Understanding SSDs

Over the past couple of years, solid-state drives have become one of the most sought-after PC upgrades. Let's see how they work

From reading the advertising copy, you could be forgiven for thinking that SSDs have no downsides. Much faster than traditional spinning hard drives, far more resilient to being dropped, ultra quiet, it seems the only negative anyone can point to is the price. In reality, there are a couple of issues you should be aware of when using SSDs, and why the operating system you use is suddenly very important.

When you use a PC day in, day out, you get used to how the machine as a whole runs with your particular mix of applications. Some of these applications could be compute-bound – that is, the bottleneck in using the app is determined purely by how quickly the CPU is able to run. A great example of this is converting video from one format to another – from DVD to MP4 to play on your iPod Touch, for example. No matter what you do, the speed of the conversion is entirely down to the CPU; the disk subsystems are well able to cope reading and writing data.

Some applications are different – they're I/O-bound. Your perception of the application's speed is governed by how quickly the disk subsystems read and write. An example of this is an application you don't really think of as such: booting up your PC. When you boot your PC, the boot manager has to load various disparate drivers and applications into memory from your boot disk and set them all running. Modern OSes use hundreds of these programs, drivers, and services and applications that try to help you speed up your boot times often just concentrate on minimising what gets loaded during the boot process as a whole.

Disk delays

Standard disk drives don't do brilliantly at I/O-bound applications. The reasons are essentially two-fold. The first is that the head has to be positioned to the right track on the correct platter, ready for the correct sector to rotate round under the head. This is the seek time. Once positioned, the head has to wait for the right sector to appear (the rotational delay) so that it can read the data requested.

Another delay might be caused by the disk having to be spun up, because many systems – especially laptops running on batteries – stop the platters rotating after a period of no activity to conserve energy.

The first delay was significant in the early days of disk drives when it was in the order of half a second or so, but the seek time has been continually refined so that it's now around 10ms for standard desktop or mobile drives.

The second delay is directly proportional to the rotational speed of the platters. Over the years, the speed of drives has slowly increased, with standard mobile drives running at 5,400rpm and desktop drives at 7,200rpm – although high-end laptops (except, significantly, Apple's) tend to have 7,200rpm drives as standard these days. You can buy 10,000rpm drives for desktops,

Spotlight on… Wear-levelling

Cells in an solid state drive have a relatively short shelf-life: they can only go through a program/erase cycle approximately 10,000 times before degrading to a point where they break down. Certain operating system programs update data on disk very regularly, and if the SSD actually reused the same pages – or, to put it another way, if an LBA was permanently mapped to an SSD page – the NAND memory on that page (the whole block) would become worn out pretty quickly.

Therefore, SSD controllers incorporate some kind of wear-levelling algorithm. This algorithm first of all makes sure that the LBA to page mapping is dynamic. An LBA may refer to this page at one point, and to another at a later point in time. As we found in the main article, this is easy to ensure by writing to new pages instead of trying to update the existing page through the read-erase-write cycle.

The next thing to try and do is to ensure that pages whose data stays static (written once, read many times, such as program EXE files, and so on) are also moved around on a regular basis. This is known as static wear levelling. If this weren't done the SSD would have blocks that had never been erased and some erased often. Wear across the whole drive would be uneven. By moving static data, you improve the wear overall.

Note that USB thumb drives don't have this type of wear-levelling. Static pages on USB drives aren't moved to improve wear. This is known as dynamic wear levelling. ■

Parallel writes

The NAND flash memory used by SSDs is actually fairly slow. It's also the same stuff that USB drives are made from, and we all know how ponderous they can be. In general we may get some 30-40MB throughput to flash memory. How come SSDs have much more throughput than that?

The answer is parallelism: solid state drives will separate the data into channels. Each read or write will then operate simultaneously over all channels with the data spread out among them. If the SSD has, say, four channel controllers, a page of 4KB will be written simultaneously in 1KB chunks by the four controllers. Throughput would increase four times. Eight channel controllers would result in eight times the basic thoughput. ■

and some recommend using them for your boot drives. For comparison, the drives found in iPod Classics are 4,200rpm drives (other current iPods just use Flash memory).

