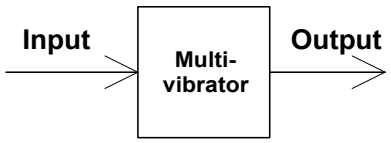
DIGITAL ELECTRONICS
Part 2: Sequential Logic

INTRODUCTION

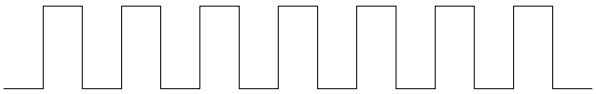
The principal behavior to be developed in this section is that of ***memory***. This is a capability critical to successful application of digital electronics in a widespread way since without it, no intermediate results could be held for future use so that sequences of operations would be impossible. Our implementation of memory will be developed from the basic operations introduced in ***Part 1*** and will be centered on the ***flip-flop***. Flip-flops are an important class of circuits and provide many capabilities other than memory, such as counters, shift registers, *etc.*

To develop flip-flops, we will begin with a broader family of circuits, the ***multivibrators***. Multivibrators fall into three categories: ***astable, monostable,*** and ***bistable***. Each member (except astable) can be envisioned as having an input and an output taking on a binary set of states ("0" and "1") as indicated schematically.



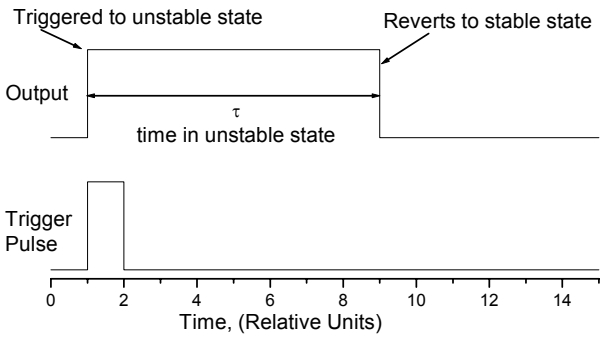
The defining characteristics of each member of the multivibrator family are:

Astable Multivibrators: The output is unstable in either state. That is, the output continually changes from 0 to 1 to 0, *etc.* Obviously, this is the same as an oscillator and this condition is sometimes referred to as "free-running." Moreover, the ***astable multivibrator*** has no input. Astable multivibrators are used as the time base for automatic and / or sequential devices.



Astable Multivibrator Waveform

Monostable Multivibrators: The output is stable in one state and unstable in the other. The specific behavior is that the "input" is a "trigger" which causes the output to make a transition from the stable to the unstable state, where it remains for a length of time (τ), and then reverts to the stable state. Monostable multivibrators are sometimes referred to as ***one-shots*** since one "shot" from the trigger creates a change in the output. One-shots are useful in timing applications, and for small-scale sequencing circuits, although their precision is less than that available from oscillator-based circuits.



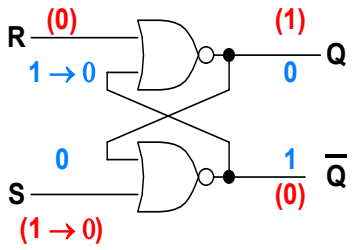
Monostable Multivibrator Waveform

Bistable Multivibrators The output is stable in both states although the input can "trigger" a transition from one state to the other. Bistable circuits are sometimes referred to as "binaries." We will introduce this circuit as the ***flip-flop*** which will form the basis of the remainder of our discussions.

FLIP-FLOPS

We will develop several versions of flip-flops; however all will be based on the most primitive form, the **RS flip-flop or RSFF.** A common implementation of the RSFF is from NOR gates as sketched below.

