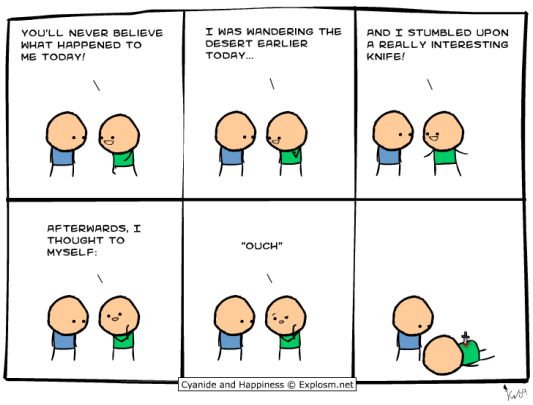
6. Sequential Logic & Flipflops

6.1 Combinational & Sequential Logic

- **Combinational logic** is where the output of a digital circuit is produced by combining the inputs in a manner defined by the logic.

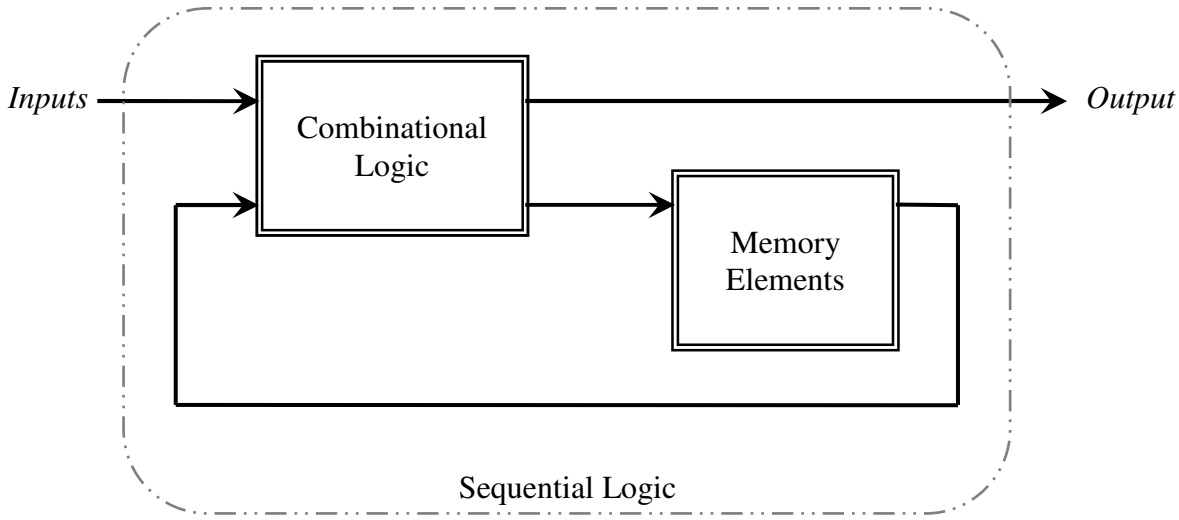
- Until now, all our circuits have been based on combinational logic only. However many digital systems contain sequential logic.



- **Sequential logic** is where the **output** of a digital circuit **depends on both the inputs and also on the previous output state** of the circuit.

- Such a system is described as *having memory* because it can remember its output state.

- The following diagram illustrates the fundamental difference between combinational and sequential logic circuits:



- Sequential systems are categorised in two ways, namely synchronous and asynchronous.

- A **synchronous** sequential circuit is where its behaviour depends on knowledge of the inputs **at a discrete instant of time,** usually determined by a **clock**.

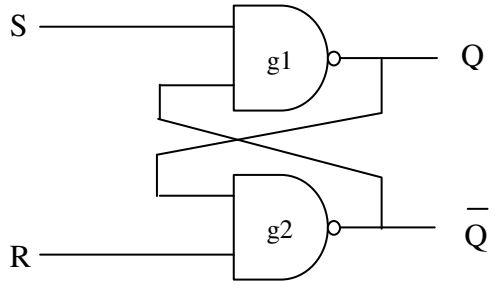
- An **asynchronous** sequential circuit is where its behaviour can be affected by changes in the input signals at **any instant of time**.

- One of the most basic but fundamental sequential circuit is the flip-flop, which is a bistable device, i.e. it settles in one of two possible states (typically SET and RESET).

- We are now going to look at four common types of flipflops, namely SR, D, JK and T.

