

### 3 Flip-Flops and Introductory Sequential Logic

We now turn to digital circuits which have states which change in time, usually according to an external clock. The *flip-flop* is an important element of such circuits. It has the interesting property of *memory*: It can be set to a state which is retained until explicitly reset.

#### 3.1 Simple Latches

The following 3 figures are equivalent representations of a simple circuit. In general these are called flip-flops. Specifically, these examples are called SR (“set-reset”) flip-flops, or SR latches.

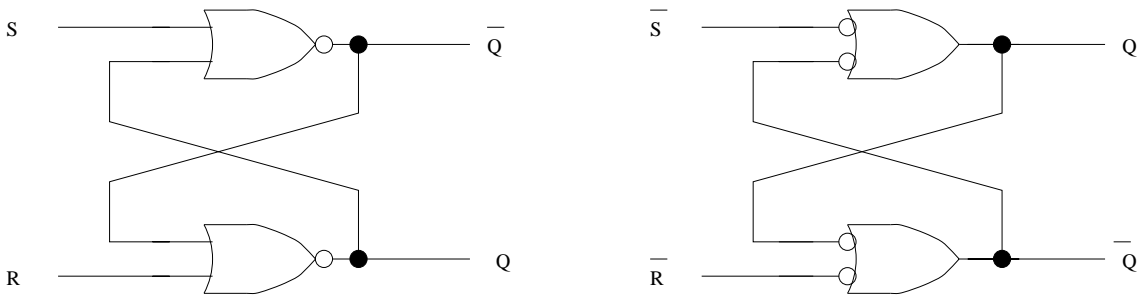


Figure 11: Two equivalent versions of an SR flip-flop (or “SR latch”).

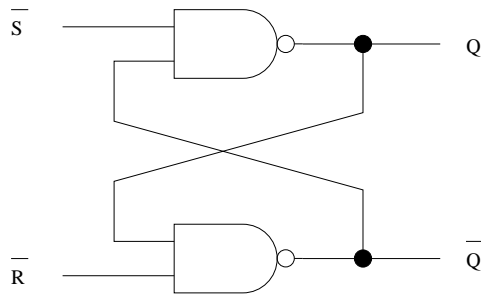


Figure 12: Yet another equivalent SR flip-flop, as used in Lab 3.

The truth table for the SR latch is given below.

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

The state described by the last row is clearly problematic, since  $Q$  and  $\bar{Q}$  should not be the same value. Thus, the  $S = R = 1$  inputs should be avoided.

From the truth table, we can develop a sequence such as the following:

1.  $R = 0, S = 1 \Rightarrow Q = 1$  (set)
2.  $R = 0, S = 0 \Rightarrow Q = 1$  ( $Q = 1$  state retained: “memory”)
3.  $R = 1, S = 0 \Rightarrow Q = 0$  (reset)
4.  $R = 0, S = 0 \Rightarrow Q = 0$  ( $Q = 0$  state retained)

In alternative language, the first operation “writes” a **true** state into one bit of memory. It can subsequently be “read” until it is erased by the reset operation of the third line.

### 3.1.1 Latch Example: Debounced Switch

A useful example of the simple SR flip-flop is the debounced switch, like the ones on the lab prototyping boards. The point is that any simple mechanical switch will bounce as it makes contact. Hence, an attempt to provide a simple transition from digital **HIGH** to **LOW** with a mechanical switch may result in an unintended series of transitions between the two states as the switch damps to its final position. So, for example, a digital counter connected to  $Q$  would count every bounce, rather than the single push of the button which was intended.

The debounced configuration and corresponding truth table are given below. When the switch is moved from  $A$  to  $B$ , for example, the output  $Q$  goes **LOW**. A bounce would result in  $A = B = 1$ , which is the “retain previous” state of the flip-flop. Hence, the bounces do not appear at the output  $Q$ .

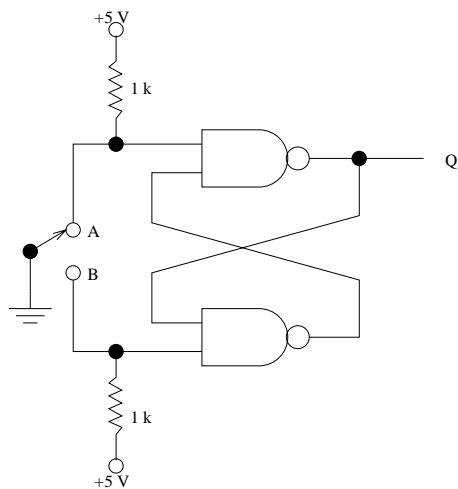


Figure 13: A debounced switch.

