**Switching theory & logic Design**

**UNIT -4**

**SEQUENTIAL CIRCUITS:**Unlike [Combinational Logic](http://www.electronics-tutorials.ws/combination/comb_1.html) circuits that change state depending upon the actual signals being applied to their inputs at that time, **Sequential Logic** circuits have some form of inherent “Memory” built in.



The word “Sequential” means that things happen in a “sequence”, one after another and in **Sequential Logic** circuits, the actual clock signal determines when things will happen next. 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**](http://www.electronics-tutorials.ws/logic/logic_5.html) and/or [**NOR Gates**](http://www.electronics-tutorials.ws/logic/logic_6.html) in a particular combinational way to produce the required sequential circuit.

This means that sequential logic circuits are able to take into account their previous input state as well as those actually present, a sort of  “before” and “after” effect is involved with sequential circuits.

In other words, the output state of a “sequential logic circuit” is a function of the following three states, the “present input”, the “past input” and/or the “past output”. Sequential Logic circuits remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits “Memory”.

Sequential logic circuits are generally termed as two state or **[Bistable](http://www.electronics-tutorials.ws/waveforms/bistable.html)** devices which can have their output or outputs set in one of two basic states, a logic level “1” or a logic level “0” and will remain “latched” (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause the bistable to change its state once again.

**SR Flip-Flop**

The **SR flip-flop**, also known as a *SR Latch*, can be considered as one of the most basic sequential logic circuit possible. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will “SET” the device (meaning the output = “1”), and is labelled S and another which will “RESET” the device (meaning the output = “0”), labelled R.

Then the SR description stands for “Set-Reset”. The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level “1” or logic “0” depending upon this set/reset condition.

A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it’s current state or history. The term “Flip-flop” relates to the actual operation of the device, as it can be “flipped” into one logic Set state or “flopped” back into the opposing logic Reset state.

## The NAND Gate SR Flip-Flop

The simplest way to make any basic single bit set-reset SR flip-flop is to connect together a pair of cross-coupled 2-input NAND gates as shown, to form a Set-Reset Bistable also known as an active LOW SR NAND Gate Latch, so that there is feedback from each output to one of the other NAND gate inputs. This device consists of two inputs, one called the Set, S and the other called the Reset, R with two corresponding outputs Q and its inverse or complement Q (not-Q) as shown below.

### The Basic SR Flip-flop



### The Set State

Consider the circuit shown above. If the input R is at logic level “0” (R = 0) and input S is at logic level “1” (S = 1), the NAND gate Y  has at least one of its inputs at logic “0” therefore, its output Q must be at a logic level “1” (NAND Gate principles). Output Q is also fed back to input “A” and so both inputs to NAND gate X are at logic level “1”, and therefore its output Q must be at logic level “0”.

Again NAND gate principals. If the reset input R changes state, and goes HIGH to logic “1” with S remaining HIGH also at logic level “1”, NAND gate Y inputs are now R = “1” and B = “0”. Since one of its inputs is still at logic level “0” the output at Q still remains HIGH at logic level “1” and there is no change of state. Therefore, the flip-flop circuit is said to be “Latched” or “Set” with Q = “1” and Q = “0”.

### Reset State

In this second stable state, Q is at logic level “0”, (not Q = “0”) its inverse output at Q is at logic level “1”, (Q = “1”), and is given by R = “1” and S = “0”. As gate X has one of its inputs at logic “0” its output Q must equal logic level “1” (again NAND gate principles). Output Q is fed back to input “B”, so both inputs to NAND gate Y are at logic “1”, therefore, Q = “0”.

If the set input, S now changes state to logic “1” with input R remaining at logic “1”, output Q still remains LOW at logic level “0” and there is no change of state. Therefore, the flip-flop circuits “Reset” state has also been latched and we can define this “set/reset” action in the following truth table.

### Truth Table for this Set-Reset Function

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| State | S | R | Q | Q | Description |
| Set | 1 | 0 | 0 | 1 | Set Q » 1 |
| 1 | 1 | 0 | 1 | no change |
| Reset | 0 | 1 | 1 | 0 | Reset Q » 0 |
| 1 | 1 | 1 | 0 | no change |
| Invalid | 0 | 0 | 1 | 1 | Invalid Condition |

It can be seen that when both inputs S = “1” and R = “1” the outputs Q and Q can be at either logic level “1” or “0”, depending upon the state of the inputs S or R BEFORE this input condition existed. Therefore the condition of S = R = “1” does not change the state of the outputs Q and Q.

