Sequential Logic

Flip-flops and Latches

Sequential Logic

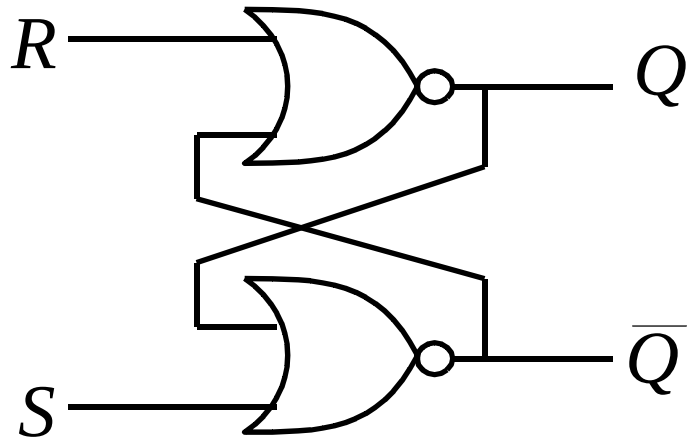
- The logic circuits discussed previously are known as *combinational*, in that the output depends only on the condition of the latest inputs
- However, we will now introduce a type of logic where the output depends not only on the latest inputs, but also on the condition of earlier inputs. These circuits are known as *sequential*, and implicitly they contain *memory* elements

Memory Elements

- A memory stores data – usually one bit per element
- A snapshot of the memory is called the *state*
- A one bit memory is often called a *bistable*, i.e., it has 2 stable internal states
- *Flip-flops* and *latches* are particular implementations of bistables

RS Latch

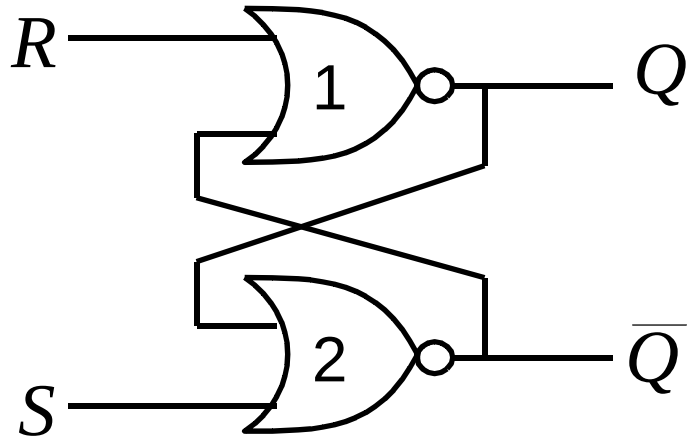
- An RS latch is a memory element with 2 inputs: Reset (R) and Set (S) and 2 outputs: Q and $\overline{Q}$.



S R	Q' $\overline{Q}'$	comment
0 0	Q $\overline{Q}$	hold
0 1	0 1	reset
1 0	1 0	set
1 1	0 0	illegal

Where Q' is the next state and Q is the current state

RS Latch - Operation

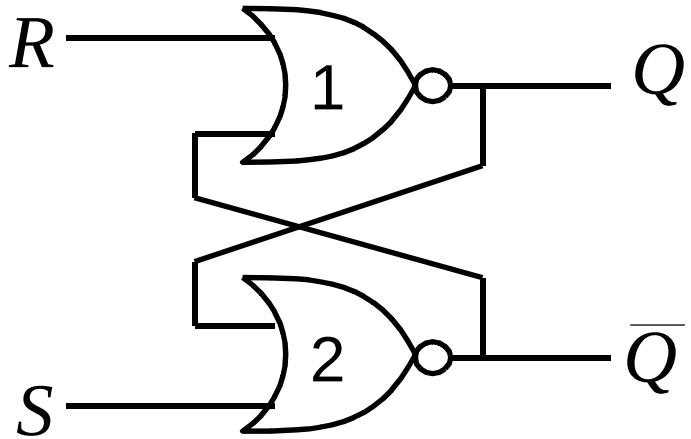


NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- R = 1 and S = 0
  - Gate 1 output in ‘always 0’ condition, $Q = 0$
  - Gate 2 in ‘complement’ condition, so $\overline{Q} = 1$
- This is the (R)eset condition

RS Latch - Operation

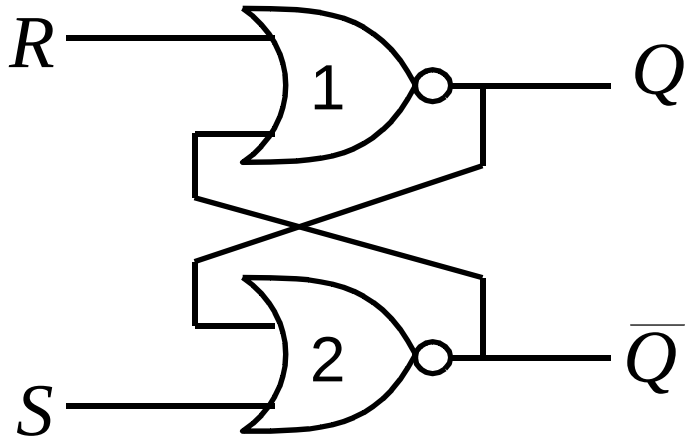


NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- S = 0 and R to 0
  - Gate 2 remains in ‘complement’ condition, $\overline{Q} = 1$
  - Gate 1 into ‘complement’ condition, $Q = 0$
- This is the hold condition

