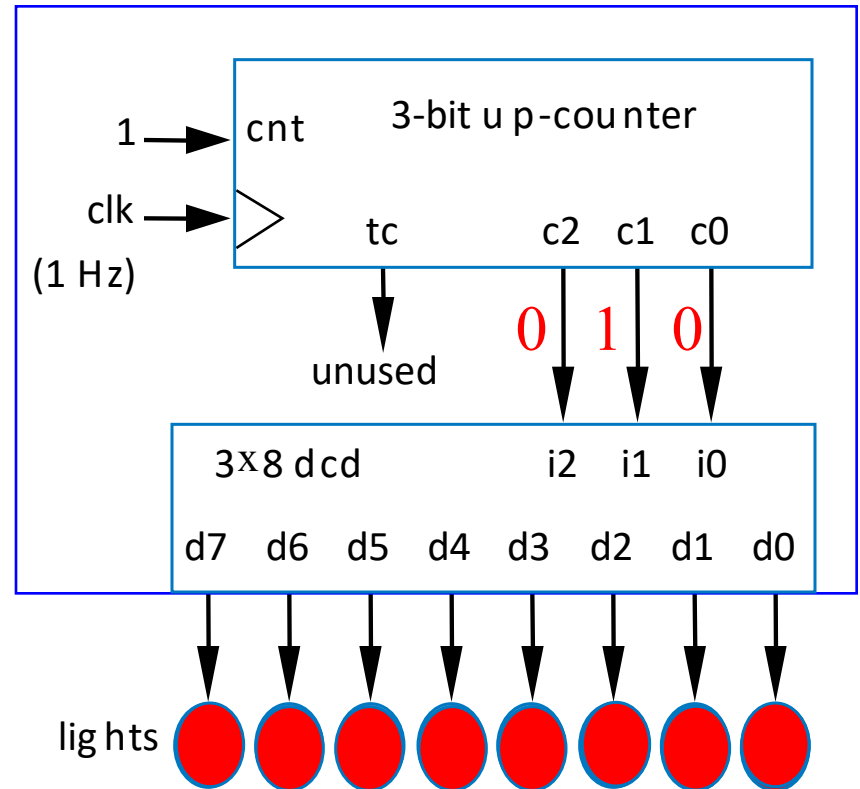
w_1	w_0	P_0	P_1	P_2	P_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

functional table of 2×4 Decoder



Ring Counter Example: Light Sequencer

- Illuminate 8 lights from right to left, one at a time, one per second
- Use 3-bit up-counter to count from 0 to 7
- Use 3x8 decoder to illuminate appropriate light



Johnson Counter

- Counting sequence
 - $1000 \rightarrow 1100 \rightarrow 1110 \rightarrow 1111 \rightarrow 0111 \rightarrow 0011 \rightarrow 0001 \rightarrow 0000 \rightarrow \dots$
- In each state transition, only one bit has to be changed
 - less state transition effort
 - but less counting state too

