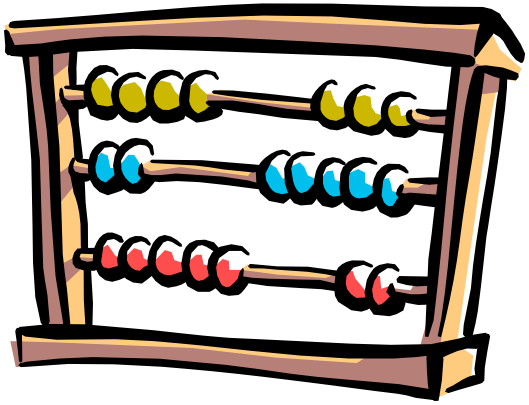
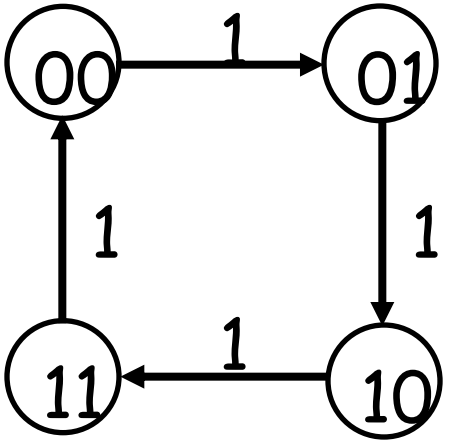
Counter

Counters

- Counters are a specific type of sequential circuit.
- Like registers, the state, or the flip-flop values themselves, serves as the "output."
- 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



Benefits of counters

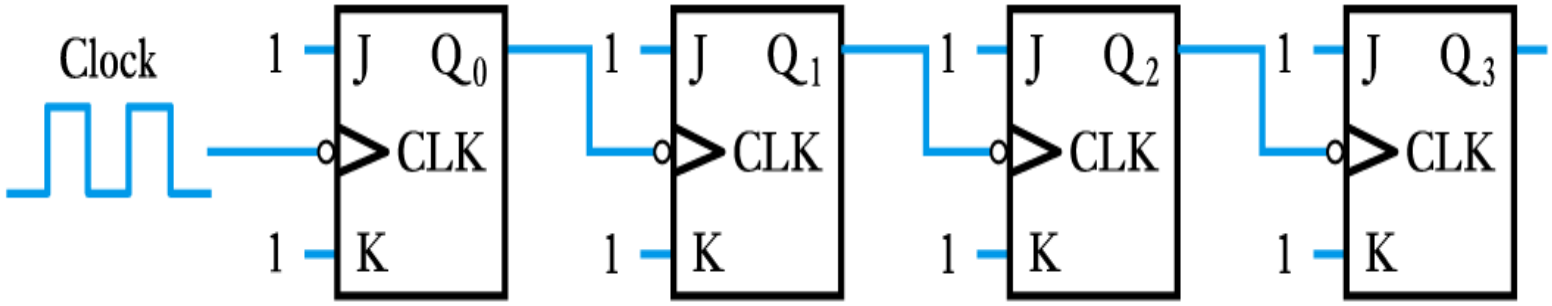
- Counters can act as simple clocks to keep track of “time.”
- You may need to record how many times something has happened.
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?
- All processors contain a program counter, or PC.
  - Programs consist 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.

- In digital logic and computing, a **counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal.

Classifications of Counters

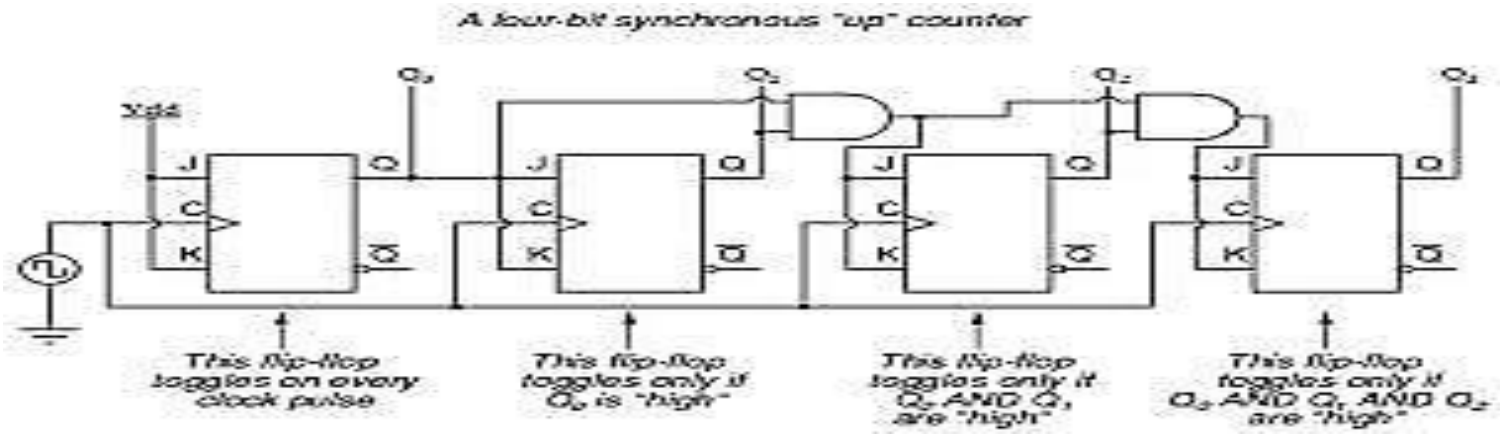
Asynchronous Counters

- Only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop.means output of previous flip-flop is connected to clock input of next flip flop.
- Asynchronous counters are slower than synchronous counters because of the delay in the transmission of the pulses from flip-flop to flip-flop.
- Asynchronous counters are also **called *ripple-counters*** because of the way the clock pulse ripples it way through the flip-flops.



Synchronous Counters

- All flip-flops are clocked simultaneously by an external clock. Means clock input of all flip flops are connected to same external clock.
- Synchronous counters are faster than asynchronous counters because of the simultaneous clocking.
- Synchronous counters are an example of *state machine* design because they have a set of states and a set of transition rules for moving between those states after each clocked event.



States / Modulus / Flip-Flops

- The number of flip-flops determines the count limit or number of states.

  (STATES = $2^{\#\ of\ flip\ flops}$)

- The number of states used is called the *MODULUS.*

- For example, a Modulus-12 counter would count from 0 (0000) to 11 (1011) and requires four flip-flops (16 states - 12 used).

❖ Electronic counters -- Examples

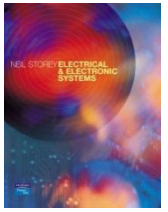
1. Up/down counter – counts both up and down, under command of a control input

