Don Lancaster
SYNERGETICS

Clocked Logic

... Part 2: Some basic applications

Last month Don Lancaster presented us with a good introduction to the world of flip-flops. His discussion this month covers devices such as basic counters, dividers, shift and storage registers, and multivibrators. In this second of a three part series he has taken material from his upcoming book entitled CMOS Cookbook *(to be published by Howard W. Sams). – John.*

Let's discuss some of the many different things we can do with the basic clocked logic D and JK flip-flops. These techniques are useful with the 4013 and 4027 by themselves or in simple circuits.

Most often, you'll probably only want to use a few 4013s or 4027s in your circuit as the fancier MSI blocks cram much more performance in a single package. If you find yourself using lots and lots of JK or D flops, try to find a MSI substitute or a different approach that will simplify the job for you. On the other hand, it's a very rare CMOS circuit that doesn't have two or three 4013s and maybe a 4027 tucked away in a corner somewhere to pick up some loose ends that the MSI can't handle directly. So it pays to be aware of all the different good things you can do with these basic clocked logic blocks.

Binary Counters

Binary counters are probably the oldest of clocked flip-flop uses. We can get a single stage to divide by two either by cross coupling $\overline{Q}$ to D on a 4013 or by making both J and K high on a 4027. The output alternates states, giving us a square wave with a 50-50 duty cycle of one-half the input clock frequency.

We can *cascade* binary counters as shown in Fig. 7. This lets us count to numbers higher than two or divide an input clock by a higher ratio. If the output of one divide-by-two (Fig. 7(a)) is connected to a second so its output clocks the second stage, we end up with the divide-by-four of Fig. 7(b). Add another stage, and we pick up the divide-by-eight of Fig. 7(c). More stages mean more possible count states and a higher division ratio. Four stages is particularly interesting. By itself, it can represent sixteen different things, count to sixteen, or scale an input frequency by sixteen. But, if we properly tamper with the count sequence, we can shorten our divide-by-sixteen into a divide-by-ten or decimal counter and do "by tens" counting and arithmetic.

These binary counters are called *ripple* counters. One stage has to change completely before the next stage can start its changing. Note

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

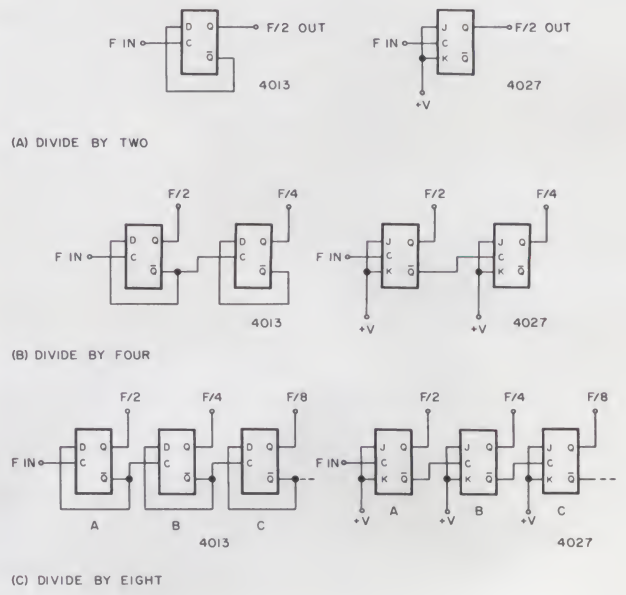
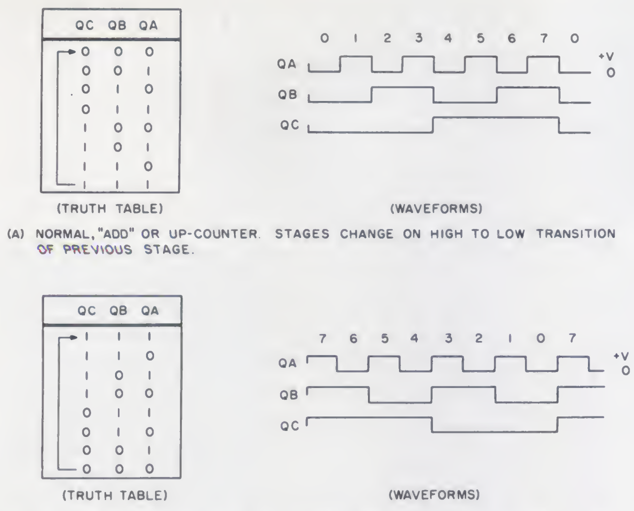


Fig. 7. Binary ripple counters.



