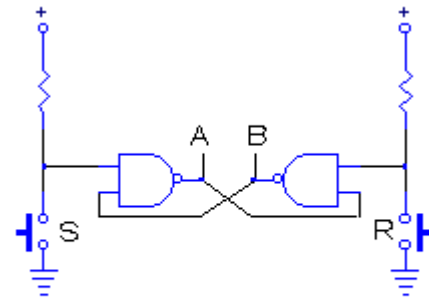


## Chapter 4 – Flip-Flops and Registers

Digital Design has two main areas: combinational logic, and clocked or registered logic. We have already discussed the former in depth. We now need to take a look at the latter.

In combinational logic, the output is the instantaneous function of the inputs based on some logic relationship. If an input changes in such a way to change the result of the logic, the output changes correspondingly, essentially instantaneously. There is a whole area of logic where we wish the output to be not only a function of present inputs, but also a function of history, of what went on at some previous time. For this we need to have some logic element that is memory, specifically we need a construct that is one bit of memory, that can remember a one or zero indefinitely even if the conditions that produce the one or zero have long since disappeared.

Consider the circuit of Figure 4-1. The circuit consists of two NAND gates. One input of each gate is tied to the +V rail through a resistor. The other input is connected to the output of the other NAND gate. What is the output, Q? Let us assume that it is high. This high is applied to the input of gate B. Both inputs of B are high, and the output is true (low). The low output of B is connected to an input of A forcing the output of A to be high, just as we assume. It looks like we guessed right! Supposed we assumed that the output of A was low. In that case, the low applied to B would force its output high. The output of B applied to the input of A would make A true or low. It looks like we would be right again. They both can't be right, can they?



**Figure 4-1 R-S Flip-flop made with NAND gates**

The circuit of Figure 4-1 is called a flip-flop. (Originally it was called an Eccles-Jordan circuit (after its inventors who used vacuum tubes to make it), and also a bi-stable multi-vibrator). A flip-flop has two stable states. Whatever state it is in, it remains happy in that state and does not change spontaneously.

Consider what happens when switch S is pressed. The input S is connected to goes low, forcing the output of gate A high. This now forces the output of B low, which when applied to A, holds A high (which holds B low). Now if the switch is released, A remains high. Let us call the output of A, Q, the output of our flip-flop. The switch S causes the flip-flop to be *set*, that is, made a one. Similarly, if switch R is pressed momentarily, the output of gate B goes high, making Q go low. When the switch is released, Q stays low. We say that the switch R *resets* flip-flop Q, or makes it zero. This is called a set-reset flip-flop, or more appropriately a direct set-reset flip-flop. The *direct* means the setting or resetting action takes place immediately.

It should be observed that if no switches are currently being pressed, the output of B is always the invert of the output of A, or Q-bar! But if both switches are pressed, the outputs of both gates A and B are high. If one is released, the switch last pressed wins. But what happens if both are released at the same time. Who wins? Nobody knows. Simultaneous release of the S and R switches results in an undetermined state. Note, that either one or the other will be high. There is no halfway state. Because of this ambiguous state possibility, S-R flip-flops are used only when the setting and resetting are known not to conflict.

A good many of applications of flip-flops require that they change state at precise times and synchronize with some repetitive event, usually a clock. Aside from the possible state ambiguity, the type of flip-flop just described is inappropriate for a synchronous or clocked system. What is needed is a clocked flip-flop. In the direct set/reset flip-flop just described, the output changes as soon as an input is applied. In a clocked flip-flop, the input changes to indicate the future state of the flip-flop, but the flip-flop doesn't change until a clock, or rather a clock edge, occurs.

### J-K Flip-Flop

Figure 4-2 shows a clocked flip-flop called a J-K flip-flop. Logic family flip-flops also usually have direct set and direct clear inputs that work independently and take precedence over the clocked J and K inputs. They also take

precedence. Typical of TTL, a 74X family flip-flop has low-true direct set and clear. The inputs J and K when applied have no immediate effect on the flip-flop. Instead, they indicate what state the flip-flop should take when a clock input occurs.

Traditionally, for a J-K flip-flop, the clock is a low-going clock. That is, the clocking action takes place when the clock goes from a high voltage to a low voltage. Based on the current condition of the J and K inputs, the flip-flop is either set or reset a very short time after the clock edge occurs. The inputs J and K correspond to S and R with one exception. When J and K are both true, the flip-flop complements. That is, if it is a one, it becomes a zero and vice versa. When J and K are both zero. The flip-flop retains its current state. J alone sets the flip-flop. K alone resets the flip-flop. Figure 4-2 shows the flip-flop and a table showing the state transition based on the inputs and the current state when a clock occurs.

