DRAWING BOARD

After some final words on video, we'll continue with our controller.

ROBERT GROSSBLATT

A few months ago I did a series on video that ran over several issues of the magazine (**Radio-Electronics**, January 1990–August 1990). As with most of the stuff done in this column, the general idea I had was to go through the basics of video theory and production. Since it would be wrong to assume that any of you have a real familiarity with any of the subjects discussed here, I have to be sure and lay a good theoretical foundation before I start developing hardware designed to demonstrate the theory we're talking about.

Every time we start a new subject, there's a conflict between the scope of the subject and the amount of space I have to cover it. The column space is limited and, to be fair to everybody, I can spend only a certain number of columns on any one particular subject. Remember that not everybody is interested in everything.

When I started the series on video, I fully intended to take it a bit further and describe the theory and design of circuitry that would put images up on the screen. It seemed to me to be a good way to end the subject. But, as with a lot of things in life, it just didn't turn out that way.

In the first place, it took longer than I thought to go through the theory of the video standard, and designing circuitry to generate the video waveform was complex enough to fill several consecutive issues of the magazine. It doesn't do anyone any good to dump schematics on the page unless they're well documented. Since the purpose behind all the stuff I do is to have people learn, complete descriptions of the hardware we develop here is every bit as important as the hardware itself.

By the time I finished the basic video hardware I had already spent several months on the subject and I realized there would be no room to go into producing images as well. But there's more to it than that. If you followed the video series you should realize that the circuitry needed to generate things like bars and dots really has nothing to do with video. As I mentioned in the series, all you need to put these type of images up on the screen is a regularly produced stream of highs and lows whose timing is locked to the sync signals generated by the circuitry we designed.

This certainly isn't a trivial thing to do and, while there's all sorts of stuff you can learn by designing something like that, it's also a fact that it's not video circuitry either. Even though it would have been great to do it here, it would have taken too much room to do properly. And since we had gone over all the necessary theory, I decided to use the subject as the basis of a contest. If you've seriously followed the video series, understood the theory and built the hardware, there's no reason why you can't work out the image-producing circuitry yourself. It's a good exercise in design, and figuring out how to lock it to sync is the best way I know of to see if you really understand the video circuitry we developed together.

I've already gotten some contest entries in the mail, and I'm amazed at some of the schemes used to put images on the screen. My hat's off to everyone.

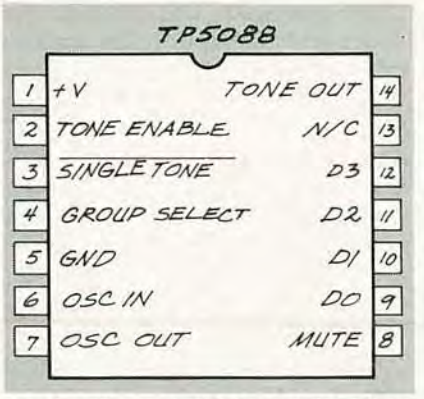

FIG. 1—THE 5088 DTMF GENERATOR is designed to be used under microprocessor control and, instead of the eight lines normally needed for keypad control, the digit is sent as four-bit binary information to the four data inputs.

Before we get back to the hardware stuff we were talking about last month, there's one more thing I want to get off my chest. A lot of people have written in asking me to show them how to beat scrambling and to put the circuit details in the column. I went through the basics of video scrambling at the end of the series and, if you read it, you know that there are lots of scrambling schemes in use across the country. There's no way for me to do it—it's just not going to happen here. You can't design circuitry to beat scrambled video without understanding how unscrambled video works. Once you've got a good handle on video, you should be able to throw the scrambled signal on a scope, figure out what's being done to mess it up, and work out what you have to do to straighten it out.

I've said this lots of times before but Grossblatt's twelfth law is worth repeating: You have to know the rules to be able to break the rules. So, the only way you're going to be able to correct a scrambled signal is to know what's being done to scramble it in the first place. Enough.

Latches and more latches

The last time we left our controller design, we were talking about latches. So, before we go any further, you should have enough latches on your breadboard to cover all the port lines you plan on using. When I design circuitry, I have a real aversion to unused silicon. I can't tell you why, but it really bugs me to have unused gates and IC halves sitting on the board that do nothing other than draw current. It may have something to do with my toilet training, but I'll spend a lot of time making sure that everything on the board has a purpose.

It's impossible to know how many latches to add to the board unless you know what you're designing, so the time has come to decide what we

want the circuit to do. Now everybody has their own ideas but since I have no way of finding out what's in your mind (columns are written several months before publication), it seems that a neat thing to design would be a telephone dialer.

Everybody uses a telephone and having a circuit that hangs off a parallel port and talks to the phone line can open a wide range of design possibilities. Lots of bells and whistles can be added to it and, if the software is well written, the circuit can be the basis of a really useful product.

The heart of any telephone dialer is the generation of DTMF tones and that's something we've done several times in the past. It's worth your time to hunt through the back issues of RE and find the ones in which we designed our remote control system since those issues contain a good description of the entire DTMF standard (**Radio-Electronics**, January, March, and April 1987). If you don't keep any back issues around (although you really should), you might be able to find copies of those months in your local library.

Generating DTMF tones used to be a real pain in the neck but things got better several years ago when semiconductor manufacturers saw the problem as a hole in the market. As things stand today, there are a slew of DTMF generator chips around and once you've worked out exactly what your needs are, you can be sure there's a chip available with all the ins and outs you're looking for.

Since we're going to drive the chip off the parallel port, one of the overriding considerations for us to keep in mind is that we should conserve lines. Most of the DTMF chips want to see the row and column inputs produced by keypads. This means that eight lines have to be devoted to selecting the tone you want to generate. We have eight data lines at the output of the parallel port but it's a good idea to be stingy with them since using them all to control tone generation will limit what we can do later on.

Fortunately for all of us, National Semiconductor (and possibly other manufacturers), makes the 5088 DTMF generator. This chip has been designed specifically to be used under microprocessor control and is driven with straight binary. Instead of the eight lines needed for keypad control, the digit is sent as four-bit binary information to the four data inputs of the 5088. The pinouts of the 5088 are shown in Fig. 1 and, if you're at all familiar with DTMF generators, you should be able to recognize the rest of the controls on the chip.

If you have a hard time finding this chip, or if you decide—for whatever reason—to use a different one, you'll have to modify the circuitry we'll be developing. This is especially true if your chip has to be driven with eight data lines. As an alternative, it doesn't take a lot of circuitry to convert the four lines I'll be using to the eight lines needed by other DTMF generators. Probably the easiest way to do something like that is to program an EPROM to do the conversion for you. It wastes a lot of EPROM space but it's quick.

Aside from the weighted binary inputs, the 5088 has a TONE ENABLE input that, besides inhibiting the generation of tones, also puts the chip into a low-power state that drops the current requirements to only 55 microamps. That isn't really such a big deal since even while the chip is active it draws only 1.5 milliamps.

Pins 3 and 4 allow you to separately generate the high and low group tones that make up the DTMF standard. That's handy when you're setting things up since you can check the frequencies. The MUTE pin is an output that becomes active when the chip is generating tones. It's generally used to turn off the telephone handset to prevent the tones from being heard while they're being generated. The main use for that is in the design of telephones, but most of the phone designs I've seen don't bother doing that.

Using the chip is easy since it only takes the addition of a cheap colorburst crystal to make things work. When we get together next month, we'll put it in the circuit, write some minimal software, and add some bells and whistles to the design. **R-E**