

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

CONSIDERING THE DETAILS

Bob Perrin

Getting Started with Embedded PCs

Sure, embedded 'x86 products are flooding the embedded-systems market, but Bob still had a few questions about embedded PCs, so he took the beginner's route and ordered the Flashlite V25+ development system from JK Microsystems. This month, Bob considers the details of getting started with an embedded PC.



After attending the Embedded Systems Conference (ESC) in San Jose, it became clear to me that embedded 'x86 products dominate the embedded-systems market.

Sure, there are still dozens of small companies making custom hardware and development tools based around other processors. For example, Parallax is still peddling the PIC-based Basic Stamp, Z-World still produces 8-bit Zilog Z180-based controllers, and Wilke has their 16-bit BASIC Tigers. Thumb through the Idea Box ads in *Circuit Cellar* magazine and you can still find a handful of companies selling non-'x86-based custom hardware and software. But, the 800-lb gorilla on the block is the embedded 'x86.

I have experience with all the above products and they're all fine in their respective niche, but after the ESC, I decided to broaden my horizons to include the dominant technology in the embedded-controller marketplace.

I went looking for a good how-to-use-an-embedded-PC text. I found many articles, but they were either over my head or devoid of details (or both).

At long last, I found and purchased Ed Nisley's book, *The Embedded PC's ISA Bus: Firmware, Gadgets and Practical Tricks* [1]. This is an excellent text and I highly recommend it to anyone attempting to build ISA cards, but it still didn't fully satisfy my curiosity.

How do you get an embedded PC without a keyboard, monitor, or disk drive to boot and run your application code? How do you control the resources on an embedded PC from a C program? I decided the best way for me to answer these questions was to buy an embedded PC and tinker. After all, the best learning is accomplished by doing.

WHAT IS AN EMBEDDED PC?

The first issue to address is the difference between an embedded PC and an embedded 'x86. An embedded PC falls into a subclass of the more generalized embedded 'x86 universe.

An embedded PC should run one or more of the same OSs that a normal desktop PC runs. The development tools used to write code for an embedded PC should be the same tools used to write code for a desktop.

For example, Tern will sell you 'x86 hardware and Borland C for software development. However, Tern's products don't support DOS. So the executable image generated by the Borland tools must be handed off to a relocater written by Tern. I don't consider these products to be truly embedded PCs.

The idea of taking executable code generated by DOS-based tools for a DOS-based target system and relocating it in memory on a target system without DOS just gives me the willies.

This isn't meant to be a slap at Tern's product line, which is quite broad and successful (just check out their web site to see how NASA used their 'x86s). It's just to say that, if I'm going to use DOS tools to generate code for a DOS system, I would feel better if

the target system had DOS running on it.

An embedded PC must make resources commonly found on a desktop PC available to the application-code. These resources include, at a minimum, a file system and a console. In the world of embedded PCs, the file system is often not kept on a mechanical disk. A flash memory-based or battery-backed SRAM file system is the usual storage medium. The system console is often a serial port.

In my opinion, an embedded PC need not have an expansion bus, such as PC-104 or Compact PCI, although most do sport some expansion bus.

SELECTING AN EMBEDDED PC

When I began my quest to find my first embedded PC, I was concerned with price, service, and simplicity. I didn't want to pay a mint for a circuit board, but at the same time, I wanted to get good technical support. Furthermore, I wanted an embedded PC that was fairly simple. I didn't want one that was just a desktop PC shrunk in size.

Ampro, Adastra, WinSystems, and Ziotech all have offerings in the embedded PC market, but I finally settled on the Flashlite V25+ board from JK Microsystems. The development system was \$159 and included everything I needed to get up and running. Certainly the cost was right, but what about customer support?

When you call JK Microsystems, you get to talk to an engineer. And when I say engineer, I don't mean some sales engineer, or a customer-support technician, I mean a seasoned design engineer. Often the person answering your questions is the designer of the board you're asking about. That was my primary reason for selecting the JK embedded PC. I knew I could get tech support without long phone delays or hassles.

