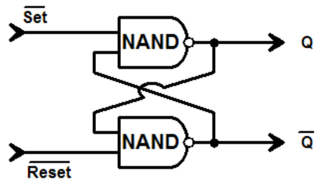
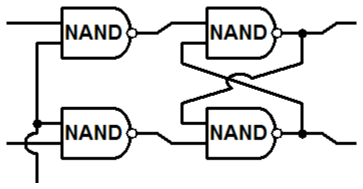
Lecture #24 – April 7, 2004

More about Flip-Flops

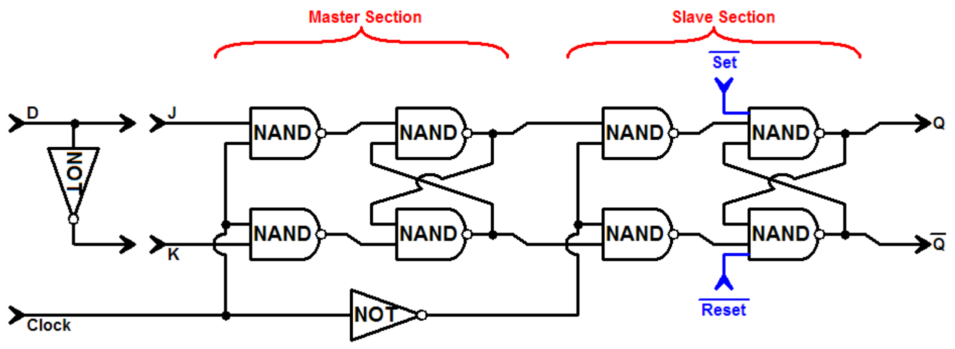
From the previous lecture we use the NAND-gate based flip-flop as a starting point. This form, shown below, is called a ***set-reset*** flip-flop. Normally, the inputs are at their resting state where both have the value 1. Bringing the top input line to 0 forces the top output to 1 (and the bottom output to 0), thus *setting* the flip-flop. Bringing the bottom input line to 0 forces the bottom output to 1 (and the top output to 0), thus *resetting* the flip-flop. The inputs are labeled Set and Reset, with a bar over top to indicate that they are active-when-0 instead of active-when-1. The outputs are traditionally called Q and Q-bar (or Not-Q), where Q-bar is a Q with a line over top. Under normal circumstances the outputs Q and Q-bar always have opposite values. (The one case where they do not is when both Set and Reset are brought to 0; both will have output values of 1, but this is an unstable condition. The flip-flop will settle into one of the two legal states depending on which input line goes to 1 first. If both input lines go from 0 to 1 simultaneously, minute differences in the manufacturing of the gates will cause one to be slightly faster than the other and the flip-flop generally will settle into its "preferred" state.)



By placing a NAND-gate in front of each input of the set-reset flip-flop, we can control whether or not new values get written into the flip-flop. The control line common to the new NAND-gates is normally 0, which forces their outputs to 1 (the resting state of the NAND-based set-reset flip-flop). When the control line is brought to 1, the new NAND-gates act like NOT-gates, and whichever of their data inputs is 1 forces the corresponding input of the flip-flop to 0, setting or resetting it appropriately.



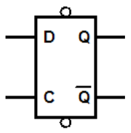
Now, if we take two of these new gated flip-flop modules and connect the outputs of the first to the inputs of the second, we have a composite structure called a ***master-slave flip-flop***. The first stage is the master, and its control line determines whether new values are written into its flip-flop. The second stage is the slave, and its control line determines whether the values from the master are copied into its flip-flop. By making the control lines of the two stages operate alternately, we can isolate the action of capturing an input value from the action of storing and displaying it. This is best done by connecting the two control lines with an ***inverter*** (another name for a NOT-gate), and calling the new control structure the ***clock*** line. When the clock is low (equal to 0), the input stage to the master is turned off, and no changes at the data input lines (now called J and K) affect the master flip-flop. Whatever happens to be in the master flip-flop is copied to the slave flip-flop. When the clock goes high (equal to 1), the slave locks in its current value, and the master reads its new value from the input lines. Bringing the clock back down to 0 locks in the last value from the input lines into the master and copies it to the slave. Since the J and K lines must contain opposite values, we can enforce this condition through an extra inverter and call the new input the D line (D for Data). Also, an extra set of inputs on the slave flip-flop allows us to set or reset the output as desired. This complete new structure is shown below.



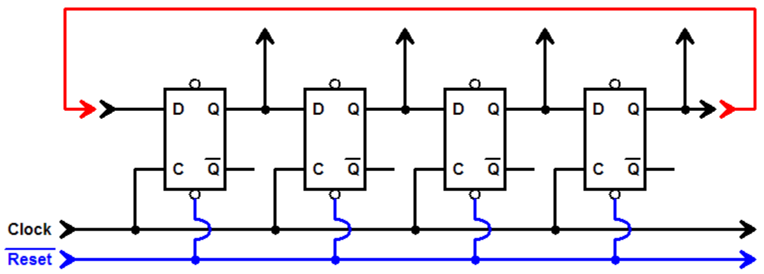
All the time that the clock is high, any changes to the D line are immediately written into the master flip-flop (but the slave is locked). This is called a ***level-sensitive*** or ***level-triggered*** flip-flop. This particular implementation is actually pretty bad. In this design there is a potential ***race condition***, which means that correct operation is dependent on timing, the speed of the clock pulses, and the ***propagation delay*** of the individual gates (how fast the gates operate, usually on the order of nanoseconds). For example, when the clock line goes low (from 1 to 0) the control NAND-gates in the master section *must* shut off *before* the control NAND-gates in the slave are activated, otherwise last (nano-)second changes in the data line might propagate all the way to the slave, causing a noisy output. The NOT-gate in the clock line *probably* introduces enough delay to insure proper operation.

