# DRAWING BOARD

**ROBERT GROSSBLATT,**
CIRCUITS EDITOR

## A complete circuit

OVER THE LAST COUPLE OF MONTHS we've gone through the steps needed to design custom-character generators and looked at some simple ways to use them. Now let's turn all the pieces into a useful circuit.

The handiest thing to come up with is a way to use one character generator to drive several digits.

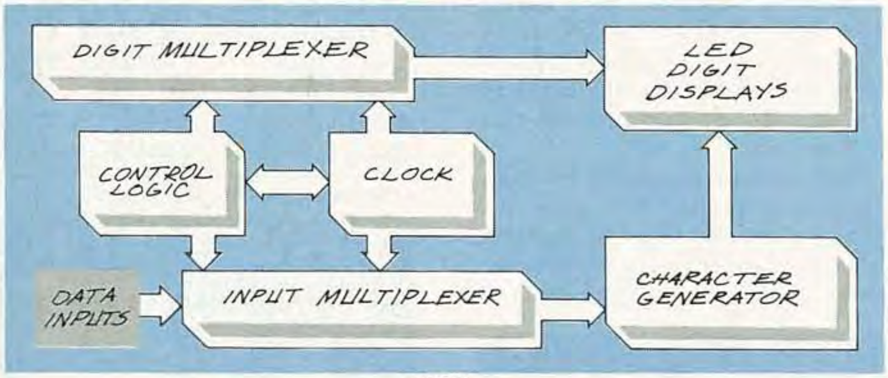

FIG. 1

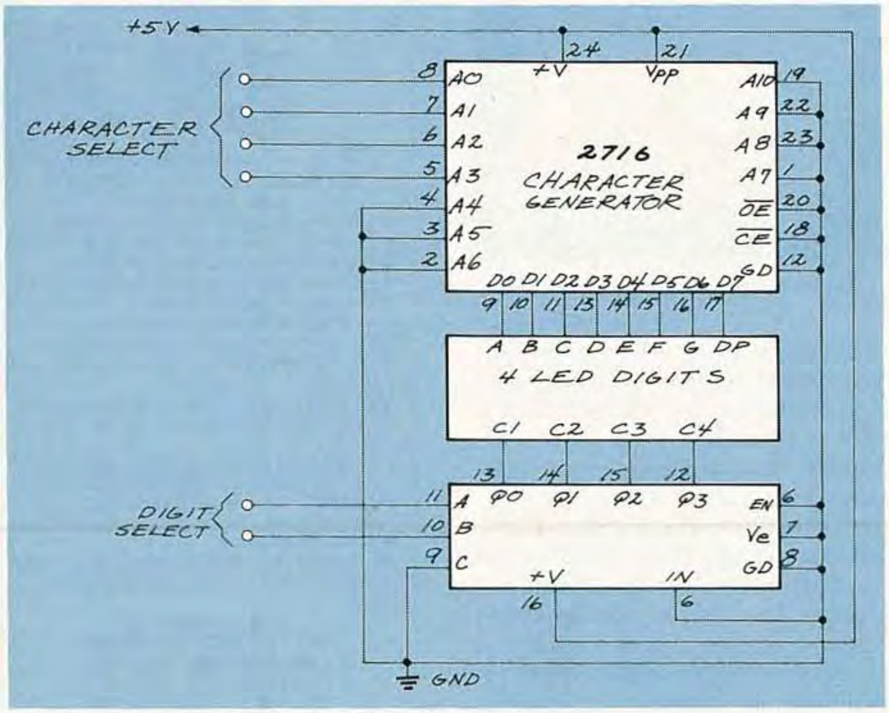

FIG. 2

| 257#1 OE | 257#1 SEL | 257#2 OE | 257#2 SEL | DISPLAYED DIGIT |
|---|---|---|---|---|
| L | L | H | X | INPUT 1 |
| L | H | H | X | INPUT 2 |
| H | X | L | L | INPUT 3 |
| H | X | L | H | INPUT 4 |

FIG. 3

| OE | SEL#1 | SEL#2 | DISPLAY |
|---|---|---|---|
| L | L | X | INPUT 1 |
| L | H | X | INPUT 2 |
| H | X | L | INPUT 3 |
| H | X | H | INPUT 4 |

FIG. 4

The basic idea here is to build a general-purpose display circuit.

Let's lay down some criteria:

**1.** The circuit will drive four digits.
**2.** Each digit will have its own set of inputs.
**3.** Only one character generator will be used.
**4.** The circuit will display all the hex digits from 0000h to FFFFh.

