ElectronicsTutorials

SEQUENTIAL LOGIC

For Students, Professionals and Beyond

eBook 22

ASPENCORE

WWW.ELECTRONICS-TUTORIALS.WS

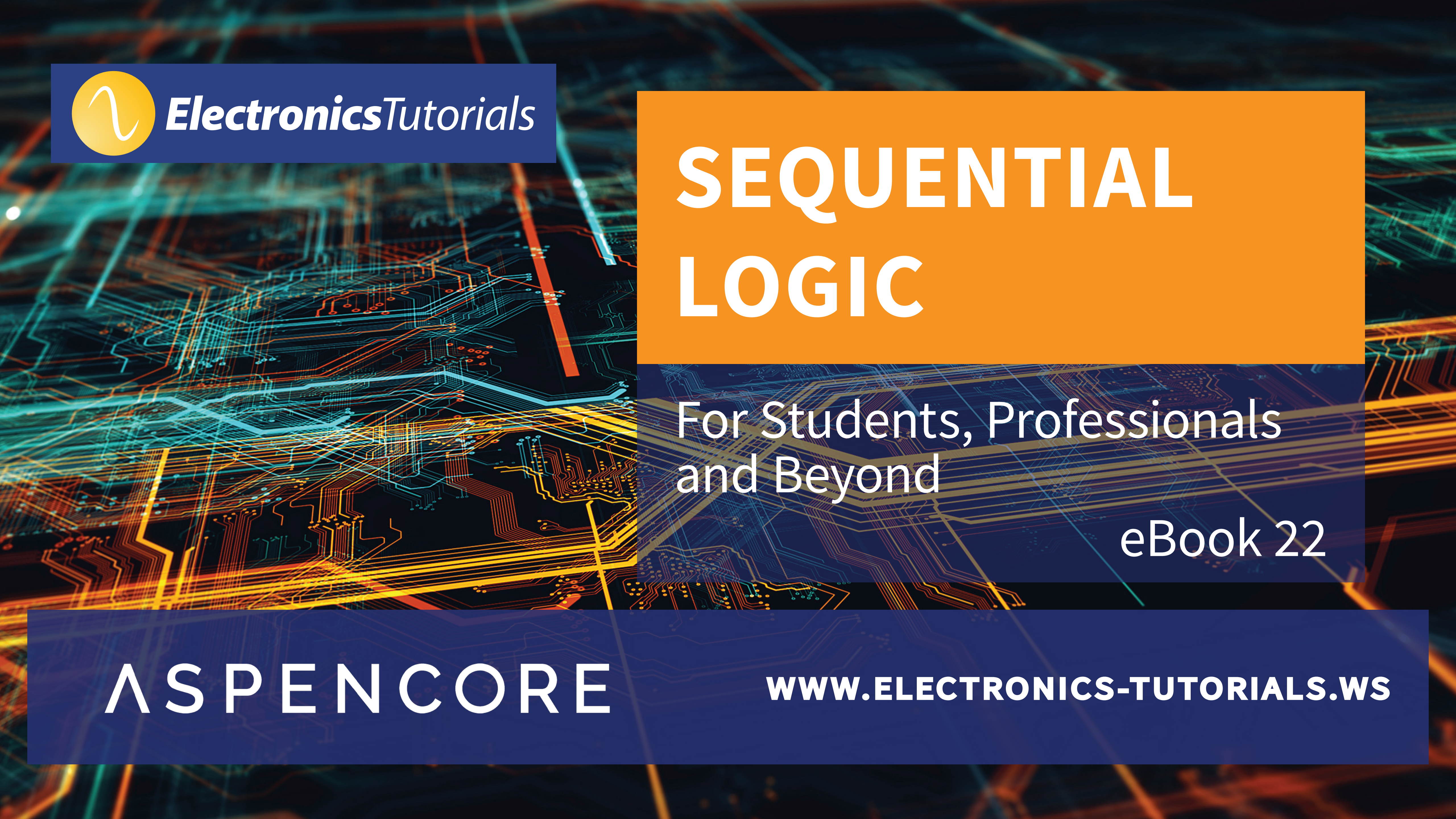


TABLE OF CONTENTS

1. Classification of Sequential Logic

Sequential logic is a type of digital circuit which has some form of positive feedback built into its design. Its actual output state depends on the present value of its input signals as well as on the switching sequence of its past inputs.

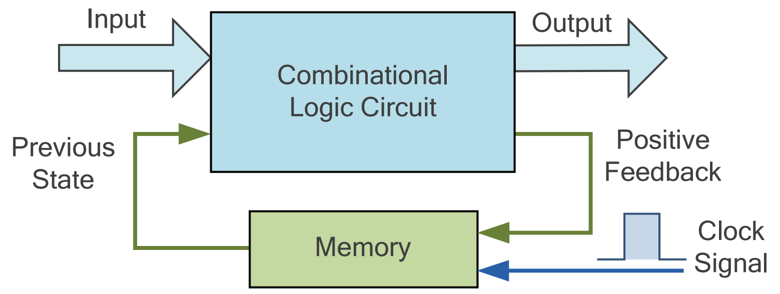
That is, **Sequential Logic Circuits** are able to take into account their previous input state as well as those actually present, producing a sort of "before" and "after" effect. Then the corresponding output state of a sequential logic circuit is a function of the following three conditions. The "present input", the "past input" and/or the "past output".

Sequential Logic circuits remember these conditions and will stay fixed in their current active state indefinitely until some external signal or influence changes one of these states. This gives sequential logic circuits a form of memory acting in much the same way as a bistable multivibrator circuit since it has two stable states.

The word "*Sequential*" means that things happen in a "sequence", one after another and in sequential logic circuits, an external controlling clock signal determines when things will happen next. Then we can define a sequential circuit by a time sequence of inputs, outputs, and internal states.

Thus, the addition of a memory device to a combinational circuit (a collection of logic gates) allows its output to be fed back to the input producing a sequential circuit as shown in Figure 1.

Figure 1. Sequential Logic Representation



Simple sequential logic circuits can be constructed from standard Bistable circuits such as: Flip-flops, Latches and Counters and which themselves can be made by simply connecting together universal NAND gates and/or NOR gates in a particular combinational way to produce the required sequential circuit.

Sequential circuits are digital logic circuits with in-built memory

Just as standard logic gates are the building blocks of combinational logic circuits, bistable latches and flip-flops are the basic building blocks of sequential logic circuits.

Sequential logic circuits can be constructed to produce either simple edge-triggered flip-flops or more complex sequential circuits such as storage registers, shift registers, memory devices or counters. Either way sequential logic circuits can be divided into the following three main categories depending on the timing of their signals:

1. **Event Driven –** Asynchronous Sequential circuits which switch or change state immediately when enabled
2. **Clock Driven –** Synchronous Sequential circuits that are synchronised to a specific external clock signal
3. **Pulse Driven –** These are a combination of the previous two that respond to triggering pulses

Then sequential logic is an essential part of digital circuit design and is used to implement a wide variety of digital functions, from simple counters to complex state machines.

2. The Set - Reset Latch

Latches are binary storage devices composed of two or more standard logic gates but with some form of feedback built-in. Latches are the most basic digital circuits from which all sequential circuits are constructed. They operate with signal voltage levels making them *Asynchronous Sequential Circuits* since their output changes state quickly after the input changes.

Note that Asynchronous simply means not occurring at the same time so their outputs depend upon the order in which the input variables change and at any instant of time.

