Clocked Logic

... Part 1: The

Don Lancaster
SYNERGETICS

Here it is. You software types and novices who have been looking for some good introductory material to logic elements have found the place! You say you've been looking for something on flip-flops and how they work? So have we, and it's for sure we couldn't have found a better writer to present it for you than Don Lancaster. In this first of three articles he's taken material from his upcoming book entitled CMOS Cookbook *(to be published by Howard Sams) and come up with some of the best material you'll find on the operation of the JK and D-Type flip-flops. Although he's discussing the operation of the CMOS 4027 Dual JK and the 4013 Dual D-Type, the fundamentals and operation will apply equally well to their TTL cousins, the 7473 and 7474.*

If this turns out to be an education for you, consider it a state-of-the-art one. You'll be doing your learning around a discussion of Complementary Metal-Oxide Semiconductor (CMOS) devices. With their low-power characteristics they're destined to become the next logic family to come on the scene and stir things up a little. — John.

In *clocked, synchronous* or *step-by-step* logic, the outputs of logic blocks don't change immediately after their inputs change. Instead, the logic block waits till a specific time set by a waveform on a *clock* input. Only then are output changes allowed. There are two essential steps to the clocked logic process. In the *setup* step, inputs decide what the logic block is going to do. In the *clocking* step, the logic block actually does what it was told to and provides an output.

There are lots of advantages to clocked logic. First and foremost is the orderliness of the process. Logic signals move one and only one stage at a time. This lets us move data from one block to another without unchecked races and domino effects, where a logic one or zero goes galloping several stages beyond where it was first headed. What's equally important is that a logic block's *outputs* can now determine or at least influence its own *next* output conditions without any preferential states or wild oscillations taking place.

Clocked logic also internalizes the variable processing delays from logic block to logic block. So long as the *slowest* block completes its internal operations before the next clock arrives, all outputs of all stages will be valid and predictable at the instant of clocking, so we automatically know when to look for valid data. As a side benefit, most modern clock logic is *edge sensitive*. This edge sensitivity eliminates any need for resistors and capacitors to determine a leading or a trailing edge of a logic signal.

Clocked logic is used in virtually all advanced electronic systems. This is particularly true if counting, shifting of data, or storage of characters is needed. In this article, we'll be looking first at a *do-it-yourself* clocked logic block, followed by a check into the 4013 Type D flip-flop and the 4027 JK flip-flop. These devices are extremely useful by themselves as the detailed applications catalog later in the article will show you. The same operating principles will be important as the basic building blocks when discussing the heavier counters and registers.

CMOS Clocked Logic

Most CMOS logic blocks are clocked on the *positive edge* of the clock. This is the ground to positive transition of the clock input. Clocking is defined in a positive logic sense for most CMOS devices.

There are a few exceptions to this positive clocking rule. Binary ripple counters such as

This article is excerpted from the *CMOS Cookbook*, copyright 1977 by Howard Sams. Reprinted by permission.

D type and JK flip-flops

the 4020, 4024, 4040, and 4060 are clocked on the negative edge or positive to ground transition of the clock. This lets you cascade binary stages for longer count lengths. A few CMOS counters give you a choice of clock polarity, set by a logic signal on a separate pin. The 4518 and 4520 dual decade and dual hexadecimal counters are the most important of these. They give you a choice of positive edge clocking for synchronous counting systems or negative edge clocking for cascaded ripple counting.

Except for a few easy-to-live-with setups and hold time limitations, *it is only the input conditions that exist at the instant of the clocking transition or edge that matter*. Inputs can change regardless of whether the clock is high or low, eliminating the *one swallowing* problems that plagued early TTL level clocked flip-flops.

There is one important clock restriction that remains with CMOS and applies to just about any logic restriction that remains with CMOS and applies to just about any logic family:

> In any clocked logic system, the clock must cycle only once, noiselessly and bounce-free, per intended output change.

This means that all our clocking signals must be clean. In particular, clocking commands that come from the outside world or from mechanical pushbuttons *must* be properly conditioned to give you one and only one clean transition per desired output change.

With CMOS, it also pays to keep the clock risetime as fast as possible. Five microseconds is a normal worst-case maximum clock transition time. If possible, make your clock signals have much faster risetimes than this. Slower risetimes may let one stage output a new state before the next stage has a chance to complete clocking. This mixes old and new data and generates garbage for you. In large CMOS systems, it pays to avoid *clock slew* problems by deriving all clock signals from the same source or from parallel sources with identical delay.