2. Ring counter – formed by a shift register with feedback connection in a ring

3. Johnson counter – a *twisted* ring counter

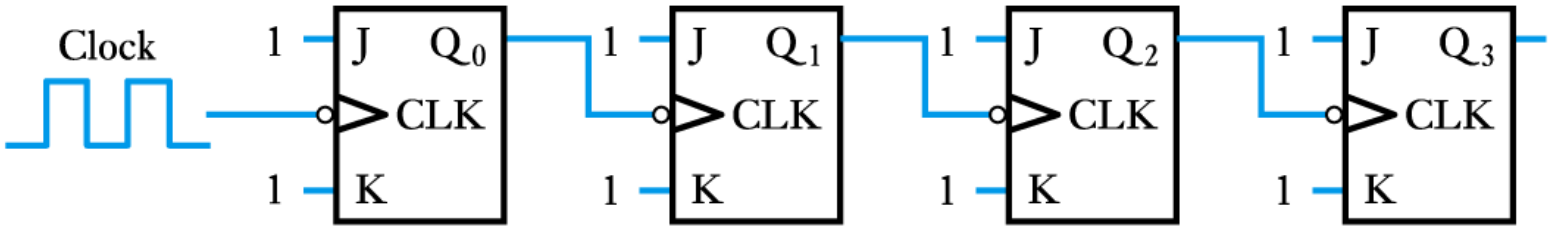
4. Cascaded counter

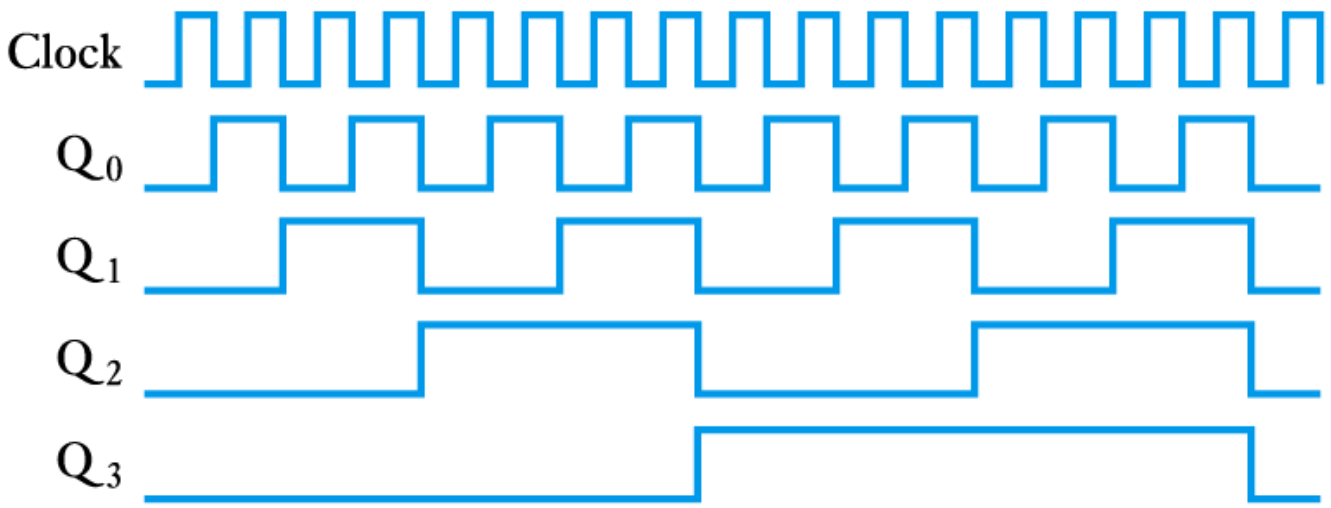
5. Decade Counter



Asynchronous Counters

- **Asynchronous Counter/Ripple counters**
  - can be constructed using several flip flops
  - consider the following arrangement
  - with $J = K = 1$ each flip flop toggles on the falling edge of its clock input



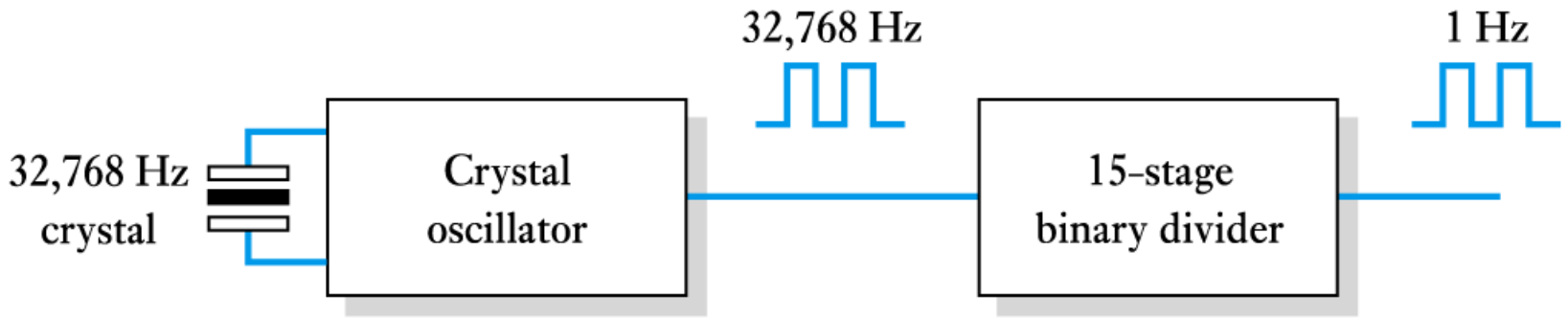


- Each stage toggles at half the frequency of the previous stage
  - acts as a frequency divider
  - divides frequency by 2^n (n is the number of stages)

- Application of a frequency divider

Clock generator for a digital watch

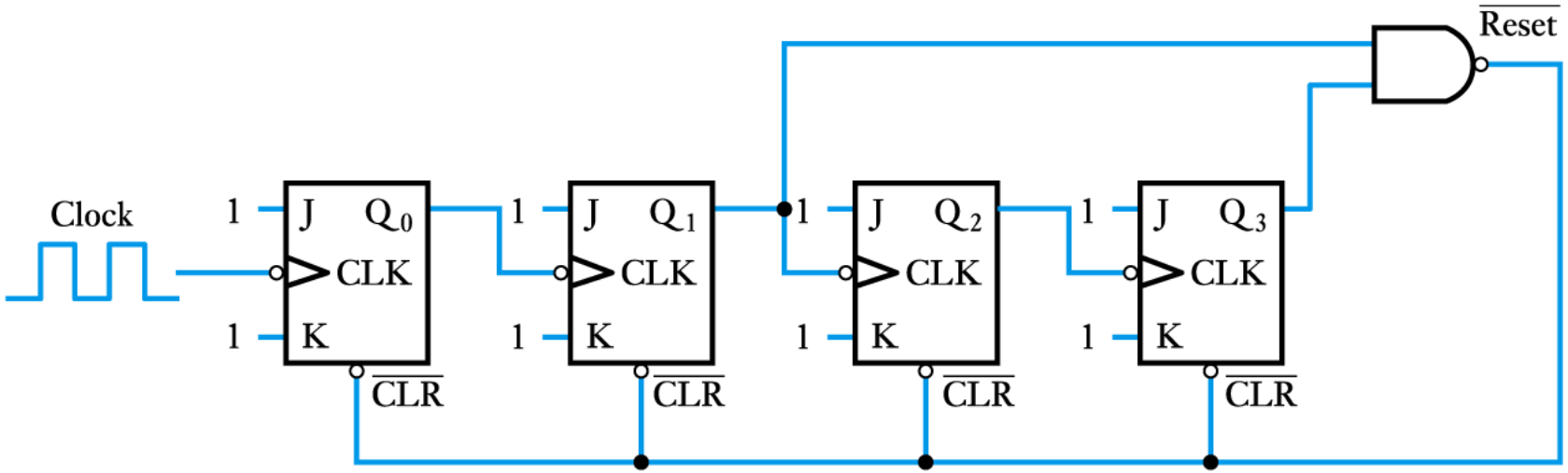
  – 15-stage counter divides signal from a crystal oscillator by 32,768 to produce a 1 Hz signal to drive stepper motor or digital display



- Consider the pattern on the outputs of the counter as shown – displayed on the right
- the outputs count in binary from 0 to 2^n-1 and then repeat
  - the circuit acts as a **modulo-2^n counter**
  - since the counting process propagates from one bistable to the next this is called a **ripple counter**
  - circuit shown is a **4-bit** or **modulo-16** (or **mod-16**) ripple counter

Number of clock pulses	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
17	0	0	0	1
18	0	0	1	0
19	0	0	1	1
20	0	1	0	0

- **Modulo-*N* counters**
  - by using an appropriate number of stages the earlier counter can count modulo any power of 2
  - to count to any other base we add reset circuitry
  - e.g. the modulo-10 or decade counter shown here