6.2 The Asynchronous SR Flipflop

- The following circuit represents a NAND implementation of an asynchronous SR flipflop:

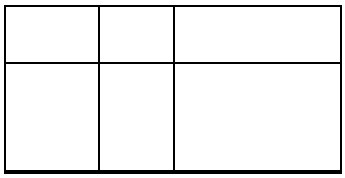


- The circuit has two inputs S and R and two outputs Q and $\overline{Q}$, as shown. Note how the outputs are both fed back to act as inputs to the NAND gates.

- Recall, once again, the truth table of the NAND gate as follows:

A	B	f
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

NAND



NAND

- We can work out the **next output Q(τ)** of the above circuit for all combinations of the inputs S, R and **Q** (**the previous output**). This is similar to the truth table approach for combinational logic circuits, but we now refer to the table as a **state table**.

- Let's examine the circuit in relation to each combination of inputs in turn, as follows:

- Let **S = 0** and **R = 0**:

  - Both gates are disabled and the outputs are therefore 1 irrespective of the previous value of Q. Hence $Q(\tau) \rightarrow 1$ and $\overline{Q}(\tau) \rightarrow 1$.

  - The starting value of Q (either 0 or 1) will not affect this final outcome.

  - However we require that $Q \neq \overline{Q}$ and therefore this combination of inputs produces what is considered ***an indeterminate or undefined output***.

- Let **S = 0** and **R = 1**:

  - Gate g1 is disabled and the output is therefore 1 irrespective of the previous value of Q. Hence $Q(\tau) \rightarrow 1$.

- For gate g2, R = 1 and therefore its output will be the opposite of the other input, i.e. Q(τ) from gate g1. Hence $\overline{Q}(\tau) \rightarrow 0$.
- The starting value of Q (either 0 or 1) will not affect this final outcome.

- Let **S = 1** and **R = 0**:
  - Gate g2 is now disabled and the output is therefore 1 irrespective of the previous value of Q. Hence $\overline{Q}(\tau) \rightarrow 1$.
  - For gate g1, S = 1 and therefore its output will be the opposite of the other input, i.e. $\overline{Q}(\tau)$ from gate g2. Hence $Q(\tau) \rightarrow 0$.
  - The starting value of Q (either 0 or 1) will not affect this final outcome.

- Finally, let **S = 1** and **R = 1**:
  - For gate g1, S = 1 and therefore its output will be the opposite of the other input, i.e. $\overline{Q}(\tau)$ from gate g2.
  - For gate g2, R = 1 and therefore its output will be the opposite of the other input, i.e. Q(τ) from gate g1.
  - Hence, in this case, the starting value of Q (either 0 or 1) will affect the final outcome.
  - In the case where Q is 0, this implies that $\overline{Q}(\tau) \rightarrow 1$ and $Q(\tau) \rightarrow 0$.
  - In the case where Q is 1, this implies that $\overline{Q}(\tau) \rightarrow 0$ and $Q(\tau) \rightarrow 1$.

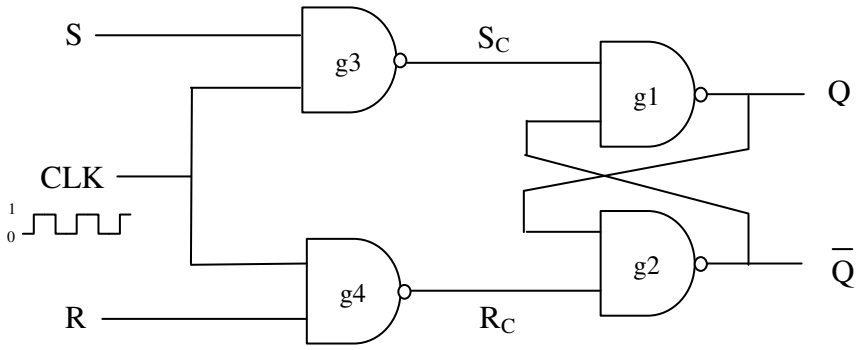
- Thus, by taking each combination of the inputs, and working through the circuit, as we have done, we can derive the following **state table**:

S	R	Q	Q(τ)	
0	0	0		*Undefined*
0	0	1		
0	1	0		*Q is Set to 1*
0	1	1		
1	0	0		*Q is Reset to 0*
1	0	1		
1	1	0		*Q*
1	1	1		

6.3 Clocked version of the SR Flipflop

- As it stands, the previous design will react to changes in the input **at any instant** of time, i.e. it is asynchronous.