QC	QB	QA
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

(A) NORMAL, "ADD" OR UP-COUNTER. STAGES CHANGE ON HIGH TO LOW TRANSITION OF PREVIOUS STAGE.

QC	QB	QA
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

(B) REVERSE, "SUBTRACT" OR DOWN-COUNTER. STAGES CHANGE ON LOW TO HIGH TRANSITION OF PREVIOUS STAGE.

Fig. 8. Binary counter waveforms.

that invalid output counts will happen during the *settling times* caused by the stage-to-stage *propagation delays*.

We can control the count direction, depending on how we drive the clock of each stage. Fig. 8 gives details. If we clock from $\overline{Q}$ of the previous stage, we get a normal, *add*, or binary up sequence as shown in Fig. 8(a). On the other hand, if we use the Q output to drive a positive edge clocked next stage, we end up with a backwards, *subtract* or down counter, as shown in Fig. 8(b). If our logic blocks are negative edge clocked (such as the 4024), the exact opposite is true — cascade from Q for a normal or up sequence and from $\overline{Q}$ for a reverse or down sequence.

Divide-by-Three

Fig. 9 shows us a *synchronous* divide-by-three counter using a 4027. Note that both stages are clocked at the same time from the input, so we don't have the propagation and ripple delay effects of cascaded stages. The output of this counter is said to be *weighted* 1-2, meaning that one output counts for "1" if it's there and the other one counts for "2" if it is present. So, you can directly look at the states and immediately tell what count is stored in the circuit. This circuit is the shortest example of the odd length walking ring counter. Two of the three counter states are self-decoding; the third is picked up with the NOR gate shown in Fig. 9(c).

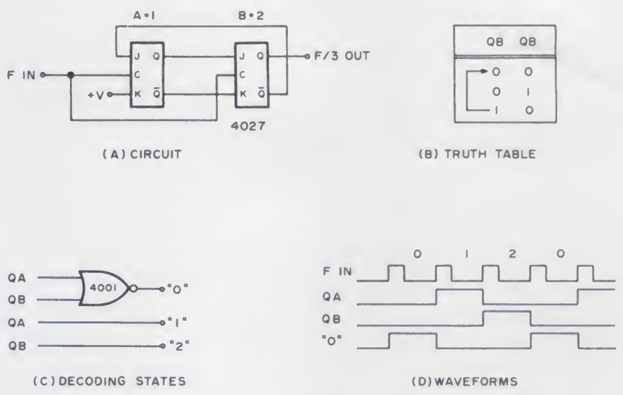
Divide-by-Four

A synchronous alternate to the ripple divide-by-four is shown in Fig. 10. We use the J and K low "do-nothing" state of a 4027 to inhibit the counting of the second stage half the time. Weighting is also 1-2. Four two-input AND gates may be used to decode the individual stages as shown.

This "do-nothing" inhibiting of a JK flip-flop is the key to longer synchronous counter. For a divide-by-eight, you only let the third stage count one-fourth of the time and inhibit it three-fourths of the time. A divide-by-sixteen can count only one-eighth of the time and so on. You can either use multiple input gates or a cascaded sequence of enabling two-input gates for longer synchronous counters.

