# Gate Delay, and the S-R Flip-Flop

(Revised 12 April 2004)

## Representing Truth Values in Electronic Equipment

We have seen how numbers can be represented in a computer. These representations are imperfect models of the objects they represent. For instance, the data type "real" in a computer is only a finite subset of the set **R** of all real numbers. We now look at an entirely different representation of mathematical logic in a computer: representation of propositional logic in the electronic circuits of the computer itself.

In an electronic circuit, the truth values T and F are represented by ranges of voltages that may appear on a signal line or wire. The ranges chosen depend on the types of electronic components used. If the voltage used to represent T is larger (more positive) than the voltage for F, the circuit said to use positive logic. If the voltage used to represent F is larger, the circuit is said to use negative logic.

One very common convention using positive logic is:

T is represented by the range +2.4 volts to +5.0 volts

F is represented by the range 0.0 volts to +0.8 volts.

These ranges are known as TTL (transistor-transistor logic) levels and are shown in Figure 1. Circuits using this convention use a single 5-volt power supply, so voltages above 5 and below 0 volts are irrelevant. Note, however, that there is a 1.6-volt wide "buffer zone" between T and F. This is required because physical quantities such as voltage can never be measured perfectly. If there were no buffer zone and a voltage were close to the T-F boundary, it might be impossible to tell whether it represented T or F.

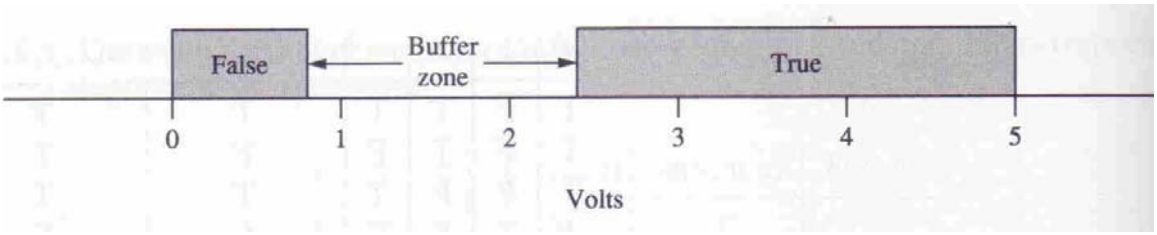


Figure 1. TTL Logic Levels

In circuit design, the truth values T and F are often called 1 and 0, or "high" and "low", the latter terms corresponding to the voltage levels representing the truth values.

## Gates and Gate Delay

A logical connective, such as "not", "and", or "or" is implemented by an electronic circuit called a **gate**. A gate that represents "not", and has one input and one output, is called an inverter.

Figure 2 shows the performance of a TTL inverter. If the input $p$ to the inverter is T, the output $q$ is F. If $p$ changes from T to F, the output $q$ must change from F to T. But since no physical quantity can change instantaneously, the output moves gradually out of the F range, through the buffer zone and into the T range. During this period, called the **gate delay**, the output of the gate is wrong or indeterminate. The circuit designer must be careful not to use the output of a gate until the gate delay has elapsed.