Another factor in the overall speed of disk drives is how quickly you can read data from the platter and get it into RAM. Here, coupled with the move to SATA interfaces from the older PATA interfaces, the rate is proportional to the density of data on the platter, with the density slowly but surely increasing annually.

Nevertheless, the speed of hard drive technology is bound by physics and mechanics. Yes, more data is being stuffed into smaller spaces, but the overall speed of the drive is still governed by the speed of the spinning platter. In order to get more speed, manufacturers moved to different technology: flash memory.

The birth of the SSD

It had to happen at some point after flash memory became reliable and cheap enough: stuff as many flash memory chips as possible in a hard drive enclosure, add a controller and you remove the mechanical performance issues in one fell swoop. Enter the modern SSD.

The flash memory in an SSD is known as NAND-flash. There are two types: SLC (single-level cell) and MLC (multi-level cell). In SLC, each cell of memory stores one bit, and in MLC, it usually stores two bits. MLC is generally cheaper to manufacture than SLC (each cell holds double the information, so the same number of cells will hold more data), and that's what we usually find in retail SSDs.

The value of the cell is obtained by testing the cell with some voltage. The SLC cell will respond to a certain voltage: at one level, the cell is assumed to hold 0; at another level the cell is assumed to hold 1. With MLC, the cell will respond to one of four different levels of voltage, which we can denote as 00, 01, 10, and 11. Notice what this means: with MLC memory, to read a cell you have four times the number of tests to make, which takes longer. Nevertheless, we're talking orders of magnitude faster reads than the fastest disk drives can manage. Except it's not that simple.

The issue is that setting a cell to hold a value involves two different voltages too. There's the programming voltage, which essentially sets the cell to 0, and there's the higher erasure voltage, which sets the cell to 1. The programming and erasure voltages are higher than the read test voltages because they need to force electrons to tunnel over an oxide substrate between two gates.

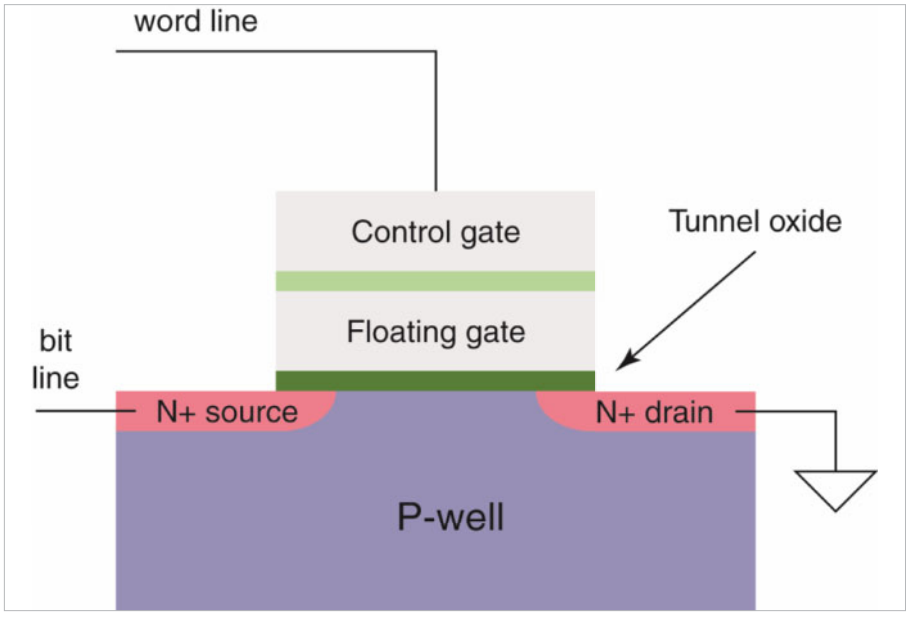
Imagine the following scenario, which illustrates how these gates work: you have two jars of water connected by a pipe. The pipe has a tap and one jar is lower than the other. Fill the top jar. When the top jar is full, the system is assumed to store a 1. If you now open the tap (this doesn't take much work at all), the water drains down to the bottom jar (in SLC terms, this is programming the system). The system is now assumed to store a 0. Now consider what you must do to set the system back to 1: you have to force the water back uphill and close off the tap, which is a lot of work (this is erasing the system in SLC terms).