Even though we've been designing a character generator that can handle a lot of the ASCII characters, limiting our display to hex will keep the circuit simpler. If you absolutely must display ASCII characters, the circuit will be basically the same, but you'll need more bits assigned to each of the digits. A hex display only cares about the lower four bits while a full ASCII display has to deal with seven bits.

When you come right down to

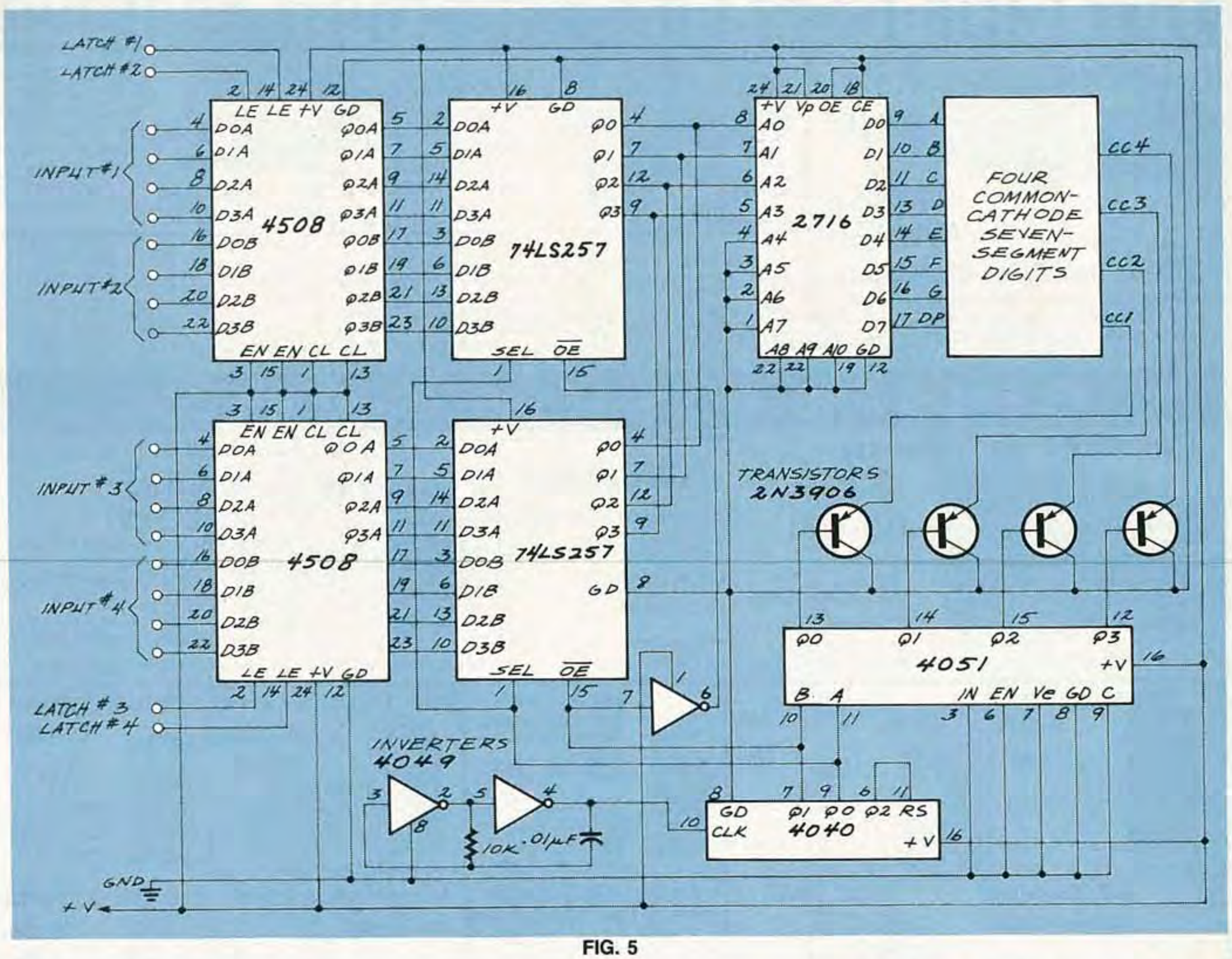

FIG. 5

it, we want the display circuit to be your basic black box with sixteen inputs—four for each of the four digits we'll be driving. Since we're actually building something that can be used elsewhere, we can eliminate some of the parameters we put into the EPROM. We're already disregarding the ASCII stuff and now we'll make the decision to use common-cathode displays.

That last decision is no big deal because it's a relatively trivial thing to convert the circuit to work with a common-anode digit...but we're getting ahead of ourselves.