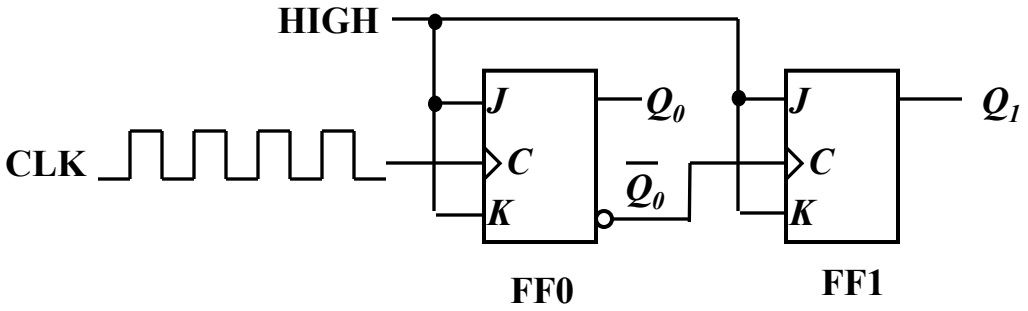
- **Down and up/down Counters**
  - a slight modification to the earlier circuit will produce a counter that counts from 2^n-1 to 0 and then restarts
  - this is a **down counter**
  - a further modification can produce an **up/down counter** which counts up or down depending on the state of a control line (usually labelled up/$\overline{\text{down}}$)
    - when this is 1 the counter counts up
    - when this is 0 the counter counts down

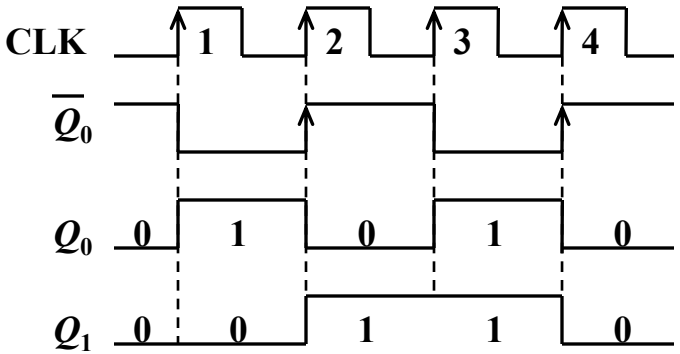
Drawbacks/Limitation of Ripple Counter

- **Propagation delay in counters**
  - while ripple counters are very simple they suffer from problems at high speed
  - since the output of one flip-flop is triggered by the change of the previous device, delays produced by each flip-flop are summed along the chain
  - the time for a single device to respond is termed its **propagation delay time** t_{PD}
  - an n-bit counter will take $n \times t_{PD}$ to respond
  - if read before this time the result will be garbled

Asynchronous (Ripple) Counters

- Example: 2-bit ripple binary counter.
- Output of one flip-flop is connected to the clock input of the next more-significant flip-flop.



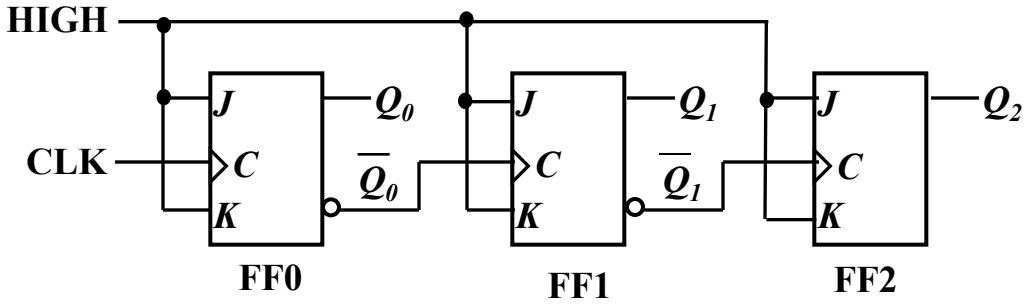


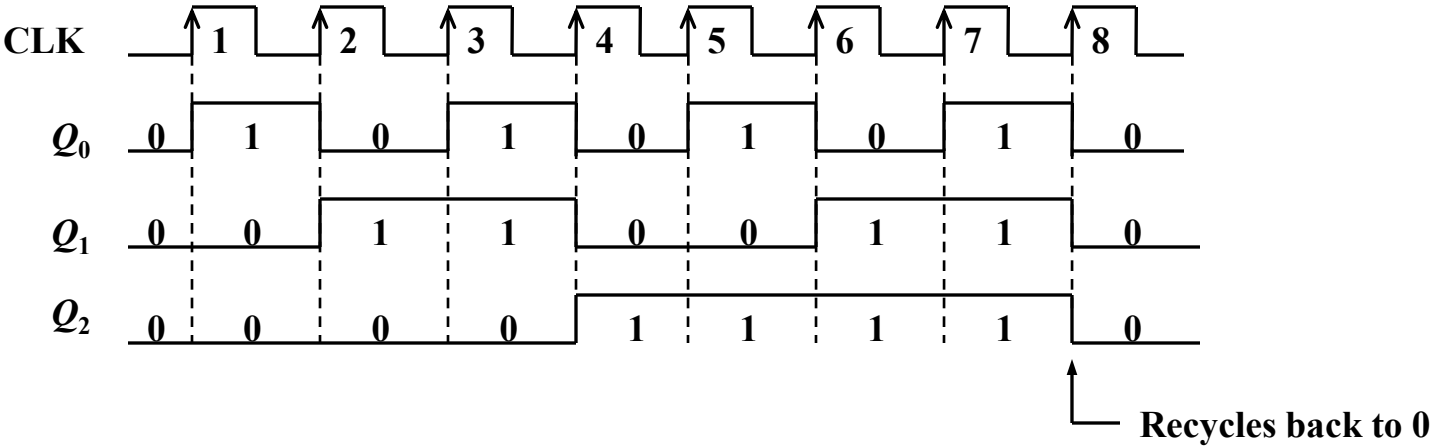
Timing diagram

$00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$ ...

Asynchronous (Ripple) Counters

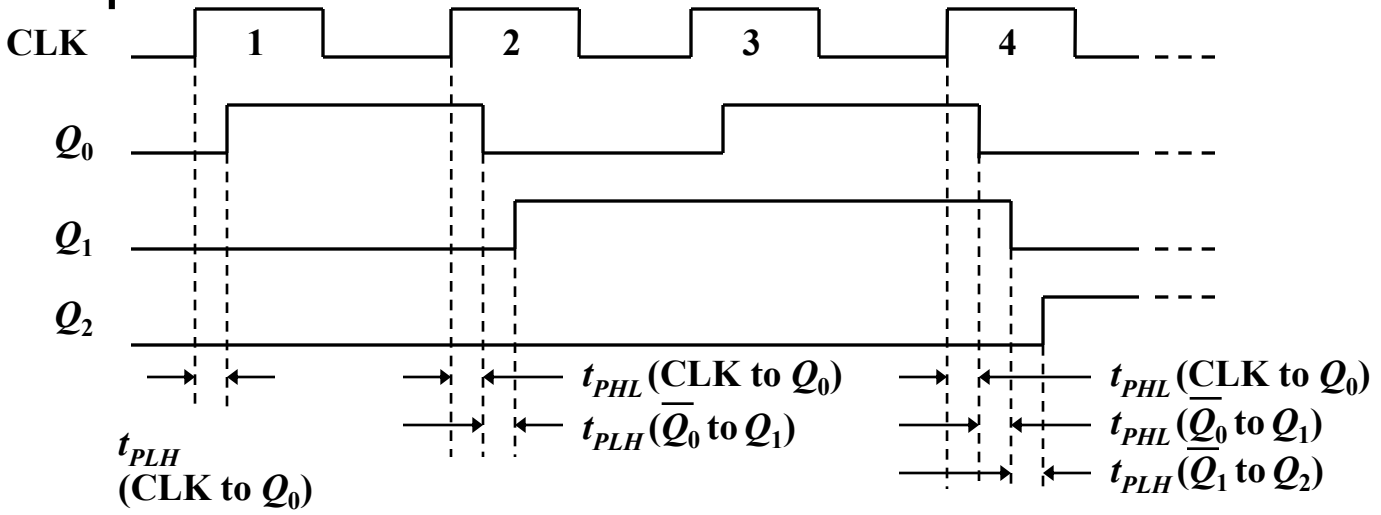
- Example: 3-bit ripple binary counter.