While the labeling of the circuit is conventional, the Q and $\overline{Q}$ labels are not strictly correct as we will see in the analysis to follow. The truth table for the circuit is:



RSFF From NOR's

RSFF Truth Table			
R	S	Q	$\overline{Q}$
1	1	0	0
0	1	1	0
1	0	0	1
0	0	0	1
(0)	(1)	(1)	(0)
(0)	(0)	(1)	(0)

The truth table indicates that the state R=0, S=0 (the "0,0" state) gives different output results depending on the immediately preceding state. This is an important deduction and needs further discussion. Specifically, for the initial state R=1, S=0 (indicated in blue and without parentheses), the outputs are Q=0 and $\overline{Q} = 1$. If R is changed to 0 and S held at 0, the question is whether or not there is any reason for the outputs to change. Note that the ***initial*** value of R=1 forced Q=0 this and S=0 made $\overline{Q} = 1$; . However, the change of R to "0" does not control the Q output since it is an input to a NOR. In fact, $\overline{Q}$ is and is connected to the same NOR as S; thus $\overline{Q} = 1$ forces Q=0 which combines with S=0 to hold $\overline{Q} = 1$. Similarly, for the case in parentheses (and in red), starting with the initial state R=0, S=1 and changing S to "0" while keeping R=1 leads to no change in the outputs (this time being Q=1 and $\overline{Q} = 0$).

Based on this realization, we can re-state the truth table as indicated to the right. The main point is that the "0,0" state leads to no change in the output state. This behavior is important and central to the many uses of flip-flops, including memory.

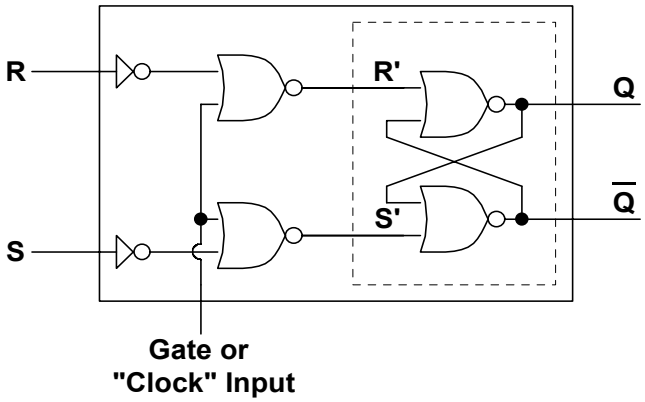
(A minor point is that the "1,1" state is usually excluded in the more widely used versions of flip-flops. Also, the "R" and "S" labels for the inputs are abbreviations of "Reset" and "Set," respectively. The names come from the facts that the S=1, R=0 inputs make or "set" Q to 1; likewise the S=0, R=1 inputs "reset" Q to 0. Resetting Q is also referred to as "clearing" it.)

RSFF Truth Table			
R	S	Q	$\overline{Q}$
1	1	0	0
0	1	1	0
1	0	0	1
0	0	no change from previous	

While the most primitive of the flip-flops we will consider, the RSFF is at the heart of all others. It is important also that the NOR version is only one way to implement the RSFF; in particular NAND gates in the came configuration also leads to the RSFF behavior. However, with NAND's the "no change" behavior is exhibited by the "1,1" input states rather than the "0,0" combination found for NOR's. ***In summary, the important point is that there exists a "no-change" state and not the way in which it is implemented.***

THE "GATED" OR "CLOCKED" RSFF

The next step in developing useful versions of the flip-flop is to introduce additional NOR's for adding a control function as indicated in the sketch. (The NOT's in the R and S input lines are not strictly necessary; they are inserted in this case to cancel the NOT introduced by the additional NOR gates.)



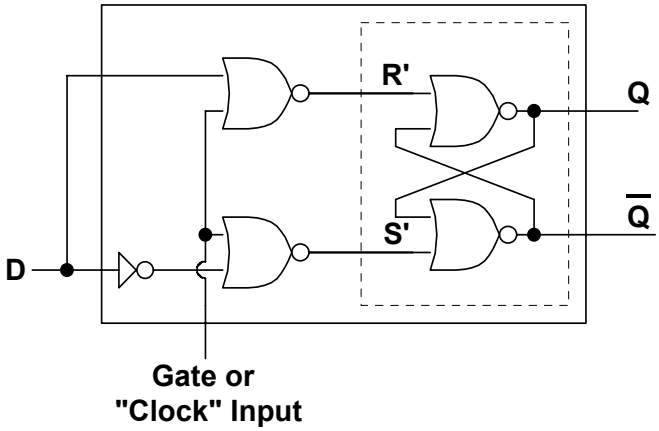
Gated (Clocked) RSFF

The operation of this circuit is as follows: when the GATE signal is "0", the circuit behaves the same as the simple RSFF described above. However, when the GATE signal is "1," the simple RSFF enclosed by the dotted lines is isolated from the R&S inputs. Moreover the GATE = 1 state forces R' and S' to "0" with the

result that the outputs are not changed (unless R=S=1).

