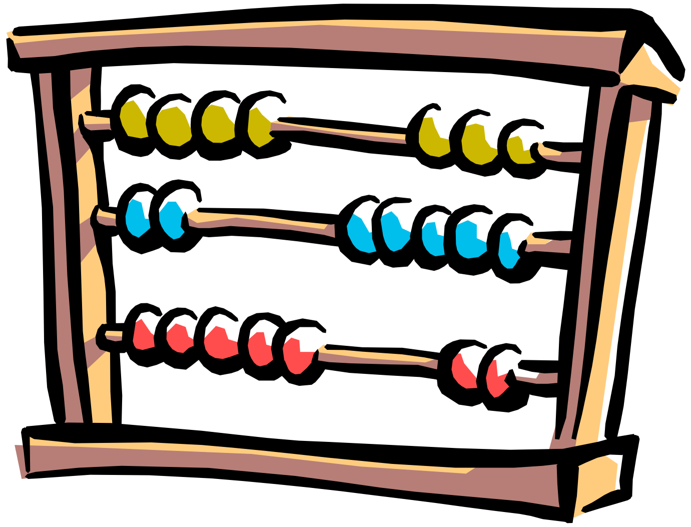
Counters



- We'll look at different kinds of counters and discuss how to build them
- These are not only examples of sequential analysis and design, but also real devices used in larger circuits

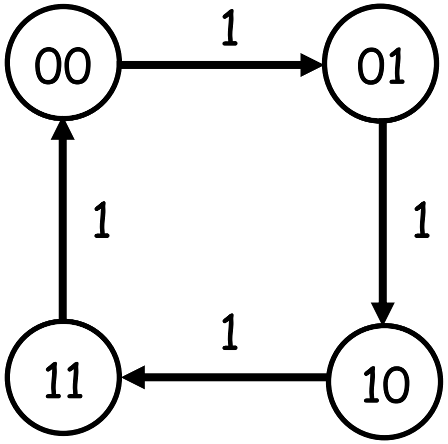
Introducing counters

- Counters are a specific type of sequential circuit
- The state serves as the “output” (Moore)
- A counter that follows the binary number sequence is called a binary counter
  - n-bit binary counter: n flip-flops, count in binary from 0 to 2^n-1
- Counters are available in two types:
  - Synchronous Counters
  - Ripple Counters
- Synchronous Counters:
  - A common clock signal is connected to the C input of each flip-flop

Synchronous Binary Up Counter

- The output value increases by one on each clock cycle
- After the largest value, the output “wraps around” back to 0
- Using two bits, we’d get something like this:

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

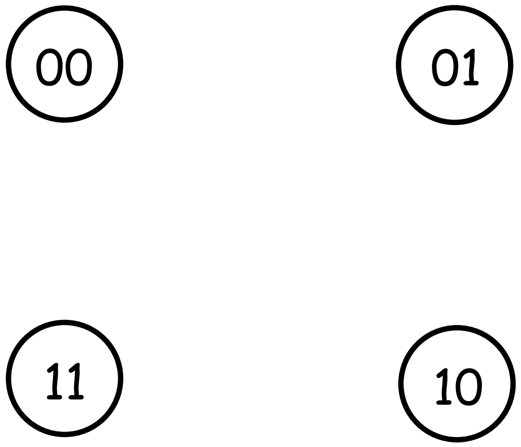


What good are counters?

- Counters can act as simple clocks to keep track of "time"
- You may need to record how many times something has happened
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?
- All processors contain a program counter, or PC
  - Programs consist of a list of instructions that are to be executed one after another (for the most part)
  - The PC keeps track of the instruction currently being executed
  - The PC increments once on each clock cycle, and the next program instruction is then executed.

Synch Binary Up/Down Counter

- 2-bit Up/Down counter
  - Counter outputs will be 00, 01, 10 and 11
  - There is a single input, X.
    - X= 0, the counter counts up
    - X= 1, the counter counts down
- We'll need two flip-flops again. Here are the four possible states:



The complete state diagram and table

- Here’s the complete state diagram and state table for this circuit

$$\begin{tikzcd}
00 \arrow[r, shift left, "0"] \arrow[d, shift left, "1"] & 01 \arrow[l, shift left, "1"] \arrow[d, shift left, "0"] \\
11 \arrow[u, shift left, "0"] \arrow[r, shift left, "1"] & 10 \arrow[l, shift left, "0"] \arrow[u, shift left, "1"]
\end{tikzcd}$$

Present State		Inputs	Next State	
Q_1	Q_0	X	Q_1	Q_0
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

D flip-flop inputs

- If we use D flip-flops, then the D inputs will just be the same as the desired next states
- Equations for the D flip-flop inputs are shown at the right
- Why does $D_0 = Q_0'$ make sense?