J	K	Q <sub>n</sub>	Q <sub>n+1</sub>
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	1	0

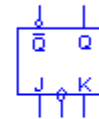


Figure 4-2 J-K Flip-flop

### D Flip-Flop

For making any kind of sequential logic, J-K flip-flops are hard to beat. However, for use in computer circuitry, a flip-flop is most often used for temporary storage. Here, needing two inputs to define the desired state is an inconvenience. This gave rise to the *Data* or D-type flip-flop. A D-flip-flop has a single bias input called D. When D is zero, the next high-going clock edge will lock a low into the flip-flop. If D is high, the next high-going clock edge will lock a one into the flip-flop. Note, that unlike a J-K, it is a high-going edge that clocks the data into the flip-flop. This is not a hard and fast rule, but traditionally, all D flip-flops have high-going clocks. Figure 4-3 shows a D-flip-flop with its transition table.

Note, that for both types of flip-flops discussed, what is on the inputs matters only the instant that the clock edge occurs. At other times the D, J and K inputs can be either high or low or moving rapidly back and forth. The input cannot change the instant it is clocked, however. The input must be stable for a minimum time before the clock edge. This time is  $t_{SU}$  or setup time. Also, the input should remain briefly after the clock edge occurs. This is called  $t_H$  or hold time. For all practical purposes, the hold time is usually zero. This means that if the input changes at the same time of the clock edge, it is too late to have any effect.

D	Q <sub>n</sub>	Q <sub>n+1</sub>
0	0	0
0	1	0
1	0	1
1	1	1

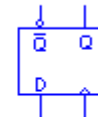


Figure 4-3 D Flip-flop

### Transparent Latch

There are some instances, admittedly rare, when it is desirable for the output of a flip-flop to change before the clock edge that locks in the data occurs. Here is one example. On some microprocessors, a data bus is used for multiple purposes, typically for data and to present a portion of the address for latching. At some time in a read or write cycle a signal is asserted called ALE for Address Latch Enable. Shortly afterwards addresses become available. Finally the ALE signal is negated and the address is latched into flip-flops so that the bus can be used for data purposes. It is highly desirable that the address be available as long as possible. The address is stable before ALE falls. To get the maximum time use of that address it would be nice if the latch could actually change state before the locking edge. That is how a transparent latch works. In lieu of a clock, a transparent latch has a gate or enable signal. The latch is transparent, that is the output is the same as the input when the gate is high. When the gate goes low, the data is locked in. The 74X373 is an example of a transparent latch.

### Toggle and T flip-flops

There is a reason why a JK flip-flop clocks on a falling edge. If J and K are both held high, a JK will toggle (change state) for every clock. The earliest counters were just toggled flip-flops. Suppose we toggle continuously a flip-flop which represents the lowest bit of a counter. The Q output of this flip-flop we will connect to the clock of the next flip-flop higher. The second flip-flop will toggle when the first goes from high to low. Thus we get the sequence: 00, 01, 10, 11, 00, etc. If we now take the output of the second flip-flop and use it to clock the third, it will toggle when the second flip-flop goes from high to low (which happens when the first goes from high to low). The

sequence is: 000, 001, 010, 011, 100, etc. A popular early flip-flop was a toggle flip-flop which toggled when ever the clock went low. There were no other inputs. In fact, the clock input was labeled 'T' for toggle. A toggle flip-flop is shown in Figure 4-4 a. (This practice of labeling the clock t carried on to J-K and D flip-flops.)

There is a flip-flop that has a single input called T and a clock. This is the T flip-flop. When T is a one, the flip-flop toggles when clocked. When zero, the state remains unchanged. This is precisely what a J-K flip-flop does when J and K are tied together and used as a single input. Some people call this a toggle flip-flop for obvious reasons, and that's where the T comes from, but it probably should be called a T flip-flop to distinguish it from the very early toggle flip-flop that had no input but a clock.

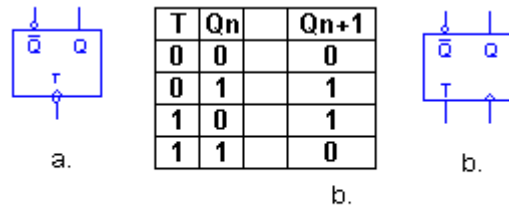


Figure 4-4 Toggle (a) and T (b) Flip-flops

T flip-flops are particularly attractive for making linear counters since a single AND term suffices to determine the T function. T is true when every lower flip-flop is a one.

### Other Flip-flops

We have discussed all of the flip-flops that are in current use. Digital design books in their discussion of flip-flops usually feel compelled to talk about master-slave flip-flops. This designation describes how they are made, not how they work externally. Since the advent of TTL about forty years ago, it hasn't been necessary for digital engineers to design their own flip-flops. While knowing what is inside may be interesting, it is not essential to designing with flip-flops. Essentially, a master-slave flip-flop is a structure wherein level inputs applied immediately affect the state of an input flip-flop and the clock transfers the contents of the input flip-flop to the output flip-flop. The problem with digital textbooks is that contain a lot of material that is interesting and historical but not particular relevant to current design. The authors feel compelled to include the same information that their predecessors did while not adding anything unique or original.

### Clocking Flip-flops

For clocked flip-flops, locking action takes place with one of the clock edges. If you examine the specs for an LS flip-flop like the 74LS74, you will observe that there is a spec for the rise time of the clock. If the clock rises too slowly it is possible that the flip-flop may not get clocked, or may get clocked twice! It is therefore incumbent of the designer to insure that any clocks have fast enough edges.

