Chapter 3: Flip-flops

Overview

In the previous labs, you gained familiarity with NAND and NOR gates. Those gates, in theory, act instantaneously so that the output always represents a logical result of the current inputs. However, in order to create a computer from digital logic, we must teach it to *remember*, so that its output will *not* change when the inputs change – unless we want it to. The first step in creating memory is the construction of a *flip-flop*. This is a device with an output that can be toggled between two states. There might be several combinations of inputs that can cause it to toggle, but there are many other combinations that it will simply ignore. Eventually, we will build data buses and memory banks from such simple elements. First, we will have a brief digression on digital inputs and outputs.

Inputs and Outputs

The simple logic gates we have used so far all have a similar numbering system. A quad NAND is a 74_00, where the blank represent a combination of letters that describe the type of internal circuitry in the IC. There are two common types: TTL (transistor-transistor logic) and CMOS (complementary metal oxide semiconductor). The TTL chips (LS ALS, AS, F, or no letters) typically have a narrow range of operating supply voltages (e.g. ±5% of nominal), require a significant input current (0.25 mA for LS type) to pull an input to the low state, and can provide a lot of output current. The CMOS chips (C, HC, HCT, AC, or ACT) accommodate a wider supply voltage range and require almost no input current. In your own designs, you will probably want to use high speed CMOS (HC) as your default choice. If you want to connect to existing TTL logic, then you must be somewhat careful, and you should probably use the HCT series.

DO NOT TIE OUTPUTS TOGETHER!

No matter what type of IC you use, you should *never* violate this simple rule. We will eventually find a way around this rule when we create data buses and output ports later in the semester. For now, just stick with the rule. The basic problem is that if one chip is trying to pull the output low while another chip is trying to pull the output high it acts like a short from high to low running through the chip. In that case, who knows what the output will actually be? The result will likely be smoke or at best flakey logic.

Inputs, on the other hand, can be tied together. Since an output can only supply so much current and every input takes in some input leakage, you can only drive so many inputs from a single output. The *fan-out* specifies how many inputs a given chip's output can drive. A fan-out of 10 is common but they vary significantly in the different logic series. If you want to drive a lot of inputs check the data sheet (with the correct letters) to be sure you are OK.

$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	1	Q_0	$\overline{Q}_0$
0	1	1	0
1	0	0	1
0	0	1	1

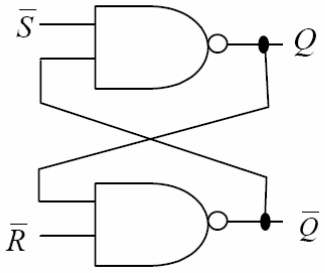


Figure 3-1: Truth table and diagram for an SR latch.

D	C	Q	$\overline{Q}$
X	0	Q_0	$\overline{Q}_0$
1	1	1	0
0	1	0	1

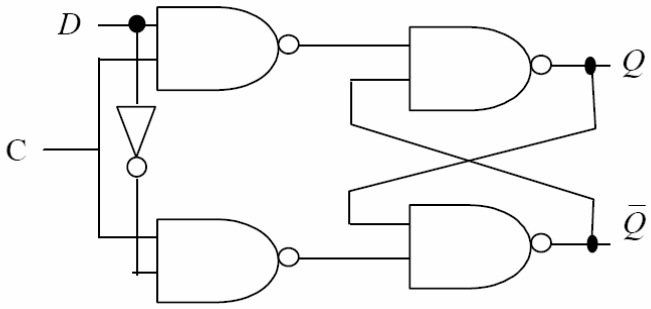


Figure 3-2: Truth table and diagram for a clocked D-type latch.