RS Latch - Operation

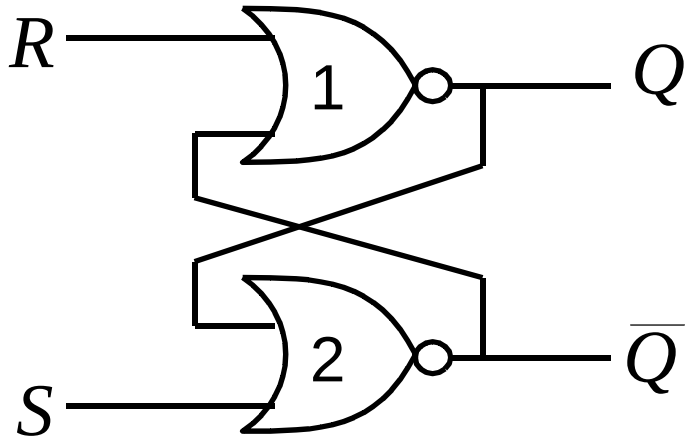


NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- S = 1 and R = 0
  - Gate 1 into ‘complement’ condition, $Q = 1$
  - Gate 2 in ‘always 0’ condition, $\overline{Q} = 0$
- This is the (S)et condition

RS Latch - Operation



NOR truth table

a	b	y	
0	0	1	b complemented
0	1	0	
1	0	0	always 0
1	1	0	

- S = 1 and R = 1
  - Gate 1 in ‘always 0’ condition, $Q = 0$
  - Gate 2 in ‘always 0’ condition, $\overline{Q} = 0$
- This is the illegal condition

RS Latch – State Transition Table

- A *state transition table* is an alternative way of viewing its operation

Q	S	R	Q'	comment
0	0	0	0	hold
0	0	1	0	reset
0	1	0	1	set
0	1	1	0	illegal
1	0	0	1	hold
1	0	1	0	reset
1	1	0	1	set
1	1	1	0	illegal

- A state transition table can also be expressed in the form of a *state diagram*

RS Latch – State Diagram

- A *state diagram* in this case has 2 states, i.e., Q=0 and Q=1
- The state diagram shows the input conditions required to transition between states. In this case we see that there are 4 possible transitions
- We will consider them in turn

RS Latch – State Diagram

Q	S	R	Q'	comment
0	0	0	0	hold
0	0	1	0	reset
0	1	0	1	set
0	1	1	0	illegal
1	0	0	1	hold
1	0	1	0	reset
1	1	0	1	set
1	1	1	0	illegal

$Q = 0 \quad Q' = 0$

From the table we can see:

$$\bar{S}.\bar{R} + \bar{S}.R + S.R =$$

$$\bar{S}.(\bar{R} + R) + S.R = \bar{S} + S.R =$$

$$(\bar{S} + S).(\bar{S} + R) = \bar{S} + R$$

$Q = 1 \quad Q' = 1$

From the table we can see:

$$\bar{S}.\bar{R} + S.\bar{R} = \bar{R}.(\bar{S} + S) =$$

$$\bar{R}$$

RS Latch – State Diagram

Q	S	R	Q'	comment
0	0	0	0	hold
0	0	1	0	reset
0	1	0	1	set
0	1	1	0	illegal
1	0	0	1	hold
1	0	1	0	reset
1	1	0	1	set
1	1	1	0	illegal

$Q = 1 \quad Q' = 0$

From the table we can see:

$\overline{S}.R + S.R =$

$R.(\overline{S} + S) = R$

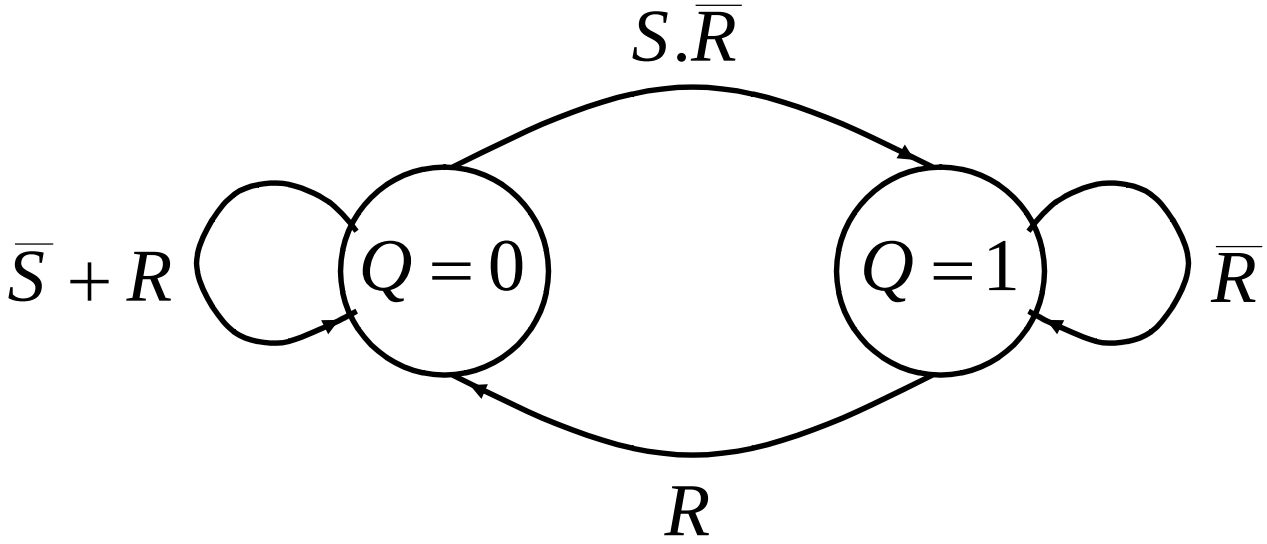
$Q = 0 \quad Q' = 1$

From the table we can see:

$S.\overline{R}$

RS Latch – State Diagram

- Which gives the following state diagram:



- A similar diagram can be constructed for the $\overline{Q}$ output
- We will see later that state diagrams are a useful tool for designing sequential systems

Clocks and Synchronous Circuits

- For the RS latch we have just described, we can see that the output state changes occur directly in response to changes in the inputs. This is called *asynchronous* operation
- However, virtually all sequential circuits currently employ the notion of *synchronous* operation, that is, the output of a sequential circuit is constrained to change only at a time specified by a global *enabling* signal. This signal is generally known as the system *clock*