A Clocked Logic Block

One inherent feature of clocked logic is that it takes *two* regular flip-flops or storage devices to build one clocked one. One of these flip-flops takes care of accepting and setting up the input information, while the second actually carries out the intended operation and holds the result for us as an output.

In the dim distant past, one of these two storage elements consisted of a diode-capacitor memory or *steering network*. More recently, the stored charge in a base-emitter junction of a transistor served the same purpose, with presence or absence of stored charge representing a one or a zero. Today's CMOS, along with many other families, uses two distinct flip-flops — an input or *setup* storage device called the *master* and an *output* flip-flop called the *slave*. On the clocking edge, the contents of the previously setup master flip-flop is transferred to the slave. The slave flip-flop then provides us with the final output and between-clocking storage of output data.

Once again, the all important purpose of the two step process is to give us an orderly one-stage-at-a-time shift of data between clocked logic blocks and to let us count or binarily divide without getting into preferential state and hangup problems. Let's see what kind of trouble we can get into by trying to make an ordinary set-reset flip-flop binarily divide, alternating states on every input command:

In Fig. 1a is a NOR logic set-reset flip-flop. With both set and reset low, the flip-flop holds the last state it was put into, with Q and $\overline{Q}$ providing complementary outputs. A high on SET drives Q high and $\overline{Q}$ low, while a high on RESET does the opposite. Driving both SET and RESET high at the same time gives us a disallowed state, and the last input to go to ground decides the final result.

In Fig. 1b, we've converted this into sort of a clocked set-reset flip-flop. We do this by adding AND gates to the inputs, controlled by a new CLOCK line. When CLOCK is low, inputs are ignored. When CLOCK is high, inputs are accepted. We can now at least set up what the flip-flop is going to do while the clock is low and actually carry out the operation by briefly bringing the clock high.

So far, so good. This is a useful clocked logic block. We can obviously make it binarily divide by cross coupling $\overline{Q}$ to SET and Q to RESET (Fig. 1c). Now every time the clock goes high, the flip-flop will change state,

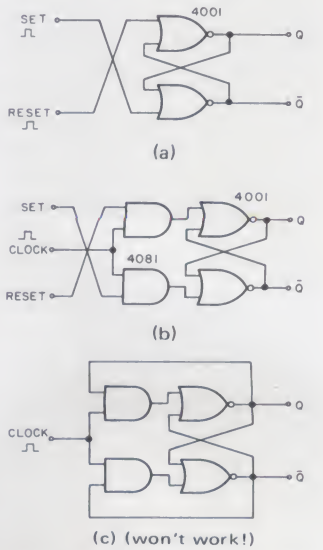

Fig. 1. Steps toward clocked logic flip-flops. (a) NOR logic set-reset flip-flop. (b) Adding AND gates gives clocking ability . . . clock input must go high to allow change of state. (c) An attempt at building a binary divider or counter that fails miserably.

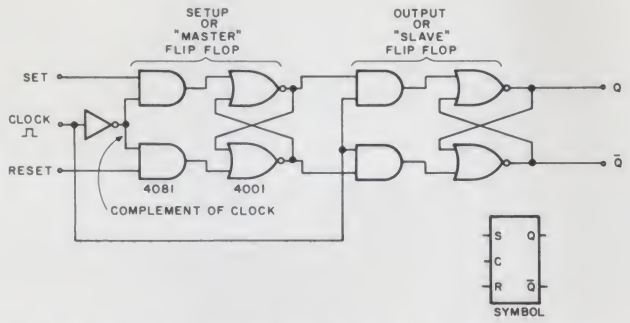

Fig. 3. Key to reliable clocked logic is the use of Master-Slave pairs of clocked flip-flops. Only one flip-flop is active at any time, eliminating unchecked races and preferential states.

since it was told to go to the opposite state. Right?

Well, not quite.

Sure enough, the instant the clock goes high, the outputs change state. But what if the clock *stays* high? These new output states reach around and change the input which changes the output which changes the input which . . . . What you really end up with is a complicated and unpredictable gated oscillator that runs while the clock is high and stops in one state or the other while the clock is low. Hardly what we had in mind.