The most basic single-bit latch by far is the SR (Set-Reset) Latch. It can be created using two cross-coupled **NOR** (TTL **74LS02**) or **NAND** (TTL **74LS00**) gates in a back-to-back configuration as shown in Figure 2.

Figure 2. Basic SR Latch Implementation

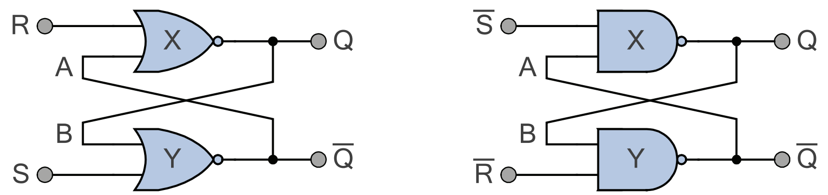


Fig. 2a NOR gates   Fig. 2b NAND gates

In the **NOR** implementation of Figure 2a, both the **Set** (**S**) and **Reset** (**R**) are active-high level-triggered inputs, while the **NAND** implementation is active-low (over-bar indicates this) meaning that they change state depending on the level (low or high) of the input signal to a logic gate.

Each SR latch has two outputs, **Q** and $\overline{Q}$, (not-Q). The **Q** and $\overline{Q}$ outputs are known as the **True** (Boolean) and **Complementary**, respectively. Thus, the two outputs are always opposite or complementary to each other with either or both of these outputs available for use within a circuit. Also, each latch has two stable states, which we will label logic-0 (LOW) and logic-1 (HIGH). So how do the work?

We remind ourselves here that a **NOR** gate produces a logic-1 (HIGH) output only when BOTH of its inputs are at logic-0 (LOW), otherwise the output is 1 (HIGH). So for the **NOR** gate implementation of Figure 2a, let's us initially assume that the SR latch is in the Set state by making **S = 1** and **R = 0** giving output **Q = 1**, and $\overline{Q}$ **= 0**.

If we make **R = 0** and **S = 0**. The inputs to the "X" **NOR** gate are **R = 0** and **A = 0**, so **Q = 1**, and inputs to the "Y" **NOR** gate are **S = 0** and **B = 1**, so $\overline{Q}$ **= 0**. Therefore, the SR NOR latch remains in its current Latched or Set state. That is, no state change.

If we now make **R = 1** and keep **S = 0**, inputs to the "X" **NOR** gate become **R = 1** and **A = 1**, so **Q = 0**, and inputs to the "Y" **NOR** gate become **S = 0** and **B = 0**, so $\overline{Q}$ **= 1**. Therefore, the SR NOR latch changes state to the Reset state, (**Q = 0,** $\overline{Q}$ **= 1**) and remains there.

Returning again to **S = 1** and **R = 0**, the inputs to the "X" **NOR** gate are **R = 0** and **A = 0**, so **Q = 1**, and inputs to the "Y" **NOR** gate become **S = 1** and **B = 1**, so $\overline{Q}$ **= 0**. Therefore, the SR NOR latch changes state back to its Set state, (**Q = 1,** $\overline{Q}$ **= 0**) and remains there.

Then we can see that if **S = 1** and **R = 0** the latch is in its set state. If **S= 0** and **R = 1** the latch is in its reset state. But if **S = 0** and **R = 0** nothing happens. That is, no state change. Note that once the SR latch is in either its set or reset state, any subsequent activity on the corresponding input will have no effect on the output state, so is permanently latched.

You may be thinking, but what if we make both the set and reset inputs HIGH at the same time, **S = R = 1**. Since the basic SR latch is constructed using two cross-coupled logic gates, an input of **S = R = 1** would create a conflict between the two halves resulting in metastable oscillations. These oscillations would eventually stop in either its set or reset condition thus producing an uncertain outcome. Then the condition of **S = R = 1** is invalid.

We can summarise the operation of the SR latch in the following truth table of Table 1.

Table 1. SR NOR Latch Truth Table

Inputs		Outputs		Condition
Reset (R)	Set (S)	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	No State Change
0	1	1	0	Set Q = 1
1	0	0	1	Reset Q = 0
1	1	x	x	Invalid

Since **S = R = 1** is an invalid input condition with an unknown outcome. It is therefore indicated as being in "x" (don't know) state in the truth table.

In the **NAND** implementation of Figure 2b, both the **Set** (S) and **Reset** (R) are active-low inputs (NAND gate principles). This results in the switching conditions of latched or set when **S = 1** and **R = 0**, and reset when **S = 0** and **R = 1**. The difference this time, is that the input condition of **S = 1** and **R = 1** results in no state change, while **S = 0** and **R = 0** is invalid as shown in Table 2.

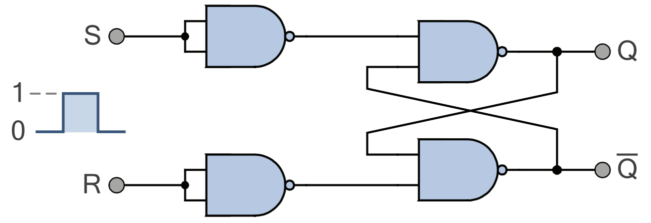
TABLE 2. SR NAND LATCH TRUTH TABLE

Inputs		Outputs		Condition
Reset (R)	Set (S)	Q	$\overline{Q}$	
0	0	x	x	Invalid State
0	1	1	0	Set Q = 1
1	0	0	1	Reset Q = 0
1	1	Q	$\overline{Q}$	No State Change

If any input to a digital logic **NAND** gate is at logic-0, the output will be forced to logic-1. Thus it is only if both inputs to a **NAND** are at logic-1 will the output will be logic-0.

But we can convert this basic active-low **SR NAND** circuit to one that changes state by the application of positive going input signals with the addition of two extra **NAND** gates connected as inverters to the **S** and **R** inputs as shown in Figure 3.

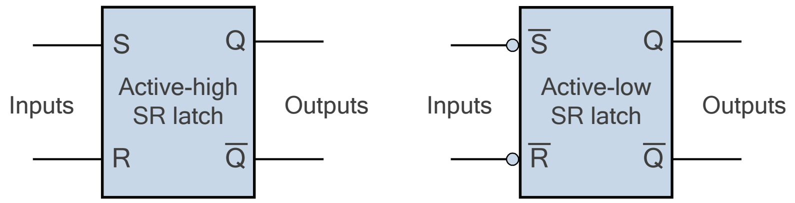
FIGURE 3. POSITIVE INPUT SR NAND LATCH



Here, the operation of the circuit of Figure 3 is exactly the same as for the **NOR** gate latch. That is, **S** is used to "Set" the latch (set it to one). **R** is used to "Reset" or "Clear" the latch (set it to zero).

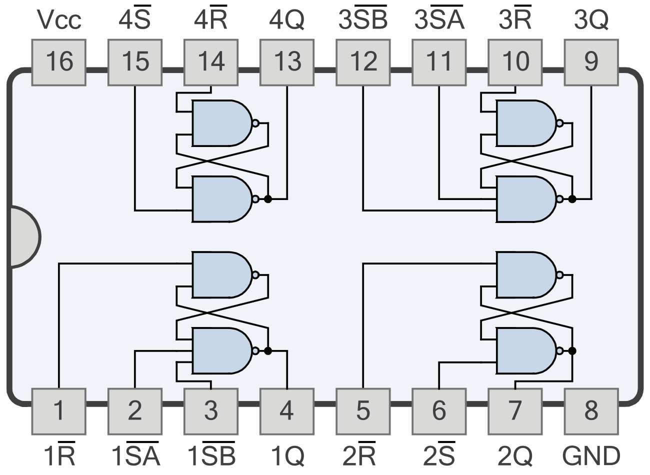
We can define the basic SR Latch as being a voltage-triggered regenerative switch which relies on positive feedback to switch between one of two stable states making it a single-bit storage element. The common symbol used to define an SR Latch is given in Figure 4.