Present State		Inputs	Next State	
Q_1	Q_0	X	Q_1	Q_0
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

$$\begin{array}{c|c|c|c|c|}
 & & & \multicolumn{2}{c}{Q_0} \\
\hline
 & 0 & 1 & 0 & 1 \\
\hline
Q_1 & 1 & 0 & 1 & 0 \\
\hline
 & & \multicolumn{2}{c}{X} &
\end{array}$$

$$D_1 = Q_1 \oplus Q_0 \oplus X$$

$$\begin{array}{c|c|c|c|c|}
 & & & \multicolumn{2}{c}{Q_0} \\
\hline
 & 1 & 1 & 0 & 0 \\
\hline
Q_1 & 1 & 1 & 0 & 0 \\
\hline
 & & \multicolumn{2}{c}{X} &
\end{array}$$

$$D_0 = Q_0'$$

JK flip-flop inputs

- If we use JK flip-flops instead, then we have to compute the JK inputs for each flip-flop
- Look at the present and desired next state, and use the excitation table on the right

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Present State		Inputs	Next State		Flip flop inputs			
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	0	1	0	x	1	x
0	0	1	1	1	1	x	1	x
0	1	0	1	0	1	x	x	1
0	1	1	0	0	0	x	x	1
1	0	0	1	1	x	0	1	x
1	0	1	0	1	x	1	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	0	x	0	x	1

JK flip-flop input equations

Present State		Inputs	Next State		Flip flop inputs			
Q_1	Q_0	X	Q_1	Q_0	J_1	K_1	J_0	K_0
0	0	0	0	1	0	x	1	x
0	0	1	1	1	1	x	1	x
0	1	0	1	0	1	x	x	1
0	1	1	0	0	0	x	x	1
1	0	0	1	1	x	0	1	x
1	0	1	0	1	x	1	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	0	x	0	x	1

- We can then find equations for all four flip-flop inputs, in terms of the present state and inputs. Here, it turns out $J_1 = K_1$ and $J_0 = K_0$

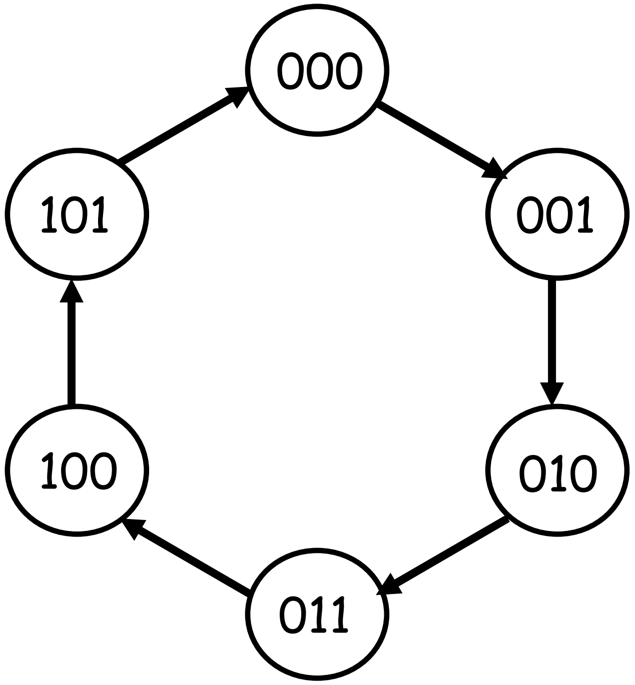
$$J_1 = K_1 = Q_0' X + Q_0 X'$$
$$J_0 = K_0 = 1$$

- Why does $J_0 = K_0 = 1$ make sense?

Unused states

- The examples shown so far have all had 2^n states, and used n flip-flops. But sometimes you may have unused, leftover states
- For example, here is a state table and diagram for a counter that repeatedly counts from 0 (000) to 5 (101)
- What should we put in the table for the two unused states?

