Chapter 15L

Lab 15/D2: Flip Flops

Contents

15L.1A primitive flip-flop: SR Latch ................................................. 2
15L.2D Type ............................................................................. 2
15L.2.1 Basic operations: Saving a Level; Reset ........................................ 2
15L.2.2 Toggle Connection: Version I: ALWAYS Change or “Divide-by-two” ........................................................................... 3
15L.2.3 Toggle Connection: Version II: Change WHEN TOLD TO or “T Flop” ........................................................................ 4
15L.3 Counters: Ripple and Synchronous ............................................. 5
15L.3.1 Ripple Counter ...................................................................... 5
15L.3.2 Synchronous Counter ........................................................... 6
15L.4 Switch Bounce, and Three Debouncers ............................................. 6
15L.4.1 Watching Switch Bounce (Optional: for scope enthusiasts) ............... 7
15L.4.2 Eliminating Switch Bounce: Two Methods .................................. 7
15L.5 Shift Register ........................................................................... 9
15L.5.1 One Flop: Synchronizer ........................................................ 9
15L.5.2 Several Flops: Delay ................................................................ 9
15L.5.3 Several Flops Plus Some Gates: Double-Barreled One-Shot .............. 10
15L.1 A primitive flip-flop: SR Latch

This circuit, the most fundamental of flip-flop or memory circuits, can be built with either NAND’s or NOR’s. We will build the NAND form. It is called an “SR” latch because it can be “Set” or “Reset.” In the NAND form it also is called a “cross-coupled NAND latch.”

Build this latch, and record its operation. Note, particularly, which input combination defines the “memory state;” and make sure you understand why the state is so called.

*Leave this circuit set up.* We will use it shortly.

Practical Flip-Flops (30 min.)

It turns out that the simple latch is very rarely used in circuit design. A more complicated version, the *clocked* flip-flop is much easier to work with.

15L.2 D Type

The simplest of the clocked flip-flop types, the *D*, simply saves at its output (Q) what it saw at its input (D) just before the last clocking edge. The particular D flop used below, the 74HC74, responds to a *rising* edge.

The D flop is the workhorse of the flop stable. You will use it 100 times for each time you use the fancier J-K (a device you may have read about, but which we are keeping out of the labs, on the ground that it is almost obsolete). Perhaps you will never use a J-K.

15L.2.1 Basic operations: Saving a Level; Reset

The D’s performance is not flashy, and at first will be hard to admire. But try.
Feed the D input from a breadboard slide switch. Clock the flop with a “debounced” pushbutton, the buttons on the left side of the breadboard will do. Note that these switch terminals need pull-up resistors, since they have open-collector outputs. (This you saw last time; but perhaps you’ve forgotten.)

**Warning: a surprising hazard** The value of the resistor used as pullup on CLOCK turns out to matter. A large pullup value (say, \( \geq 10k \)) is likely to cause mischief that is explored in § 15L.2.2.1 on the following page.

Dis-assert \( \overline{\text{Reset}} \) and \( \overline{\text{Set}} \) (sometimes called \( \overline{\text{Clear}} \) and \( \overline{\text{Preset}} \)), by tying them high.

Note that the ’74 package includes two D flops. Don’t bother to tie the inputs of the unused flop high or low: that is good practice when you build a permanent circuit (averts possible intermediate logic state that can waste power, as you saw last time)—but would slow you down unnecessarily, as you breadboard circuits.

- Confirm that the D flop ignores information presented to its input (D, for “data”) until the flop is clocked.
- Try asserting \( \overline{\text{Reset}} \). You can do this with a wire; bounce is harmless here. (Why?\(^2\)) What happens if you try to clock in a High at D while asserting \( \overline{\text{Reset}} \)?
- Try asserting \( \overline{\text{Set}} \) and \( \overline{\text{Reset}} \) at the same time (something you would never purposely do in a useful circuit). What happens? (Look at both outputs.) What determines what state the flop rests in after you release both?\(^3\) (Does the answer to that question provide a clue to why you would not want to assert both \( \overline{\text{Set}} \) and \( \overline{\text{Reset}} \) in a circuit?)