FIGURE 4. SR LATCH GRAPHIC SYMBOL



The small circle on the inputs of the active-low symbol of Figure 4. indicates the inverting action. Set-Reset bistable latches are available in standard TTL packages such as the **74LS279** Quad SR Bistable Latch IC a shown in Figure 5.

FIGURE 5. THE QUAD SR LATCH 74LS279



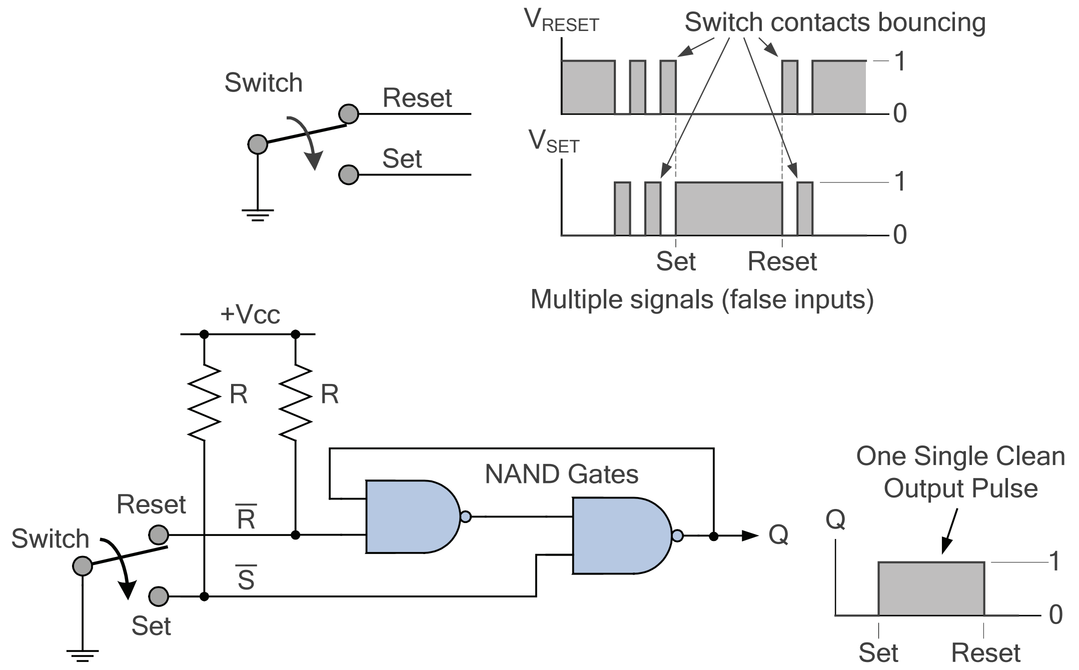
The 74LS279 SR Latch IC contains four individual **NAND** type bistables within a single package allowing different sequential logic circuits to be easily constructed.

3. Switch Debounce Circuit

One simple and useful application of either a NOR or NAND gated SR latch is in the elimination of mechanical switch "bounce". As its name implies, switch bounce occurs when the contacts of any mechanically operated switch, push-button or keypad are operated and the internal switch contacts do not fully open or close cleanly, but instead bounce together when the switch is operated.

This can give rise to a series of individual switching pulses which can be as long as tens of milliseconds that an electronic system or circuit such as a digital counter may see as a series of logic pulses instead of one long single pulse registering multiple input counts instead of one single count. Then set-reset SR latch circuit can be used to eliminate this kind of problem as shown in Figure 6.

Figure 6. SR Latch Debounce Circuit



The SR debounce circuit of Figure 6. shows the center or common contact of the switch is connected to ground providing the signal equivalent of a logic-0 since we remember that the NAND gate SR latch is an active-low circuit. When one of the two contacts, set or reset, is not connected to ground through the switch, it behaves like a logic-1 voltage level connected to V_{cc} via resistor R.

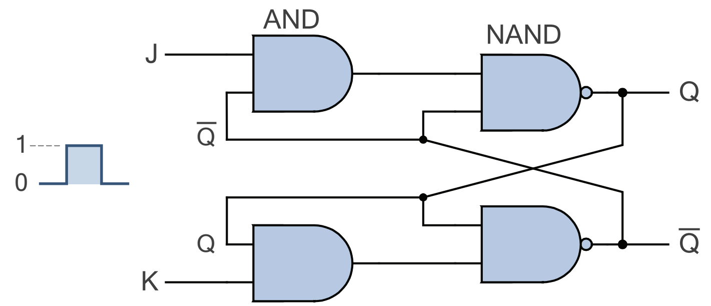
When the switch is operated from set to reset and back, the output of the SR NAND latch produces a single output pulse at Q as shown. The output at Q will remain permanently in its HIGH or LOW state unaffected by any contact bounce of the switch until the switch is returned back to the other position. The output will again exhibit a smooth transition even if there is contact bounce. The we can use a NAND latch to eliminate switch bounce.

4. The JK Latch

Whilst the basic SR NAND latch circuit has many advantages and uses in sequential logic circuits, it suffers from one basic switching problem. The **Set = 0** and **Reset = 0** condition (S = R = 0) must always be avoided.

However, if we add two AND gates to the inputs of the SR NAND latch circuit and cross-couple the outputs back the inputs of the NAND latch, we can create another type of bistable latching circuit called a JK Latch as shown in Figure 7.

Figure 7. Basic JK Latch



The JK latch of Figure 7. improves on the previous SR latch in that the invalid state of the SR latch is defined. The input **J** and **K** behave just like the input **S** and **R** to set and reset the latch respectively. The difference is that the output **Q** is AND'ed with the **K** input, while output $\overline{Q}$ is AND'ed with **J** input.

When both inputs are activated HIGH at the same time (**J = K = 1**), the JK latch will switch or toggle its output state, regardless of the previous state. That is, the JK latch switches to its complementary output state, if output **Q = 1**, it switches to **Q = 0** and vice versa as shown in Table 3.

TABLE 3. JK LATCH TRUTH TABLE

Inputs		Output	Condition
K (reset)	J (set)	Q	
0	0	Qn	No State Change
0	1	1	Set Q = 1
1	0	0	Reset Q = 0
1	1	Qn + 1	Complement

We can also convert the basic SR **NOR** gate latch into a JK latch by adding two **OR** gates to its input in a similar way to the **AND-NAND** design. Then, JK latches either **AND-NAND**, or **OR-NOR** types are popular since they can improve on the basic SR latch design, but not vice versa, and because both inputs of the JK latch can be high at the same time.

5. THE CLOCKED SR FLIP-FLOP

As we have seen previously, **Latches** are bistable circuits based around simple logic gate networks which switch or change state on the application of a set or a reset input signal. This gives bistable latches the ability to retain digital information indefinitely acting as a simple memory element. Then by default, bistable latches are asynchronous voltage level driven switching circuits.

Thus, bistable latches are free running circuits which do not require any additional timing specific requirements to produce a valid output. However, sometimes we want to control the latches output state on the application of some particular timing signal regardless of the change of state of its set or reset inputs. This is where *flip-flops* come in.

