

# Everything You Always Wanted to Know About SSD (but were afraid to ask)

*An embedded system is supposed to operate like an appliance. When you turn on a microwave oven or TV, you don't expect to wait half a minute for it to boot its operating software from a disk drive. You want it to perform its intended function instantly. Such systems, if they contain microcontrollers, normally load their software instantly from a ROM, not from a disk drive.*

There are other reasons why you may not want an embedded system to operate from a conventional disk drive. In applications with critical data integrity requirements, the "soft" read/write errors that are relatively common with disk drives are totally unacceptable. Disk drives don't work over wide temperature extremes and so are usually limited to indoor, temperature-controlled environments. Shock and vibration can also be problems. Size, power consumption, and heat generation are also reasons to avoid using disk drives in embedded systems.

For these reasons, it may seem like a good idea to ROM your embedded-PC application rather than operating it out of a disk drive like a normal PC.

You're probably getting set to hear all about "ROMing" a PC/104 based application. Expecting tips on how to embed your software by splitting it into independent code and data blocks that will go into separate ROM and RAM memory devices? Looking for compiling and linking techniques, and other nifty tips on embedded programming? Actually, you're about to see why you really don't want to ROM a PC/104 application.

## THE ITSS PRINCIPLE

You may have used microcontrollers on other projects. In that case, you probably ROMed your application. But just because that's how it's done on a microcontroller doesn't mean it's the right way to embed your software on an embedded-PC. Of course, I'm assuming you really want to harness the full potential of PC compatibility.

So, what's your biggest reason for using an embedded PC? If you're reading this column, you've probably got a list of reasons for embedding a PC. OK, let me guess what's #1 on that list. The ITSS Principle. You haven't heard of the ITSS Principal? You know the KISS Principle: keep it simple, stupid!

The ITSS Principle is the answer to "Why are embedded-PCs used for so many embedded projects today?" It's not that the PC is viewed as a lean-mean-fast computing machine! Rather, it's to take advantage of the gazillions of dollars constantly being invested in all manner of software support for the desktop PC market. In other words: "It's the software, stupid."

You want to simplify and enhance your embedded

project by tapping into the vast storehouse of PC operating system, driver, and development software, right? If so, remember this: to reap the benefits of PC compatibility, stay PC compatible!

The PC was designed to be a disk-operating-system (DOS) machine. PC software is always transferred from disk into system DRAM, where it actually runs. (Even the BIOS is generally "shadowed" in DRAM for faster execution, nowadays.) So, if you choose to ROM your embedded application (and perhaps even DOS), you abandon the PC "standard." In that case, don't be surprised when you lose access to tons of off-the-shelf device drivers, function libraries, and application programs, which rely on running in a DOS environment.

If you insist on ROMing your PC/104 application the traditional microcontroller way, you'd better be ready for the traditional microcontroller development headaches and limitations.

## EMULATION IS THE SINCEREST FORM OF FLATTERY

Instead of ROMing the application, use "solid state disk" (SSD). What's SSD? SSD "emulates" normal disk drives. With SSD, a software driver transforms accesses to a normal disk drive (floppy or hard disk) into accesses to some form of memory. It's a lot like a RAM disk, except that an SSD is typically used as a boot drive.

Since practically every program on a PC makes its disk accesses using DOS or BIOS functions, the system can't tell the difference between the SSD and a real disk drive. For this reason, SSD-based embedded system development doesn't require any special expertise - assuming you take advantage of one of the readily available forms of plug-and-play SSD discussed below.

You can develop your application on a PC with normal disk drives. You don't even need to know how to write ROMable code. You don't have to be aware of how the PC's memory space (ROM and RAM) is organized or used by DOS! You just develop and test your application using your favorite operating system, programming language, and other software tools, just like it's going to run on a conventional disk drive. Then, once you're satisfied with how the application operates from regular disk drives, you transfer the application to the SSD device.