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
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	?	?	?
1	1	1	?	?	?

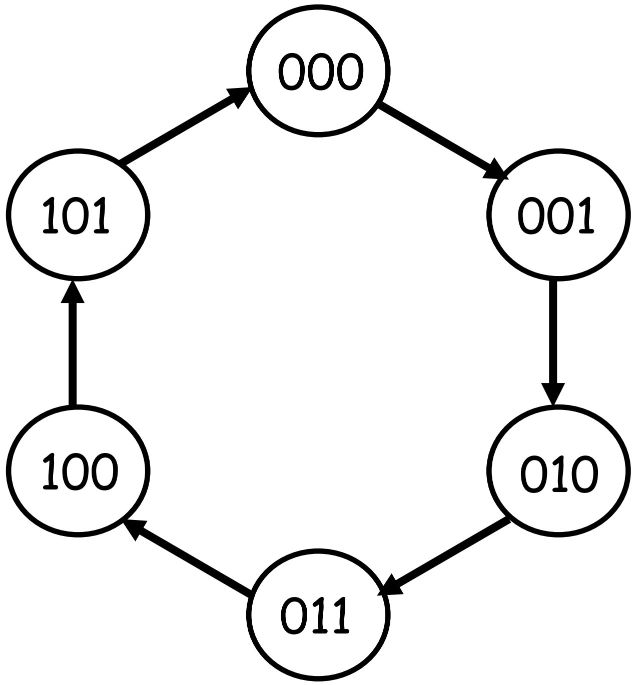


Unused states can be don't cares…

- To get the *simplest* possible circuit, you can fill in don't cares for the next states. This will also result in don't cares for the flip-flop inputs, which can simplify the hardware

- If the circuit somehow ends up in one of the unused states (110 or 111), its behavior will depend on exactly what the don't cares were filled in with

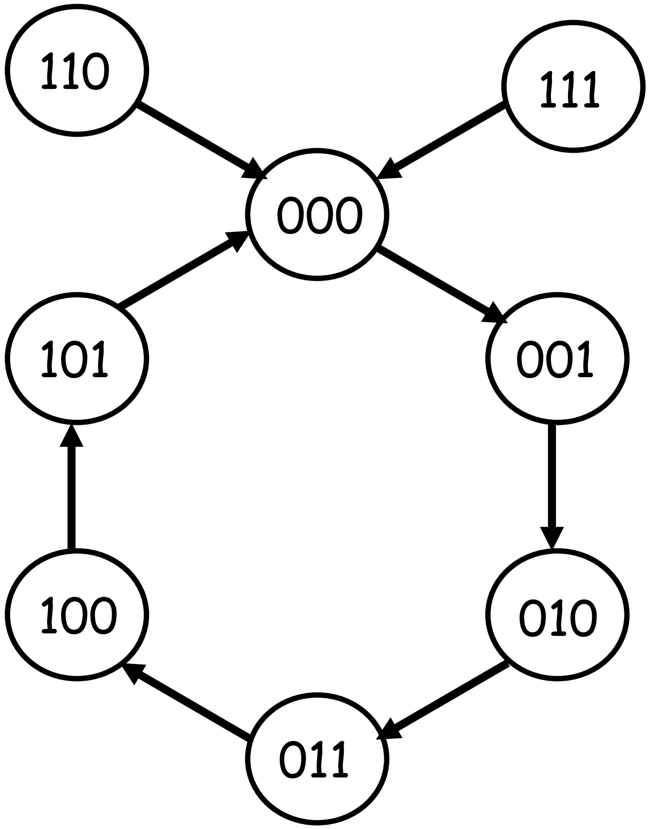
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	x	x	x
1	1	1	x	x	x