However, the input state of S = “0” and R = “0” is an undesirable or invalid condition and must be avoided. The condition of S = R = “0” causes both outputs Q and Q to be HIGH together at logic level “1” when we would normally want Q to be the inverse of Q. The result is that the flip-flop looses control of Q and Q, and if the two inputs are now switched “HIGH” again after this condition to logic “1”, the flip-flop becomes unstable and switches to an unknown data state based upon the unbalance as shown in the following switching diagram.

### S-R Flip-flop Switching Diagram



This unbalance can cause one of the outputs to switch faster than the other resulting in the flip-flop switching to one state or the other which may not be the required state and data corruption will exist. This unstable condition is generally known as its **Meta-stable**state.

Then, a simple NAND gate SR flip-flop or NAND gate SR latch can be set by applying a logic “0”, (LOW) condition to its Set input and reset again by then applying a logic “0” to its Reset input. The SR flip-flop is said to be in an “invalid” condition (Meta-stable) if both the set and reset inputs are activated simultaneously.

As we have seen above, the basic NAND gate SR flip-flop requires logic “0” inputs to flip or change state from Q to Q and vice versa. We can however, change this basic flip-flop 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.

**JK FLIP-FLOP:**

**INTRODUCTION:**

JK flip – flop is named after Jack Kilby, the electrical engineer who invented IC. A JK flip – flop is called a Universal Programmable flip – flop because, using its inputs J, K Preset and Clear, function of any other flip – flop can be imitated.

A JK flip – flop is the modification of SR flip – flop with no illegal state. In this the J input is similar to the SET input of SR flip – flop and the K input is similar to the RESET input of SR flip – flop. The symbol of JK flip – flop is shown below.



##### **JK flip flop Logic Diagram**

JK flip – flop logic diagram is shown in the below figure. As said before, JK flip – flop is a modified version of SR flip – flop. Logic diagram consists of three input NAND gates replacing the two input NAND gates in SR flip – flop and the inputs are replaced with J and K from S and R.

The design of the JK flip – flop is such that the three inputs to one NAND gate are J, clock signal along with a feedback signal from Q’ and the three inputs to the other NAND are K, clock signal along with a feedback signal from Q. This arrangement eliminates the indeterminate state in SR flip – flop.



##### **Truth Table**



### Operation

##### Case  1 : When both the inputs J and K are LOW, then Q returns its previous state value i.e. it holds the previous data.

When we apply a clock pulse to the J K flip flop and the J input is low then irrespective of the other NAND gates, the NAND gate-1 output becomes HIGH. In the same manner, if the K input is low then output of NAND gate-2 is also HIGH. So thus the output remains in the same state i.e. no change in the state of flip flop.

##### Case  2 : When J is LOW and K is HIGH, then flip flop will be in Reset state i.e. Q = 0, Q’ = 1.

When we apply a clock pulse to the J K flip flop and the inputs are J is low and K is high the output of the NAND gate connected to J input becomes 1. Then Q becomes 0. This will reset the flip flop again to its previous state. So the Flip flop will be in RESET state.

##### Case  3 : When J is HIGH and K is LOW, then flip – flop will be in Set state i.e. Q = 1, Q’ = 0

When we apply a clock pulse to the J K flip flop and the inputs are J is high and K is low the output of the NAND gate connected to K input becomes 1. Then Q’ becomes 0. This will set the flip flop with the high clock input. So the Flip flop will be in SET state.

##### Case  4 : When both the inputs J and K are HIGH, then flip – flop is in Toggle state. This means that the output will complement of the previous state.

##### **Truth Table**

The truth table of JK flip – flop is shown below.



## **D Flip Flop:**

The circuit diagram and truth table is given below.

****

D flip flop is actually a slight modification of the above explained clocked SR flip-flop. From the figure you can see that the D input is connected to the S input and the complement of the D input is connected to the R input. The D input is passed on to the flip flop when the value of CP is ‘1’. When CP is HIGH, the flip flop moves to the SET state. If it is ‘0’, the flip flop switches to the CLEAR state.

## **T Flip Flop:**

This is a much simpler version of the J-K flip flop. Both the J and K inputs are connected together and thus are also called a single input J-K flip flop. When clock pulse is given to the flip flop, the output begins to toggle. Here also the restriction on the pulse width can be eliminated with a master-slave or edge-triggered construction. Take a look at the circuit and truth table below****

**MASTER-SLAVE FLIP-FLOP:**

Master-slave flip flop is designed using two separate flip flops. Out of these, one acts as the master and the other as a slave. The figure of a master-slave J-K flip flop is shown below.