The actual procedure to do this depends on the kind of SSD you're using. But it's normally fast and easy. After you transfer the application from a "normal" drive into an SSD, you remove the normal drive and reboot the system. The system should now boot and run from SSD just like it did from the regular drive. That's all there is to it! SSD converts "software" to "firmware" - instantly and painlessly.

## HOW TO MAKE AN SSD

There are quite a few SSD approaches. Many PC/104 embedded-PCs (such as Ampro's CoreModule and Little Board products) have onboard sockets where you can plug in SSD devices, along with driver support, built into the BIOS, for them to emulate a bootable A: or C: drive. There are also PC/104 SSD modules of various types. You can also use PCMCIA cards as SSDs. There are even SSD drives that look and act like ordinary IDE or SCSI disk drives, but are based on non-volatile memory instead of magnetic media.

In general, what's needed is some type of nonvolatile memory device along with an appropriate SSD software driver. Let's take a look at some of the many SSD options available. But before looking at specific products, let's review the technologies and interface architectures used.

There are three main memory device technologies for SSDs: EPROM, NOVRAM, and "Flash."

### EPROM

As an SSD technology, EPROM has some serious limitations. EPROMs are, of course, usable only as read-only SSD drives, so they don't solve the problem of what to do when you need to be able to write data into an SSD during system operation. Since EPROMs generally can't be erased and reprogrammed while they're plugged into the target embedded-PC, you're going to need to program them in an EPROM programmer and then plug them into the embedded system.

Obviously, this makes it a nuisance (and sometimes expensive) to do field updates to an embedded system's software. On the other hand, the per-unit cost of EPROM SSD can't be beat. So, when in-system writability isn't required and cost is critical, EPROM SSD may be the answer.

### NOVRAM

Using non-volatile RAM ("NOVRAM") memory as an SSD is probably the easiest approach to deal with. It's simple, requiring one or more RAM chips (usually 32-pin DIP) and some form of nonvolatile controller and backup battery. Since it's fully read-write, NOVRAM SSDs can be programmed directly within the target embedded-PC and can be used like ordinary read-write disk drives (provided you have the right SSD driver software).

If you're designing NOVRAM sockets directly into your application, you do have to deal with how to make an SRAM nonvolatile. You can buy the necessary logic to turn an SRAM into a NOVRAM within a single tiny chip. All you need to do is add a battery and hook it up correctly. But don't underestimate the technology in that little chip! It's critical that the SRAM be protected from

accidental write strobes during system power cycles and that SRAM power be properly switched to and from the backup battery at the proper instant.

An easier way to get NOVRAM is to use special 32-pin SRAMs that have built-in backup batteries and control logic. These are available from Dallas Semiconductor, Benchmark, and others. One manufacturer even makes one with replaceable, snap-on battery.

While NOVRAM has the advantage of read-write simplicity, it also has a couple of significant disadvantages. One is cost. SRAM is the most expensive form of memory and can be many times as expensive as EPROM.

Another is limitation temperature. Batteries have limited operating temperature range, so the battery in an embedded-PC's NOVRAM SSD could prevent it from being used in certain applications. Speaking of environmental restrictions, there are also environments where batteries are simply not allowed, due to the corrosive (and sometimes explosive) chemicals they contain.

Of course, another problem with batteries is that they don't last forever. Systems containing NOVRAM SSDs will eventually need to have their batteries replaced. Which can be inconvenient and sometimes costly. It can also mean expensive system "down time" and loss of valuable data.

So if NOVRAMs have all these problems and EPROMs are read-only, is there any other memory technology that really works well as an SSD? Which brings us to . . .

### Flash

Flash memory - let's just call it "Flash" - is closely related to EPROM. Except the clever semiconductor device scientists figured out how to use quantum effects to make an EPROM that can be erased and reprogrammed electrically. No need to use a UV lamp to erase the contents like with a normal EPROM.