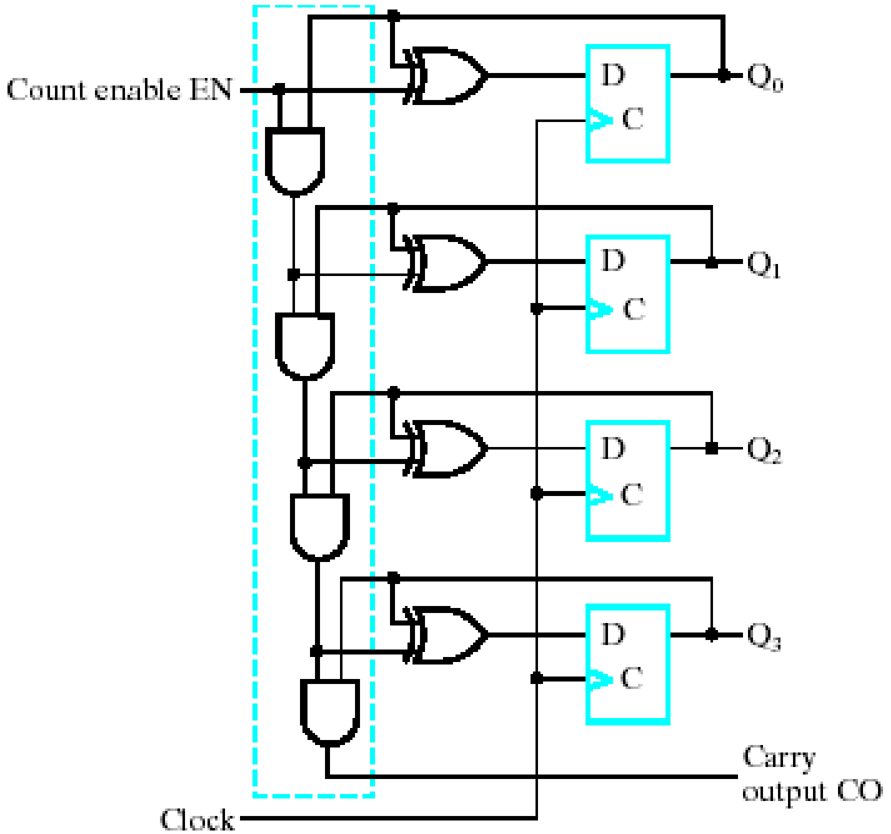
…or maybe you *do* care

- To get the *safest* possible circuit, you can explicitly fill in next states for the unused states 110 and 111
- This guarantees that even if the circuit somehow enters an unused state, it will eventually end up in a valid state
- This is called a self-starting counter

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
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	0	0	0
1	1	1	0	0	0