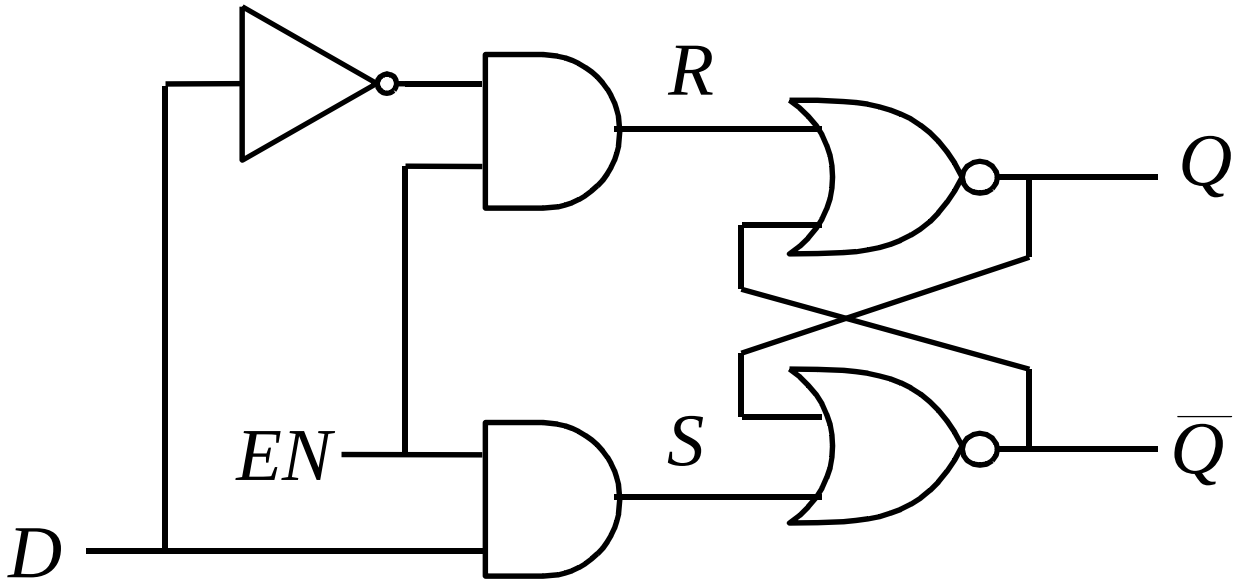
Clocks and Synchronous Circuits

- The Clock: What is it and what is it for?
  - Typically it is a square wave signal at a particular frequency
  - It imposes order on the state changes
  - Allows lots of states to appear to update simultaneously
- How can we modify an asynchronous circuit to act synchronously, i.e., in synchronism with a clock signal?

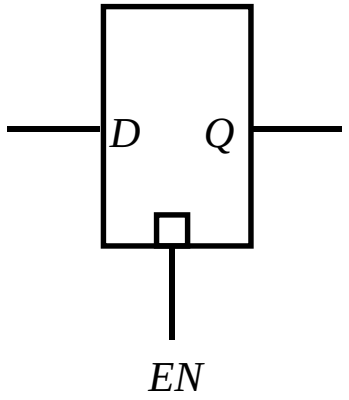
Transparent D Latch

- We now modify the RS Latch such that its output state is only permitted to change when a valid enable signal (which could be the system clock) is present
- This is achieved by introducing a couple of AND gates in cascade with the R and S inputs that are controlled by an additional input known as the *enable* (EN) input.

Transparent D Latch



Symbol

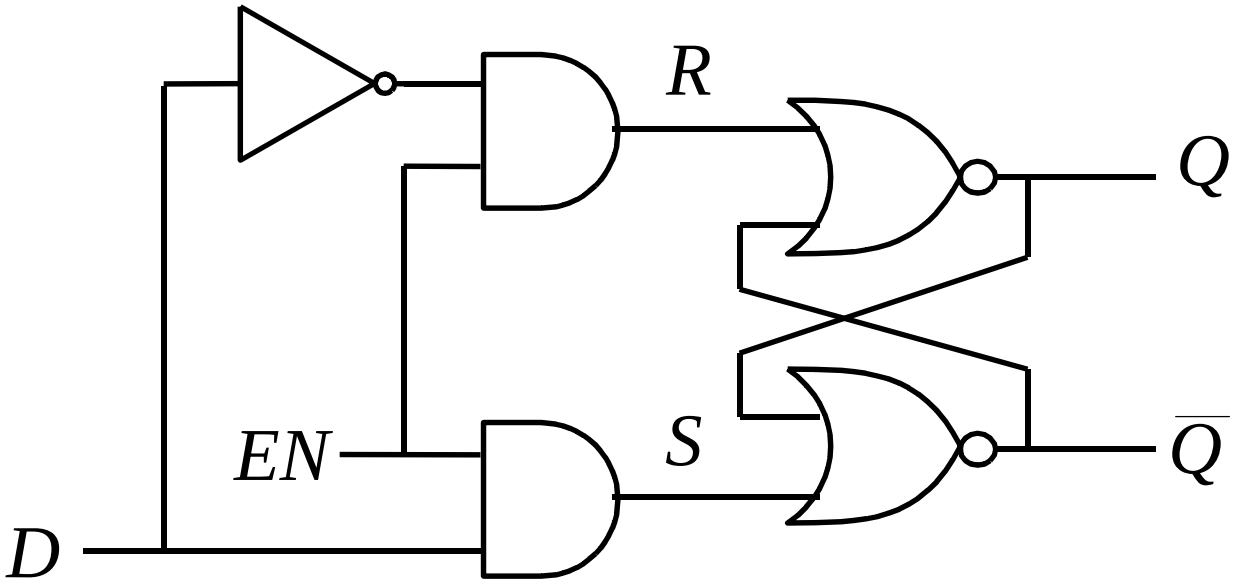


- See from the AND truth table:
  - if one of the inputs, say a is 0, the output is always 0
  - Output follows b input if a is 1
- The complement function ensures that R and S can never be 1 at the same time, i.e., illegal avoided

