SEQUENTIAL CIRCUITS 2

❖ Counters

- 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.
- We examine special types of addition and subtraction operations, which are used for the purpose of counting.
- We will show how the counter circuits can be designed using D flip-flops.

❖ Electronic counters

- In electronics, counters can be implemented quite easily using register-type circuits such as the flip-flop, and a wide variety of classifications exist:

1. Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops

2. Synchronous counter – all state bits change under control of a single clock

3. Decade counter – counts through ten states per stage

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

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

6. Johnson counter – a *twisted* ring counter

7. Cascaded counter

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.
- Asynchronous counters are slower than synchronous counters (discussed later) 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 pulses, or ripples, its way through the flip-flops.

States / Modulus / Flip-Flops

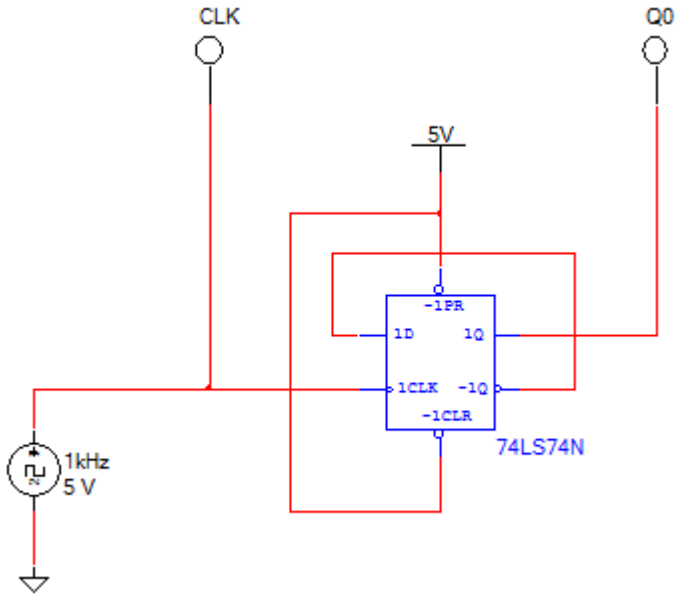
- The number of flip-flops determines the count limit or number of states:

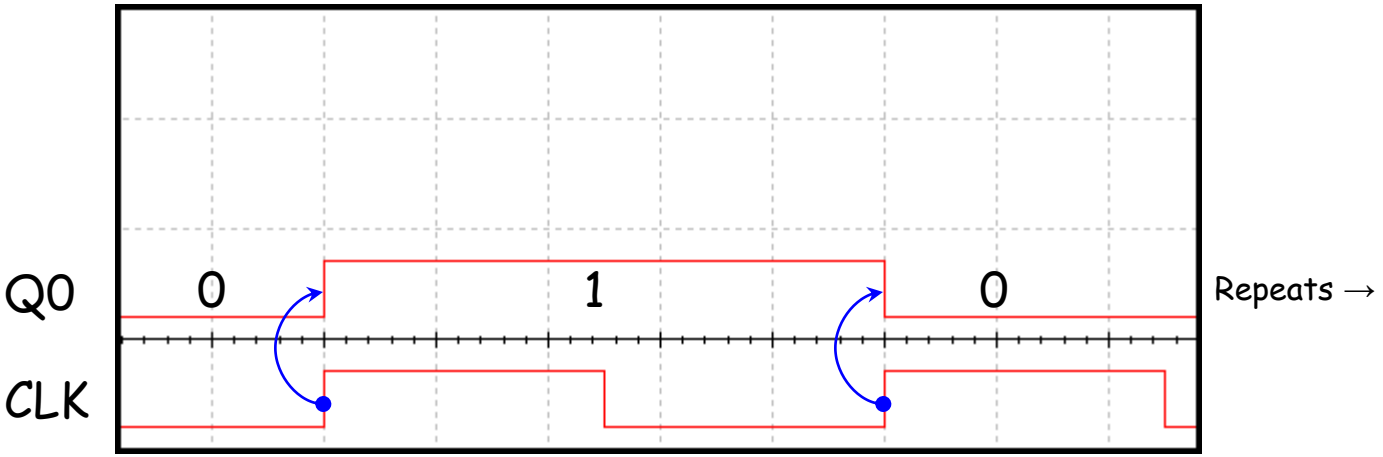
$$\text{States} = 2^{(\#\text{ of flip-flops})}$$

- The number of states used is called the MODULUS.
- For example, a Modulus-12 counter (Mod-12) would count from 0 (0000) to 11 (1011) and would require four flip-flops ($2^4 = 16$ states; 12 are used)