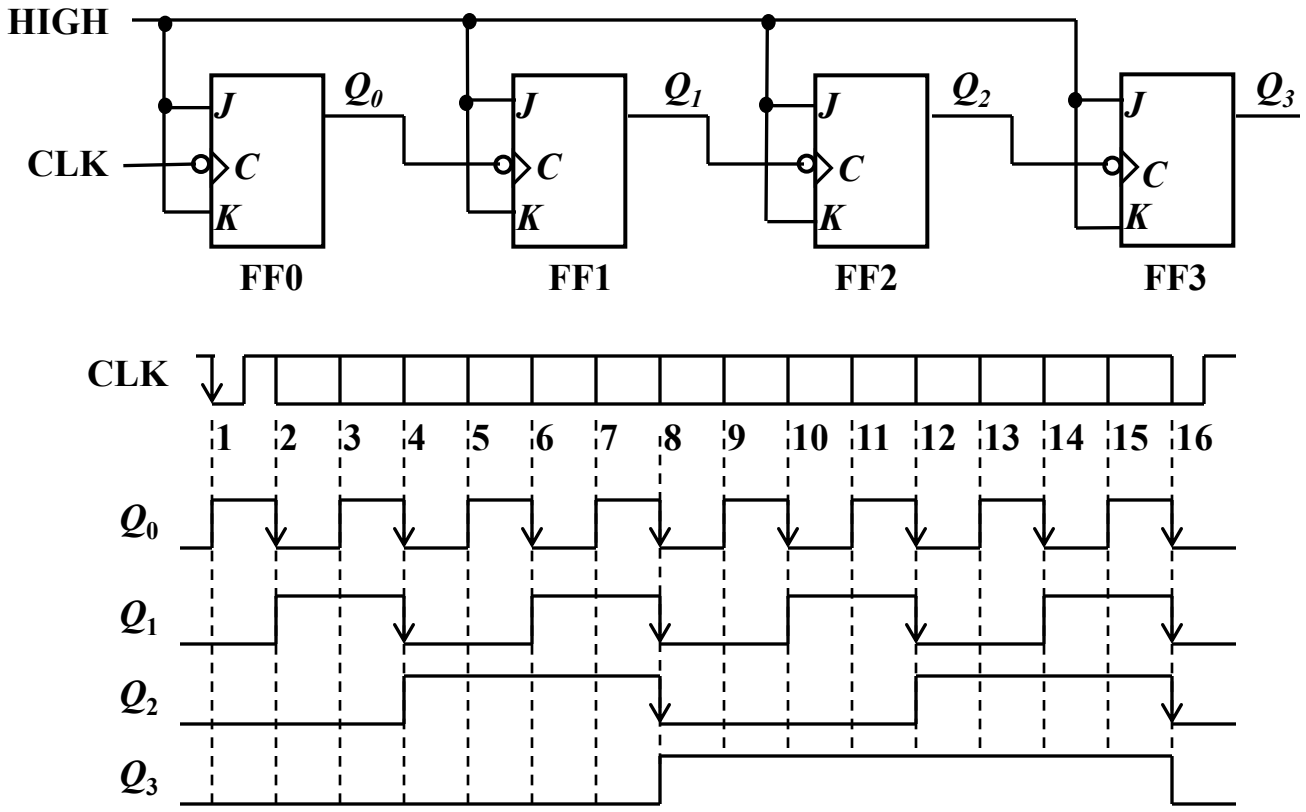
Asynchronous (Ripple) Counters

- Propagation delays in an asynchronous (ripple-clocked) binary counter.
- If the accumulated delay is greater than the clock pulse, some counter states may be misrepresented!



Asynchronous (Ripple) Counters

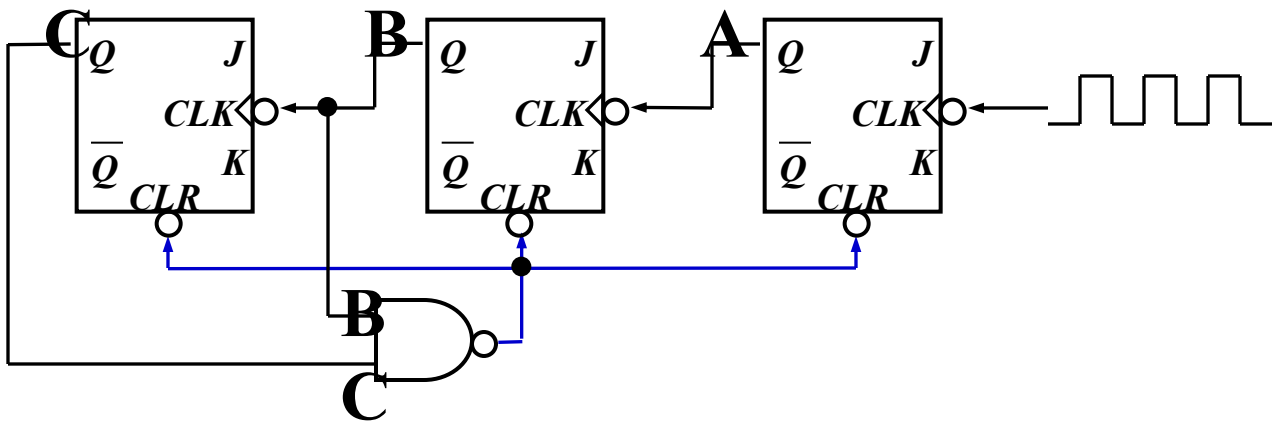
- Example: 4-bit ripple binary counter (negative-edge triggered).



Asyn. Counters with MOD no. < 2^n

- States may be skipped resulting in a truncated sequence.
- Technique: force counter to *recycle before going through all of the states* in the binary sequence.
- Example: Given the following circuit, determine the counting sequence (and hence the modulus no.)

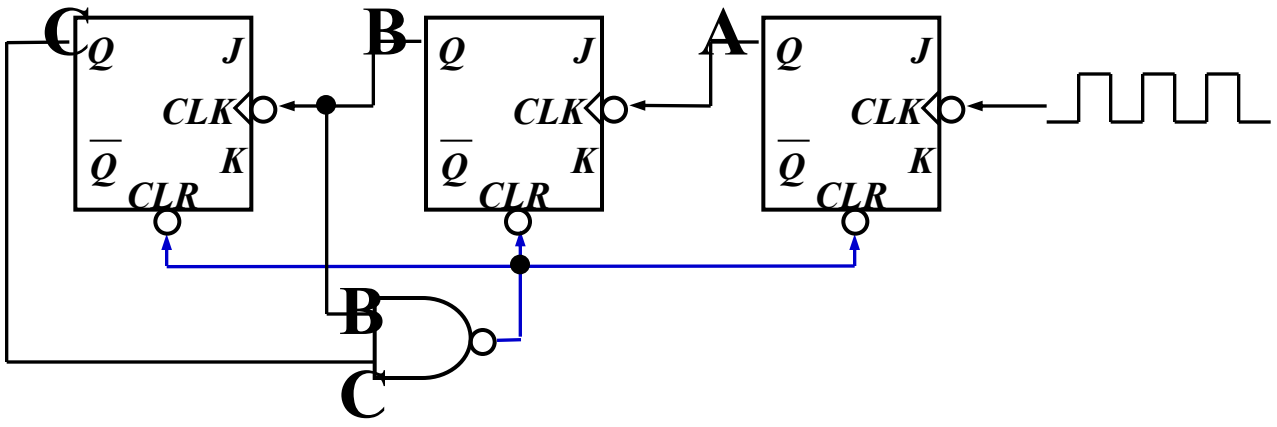
All *J*, *K* inputs are 1 (HIGH).