$A$	$B$	$Q$
1	0	0
0	1	1
1	1	retains previous
0	0	not allowed

### 3.2 Clocked Flip-flops

We will soon get used to the idea of a clock as an essential element of digital circuitry. When we speak of a clock signal, we mean a sequence of evenly spaced digital high and low signals proceeding at a fixed frequency. That is, the clock is a continuous sequence of square wave pulses. There are a number of reasons for the importance of the clock. Clearly it is essential for doing any kind of counting or timing operation. But, its most important role is in providing *synchronization* to the digital circuit. Each clock pulse may represent the transition to a new digital state of a so-called “state machine” (simple processor) we will soon encounter. Or a clock pulse may correspond to the movement of a bit of data from one location in memory to another. A digital circuit coordinates these various functions by the synchronization provided by a single clock signal which is shared throughout the circuit. A more sophisticated example of this concept is the clock of a computer, which we have come to associate with processing speed (*e.g.* 330 MHz for typical current generation commercial processors.)

We can include a clock signal to our simple SR flip-flop, as shown in Fig. 14. The truth table, given below, follows directly from our previous SR flip-flop, except now we include a label for the  $n^{\text{th}}$  clock pulse for the inputs and the output. This is because the inputs have no effect unless they coincide with a clock pulse. (Note that a specified clock pulse conventionally refers to a HIGH level.) As indicated in the truth table, the inputs  $S_n = R_n = 0$  represent the flip-flop memory state. Significantly, one notes that the interval *between* clock pulses also corresponds to the “retain previous state” of the flip-flop. Hence the information encoded by the one bit of flip-flop memory can only be modified in synchronization with the clock.

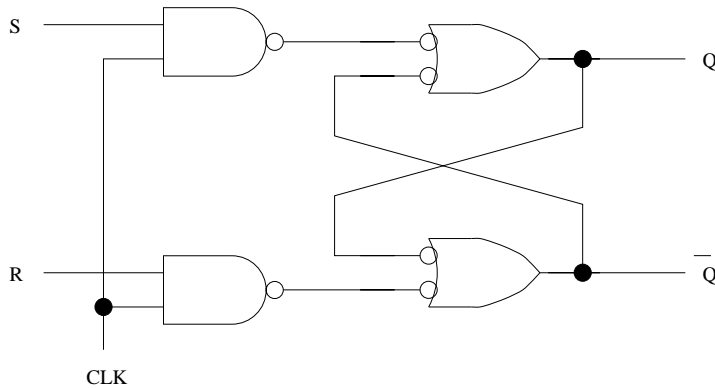


Figure 14: A clocked SR flip-flop.

$S_n$	$R_n$	$Q_n$
1	0	1
0	1	0
0	0	$Q_{n-1}$
1	1	avoid

We are now set to make a subtle transition for our next version of the clocked flip-flop. The flip-flop memory is being used to retain the state between clock pulses. In fact, the state set up by the  $S$  and  $R$  inputs can be represented by a single input we call “data”, or

$D$ . This is shown in Fig. 15. Note that we have explicitly eliminated the bad  $S = R = 1$  state with this configuration.

We can override this data input and clock synchronization scheme by including the “jam set” ( $\bar{S}$ ) and “jam reset” ( $\bar{R}$ ) inputs shown in Fig. 15. These function just as before with the unclocked SR flip-flop. Note that these “jam” inputs go by various names. So sometimes the set is called “preset” and reset is called “clear”, for example.

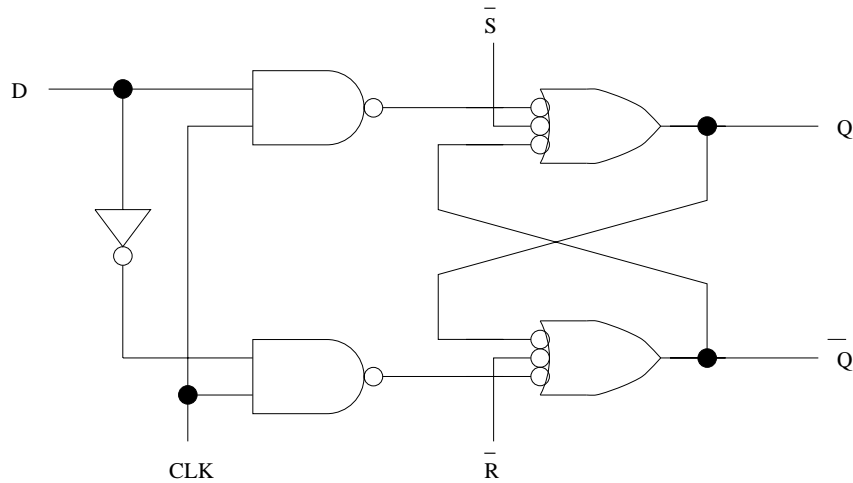


Figure 15: A “D-type transparent” flip-flop with jam set and reset.

A typical timing diagram for this flip-flop is given in Fig. 16. Note that the jam reset signal  $\bar{R}$  overrides any action of the data or clock inputs.