Asynchronous Counter

D-Flip Flop – 1 Bit

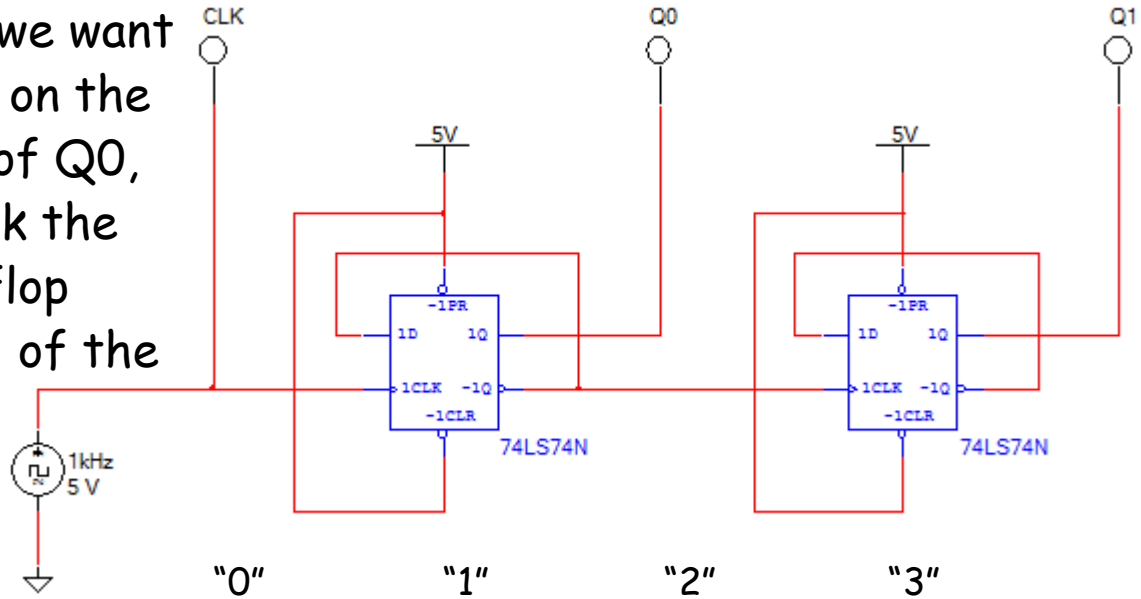


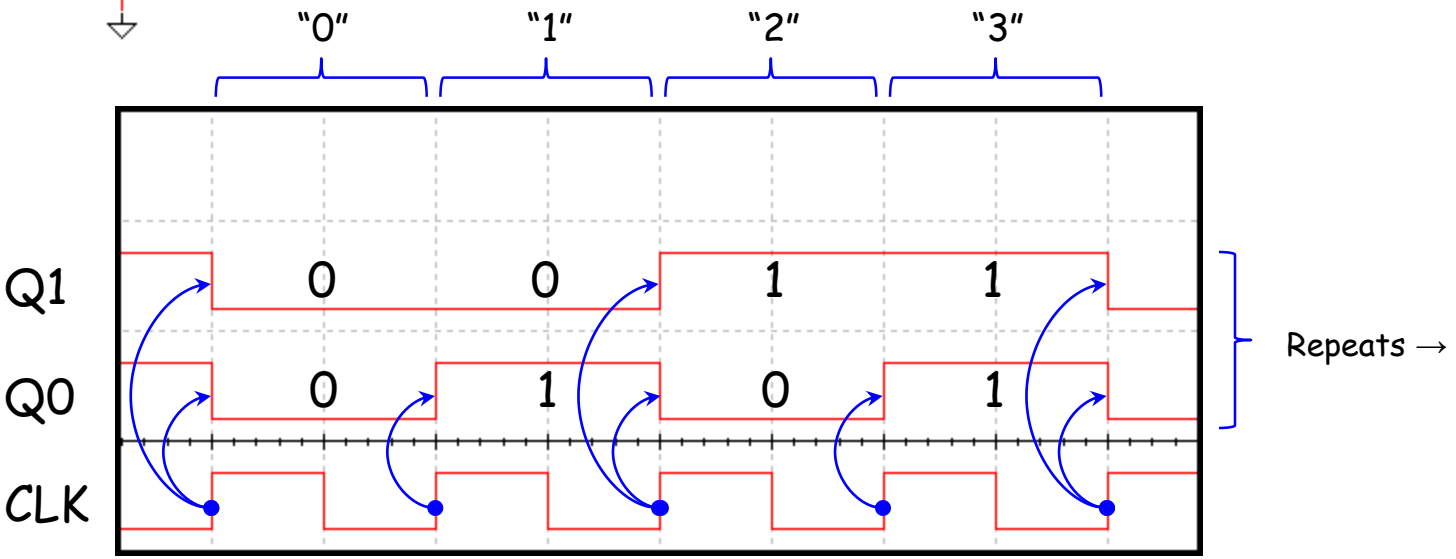


Asynchronous Counter

Up Counter – D-Flip Flops – 2 Bit

Note: Since we want Q1 to toggle on the falling edge of Q0, we must clock the second flip-flop from the $\overline{Q0}$ of the first.

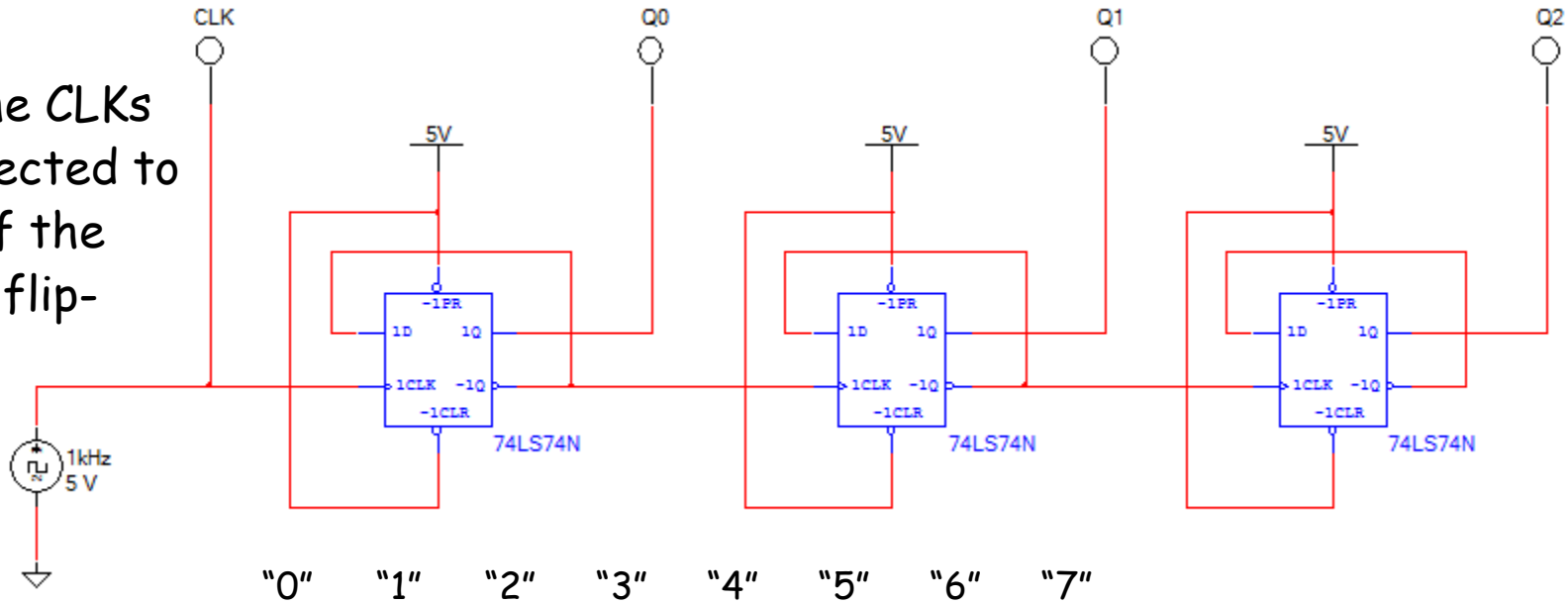


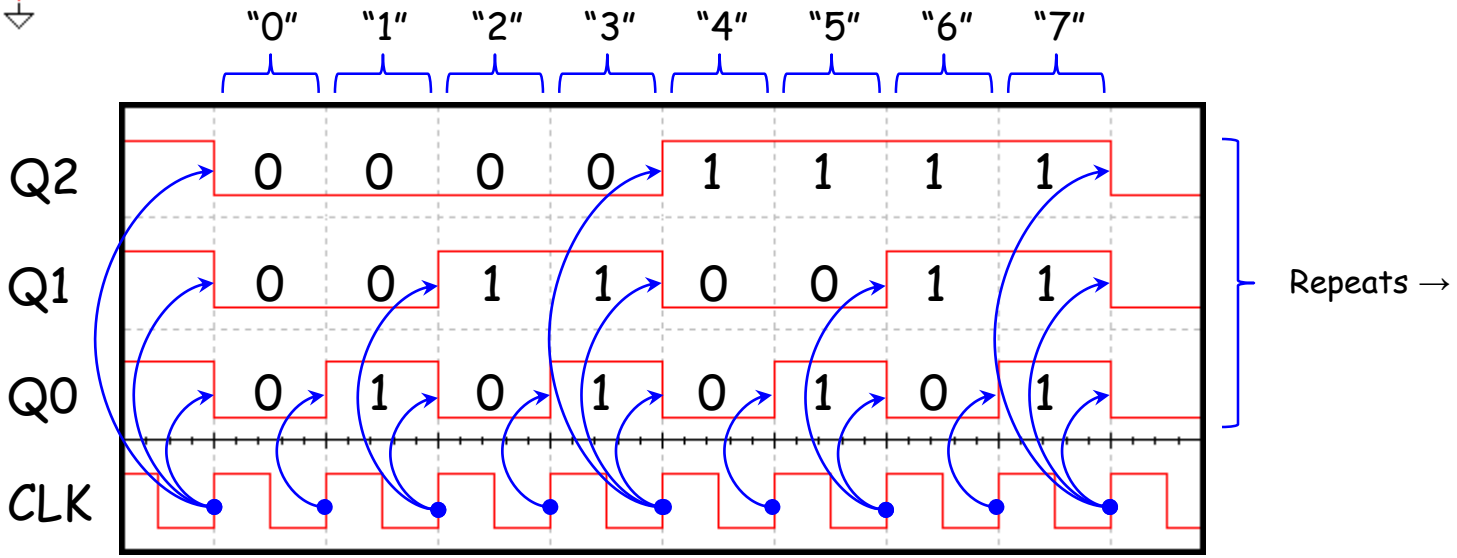


Asynchronous Counter

Up Counter – D-Flip Flops – 3 Bit

Note: The CLKs are connected to the $\overline{Q}$ of the previous 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:

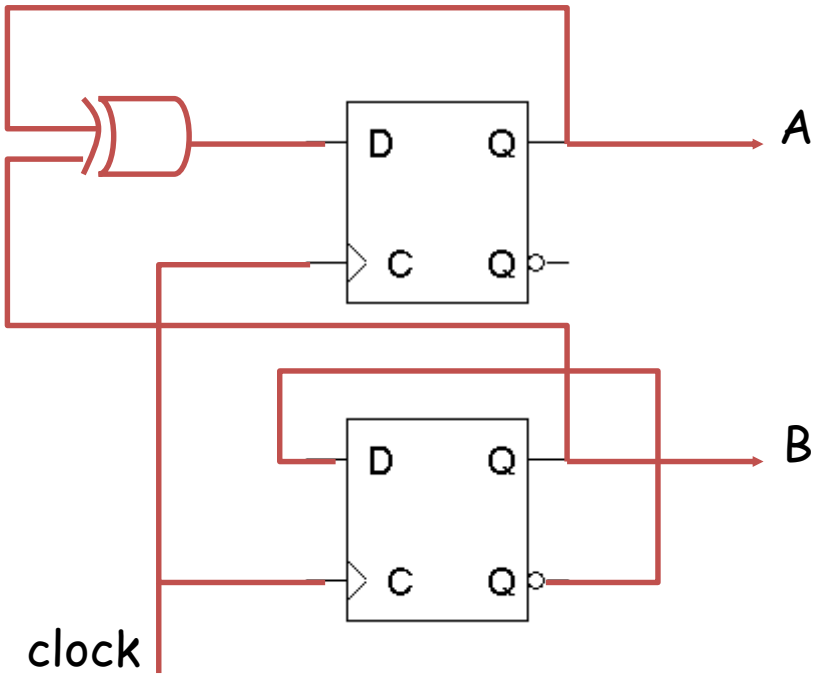
Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Synchronous Binary Up Counter

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

D1= A'B + AB'

D0= B'

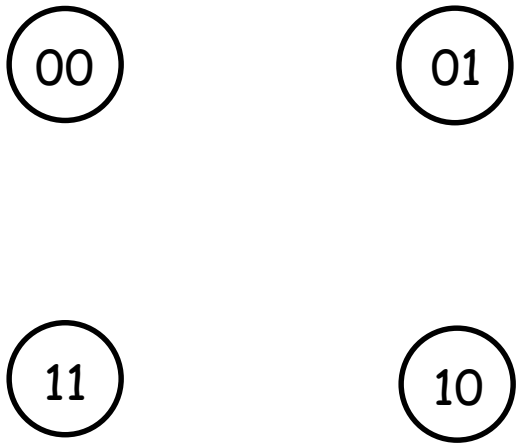


What good are counters?

- 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:



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 Q_1	Present State Q_0	Inputs X	Next State Q_1	Next State 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

Q_1 \ Q_0	0	1	0	1
1	1	0	1	0

X

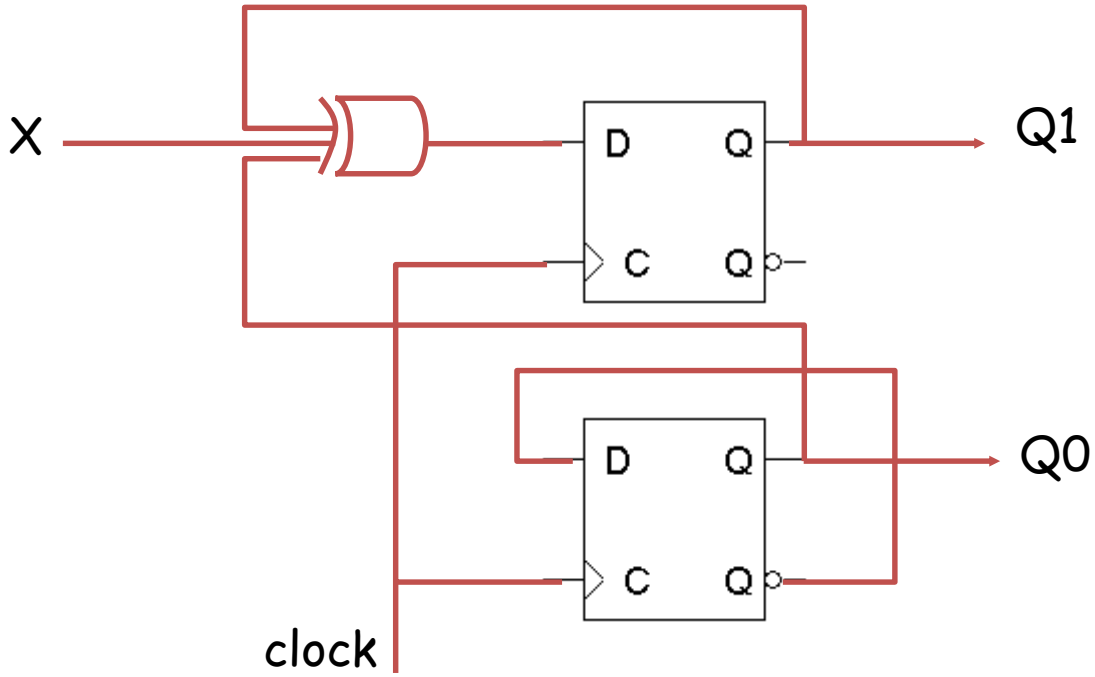
$D_1 = Q_1 \oplus Q_0 \oplus X$

Q_1 \ Q_0	1	1	0	0
1	1	1	0	0

X

$D_0 = Q_0'$

Synchronous Binary Up/Down Counter



Unused states

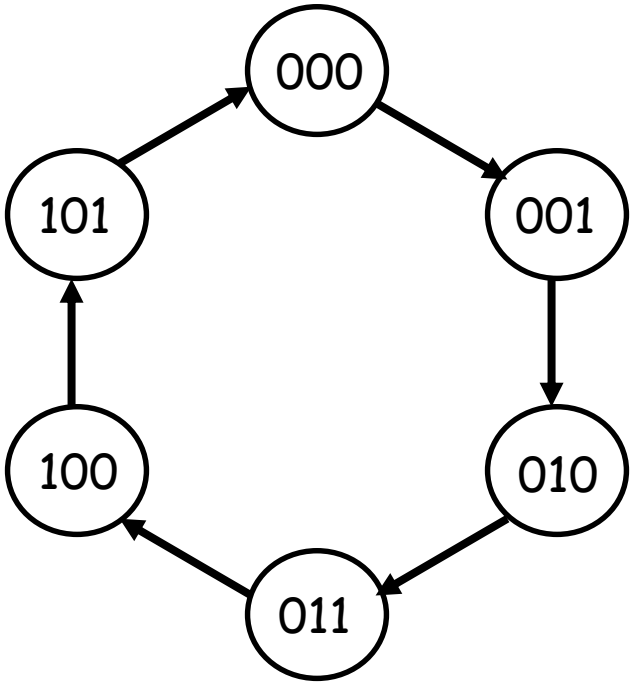
- The examples shown so far have all had 2^n states, and used n flip-flops.

  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?

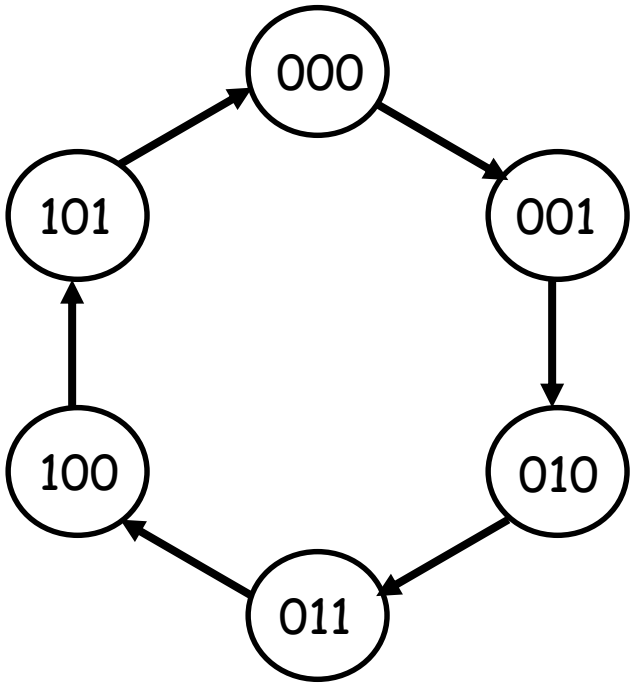
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	?	?	?
1	1	1	?	?	?