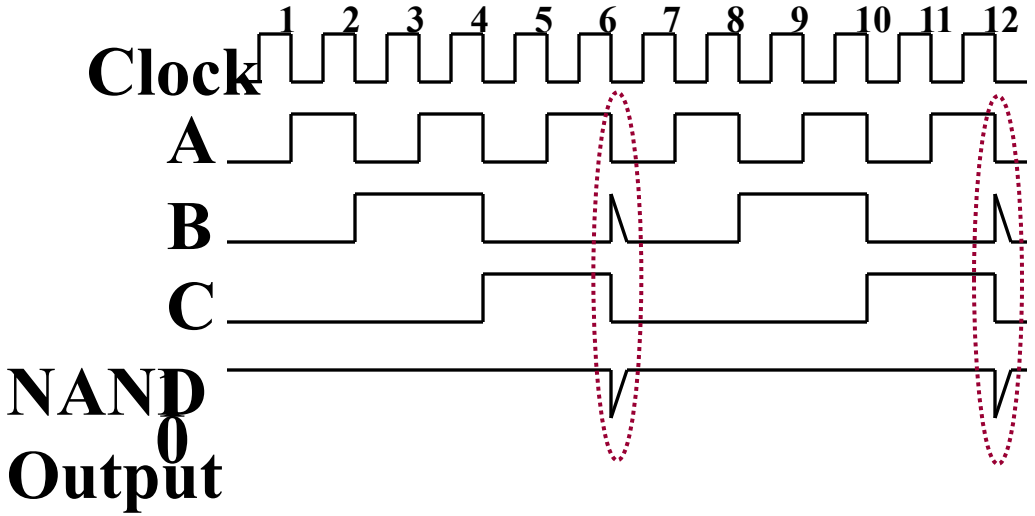


Asyn. Counters with MOD no. < 2^n

- Example (cont'd):

All *J*, *K* inputs are 1 (HIGH).

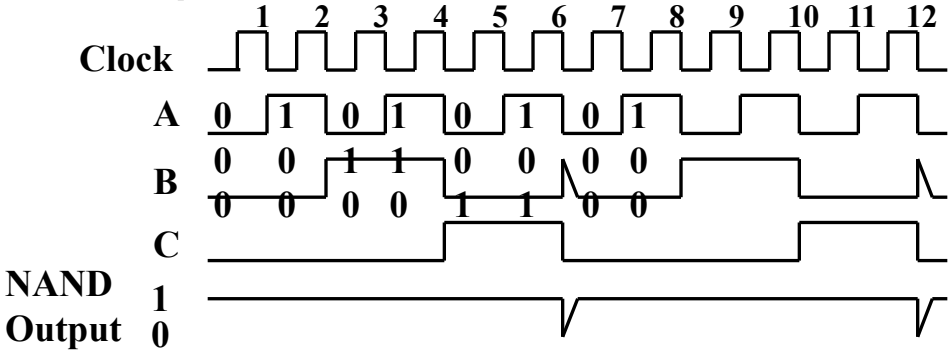


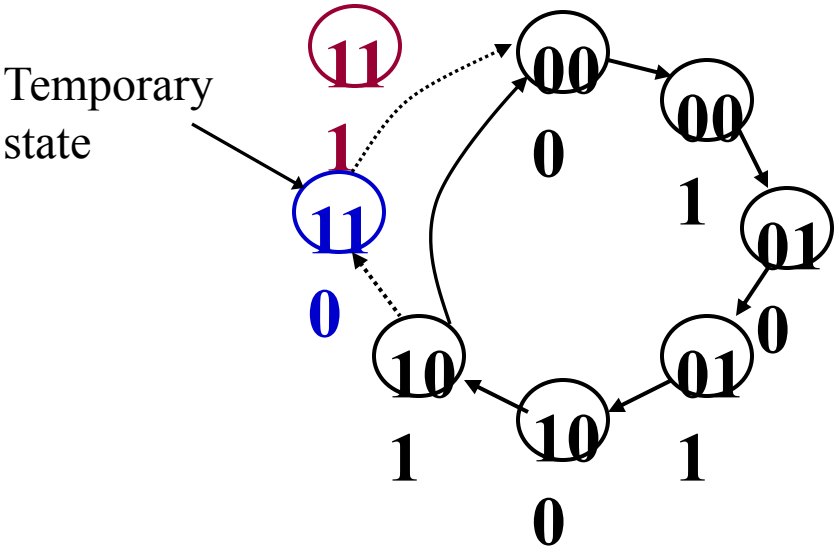


MOD-6 counter produced by clearing (a MOD-8 binary counter) when count of six (110) occurs.

Asyn. Counters with MOD no. < 2^n

- Example (cont'd): Counting sequence of circuit (in CBA order).

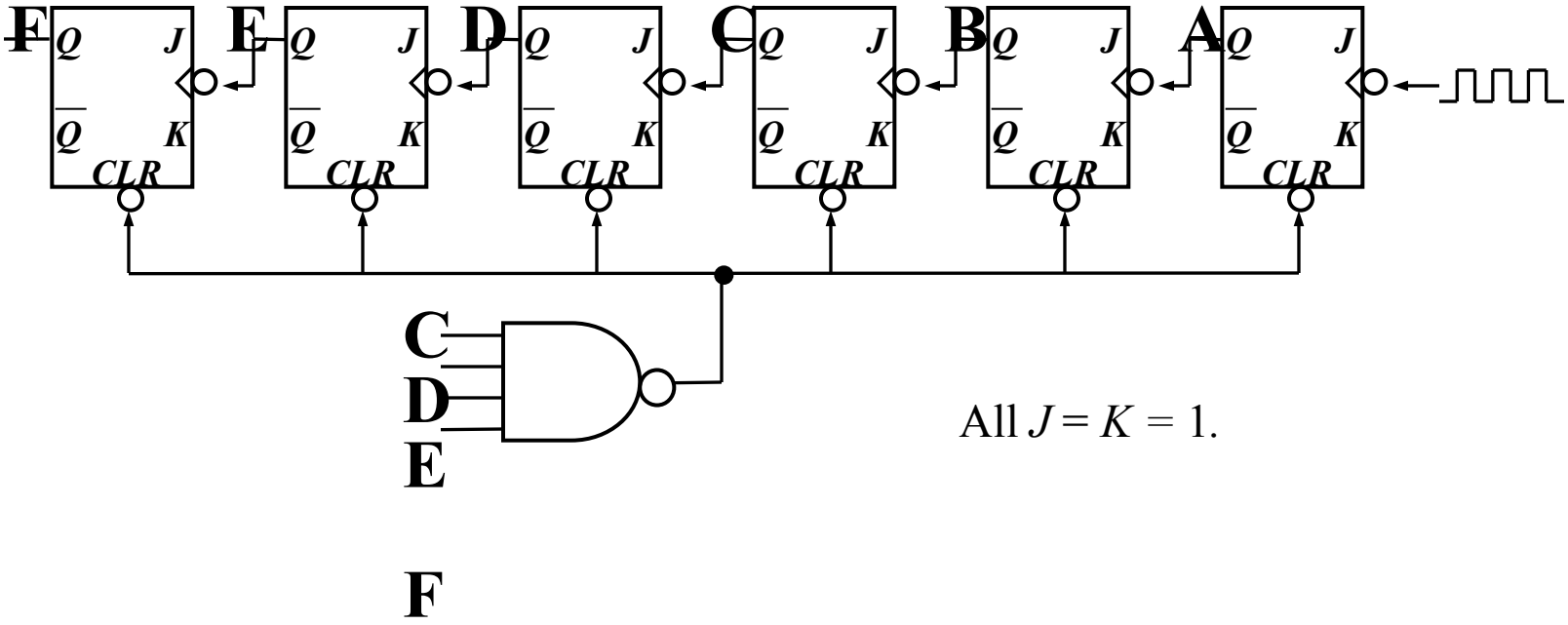




Counter is a MOD-6 counter.