We might try to beat the problem by picking just wide enough a clock pulse to let one and only one change take place. But this will be time, loading, device, temperature, and supply dependent. It will probably also depend on the price of yak butter futures. *The point is that there is now reliable way to let a single clocked flip-flop count or shift information*. That's why we have to use two separate storage elements or master-slave *pairs* of flip-flops for workable clocked logic.

An Alternate Action Pushbutton

Fig. 2 shows us an *alternate action* pushbutton that does work reliably. It changes its output state every time the button is pushed. At the same time, it provides free debouncing and contact conditioning.

While this circuit looks almost as simple as Fig. 1c, there is a crucial difference. Here we have two storage devices, a master capacitor and a slave flip-flop. The capacitor remembers what the new state is going to be. When the button is pressed, the capacitor voltage is transferred to the slave flip-flop. No race or oscillation is possible since the capacitor can't recharge much as long as the button is pressed, and after the button is released, no problem remains. This is a low frequency circuit ideally suited to manual button pressing.

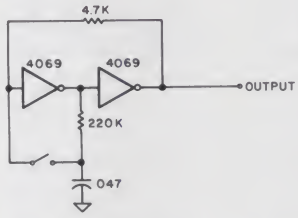

Fig. 2. Alternate action (Push on — Push off) bounceless pushbutton. Resistor and capacitor form temporary storage for "steering."

A Master-Slave Clocked Logic Block

Fig. 3 replaces the capacitor master with a conventional flip-flop. What we've done here is use two of the previous clocked NOR flip-flops. The first or master flip-flop accepts data only when the clock is low; the second or slave flip-flop only accepts data when the clock is high.

Now, when the clock is low, the master or input flip-flop can accept data and will remember the last input to go high. When the clock goes high, the input flip-flop is *disconnected* from the Set and Reset inputs and is no longer allowed to change state. But, with the clock high, the second or slave flip-flop is enabled and the contents of the master is transferred to the slave and appears as an output immediately after the clock goes high.

Even if we crosscoupled the outputs back to the inputs or cascaded stages, a wild race can't result because the next flip-flop down the line is not enabled at any particular instant.

Our circuit is said to clock on the positive edge since that's the time an output apparently appears. In reality, clocking is continuous, with the low clock state accepting data into the master and the high clock state transferring master to slave.

If we wanted a negative edge clocked flip-flop instead, we'd move the inverter so the first stage is active with clock high and the second with clock low. Note that with either system, inputs can change virtually at any time without *one swallowing* or similar problems.

We call this particular circuit a clocked RS flip-flop, and unlike our Fig. 1 circuits, its a genuinely useful building block without race or state problems. Just sitting there by itself, it can't binary divide and it still has disallowed input conditions when both Set and Reset are high, but we can fix these limitations.

It's an easy matter to convert the clocked RS flip-flop into the more useful and more common clocked logic blocks, as Fig. 4 shows us. These more common flip-flops are the type T flip-flop, the type D flip-flop, and the JK flip-flop.

The T in the T flip-flop of Fig. 4a stands for *Toggle*. By adding two external *feedback* leads from Q to reset and Q to Set, we tell the flip-flop to *change* state each time. This alternates states each clocking. Since the output changes state each positive clock transition, you only get half as many positive transitions in the output. This gives you a square wave of one half the input clocking frequency. The T flip-flop is not available separately as a CMOS package since it is easy to convert D and JK flip-flops into binary dividers. The 4024 is an example of seven cascaded T flip-flops that toggle on the negative clock edge.

A *Data* or *Delay* or Type D flip-flop is built by adding an inverter so that Reset is always the Complement of

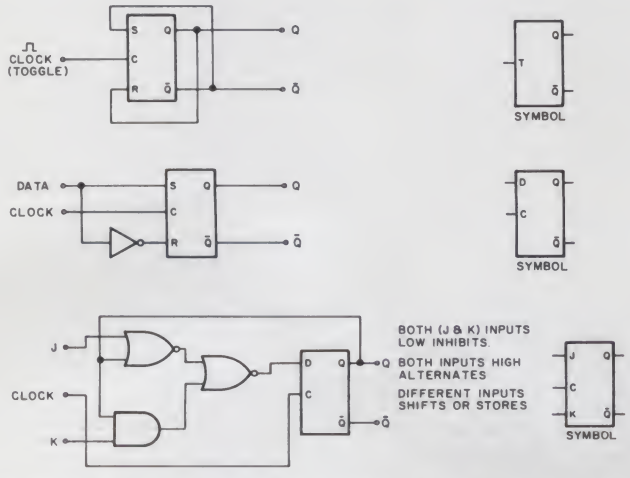

Fig. 4. Converting a clocked RS flip-flop into other clocked flip-flops.
(a) Type T flip-flop can only binary divide or alternate output states. (T represents toggle.)
(b) Type D flip-flop shifts or stores information. (D represents data or delay.)
(c) Type JK flip-flop shifts, stores, binary divides, or does nothing.