It turns out that CMOS renditions of popular flip-flops such as the 74HC74 do not have a clock edge spec. This is because clock edge speed is not a problem in CMOS. The clock apparently becomes effective as it passes a threshold, and it does not have to pass it at high speed. It appears that a relatively slow rising clock edge works perfectly well. On the other hand, regardless of any dependency on clock edge speed, the designer should always take care that clock edges be clean and free of noise. It is a good idea to make clock edges fast and clean even if the device doesn't seem to require it.

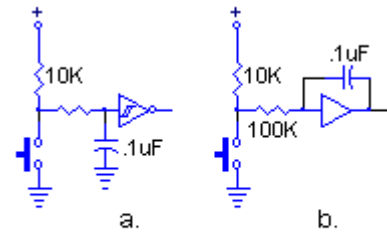
A particular problem is clocking a flip-flop manually with a pushbutton. Most all mechanical switches, whether momentary or not exhibit a phenomenon called "bounce". When the two contact surfaces come together, they can literally bounce and a sequence of make/break actions can take place within a very short time. If the contacts could be examined microscopically it would be seen that the contacts are not smooth but have peaks and valleys. As the contacts come together some contact is made, broken, and made again before making solid contact. Effectively, the switch goes on and off and on again in a very short time. If the switch is used to clock a flip-flop, then the flip-flop can be erroneously clocked more than once. Accordingly, it is good practice to *debounce* switches for which bounce could cause problems.

Figure 4-5 a. shows a traditional technique for debouncing a switch. The heart of the circuit is an R-C time delay. Initially, the capacitor is charged up via the 10K pull-up resistor. When the switch is closed, the cap begins to discharge toward ground through the 100K resistor. Since the voltage on the cap starts far from the gate logic threshold, any bouncing occurs away from the threshold and the voltage goes smoothly through the threshold. The

R-C is followed by a Schmitt trigger inverter so that the slowly falling (or rising) signal is sharply squared up. As mentioned already, it appears that there is no speed requirement on the clock on a CMOS flip-flop, but for safety sake, I'd include the Schmitt trigger. It should be noted that a Schmitt trigger can be made from any non-inverting

gate. For CMOS, it would consist of an input resistor of say 10K and a feedback resistor of 100K. The positive feedback from the non-inverting gate gives the snap action and creates the threshold hysteresis.

, but when the charging or discharging capacitor hits the logic threshold, the output pulls the input in the same direction. Since the voltage cannot change instantaneously on the capacitor, the input is pulled sharply well past the threshold. I've used this very reliable circuit for more than a couple of decades. The 4050 hex buffer of the 4000 family has been a favorite, but as I mentioned, any non-inverting gate will work, including octal bus buffers (74HC245, etc.) which are readily available and cheap.



**Figure 4-5 Circuits to debounce clocks for flip-flops**

For use with programmable logic, it makes no difference whatsoever if the debouncing circuit inverts the input signal or not since signals can be defined as high-true or low-true as desired.

## Registers

A collection of flip-flops that act together are called a register. Computers use registers to hold binary entities. Registers are nearly always D-type flip-flops with a single input and a single output. 74X00 family octal registers use 20-pin packages. With eight inputs, eight outputs, plus and ground and a clock, one pin is left. This pin may function as an output enable, a clear input or a clock enable, depending upon the part number. An octal register implemented in a CPLD can have all of those capabilities and more.

A special kind of register is one where the individual flip-flops are linked right to left, left to right, or both. This is called a shift register. A shift clock causes the pattern in the register to shift one bit position left or right. A CPLD can implement a shift register.

## PLD Flip-flops and Macrocells

For some time, PLDs (Programmable Logic Devices) have used what is called a macrocell. This consists of a large AND/OR array for implementing Boolean equations expressed in sum of products form and a flip-flop. The user can configure the macrocell in a number of ways. It can be configured to connect the combinational logic from the AND/OR array directly to the I/O pin associated with the macrocell, omitting the flip-flop altogether. Or it can use the array to control an input to the flip-flop. Whether or not the flip-flop is used, the output typically can be high-true, low-true or tri-state. Furthermore, the type of flip-flop may be programmable as well.

Some early PLDs employed macrocells that could be D, J-K or R-S flip-flops. (The RS I could not understand since an R-S is a flawed subset of a J-K). A J-K flip-flop, however, needs two logic functions to control it, one for the J input and one for the K. That means two AND/OR arrays for the flip-flop. This is not desirable or practical. So more modern PLDs tend to provide flip-flops that have only one logic input. Minimally this is a D flip-flop. Another type of single-input flip-flop is the T flip-flop, which is particularly suitable for linear counters. The macrocells in the Xilinx 9500 family of CPLDs may be either D or T flip-flops.

Any circuitry that would require more than two J-K flip-flop logic chips is probably implement cheaper with a PLD. Accordingly, with increased use of PLDs, J-K flip-flops will become mainly of historical interest. I am a fan of using J-K flip-flops, and minimizing their importance hurts a little. But design should be ruled by reliability and cost, and not nostalgia.