"Isn't that EEPROM?" you might be thinking. Not exactly. EEPROM was the first form of electrically erasable/programmable ROM. But boy, was it expensive! Worse than SRAM! Too expensive to ever become popular. What's special about Flash is that it is only slightly more expensive than EPROM.

Unfortunately, even with the help of quantum mechanics, Flash isn't as easy to erase and reprogram as an SRAM. But given the low cost of Flash, so what if it takes a little extra effort! Flash device erasing and programming requires careful attention to a number of details. For one thing, the data and write control signals must be sequenced in just the right way, or the data either won't be recorded or won't be programmed fully into the memory cells.

Also, Flash has an unusual property for a semiconductor memory: it wears out after a specified number of erase/write cycles. You can't, for example, use a single location within a Flash memory for constant updates to a parameter because it will eventually fail to record the data you are writing. Fortunately, Flash devices are rated for hundreds of thousands of cycles, so there are approaches to managing the lifecycle of

**Ad**  
**US Software**

Device	System Interface	Size (in.)	Maximum Capacity	Sustained Read Rate	Sustained Write Rate	RTOS Support
EPROM chip	32-pin DIP socket	1.8 x 0.6 x 0.3	1 MByte	fast	(read only)	No
NOVRAM module	32-pin DIP socket	1.8 x 0.6 x 0.4	0.5 MByte	fast	fast	no
NOR Flash chip	32-pin DIP socket	1.8 x 0.6 x 0.3	0.5 MByte	fast	(read only)	no
DiskOnChip 1000 (NOR)	32-pin DIP socket	1.5 x 0.6 x 0.3	2 MBytes	fast	slow	no
DiskOnChip 2000 (NAND)	32-pin DIP socket	1.8 x 0.75 x 0.3	12 MBytes	fast	medium	yes
1.8 in. IDE Flash drive	IDE interface	3.0 x 2.0 x 0.4	240 MBytes	medium	medium	yes
CompactFlash	IDE interface	1.4 x 1.7 x 0.2	20 MBytes	medium	medium	yes
PC/104 Flash Disk	PC/104 bus	3.6 x 3.8 x 0.6	32 MBytes	fast	slow	yes
MiniModule/ SSD + EPROMs	PC/104 bus	3.6 x 3.8 x 0.6	4 MBytes	fast	(read only)	no
MiniModule/ SSD + SRAMs	PC/104 bus	3.6 x 3.8 x 0.6	2 MBytes	fast	fast	no
PCMCIA-ATA	PCMCIA slot	3.4 x 2.1 x 0.1	300 MBytes	medium	medium	yes
PCMCIA linear flash	PCMCIA slot	3.4 x 2.1 x 0.1	64 Mbytes	medium	medium	limited
PCMCIA linear NOVRAM	PCMCIA slot	3.4 x 2.1 x 0.1	64 MBytes	fast	fast	limited

Table 1. Table of SSD device alternatives

Flash memory that make it useful as an SSD medium. The lifecycle of Flash is called "endurance."

A common method used by the software drivers is to read the data back after writing to each location in a Flash device, to verify that it has been written successfully. As a Flash location begins to wear out, you might need to write data to that location multiple times to get it to program successfully! Eventually, it fails completely. This technique extends the life of Flash memory.

If a Flash device needs to be written many times during its life, it's critical that the writes to it be evenly distributed throughout the device. Otherwise, it may begin to wear out prematurely. Sounds a lot like rotating the tires on your car, doesn't it? This process (in Flash) is called "wear leveling" and is a critical function of Flash support software.

As if all this wasn't enough of a limitation, another problem with Flash is that you can't re-write a single location, but must erase and then re-write some minimum block size. That block size has been steadily getting smaller, as Flash technology has evolved, so it's now beginning to disappear as a key issue. But, it started off as an entire chip - which, you can imagine, made for some interesting SSD implementation challenges.