Set (Fig. 4b). A one on the D input gets stored in the flip-flop on the positive clock edge and appears at the Q output. A zero similarly applied gets clocked in and appears at the Q output. The type D flip-flop is useful in storing or delaying one bit of information. It is the key to the shift registers of the next article. We'll see that shift registers store data and move information on an orderly one-stage-at-a-time basis. We can convert a type D flip-flop to a type T flip-flop by externally feeding back the $\overline{Q}$ output to the D input.

The most versatile and universal clocked flip-flop is the JK flip-flop of Fig. 4c. The extra gates on the input make the JK flip-flop into a Type D flip-flop if the inputs are different. It makes the JK flip-flop into a Type T flip-flop if the inputs are both high. Finally, if both J and K are low, the *same* state gets reclocked back into the flip-flop making it *appear* to do nothing.

The JK flip-flop is then a universal one that can store data, binarily divide, or do nothing, all depending on the input conditions on the J and K inputs. There are no disallowed states or disallowed combinations of J and K logic. When all this versatility is needed, the JK flip-flop is the obvious choice to use, particularly for fancy or subtle timing sequences.

But the type D flip-flop is often in a shorter package, is slightly cheaper, uses somewhat less power, and often has a simpler and easier PC board layout. So, the D flip-flop is most often the best choice to use, and its a good policy to save the fancier JK versions only for those uses where you definitely need the do-nothing or inhibit option of both inputs low.

Direct Inputs

After we've gone to all the trouble of making our clocked logic block operate only when clocked and only when we want it to without any races or disallowed state conditions, we usually go back and add some new *direct* inputs that let us immediately set or reset the flip-flop into some state *independently* of the clocked inputs. We can use this to initialize a flip-flop into a certain state, to reset a group of counting flip-flops to zero, or to preset or *jam* a certain count or word into a register or latch.

These new inputs are called the Direct Set and Direct Reset inputs. Similar direct inputs on the fancier clocked logic blocks of the next article may be called Load, Preset, Reset, Clear, Jam, or some other name that suggests immediate operation independent of the clock.

> Note that all direct inputs to a clocked logic block must be disabled during clocked operation.

In CMOS, this usually means that any direct inputs are held *low* except when they are specifically used to setup, clear, or change the contents of the clocked logic blocks. Direct inputs usually dominate the clocked ones and are usually independent of the clock level or the conditions on the clocked inputs.

Generally, its a good rule to edge couple or pulse direct inputs when used — this keeps a steady direct high from hanging up your clocked logic system. When you use direct logic inputs, they always must be released before clocking.

Since the direct inputs behave as ordinary Set-Reset unclocked flip-flops, only one direct input should be used at a time. If you try using both direct inputs at once, you'll get a disallowed state condition. There is, of course, no reasonable way to let direct inputs shift or binary divide without problems — this is why we went to a clocked logic block in the first place.

The 4013 Dual D Flip-flop

With CMOS, we can use transmission gate techniques to greatly simplify the internal design of clocked logic blocks. Let's take a detailed look at the 4013 dual D flip-flop and the 4027 dual JK flip-flops and see how they work and how transmission gates simplify the logic for us.

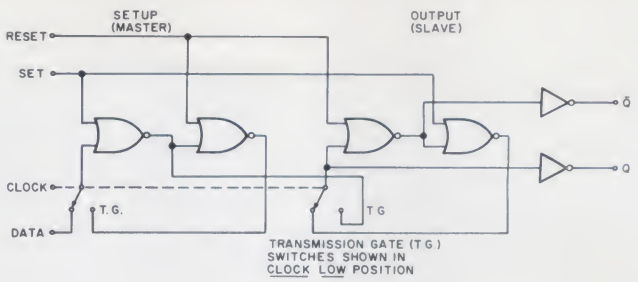

Fig. 5a. Logic diagram of half a 4013 Dual D flip-flop.