### 15L.2.2 Toggle Connection: Version I: ALWAYS Change or “Divide-by-two”

The feedback in the circuit shown below may trouble you at first glance. (Will the circuit oscillate?) The clock, however, makes this circuit easy to analyze.

In effect, the clock breaks the feedback path.

---

\(^2\)The first Low achieves the reset. Bounce—causing release and then reassertion of Reset* causes no further change in Q.

\(^3\)The result is unpredictable, because it depends on which of the two inputs—S* or R*—has the last word: that is, which crossed its threshold, rising from Low to High, last.
Build this circuit and try it.

- First, clock the circuit manually.

### 15L.2.2.1 Looking for Trouble?

Here’s a side excursion for the adventurous.

We said “clock the circuit manually,” and assume that you would carry out this step by using one of the PB503’s debounced pushbuttons—probably the NC, so that you get a clock on pressing the pushbutton:

![Figure 4: Breadboard's debounced switches: these require a pullup resistor](image)

You need to choose a value for the pullup resistor. At first glance, the value doesn’t seem to matter, so long as it’s not so tiny that it overloads the transistor switch. In fact, however, the pullup value does matter.

To help you see how the flop is behaving, display its Q on one of the breadboard’s LED’s. Now, try $R_{\text{pullup}} = 1k$. This should work well: the flop should toggle each time you press the button.

Now try $R_{\text{pullup}} = 1M$. This should work badly: you should see the Q sometimes toggle—and at other times appear not to respond at all.

This occurs because of the slow rising edge provided by the over-large $R_{\text{pullup}}$. This big R, driving stray capacitance, produces a slowly-rising edge that an edge-triggered device finds troublesome. The problem is exactly the one that you saw causing weird instability in the LM311 comparator, back in Lab Op Amps 3 (remember the “Taj Mahal by moonlight”?).

You will find a scope image detailing the effect of such a slow edge in Classnotes D3, S16N.2.1. Use a scope to watch clock and Q, triggering on Q, and see if you can make out what goes wrong when the value of $R_{\text{pullup}}$ is too large.

Then restore the small $R_{\text{pullup}}$ of 1k, to clock your later circuits properly.

- Then clock it with a square wave from the function generator (the breadboard generator is less good than an external generator, with its higher $f_{\text{max}}$). Watch Clock and Q on the scope. What is the relation between $f_{\text{clock}}$ and $f_{Q}$? (Now you know why this humble circuit is sometimes called by the fancy name “divide-by-two.”)

- Crank up the clock rate to the function generator’s maximum, and measure the flop’s propagation delay. In order to do this, you will have to consider what voltages In and Out to use, as you measure the time elapsed. You can settle that by asking yourself just what it is that is “propagating.” If the answer to this question is that it is “a change of logic level” that propagates, then what is the appropriate voltage at which to measure propagation delay?\(^4\)

### 15L.2.3 Toggle Connection: Version II: Change WHEN TOLD TO or “T Flop”

A more useful toggle circuit uses an input to determine whether the flop should change state on the next clock. This behavior is properly called “T” or “Toggle.” (The preceding circuit—which toggles always—is not called a “T” type; the best name for it is probably “divide-by-two.”)

\(^4\)Since it is logic-level change that is propagating, we should measure time the voltage that typically defines a level change. For 5-volt CMOS that is 2.5V.
Here’s how the circuit should behave:

| T | Q_{new} 
|---|---
| 0 | Q_n (holds) 
| 1 | Q_n (toggles) 

![Figure 5: T flop behavior](image)

To exercise your “T” flop, clock it from the function generator, control it with a manual switch, and watch clock and Q on a scope.

Keep this “T” flop set up; we will use it again.