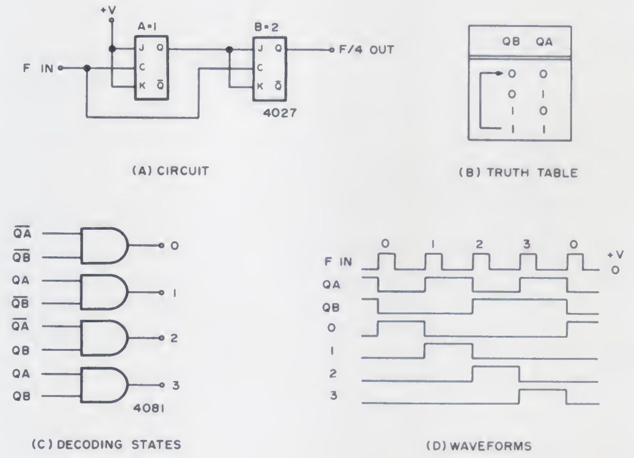
Divide-by-Five

Here is another example of our odd-length walking ring counter. As Fig. 11 shows us, the circuit is synchronous with all stages clocked di-



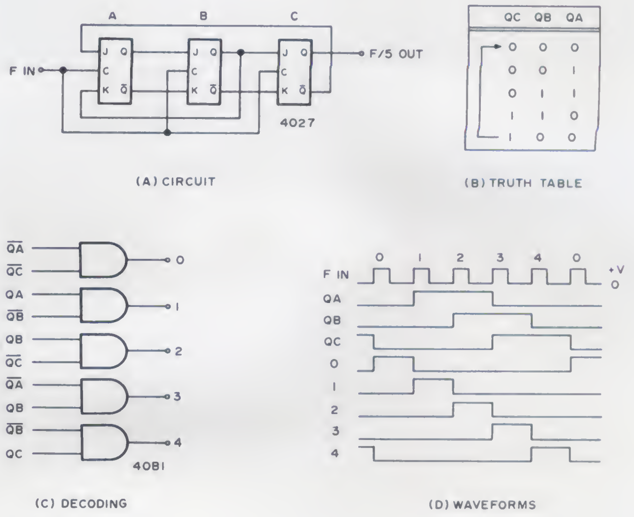
QB	QB
0	0
0	1
1	0

Fig. 9. Synchronous divide-by-three is weighted 1-2.



QB	QA
0	0
0	1
1	0
1	1

Fig. 10. Synchronous divide-by-four is weighted 1-2.



QC	QB	QA
0	0	0
0	0	1
0	1	1
1	1	0
1	0	0

Fig. 11. Synchronous divide-by-five has 3:2 output duty cycle.

rectly from the input. We can decode this particular circuit with five two-input AND gates as shown. The output is unweighted and has a 3:2 duty cycle.

Any of these counters can be reset to zero by using the Direct Reset inputs. You do have to be sure the direct input goes back low before the next clock pulse arrives. With combinations of direct set and direct reset, you can load any desired count into your circuit any time you want.

Shift Registers

A *shift register* is built as shown in Fig. 12. We cascade the Q output of a D flip-flop to the D input of the next stage. With JK flip-flops, we connect Q to J and $\overline{Q}$ to K, making sure the first stage always sees complementary data on the J and K inputs.

Each stage stores one *bit* of data, forming a *word* equal in length to the number of stages in the register. On clocking, each bit moves one stage to the right. The first stage picks up a new one or zero from the serial input. The last stage sends its output on to the outside world or loses it. The registers shown in Fig. 12 are usable as serial-in-serial-out (SISO) or serial-in-parallel-out (SIPO) registers. We can also build shift registers with parallel loading direct inputs, and, if we like, we can recirculate shift register data from output to input.

Storage Register

We can also use a pile of D flops all at once rather than having them pass data to each other. This gives us a *storage register* that accepts and holds a *parallel* word for us. An 8 bit parallel storage register is shown in Fig. 13.

Storage registers are useful to catch data on the way by, particularly from a microprocessor. They then hold the data as long as we need it. You can also use storage registers to sample data when it is known to be good, eliminating any intermediate garbage caused by settling times, propagation delays, and so on. A storage register on the output of an electronic music digital keyboard will hold the note command for us after key release. This lets the note decay and fallback continue after the note is let go, still telling the rest of the circuit what note it was working on.

Some MSI examples of storage registers include the 4175 quad, 4174 hex, and 4034 eight-bit devices.

Monostable Multivibrators

A normal monostable multivibrator using the 4013 D flop is shown in Fig. 14(a). Clocking drives Q high, which charges C through the series combination of R2 and the much smaller R1. When the cycle ends, C is rapidly discharged through R1 only. Leaving R1 off gives very fast recovery but distorts the Q output waveform. If a long recovery time is available, we can use R2 only and omit the diode.