DATA "LATCH" OR D FLIP-FLOP (DFF)

Removing the NOT generator from the R input and tying the two inputs of the gated RSFF together yields the circuit shown in the sketch. Removing the NOT from the R input and tying the resulting lines together effectively make $R = \overline{S}$ and eliminates the possibility of R=S=0. In addition, the R and S inputs can never be the same; specifically, when D=0, S=0 and R=1; when D=1, S=1 and R=0. Thus, when GATE=0, Q=D (since Q=S and S=D), and when GATE=1, Q equals D as it was when GATE became 1.

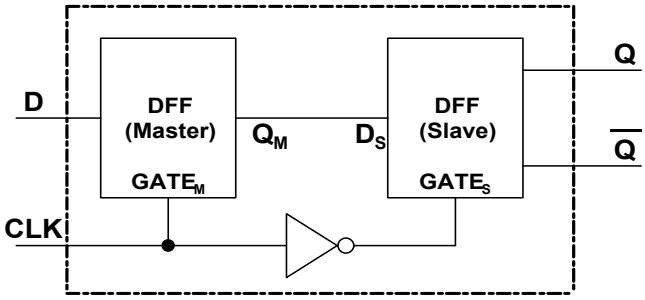


DFF (from gated RSFF)

In other words, when GATE=0, Q is the ***stored*** value of D. Thus the DFF is a primitive one-bit ***memory*** cell. D is the data input, GATE is the WRITE / STORE control, and Q is the stored value.

MASTER-SLAVE DFF

The DFF circuit above has the property that the Q output changes as the D input changes when GATE=1. In many cases it is better to have Q isolated from D. One way to do so is to use the "master-slave" approach as indicated in the sketch.



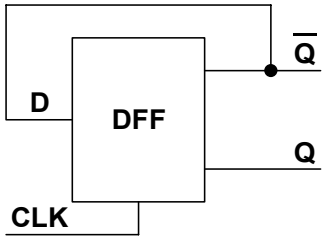
Master-Slave DFF

In the master-slave circuit, the output of the master (Q_M) changes as long as D changes when CLK=1. Since the NOT makes $GATE_S$=0 when CLK=1, the overall output does not change while CLK=1. However, when CLK=0, the output of the master (Q_M) is "transferred" to the overall output since $GATE_S$=1 when CLK=0. But, since $GATE_M$=CLK=0, Q_M is isolated from the input (D). It is important to realize that the data stored is that present at D the moment CLK goes from 1 to 0 and that transfer from D to Q requires a complete cycle of CLK–from 0 to 1 to 0.

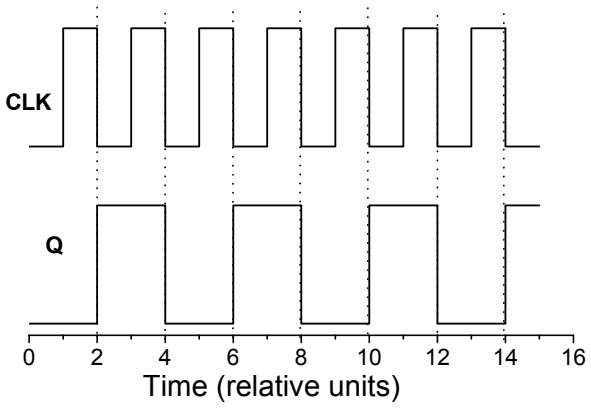
INTRODUCTION TO COUNTING: THE "TOGGLE" OPERATION

Shown in the sketch is a DFF with the $\overline{Q}$ output connected back to the D input. (This works only for DFF's with Q and D isolated as is the case with the master-slave configuration.) With this connection, D is connected to "what it isn't." As a result, with each full cycle of CLK, Q changes either from 0 to 1 or from 1 to 0. The output waveform (Q) in relation to the input (CLK) is shown in the sketch.

