

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

LESSONS FROM THE TRENCHES

George Martin

Embed This PC

Part 3: Emulator to the App Test

Now that George has explained some of the requirements for embedding a 486, it's time to get things going and start testing. Before he's done with Part 3, you'll have a simple design, some startup code for the emulator, and a project.



In this series of articles about embedding a 486, I identified a laser light show as an application suitable for an embedded 486 project. The application continuously outputs data as fast as possible to a couple D/A converters. I discussed one DRAM device and AMD's ELAN processors and covered many of the issues necessary to de-

velop an embedded 486 project.

Last time, I left you with a table of connections that wire the DRAM to the CPU. I will continue from that point and detail the emulator, as well as discuss real- versus protected-mode software development.

EMULATION CHECK POINT

For emulation, I discussed an EPROM emulator and identified Grammar Engine's PromIce and Paradigm's PDREMOTE software as good tools for a low-cost, useful solution. Let's explore this solution further.

By using a 29F040 flash-memory device for the startup (BIOS) code that's packaged in a standard 32-pin DIP format, I can use an EPROM emulator and plug it into the socket that replaces the BIOS flash device. (By the way, for the purpose of this discussion, I'll use the terms flash memory and EPROM interchangeably.)

If your code is small enough, an EPROM might be suitable. EPROMs can be erased with UV light and reprogrammed. The flash-memory devices are larger, but they need to be electrically erased before they're re-

	DRAM 1		DRAM 3		DRAM 4	
	DRAM 2		DRAM 3		DRAM 4	
	Bank 0		Bank 1		Buffer	
Signal	Low	High	Low	High		
MA0	17	17	17	17	18	74LVTH244ADW 2
MA1	18	18	18	18	16	74LVTH244ADW 4
MA2	19	19	19	19	14	74LVTH244ADW 6
MA3	20	20	20	20	12	74LVTH244ADW 8
MA4	23	23	23	23	9	74LVTH244ADW 11
MA5	24	24	24	24	7	74LVTH244ADW 13
MA6	25	25	25	25	5	74LVTH244ADW 15
MA7	26	26	26	26	3	74LVTH244ADW 17
MA8	27	27	27	27	18	74LVTH244ADW 2
MA9	28	28	28	28	16	74LVTH244ADW 4

Table 1—Here are the DRAM connections for the Elan SC400. As you can see, DRAM 1 and 2 share Bank 0, while DRAM 3 and 4 share Bank 1.

grammed, which ties up the programmer (see Table 1).

The BIOS EPROM contains the reset vectors for the CPU. At power up, the CPU vectors address the device, fetch the code, and execute. Your PC has the same sort of device. Code from the PC BIOS EPROM is copied into the DRAM for faster execution. The BIOS EPROM typically has an 8-bit data path and slower access time than the CPU and DRAM.

Also, if the BIOS EPROM is a flash-memory device, it's possible to reprogram it in the field. But, no matter what the detail of your particular design, you can have full debugging access through the 32-pin DIP socket. The PromIce emulator plugs into the socket used by the BIOS device.

Before you're ready to go, you need to make two extra connections to the hardware. First, the reset signal from the PromIce must go to the reset line of your system. The reset line holds the CPU reset, while PromIce loads the device contents. Secondly, the write line from the CPU needs to be connected to the PromIce. PromIce uses the write line to intercept commands and data from the CPU.

The PromIce emulator includes a LoadIce program, which loads or forces code into the EPROM. Now, let's work on something to load.

THE PROGRAM

This is where Paradigm's C++ tools come in. The software package, PDREMOTE, is a stand-alone product that can be used with several software development packages. Paradigm also has its own integrated development environment, which appears to be licensed from Borland and, as you know from my last column, I highly recommend Turbo C 3.0.

From the Paradigm inte-

grated environment, you can create several types of applications, including a remote startup application. Feel free to download a copy of a typical PDREMOTE startup source file for an SC400 (**download pdrema.asm**). This startup source file gets the CPU under control, configures the hardware, and transfers control to the PDREMOTE software.

Paradigm has done a good job of separating all you need to know about the PDREMOTE software from what you need to alter. It's easy to bring up a nonstandard system—which, as you know, includes just about everything.

If you download the source file, you'll notice a SC400RegInit table, which loads the configuration registers into the SC400. I copied this table

from the Paradigm software that runs on the AMD demo boards, which saved me a lot of time.

Just after the table, I placed the startup label, which is where the CPU reset vector is set to point. You'll notice several "This loops forever" comments. In the first one, the `jmp mm9` instruction completely bypasses the loop. But, if that instruction is commented out, the CPU continuously performs the `out dx,al` and `in al,dx` instruction pair.

When I comment out the `jmp mm9` instruction, I can look at my hardware and see the I/O write and read instruction activity. Also, I see the decode for the specific I/O address loaded into the DX register. With the loop enabled, if the CPU starts running, it will get stuck in the loop, and I can observe the activity with a scope.

When you're bringing up a new piece of hardware, techniques such as these are invaluable. They help you figure out if the CPU is running, and you can observe the power supplies and clock signals. If you get to the loop and remain stuck, you know the reset vector and BIOS EPROM instruction fetches are working.

With that first loop disabled, the registers are loaded, and the RTC oscillator is enabled. Next, a wait is inserted to let all the SC400 hardware catch up with what you've just done. The waiting period probably needs to be adjusted for your specific application.

PDREMOTE software is initialized and ends with a call to `PROGmain`, which is the actual PDREMOTE software. The PDREM software, at this point, should be running. Now, you should have an emulator and debugger for your applica-

ADDITIONAL *CIRCUIT CELLAR* ARTICLES

If you need to know more about real vs. protected mode, check out the following *Circuit Cellar* articles:

Circuit Cellar 48–68 Firmware Furnace: Journey to the Protected Land.