15L.3 Counters: Ripple and Synchronous

15L.3.1 Ripple Counter

If you wire a flop to toggle on EVERY clock (the easiest way to do this is with a D flop), and cascade two such flops, so that the Q of one flop drives the clock of the next, you form a “divide by four” circuit. One flop changes on every clock; the other changes on every-other clock. Evidently, you could extend this scheme so as to form a divide-by-a-lot. Today, we won’t go farther than divide-by-four.

Wire a second D-biting-its-tail flop; cascade the two flops to form just such a “divide-by-four” ripple counter. Show what the circuit looks like:

Divide-by-four ripple counter: made from D’s (your design)

- Watch the counter’s outputs on two LEDs while clocking the circuit at a few Hertz. Does it “divide by four?” If not, either your circuit or your understanding of this phrase is faulty. Fix whichever one needs fixing.

This counter behaves oddly, in one respect: it counts down. An easy way to make it count up (or are we only making it appear to count up?) is just to watch the Q outputs rather than the Q’s. (Ripple counters usually are built with falling-edge clocks, instead; then the Q’s do count up.)

- Now clock the counter as fast as you can, and watch Clock and first Q_0 then Q_1 on the scope. Trigger on Q_1.

- Watch the two Q’s together and see if you can spot the “rippling” effect that gives the circuit its name: a lag between changes at Q_0 and Q_1. (If you are using an analog scope, you will have to sweep the scope about as fast as it will go, while clocking the counter fast so as to make the display acceptably bright.)

---

You don’t need the fancier “T” behavior, to build a ripple counter. You will need it to make a synchronous counter.
15L.3.2  Synchronous Counter

The timing diagram below shows how a *synchronous* divide-by-four counter ought to behave.

![Timing Diagram showing behavior wanted from synchronous divide-by-four counter](image)

As you know, the diagram is the same as for a ripple counter—except for one important point: in the synchronous design, *all flops change at the same time* (at least, to a tolerance of a few nanoseconds). Achieving this small change in behavior requires a thorough redesign of the counter.

You will need a “T” flop in order to make a synchronous counter. Then, the key is to find, on the timing diagram, the pattern of pre-clock conditions that determine whether $Q_1$ ought to change (“toggle”). To permit synchronous behavior, we must look at conditions *before* the clock edge (during “setup time”); no fair to say something like “let $Q_1$ change if $Q_0$ falls;” that scheme would carry us back to the ugly “ripple” behavior).

Once you have discovered the pattern, drive $T_0$ appropriately, and try your design. Again, *keep this counter set up.*

Synchronous divide-by-four counter (*your design*)

See if you can use the scope to confirm that the *ripple* delay now is gone: watch $Q_0$ and $Q_1$, again. Synchronous counters are standard, ripple counters rare. In many applications, the ripple counters slowness in getting to a valid state, and its production of transient false states are unacceptable.

15L.4  Switch Bounce, and Three DeboUNCers

Here is a storage scope\(^6\) photograph showing a pushbutton bouncing its way from a high level to low:

---

\(^6\)Incidentally, this “storage scope” was an ordinary scope fed by a microcomputer of the kind you will build later in this course. The computer took samples during the bouncing process, stored them in memory, then played them back repeatedly to give a stable display. You will have a chance to try this, if you like, during the final lab sessions.
To see the harmful effect of switch bounce, clock your divide-by-four counter with a (bouncy-) ordinary switch such as a microswitch pushbutton. Watch the counter’s outputs on two LEDs. The bouncing of the switch is hard to see on an analog scope (easy on a digital), but its effects should be obvious in the erratic behavior of the counter.

![Figure 7: Microswitch bouncing from high to low (pulled up through 100k)](image)

15L.4.1 Watching Switch Bounce (Optional: for scope enthusiasts)

Switch bounce is hard to see on an analog scope, because it does not happen periodically and because the bounces in any event do not occur at exactly repeatable points after the switch is pushed. (On a digital scope, on the other hand, freezing an image of recorded switch bounce is dead easy.)