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

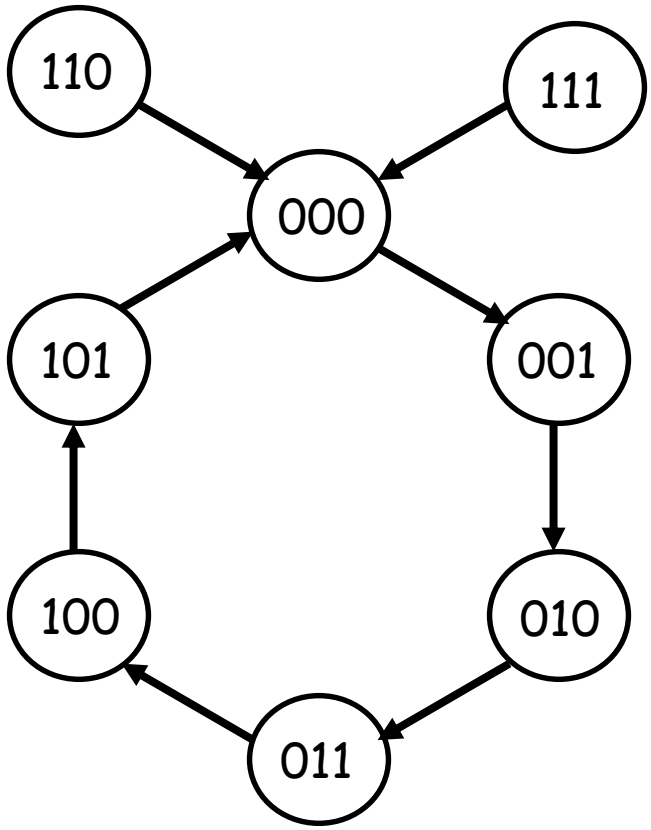
Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	x	x	x
1	1	1	x	x	x



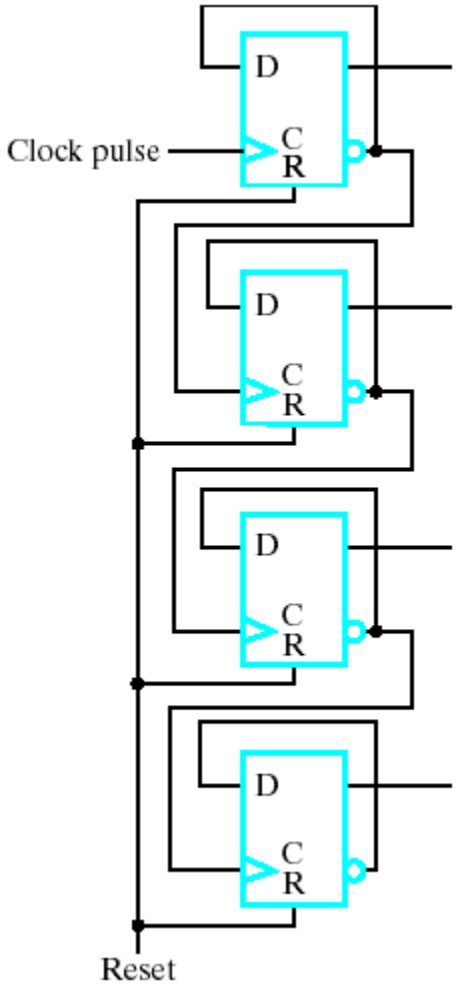
…or maybe you *do* care

- 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



Ripple Counter

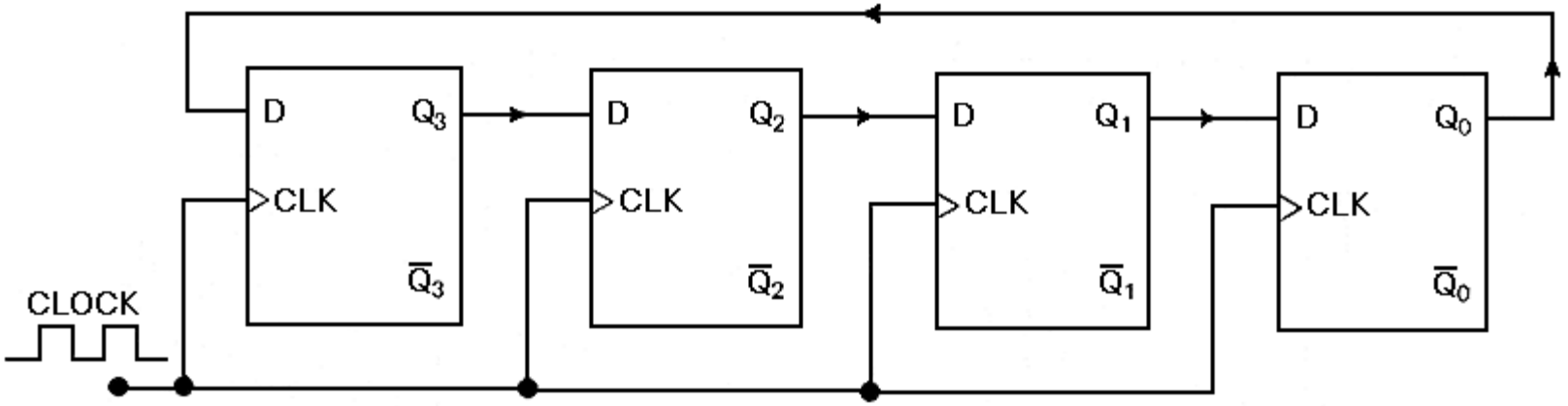


Upward Counting Sequence

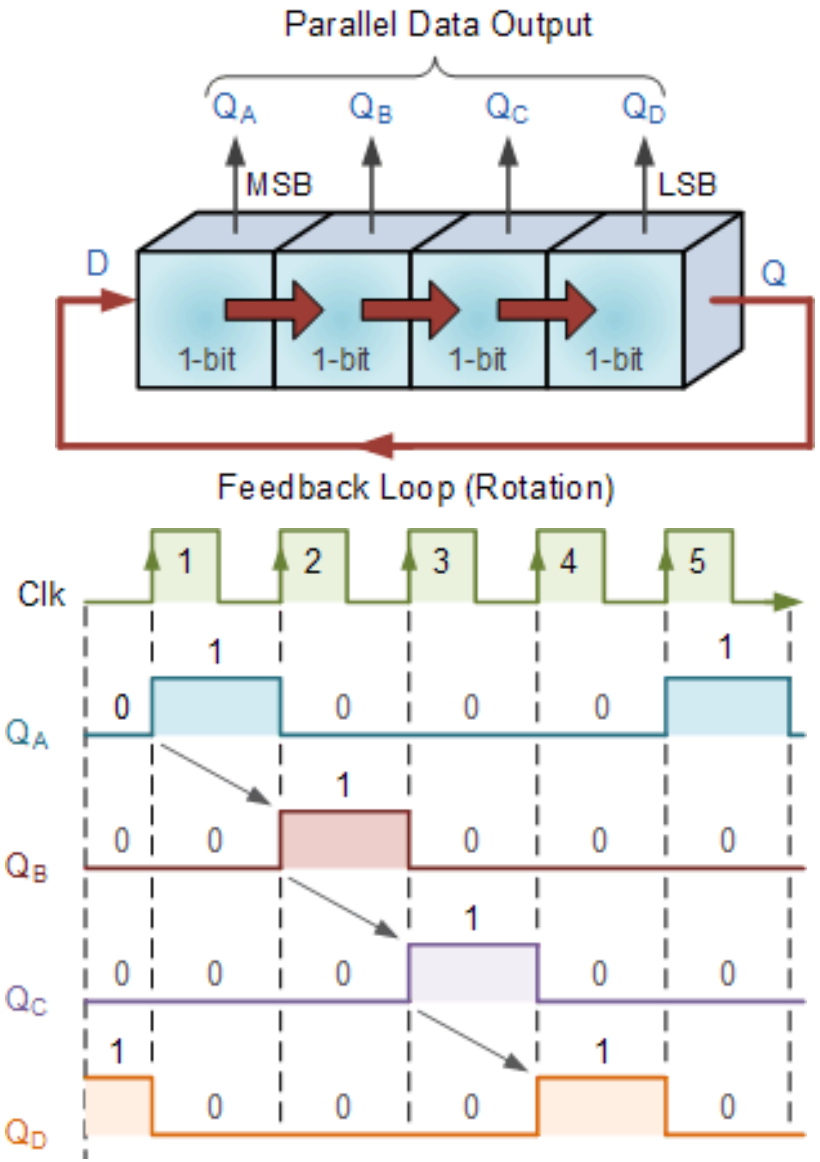
Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