*Master Slave Flip Flop*

From the above figure you can see that both the J-K flip flops are presented in a series connection. The output of the master J-K flip flop is fed to the input of the slave J-K flip flop. The output of the slave J-K flip flop is given as a feedback to the input of the master J-K flip flop. The clock pulse [Clk] is given to the master J-K flip flop and it is sent through a NOT Gate and thus inverted before passing it to the slave J-K flip flop.

## Working

When Clk=1, the master J-K flip flop gets disabled. The Clk input of the master input will be the opposite of the slave input. So the master flip flop output will be recognized by the slave flip flop only when the Clk value becomes 0. Thus, when the clock pulse males a transition from 1 to 0, the locked outputs of the master flip flop are fed through to the inputs of the slave flip-flop making this flip flop edge or pulse-triggered. To understand better take a look at the timing diagram illustrated below.



*Master Slave J-K Flip Flop Timing Diagram*

Thus, the circuit accepts the value in the input when the clock is HIGH, and passes the data to the output on the falling-edge of the clock signal. This makes the Master-Slave J-K flip flop a Synchronous device as it only passes data with the timing of the clock signal

**SHIFT REGISTERS:**

Basically shift registers are of 4 types. They are

* Serial In Serial Out shift register
* Serial In parallel Out shift register
* Parallel In Serial Out shift register
* Parallel In parallel Out shift register

### Serial in Serial Out Shift Register

The input to this register is given in serial fashion i.e. one bit after the other through a single data line and the output is also collected serially. The data can be shifted only left or shifted only right. Hence it is called Serial in Serial out shift register or a SISO shift register.

As the data is fed from right as bit by bit, the shift register shifts the data bits to left. A 4-bit SISO shift register consists of 4 flip flops and only three connections.

The registers which will shift the bits to left are called “Shift left registers”.

The registers which will shift the bits to right are called “Shift right registers”.

Example: If we pass the data 1101 to the data input, the shifted output will be 0110.



This one is the simplest register among the four types. As the clock signal is connected to all the 4 flip flops, the serial data is connected to the left most or right most flip flop. The output of the first flip flop is connected to the input of the next flip flop and so on. The final output of the shift register is collected at the outmost flip flop.



In the above diagram, we see the shift right register; feeding the serial data input from the left side of the flip flop arrangement.

In this shift register, when the clock signal is applied and the serial data is given; only one bit will be available at output at a time in the order of the input data. The use of SISO shift register is to act as temporary data storage device. But the main use of a SISO is to act as a delay element.

### Serial in Parallel Out shift register

The input to this register is given in serial and the output is collected in parallel



The clear (CLR) signal is connected in addition to clock signal to all the 4 flip flops in order to RESET them and the serial data is connected to the flip flop at either end (depending on shift left register or shift right register). The output of the first flip flop is connected to the input of the next flip flop and so on. All the flip flops are connected with a common clock.



Unlike the serial in serial out shift registers, the output of Serial in Parallel out (SIPO) shift register is collected at each flip flop. Q1, Q2, Q3 and Q4 are the outputs of first, second, third and fourth flip flops, respectively.

The main application of Serial in Parallel out shift register is to convert serial data into parallel data. Hence they are used in communication lines where demultiplexing of a data line into several parallel line is required.

### Parallel in Serial out shift register

The input to this register is given in parallel i.e. data is given separately to each flip flop and the output is collected in serial at the output of the end flip flop.



The clock input is directly connected to all the flip flops but the input data is connected individually to each flip flop through a mux (multiplexer) at input of every flip flop. Here D1, D2, D3 and D4 are the individual parallel inputs to the shift register. In this register the output is collected in serial.



The output of the previous flip flop and parallel data input are connected to the input of the MUX and the output of MUX is connected to the next flip flop. A Parallel in Serial out (PISO) shift register converts parallel data to serial data. Hence they are used in communication lines where a number of data lines are multiplexed into single serial data line.

### Parallel in Parallel out shift register

In this register, the input is given in parallel and the output also collected in parallel. The clear (CLR) signal and clock signals are connected to all the 4 flip flops. Data is given as input separately for each flip flop and in the same way, output also collected individually from each flip flop.



The above diagram shows the 4 stage parallel in parallel out register. Qa, Qb, Qc and Qd are the parallel outputs and Pa, Pb, Pc and Pd are the individual parallel inputs. There are no interconnections between any of the four flip flops.