Flip-flops are *synchronous sequential circuits* which switch or change state at discrete instants of time. Thus flip-flops have a clock signal as one of their inputs and will change state or stay the same, depending on the levels at the data inputs when the clock signal (commonly a pulse) arrives.

Clearly then, the simplest clocked flip-flop is just our original SR latch with a pair of added gates controlled by the clock to enable the SET and RESET inputs as shown in Figure 8.

FIGURE 8. BASIC CLOCKED SR NAND FLIP-FLOP

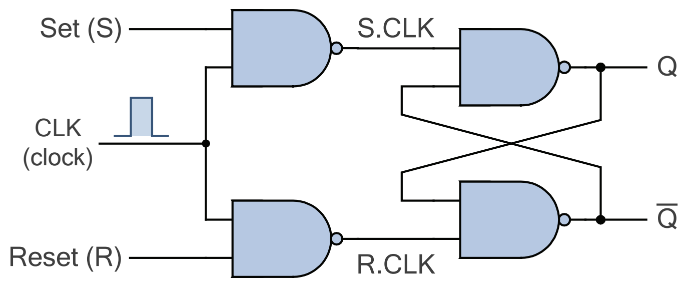
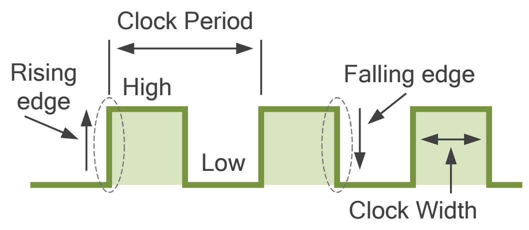


Figure 8. shows that the clocked SR flip-flop has a clock input labelled as **CLK**. Commonly for clocked flip-flops, the **CLK** input is edge-triggered. Meaning that the **CLK** input is activated by a signal transition.

FIGURE 9. CLOCK SIGNAL



That is, the SR flip-flop is triggered by either positive (rising) edge or negative (falling) edge of clock signal as shown in Figure 9. The output of the SR flip-flop can only be changed when the clock pulse is applied. If the clock signal changes from low to high state and the output changes due to the inputs, it is called positive edge triggering. When the clock signal changes from high to low state and the output changes due to the inputs, this condition is called negative edge triggering.

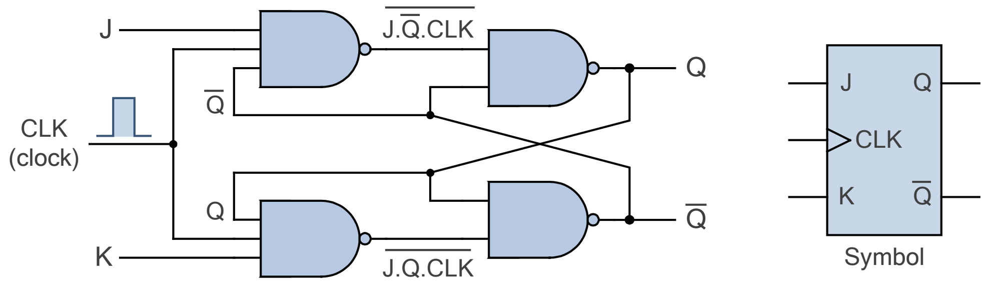
We can convert any configured asynchronous latch into a synchronous flip-flop circuit simply by adding an additional clock input. The synchronization is accomplished through the use of clock signal on the rising or falling edge.

Asynchronous latches are voltage level driven. Synchronous flip-flops are edge triggered

6. The Clocked JK Flip-flop

The clocked **JK flip-flop** has three inputs, the two inputs **J** and **K** plus a clock, **CLK** input as shown in Figure 10. The clock input on the symbol is indicated by a small triangle.

Figure 10. The Clocked JK Flip-flop



The switching operation of the clocked JK flip-flop is the same as that for the JK latch of Figure 7. Note that it is not necessary to use the **AND** gates of Figure 7, since the same switching function is performed by adding an extra input terminal to each of the two input **NAND** gates.

The inputs, **J** and **K**, are only 'enabled' on the rising (changing from 0 to 1) or falling (1 to 0) edge of the clock input. If when the first rising edge of the clock signal arrives and both **J** = **1** and **K** = **1**, output **Q** "toggles" or changes state. On the next rising edge of the clock signal, **J** = **1** and **K** = **0**, so **Q** becomes 1. The reverse is true for the next rising edge since both **J** and **K** are 0 and the value of **Q** remains the same. So the JK flip-flop operates again as a toggle switch. Thus there is no metastable state, but this simple JK flip-flop circuit will only work properly with very short clock input pulses toggling once when **J** = **K** = **1**.

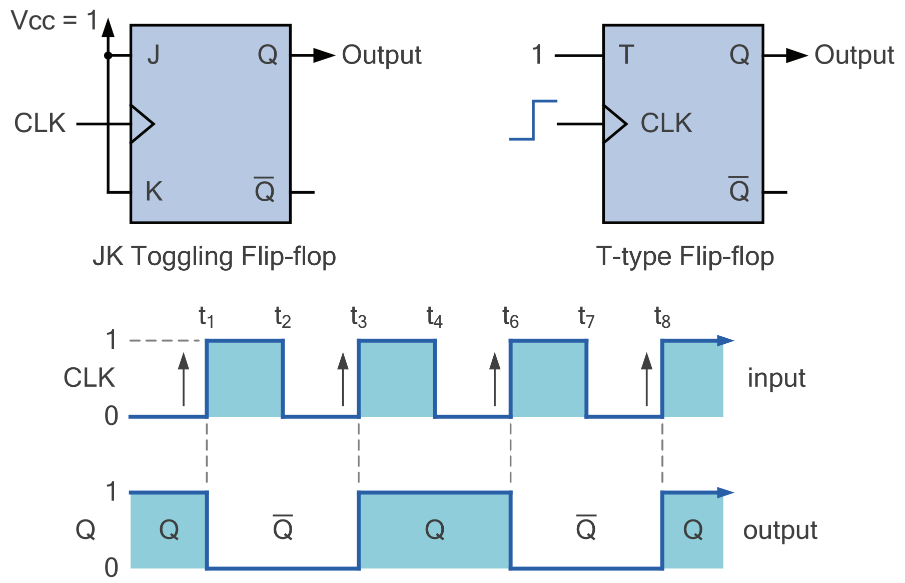
Then the only difference between the circuitry of edge-triggered clocked JK flip-flop and that of clocked SR flip flop is that in JK flip-flop circuit, **Q** and **$\overline{Q}$** outputs are fed back to the **NAND** gates. Because of this feedback connection the JK flip-flop produces a toggle operation for the **J** = **K** = **1** condition.

Then by connecting both the J and K inputs directly to VCC (HIGH) we can create another type of flip-flop circuit called a toggle flip-flop.

7. The Toggle Flip-flop

We can transform a JK flip-flop into a **Toggle flip-flop**, or *T-type flip-flop* whose outputs change state every time a pulse is applied to the clock input as shown in Figure 11.

Figure 11. The Toggle Flip-flop



We know that for the Toggle flip-flop, when T is HIGH it changes state (complemented). When T is LOW, the state of the flip-flop remains unchanged. Therefore, for a 0 to 0 and a 1 to 1 transition, T must be 0. But for a 0 to 1 and a 1 to 0 transition, T must be 1.

