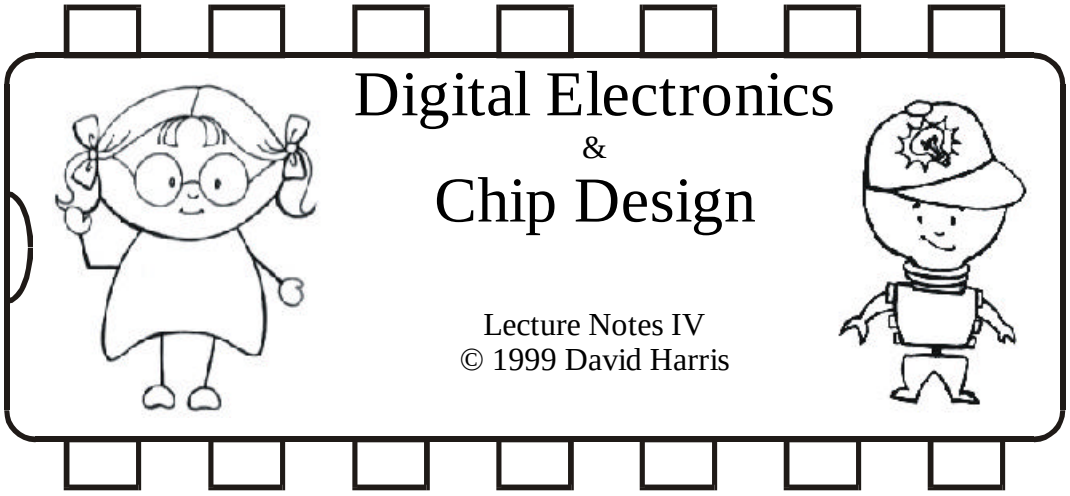


By now, we understand the workings of logic gates and can design circuits to compute functions of Boolean variables. However, we are still missing one very important capability: the ability for a circuit to "remember" information. Memory elements are crucial for most useful electronic systems. For example, if we were building a traffic light controller, we would need to have inputs detecting when traffic was waiting at the intersection, but we would also need to remember the current colors of the lights. As we take the use of memory to an extreme, we encounter modern computers which use millions and even billions of electronic, electromagnetic, and optical memory elements. In these notes, we will begin with the simplest circuit that we can build to display memory properties and will work our way up to more complex forms of flip-flops.

The R-S Latch

Consider the circuit in Figure 1. Unlike any circuit we have previously studied, it employs feedback; i.e., outputs of the circuit are connected back to inputs of the cross-coupled NOR gates. Before reading on, try to fill in the truth table below for this circuit:

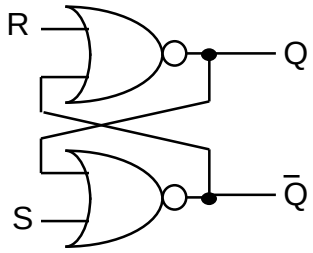


Figure 1: R-S Latch

R	S	Q	$\bar{Q}$
0	0		
0	1		
1	0		
1	1		

Truth Table

If you have never seen a circuit like this before, you probably had difficulty with the truth table. When R or S are 1, the two outputs are forced to particular values. But when both R and S are 0, what happens? In order to resolve this problem, we must introduce the concept of state. The output Q had some state before we examined the circuit; i.e., Q was either 0 or 1. Let us call this value X. When R and S are both 0, this circuit doesn't

change anything; the value at Q remains X. Thus, the circuit remembers this value X that was placed at the output.

We call this circuit an "R-S Latch." R stands for Reset; S stands for Set. Q is the output and $\overline{Q}$ is the complement of the output. When R is high, the output Q is reset to 0. When S is high, the output Q is set to 1. In either case, $\overline{Q}$ is the complement of Q. When both R and S are high, the circuit enters an unstable state where both Q and $\overline{Q}$ are low. As soon as either R or S falls low, the circuit enters one of the two stable states just described. Finally, when both R and S are low, the latch remembers the old value X last set or reset at Q. Thus, using X to indicate the old state of Q, we can fill in the truth table.