A Parallel in Parallel out (PIPO) shift register is used as a temporary storage device and also as a delay element similar to a SISO shift register.

**ASYNCHRONOUS COUNTER**:

In the previous section, we saw a circuit using one J-K flip-flop that counted backward in a two-bit binary sequence, from 11 to 10 to 01 to 00. Since it would be desirable to have a circuit that could count *forward* and not just backward, it would be worthwhile to examine a forward count sequence again and look for more patterns that might indicate how to build such a circuit.

Since we know that binary count sequences follow a pattern of octave (factor of 2) frequency division, and that J-K flip-flop multivibrators set up for the “toggle” mode are capable of performing this type of frequency division, we can envision a circuit made up of several J-K flip-flops, cascaded to produce four bits of output. The main problem facing us is to determine *how* to connect these flip-flops together so that they toggle at the right times to produce the proper binary sequence. Examine the following binary count sequence, paying attention to patterns preceding the “toggling” of a bit between 0 and 1:



Note that each bit in this four-bit sequence toggles when the bit before it (the bit having a lesser significance, or place-weight), toggles in a particular direction: from 1 to 0. Small arrows indicate those points in the sequence where a bit toggles, the head of the arrow pointing to the previous bit transitioning from a “high” (1) state to a “low” (0) state:



Starting with four J-K flip-flops connected in such a way to always be in the “toggle” mode, we need to determine how to connect the clock inputs in such a way so that each succeeding bit toggles when the bit before it transitions from 1 to 0. The Q outputs of each flip-flop will serve as the respective binary bits of the final, four-bit count:



If we used flip-flops with negative-edge triggering (bubble symbols on the clock inputs), we could simply connect the clock input of each flip-flop to the Q output of the flip-flop before it, so that when the bit before it changes from a 1 to a 0, the “falling edge” of that signal would “clock” the next flip-flop to toggle the next bit:



This circuit would yield the following output waveforms, when “clocked” by a repetitive source of pulses from an oscillator:



The first flip-flop (the one with the Q0 output), has a positive-edge triggered clock input, so it toggles with each rising edge of the clock signal. Notice how the clock signal in this example has a duty cycle less than 50%. I’ve shown the signal in this manner for the purpose of demonstrating how the clock signal need not be symmetrical to obtain reliable, “clean” output bits in our four-bit binary sequence. In the very first flip-flop circuit shown in this chapter, I used the clock signal itself as one of the output bits. This is a bad practice in counter design, though, because it necessitates the use of a square wave signal with a 50% duty cycle (“high” time = “low” time) in order to obtain a count sequence where each and every step pauses for the same amount of time. Using one J-K flip-flop for each output bit, however, relieves us of the necessity of having a symmetrical clock signal, allowing the use of practically any variety of high/low waveform to increment the count sequence.

As indicated by all the other arrows in the pulse diagram, each succeeding output bit is toggled by the action of the preceding bit transitioning from “high” (1) to “low” (0). This is the pattern necessary to generate an “up” count sequence.

A less obvious solution for generating an “up” sequence using positive-edge triggered flip-flops is to “clock” each flip-flop using the Q’ output of the preceding flip-flop rather than the Q output. Since the Q’ output will always be the exact opposite state of the Q output on a J-K flip-flop (no invalid states with this type of flip-flop), a high-to-low transition on the Q output will be accompanied by a low-to-high transition on the Q’ output. In other words, each time the Q output of a flip-flop transitions from 1 to 0, the Q’ output of the same flip-flop will transition from 0 to 1, providing the positive-going clock pulse we would need to toggle a positive-edge triggered flip-flop at the right moment:



One way we could expand the capabilities of either of these two counter circuits is to regard the Q’ outputs as another set of four binary bits. If we examine the pulse diagram for such a circuit, we see that the Q’ outputs generate a *down*-counting sequence, while the Q outputs generate an *up*-counting sequence:





Unfortunately, all of the counter circuits shown thusfar share a common problem: the *ripple* effect. This effect is seen in certain types of binary adder and data conversion circuits, and is due to accumulative propagation delays between cascaded gates. When the Q output of a flip-flop transitions from 1 to 0, it commands the next flip-flop to toggle. If the next flip-flop toggle is a transition from 1 to 0, it will command the flip-flop after it to toggle as well, and so on. However, since there is always some small amount of propagation delay between the command to toggle (the clock pulse) and the actual toggle response (Q and Q’ outputs changing states), any subsequent flip-flops to be toggled will toggle some time *after* the first flip-flop has toggled. Thus, when multiple bits toggle in a binary count sequence, they will not all toggle at exactly the same time:



As you can see, the more bits that toggle with a given clock pulse, the more severe the accumulated delay time from LSB to MSB. When a clock pulse occurs at such a transition point (say, on the transition from 0111 to 1000), the output bits will “ripple” in sequence from LSB to MSB, as each succeeding bit toggles and commands the next bit to toggle as well, with a small amount of propagation delay between each bit toggle. If we take a close-up look at this effect during the transition from 0111 to 1000, we can see that there will be *false* output counts generated in the brief time period that the “ripple” effect takes place:



Instead of cleanly transitioning from a “0111” output to a “1000” output, the counter circuit will very quickly ripple from 0111 to 0110 to 0100 to 0000 to 1000, or from 7 to *6* to *4* to *0* and then to 8. This behavior earns the counter circuit the name of *ripple counter*, or *asynchronous counter*.

In many applications, this effect is tolerable, since the ripple happens very, very quickly (the width of the delays has been exaggerated here as an aid to understanding the effects). If all we wanted to do was drive a set of light-emitting diodes (LEDs) with the counter’s outputs, for example, this brief ripple would be of no consequence at all. However, if we wished to use this counter to drive the “select” inputs of a multiplexer, index a memory pointer in a microprocessor (computer) circuit, or perform some other task where false outputs could cause spurious errors, it would not be acceptable. There is a way to use this type of counter circuit in applications sensitive to false, ripple-generated outputs, and it involves a principle known as *strobing*.

Most decoder and multiplexer circuits are equipped with at least one input called the “enable.” The output(s) of such a circuit will be active only when the enable input is made active. We can use this enable input to *strobe* the circuit receiving the ripple counter’s output so that it is disabled (and thus not responding to the counter output) during the brief period of time in which the counter outputs might be rippling, and enabled only when sufficient time has passed since the last clock pulse that all rippling will have ceased. In most cases, the strobing signal can be the same clock pulse that drives the counter circuit:



With an active-low Enable input, the receiving circuit will respond to the binary count of the four-bit counter circuit only when the clock signal is “low.” As soon as the clock pulse goes “high,” the receiving circuit stops responding to the counter circuit’s output. Since the counter circuit is positive-edge triggered (as determined by the *first* flip-flop clock input), all the counting action takes place on the low-to-high transition of the clock signal, meaning that the receiving circuit will become disabled just before any toggling occurs on the counter circuit’s four output bits. The receiving circuit will not become enabled until the clock signal returns to a low state, which should be a long enough time *after* all rippling has ceased to be “safe” to allow the new count to have effect on the receiving circuit. The crucial parameter here is the clock signal’s “high” time: it must be at least as long as the maximum expected ripple period of the counter circuit. If not, the clock signal will prematurely enable the receiving circuit, while some rippling is still taking place.

Another disadvantage of the asynchronous, or ripple, counter circuit is limited speed. While all gate circuits are limited in terms of maximum signal frequency, the design of asynchronous counter circuits compounds this problem by making propagation delays additive. Thus, even if strobing is used in the receiving circuit, an asynchronous counter circuit cannot be clocked at any frequency higher than that which allows the greatest possible accumulated propagation delay to elapse well before the next pulse.

The solution to this problem is a counter circuit that avoids ripple altogether. Such a counter circuit would eliminate the need to design a “strobing” feature into whatever digital circuits use the counter output as an input, and would also enjoy a much greater operating speed than its asynchronous equivalent

**SYNCHRONOUS COUNTER:**

A *synchronous counter*, in contrast to an *asynchronous counter*, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time:



Now, the question is, what do we do with the J and K inputs? We know that we still have to maintain the same divide-by-two frequency pattern in order to count in a binary sequence, and that this pattern is best achieved utilizing the “toggle” mode of the flip-flop, so the fact that the J and K inputs must both be (at times) “high” is clear. However, if we simply connect all the J and K inputs to the positive rail of the power supply as we did in the asynchronous circuit, this would clearly not work because all the flip-flops would toggle at the same time: with each and every clock pulse!