4-bit Counter with Serial Gating



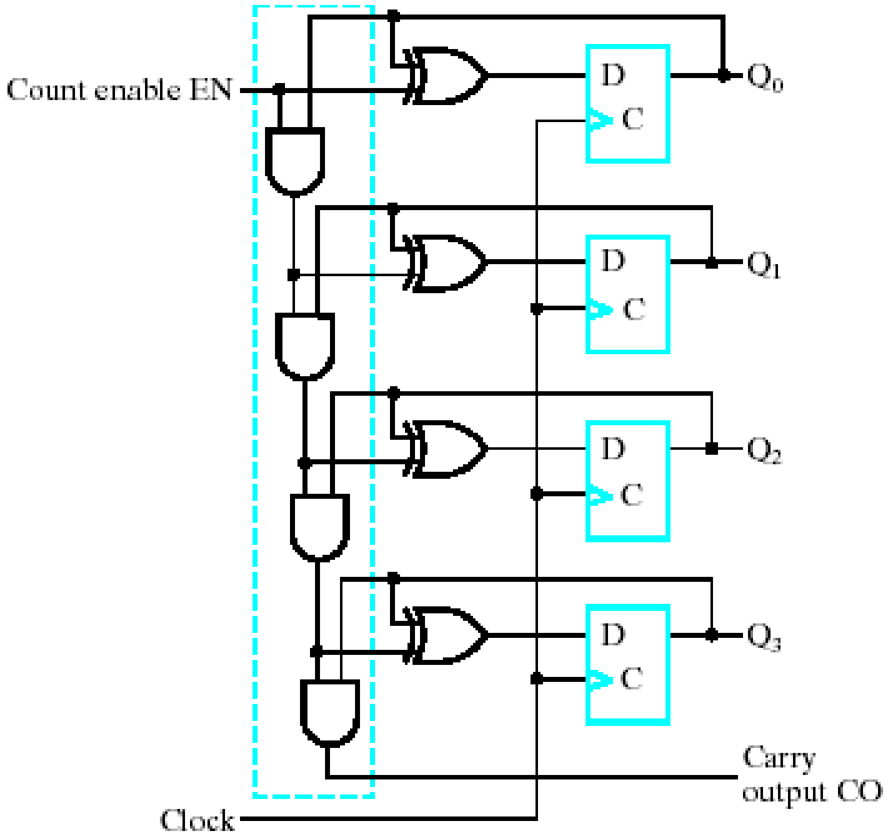
(a) Logic Diagram-Serial Gating

CO= 1 when 1111

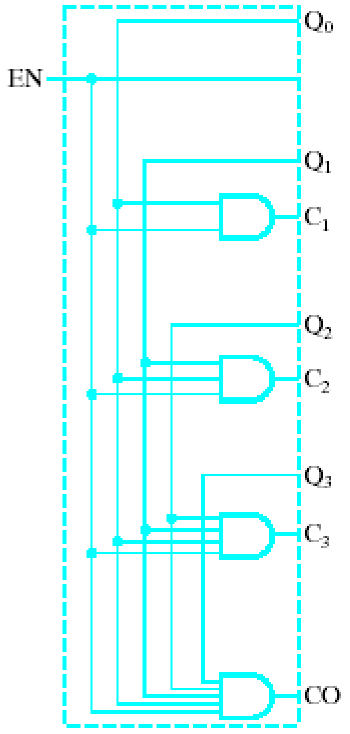
› 4 gate delays

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

4-bit Counter with Parallel Gating

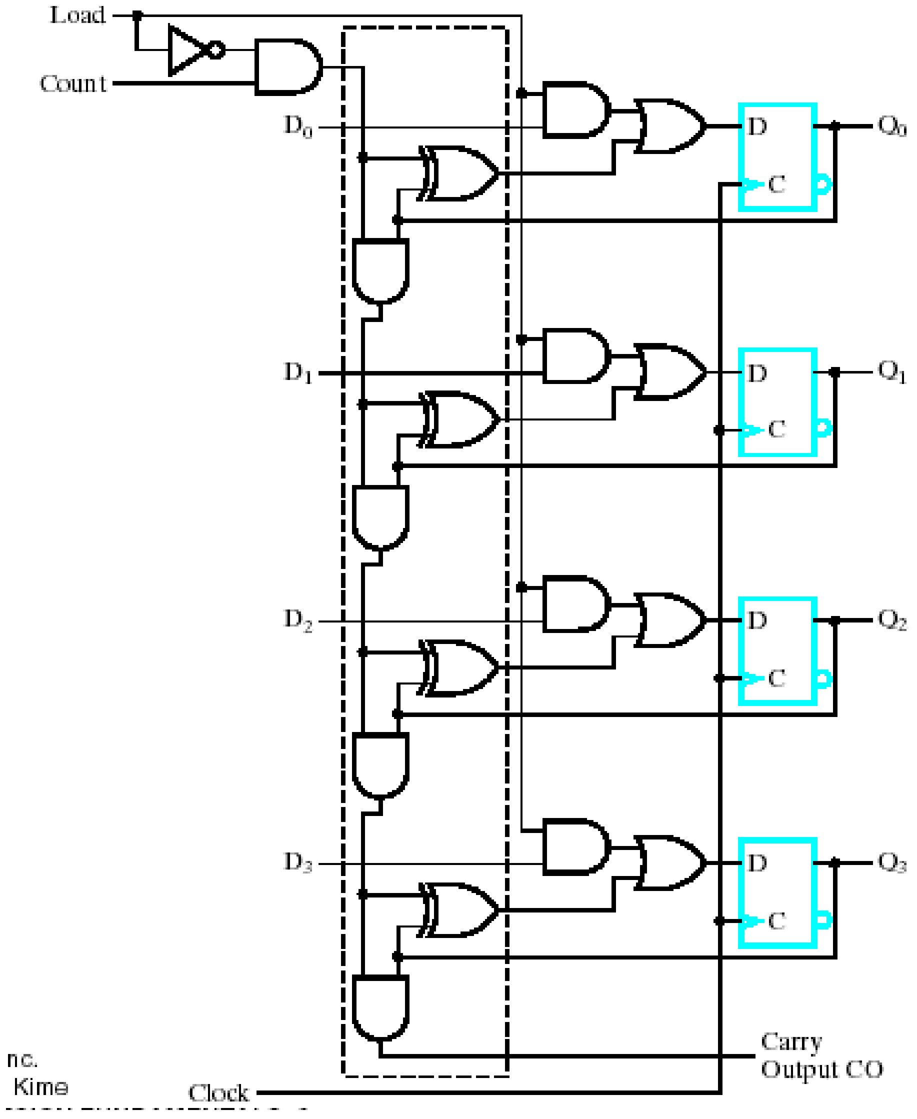


(a) Logic Diagram-Serial Gating



(b) Logic Diagram-Parallel Gating

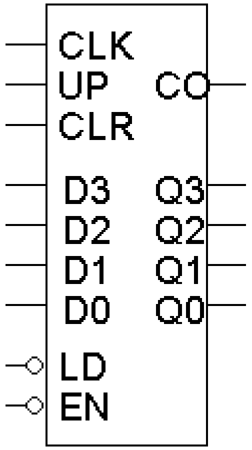
4-bit Binary Counter with Parallel Load



Load	Count	
1	x	Parallel Load
0	0	No Change
0	1	Count

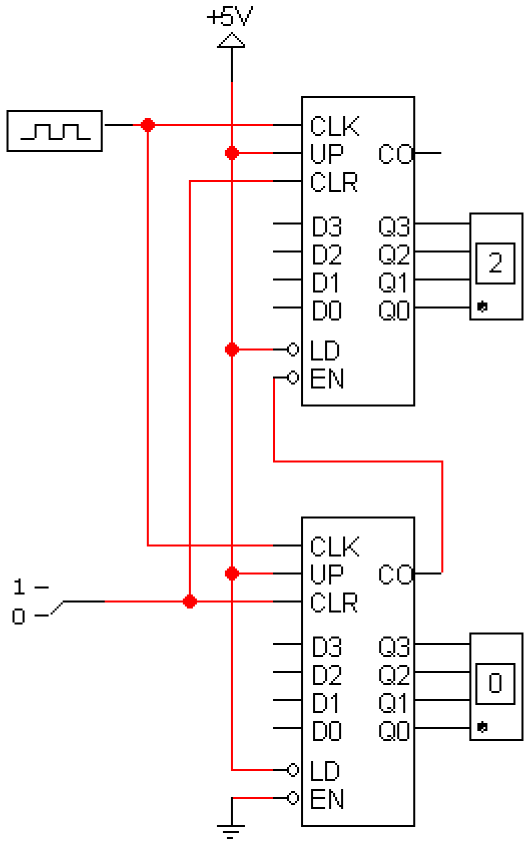
More complex counters

- More complex counters are also possible:
  - It can increment or decrement, by setting the UP input to 1 or 0
  - You can immediately (asynchronously) clear the counter to 0000 by setting CLR = 1
  - You can specify the counter's next output by setting D_3-D_0 to any four-bit value and clearing LD