TABLE 4. TRUTH TABLE OF THE T FLIP-FLOP

CLK	T	Q	$\overline{Q}$	
0	x	Q	$\overline{Q}$	No change
↑1	0	Q	$\overline{Q}$	Hold
↑1	1	$\overline{Q}$	Q	Toggle

The characteristic equation for the T-type flip-flop can be given as:

$$Q^* = T\overline{Q} + \overline{T}Q$$

where $Q^* = Q$ represents the Hold condition, and $Q^* = \overline{Q}$ represents the Toggle condition.

It therefore becomes obvious from the operational principle of the T-type flip-flop that the switching or toggling speed is dependant on the frequency of the clock signal, and that the output at Q will be half the frequency of the clock signal. This makes for a very good divide-by-2 counter circuit.

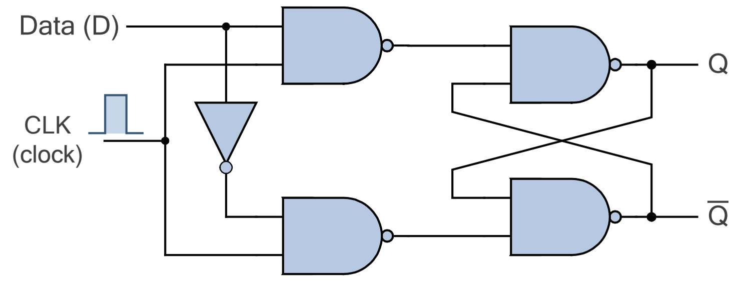
8. THE DATA LATCH

We remember from section 2, that the invalid input combination of S = 1 and R = 1 for the SR NOR gate latch (or S = R = 0 for the NAND version) should never occur. But if S and R are at different logic levels, one HIGH and the other LOW, or vice versa, the bistable SR latch will change state.

The *D-type Flip-flop*, also known as a *Delay flip flop*, *D latch*, *D-type Bistable, transparent latch*, or simply **Data Latch** as it is more generally called, is another type of edge-triggered device which can be used to provide temporary storage of one bit of information.

If we connect an inverter (NOT gate) from the S input to the R input of a controlled SR latch as shown in Figure 12, the SR circuit now becomes a controlled Data latch. Since there is now no need for a Reset (R) signal this input can be ignored with the Set (S) input becoming the new Data (D) input as shown.

FIGURE 12. THE DATA LATCH



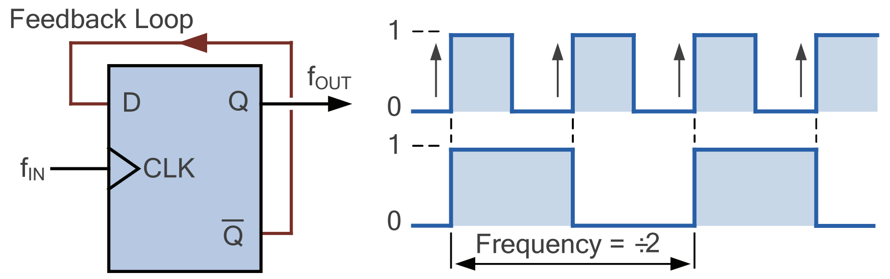
A Data latch has two inputs, D and CLK and two outputs, Q and $\overline{Q}$. The D input is same as the set input and the complement of D is applied to the old reset input. Thus, R and S can never be equal to logic-0 or logic-1 simultaneously due to the inverter.

The Data latch transfers the D input to the output Q only when the clock input is HIGH. If the clock input is LOW, one of the inputs to each of the two cross-coupled NAND gates will be HIGH (1). Thus, the outputs remain unchanged regardless of the value of the D input.

Note that this type of Data flip-flop is also referred to as a "*Transparent Latch*" since the output at Q will always follow the Data input whenever CLK = 1 (HIGH). That is, data transfer from D to Q is virtually transparent, as if the flip-flop was not there.

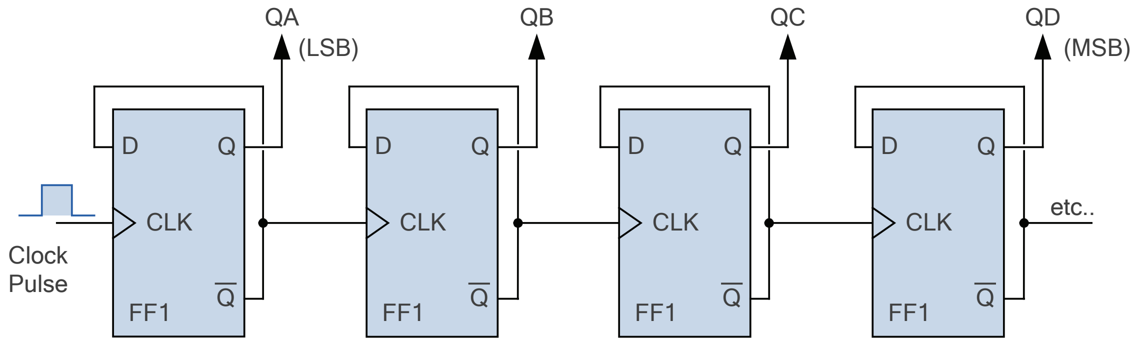
One main use of a Data latch is as a Binary Divider, or a Frequency Divider, to produce a "divide-by-2" circuit. If the $\overline{Q}$ output is connected directly back to the D input, it gives the circuit closed loop feedback so successive clock pulses will force the latch to toggle once every two clock cycles. Thus the output at Q will be half the frequency of the clock (CLK).

FIGURE 13. FREQUENCY DIVISION



We can take this divide-by-2 concept further if we connect together in series, two or more Data latches (or T-type flip-flops). The initial input frequency will be "divided-by-two" by the first flip-flop ($f \div 2$) and then "divided-by-two" again by the second flip-flop ($f \div 2$) $\div 2$, giving an output frequency which has effectively been divided four times, then its output frequency becomes one quarter value (25%) of the original clock frequency, ($f \div 4$).

FIGURE 14. 4-BIT FREQUENCY DIVIDER



Thus, each time we add another data latch to the chain, the output clock frequency is halved or divided-by-2 again and so on producing a 2-bit divider. This gives an output frequency of 2^n where "n" is the number of flip-flops used in the sequence.

D-type and JK flip-flops can act as binary divide-by-2 counter circuits

So for a two-stage circuit, there are $2^2 = 4$ output combinations with the final output frequency being equal to one-fourth of the input clock frequency, CLK/2^2. For a four-stage circuit, there are $2^4 = 16$ output combinations with the final output frequency being equal to one-sixteenth of the input clock frequency, CLK/2^4, etc.

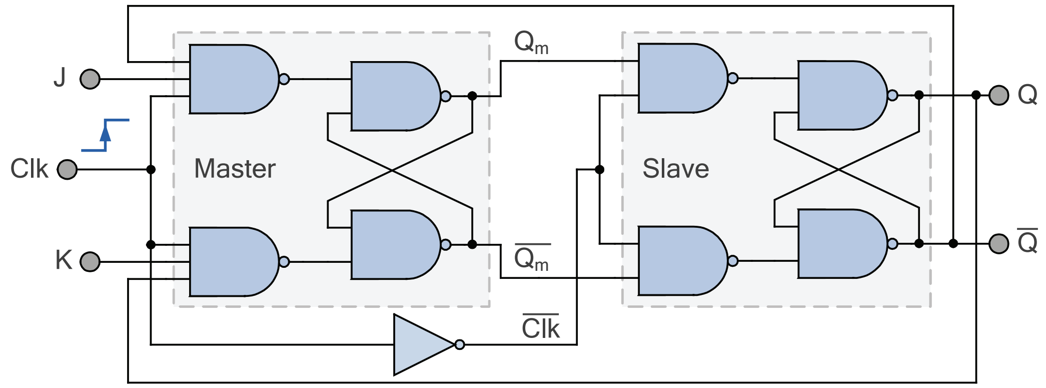
Then we have seen that Bistable SR latches and clock controlled JK flip-flops are basically voltage triggered switches which can be used a simple 1-bit storage devices. But the versatility and invalid switching states of these basic circuits can be greatly improved by connecting them together in a master-slave relationship.