AND truth table

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Transparent D Latch



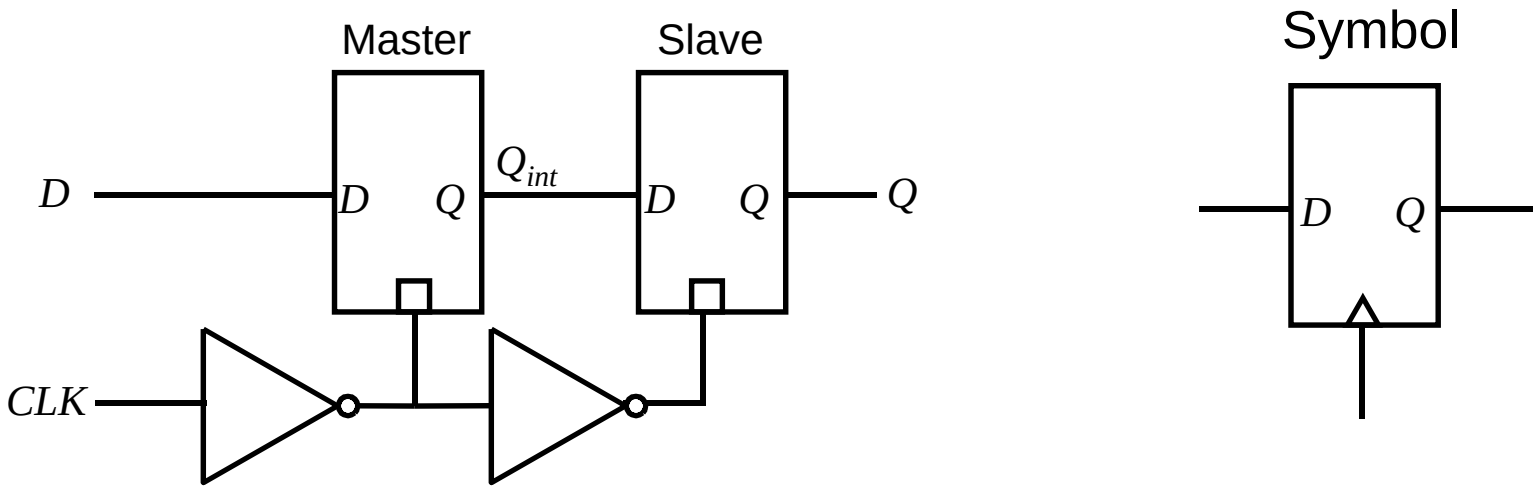
D	EN	Q'	$\overline{Q}'$	comment
X	0	Q	$\overline{Q}$	RS hold
0	1	0	1	RS reset
1	1	1	0	RS set

- See Q follows D input provided EN=1.
  If EN=0, Q maintains previous state

Master-Slave Flip-Flops

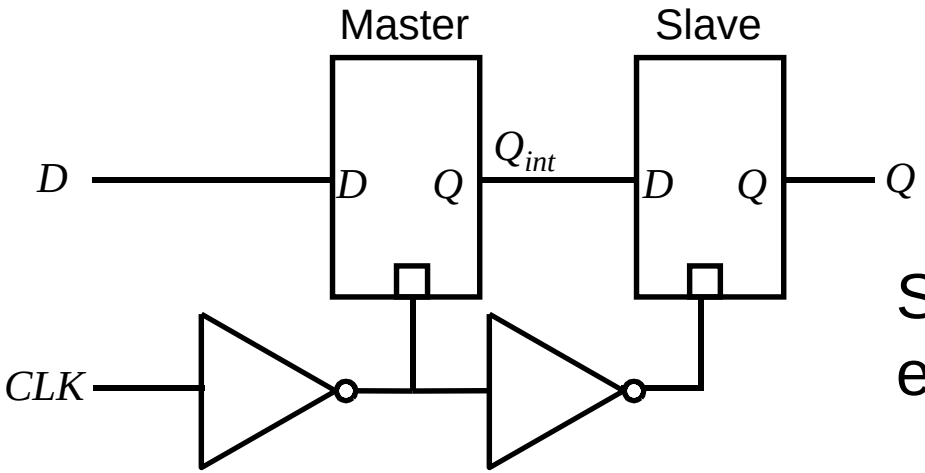
- The transparent D latch is so called '*level*' triggered. We can see it exhibits transparent behaviour if *EN*=1. It is often more simple to design sequential circuits if the outputs change only on the either rising (positive going) or falling (negative going) '*edges*' of the clock (i.e., enable) signal
- We can achieve this kind of operation by combining 2 transparent D latches in a so called *Master-Slave* configuration