The Flashlite is a single-board PC with only six ICs on the PCB. These are the NEC V25+ MPU, an SRAM, a FLASH, a PAL and two RS-232-level converters. Although simple, the board provides a fairly extensive I/O set.

There are two RS-232-level serial ports on the V25+ Flashlite. There are 0.100" headers that pinout 32 digital

I/O points and much of the processor's bus. Eight of the digital I/O points have variable input thresholds that are set in software. This arrangement provides a method to implement a crude ADC.

UP AND GOING

The Flashlite does not have an onboard regulator, it requires a single 5-V supply and draws about 200 mA. I used my trusty benchtop supply to provide power.

The board has a PC-compatible BIOS in the flash memory. The rest of the flash memory is used as a disk. The flash-memory disk is split between A: and B:. The A: disk is the boot disk, is read only, and contains DOS. The B: disk is 128 KB and is where user applications are stored.

On booting, the V25 will start in the BIOS, then move on to DOS, then run *autoexec.bat* off the A: disk. The last line of the *autoexec.bat* file will attempt to run a file named *STARTUP* on the B: disk. *STARTUP* may be an *.exe*, *.com*, or *.bat* file.

The Flashlite uses a serial port (serial port 0) as the console. Just plug in any serial terminal or a PC running a terminal program, such as HyperTerminal. As the Flashlite comes up and DOS begins to run, the console gets a startup message and then a B:\> prompt is displayed.

If you have the Flashlite's console port attached to a terminal, you can use the terminal to do all the normal things you can do with a desktop running DOS. Commands such as *DIR*, *RENAME*, and *COPY*, all work just like you would expect. There's a utility provided on the flash memory-based read-only A: disk, called *UP.COM*.

This program is used to transfer files from the desktop PC on the Flashlite's console port to the B: disk on the Flashlite. This is the sole method used to transfer applications and data from the desktop development system to the target (Flashlite). *UP.COM* uses an X-MODEM protocol to transfer the files.

HELLO WORLD

Once the Borland C tools are installed on the development system (desktop PC), you can generate *.exe* files that will run on either the desktop

PC or the Flashlite.

The first program I tried was the classic *hello world* Program. When setting up the Borland C project, I had to tell it to generate code for a "DOS platform." I selected a "Compact" memory model, just to be on the safe side. Then I keyed in the following code:

```
#include <stdio.h>
void main( void )
{
    printf("\nhello world");
}
```

I compiled the code and ran *hello.exe* on my desktop PC in a DOS box (under Windows NT 4.0). Sure enough, the code printed the classic "hello world" text. This is exactly what I expected since *printf* sends its output to *stdout*, which, on a DOS machine, goes to the system console.

The next step was to upload the *.exe* to the Flashlite. I fired up HyperTerminal and hooked up the Flashlite's "Serial Port 0" to my desktop's free serial port (COM2). I set HyperTerminal's connection to 9600 bps, 8 data bits, 1 stop bit. I turned on the Flashlite and the JK Microsystem's startup message appeared in my HyperTerminal window. Communications had been established.

I started *UP.COM* on the embedded PC. I was prompted for the name of the file I was about to send from the desktop to the embedded PC. I keyed in *hello.exe*. Actually, this is just the name that *UP.COM* will give the file on the Flashlite's B: disk. This name doesn't have to be the same as the file name on the host.

