BY ROBERT GROSSBLATT

This month we start the design of a general-purpose controller circuit.

WHENEVER I HAVE TO START ON A NEW PROJECT HERE, THE FIRST PLACE I SEARCH FOR IDEAS IS IN THE MAIL. HISTORICALLY SPEAKING, MY MAIL HAS CHANGED OVER THE YEARS.

Once upon a time, the majority of the requests I received were for coverage of very general subjects—a tutorial on op-amps, for example, or how to design with FETs. In the last few years, though, the requests have changed.

Most of the mail now concerns specific applications rather than general ones. This is okay with me because I've always believed that you learn more by working with real-world circuits. The only problem with doing this is that the projects often get so complex that the general theory behind them tends to get lost. Without a thorough understanding of the underlying theory, all you're really doing is building a kit. This becomes evident if you try to change to the design because you suddenly realize that while you might have built a circuit, you don't understand how it works—or why it doesn't work!

The best way for me to answer all the application-specific requests I've received is to design a circuit that can do many things with minimum changes to the design. The best way to design such a chameleon-like circuit is to base it on a microprocessor. Once you have a basic microprocessor circuit working with a good amount of controllable input/output (I/O), you can alter its function just by writing new software.

Designing a general-purpose controller circuit isn't all that difficult, but it is definitely more complicated than using discreet logic. The price you pay for a microprocessor's versatility is that you must know how the microprocessor works and how it can be programmed.

Different microprocessors require different support hardware and software. However, they differ only in the details—the principles are the same. If you can design around Intel microprocessors, you won't have any trouble adjusting to the Motorola family. I'll be using an Intel microprocessor here because they're inexpensive and easy to find. Perhaps the most important reason to use Intel processors is that you'll need to write software, and every version of DOS comes with utilities that make the job of converting source code to Intel-compatible binary files easy to do. Finally, I've designed many circuits around Intel processors, seen most of the problems, and have worked out most of the answers.

Over the next several cloumns,

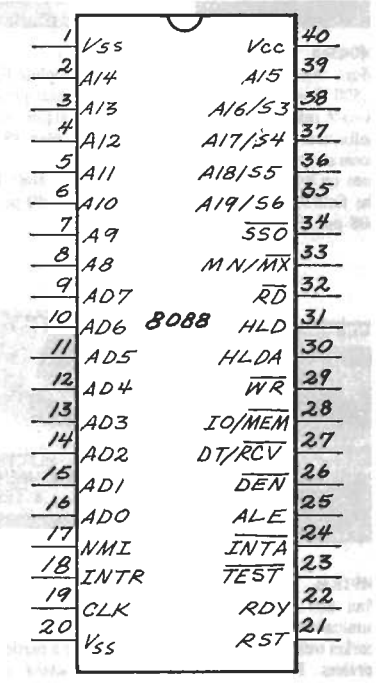


FIG. 1—THE 8088 MICROPROCESSOR is really a 16-bit chip squeezed into an 8-bit package.

I'm going to design a general-purpose, microprocessor-based controller circuit, with software configurable I/O. Although I'll go through each step in detail, it will be impossible for you to follow unless you have the following:

1. Data sheets on the ICs we'll be using
2. A basic understanding of microprocessors
3. A general understanding of programming
4. Access to an oscilloscope
5. A familiarity with hex code
6. Access to an EPROM programmer

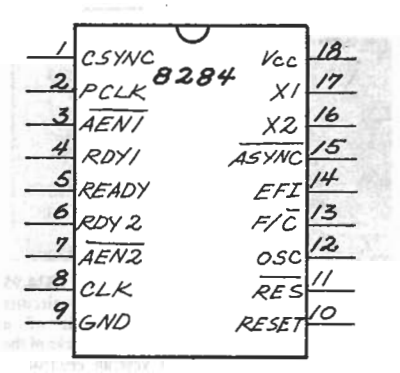


FIG. 2—THE 8284 CLOCK CONTROLLER will produce clock signals that are exactly tailored to driving the 8088.

It's sometimes hard to find books on this subject and that's why I wrote one myself. It's called "The 8088 Project Book," Book #3171 from TAB Books (800) 822-8158. You won't find data sheets (or an oscilloscope) in the book, but it will probably answer any questions you have concerning microprocessors.

The 8088 CPU

The 8088 microprocessor, whose pinout is shown in Fig. 1, is a field-tested CPU that you can buy for about a buck. If you look at the pin names, you'll notice that some of the pins perform double duty. This is because the 8088 is really a 16-bit chip squeezed into an 8-bit package. The standard 40-pin DIP just doesn't have enough pins for every line to be brought out separately.