The block diagram of the circuit we'll be designing is shown in Fig. 1. The heart of the circuit is really the control logic because it has the job of keeping everything in sync. We have to be sure that when we're sending character number 1 to the input multiplexer, that we're also turning on seven-segment LED number 1. If things get out of sync you might have something up on the display but it's not going to be anything useful.

The starting point of the circuit is shown in Fig. 2. It's similar to the circuit that we looked at in May, but there are two main differences. The first is that we're only using A0–A3 on the EPROM and the second is that the 4051 is going to drive only four digits so the "C" input (pin 9) is tied low. The same thing is done with all of the unused EPROM address lines.

We've already decided on the 4051 as the digit multiplexer so let's take a look at the input multiplexer as well before getting to the control logic. After all, you can't design control logic until you know what you have to control.

Just as the 4051 will sequentially turn on one digit after another, the input multiplexer has to select the corresponding digit data to be displayed. What we need is the electronic equivalent of a four-pole, four-position rotary switch, and one way to do that is to use a pair of 74LS257's. You can use the TTL version of the chip or the 74HC257 or 74HCT257 pin-equivalent CMOS parts.

The inputs that will appear at the outputs depend on the state of the SEL input. Making that pin low will select the first set of inputs and making it high will select the others. What makes the 257 a good IC for our application is that it also has an OUTPUT-ENABLE pin so that our output can have three states.

Now that we know what multiplexers we'll be using, we can work out what we need for control logic. Designing this kind of circuitry can be a really brain-bending exercise but one way to cut it down to size is to use a truth table like the one shown in Fig. 3.

It may seem a bit confusing at first glance, but one thing it tells us right away is that the OUTPUT-ENABLE pins of the 257's are always opposite each other. When one is high, the other is low, and vice versa. That means we can tie them

*continued on page 85*

*continued from page 29*

together through an inverter and eliminate one column in the table. We can tie the two SEL inputs together as well since their state is only important when the chip is enabled. By re-arranging the truth table slightly we'll wind up with the one shown in Fig. 4, and that, as you should realize, is simple to implement because it's just counting up in a plain binary.

Not only that, but we can use a straight binary counter and use the "C" output to toggle the reset. Putting that into practice will produce a circuit like the one shown in Fig. 5. You should work out the logic in your own mind to make sure that you understand what's going on there.

The last part of the control logic is the counter and that can be any binary counter. You can use a 74LS93, 4040, half a 4520 or 4518, half a 74LS393, or just about anything that can count up to four in binary. The 4040 in Fig. 5 is a good choice because it's easy to use and is a mainstream CMOS part, but don't hesitate to use something you happen to have lying around.

The circuit we've come up with will fill all the design criteria we laid out earlier but it has three sections we still have to go over. The first is the clock, the second is the input latching, and the third is the use of pass transistors to drive the cathodes.

Some time ago we spent several columns talking about the ins and outs of scan oscillators and we found that you can use any frequency as long as it's high enough to eliminate flickering. Anything over 10 kHz or so will fill the bill and any oscillator capable of driving the clock inputs will be as good as any other. I've built mine out of a pair of inverters but if your application has a handy clock line that fills those minimal requirements, you may use that.

Input latches aren't really necessary but they can be useful if the data you want displayed doesn't stay around too long. That would be the case if you want to snatch bus data when a certain pulse shows up elsewhere in the circuit.

I'm using 4508 octal latches but only because I happened to have a bunch of them in the parts box. Notice that I didn't say "junk-box"—the only thing junk parts are good for is building junk. One thing that's nice about 4508's is that they're really two separate four-bit latches in a single package.

The last thing to talk about is the use of pass transistors. If you put together the circuit we showed you in May you probably noticed that the display was rather dim. There's nothing you can do about the scan time, but it is possible to zip more current through a digit when it's selected. That is exactly what the pass transistors are doing.

One disadvantage of that approach is that the brightness of the digit will depend somewhat on how many segments are being lit, but it's not enough of a problem to make the use of individual current-limiting resistors in your circuit an absolute necessity.

Breadboard the circuit of Fig. 5 and feed the sixteen inputs with two cascaded 4040's. You'll see the display count up in true binary, and you'll also know that the circuit works. The circuit is extremely useful and it's well worth the time to generate a PC board for it. The complexity is such that it will more than likely require a double-sided board, but once you've got it done, you can make as many of them as you want. I know it's not easy to produce a double-sided board, but there are some tricks you can use to make the job easier. We'll be looking at that, and some other things as well, next time. **R-E**