9. MASTER-SLAVE GATED FLIP-FLOPS

A **master-slave gated flip-flop** consists of two basic clocked flip-flops connected together in a series formation. The first flip-flop is called the "*master*" and the second flip-flop which has an inverted clock signal is called the "*slave*".

The outputs from Q and $\overline{Q}$ from the slave flip-flop are fed back to the inputs of the master with the outputs of the master flip-flop being connected to the two inputs of the slave flip-flop as shown in Figure 15.

FIGURE 15. JK MASTER-SLAVE FLIP-FLOP



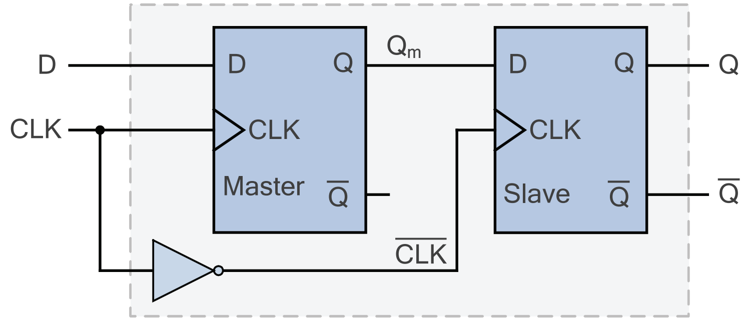
The basic JK master-slave flip-flop uses an inverter (NOT gate) to invert the positive going clock pulses to trigger the master flip-flop. Thus data is clocked into the master flip-flop at the rising edge of the CLK input. Any additional state changes in data at J or K has no affect on the state of the master flip-flop since the feedback from Q and $\overline{Q}$ will always disable which ever of the two input NAND gates is trying to make the change.

Due to the inverter in the clock line, on the falling edge of the external clock pulse the slave flip-flop will see a rising edge, accepting the masters data on its Q_m and $\overline{Q}_m$ outputs, changing the state of its own Q and $\overline{Q}$ outputs producing the characteristic toggling action. Thus, the master-slave flip-flop circuit accepts input data from J and K at the rising edge of CLK, and outputs it on Q and $\overline{Q}$ on the falling edge of CLK.

The master-slave structure can also be implemented using two edge-triggered D-type flip-flops. As before, the master and the slave flip-flop are both positive rising-edge triggered devices, with an inverted or complemented version of the clock (CLK) pulse fed to the input of the slave.

Therefore data is taken into D at the rising edge of the clock pulse, and appears at *Q* on the negative falling-edge of the clock pulse as shown in Figure 16.

FIGURE 16. THE D-TYPE MASTER-SLAVE FLIP-FLOP



10. SHIFT REGISTERS

A **Shift Register** is a sequential device which loads the data present on its inputs and then moves or "shifts" it to its output once every clock cycle. A shift register basically consists of several single-bit D-type data latches, one for each data bit, either a logic-0 or logic-1, connected together in a serial type daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on.

Data bits may be fed in or out of a shift register serially, that is one after the other from either the left or the right direction, or all together at the same time in a parallel configuration.

The number of individual data latches required to make up a single Shift Register device is usually determined by the number of bits to be stored with the most common being 8-bits (one byte) wide constructed from eight individual data latches.

Generally, shift registers operate in one of four different modes with the basic movement of data through a shift register being:

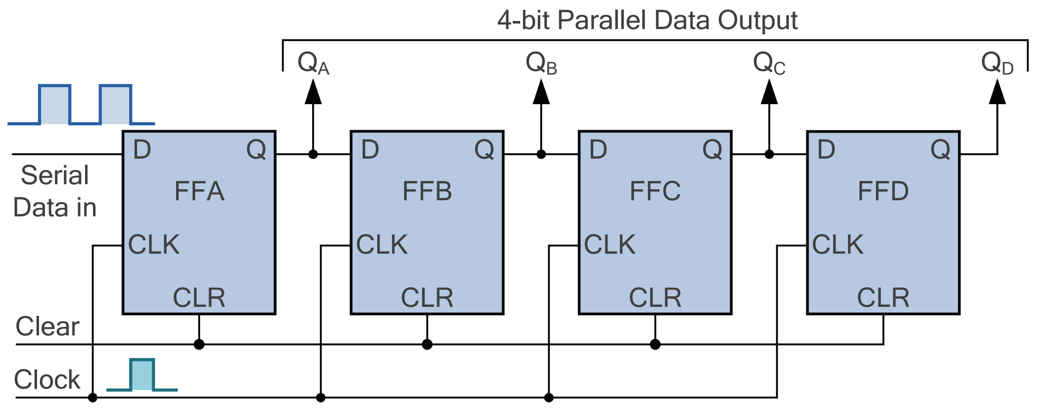
- Serial-in to Parallel-out (SIPO) – the register is loaded with serial data, one bit at a time, with the stored data being available at the output in parallel form.
- Serial-in to Serial-out (SISO) – the data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.
- Parallel-in to Serial-out (PISO) – the parallel data is loaded into the register simultaneously and shifted out serially one bit at a time under clock control.
- Parallel-in to Parallel-out (PIPO) – the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the clock pulse.

The directional movement of the data through a shift register can be to the left direction (left shifting), to the right direction (right shifting), or left-in but right-out, (rotation).

9.1 Serial-in to Parallel-out (SIPO)

A Serial-In Parallel-Out (SIPO) shift register loads data in serially (one bit at a time) and is read out in parallel (side by side). On each rising clock edge, a new serial input bit is clocked into the first flip-flop, and each one in succession loads its new value based on its predecessor's value. An n-bit **SIPO** register reflects the state of the last n-bits shifted in.

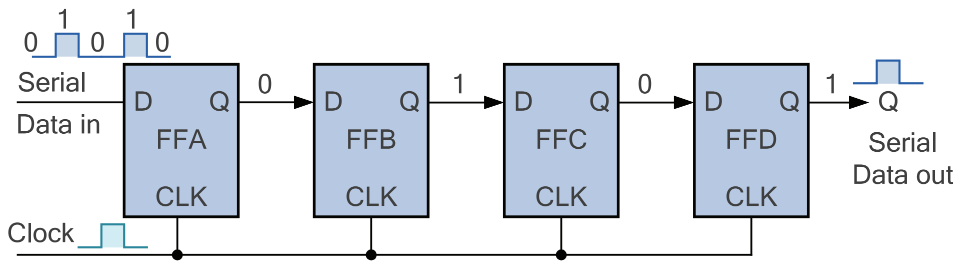
FIGURE 17. 4-BIT SERIAL-IN PARALLEL-OUT CONFIGURATION



9.2 Serial-in to Serial-out (SISO)

Figure 18. is a **Serial-In Serial-Out** (**SISO**) shift register which loads data in serially (one bit at a time) and is read out serially (one bit at a time). Thus the data is allowed to flow straight through the register and out of the other end.