You can see the bounce, at least dimly, however, if you trigger the scope in Normal mode with a sweep rate of about 0.1 ms/cm. You will need some patience, and some fine adjustments of trigger level. Some switches bounce only feebly. We suggest a nice snap-action switch like the microswitch type.

15L.4.2 Eliminating Switch Bounce: Two Methods

15L.4.2.1 Cross-coupled NANDs as debouncer

Return to the first and simplest flip-flop (which we hope you saved), the cross-coupled NAND latch, also called an “R-S” flop. As input use the bouncy pushbutton. Ground the switch’s common terminal, and make sure to include pullup resistors on both flop inputs (they should be there, still).

Why does the latch—a circuit designed to “remember”—work as a debouncer? It does because bounce makes the inputs revert to their memory state, in which they seemingly hold the state into which they were pushed when the particular input earlier went low.

---

7It does because bounce makes the inputs revert to their memory state, in which they seemingly hold the state into which they were pushed when the particular input earlier went low.
2- CMOS buffer as debouncer (positive feedback)

Here’s a much simpler way to debounce, when you use a double-throw switch (SPDT) like the one used in the preceding exercise: just use a non-inverting CMOS gate, with its output talking to its input:

![CMOS AND gate as switch debouncer](image)

This one is no fun to watch—because no bounce will be visible, even at the input. Once you see why this is so, you will have understood why this circuit works as a debouncer. What happens, particularly, during the time when the switch input to the AND is connected to neither +5 nor ground?

3- D-Flop as debouncer; discovering duration of switch bounce

A plain D flop can debounce, if clocked appropriately. Wire up this circuit to test the notion:

![D flop as debouncer](image)

Fig. 11 suggests using one flop of a 4-flop '175 because you are about to use this part in another circuit.

Start with a high-frequency square wave driving the D-flop clock (say, 100kHz or more). You should see evidence of bounce in the counter’s misbehavior. Now lower the clock rate to 100Hz. Bounce should cease to trouble the counter. Raise the clock frequency till evidence of bounce reappears. Note the clock period (you’ll get the best information by watching the clock on the scope, of course; you’ll get a ball-park frequency from the dial on the function generator). That clock period reveals how long your switch is bouncing.
15L.5  Shift Register

(This is an important device—but some people will run out of time, around this point. Don’t worry if this happens to you. The circuit is not hard to understand, even if you don’t get a chance to build it. The digital one-shot of 15L.5.3 on the following page, based on the shift-register, is used in a later circuit, a “capacitance meter.” This circuit is described in Lab D3 and also appears among suggested digital projects. But you can build all of this later if you find you need the one-shot and don’t have time for it today. Some people will not build the C-meter and thus will not need the one-shot.)

**Note:** Please build the circuit below (a digitally-timed one-shot that evolves from the shift-register) on a private breadboard; you may use this circuit next time.

In fig. 12 we use a convenient structure that includes four flops with a common clock. Such a configuration is called a “register,” and here it is applied to the particular use as shift-register. The circuit of fig. 12 delays the signal called “IN,” and synchronizes it to the clock. Both effects can be useful. You will use this circuit in a few minutes as a one-shot—a circuit that generates a single pulse in response to a “Trigger” input (here, the signal called “IN”).

Clock the circuit with a logic signal from an *external* function generator; use the breadboard’s oscillator to provide “IN.” Let $f_{\text{clock}}$ be at least $10 \times f_{\text{IN}}$.

15L.5.1 One Flop: Synchronizer

- Use the scope to watch IN, and $Q_0$ (Q of the leftmost flop); trigger the scope on IN.
- What accounts for the jitter in signal $Q_0$?
- Now trigger on clock, instead. Who’s jittery now?
- Which signal is it more reasonable to call jittery or unstable? (Assume that the flops are clocked with a system clock: a signal that times many devices, not just these 4 flip-flops.)