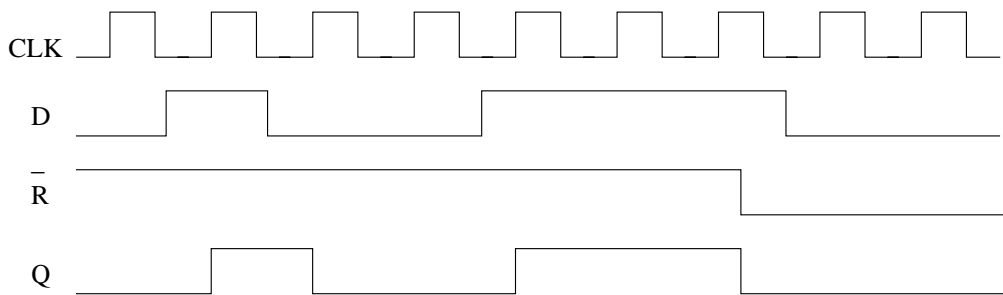


Figure 16: Example of timing diagram for the transparent D flip-flop. (It is assumed that  $\bar{S}$  is held HIGH throughout.)

### 3.2.1 Edge Triggered Flip-Flops

We need to make one final modification to our clocked flip-flop. Note that in the timing diagram of Fig. 16 that there is quite a bit of apparent ambiguity regarding exactly when the  $D$  input gets latched into  $Q$ . If a transition in  $D$  occurs sometime during a clock HIGH, for example, what will occur? The answer will depend upon the characteristics of the particular electronics being used. This lack of clarity is often unacceptable. As a point of terminology,

the clocked flip-flop of Fig. 15 is called a *transparent D-type* flip-flop or latch. (An example in TTL is the 7475 IC.)

The solution to this is the *edge-triggered* flip-flop. We will discuss how this works for one example in class. It is also discussed some in the text. Triggering on a clock rising or falling edge is similar in all respects to what we have discussed, except that it requires 2–3 coupled SR-type flip-flops, rather than just one clocked SR flip-flop. The most common type is the *positive-edge triggered D-type* flip-flop. This latches the  $D$  input upon the clock transition from LOW to HIGH. An example of this in TTL is the 7474 IC. It is also common to employ a *negative-edge triggered D-type* flip-flop, which latches the  $D$  input upon the clock transition from HIGH to LOW.

The symbols used for these three D-type flip-flops are depicted in Fig. 17. Note that the small triangle at the clock input depicts positive-edge triggering, and with an inversion symbol represents negative-edge triggered. The JK type of flip-flop is a slightlier fancier version of the D-type which we will discuss briefly later. Not shown in the figure are the jam set and reset inputs, which are typically included in the flip-flop IC packages. In timing diagrams, the clocks for edge-triggered devices are indicated by arrows, as shown in Fig. 18.

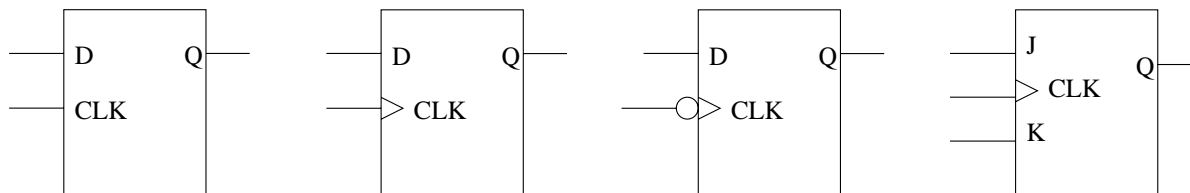


Figure 17: Symbols for D-type and JK flip-flops. Left to right: transparent D-type, positive-edge triggered D-type, negative-edge triggered D-type, and positive-edge triggered JK-type.



Figure 18: Clocks in timing diagrams for positive-edge triggered (left) and negative-edge triggered (right) devices.

For edge-triggered devices, the ambiguity regarding latch timing is reduced significantly. But at high clock frequency it will become an issue again. Typically, the requirements are as follows:

- The data input must be held for a time  $t_{\text{setup}}$  before the clock edge. Typically,  $t_{\text{setup}} \approx 20$  ns or less.
- For some ICs, the data must be held for a short time  $t_{\text{hold}}$  after the clock edge. Typically  $t_{\text{hold}} \approx 3$  ns, but is zero for most newer ICs.
- The output  $Q$  appears after a short propagation delay  $t_{\text{prop}}$  of the signal through the gates of the IC. Typically,  $t_{\text{prop}} \approx 10$  ns.

From these considerations we see that for clocks of frequency much less than  $\sim 1/(10\text{ns}) = 100$  MHz, these issues will be unimportant, and we can effectively consider the transitions to occur instantaneously in our timing diagrams.