Let’s examine the four-bit binary counting sequence again, and see if there are any other patterns that predict the toggling of a bit. Asynchronous counter circuit design is based on the fact that each bit toggle happens at the same time that the preceding bit toggles from a “high” to a “low” (from 1 to 0). Since we cannot clock the toggling of a bit based on the toggling of a previous bit in a synchronous counter circuit (to do so would create a ripple effect) we must find some other pattern in the counting sequence that can be used to trigger a bit toggle:

Examining the four-bit binary count sequence, another predictive pattern can be seen. Notice that just before a bit toggles, all preceding bits are “high:”



This pattern is also something we can exploit in designing a counter circuit. If we enable each J-K flip-flop to toggle based on whether or not all preceding flip-flop outputs (Q) are “high,” we can obtain the same counting sequence as the asynchronous circuit without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time:



The result is a four-bit *synchronous* “up” counter. Each of the higher-order flip-flops are made ready to toggle (both J and K inputs “high”) if the Q outputs of all previous flip-flops are “high.” Otherwise, the J and K inputs for that flip-flop will both be “low,” placing it into the “latch” mode where it will maintain its present output state at the next clock pulse. Since the first (LSB) flip-flop needs to toggle at every clock pulse, its J and K inputs are connected to Vcc or Vdd, where they will be “high” all the time. The next flip-flop need only “recognize” that the first flip-flop’s Q output is high to be made ready to toggle, so no AND gate is needed. However, the remaining flip-flops should be made ready to toggle only when *all* lower-order output bits are “high,” thus the need for AND gates.

To make a synchronous “down” counter, we need to build the circuit to recognize the appropriate bit patterns predicting each toggle state while counting down. Not surprisingly, when we examine the four-bit binary count sequence, we see that all preceding bits are “low” prior to a toggle (following the sequence from bottom to top):



Since each J-K flip-flop comes equipped with a Q’ output as well as a Q output, we can use the Q’ outputs to enable the toggle mode on each succeeding flip-flop, being that each Q’ will be “high” every time that the respective Q is “low:”



Taking this idea one step further, we can build a counter circuit with selectable between “up” and “down” count modes by having dual lines of AND gates detecting the appropriate bit conditions for an “up” and a “down” counting sequence, respectively, then use OR gates to combine the AND gate outputs to the J and K inputs of each succeeding flip-flop:



This circuit isn’t as complex as it might first appear. The Up/Down control input line simply enables either the upper string or lower string of AND gates to pass the Q/Q’ outputs to the succeeding stages of flip-flops. If the Up/Down control line is “high,” the top AND gates become enabled, and the circuit functions exactly the same as the first (“up”) synchronous counter circuit shown in this section. If the Up/Down control line is made “low,” the bottom AND gates become enabled, and the circuit functions identically to the second (“down” counter) circuit shown in this section.

To illustrate, here is a diagram showing the circuit in the “up” counting mode (all disabled circuitry shown in grey rather than black):



Here, shown in the “down” counting mode, with the same grey coloring representing disabled circuitry:



Up/down counter circuits are very useful devices. A common application is in machine motion control, where devices called *rotary shaft encoders* convert mechanical rotation into a series of electrical pulses, these pulses “clocking” a counter circuit to track total motion.

RINGLE JOHNSON COUNTER:

A Johnson counter is a modified ring counter, where the inverted output from the last flip flop is connected to the input to the first. The register cycles through a sequence of bit-patterns. The MOD of the Johnson counter is 2n if n flip-flops are used. The main advantage of the Johnson counter counter is that it only needs half the number of flip-flops compared to the standard [ring counter](http://electronics-course.com/ring-counter) for the same MOD.

It can be implemented using [D-type flip-flops](http://electronics-course.com/d-flip-flop) (or JK-type flip-flops).

Notes:

* Enable the flips flops by clicking on the RESET (Green) switch. The RESET switch is a on/off switch (similar to a room light switch)
* Click on CLK (Red) switch and observe the changes in the outputs of the flip flops. The CLK switch is a momentary switch (similar to a door bell switch - normally off).
* The D flip flop clock has a rising edge CLK input. For example Q1 behaves as follows:
	+ The D input value just before the CLK rising edge is noted (Q0).
	+ When CLK rising edge occurs, Q1 is assigned the previously noted D value (Q0).



* The MOD or number of unique states of this 3 flip flop johnson counter is 6.

|  |
| --- |
| **Truth Table** |
| **State** | **Q0** | **Q1** | **Q2** |
| **0** | 0 | 0 | 0 |
| **1** | 1 | 0 | 0 |
| **2** | 1 | 1 | 0 |
| **3** | 1 | 1 | 1 |
| **4** | 0 | 1 | 1 |
| **5** | 0 | 0 | 1 |