R	S	Q	$\overline{Q}$
0	0	X	$\overline{X}$
0	1	1	0
1	0	0	1
1	1	0	0

The Transparent Latch

The R-S latch provides the fundamental ability to remember an old state, but it is seldom used directly in a circuit because controlling R and S independently is frequently a bother. Instead, by increasing the complexity of the latch, we can make it easier to use. This is usually a good tradeoff; transistors are cheap and engineering effort is expensive. The first improvement we will examine is called the transparent latch, shown in Figure 2 below:

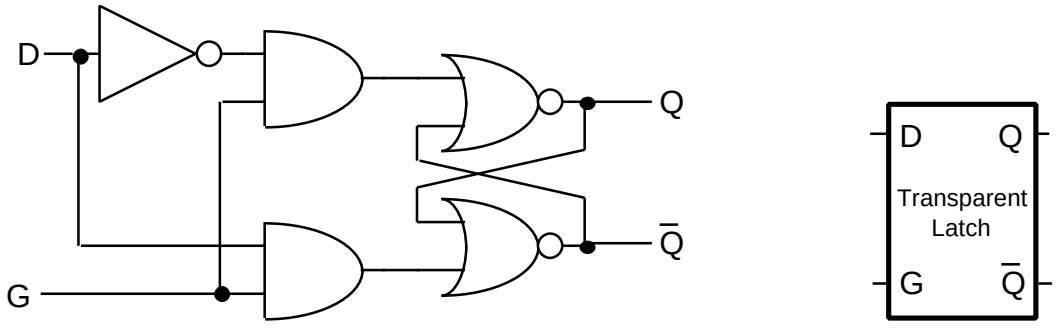
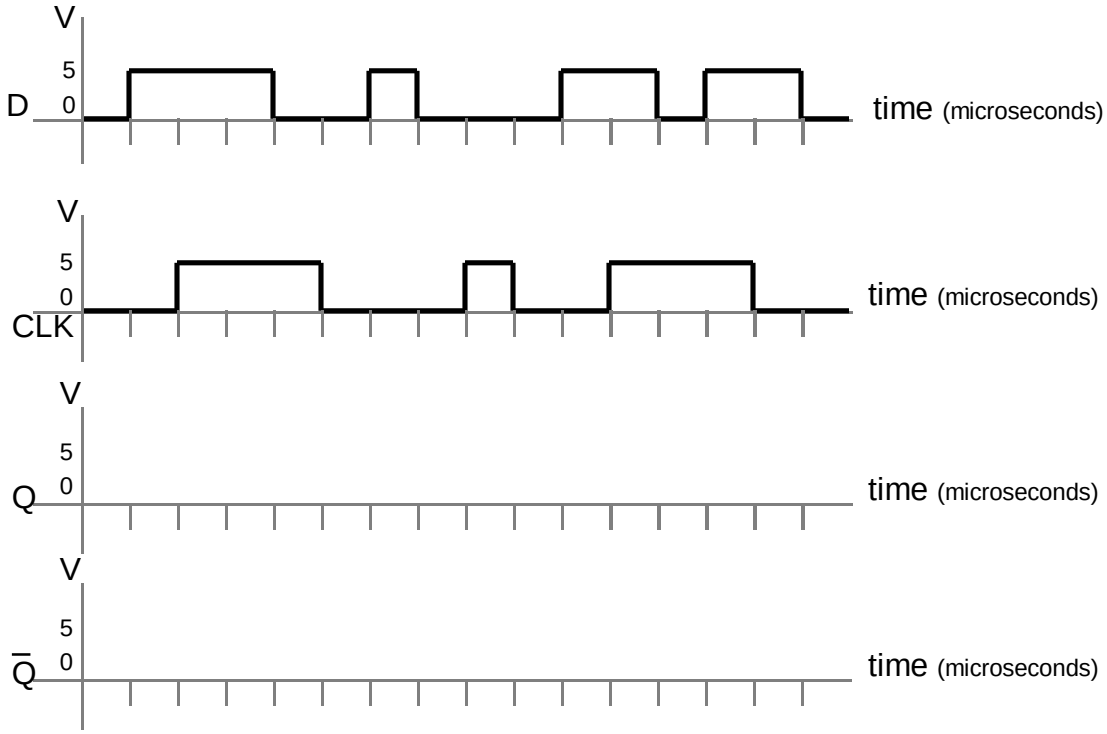


Figure 2: Transparent Latch: Circuit and logic symbol

Instead of explicitly drawing the gates, we usually represent the transparent latch with a box as shown. Let us figure out how the circuit works. When G is low, both AND gates produce 0. Thus, the circuit retains its old state. When G is high, there are two possibilities. If D is 1, a 1 is applied to the S input of the R-S latch, setting Q to 1. If D is 0, a 1 is applied to the R input of the R-S latch, clearing Q to 0. In other words, when G is high, Q gets the value at D. In no case do we have to deal with the unstable state of both R and S being 1.

We call D the data input and G the gate.[1] When the gate is low, the latch retains its old value. When the gate is high, the output follows the data input. In either case, $\overline{Q}$ is the complement of Q. This circuit is more useful because we can have a single bit of data to store and another circuit generating the gate signal indicating when to record the bit. To understand the circuit better, complete the timing diagram below:



The D Flip-flop

While the transparent latch is a great improvement over the R-S latch, there is yet another improvement we like to make on our memory elements, shown in Figure 3:

[1]Note that this is a different use of the word gate than in the phrase "logic gate." Here, the gate controls whether the data input can pass into the latch.