The logic diagram for half of a 4013 appears as Fig. 5a. While we could use AND gates for clocked logic with CMOS, the CMOS transmission gate set up as a SPDT switch greatly simplifies things for us.

Assume that the direct set and reset inputs are low. This reduces our master flip-flop to a pair of cross-coupleable inverters and does the same for the slave.

Assume further that our clock is low. The slave flip-flop is cross-coupled through its transmission gate switch, so it remembers a previous answer for us and outputs it via the buffered Q and $\overline{Q}$ outputs. These inverting buffers prevent outside loading from affecting the state or speed of operation. With the clock low, our master flip-flop is *not* cross-coupled. Instead it *follows* the data input. It will keep following the data input and remembering its instantaneous value so long as the clock is low.

As the clock suddenly goes high, the two SPDT transmission gate switches jump to the other side. This now cross-couples the master flip-flop, disconnects the master from the D input, and forces the master to remember the last value on the D input at the instant the clock went high. Since the D input goes nowhere when the clock is high, anything new to happen to the D input after the positive clock edge is ignored.

When the clock goes high, it also breaks the cross-coupling on the slave flip-flop, turning the slave into a pair of inverters that reflect the state of the master. Thus, with the clock high, the master is holding data for us and ignoring any new D inputs. The slave is simply passing on (without remembering) the master's contents directly to the outputs.

What happens when the clock goes back low? From the outside world, apparently nothing. The switches flip over to the other side. This cross-couples the slave output so it now remembers the data for us independently of what the master is up to. The master is now released and allowed to follow new input data. So, while a rather dramatic internal change takes place on the falling clock edge, no outputs can change, and things externally appear to stay as they were.

The clock rise time must be fast. Five microseconds is the usual limit. The clock must be conditioned and bounce free. A slow rise or fall time can cause switching problems where old and new data can get mixed. Note that the fall time is equally important as the rise time for proper operation. Both must be fast and clean. Note that this circuit is fully static. It can remain in the clock high or clock low states indefinitely.

We can summarize the rules for the 4013:

Both Direct inputs must be

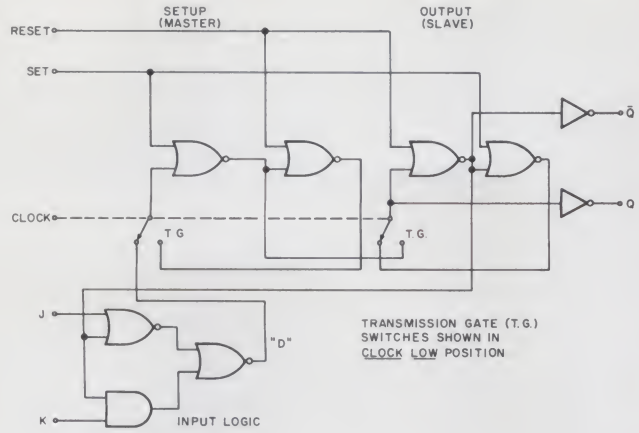

Fig. 6a. Logic diagram of half a 4027 Dual JK flip-flop.

low for normal clocked operation.

If the D input is high, the flip-flop goes or stays in the state with Q high and $\overline{Q}$ low on the positive edge of the clock.

If the D input is low, the flip-flop goes or stays in the state with Q low and $\overline{Q}$ high on the positive edge of the clock.

If the D input is cross-coupled to the $\overline{Q}$ output, the flip-flop changes to the other state on the positive edge of the clock, behaving as a binary divider.

If the Direct Set input is made high by itself, the flip-flop will immediately go or stay in the state with Q high and $\overline{Q}$ low.

If the Direct Reset input is made high by itself, the flip-flop will immediately go or stay in the state with Q low and $\overline{Q}$ high.

If the Direct Set and Direct Reset inputs are simultaneously made high, a disallowed state results with both Q and $\overline{Q}$ high, independently and dominantly over the clock and D inputs. This state is normally avoided. The last direct input to go low decides the final result.

Both direct inputs must be returned to ground before clocking can resume.

The clock must be bounceless and noise free with rise and fall times faster than five microseconds.

Fig. 5b summarizes these rules in a pair of truth tables.