FIGURE 18. 4-BIT SERIAL-IN SERIAL-OUT CONFIGURATION

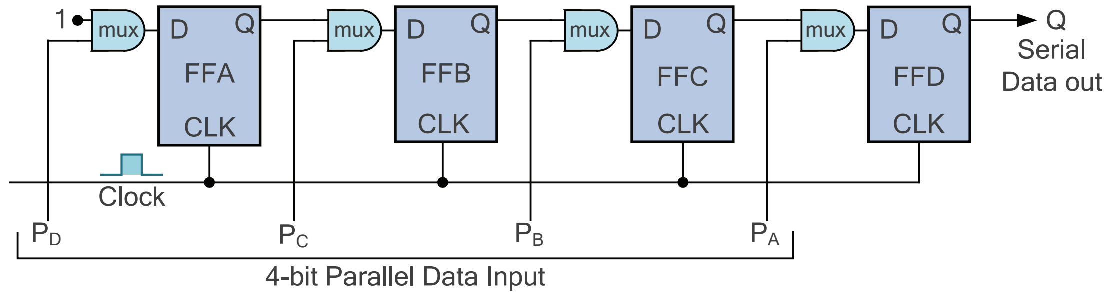


Clock (**CLK**) pulses are applied to each flip-flop in the series string. After each clock pulse, the data is moved through the register by one position from left to right. Thus the timing for the shift operations is provided by the input clock signal.

9.3 Parallel-in to Serial-out (PISO)

The **Parallel-In Serial-Out** (**PISO**) shift register loads its data in a parallel format in which all the data bits enter their inputs simultaneously. The data is read out sequentially in the normal shift-right mode to appear as a serial output as shown in Figure 19.

FIGURE 19. 4-BIT PARALLEL-IN SERIAL-OUT CONFIGURATION

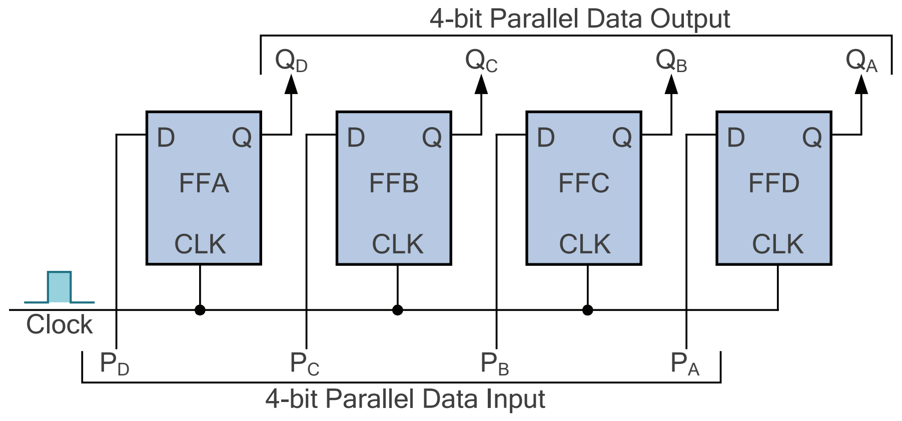


Data is outputted one bit at a time on each clock cycle in a serial format. Note that with the **PISO** register a clock pulse is not required to parallel load the register as it is already present on pins P_A to P_D, but four clock pulses are required to unload the data serially.

9.4 Parallel-in to Parallel-out (PIPO)

The **Parallel-In Parallel-Out** (**PIPO**) acts as a temporary storage device or as a time delay device similar to the previous **SISO** configuration. The data is presented in a parallel format to the parallel input pins P_A to P_D and then transferred together directly to their respective output pins Q_A to Q_D by the same clock pulse. Thus, one clock pulse loads and unloads the register as shown in Figure 20.

FIGURE 20. 4-BIT PARALLEL-IN PARALLEL-OUT CONFIGURATION



Then the **PIPO** shift register is the simplest of the four configurations as it has only three connections. Parallel in (**P**) which determines what enters the register. Parallel out (**Q**) and the sequencing clock signal (**CLK**).

There are some commercially available high speed bi-directional "universal" type shift registers available such as the TTL **74LS194**, or the **74LS195** which are available as 4-bit multi-function devices. They can be configured and used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, or as a parallel-to-parallel bidirectional configurations, hence their name "Universal".

11. Asynchronous Counters

Flip-flops can also be cascaded together to form counting circuits and are commonly classified according to the way they are clocked. These called **Asynchronous Counters** or **Synchronous Counters**.

In *Asynchronous counters*, (ripple counter) the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop.

Counting circuits can be either Asynchronous or Synchronous driven

In *Synchronous counters*, the clock input is connected to all of the flip-flops so that they are clocked simultaneously by the same clock pulse. Depending on the number of flip-flops (stages) counters of 2-bit, 4-bit, 8-bit, n-bit stages can be formed for counting up, down or both ways based on their sequential states.

10.1 Asynchronous Counters

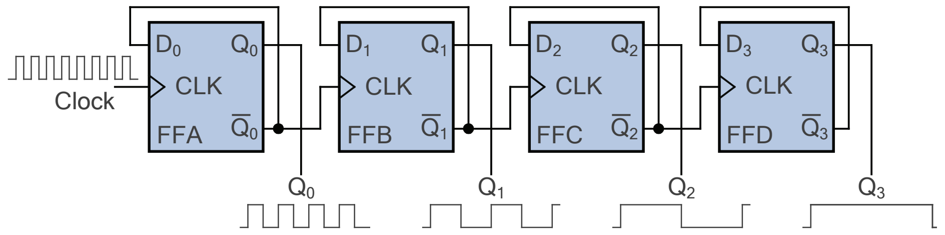
Asynchronous Counters, also know as *ripple counters* are built from positive rising edge triggered D-type (or T-type) flip-flops connected sequentially in toggle mode. Clock pulses are fed into the clock input of the first flip-flop (**FFA**) whose output, Q_0 provides the clock pulse for the second flip-flop (**FFB**) and so on as each successive flip-flop is clocked by the previous one producing a kind of rippling effect through the counter.

We saw in section 7. that the **Data Latch** such as the TTL **74LS74** can be configured as a *divide-by-2* frequency dividers. The maximum input clock frequency of an asynchronous counter decreases with the increase in the number of flip-flop stages or bits.

We can use the frequency division principles of data latches to produce asynchronous counters. These counters can have counting states of any value we want with the number of different binary counting states defining the *modulus* (**MOD**) of the counter.

For example, a counter that counts from 0 to 15 would be called a MOD-16 counter. Then for an asynchronous ripple counter to count from 0 to 15, it must have four binary outputs to produce the four bits and one clock input as shown in Figure 21.

FIGURE 21. 4-BIT ASYNCHRONOUS BINARY UP COUNTER



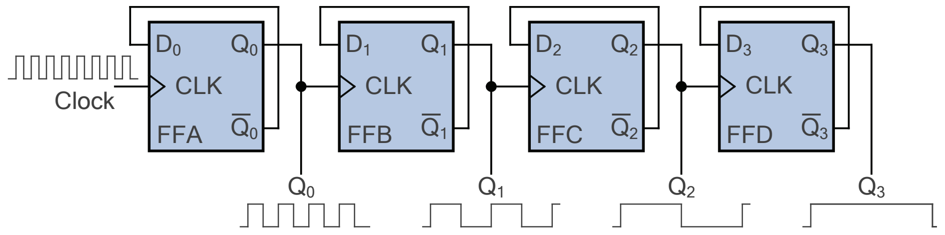
The 4-bit asynchronous binary counter circuit of Figure 21. has 16 different output states, MOD-16 each one corresponding to a count value. The **Q** outputs then represent a four-bit binary count with Q_0 to Q_3 representing decimal 2^0 (1) to 2^3 (8) respectively.