SR Latch

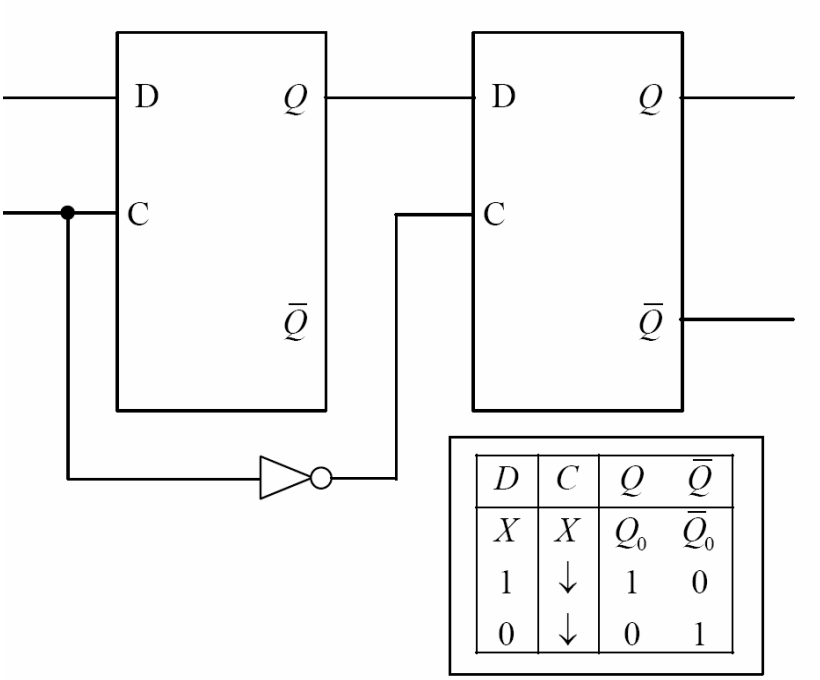
The easiest way to introduce memory into a digital circuit is by including *feedback*. The SR (for Set/Reset) latch is a simple device that can be made from just two NAND gates. It has two outputs. In normal operation these two outputs have opposite states and are called Q and $\overline{Q}$. The design uses the outputs as part of the input as shown in Figure 3-1. If both inputs are high, then the output states will be self-consistent as either $Q, \overline{Q}$ = 0,1 or 1,0. So, the "normal" state of the inputs to this latch is $\overline{S} = \overline{R} = 1$. As mentioned in Chapter 2, a bar over an input indicates that this variable causes an action when it is 0. This type of use of negative true input is also called an *active low* input.

If either $\overline{S}$ or $\overline{R}$ is brought low, then the output of the corresponding gate will go high and stay there even when its input returns back to high. You can use this simple latch as a button debouncer since it only reacts once to the input's transition from high to low. Even if you make multiple low transitions on *S*, for example, *Q* will go high once and stay there (unless you take $\overline{R}$ low).

Clocked Latches

We can introduce a more sophisticated form of memory into our latch by forcing the SR inputs to be in the high position unless a high level on another input line (the *clock*) tells them to read data line. As shown in Figure 3-2, all we must do is use the output of two NAND gates to drive the $\overline{R}$ and $\overline{S}$ inputs of the RS flip-flop. Note that the output only becomes the value of the *D* when the clock is high. We can insure that the $\overline{R}$ and $\overline{S}$ inputs never see an illegal double-low condition by using only one input, *D* (for *data*),

which is inverted and also sent to the other input NAND gate. This way, the input data only matters when the clock is high. If the clock is high then the output will follow the input. When the clock goes low it will remember the last input state until the clock goes high again.



D	C	Q	$\bar{Q}$
X	X	Q_0	$\bar{Q}_0$
1	↓	1	0
0	↓	0	1

Figure 3-3: Truth table and block diagram for a master/slave flip-flop

Master/Slave Flip-Flop

Our final step in creating a full-featured flip-flop is to put two of these clocked latches together, one behind the other as shown in Figure 3-3. The output of one latch (master) will drive the input lines the second (slave) latch. The data input to the master comes from a single line. This line is automatically inverted to drive the *R* input of the master. Finally, the clock line drives the clock input of the master, and is then inverted to drive the clock input of the slave.

This flip-flop works as follows: When the clock line is high, the master transfers the *D* input to its outputs, but the slave stays locked at its current value. Then, when the clock line goes low, the master stays set to the last *D* value *before the clock line went low*, and the slave output follows the (locked) output of the master. Thus, the final output is the same as the input *just before* the clock went low.

This is called an edge triggered *D*-type flip-flop and is available prepackaged as a 74_74 chip. As packaged, it also has *S* and *R* inputs so the state can be set to a specific value.

Note how the standard diagram for these more complicated digital circuits is simply a box with the inputs on one side and the outputs on the other. This will become more common as the semester progresses.

Design Exercise 3-1: Layout the circuit required to build a Master/Slave edge-trigger flip-flop using only NAND gates and inverters.

Design Exercise 3-2: Create an FPGA project in Quartus II for a 2-input 8-bit multiplier with Verilog HDL code, which does not produce any glitches. The synchronous circuit should trigger on the falling edge of your clock signal. Simulate the circuit: Do the glitches go away?

Use the Technology Map Viewer to produce a block diagram of the schematic for the circuit to be programmed into the FPGA.