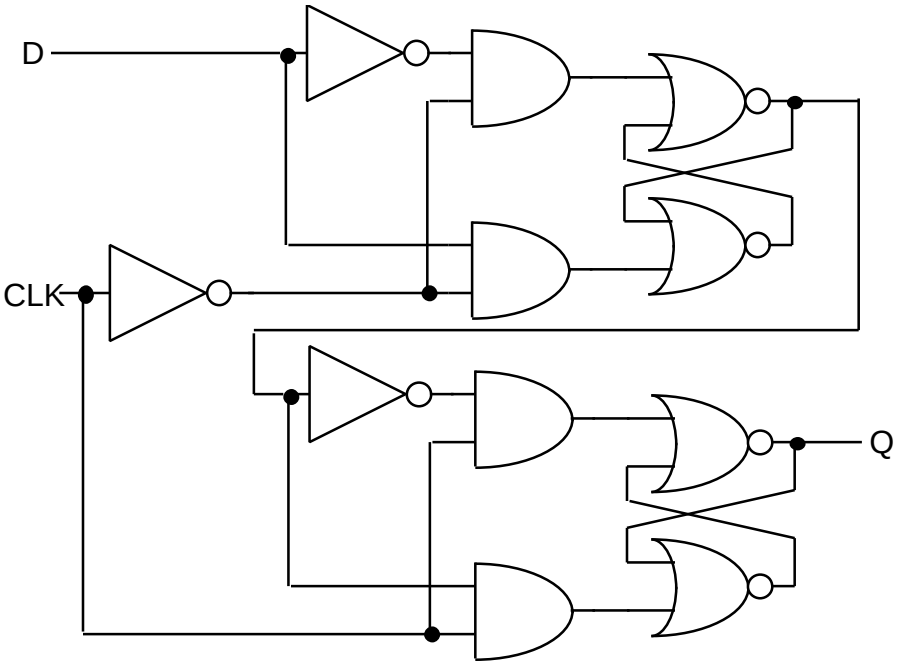


Figure 3: Flip-Flop

Looking closely, you may realize that this circuit is really two cascaded transparent latches, controlled by a clock input. A condensed representation and a schematic element for the D flip-flop are shown below:

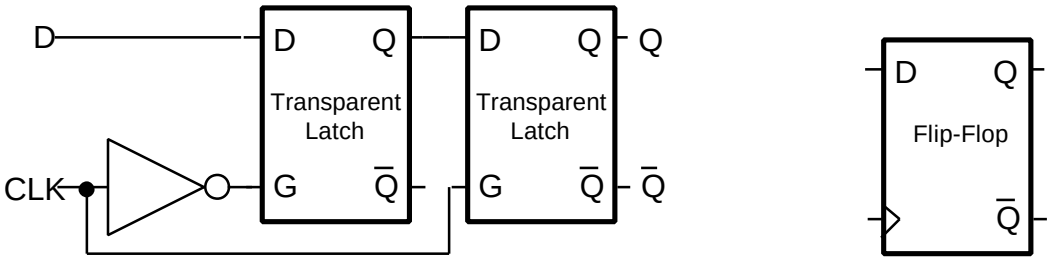


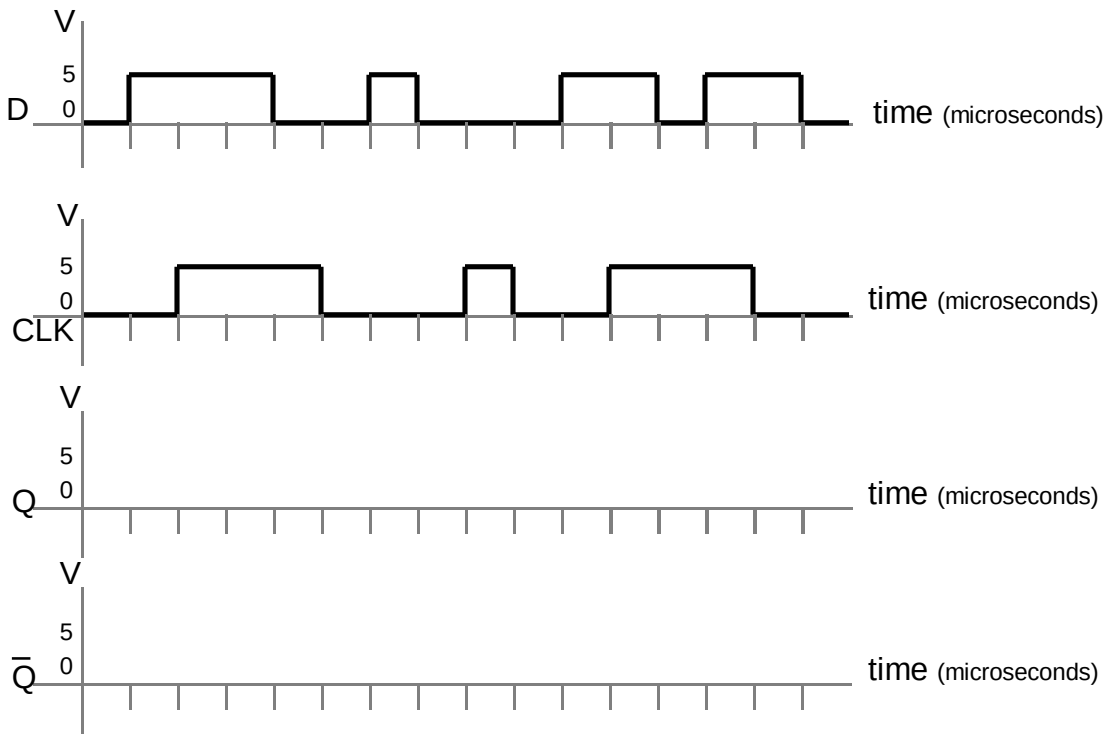
Figure 4: Alternate representations for a flip-flop