- A **synchronous** version of the flipflop (i.e. one controlled by a clock) can be achieved through the addition of two NAND gates as follows:



- Now, let us examine the operation of this circuit, as follows:

- If **S = 0** and **R = 0**:
  - then $S_C = R_C = 1$ **irrespective of the CLK input**
  - both g1 and g2 are enabled $\Rightarrow Q(\tau) = Q$ and $\overline{Q}(\tau) = \overline{Q}$

- If **S = 0** and **R = 1**:
  - then $S_C = R_C = 1$ **until CLK is set high** (i.e. CLK $\rightarrow$ 1)
  - then $R_C \rightarrow 0$, ($S_C = 1$) and $\overline{Q}(\tau) \rightarrow 1$ and $Q(\tau) \rightarrow 0$

- If **S = 1** and **R = 0**:
  - then $S_C = R_C = 1$ **until CLK $\rightarrow$ 1**
  - then $S_C \rightarrow 0$, ($R_C = 1$) and $Q(\tau) \rightarrow 1$ and $\overline{Q}(\tau) \rightarrow 0$

- If **S = 1** and **R = 1**:
  - then $S_C = R_C = 1$ **until CLK $\rightarrow$ 1**
  - then $S_C \rightarrow 0$ and $R_C \rightarrow 0$ and $Q(\tau) \rightarrow 1$ and $\overline{Q}(\tau) \rightarrow 1$ (**not permitted**)

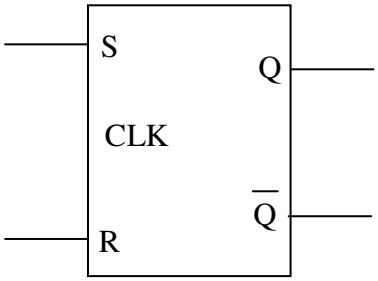
- Hence, we can derive the following **state table for the synchronous SR flipflop**:

S	R	Q	Q(τ)	
0	0	0		*Q*
0	0	1		
0	1	0		*Q is **R**eset to 0*
0	1	1		
1	0	0		*Q is **S**et to 1*
1	0	1		
1	1	0		*Undefined*
1	1	1		

- For convenience, we can also obtain a **concise form of the state table** as follows:

S	R	Q(τ)
0	0	
0	1	
1	0	
1	1	

- The flipflop is **S**ET by S = 1, R = 0 and is **R**ESET by S = 0, R = 1 and hence the name **SR** flipflop.
- Note that the output values are only set when the clock is enabled, i.e. CLK = 1.
- When the clock is disabled (i.e. CLK = 0), the S_C and R_C are set to 1, thereby maintaining the previous output values. Any change in S and R will therefore not affect the output.
- The circuit is now synchronous.
- The standard symbol for the flipflop is:



Requirements table …

- An alternative means for expressing the behaviour of the flipflop is in terms of a **requirements table**. This particular table is useful when we consider the design of counters later in this module.

- The requirements table basically defines the possible transitions between states. As we have only 2 possible states (or outputs), we thus have 4 possible transitions as follows:

    - GO from 0 to 1
    - GO from 1 to 0
    - STAY at 0
    - STAY at 1

- By carefully analysing the previous state table, we can derive the following requirements table for the SR flipflop, where X represents don't care terms:

Operation	**S**	**R**
Stay at 0		
Stay at 1		
Go to 0		
Go to 1		

Next State Equation …

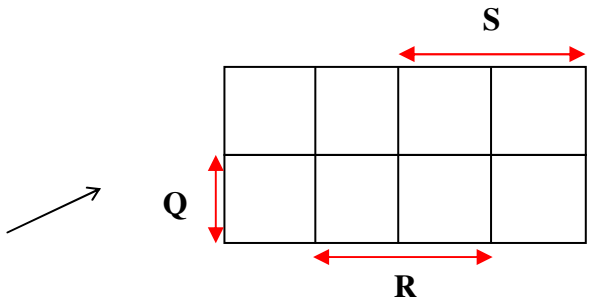
- We can also express the flipflop behaviour in terms of a **Next State Equation**. This is simply a Boolean expression that relates the next output Q(τ) to the inputs S and R and the previous output Q.