Inspection of the waveforms shows that the frequency of the output (Q) is half that of the input (CLK). Thus the most obvious result of DFF toggle operation is division of a frequency by one-half.

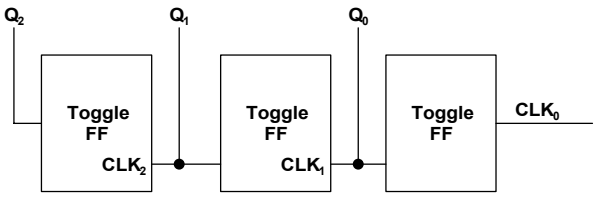


DFF Connected for the Toggle Operation



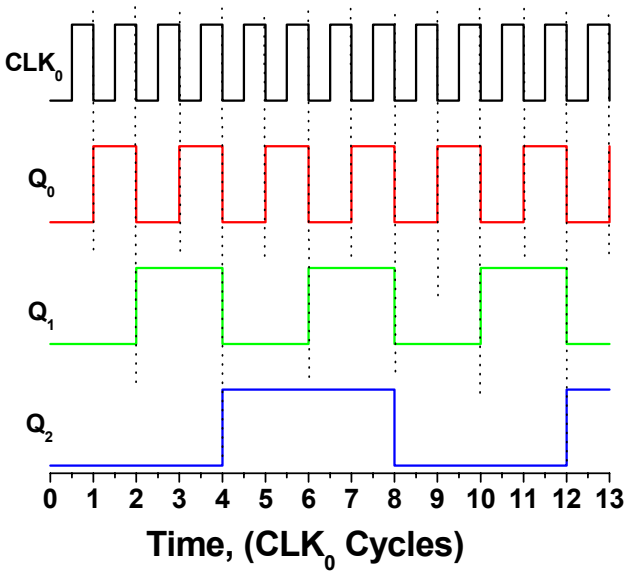
Q *vs.* CLK Waveforms for the DFF Toggle Operation

However, the toggle behavior can also be developed into a counting circuit. For example, consider the circuit shown below:
Waveforms relating the input (CLK_0) to the three outputs (Q_0, Q_1, and Q_2) are sketched below and the relationships between the output states and the input are listed in the adjacent table. (Note that the transitions of all Q's occur on the 1 to 0



3-Stage Toggle FF arrangement

(high to low) transitions of their input. Thus, the "code" resulting from the n_{th} CLK pulse occurs on the high-to-low transition of CLK_0. These are indicated by the vertical dotted lines in the figure.)



Sequence of Codes

Q_2	Q_1	Q_0	CLK_0
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	8
0	0	1	9
0	1	0	10
0	1	1	11
1	0	0	12

Obviously, this sequence of codes represents the number of CLK_0 cycles since all Q's were 0. Thus, the circuit provides a 3-bit binary representation of the number of pulses and is a counter. Extension of the circuit to higher numbers of bits is straightforward and requires only that additional toggle flip-flops be added.

An important observation is that the count is a binary representation. However, binary representations are not usually convenient to us as we are culturally more familiar with decimal. So the question arises as to how we can create a counter with decimal character. The answer comes from our previous experience with combinatorial logic and recollection that the "base" of a count is determined by the highest number it can represent before "rolling over" to all 0's. For a 3-bit binary counter as in our example above, this is 7. For a decimal counter the "terminal count" is 9. Thus, if we can cause the counter to become all 0's on the 10^{th} count, we will have created a decimal counter. To do so, we need flip-flops with the capability of being

"directly cleared," a feature of most readily available FF's. (See for example the data sheet of the 74x74 DFF.) With the direct clear capability, we need only to monitor the states of the Q's and activate the "direct clear" on the combination indicating "9" AND a count pulse: $RST = Q_0 \bullet \overline{Q_1} \bullet \overline{Q_2} \bullet Q_3 \bullet CLK_0$. (Note also that counting beyond 7 requires 4 flip-flops with outputs Q_0-Q_3 .)