- The '386SX Takes the First Step
- Segments All the Way Down
- Smashing Bugs in Gates
- Booting into Protected Mode
- Base Camp at 1 Megabyte
- Fancy Text Output and a Boot Mystery
- Serious CISC Meets the Taskettes
- Infrastructure Improvement
- Smashed Gates & Conforming Code
- With Interrupts, Timing is Everything
- Memory, Time and I/O
- How the PC Keyboard Got its Bits
- The Mystery of Scan Code Set 3
- Of Characters and Keystrokes
- Entering Virtual-86 Mode
- Looking at the Virtual-86 Monitor
- Real Interrupts in Virtual-86 Mode
- Behind the Interrupt Curtain
- Getting Vid-Link in Sync
- Part 1: Unscrewing the Inscrutable
- Part 2: Bits to Dots
- Flat Surfaces, Widowmakers, and a Surfeit of Bugs

Tom Shanley, Protected-Mode Software Architecture, Addison Wesley. The book is available at <http://www.awl.com/corp/> for \$33. I'm not sure how valuable this book is, but it's about the only one I could find still in print!

tion software.

The Paradigm design environment was used to create this PDREMOTE application. Depending on your specific design, you can locate the PDREMOTE at a location such as F000:0000. I'll explain this notation later for those of you who are not familiar with the segmented architecture.

The compiler-linker then creates a loadable module. That module is loaded using the LoadIce utility from Grammar Engine. The LoadIce utility clears the emulator memory, loads and verifies the program, resets the CPU, and starts running. The EPROM emulator can be powered separately or from your CPU circuitry. I typically load the emulator with the CPU power off. Then after LoadIce is complete, I power up the CPU.

REAL MODE

Most 486s start in operating modes that permit software written for the original PC of the early '80s. So, this means that addressing is limited to 20 bits and that the processor is in real mode. Essentially, you have an 8086 with the 20-bit addressing of the original PCs.

Some programs relied on the addresses increasing from 00000 up to FFFFF and then rolling over to 00000, which is the 20-bit addressing limit. The real mode uses 16-bit registers such as AX, not the EAX, and the addresses are made up of a 16-bit segment with a 16-bit offset in that segment. This, of course, explains the F000:0000 notation. You're looking at the SEGMENT:OFFSET.

No protection mechanism is supported in the CPU to prevent programs from accessing resources.

PROTECTED MODE

The other mode is called the protected mode. In the protected mode, various types of protection are offered. In this mode, one program can run independent of all others and be protected. Registers become 32 bits wide. In its simplest form, you can have one program that has all the resources of the CPU. If you're interested in all the complexities of real and protected

mode, check out the "Additional Ink Articles" sidebar for other references.

The CPU starts in real mode and then switches to protected mode. If you're running an RTOS, you can find startup code that does this switch. Once the PDREMOTE software is loaded and running, the Paradigm design environment connects to the PDREMOTE software with some configuration of the environment. I used the parallel port to communicate between the development PC and the PromIce device. With this, I have a real emulator that can inspect the CPU, memory, and I/O just like an expensive hardware emulator solution.

Next, you need to create an application that you load and debug. Let's look at creating a real-mode application that can be loaded and run with the debugger.

GETTING AN APP UP AND RUNNING

At present, Paradigm only works in 16-bit real mode, although I noticed some 32-bit register optimizations and understand it is releasing a 32-bit protected-mode version.

To create a project in the Paradigm environment, use the Paradigm project wizard software to generate a set of startup modules and a dummy main module. You can add your application code to the module set. Be sure to tell the linker which libraries to use and the locator where to place all the code.

While you debug the code and have PDREMOTE running, you need to map differently than for the final production version. The PDREMOTE code currently resides at F000:0000 as a full 64-KB segment, although it is much smaller. I moved the application to D000:0000, which is another free segment in the BIOS flash memory.

You also must defeat the setting of the CPU configuration registers in and out application. PDREMOTE is running and already did that setup. You can insert a jump in the assembly language startup module to bypass the register setting. You might consider that an ugly approach, but beauty is in

the eye of the beholder. Perhaps a define or conditional compile (assemble) flag would be more elegant.

Actually, I created a directory in the working project directory, which I named EPROM. All the code is the same in the two directories, except for changes I'm discussing. I can edit and debug code in the debug directory. Then, when I want to make an EPROM, I can copy all the C and H files to the EPROM directory and compile. By using this method, a file-compare program can contrast the two sets of files and find the minor differences created during debugging.

WHAT'S NEXT

So, now I have an emulator, a simple design of a 486 using AMD SC400, startup code for the emulator, and a project. Next month, I'll take a look at some of the details of the SC400 and the addition of the D/A converters. ☒

George started his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and cofounded a design and manufacturing firm. Typical systems that George designs include servo-motion control, graphical input and output, data acquisition, and remote control. George is a charter member of the Ciarcia Design Works Team and most recently, he's been working on the people-tracking system for Bill Gates' new house. You can reach him at george.martin@worldnet.att.net.

SOURCES

Turbo C 3.0

Borland
(408) 431-1000
Fax: (831) 431-4122
www.inprise.com

C++ Tools

Paradigm
(607) 748-5966
Fax: (607) 748-5968
www.devtools.com

Grammar Engine

(614) 899-7878
Fax: (614) 899-7888
www.gei.com

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199, subscribe@circuitcellar.com or www.circuitcellar.com/subscribe.htm.