- The next state equation can be derived in two ways:

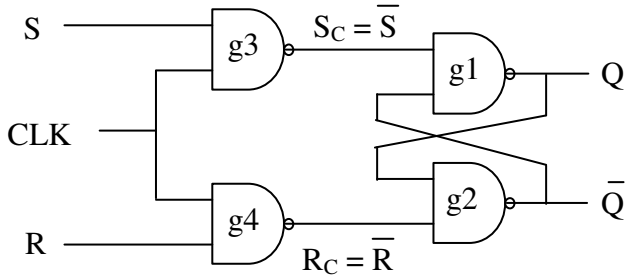
1) Karnaugh Map

S	**R**	**Q**	**Q(τ)**
0	0	0	**0**
0	0	1	**1**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**-**
1	1	1	**-**

Can't happen – assign as don't care terms



2) Logic Circuit

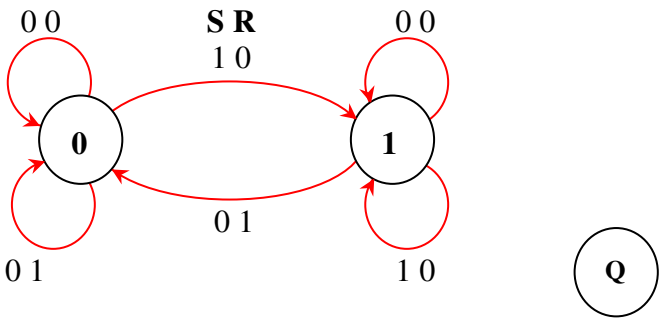


- When CLK is enabled, $S_C = \overline{S}$ and $R_C = \overline{R}$. Hence:

	$Q(\tau) = \overline{\overline{S}\overline{Q}}$	*… from gate g1*
But:	$\overline{Q} = \overline{Q\overline{R}}$	*… from gate g2*
Hence:	$Q(\tau) = \overline{\overline{S}\overline{Q\overline{R}}}$	
DeMorgan's:	$Q(\tau) = S + Q\overline{R}$	*(as before)*

State Diagram …

- Finally, we can also express the operation of flipflop in a graphical fashion known as a **state diagram**.
- The state diagram depicts the possible states of the flipflop along with the conditions for how it can transition between these states.
- The state diagram for the synchronous SR flipflop is:

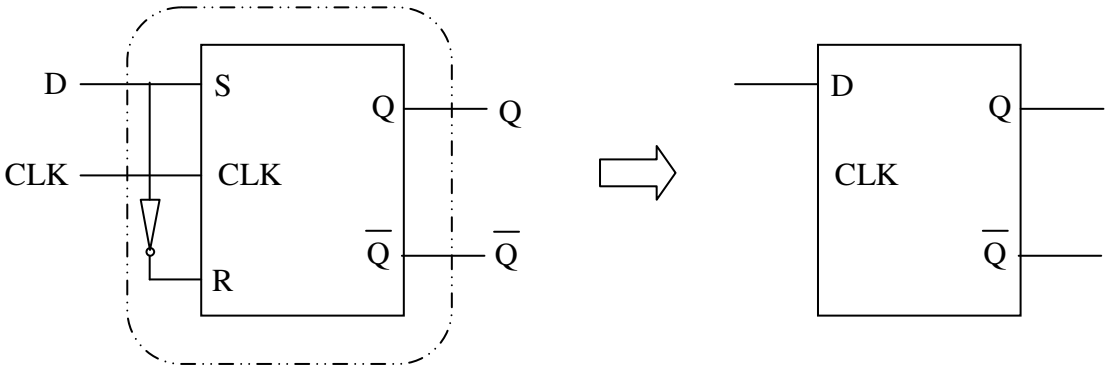


- In this diagram Q is the **state variable** and S and R are referred to as the **control variables**.

6.4 The D (Data) Flipflop

- The D or data flipflop is the simplest of all the flipflops. It basically allows the transfer of data in a synchronised manner.

- The D flipflop is a modified version of the SR flipflop where the inputs S and R are tied together with an inverter between them, as shown below:



State Table …

- The state table is easily derived from that of the SR flipflop as follows:

D	S	R	Q(τ)

Q is set to 0

Q is set to 1

D	Q(τ)

- Note that the condition S = R cannot occur due to the presence of the inverter and, hence, the indeterminate state of the SR flipflop is not an issue here.

- In brief, for the D flipflop, *the next output is simply the same as the input* (once the CLK is high).