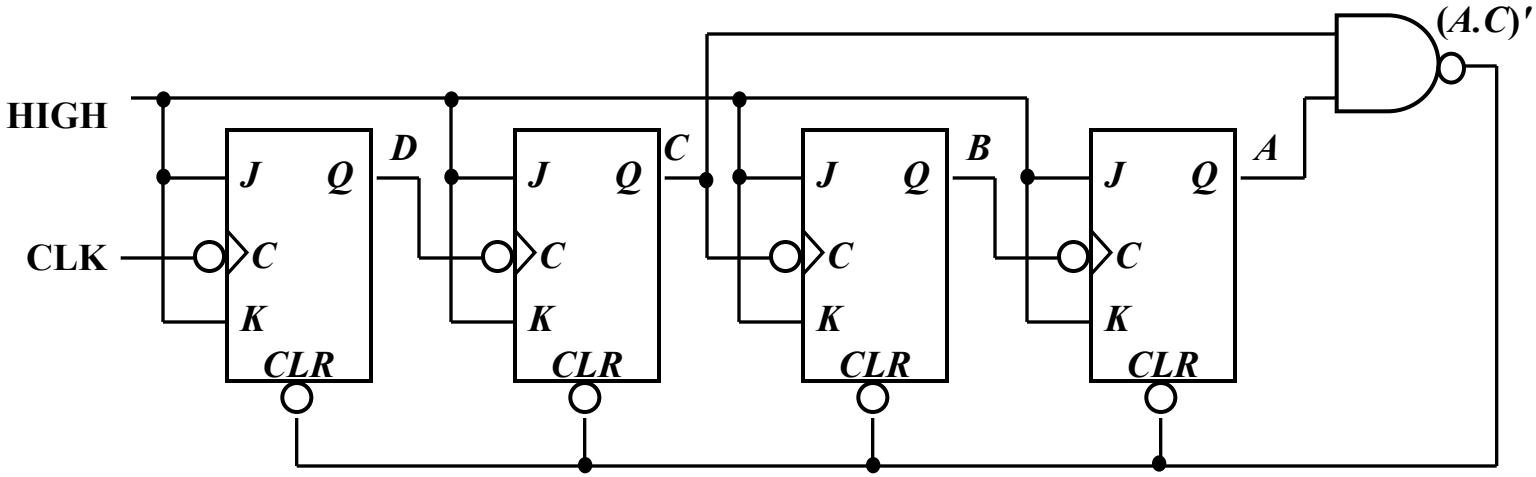
Asyn. Counters with MOD no. < 2^n

- *Exercise:* How to construct an asynchronous MOD-5 counter? MOD-7 counter? MOD-12 counter?
- *Question:* The following is a MOD-? counter?



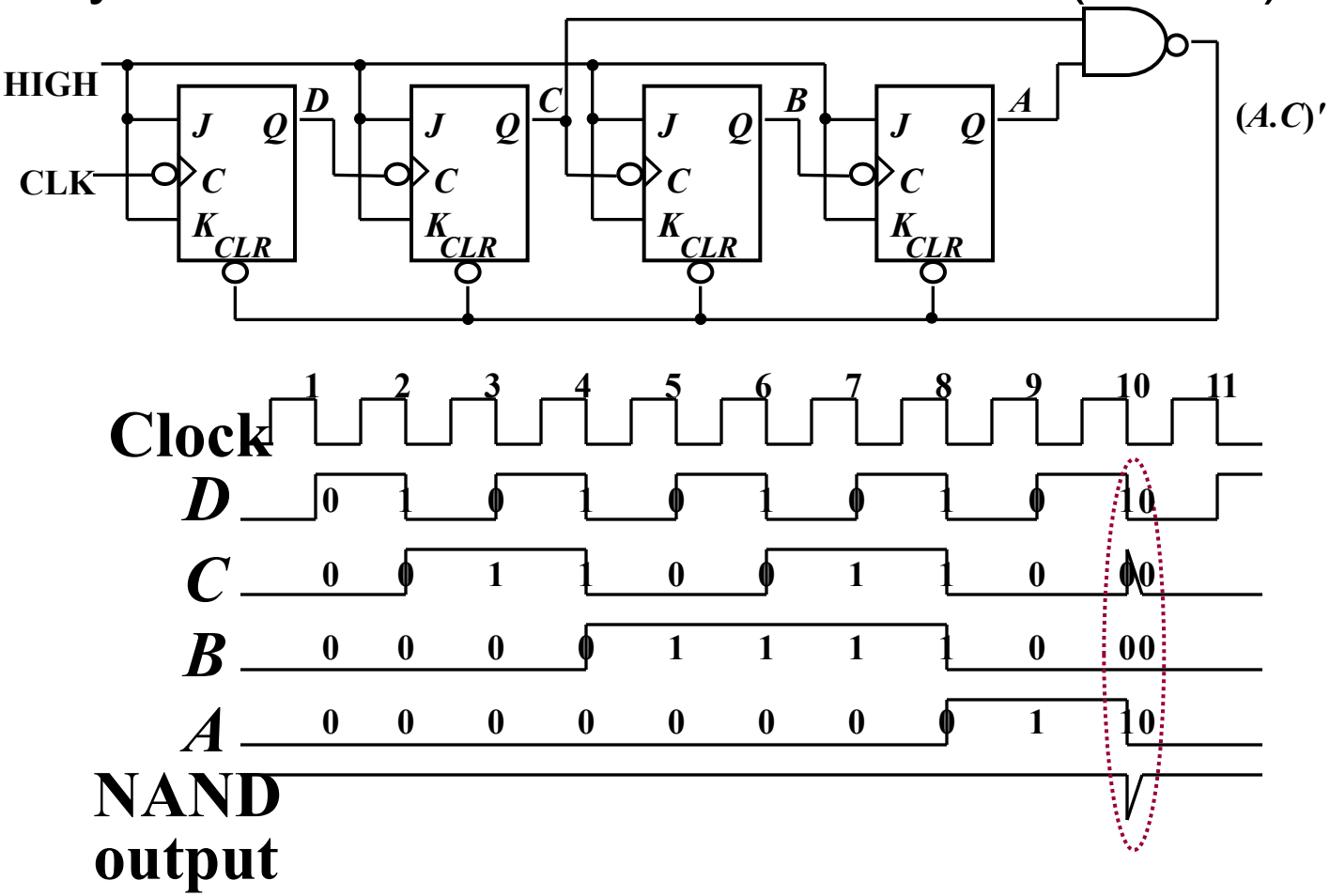
Asyn. Counters with MOD no. < 2^n

- Decade counters (or BCD counters) are counters with 10 states (modulus-10) in their sequence. They are commonly used in daily life (e.g.: utility meters, odometers, etc.).
- Design an asynchronous decade counter.



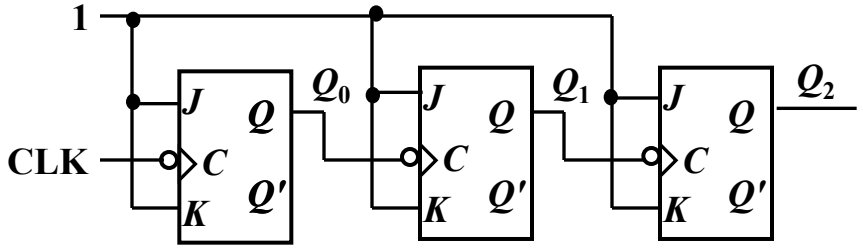
Asyn. Counters with MOD no. < 2^n

- Asynchronous decade/BCD counter (cont'd).