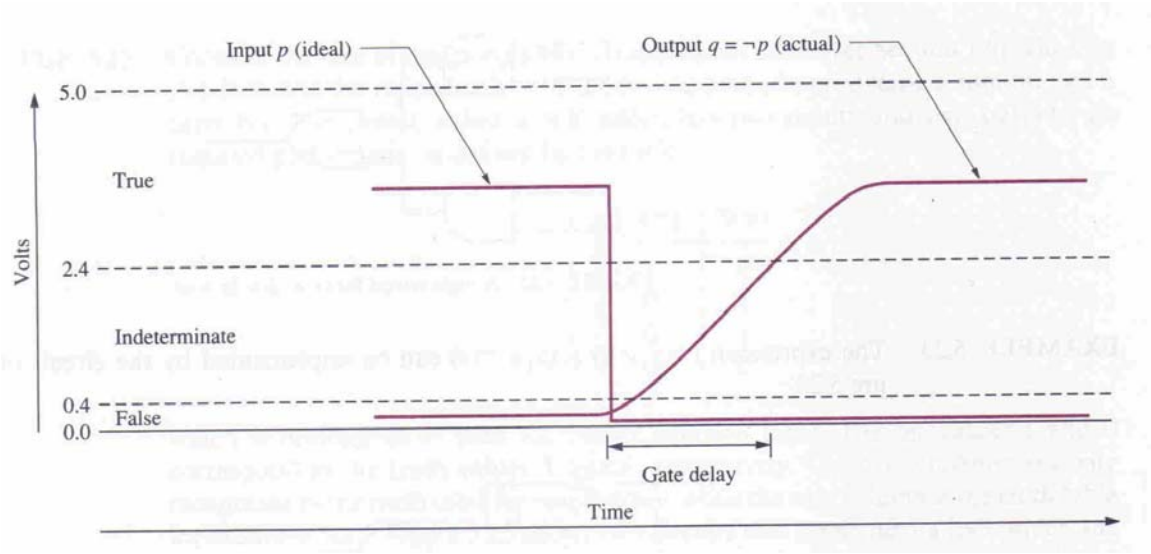


Figure 2. Actual Performance of an Inverter Circuit

The speed of a computer is limited by the gate delays in its circuits. In early vacuum tube computers, gate delays were measured in microseconds. Delays of 10 nanoseconds were achieved by the early 1970's. The fastest computers today [1991] have delays measured in picoseconds.

**Feedback and Memory Circuits**

Up to this point, we have interconnected gates only to form circuits for logical expressions. In the circuits we have drawn, the inputs are on the left and the output is on the right, and signal flow is always from left to right. To make more sophisticated circuits, we can use feedback: the output of a circuit is "fed back" into the input. In a circuit with feedback, there is a loop in the signal flow path.

The simplest useful circuit with feedback is the set-reset flip-flop (or S-R flip-flop) shown in Figure 3. In this circuit, the output of each gate is fed back to one input of the other gate. The loop in the signal path is a figure-8.

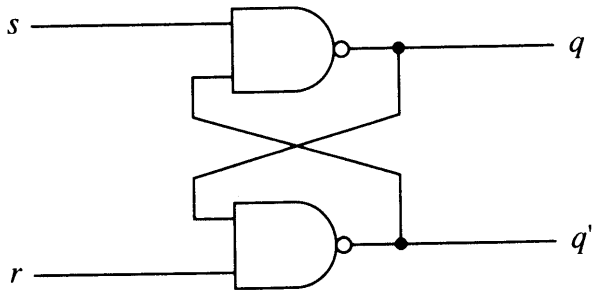


Figure 3. The S-R Flip-Flop

To analyze this circuit, we must first write some equations describing it. There is one equation for each gate. The equations are:

$$q = \neg(s \wedge q')$$
$$q' = \neg(r \wedge q)$$

We may look at this as a system of two equations in four unknowns. From our experience with algebraic equations, we may expect that there are several solutions. This is indeed the case. To find them, we can list all possible combinations of truth-values for *s, r, q,* and *q′*, and compute the truth-values of

$$q \Leftrightarrow \neg(s \wedge q')$$

and

$$q' \Leftrightarrow (r \wedge q)$$

for each combination. The computations are left to you as an exercise. The solutions are shown in the following table:

| *Case* | *s* | *r* | *q* | *q′* |
|---|---|---|---|---|
| (a) | T | T | T | F |
| (b) | T | T | F | T |
| (c) | F | T | T | F |
| (d) | T | F | F | T |
| (e) | F | F | T | T |

We can see from this table that if *s* or *r* is false (cases c, d, and e), the inputs *s* and *r* determine the outputs *q* and *q′*. But if *s* and *r* are both true (cases a and b), the circuit can be in either of two states. The circuit is stable in both of the states (a) and (b): the outputs will not change unless one of the inputs changes. Thus, when the inputs are both T, the circuit is a one-bit memory. It "remembers" what state it has been in.

The use of the flip-flop is illustrated in Figure 4. The figure plots the truth-values of *s, r, q*, and *q′* as functions of time. This type of diagram is called a timing diagram. At time $T_1$, the circuit is in state (b). It stays in this state until the input *s* changes to F.