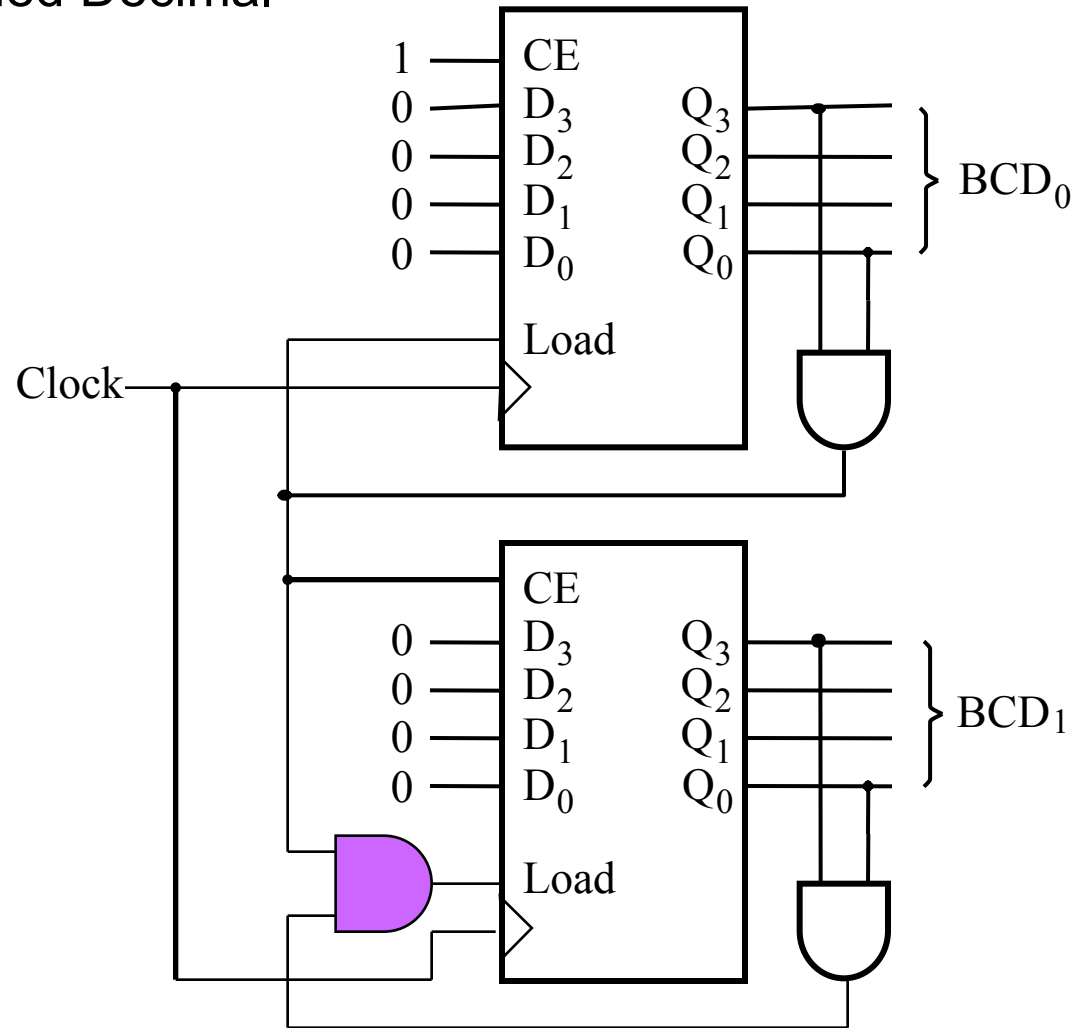
BCD Counter

- A counter doesn't have to follow through the entire counting sequence
- Example: BCD (Binary Coded Decimal) counter

Q3	Q2	Q1	Q0	Q3 ⁺	Q2 ⁺	Q1 ⁺	Q0 ⁺
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X			
	.	.	.				
	.	.	.				
1	1	1	1				

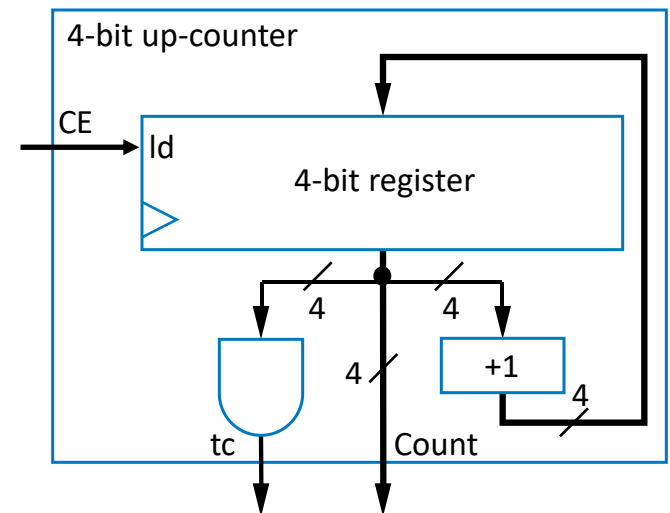
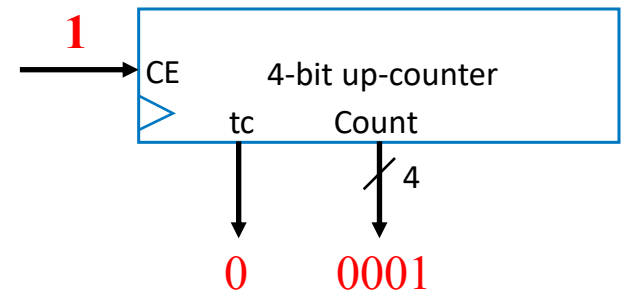
2-Digit BCD Counter