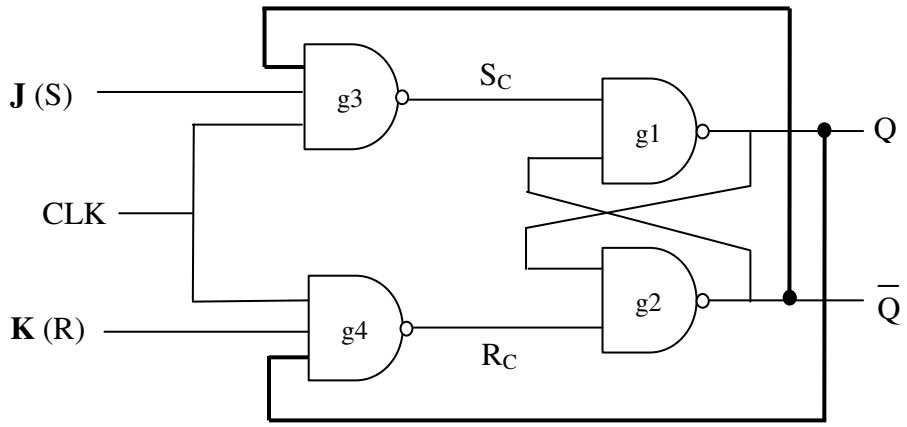
Requirements Table …

Operation	D
Stay at 0	
Stay at 1	
Go to 0	
Go to 1	

6.5 The JK Flipflop

- The JK flipflop is a refined version of the SR flipflop, where the indeterminate state of the SR flipflop is now defined.

- Here, **S** now becomes **J** and **R** becomes **K**, as follows:



- If **J = 0** and **K = 0**:
  - Then $S_C = R_C = 1$, both g1 and g2 are enabled $\Rightarrow Q(\tau) = Q$ and $\overline{Q}(\tau) = \overline{Q}$ (same as the SR flipflop)

- If **J = 0** and **K = 1**:
  - Then when CLK $\rightarrow$ 1, $S_C = 1$, $R_C \rightarrow \overline{Q}$
  - Since $R_C = \overline{Q}$, $\overline{Q}(\tau) \rightarrow \overline{Q\,\overline{Q}} = 1$ and since $S_C = 1$, $Q(\tau) \rightarrow 0$
  - Hence $\overline{Q}(\tau) \rightarrow 1$ and $Q(\tau) \rightarrow 0$ (same as the SR flipflop)

- If **J = 1** and **K = 0**:
  - Then when CLK $\rightarrow$ 1, $R_C = 1$, $S_C \rightarrow \overline{\overline{Q}} = Q$
  - Since $S_C = Q$, $Q(\tau) \rightarrow \overline{Q\,\overline{Q}} = 1$ and since $R_C = 1$, $\overline{Q}(\tau) \rightarrow 0$
  - Hence $Q(\tau) \rightarrow 1$ and $\overline{Q}(\tau) \rightarrow 0$ (same as the SR flipflop)

- If **J = 1** and **K = 1**:
  - Then when CLK $\rightarrow$ 1, $S_C \rightarrow Q$ and $R_C \rightarrow \overline{Q}$
  - Hence $Q(\tau) \rightarrow \overline{Q\,\overline{Q(\tau)}}$ and $\overline{Q(\tau)} \rightarrow \overline{\overline{Q}\,Q(\tau)} = \overline{\overline{Q}\,\overline{(Q\,\overline{Q(\tau)})}} = Q + Q\overline{Q(\tau)} = Q$
  - And hence $Q(\tau) \rightarrow \overline{Q\,Q} = \overline{Q}$ and thus $Q(\tau) \rightarrow \overline{Q}$ and $\overline{Q}(\tau) \rightarrow Q$

- Overall, we can state that the operation of the JK flipflop is the same as that of the SR flipflop except for J = K = 1.

- In the case of the SR flipflop this resulted in an indeterminate state. In the case of the JK flipflop, this condition acts as a toggle flipflop. In other words, it complements the output.

State Table …

- Hence, we can derive the following **state table for the synchronous JK flipflop**. The concise form of the state table is also shown.