Note that the triangle is used to indicate a clock input. Clocks will be used frequently in later circuits; the triangle always indicates a clock.

Now, let us figure out how the D flip-flop operates. When the clock is low, the first transparent latch follows the value of D and the second retains its old state. When the clock is high, the first latch retains its old value from the time that the clock rose and the second latch gets the value of the first latch. In effect, we are isolating the output from the changing input. The input may change as it will, but at the moment the clock rises from low to high, the value that is at D gets latched and comes out at the output Q. As always, $\overline{Q}$ is the complement of Q.

With a D flip-flop, we don't have to worry about shaping our gate pulses to grab the value of D only when it is valid; instead, we design circuits to that the outputs are valid at the

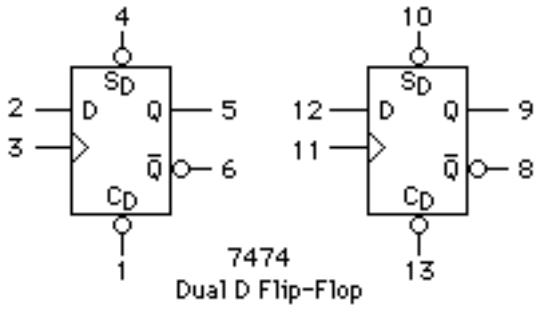
time of a rising clock edge and latch the input at that moment. To appreciate the difference between a transparent latch and a flip-flop, fill in the timing diagram below:



There are a few other species of flip-flops in common use including the J-K and T flip-flops. They are extensions of the basic R-S latch following the same principles we have developed with the D flip-flop.

The 7474 Dual Flip-flop

The 7474 chip contains two flip-flops with additional clear and preset functionality. Below is a diagram showing the pinouts:



The D, clock, Q and $\bar{Q}$ signals act as we have described earlier. The S_D and C_D set and clear, respectively, the output Q, independent of the clock. Thus, they allow the chip to act much like an R-S flip-flop when that is convenient. Note the inversion circles on those

set and clear inputs. They indicate "negative logic;" i.e. they are active when low, not high. Therefore, in normal operation of the 7474, connect the S_D and C_D inputs to 5 volts to prevent inadvertently setting or clearing the output.

Memory Elements

We frequently wish to store multiple bits of data. For instance, if we were building a circuit to perform division, we might have to store the divisor and the dividend, then perform repeated subtractions. Each of these quantities could be a multi-bit number. To do this, we build registers. A register is simply a collection of D flip-flops all controlled by the same clock.

Now that we have developed the essential ability to remember a bit, we can also understand how computer memories work. A typical static memory chip (SRAM) is built from flip-flops. Using 8 adjacent flip-flops, one can store a single byte of information. Larger memory arrays consist of thousands or millions of flip-flops with extra logic circuitry called decoders and selectors that look up one bit or byte of the information at a time.

Flip-flops, unfortunately, take up a fairly large amount of space on a chip, limiting the amount of memory that one can efficiently place on a chip. Therefore, engineers have developed dynamic memory (DRAM) chips. DRAMs work on the principle of storing charge on a tiny capacitor; their charge leaks away over a period of milliseconds, so they must be refreshed by special circuitry. However, the space required to store a single bit of information is so small that chips today store up to 64 million bits (64 Mbits) of information each. As each bit requires a transistor, your home PC with 128 megabytes of memory uses more than one billion transistors in its memory. Producing such large quantities of complex devices so cheaply is a very impressive feat of modern engineering.