Master-Slave D Flip-Flop

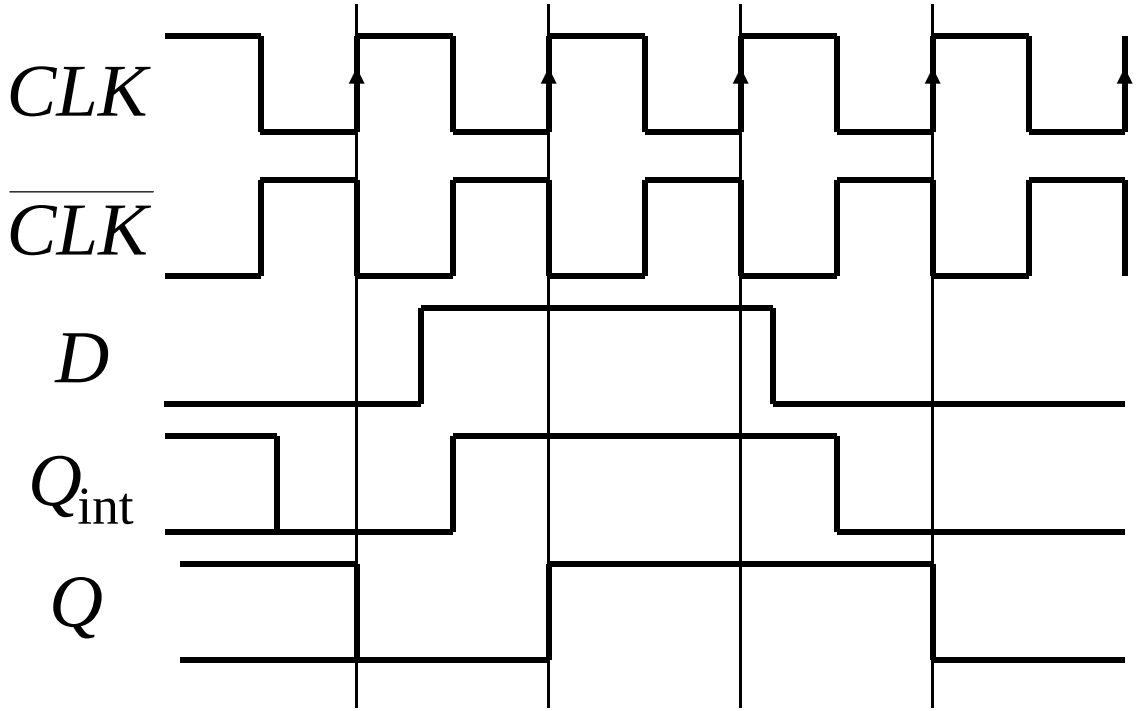


- To see how this works, we will use a timing diagram
- Note that both latch inputs are effectively connected to the clock signal (admittedly one is a complement of the other)

Master-Slave D Flip-Flop



See Q changes on rising edge of CLK



Note propagation delays have been neglected in the timing diagram

D Flip-Flops

- The Master-Slave configuration has now been superseded by new F-F circuits which are easier to implement and have better performance
- When designing synchronous circuits it is best to use truly edge triggered F-F devices
- We will not consider the design of such F-Fs on this course

Other Types of Flip-Flops

- Historically, other types of Flip-Flops have been important, e.g., J-K Flip-Flops and T-Flip-Flops
- However, J-K FFs are a lot more complex to build than D-types and so have fallen out of favour in modern designs, e.g., for field programmable gate arrays (FPGAs) and VLSI chips

Other Types of Flip-Flops

- Consequently we will only consider synchronous circuit design using D-type FFs
- However for completeness we will briefly look at the truth table for J-K and T type FFs

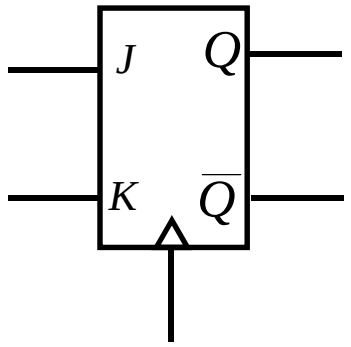
J-K Flip-Flop

- The J-K FF is similar in function to a clocked RS FF, but with the illegal state replaced with a new ‘toggle’ state

J K	Q' $\overline{Q}'$	comment
0 0	Q $\overline{Q}$	hold
0 1	0 1	reset
1 0	1 0	set
1 1	$\overline{Q}$ Q	toggle

Where Q' is the next state and Q is the current state

Symbol



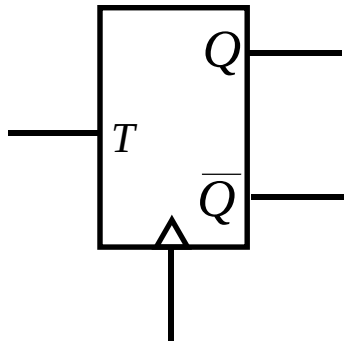
T Flip-Flop

- This is essentially a J-K FF with its J and K inputs connected together and renamed as the T input

T	Q' $\overline{Q}'$	comment
0	Q $\overline{Q}$	hold
1	$\overline{Q}$ Q	toggle

Where Q' is the next state
and Q is the current state

Symbol



Asynchronous Inputs

- It is common for the FF types we have mentioned to also have additional so called ‘asynchronous’ inputs
- They are called asynchronous since they take effect independently of any clock or enable inputs
- Reset/Clear – force Q to 0
- Preset/Set – force Q to 1
- Often used to force a synchronous circuit into a known state, say at start-up.

Timing

- Various timings must be satisfied if a FF is to operate properly:
  - *Setup time*: Is the minimum duration that the data must be stable at the input before the clock edge
  - *Hold time*: Is the minimum duration that the data must remain stable on the FF input after the clock edge

Applications of Flip-Flops

- Counters
  - A clocked sequential circuit that goes through a predetermined sequence of states
  - A commonly used counter is an n-bit binary counter. This has n FFs and 2^n states which are passed through in the order 0, 1, 2, ….2^n-1, 0, 1, .
  - Uses include:
    - Counting
    - Producing delays of a particular duration
    - Sequencers for control logic in a processor
    - Divide by m counter (a divider), as used in a digital watch

Applications of Flip-Flops

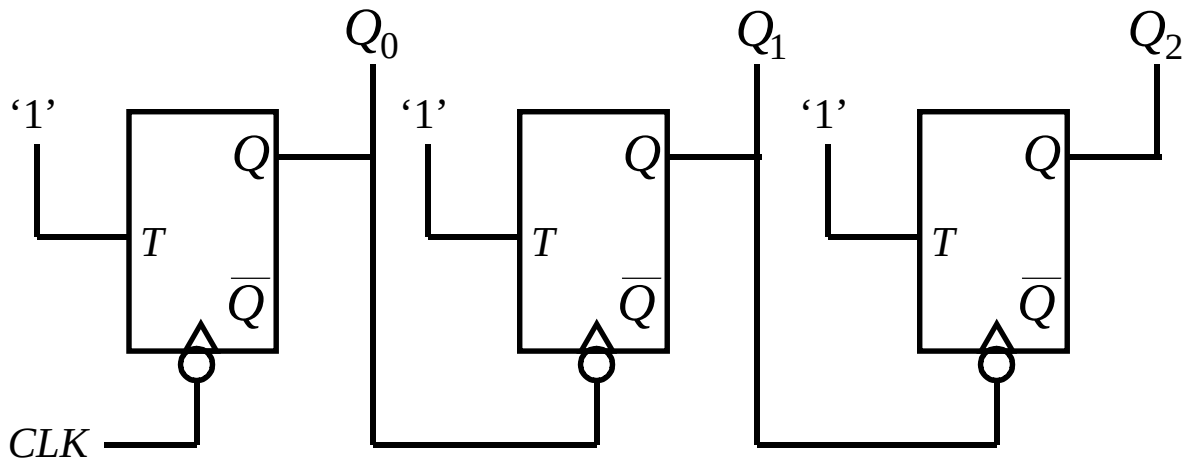
- Memories, e.g.,
  - Shift register
    - Parallel loading shift register : can be used for parallel to serial conversion in serial data communication
    - Serial in, parallel out shift register: can be used for serial to parallel conversion in a serial data communication system.