- BCD: Binary Coded Decimal



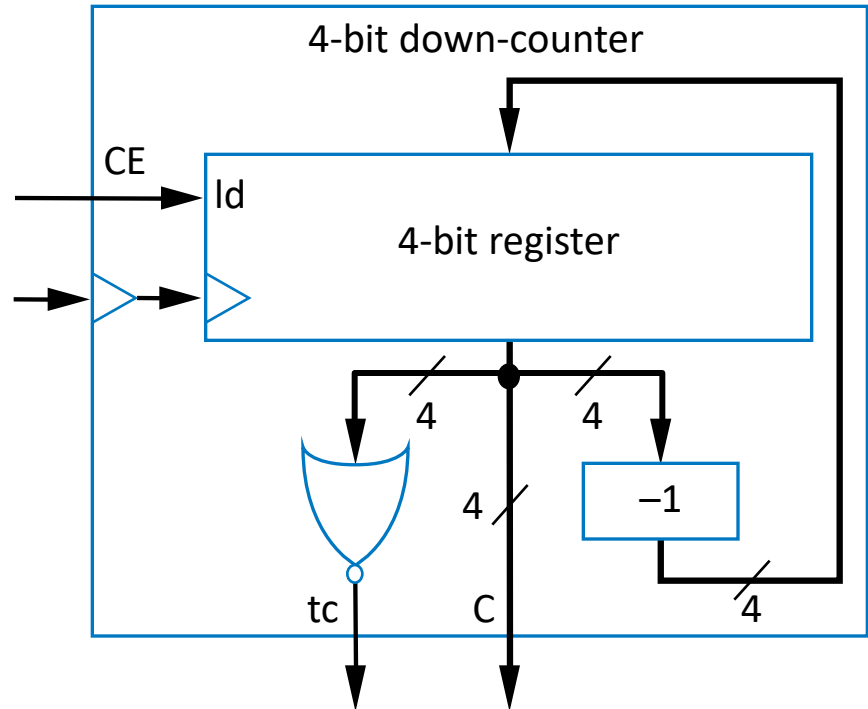
Alternative Design of Binary Counter

- Higher level (Register Transfer Level – RTL) design with combinational and sequential building blocks
 - Register
 - Incrementer
 - N-input AND gate to detect terminal count (TC), TC=1 when terminal number is reached
- Example: up counter



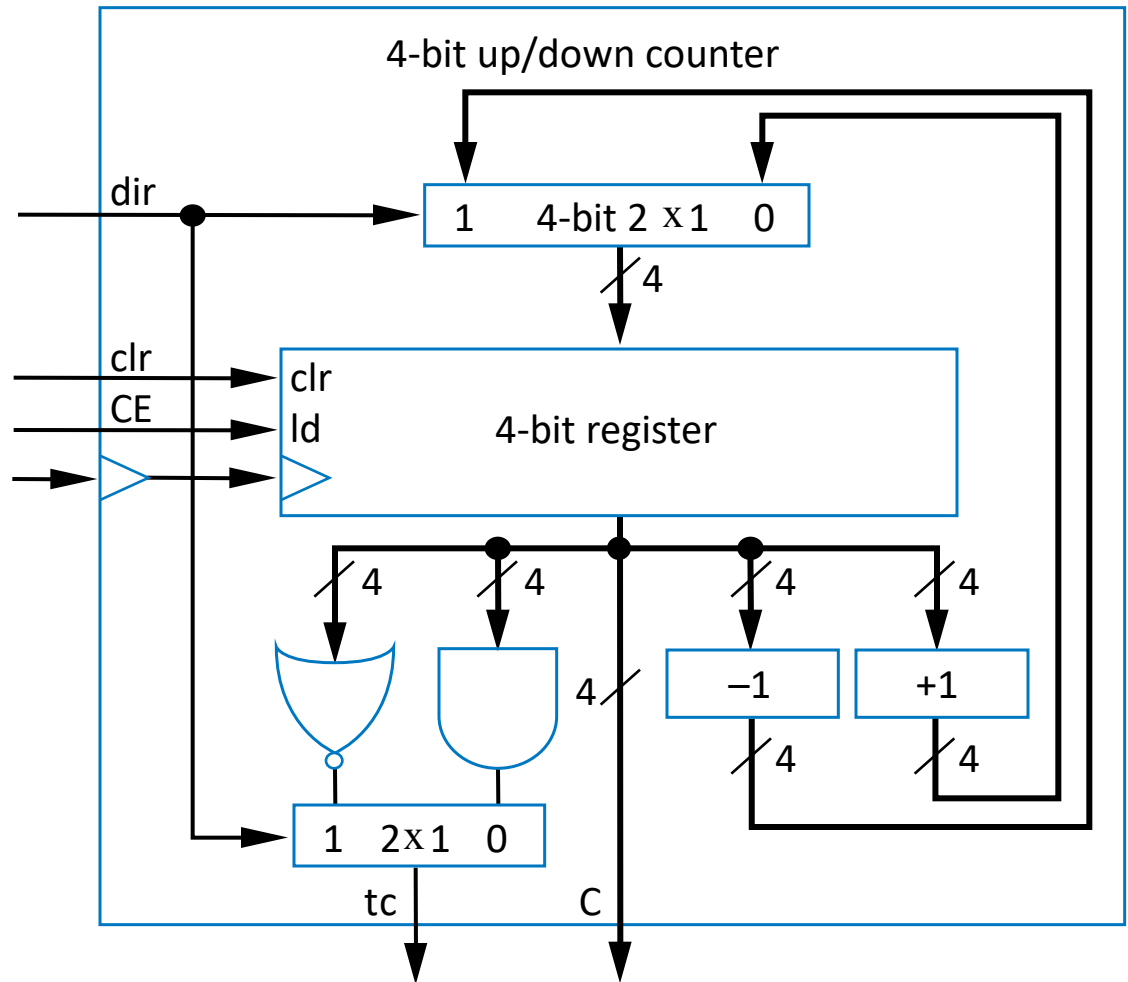
Down-Counter

- 4-bit down-counter
 - Terminal count is 0000
 - Use NOR gate for tc
 - Need decrementer (-1)



Up/Down-Counter

- Can count either up or down
 - Includes both incrementer and decrementer
 - Use dir input to select, using 2x1 MUX, dir=0 means count up
 - Likewise, dir selects appropriate terminal count value



Alternative Design of Counter with Control

- Up-counter that can be loaded with external value
 - Designed using 2x1 mux
 - Id input selects incremented value or external value
 - Load the internal register when loading external value or when counting