15L.5.2 Several Flops: Delay

- Now watch a later output—$Q_1$, $Q_2$, or $Q_3$, along with IN. (We’ll leave the triggering to you, this time.)
- Note the effect of altering $f_{\text{clock}}$.

---

8This circuit does not work reliably with TTL; it’s a CMOS special.
9Since we treat the clock as the reference timebase, when we trigger on clock we notice the uncertainty of the input waveforms timing. It makes sense to say that the input is “jittery.”
15L.5.3 Several Flops Plus Some Gates: Double-Barreled One-Shot

A little logic—used to detect particular states of the shift-register—can produce pulses of fixed duration in response to an input pulse of arbitrary length (except that the pulse must last long enough to be sure of getting “seen” on a clock edge: in other words, the input pulse must last longer than a clock period). Such a circuit is called a “one-shot,” and the input signal (to maintain the metaphor\textsuperscript{10}) is called the “trigger.”

Below, we show a pair of output pulses we would like you to produce. If you fill in the shift-register waveforms, you will discover the logic you need in order to produce those pulses.

Figure 13: Timing diagram for digitally-timed one-shot

Incidentally, this all-digital circuit—whose output pulse is timed by the clock—is not what people usually mean when they say “one-shot;” the most common form uses an RC to time the pulse width. Such a one-shot is sometimes more convenient, but lacks the great virtue of synchrony with the rest of the digital circuit.

Please draw your design:

Digitally-timed (synchronous) one-shot (double-barreled): your design

Checkout

- Slow-motion: first use a manual switch to drive Trig, and set the clock rate to a few Hertz. Watch the one-shot outputs on two of the breadboard’s buffered LEDs. Take Trig low for a second or so, then high. You should see first one LED then the other flash low, in response to this low-to-high transition.

- Full-speed: when you are satisfied that the circuit works, drive Trig with a square wave from one function generator (the breadboard’s) while clocking the device with an external function generator (at a higher rate).

How would you summarize the strengths and weaknesses of this one-shot relative to the more usual RC one-shot?\textsuperscript{11}

\textsuperscript{10}As you may have gathered already, engineers have at least some of the characteristics of good poets: they keep the images simple and vivid. They call the pedant’s “monostable multivibrator” a “one-shot;” they call the pedant’s “bistable multivibrator” a “flip-flop,” as you know. They call the active-pullup output stage of a TTL gate “totem pole,” because that’s what it looks like. The synchronizing color-burst in a TV signal sits on the waveform’s “back porch.” There’s lots of this vivid imagery in engineering. Can the language of the social sciences offer any comparable pleasures?

\textsuperscript{11}Well, in case you’re interested, here are our views: Strength: the digitally-timed one-shot’s great virtue is that its output is synchronous with the system clock, so its pulse begins and ends at a predictable time, shortly after the clock edge, and safely away from $t_{\text{setup}}$. The weakness is only the greater complexity of the circuit, relative to a traditional RC-timed one-shot, and the circuit’s latency: the output can be delayed as much as one full clock cycle from the rise of trig.
Index

74HC175 (lab), 9
74HC74 (lab), 2

counter
   ripple (lab), 5
   synchronous (lab), 6
counter (lab), 5–6

D flip-flop (lab), 2–3
debouncer (lab), 6–9

flip-flop
   D type (lab), 2–3
   debouncer (lab), 6–9
   divide-by-two (lab), 3
   one-shot, synchronous (lab), 9–10
   pathology: slow clock edge (lab), 4
   propagation delay (lab), 4
   shift-register (lab), 9
   SR (lab), 2
   T type (lab), 4

NAND latch (lab), 2

one-shot, synchronous (lab), 9–10

propagation delay (lab), 4

ripple counter (lab), 5

s-r latch (lab), 2
SET*, RESET* (lab), 3
shift-register (lab), 9
slow clock edge (lab), 4
SR flip-flop (lab), 2
synchronizer (lab), 9
synchronous counter (lab), 6

T flip-flop, 4