Counters

- In most books you will see 2 basic types of counters, namely *ripple* counters and *synchronous* counters
- In this course we are concerned with synchronous design principles. Ripple counters do not follow these principles and should generally be avoided if at all possible. We will now look at the problems with ripple counters

Ripple Counters

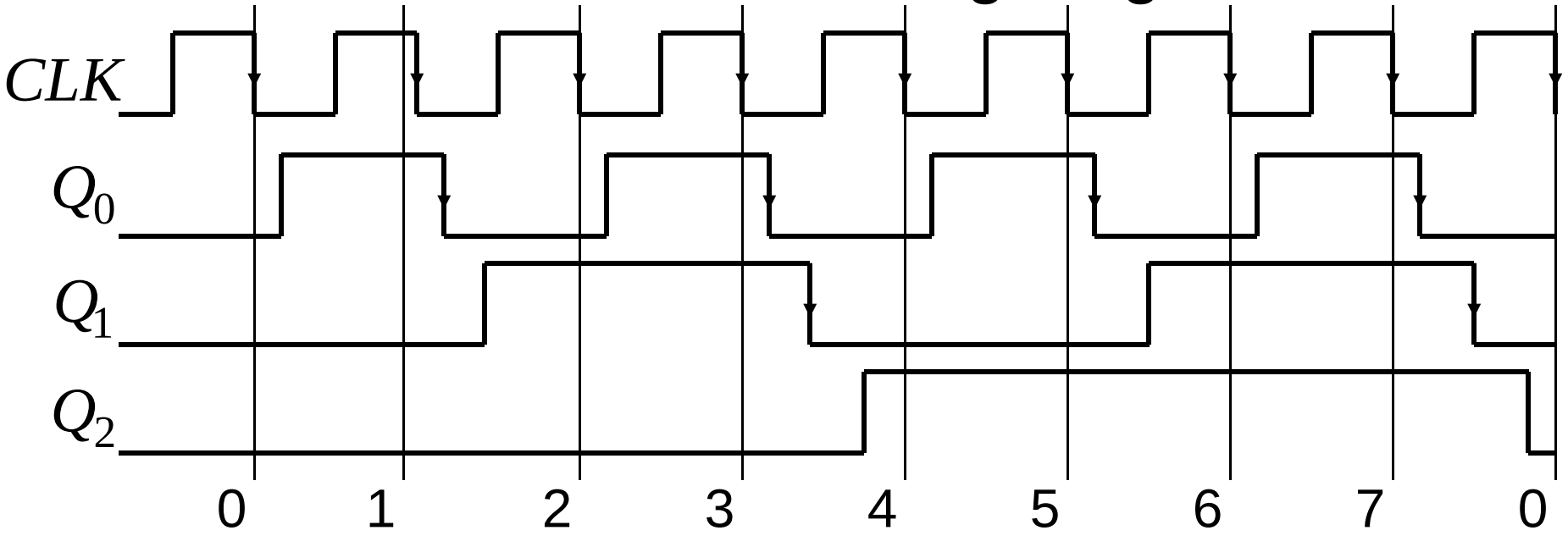
- A ripple counter can be made be cascading together negative edge triggered T-type FFs operating in ‘toggle’ mode, i.e., T =1



- See that the FFs are not clocked using the same clock, i.e., this is **not** a synchronous design. This gives some problems….

Ripple Counters

- We will now draw a timing diagram



- Problems:

See outputs do not change at the same time, i.e., synchronously. So hard to know when count output is actually valid.

Propagation delay builds up from stage to stage, limiting maximum clock speed before miscounting occurs.

Ripple Counters

- If you observe the frequency of the counter output signals you will note that each has half the frequency, i.e., double the repetition period of the previous one. This is why counters are often known as dividers
- Often we wish to have a count which is not a power of 2, e.g., for a BCD counter (0 to 9).To do this:
  - use FFs having a Reset/Clear input
  - Use an AND gate to detect the count of 10 and use its output to Reset the FFs

Synchronous Counters

- Owing to the problems identified with ripple counters, they should not usually be used to implement counter functions
- It is recommended that *synchronous* counter designs be used
- In a synchronous design
  - all the FF clock inputs are directly connected to the clock signal and so all FF outputs change at the same time, i.e., *synchronously*
  - more complex combinational logic is now needed to generate the appropriate FF input signals (which will be different depending upon the type of FF chosen)

Synchronous Counters

- We will now investigate the design of synchronous counters
- We will consider the use of D-type FFs only, although the technique can be extended to cover other FF types.
- As an example, we will consider a 0 to 7 up-counter

Synchronous Counters

- To assist in the design of the counter we will make use of a modified *state transition table.* This table has additional columns that define the required FF inputs (or e*xcitation* as it is known)
  - Note we have used a state transition table previously when determining the state diagram for an RS latch
- We will also make use of the so called '*excitation table*' for a D-type FF
- First however, we will investigate the so called *characteristic table* and *characteristic equation* for a D-type FF

Characteristic Table

- In general, a characteristic table for a FF gives the next state of the output, i.e., Q' in terms of its current state Q and current inputs

Q	D	Q'
0	0	0
0	1	1
1	0	0
1	1	1

Which gives the characteristic equation,

$$Q' = D$$

i.e., the next output state is equal to the current input value

Since Q' is independent of Q the characteristic table can be rewritten as

D	Q'
0	0
1	1

Excitation Table

- The characteristic table can be modified to give the excitation table. This table tells us the required FF input value required to achieve a particular next state from a given current state