J	K	Q	Q(τ)	
0	0	0		Q
0	0	1		
0	1	0		0
0	1	1		
1	0	0		1
1	0	1		
1	1	0		$\overline{Q}$
1	1	1		

⇨

J	K	Q(τ)
0	0	
0	1	
1	0	
1	1	

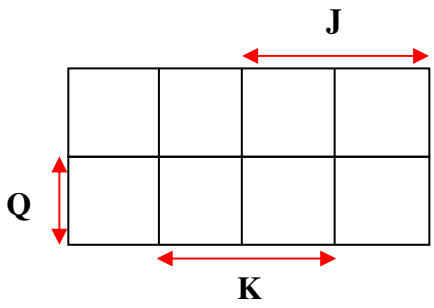
Concise state table

Requirements Table …

Operation	J	K
Stay at 0		
Stay at 1		
Go to 0		
Go to 1		

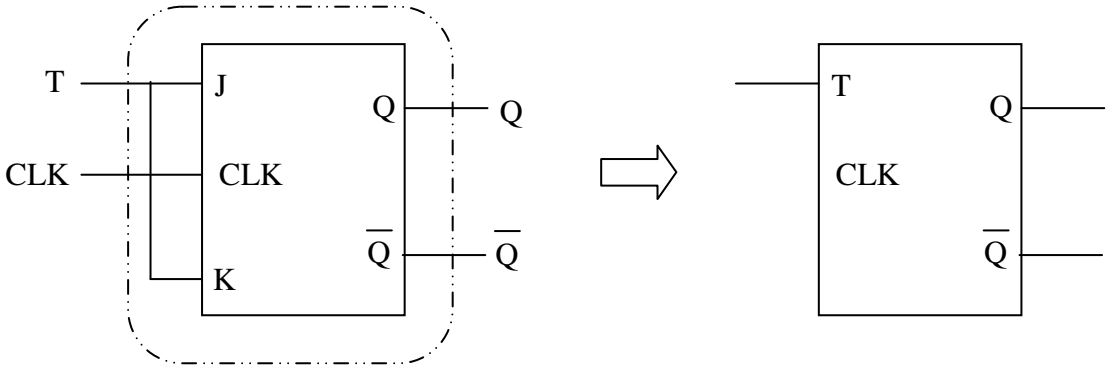
Next state equation …

- The next state equation for the JK flipflop can be derived as follows:



6.6 The T (Toggle) Flipflop

- The toggle flipflop is a modified version of the JK flipflop, where both inputs are simply tied together as follows:



- Note that by connecting J and K to T, we are restricting the J and K inputs to be either both 0 or 1. The condition J ≠ K can never occur.

State Table …

- Hence the following state table:

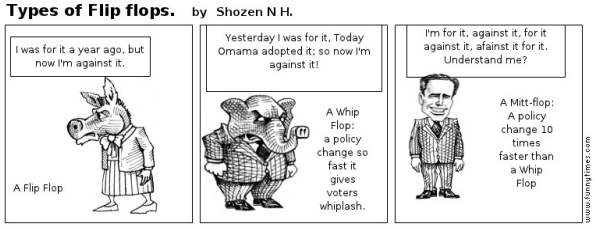
T	J	K	Q	Q(τ)
0	0	0	0	
0	0	0	1	
1	1	1	0	
1	1	1	1	

T	Q(τ)
0	
1	

- In brief, the T flipflop *complements the output when T = 1, otherwise the output remains the same when T = 0.*

Requirements Table …

Operation	T
Stay at 0	
Stay at 1	
Go to 0	
Go to 1	



6.7 Flipflops v Latches - a brief note

- Latches are very similar devices to flipflops – they are constructed in the same fashion and have the same basic operating principles. However there are two notable differences.

- Firstly, and a relatively minor different, latches tend to use an Enable (or Control) input instead of the Clock input associated with the flipflops.