A microprocessor needs a clock signal, memory, and software to tell it what to do. The 8088's pin-multiplexing scheme requires some additional, minimal hardware.

CPU satisfaction

The first of the 8088's circuit requirements is a clock signal, and this is pretty easy to satisfy. The clock must have sharp shoulders (rise and fall times of less than 10 microseconds) and a 33% duty cycle. The maximum speed depends on the CPU. A standard 8088 tops out at 4.77 MHz, the 8088-2 can run at 8 MHz, and the 8088-1 goes up to 10 MHz. Minimum speeds for all flavors of the CPU is about 2 MHz because the internal registers are made from single storage capacitors that must be refreshed in the same way as dynamic RAM. This saves space on the substrate, but it made it a bit more difficult to really slow things down to debug a circuit. Since you can't slow the clock to a crawl, you must use software to put in lots and lots of wait states to get the same effect.

When Intel first started to design the 80XX series of microprocessors, one goal was to offer a set of chips specifically designed to support the CPUs. These included clock generators, bus controllers, coprocessors, DMA controllers, and so on. Some of them became popular and others didn't. One of the most popular was the 8284 clock controller shown in Fig. 2. This chip, plus a handful of components, will produce clock signals that are exactly tailored to driving the 8088.

Getting the 8284 to generate the clock signals for the CPU is a piece of cake. As you can see in the circuit of Fig. 3, most of the control and data pins on the 8284 aren't needed for the simple job that we want the chip to do. They come into play when the clock generator must drive more than one CPU or generate more than one clock frequency. My circuit will use the 8284 in its simplest configuration—as a clock source for the 8088.

You have to get the clock circuit working before you can go any further since it's going to drive the rest of the circuitry you'll be assembling over the next few months. It's simple to find out whether or not you have the clock circuit working correctly. There are three clock outputs available on the chip: +scclk+xsc at pin 8 is the main output for the 8088. It's one third the crystal output and has a duty cycle of 33% that's required by the 8088. OSC at pin 12 is a TTL-level buffered version of the crystal frequency. PCLK at pin 2 is a TTL-level frequency that's equal to half the CLK output and has a 50% duty cycle.

If you put an oscilloscope or frequency counter on any of those outputs, you'll know immediately if the clock is working correctly. Next time I'll be adding the CPU, creating the address and data buses, adding memory, and putting some basic I/O on the board. A lot of information is going to be thrown at you and you'll stand a lot better chance of catching it if you have the basic data on paper in front of you. EN

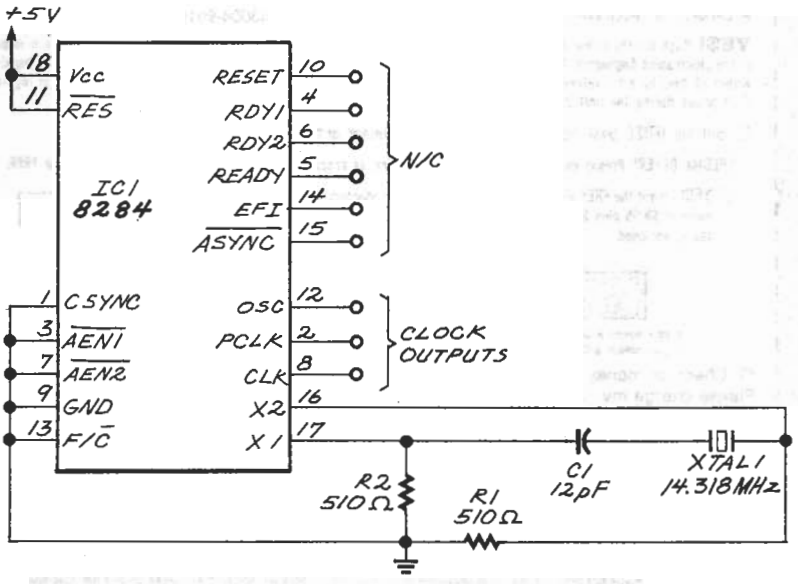
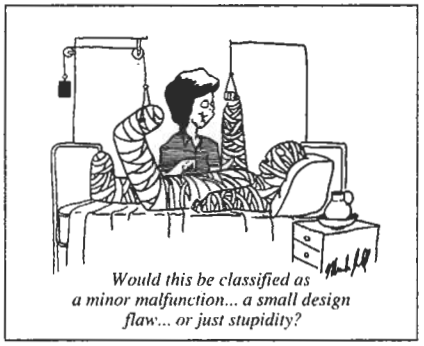


FIG. 3—WE'RE USING THE 8284 in its simplest configuration as a clock source for the 8088. Most of the control and data pins on the 8284 aren't needed for this project.



Would this be classified as a minor malfunction... a small design flaw... or just stupidity?