Q	Q'	D
0	0	0
0	1	1
1	0	0
1	1	1

As with the characteristic table it can be seen that Q', does not depend upon, Q , however this is not generally true for other FF types, in which case, the excitation table is more useful. Clearly for a D-FF, $D = Q'$

Characteristic and Excitation Tables

- Characteristic and excitation tables can be determined for other FF types.
- These should be used in the design process if D-type FFs are not used
- We will now determine the modified state transition table for the example 0 to 7 up-counter

Modified State Transition Table

- In addition to columns representing the current and desired next states (as in a conventional state transition table), the modified table has additional columns representing the required FF inputs to achieve the next desired FF states

Modified State Transition Table

- For a 0 to 7 counter, 3 D-type FFs are needed

Current state	Next state	FF inputs
$Q_2 Q_1 Q_0$	$Q_2' Q_1' Q_0'$	$D_2 D_1 D_0$
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

The procedure is to:

- Write down the desired count sequence in the current state columns
- Write down the required next states in the next state columns
- Fill in the FF inputs required to give the defined next state

Note: Since $Q' = D$ (or $D = Q'$) for a D-FF, the required FF inputs are identical to the Next state

Synchronous Counter Example

- Also note that if we are using D-type FFs, it is not necessary to explicitly write out the FF input columns, since we know they are identical to those for the next state
- To complete the design we now have to determine appropriate combinational logic circuits which will generate the required FF inputs from the current states
- We can do this from inspection, using Boolean algebra or using K-maps.

Synchronous Counter Example

Current state			Next state			FF inputs		
Q_2	Q_1	Q_0	Q_2'	Q_1'	Q_0'	D_2	D_1	D_0
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