When *s* changes to F, the output of the top gate in Figure 3 changes to T, which causes the output of the lower gate to change to F. The circuit changes to state (c) at time $T_2$. Now when the input *s* is returned to the T state, neither gate changes its output. The circuit changes to state (a) and remains in this state.

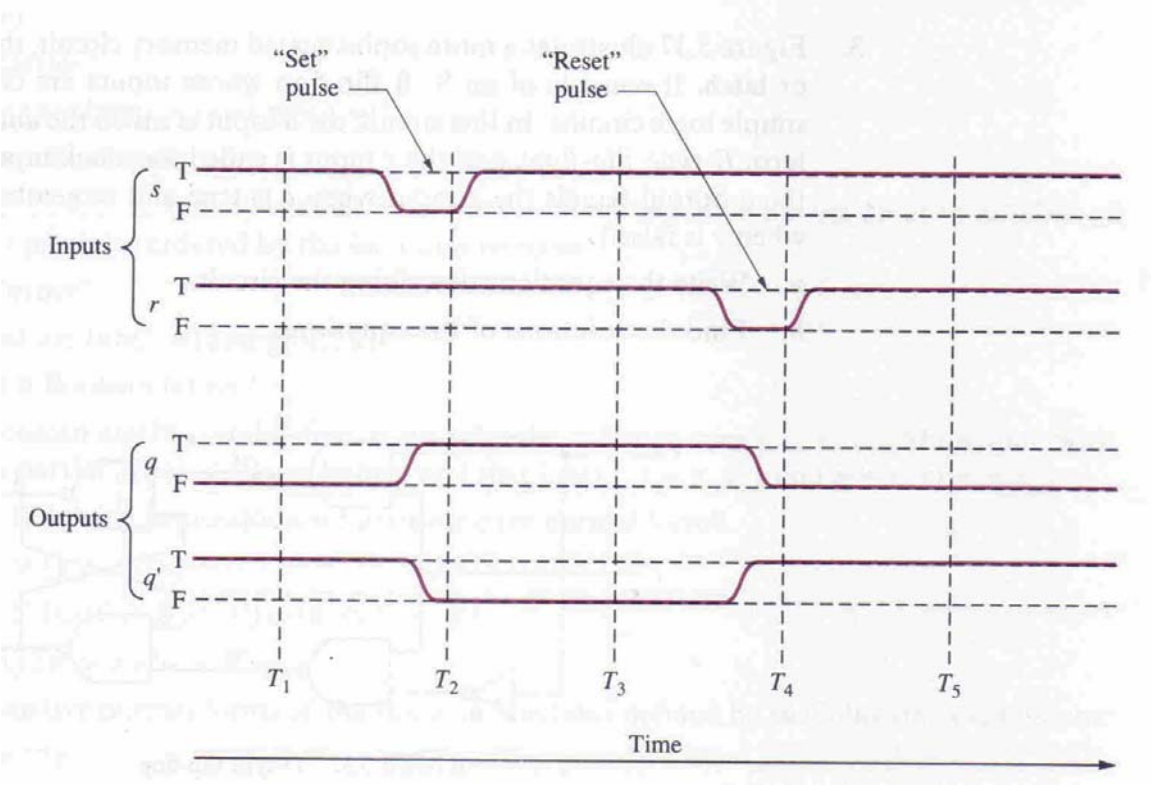


Figure 4. Setting and Resetting a Flip-Flop

The change in output state was caused by the **pulse** to F on the *s* input. A pulse is a short change in state of a signal line, after which the line returns to the previous state. The pulse on the *s* input is said to have **set** the *q* output to T. Similarly, a pulse on the *r* input will **reset** the *q* output to F. Throughout the sequence of events in Figure 4, the *q′* output is equal to the negation of the *q* output.

Notice that in the events of Figure 4, solution (e) of the equations plays no role. In fact, in state (e) it is not true that *q′* is the negation of *q*. This state is avoided in actual applications.

We have now seen how propositional logic contributes to the design of the two fundamental parts of a computer – the processor and the memory. The processor can be built of logic circuits, with the design based on formulas of Boolean algebra. Circuits with feedback can function as the computer's memory. Thus, it is possible in principle to build an entire computer out of NAND gates.