The method used to deal with this issue is to track "clean" and "dirty" blocks. New data is always written into a "clean" block and blocks with data that is no longer current are marked "dirty." After a while, a Flash device can be full of "dirty" blocks even if it appears nearly empty from a DOS perspective. When this happens, the Flash management software needs to consolidate the widely scattered fragments of good data together, so room can be made for new clean blocks. This process is called "garbage collection." The Flash

management software carefully maintains tables of which blocks are "clean" and which are "dirty".

Doing all this, while maintaining complete integrity of all the good data, is tricky. My advice: "Kids, don't try this at home!" Fortunately, there are several sources of off-the-shelf "Flash File System" software which do a good job making Flash work reliably. A popular one is "TFFS" (True Flash File System), from M-Systems.

### **NAND vs. NOR Flash**

The Flash memory discussed so far is "NOR" Flash. There's a new development in Flash, called "NAND". The two names have to do with how the logic inside the devices is structured. Although this discussion won't address the internal differences, it will call your attention to a couple of key functional issues that effect how they're used.

NOR Flash is accessed a lot like EPROM or SRAM, except for the restrictions mentioned earlier. That is, you put an address on its address pins and you read or write data on its data pins.

NAND Flash is accessed differently, in more of a serial-bit-stream manner. In this sense, NAND Flash acts a little more like a disk drive. NAND Flash was actually developed to serve as a disk-like storage medium for digital cameras and hand-held computers. So it's no surprise that NAND is quicker to program, has conveniently small erase block sizes, and boasts a high erase/write endurance. All this makes NAND well suited for SSD.

Now for the bad news. NAND is just as tricky to deal with as NOR, but for other reasons. For one thing, you don't "talk" to it like an SRAM or EPROM. You need special circuitry to interface it to the system. Another problem: NAND devices come with defects, just like hard disks. So you need to test for, and map out, the defects. Like disk drives, it's also useful to include error detection and correction using CRC logic. All this means, you need a special controller chip - and special software - to effectively use NAND Flash chips as SSDs.

## **SSD SOLUTIONS**

Are you now so intimidated that you're ready to turn back the clock 20 years, and return to using ROM-based microcontrollers? Don't despair! The good news is that there are quite a few easy-to-use, "plug-and-play" SSD solutions that stand ready to serve your PC/104 embedded-PC needs. Let's explore some of the options.

### **Byte-wide devices**

Most PC/104 embedded-PCs include one or more 32-pin "byte-wide" memory device sockets. These were originally intended for use with simple DIP or PLCC EPROM and SRAM chips. Using these devices, the capacity of a 32-pin socket is limited to 1 megabyte for EPROMs and 512 kilobytes for SRAMs. Depending on the SSD driver software, multiple byte-wide sockets can be combined into a single DOS drive letter, for larger SSD capacity. As 32-pin NOR Flash surfaced, it has usually been supported like EPROM, except that the device can be reprogrammed - usually on a full device

basis - inside the system.

The simple byte-wide SSD is pretty limited in its capacity, which is comparable to that of a floppy diskette. Although there are certainly PC/104 applications that can run entirely out of one or two simple byte-wide chips, storage requirements have really exploded as CPU performance and memory availability have skyrocketed.

A nice solution to the limitations of the simple 32-pin byte-wide SSD has been provided by M-Systems, in the form of their "DiskOnChip" Flash module. The DiskOnChip is the same size as a 28- or 32-pin DIP EPROM. Yet this compact device can contain up to 2 megabytes of NOR, or up to 12 megabytes of NAND, Flash. It contains the necessary circuitry to "look" just like a simple DIP EPROM to the PC/104 CPU. The DiskOnChip comes complete with TFFS and other support software for formatting, operation, and maintenance. Future capacities are expected to reach 72 megabytes.