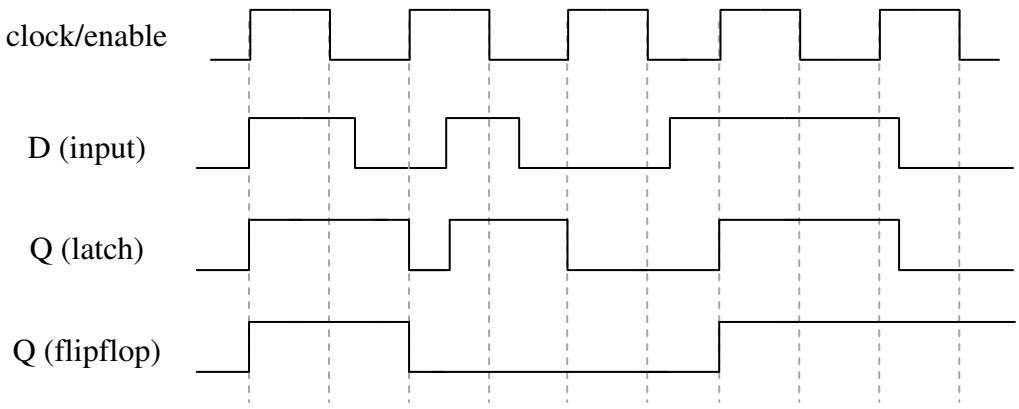
- Both work using the same principle, whereby the output of the device can only change depending on the Enable input or the Clock input being set HIGH (typically). However, the Clock tends to be a regular periodic signal, while the Enable key does not have to be.

- The second and fundamental difference relates to when the actual output is allowed to change.

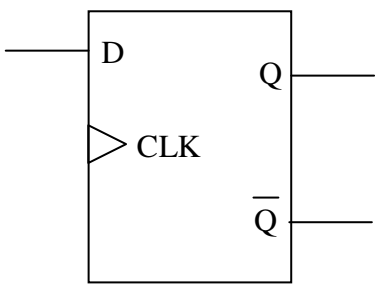
- In the case of the latch the output can change at any instant in time while the Enable input is set HIGH.

- In flipflops, the output can only change when the Clock is **in transition between two states**, either LOW to HIGH or HIGH to LOW.

- This difference has a significant impact on the timing of both devices. By way of example, consider the timing associated with a D flipflop, using a positive-edge clock transition (i.e. from 0 to 1), and a D latch:

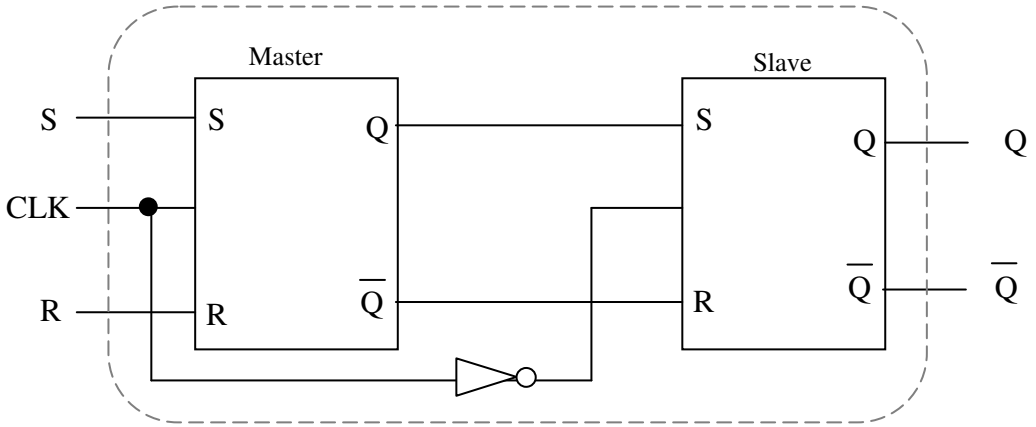


- The symbol for an edge-triggered flipflop (for example the D flipflop) looks like:



6.8 Master-Slave Flipflops

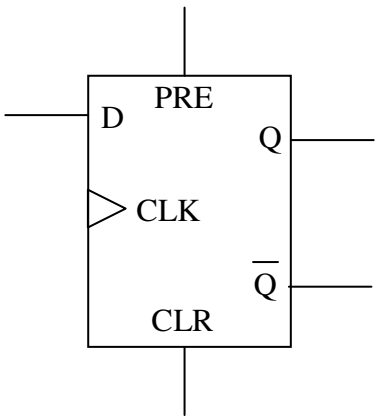
- As outlined in the previous section, a latch allows the output to change while the control or enable input remains high.
- Thus, it is possible for the latch output to change several times during this period.
- As already indicated, an edge-triggered flip-flop can resolve this issue.
- Alternatively, we could use a two-stage latch which ensures that the output does not change state until after the *trailing edge* of the clock pulse.
- This configuration is known as a master-slave flipflop.
- By way of illustration, consider the ***Master-Slave SR flipflop*** shown below:



- One circuit acts as the master and the other as the slave, hence the name!
- The operation of this device is relatively straightforward.
- When clock is HIGH, the inputs to the Master flipflop can affect its outputs. The Slave device is disabled and, as such, its outputs remain unchanged.
- When clock goes LOW, the Slave is now enabled and the outputs of the Master flipflop and, hence, the inputs to the Slave flipflop, now affect the outputs of the latter. The Master flipflop is now disabled so its outputs are unchanged during this LOW clock cycle.
- In this structure, the outputs of the overall master-slave device can only change once in one full clock period.
- *For further information in relation to the setup of various types of master-slave flipflops, please refer to a good textbook.*

6.9 Asynchronous Inputs

- Flipflops often have additional inputs that allow them to be set and reset **independently of the clock input**.
- Such inputs are known as **asynchronous inputs**.
- The inputs that asynchronously set the flipflop are called **preset**.
- The inputs that asynchronously reset the flipflop are called **clear**.
- When power is turned on to a digital system, the flipflop states are randomly set. The asynchronous inputs are used to correctly initialise the digital system before the normal clocked operation begins.
- The symbol for a D flipflop (for example) with asynchronous inputs looks like:



6.10 Application Examples

- There are various examples of applications that use flipflops.
- In the next sections of the notes, we will look in detail at two such applications, namely Counters and Registers (data storage).
- Two other common applications are now outlined.

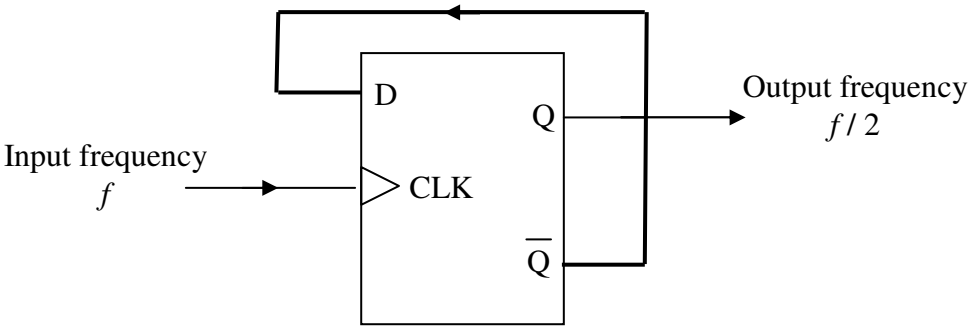
Contact bounce …

- Flipflops can also be used to eliminate the effect of contact bounce.
- When a switch is flicked, it physically vibrates and/or bounces for a short time before the contact is solid.

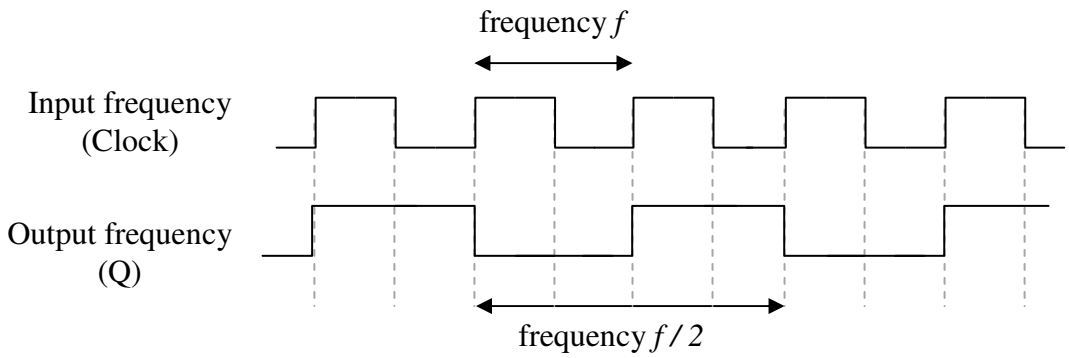
- This bouncing lasts a short time but can have undesirable effects in a digital system.
- By connecting the switch to a flipflop, the effect of the bounce is removed and a clean switching signal is achieved.

Frequency division …

- Flipflops can also be used to divide a clock frequency by 2^n for any integer n.
- For example, the following diagram shows how a D-flipflop can be used as a *divide-by-2* counter:



- The input (clock) and output waveforms for this counter are as follows:



- Clearly, we can see how the input frequency signal has been divided by 2.
- If we have a sequence of such devices, we can then easily build a system to divide by 2^n.