Next, I selected "Send File" from HyperTerminal's Transfer menu. HyperTerminal prompted for the file I wanted to send and the protocol I wanted to use. I selected X-MODEM as a protocol and *c:\bc\proj\hello\hello.exe*. The file went across without a hitch. After the transfer had completed, using HyperTerminal as the Flashlite's console, I was able to execute a *DIR* command on the target system. My *hello.exe* file was on the B: FLASH disk. I executed it on the embedded PC, and sure enough, the "hello world" text

Listing 1—Just as the "hello world" program establishes basic confidence in the tool path, this program instills confidence that the V25+'s SFRs can be manipulated with C.

```
#include <stdio.h>
#include <dos.h>

char far * p0 = MK_FP(0xF000, 0xFF00);
char far * pm0 = MK_FP(0xF000, 0xFF01);
char far * pmc0 = MK_FP(0xF000, 0xFF02);

void main (void)
{
    printf("\nClearing all the bits on the V25's Port P0");

    // for information on the data bytes below
    // and the meaning and addresses of the SFRs
    // Consult the NEC V25+ User's Manual

    *pmc0 = 0x00;    // no special control features being used
    *pm0 = 0x00;    // set all bits on port0 to be outputs
    *p0 = 0x00;     // clear all bits on port0
}
```

printed on the Flashlite's console (my desktop PC's HyperTerminal window).

Now that communications were established between the host and the target, and I understood the basic tool path, it was time to start exploring the other hardware resources on the single-board embedded PC.

CONTROLLING RESOURCES

The V25+ chip from NEC provides two serial ports, three bidirectional parallel ports, and a byte-wide set of inputs with programmable thresholds. The Flashlite pins out all of these resources on headers. The problem is getting C code to talk to these ports.

Unfortunately, the 33-page user's manual for the Flashlite is a bit weak on Borland C examples. This setback, coupled with a lack of Borland C drivers, makes the board a bit of a mystery to the novice. Fortunately, NEC provides a 326-page tape-bound user's manual for the V25+, free of charge. This document is invaluable for Flashlite users.

After ordering, receiving, and reading the NEC user's manual, it was clear how the V25+ resources worked. The processor's peripherals are straightforward. In fact, after reading NEC's documentation, I found the 33-page JK manual to be a handy reference. The next step was to bang out a bit of C code.

All of the peripherals on the V25 are accessed through special function registers (SFRs), just like an 8051. The V25 memory maps its SFRs. This arrangement means dereferencing a char pointer or using Borland's *pokeb* provides access to the SFRs. I chose the char pointer route.

The trick to getting a pointer to point at one of the V25+'s SFRs is to specify the pointer as a "far" pointer. Borland C seems to treat the modifiers *far*, *_far*, and *__far* the same. When I tried to find out why three different modifiers existed, I was decidedly unimpressed with the documentation available on older versions of Borland C. I scoured the local bookstores for texts on Borland C version 4.52, but never found one that seemed complete.

In the end, I just settled on using the *far* modifier without an underscore. My code compiled and ran fine.

To define and initialize a far pointer, I used the following syntax:

```
char far * p0 = MK_FP(0xF000,
0xFF00);
```

The *MK_FP* macro takes a segment (0xF000) and an offset (0xFF00) and turns them into a far pointer. That value is assigned to the pointer *p0*. In this case, the far pointer points to the memory location where the port0 data register resides inside the V25+. The

MK_FP macro is found in *DOS.H*.

To access the contents of the *p0* SFR, simply dereference the pointer. For example, to write data on port *p0* (assuming the data direction and control registers have been properly set up), just use **p=DATA*, where *DATA* is the byte-wide value you wish to write to the port.

Listing 1 shows the first program I wrote to access the digital I/O pins on port 0. This program simply sets all eight pins of port0 to a logic low (GND) value. The default state of port0 upon system RESET is High-z.

To verify that my program worked, I hooked up my trusty Fluke 77 to one of the I/O pins on port0. I used a 10-kilohm resistor tied to +5 V to pull the line high. Next, I tied the resistor to ground and pulled the line low. This process verified that the port0 I/O line was in a high-impedance state after the Flashlite was reset. Then I ran the code shown in Listing 1. Sure enough, the port pin was pulled hard low.