Neither one disadvantage of the original NOR version of the DiskOnChip (DOC1000) was the relatively slow write cycle time and the periodic long "garbage collection" latencies. This is no longer an issue with the new higher capacity version of the product (DOC2000), which benefits from the fast write cycles and small erase block sizes of NAND technology.

### **Drive-like modules**

Several companies now offer small size (1.3 and 1.8 inch formfactor) Flash drives that look precisely like small IDE hard disk drives. They have the same physical footprint, the same mounting holes, the same interface connectors, and the same electrical functional interface as their magnetic media counterparts.

IDE Flash drives contain a microcontroller that takes care of all the required Flash management functions such as wear leveling, so no special driver software is required. The IDE Flash approach was pioneered and popularized by SanDisk (formerly SunDisk) and is available in capacities from 4 through 300 megabytes. This approach offers the highest capacities.

To use these tiny IDE Flash drives, you just install and operate them exactly like normal (magnetic media) IDE drives. Nothing could be simpler! One possible catch is that you need to have an IDE interface in your PC/104 system to use them. However, most PC/104 CPUs now include IDE interfaces at no extra cost, so this is not a problem.

An important advantage of IDE Flash drives is that they are operating system independent, since all operating systems provide IDE hard disk support and the Flash management functions are handled directly within the drive. Also, this means you can replace a drive in the future without worrying whether you have the proper software driver for the specific Flash technology inside the drive. In fact, you need not be concerned at all with what's inside the drive!

### **PC/104 SSD modules**

Why not place EPROM, NOVRAM, or Flash SSD devices on a dedicated PC/104 module? This, too, is readily available in a variety of formats. PC/104 SSD

modules come with four or more 32-pin byte-wide sockets for individual plug-in EPROM, SRAM, or Flash chips (e.g., the Ampro "MiniModule/SSD"). They are also available with neither soldered on NOR Flash memory for up to 32 megabytes SSD capacity (e.g. the M-Systems "PC/104 Flash Disk").

### **PCMCIA memory cards**

Last, but certainly not least, is PCMCIA. It is no surprise that this well-known standard, whose acronym comes from "Personal Computer Memory Card International Association," offers a broad range of SSD capabilities. In fact, every memory technology already mentioned is available on PCMCIA cards (now beginning to be called "PC Cards", by the way). This includes NOVRAM, EPROM, and various kinds of Flash.

PCMCIA cards have several advantages, as well as a number of disadvantages. On the positive side, perhaps the main advantage of PCMCIA cards is that they are removable. They can be inserted and removed while the system is running, just like floppy diskettes. This makes them useful for storing data, loading parameters, and updating firmware. Another plus is that because PCMCIA cards are so popular as options for laptop PCs, they are sold by all major computer retailers and are therefore available at competitive prices.

One disadvantage of using PCMCIA cards for SSD is their need for a special PCMCIA card slot. You can't just plug them into a byte-wide memory socket or cable them to an IDE interface. Instead, you need a PCMCIA controller or interface module. This means extra cost and complexity. If you don't need removable media, you might prefer using a DiskOnChip or an IDE Flash drive - unless you need NOVRAM for its speed and truly unlimited re-writability.

Incidentally, PCMCIA offers two different Flash card configurations.

One, called "PCMCIA-ATA," is another invention of SanDisk. It is functionally almost identical to the IDE Flash drive, except that it is accessed through a PCMCIA card slot instead of through an IDE interface. It provides the identical system-level interface as IDE, from a command set perspective. It is even possible to create a "passive" adapter between PCMCIA-ATA Flash cards and standard IDE interfaces (several are available as standard products). Like IDE Flash drives, each PCMCIA-ATA card has an internal microcontroller that handles all Flash management and IDE command set functions. (In case you're wondering, ATA means AT Attachment Interface, which is actually just another name for IDE, which stands for Integrated Drive Electronics.)