  - The active-low EN input enables or disables the counter
    - When the counter is disabled, it continues to output the same value without incrementing, decrementing, loading, or clearing
  - The "counter out" CO is normally 1, but becomes 0 when the counter reaches its maximum value, 1111



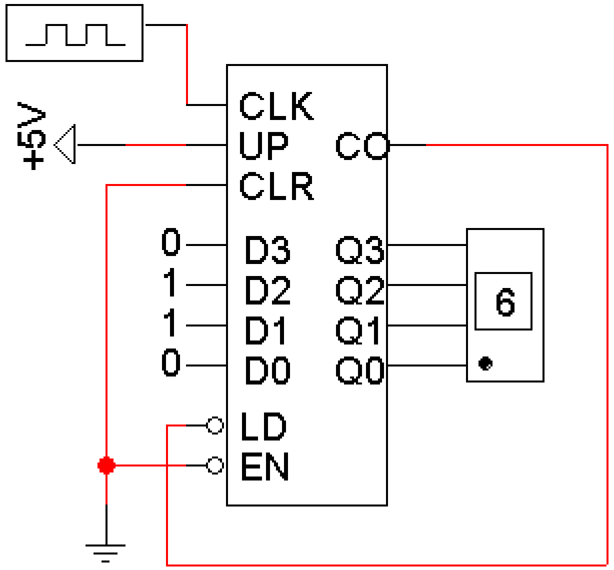
An 8-bit counter

- As you might expect by now, we can use these general counters to build other counters
- Here is an 8-bit counter made from two 4-bit counters
  - The bottom device represents the least significant four bits, while the top counter represents the most significant four bits
  - When the bottom counter reaches 1111 (i.e., when CO = 0), it enables the top counter for one cycle
- Other implementation notes:
  - The counters share clock and clear signals
  - Hex displays are used here