The code in Listing 1 is the equivalent of "hello world" as far as accessing the V25+'s SFRs. Once I had this little piece of code working, I knew I could talk to the SFRs and could therefore control all of the onboard peripherals.

THANK YOU, INTEL

The code in Listing 1 uses the *MK_FP* macro to initialize the pointers. *MK_FP* accepts two parameters: a SEGMENT and an OFFSET. Some people may not be clear on exactly what these parameters are. Allow me to explain.

For some reason, back in the 1980s, Intel decided that having a memory space accessed by a single 20-bit address register was not the most optimal method. Nevermind the fact that Motorola's MC68000 family had a 24-bit address space accessed by using 32-bit internal address registers (the most significant 8 bits were not put out on the external 24-bit address bus). Instead, Intel decided to use two 16-bit values (a SEGMENT and an OFFSET) to compute a 20-bit physical address.

The computation is simple. Left shift the SEGMENT a nibble and add it to the OFFSET. The least significant 20 bits of the sum comprise the physical address. Although that all seems pretty

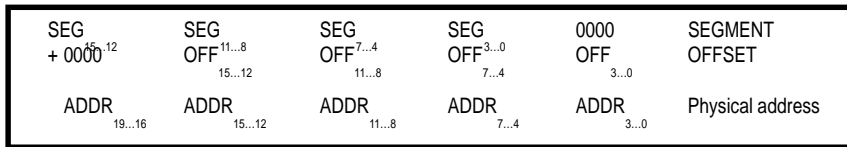


Figure 1—The SEGMENT + OFFSET arithmetic is simple, once you see how the computation is done.

easy, consider that there is no longer only one unique representation of a physical address. As long as the bits in the SEGMENT and OFFSET values add to the desired 20-bit sum, that combination of SEGMENT and OFFSET registers is a legitimate representation of the physical address.

Figure 1 shows how the SEGMENT and OFFSET are aligned to compute a physical address.

My understanding of Intel's reasoning for this scheme was that the SEGMENT/OFFSET method would make multitasking easier. Segments could be assigned to separate tasks and, when required, segments could overlap. Unfortunately, 15 years later we still have to live with the legacy of this kooky idea.

APPLICATION TERMINATED

Embedded PCs can seem a bit intimidating at first look, but like most things, once you have one on your desk to play with, the mist lifts and the picture becomes clear. Starting with a simple board and taking things one step at a time makes the learning easier. The bottom line seems to be, if you can write a bit of code in C and can read a datasheet, you can use an embedded PC.

Over the last ten years, Bob has designed instrumentation for agronomy, soil physics, and water activity research. He has also designed embedded controllers for a variety of other applications. For more technical resources and articles, visit Bob's online library at www.engineerbob.com. You may reach Bob with comments and questions at bob@engineerbob.com.

REFERENCE

[1] E. Nisley, *Embedded PC's ISA Bus: Firmware, Gadgets, and Practical Tricks*, Peer-to-Peer Communications, San Jose, CA, 1997.

SOURCES

Embedded PCs

Adastra Systems Corp.
(510) 732-6900
Fax: (510) 732-7655
www.adastra.com

Ampro Computers, Inc.
(408) 360-0222
Fax: (408) 360-0222
www.ampro.com

JK Microsystems
(530) 297-6073
Fax: (530) 297-6074
www.jkmicro.com

Parallax, Inc.
(888) 512-1024
(916) 624-8333
Fax: (916) 624-8003
www.parallaxinc.com

Tern, Inc.
(530) 758-0180
Fax: (530) 758-0181
www.tern.com

Wilke BASIC Tiger

Kg Systems, Inc.
(800) 292-4303
(973) 515-4664
Fax: (973) 515-1033
www.industrialcontroller.com/wilke/

ZB4100

Z-World
(530) 757-4616
Fax: (916) 753-5141
www.zworld.com

Ziatech
(805) 541-0488
Fax: (805) 541-5088
www.ziatech.com