The 4027 Dual JK Flip-flop

A JK flip-flop has two advantages over a type D flip-flop. We can make it binarily divide under external control and we can make it appear to do nothing (not change) despite repeated clockings. These extra performance features are obtained at the cost of a somewhat larger and more expensive IC that takes slightly more supply power in a usually more complex PC layout. The JK flip-flop is important where full performance is needed, such as in sequencers, odd-length walking ring counters, divide-by-three circuits, fully synchronous counters, and some other special uses.

The logic diagram of one half a 4027 is shown in Fig. 6a. It is the D flop circuit repeated with some funny gates added to the input. These gates respond to a J input, a K input, and an internal feedback line that monitors the *present* Q output. Since each flip-flop has one new input, we end up with a total of 16 pins, compared to the 14 of the dual 4013.

Suppose both J and K are low when we bring the clock from the low to the high state. What happens? The low K input disables the AND gate, holding its output low. The low J input is ignored by the NOR gate, and the present Q output is inverted twice and presented to point D. On clocking, the old state of the flip-flop gets *reentered.* To the outside world it looks like nothing happens at all. If J and K are both low, clock commands appear to be ignored.

What happens if J and K are both high? This will disable the NOR gate and enable the AND gate. The Q output gets inverted once and sent to D. Clocking will change the flip-flop to the other state. We alternate states or binarily divide when J and K are both high.

If J is high and K is low, the AND gate is disabled and a one unconditionally appears at D and is loaded. Similarly, if J is low and K is high, a zero unconditionally appears at point D. This zero results as a *don't care* condition. If Q is high, it goes through the AND gate, gets inverted once and ends up a zero. If Q is low, it goes through the NOR gate, gets inverted twice, but *still* ends up a zero. Either way, J low and K high loads a zero.

Our JK flip-flop acts like a type D flip-flop if the inputs are different. If both J and K are low, the circuit appears to ignore clock pulses. J and K high binarily divides.

We can summarize the rules for the 4027:

Both direct inputs must be low for normal clocked operation.

If J is low and K is low, no apparent output change takes place on the positive edge of the clock.

If J is high and K is low, the flip-flop goes or stays in the state with Q high and $\overline{Q}$ low on the positive edge of the clock.

If J is low and K is high, the flip-flop goes or stays in the state with Q low and $\overline{Q}$ high on the positive edge of the clock.

If J is high and K is high, the flip-flop changes output

CLOCKED INPUTS:

D	CLOCK	Q	$\overline{Q}$
0	↓	0	1
1	↓	1	0
$\overline{Q}$	↓	CHANGES	

DIRECT INPUTS:

R	S	Q	$\overline{Q}$
0	0	CLOCKED OPERATION	
0	1	1	0
1	0	0	1
1	1	1 (DISALLOWED)	1

(DIRECT INPUTS MUST BE LOW FOR CLOCKED OPERATION)

Fig. 5b. Truth tables for 4013.

CLOCKED INPUTS:

K	J	CLOCK	Q	$\overline{Q}$
0	0	↓	NO CHANGE	
0	1	↓	1	0
1	0	↓	0	1
1	1	↓	CHANGES	

DIRECT INPUTS:

R	S	Q	$\overline{Q}$
0	0	CLOCKED OPERATION	
0	1	1	0
1	0	0	1
1	1	1 (DISALLOWED)	1

Fig. 6b. Truth tables for 4027.

states, binarily dividing on the positive edge of the clock.

If the Direct Set input is made high by itself, the flip-flop will immediately go or stay in the state with Q high and $\overline{Q}$ low.

If the Direct Reset input is made high by itself, the flip-flop will immediately go or stay in the state with Q low and $\overline{Q}$ high.

If the Direct Set and Direct Reset imputs are simultaneously made high, a disallowed state results with both Q and $\overline{Q}$ high, independently and dominantly over the clock and D inputs. This state is normally avoided. The last direct input to go low decides the final result.

Both direct inputs must be returned to ground before clocking can resume.

The clock must be bounceless and noise free with rise and fall times faster than five microseconds.

Fig. 6b summarizes these rules in a pair of truth tables.

An easy way to remember the operation of the direct inputs is that if you do nothing to them (keep them low), they do nothing. On the D flip-flop, the D input gets passed across the flip-flop to the Q output on clocking. The same thing happens to the JK flip-flop with different J and K inputs. Do nothing to J and K (keep them low) and it does nothing. Do everything to J and K (both high), and you get a binary divider. ■