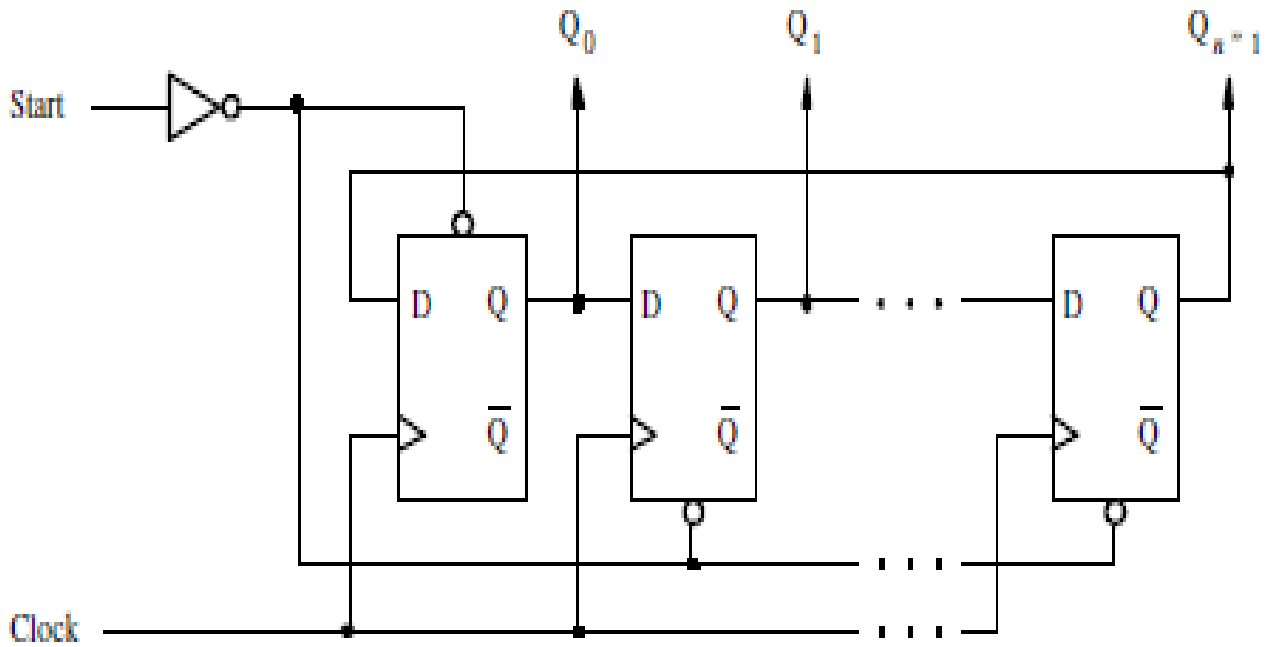
Simple, yet asynchronous circuits !!!

❖Ring Counter



Ring counters are implemented using shift registers. It is essentially a circulating shift register connected so that the last flip-flop shifts its value into the first flip-flop. There is usually only a single 1 circulating in the register, as long as clock pulses are applied. (Starts 1000->0100->0010->0001 repeat)

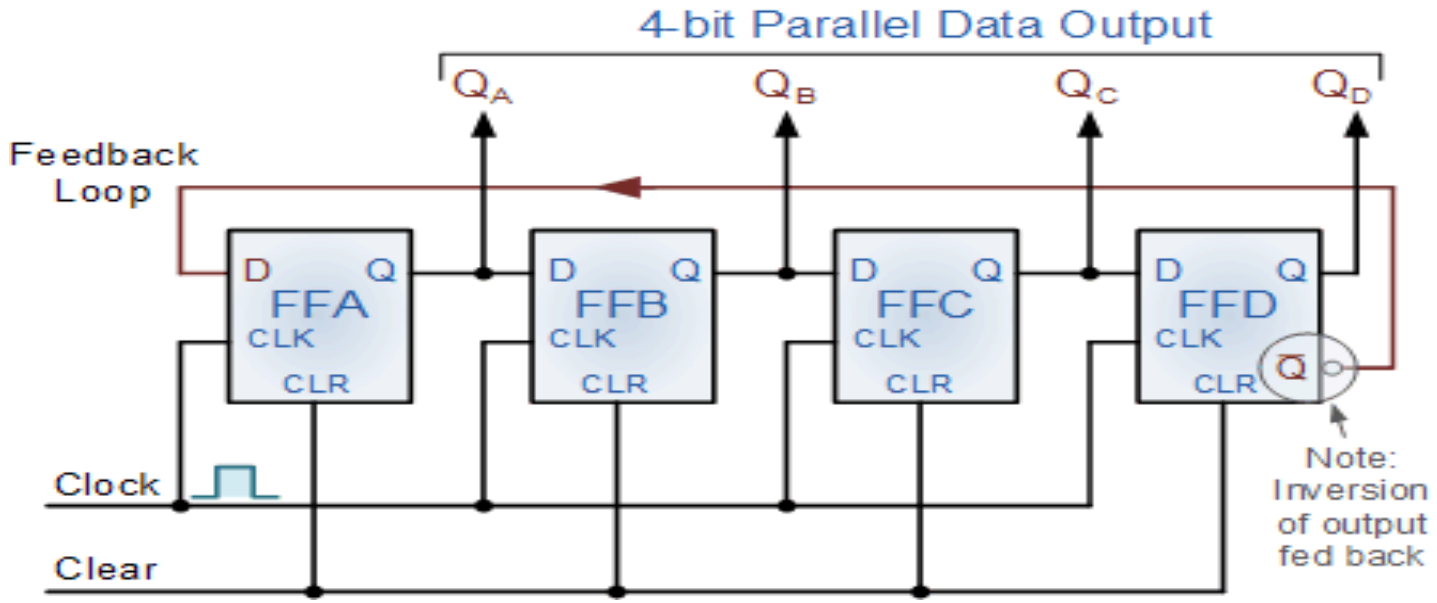

Parallel Data Output
Q_A
Q_B
Q_C
Q_D
MSB
LSB
D
Q
1-bit
1-bit
1-bit
1-bit
Feedback Loop (Rotation)
Clk
1
2
3
4
5
Q_A
Q_B
Q_C
Q_D



(a) An *n*-bit ring counter

Start control signal, which presets the left-most flip-flop to 1 and clears the others to 0.

❖Johnson Counter



The Johnson counter, also known as the twisted-ring counter, is exactly the same as the ring counter except that the inverted output of the last flip-flop is connected to the input of the first flip-flop.

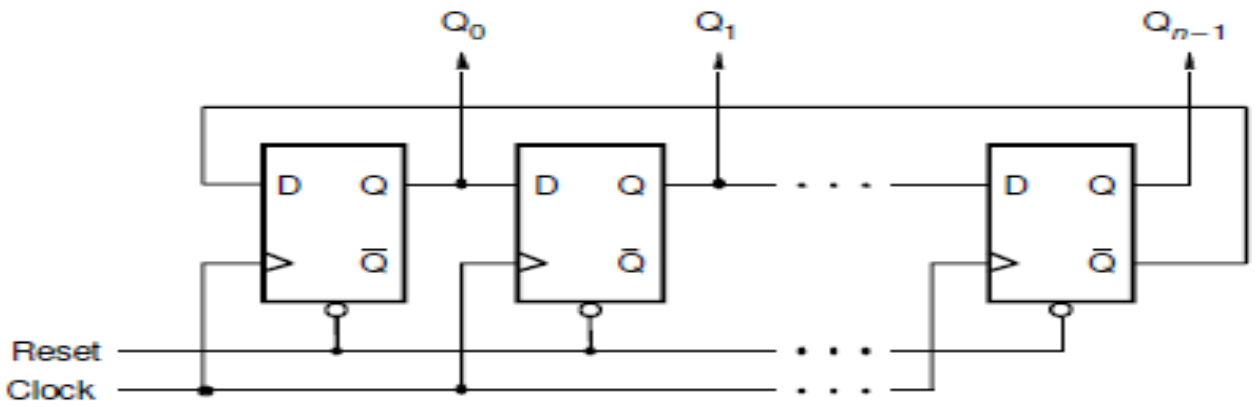
Let's say, starts from 000, 100, 110, 111, 011 and 001, and the sequence is repeated so long as there is input pulse.