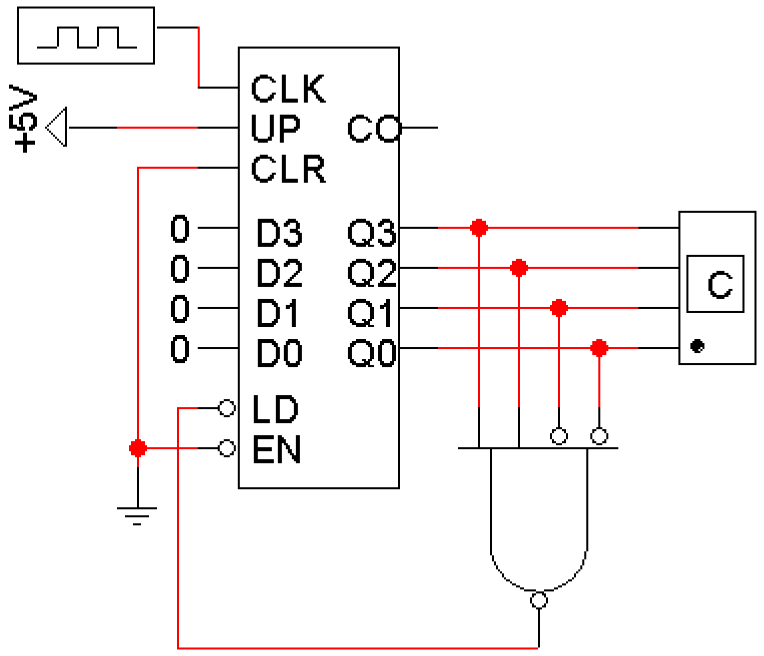
A restricted 4-bit counter

- We can also make a counter that “starts” at some value besides 0000
- In the diagram below, when CO=0 the LD signal forces the next state to be loaded from D_3-D_0
- The result is this counter wraps from 1111 to 0110 (instead of 0000)

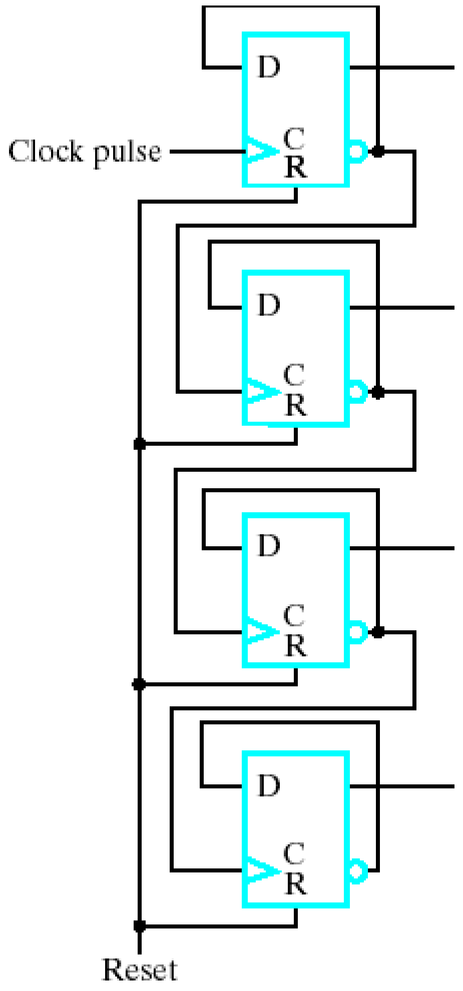


Another restricted counter

- We can also make a circuit that counts up to only 1100, instead of 1111
- Here, when the counter value reaches 1100, the NAND gate forces the counter to load, so the next state becomes 0000



Ripple 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

Simple, yet asynchronous circuits !!!

Summary

- Counters serve many purposes in sequential logic design
- There are lots of variations on the basic counter
  - Some can increment or decrement
  - An enable signal can be added
  - The counter's value may be explicitly set
- There are also several ways to make counters
  - You can follow the sequential design principles to build counters from scratch
  - You could also modify or combine existing counter devices