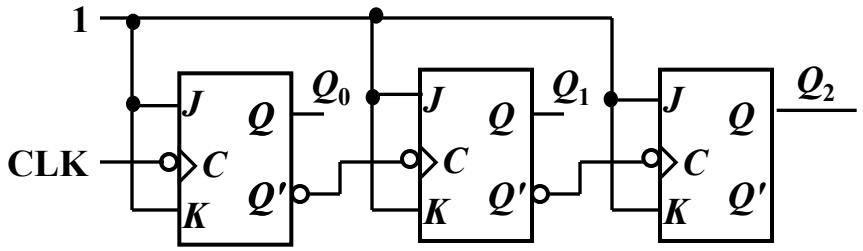


Asynchronous Down Counters

- So far we are dealing with *up counters*. *Down counters*, on the other hand, count downward from a maximum value to zero, and repeat.
- Example: A 3-bit binary (MOD-2^3) down counter.



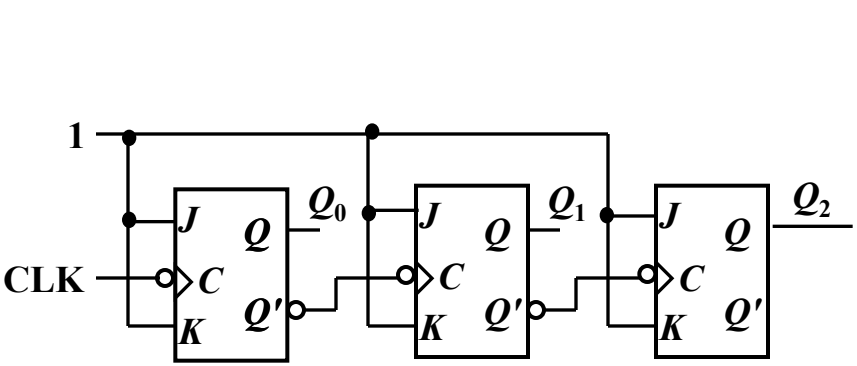
3-bit binary up counter

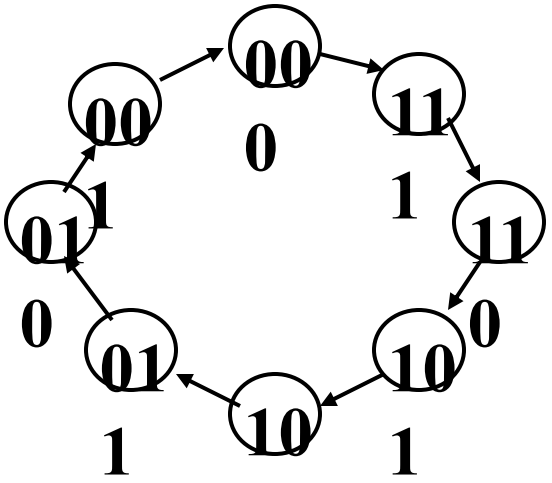


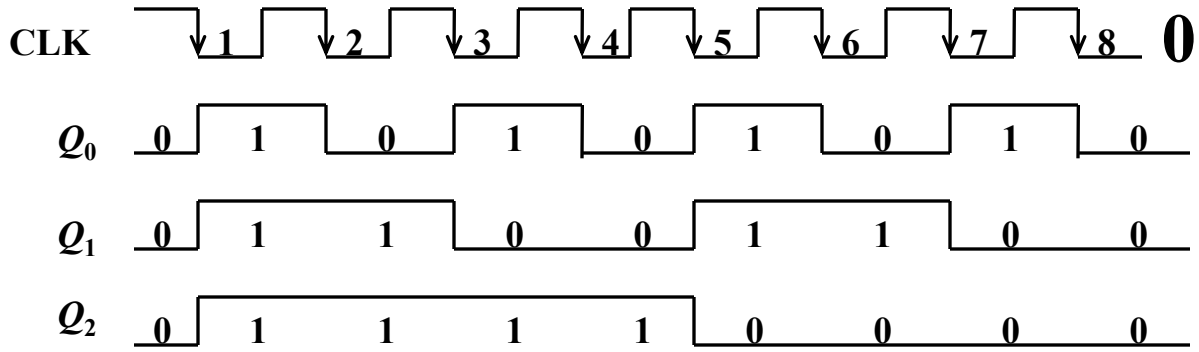
3-bit binary down counter

Asynchronous Down Counters

- Example: A 3-bit binary (MOD-8) down counter.

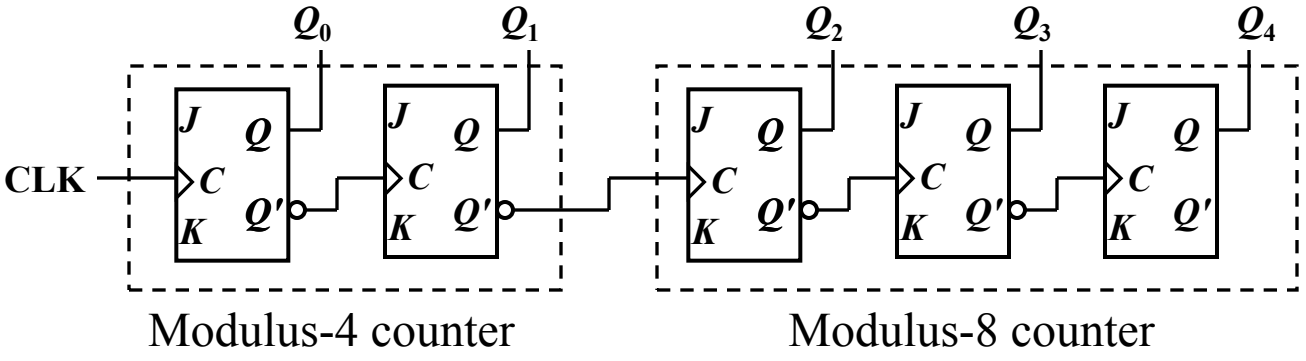






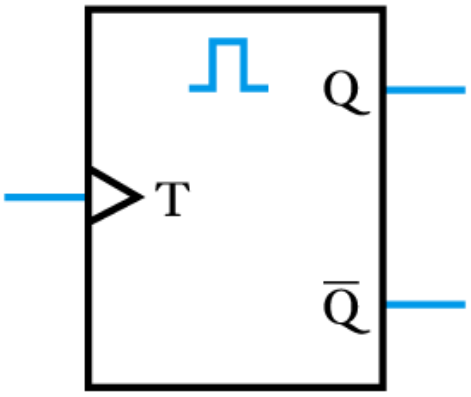
Cascading Asynchronous Counters

- Larger asynchronous (ripple) counter can be constructed by cascading smaller ripple counters.
- Connect last-stage output of one counter to the clock input of next counter so as to achieve higher-modulus operation.
- Example: A modulus-32 ripple counter constructed from a modulus-4 counter and a modulus-8 counter.