Truth Table for a 4-bit Johnson Ring Counter

Clock Pulse No	FFA	FFB	FFC	FFD
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

As well as counting or rotating data around a continuous loop, ring counters can also be used to detect or recognize various patterns or number values within a set of data. By connecting simple logic gates such as the *AND* or the *OR* gates to the outputs of the flip-flops the circuit can be made to detect a set number or value. Standard 2, 3 or 4-stage Johnson ring counters can also be used to divide the frequency of the clock signal by varying their feedback connections and divide-by-3 or divide-by-5 outputs are also available.

Johnson Counter



To initialize the operation of the Johnson counter, it is necessary to reset all flip-flops, as shown in the figure. Observe that neither the Johnson nor the ring counter will generate the desired counting sequence if not initialized properly.

Registers

- A common sequential device: Registers
  - They're a good example of sequential analysis and design
  - They are also frequently used in building larger sequential circuits
- Registers hold larger quantities of data than individual flip-flops
  - Registers are central to the design of modern processors
  - There are many different kinds of registers

Types Of Registers:

- Serial In Serial Out
- Serial In Parallel Out
- Parallel in Serial Out
- Parallel In Parallel Out

What good are registers?

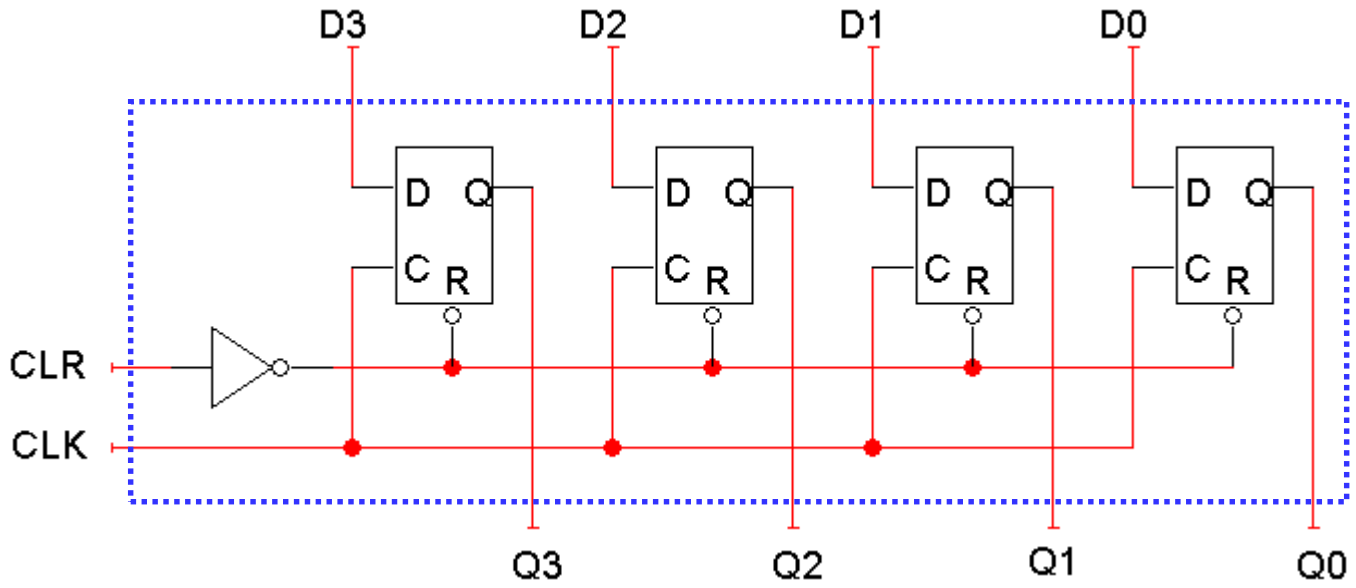
- Flip-flops are limited because they can store only one bit
  - We had to use two flip-flops for our two-bit counter examples
  - Most computers work with integers and single-precision floating-point numbers that are 32-bits long
- A register is an extension of a flip-flop that can store multiple bits
- Registers are commonly used as temporary storage in a processor
  - They are faster and more convenient than main memory
  - More registers can help speed up complex calculations

A basic register

- Basic registers are easy to build. We can store multiple bits just by putting a bunch of flip-flops together!

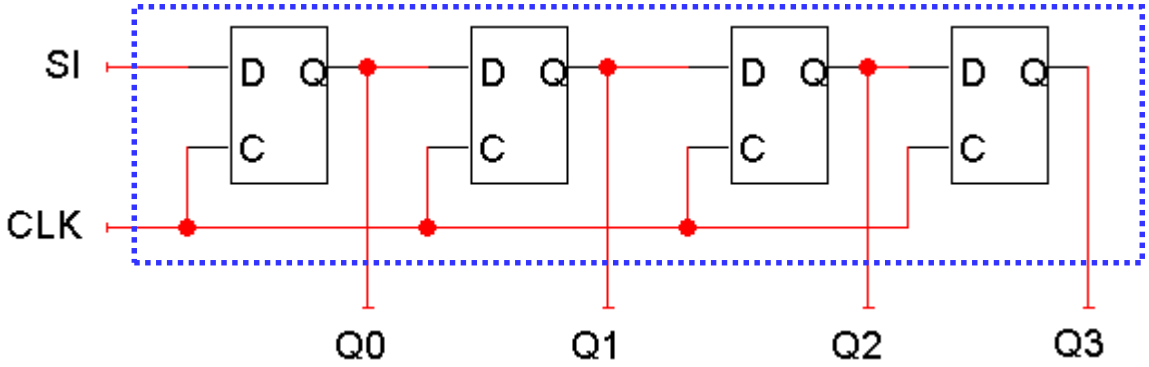
- A 4-bit register is given below

  - This register uses D flip-flops, so it's easy to store data without worrying about flip-flop input equations
  - All the flip-flops share a common CLK and CLR signal



Shift Register

- A shift register "shifts" its output once every clock cycle.



$Q_0(t+1) = SI$
$Q_1(t+1) = Q_0(t)$
$Q_2(t+1) = Q_1(t)$
$Q_3(t+1) = Q_2(t)$

- SI is an input that supplies a new bit to shift "into" the register
- For example, if on some positive clock edge

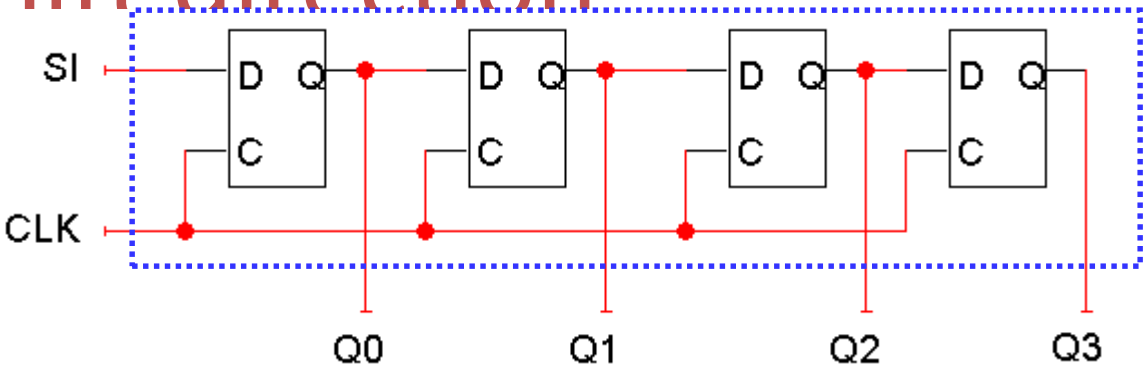
SI = 1
Q_0-Q_3 = 011(

then the next state will be:

Q_0-Q_3 = 101[illegible]

- The current Q_3 (0 in this example) will be lost on the next cycle

Shift direction



$Q_0(t+1) = SI$
$Q_1(t+1) = Q_0(t)$
$Q_2(t+1) = Q_1(t)$
$Q_3(t+1) = Q_2(t)$

- The circuit and example make it look like the register shifts "right."