Since this is a four-stage counter, the counter counts upwards in sequence due to the input clock signal starting from 0000 (decimal 0) up to an output 1111 (decimal 15). On the application of the next clock pulse, the output changes back to 0000 (decimal 0) and the count sequence repeats.

Then the Q outputs represent a binary number (from left to right) equalling the number of input pulses received at the clock input of FFA with the frequency of the clock pulses determining the count up count speed.

We can convert the "up" counter in Figure 21. to a binary "down" ripple counter by changing the connections between the flip-flops. By modifying the connection to the clock input of the next flip-flop from $\overline{Q}$ to Q, the counter can be made to count down from 1111_2 to 0000_2 as shown in Figure 22.

FIGURE 22. 4-BIT ASYNCHRONOUS BINARY DOWN COUNTER



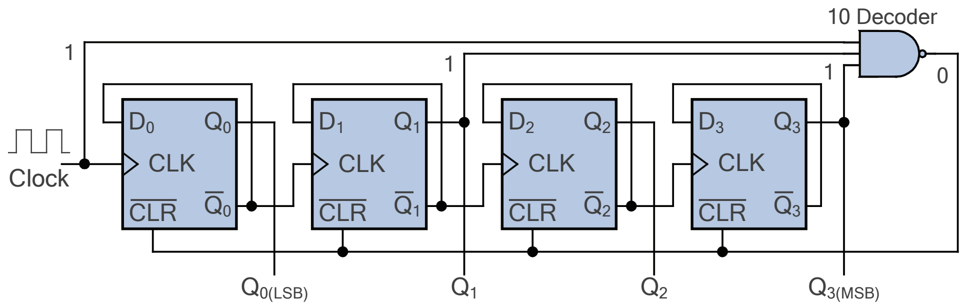
Both Figure 21. and Figure 22. are 4-bit (MOD-16) asynchronous counters with 16 possible counting states, (0-to-15_{10}). But it is also possible to use a basic n-bit counter circuit to construct counters with counting states less than their maximum output count. A counter whose modulus is less than the maximum possible is called a *truncated counter*.

By using external combinational logic gates, we can take advantage of the asynchronous outputs on the individual bistable flip-flops to create any modulus value. For example MOD-5, or MOD-8. One very common truncated counter is the MOD-10 Decade Counter.

10.2 Decade Counters

If we take the modulo-16 asynchronous counter and modified it with additional logic gates it can be made to give a decade (divide-by-10) counter output for use in standard decimal counting and arithmetic circuits. A common decade counter is the BCD counter which produces a *Binary-Coded-Decimal* sequence as shown in Figure 23.

FIGURE 23. 4-BIT ASYNCHRONOUS (BCD) DECADE COUNTER



With the addition of a Clear ($\overline{CLR}$) input, the counter counts upwards from 0000_2 on each clock pulse input. Once the counter counts to ten (1010_2), both outputs Q_1 and Q_3 are now equal to logic-1. On the application of the next clock pulse the output of the NAND gate changes state and all the flip-flops are cleared (reset) back to 0000_2 on the count of 10. The counter restarts again from 0000_2.

Thus we have a decade or Modulo-10 up-counter since none of the other output states (zero to nine) have both Q_1 and Q_3 HIGH at the same time as shown in Table 5.

TABLE 5. 4-BIT DECADE COUNTER TIMING TABLE

Pulse	Q3	Q2	Q1	Q0	Decimal
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9
11	1	0	1	0	Reset

While up, down and truncated counters can be constructed from simple bistable toggling flip-flops, using the asynchronous method for propagating the clock signal through the counter becomes a bit unreliable at high clock speeds, or when a large number of flip-flops are used to give larger MOD counts. This is due to the clock ripple effect making an *Asynchronous Counter*, at times to become unstable. Thus they are not often used.

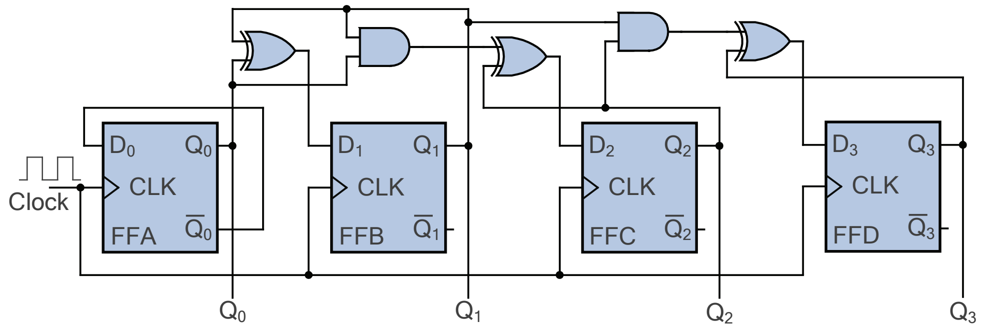
12. SYNCHRONOUS COUNTERS

The toggling of the flip-flops within **Synchronous Counters** are all clocked together at the same time by the same clock pulse. This removes any glitches caused by propagation delays. That is, the input clock pulse is applied to all the toggle flip-flops simultaneously so changes to the output occurs in "synchronisation" with the clock signal. Thus any time delay is that of one counter, resulting in a faster switching operation.

In other words, by default, a synchronous counter is a parallel counter because the clock input is connected in parallel to each and every flip-flop.

Since the synchronous counter configuration can provide us with a more reliable circuit for high speed counting purposes. We can convert our previous binary asynchronous up counter of Figure 21. to operate as a synchronous up counter with the addition of a few external combinational logic **Ex-OR** and **AND** gates as shown in Figure 24.

Figure 24. 4-bit Synchronous Binary Counter



Notice that the clock signal is applied to all the D-type flip-flops in parallel, thus receiving the same clock pulse at the same instant in time.

Then synchronous counters can be formed by connecting bistable flip-flops together and any number of flip-flops can be connected or "cascaded" together to form a "divide-by-n" binary counter. As with the previous asynchronous counters, the modulo or MOD number still applies so Decade counters or BCD counters which count from 0 to 2^n-1 can be built along with truncated sequences.

While asynchronous and synchronous counters can be built from individual flip-flops, there are dedicated commercial counter IC's available such as:

Asynchronous (Ripple) Counters:

74LS90 - Asynchronous ripple counter/divider
74LS390 - Dual decade ripple counter
74LS393 - Dual 4-stage binary ripple counter
74HC4040 - 12-Stage binary ripple counter
74HC4060 - 14-Stage binary ripple counter

Synchronous Counters:

74LS160 - Programmable synchronous BCD counter with asynchronous reset
74LS161 - 4-Bit decade counter with asynchronous reset and synchronous load
74LS163 - 4-Bit binary counter with asynchronous reset and synchronous load
74LS191 - 4-bit synchronous binary up/down counter with reset and load
74LS192 - 4-Bit synchronous BCD counter with asynchronous reset and load
74LS193 - 4-Bit synchronous binary counter with asynchronous reset and load

End of this Sequential Logic eBook

With the completion of this Sequential Logic eBook you should have gained a good and basic understanding and knowledge of the various sequential logic circuits available. The information provided here should give you a firm foundation for continuing your study of electronics and electrical engineering as well as more about sequential circuits.

For more information about any of the topics covered here please visit our website at:

www.electronics-tutorials.ws