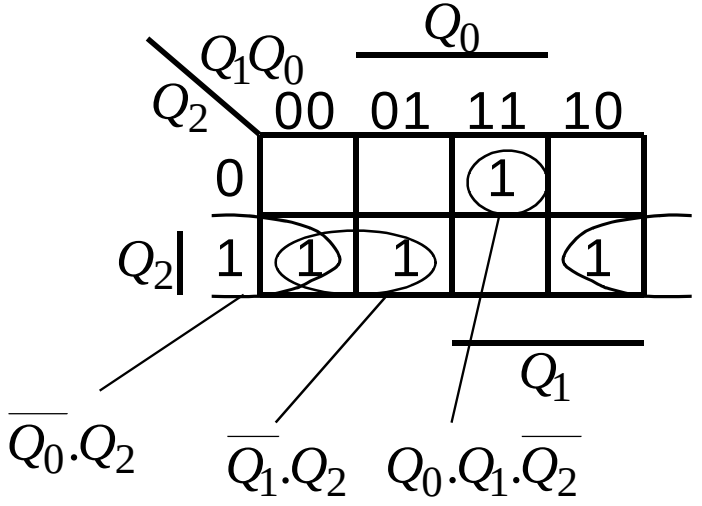
By inspection,

$$D_0 = \overline{Q_0}$$

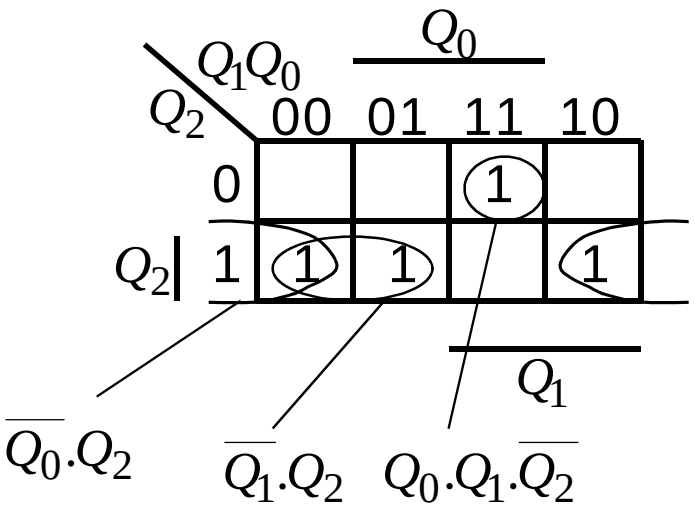
Note: FF_0 is toggling

Also, $D_1 = Q_0 \oplus Q_1$

Use a K-map for D_2 ,



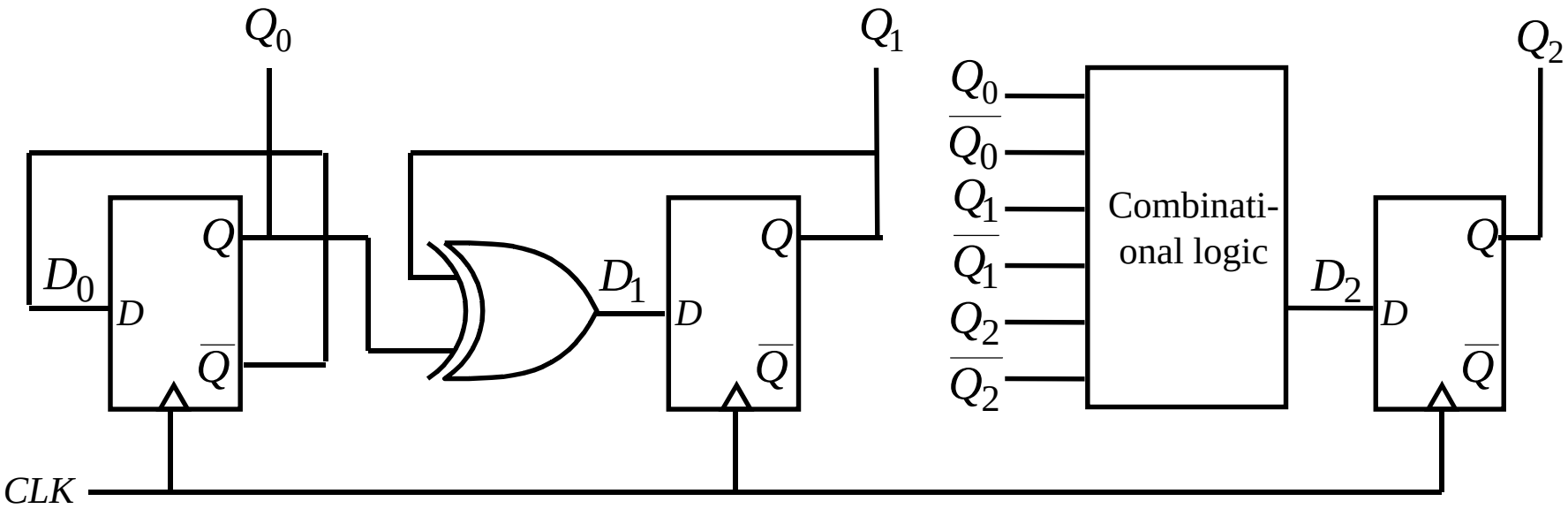
Synchronous Counter Example



So,

$$D_2 = \overline{Q_0}.Q_2 + \overline{Q_1}.Q_2 + Q_0.Q_1.\overline{Q_2}$$

$$D_2 = Q_2.(\overline{Q_0}. + \overline{Q_1}) + Q_0.Q_1.\overline{Q_2}$$

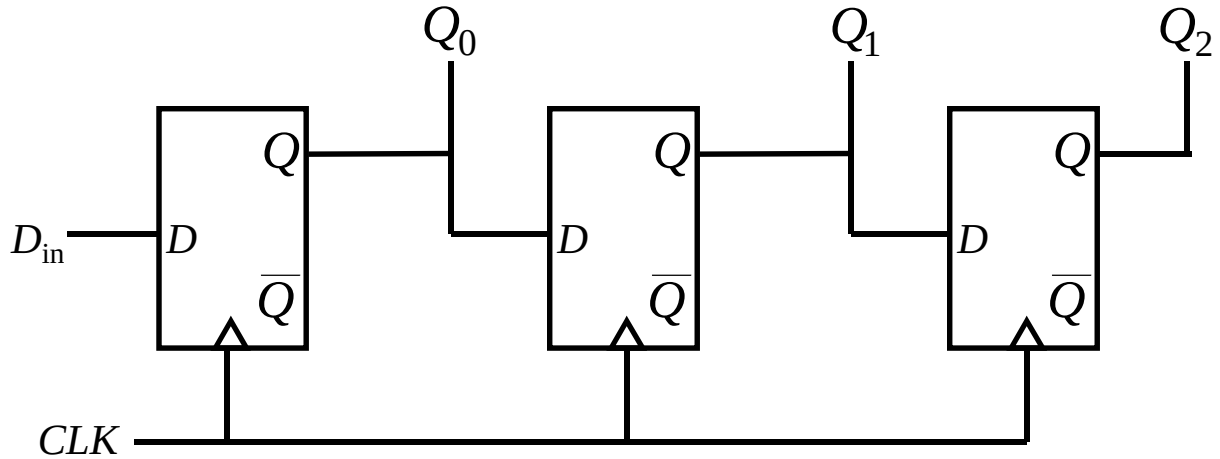


Synchronous Counter

- A similar procedure can be used to design counters having an arbitrary count sequence
  - Write down the state transition table
  - Determine the FF excitation (easy for D-types)
  - Determine the combinational logic necessary to generate the required FF excitation from the current states – **Note:** remember to take into account any unused counts since these can be used as don’t care states when determining the combinational logic circuits

Shift Register

- A shift register can be implemented using a chain of D-type FFs

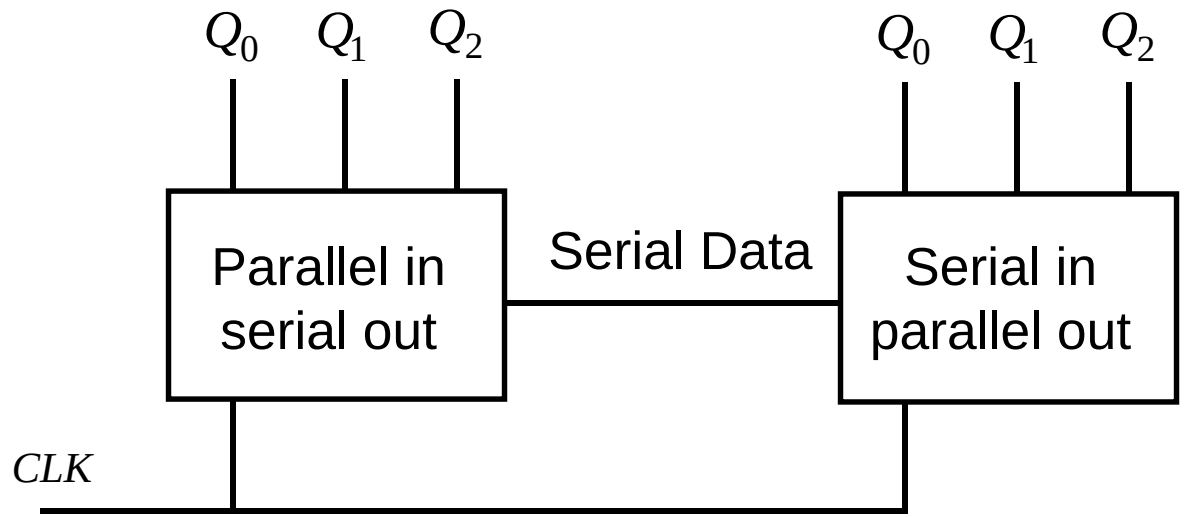


- Has a serial input, D_{in} and parallel output Q_0, Q_1 and Q_2.
- See data moves one position to the right on application of clock edge

Shift Register

- Preset and Clear inputs on the FFs can be utilised to provide a parallel data input feature
- Data can then be clocked out through Q_2 in a serial fashion, i.e., we now have a parallel in, serial out arrangement
- This along with the previous serial in, parallel out shift register arrangement can be used as the basis for a serial data link

Serial Data Link



- One data bit at a time is sent across the serial data link
- See less wires are required than for a parallel data link