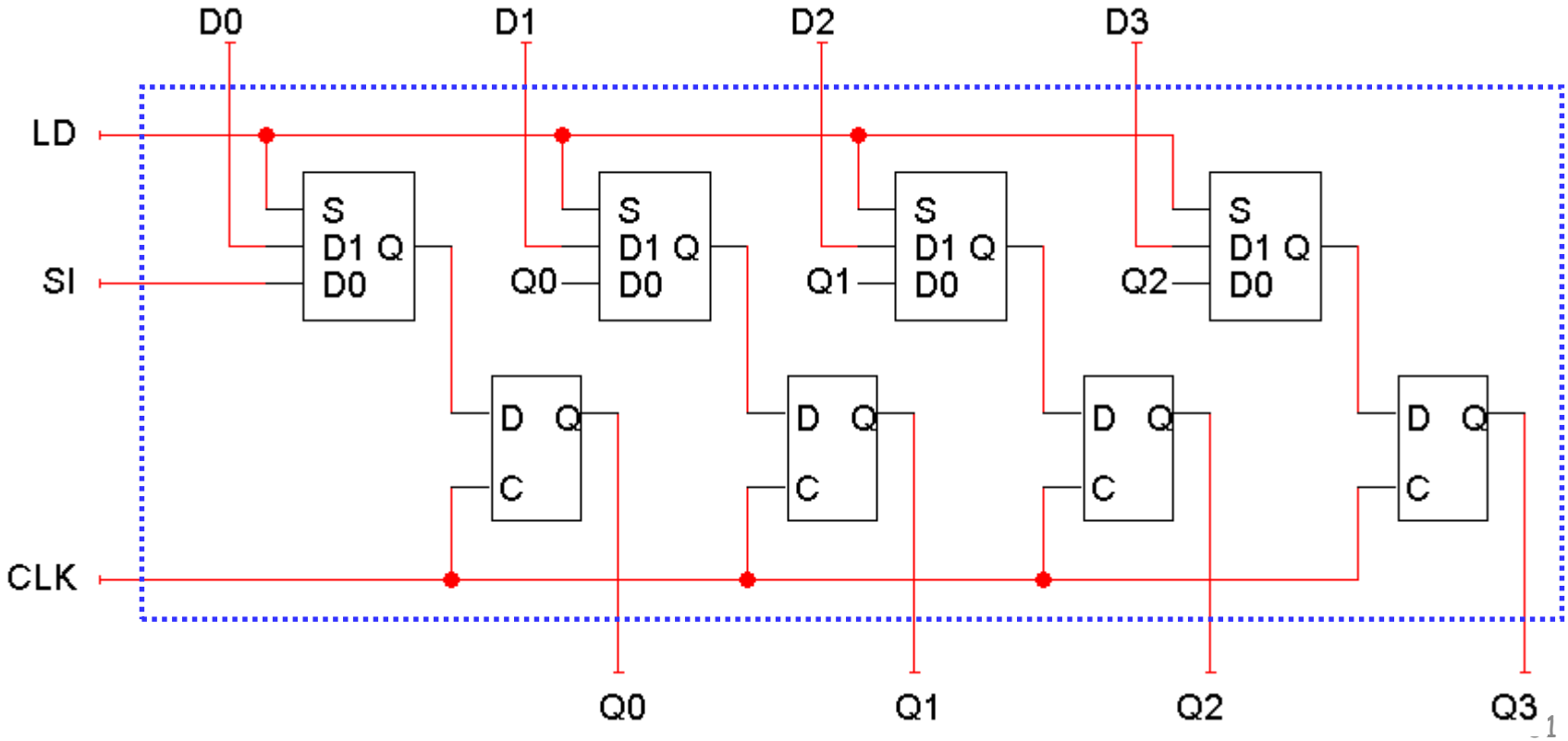
Present Q_0-Q_3	SI	Next Q_0-Q_3
ABCD	X	XABC

- But it really depends on your interpretation of the bits. If you consider Q3 to be the most significant bit, then the register shifts in the *opposite* direction!

Present Q_3-Q_0	SI	Next Q_3-Q_0
DCBA	X	CBAX

Shift register with parallel load

- We can add a parallel load, just like we did for regular registers
  - When LD = 0, the flip-flop inputs will be $SIQ_0Q_1Q_2$, so the register shifts on the next positive clock edge
  - When LD = 1, the flip-flop inputs are D_0-D_3, and a new value is loaded into the shift register, on the next positive clock edge



Shift register with parallel load

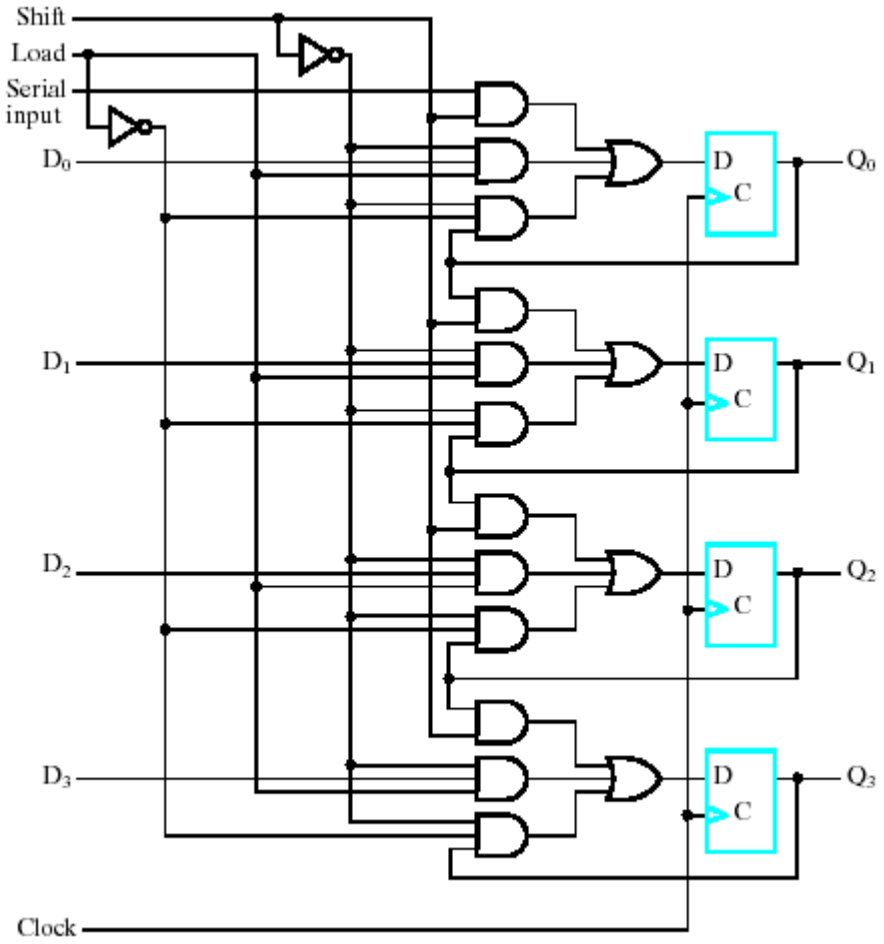
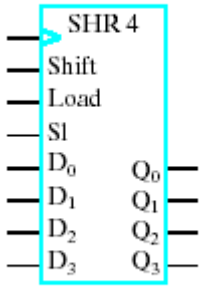