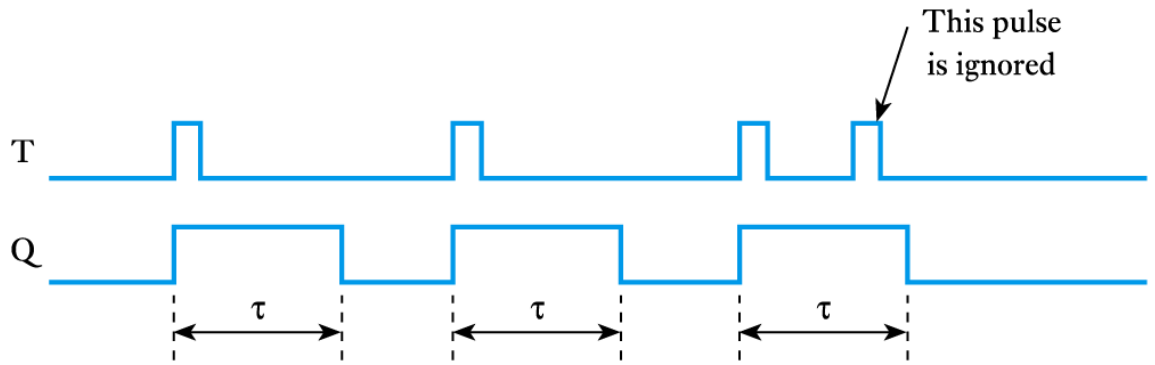
Monostables or one-shots

- Monostables are another form of multivibrator
  - while **bistables** have two stable output states
  - **monostables** have one stable & one metastable states
    - when in its stable state $Q = 0$
    - when an appropriate signal is applied to the trigger input (T) the circuit enters its metastable state with $Q = 1$
    - after a set period of time (determined by circuit components) it reverts to its stable state
    - it is therefore a **pulse generator**

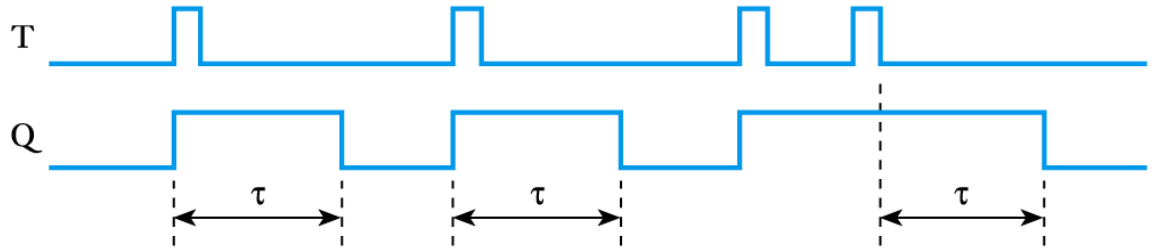


Circuit symbol

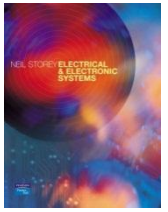
- Monostables can be **retriggerable** or **non-retriggerable**



(a) A non-retriggerable monostable



(b) A retriggerable monostable



Astables

- The last member of the multivibrator family is the **astable**
  - this has two metastable states
  - has the function of a **digital oscillator**
  - circuit spends a fixed period in each state (determined by circuit components)
  - if the period in each state is set to be equal, this will produce a square waveform

Timers

- The **integrated circuit timer** can produce a range of functions
  - including those of a monostable or astable
  - various devices
  - one of the most popular is the **555 timer**
  - can be configured using just a couple of external passive components
  - internal construction largely unimportant – all required information on using the device is in its data sheet

Key Points

- Sequential logic circuits have the characteristic of memory
- Among the most important groups of sequential components are the various forms of multivibrator
  - bistables
  - monostables
  - astables
- The most widely used form is the bistable which includes
  - latches, edge-triggered flip-flops and master/slave devices
- Registers form the basis of various memories
- Counters are widely used in a range of applications
- Monostables and astables perform a range of functions

❖ BCD

- In computing and electronic systems, **binary-coded decimal** (**BCD**) (sometimes called **natural binary-coded decimal**, **NBCD**) or, in its most common modern implementation, **packed decimal**, is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display, and allows faster decimal calculations. Its drawbacks are a small increase in the complexity of circuits needed to implement mathematical operations. Uncompressed BCD is also a relatively inefficient encoding—it occupies more space than a purely binary representation.
- In BCD, a digit is usually represented by four bits which, in general, represent the decimal digits 0 through 9. Other bit combinations are sometimes used for a sign or for other indications (e.g., error or overflow).
- Although uncompressed BCD is not as widely used as it once was, decimal fixed-point and floating-point are still important and continue to be used in financial, commercial, and industrial computing.

- **Basics for BCD**

- To encode a decimal number using the common BCD encoding, each decimal digit is stored in a 4-bit nibble:
- Decimal: 0 1 2 3 4 5 6 7 8 9

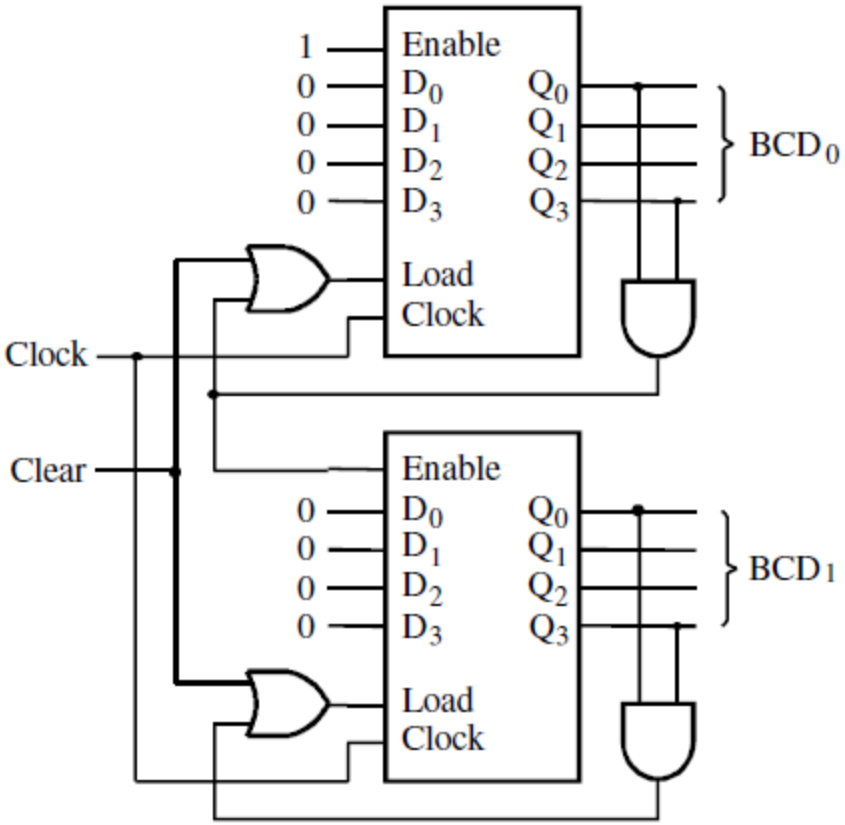
  BCD: 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001
- Thus, the BCD encoding for the number 127 would be:

  0001 0010 0111
- Whereas the pure binary number would be:

  0111 1111

- Binary-coded-decimal(BCD) counters

A Two-digit BCD Counter



- Consists of two modulo-10 counters, one for each BCD digit.
- It is necessary to reset the four flip-flops after the count of 9 has been obtained. Thus the *Load* input to each stage is equal to 1 when Q3=Q0=1, which causes 0s to be loaded into the flip-flops at the next positive edge of the clock signal.
- Keeping the Enable signal for BCD1 low at all times except when BCD0 = 9

❖ IBM and BCD

- IBM used the terms **binary-coded decimal** and **BCD** for 6-bit alphamerics codes that represented numbers, upper-case letters and special characters. Some variation of BCD alphamerics was used in most early IBM computers, including the IBM 1620, IBM 1400 series, and non-Decimal Architecture members of the IBM 700/7000 series.
- Today, BCD data is still heavily used in IBM processors and databases, such as IBM DB2, mainframes, and Power6. In these products, the BCD is usually zoned BCD (as in EBCDIC or ASCII), Packed BCD (two decimal digits per byte), or "pure" BCD encoding (one decimal digit stored as BCD in the low four bits of each byte). All of these are used within hardware registers and processing units, and in software.