A variation is one in which the act of bringing the clock line high (from 0 to 1) isolates the inputs from the master in a such a way that the isolation circuits themselves determine when it is safe to update the slave, and then do so immediately. Thus, the input value on the D line is copied safely to the Q output (and its complement to Q-bar) within a few nanoseconds of whenever the clock line is brought high. This variation is called an ***edge-triggered*** flip-flop, which is largely immune from race conditions.

From now on we will consider D-type flip-flops to be of the edge-triggered variety. In the symbol for a D flip-flop shown below the circle on top represents the Set input and the circle on the bottom represents the Reset input. Those inputs are shown as circles to indicate that they are ***active-low***, resting normally at 1 but going to 0 to perform their respective functions.

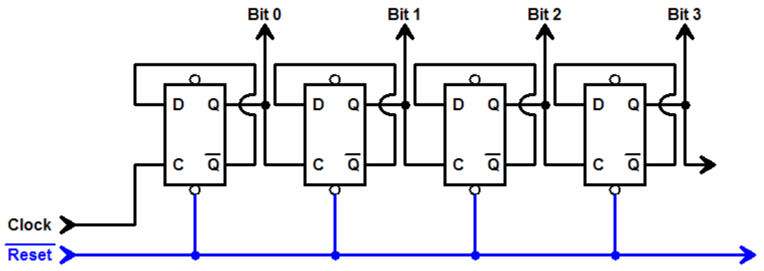


Now that we know how D flip-flops work we can consider circuits that make use of them. The simplest form is to simply connect a bunch of them together as shown below; where the Q output of one drives the D input of the next:

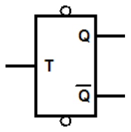


On each clock pulse, every flip-flop updates its value from the flip-flop to its left, all at the same time. Any value present at the D line of the leftmost flip-flop will move through the device, called a ***shift register***, at a rate of one flip-flop per clock pulse. Any value in the rightmost flip-flop is lost if no action is taken otherwise. By connecting the rightmost Q output back around to the leftmost D input, any values in the shift register will simply recirculate. For an N-bit shift register, the initial pattern is rotated back to its original position after every N clock pulses. Clocking in N–1 pulses is equivalent to a rotation in the opposite direction. Shift registers have numerous uses, including general purpose arithmetic registers, pseudorandom number generators, and many more.

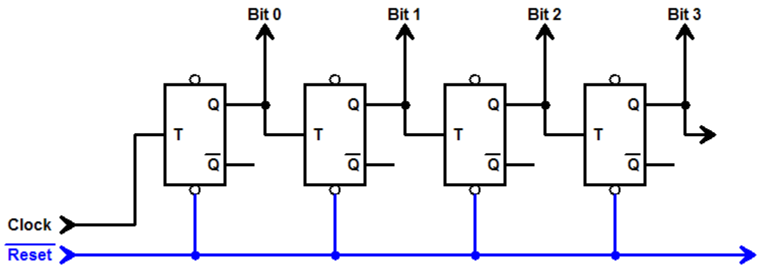
By connecting the Q-bar output of a flip-flop back to its own D input, the outputs *alternate* after every clock pulse. For example, if Q=1, then Q-bar=0 and the next clock pulse will copy the 0 into Q. If Q=0, then Q-bar=1 and the next clock pulse will copy the 1 into Q. Thus, the outputs go from 0 to 1 and back to 0 again at a rate exactly *half* of the input clock rate. Stringing a bunch of these flip-flops together so that the Q output of one drives the clock line of the next creates a ***divider chain***, where the first flip-flop's output is half the frequency of the clock, the second's output is half that rate or a quarter of the frequency of the clock, the third's output is one eighth of the clock, and so on. More importantly, looking at the binary outputs of the flip-flops in reverse order (the circuit is mirrored so that Bit 0 is on the right) reveals that the device is a ***binary counter***. It starts at zero, and then every clock pulse increases its value by 1. For N bits, the highest value (unsigned, of course) is 2^N-1. One more clock pulse brings the counter back to zero. This is essentially the heart of the ***program counter*** on a computer; when we add the extra hardware necessary to allow placement of any arbitrary value into the counter we can perform branch instructions.



The configuration just described, where the Q-bar output feeds back into the D input, is so common that it has its own symbol. This symbol, shown below, is called a ***triggered***, ***toggle***, or T flip-flop. The T input is just a relabeling of the clock input in the D flip-flop, but it indicates that a clock pulse on the T line will cause the outputs of the flip-flop to switch, or toggle, from one state to the opposite state.



The counter drawn earlier with D flip-flops is shown below using T flip-flops instead. You can see how much simpler the circuit appears in this form.



As we increase the complexity of our circuits it is necessary to increase our level of abstraction correspondingly. In future lectures we won't worry so much about the details of how many NAND-gates are in a D flip-flop, how a shift register works, or how T flip-flops toggle their values every clock pulse, but instead we will simply say "here is an N-bit shift register" or "assume an N-bit counter" and proceed from there. This is the only way we can manage devices of such complexity without suffering from information overload.