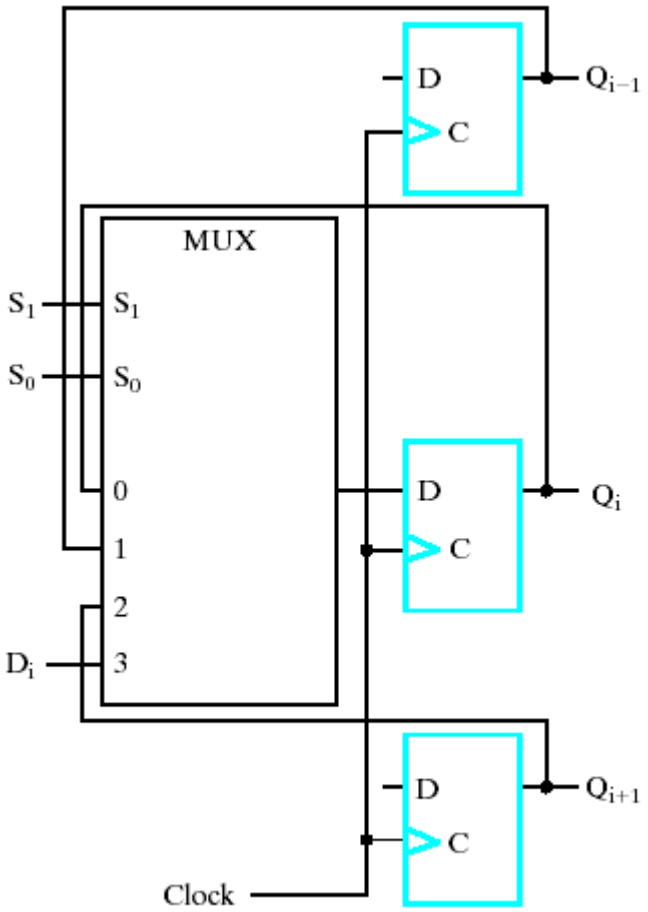
TABLE 7-6
Function Table for the Register of Figure 7-10

Shift	Load	Operation
0	0	No change
0	1	Load parallel data
1	X	Shift down from Q_0 to Q_3



(b) Symbol

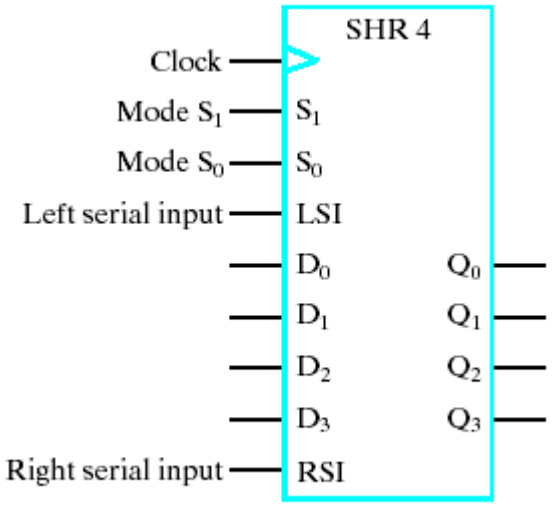
Bidirectional Shift Register



(a) Logic diagram of one typical stage

TABLE 7-7
Function Table for the Register of Figure 7-7

Mode control		Register Operation
S_1	S_0	
0	0	No change
0	1	Shift down
1	0	Shift up
1	1	Parallel load



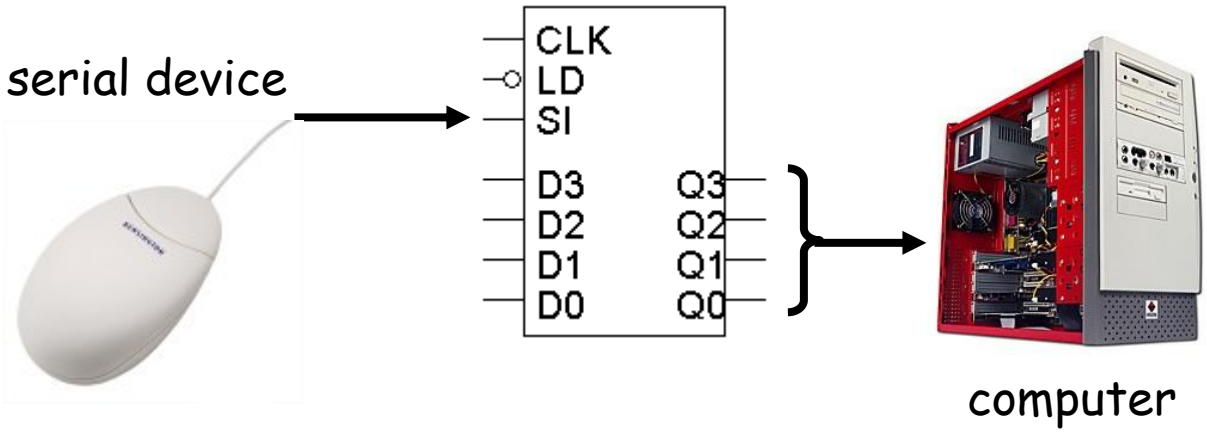
(b) Symbol

Serial data transfer

- One application of shift registers is converting between “serial data” and “parallel data”
- Computers typically work with multiple-bit quantities
  - ASCII text characters are 8 bits long
  - Integers, single-precision floating-point numbers, and screen pixels are up to 32 bits long
- But sometimes it’s necessary to send or receive data serially, or one bit at a time. Some examples include:
  - Input devices such as keyboards and mice
  - Output devices like printers
  - Any serial port, USB or Firewire device transfers data serially
  - Recent switch from Parallel ATA to Serial ATA in hard drives

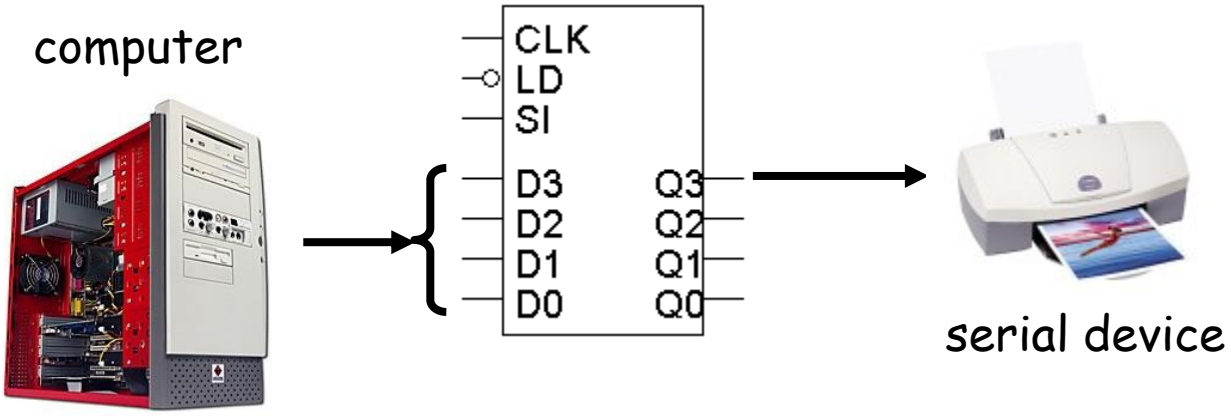
Receiving serial data

- To *receive* serial data using a shift register:
  - The serial device is connected to the register's SI input
  - The shift register outputs Q3-Q0 are connected to the computer
- The serial device transmits one bit of data per clock cycle
  - These bits go into the SI input of the shift register
  - After four clock cycles, the shift register will hold a four-bit word
- The computer then reads all four bits at once from the Q3-Q0 outputs.



Sending data serially

- To *send* data serially with a shift register, you do the opposite:
  - The CPU is connected to the register's D inputs
  - The shift output (Q3 in this case) is connected to the serial device
- The computer first stores a four-bit word in the register, in one cycle
- The serial device can then read the shift output
  - One bit appears on Q3 on each clock cycle
  - After four cycles, the entire four-bit word will have been sent



Registers in Modern Hardware

- Registers store data in the CPU
  - Used to supply values to the ALU
  - Used to store the results
- If we can use registers, why bother with RAM?

CPU	GPR's	Size	L1 Cache	L2 Cache
Pentium 4	8	32 bits	8 KB	512 KB
Athlon XP	8	32 bits	64 KB	512 KB
Athlon 64	16	64 bits	64 KB	1024 KB
PowerPC 970 (G5)	32	64 bits	64 KB	512 KB
Itanium 2	128	64 bits	16 KB	256 KB
MIPS R14000	32	64 bits	32 KB	16 MB

Answer: Registers are expensive!

- Registers occupy the most expensive space on a chip – the core
- L1 and L2 are very fast RAM – but not as fast as registers.

Registers summary

- A register is a special state machine that stores multiple bits of data
- Several variations are possible:
  - Parallel loading to store data into the register
  - Shifting the register contents either left or right
  - Counters are considered a type of register too!
- One application of shift registers is converting between serial and parallel data
- Most programs need more storage space than registers provide
  - We'll introduce RAM to address this problem
- Registers are a central part of modern processors