The other PCMCIA Flash approach corresponds to that used in the DiskOnChip. It is commonly referred to as "linear flash" because all of the Flash memory is indirectly accessible (via bank switching) to the system CPU as blocks of linear memory. The required bank selection logic is located within the PCMCIA interface controller. An advantage of linear Flash PCMCIA cards, is they don't bear the burden of an internal microcontroller. This is also the source of its biggest shortcoming. That is, with no internal microcontroller to automatically manage the Flash memory wear-leveling and

erase/write functions, these must be done by the system CPU which. This results in reduced real-time performance as well as adding a degree of operating system dependence. It also means that when you change cards (now, or in the future) you'll need to be sure your embedded system has software to properly handle the card's internal Flash technology.

### **News Flash!**

"Enough!" you say? But we can't leave the subject of SSD for PC/104 applications without taking a moment to look at the latest area of development: solid-state storage for digital cameras. Have you noticed recent ads for new film-less digital cameras? Photography is starting to go digital! Whether or not this is good news for photographers, for PC/104 embedded systems it means SSD media - especially Flash - is about to become much less expensive and much more widely available.

Unfortunately, there's no consensus on which tiny memory card standard will prevail. Sound familiar? It's the same old story. Fifteen years ago, you had to carefully analyze a dozen different manufacturers' spec sheets just to design an option for 5.25-inch floppy and hard disk drives into a computer.

Right now, there are three main contenders fighting to be the standard for miniature memory cards for the digital camera image storage. And the stakes are extremely high, as you might imagine.

The one that has made the most inroads so far is SanDisk's CompactFlash (see Photo 6). CompactFlash is an essentially a shrunken version of PCMCIA-ATA. Like its big brother, CompactFlash also has an IDE-like functional interface. So, it's relatively easy to convert from an IDE hard drive interface to a CompactFlash card socket. This makes it a great fit for PC/104 embedded systems.

While this sounds like a dream come true for embedded systems requiring miniature removable SSD cards, a fierce battle is being waged between CompactFlash and two other tiny memory card standards for dominance in the digital camera market. One, pushed by Intel, is called MiniatureCard. The other, called the "solid state floppy disk card" (SSFDC) is promoted by several Japanese and Korean memory giants. Both alternatives to CompactFlash have a similar goal: absolute minimum cost. After all, these cards will someday be sold like film -- at drug stores and supermarkets!

Unfortunately, meeting the cost goals of the mass camera market means all the Flash management "magic" must be accomplished by the host system's CPU. There's no room in the budget for a dedicated microcontroller inside the memory card! In a digital camera, that's no problem. But in a PC/104 embedded system, it means real-time operating system driver headaches and maybe even performance compromises.

One thing's for sure: you'll be hearing a lot about these new Flash memory alternatives to disk drives in the future!

## PUTTING IT ALL TOGETHER

As you can see, there are quite a few choices available to you as a PC/104 based system designer. To evaluate all these options, take a look at the data in the table (below). Hopefully, you now feel more enlightened - rather than more confused - than before! But in any case, don't worry . . . it will probably all come to you . . . in a Flash!

*Rick Lehrbaum co-founded Ampro in 1983 and was the company's Vice President of Engineering until 1991. In his current role as Executive Vice President of Strategic Development, he serves as the company's chief evangelist - developing strategic alliances and communicating the company's technology vision to customers, strategic partners, and the media.*

*Mr. Lehrbaum is the author of numerous technical articles and white papers and gives seminars throughout the world on the topics of PC/104 and*

*embedded-PC technology. In 1992, Mr. Lehrbaum*

*organized the PC/104 Consortium and has been its*

*chairman since that time; he also formed the IEEE*

*P996.1 Working Group, which is transforming*

*PC/104 into an IEEE standard.*

# COME VISIT...

**REAL TIME**  
  
**GAZETTE**  
 o n - l i n e

**THE Virtual Press Room on  
 Real-time and Embedded systems at  
<http://www.realtime-info.be>**