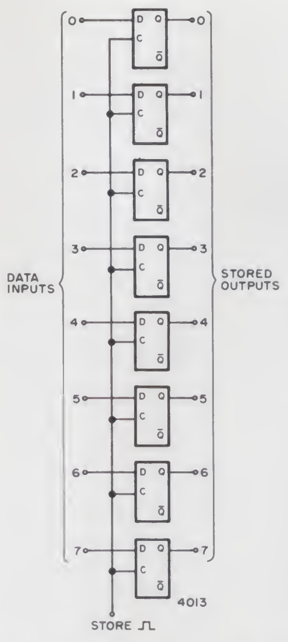


Fig. 13. 8-bit word storage latch for a microprocessor.

To pick up a retrigger ability, examine Fig. 14(b). Here the input clock low time discharges the capacitor through R1. The positive clock edge drives Q high and R2 charges C for the delay-until-reset time. The circuit may be triggered at any time and will time out from the last triggering. Note that the monostable ON cycle cannot *end* while the clock is low.

We can also use the alternate trigger method of Fig. 14(c). Here we pulse the SET input to start timing. This takes a resistor and a capacitor, but gives us a second way to positive edge trigger. The time constant on the trigger must be shorter than the ON time for proper operation.

The system reset/power-on generator of Fig. 14(d) will give you a clean reset signal shortly after power is applied to your system. Applying supply power triggers the monostable which then times out long enough for the supply to reach a stable value. The trailing edge of the monostable can then be used for a system reset. This type of circuit is handy for initializing things like microprocessors, making sure every thing comes up in a benign state when first activated. ■

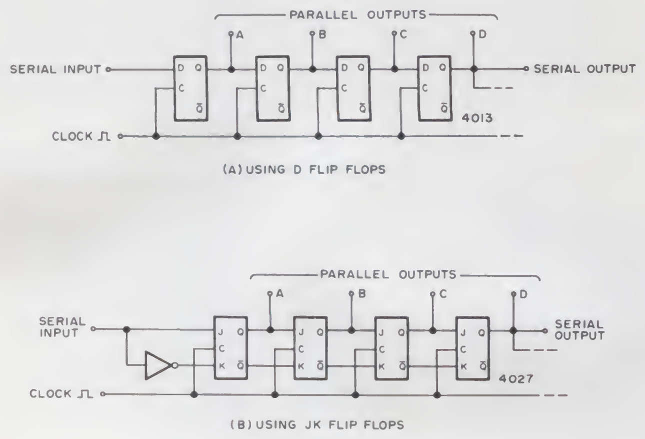
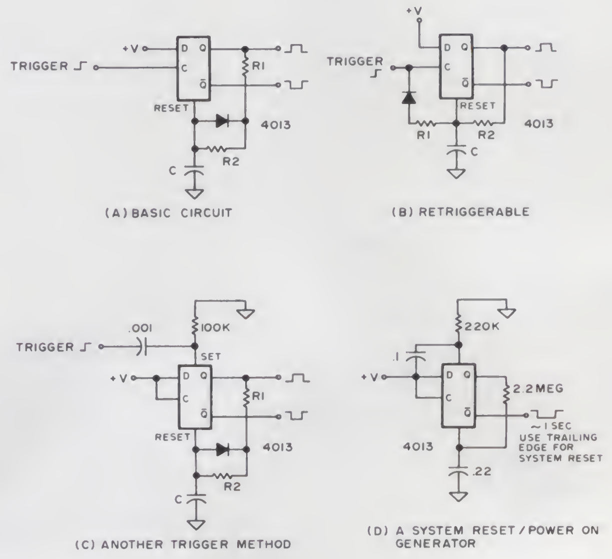


Fig. 12. Shift registers.



TRIGGER
+V
R1
R2
C
RESET
4013
(A) BASIC CIRCUIT
(B) RETRIGGERABLE
.001
100K
SET
(C) ANOTHER TRIGGER METHOD
.1
220K
2.2MEG
~1 SEC
USE TRAILING EDGE FOR SYSTEM RESET
.22
(D) A SYSTEM RESET/POWER ON GENERATOR

Fig. 14. Monostable circuits.