In essence, when talking about NAND cells, programming is easy and erasing is hard.

Solid state deterioration

Not only that, forcing electrons back and forth over the substrate causes the material to deteriorate. Eventually it deteriorates so much that the electrons do pretty much what they want to and the cell breaks down, no longer able to store a definable state. For an SLC cell that takes around 100,000 programming/erasure cycles, but then again it only has to store two states. For an MLC cell, which stores four possible states, the rate of breakdown is much faster at about 10,000 cycles.

In reality, cells aren't programmed or erased singly. They are programmed (and read) in pages of 4KB and erased in blocks of typically 128 pages (or 512KB). Doing it like this simplifies the circuitry and the controller immensely, and file systems typically read and



▲ **Figure 1: Idealised structure of a flash cell. The floating gate stores the charge denoting 'on' or 'off'.**

write in 4KB blocks anyway. Given that erasing is more destructive to a cell than programming it, this means that erasing is done less often than programming.

However, there's a catch. If you think about it, this discrepancy between programming and erasing means that a page is only writable once. Writing a page will set some of the bits in that page to 0, and the only way to get them back to 1 again is to erase them.

So, before you can write to a page again, you have to erase the whole block containing the page. Actually, to be more specific, you have to read all the active pages in the block (except the one you want to overwrite), erase the block, and then write all the active pages back, followed by the new version of the page you wanted to write in the first place.

As you can imagine, this read-erase-write process takes a long time compared to writing a page for the first time. In fact, it's comparable to writing to a traditional hard disk.

Overwriting data

So what happens when you want to overwrite a page with some new data? The answer is that the original page is marked as 'invalid' and the new version of the page's data is written in another page entirely (making sure the links to the data's position are updated). Want to update that data again? Mark the old page as invalid, write the data to a new empty page. If you like, a page has three states: empty (it's erased), used (it contains data) and invalid (it used to contain data, but that data is now out of date and can be discarded the next time the block containing the page is erased).

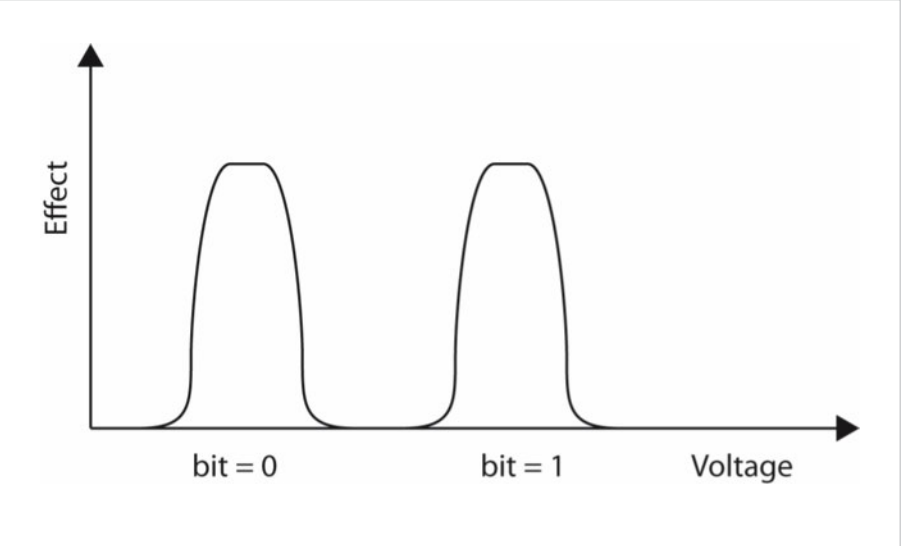
You might think this is a ridiculous way of using storage efficiently, but consider it from this angle: a block can only be erased 10,000 times before it can no longer be used. It turns out that the benefits of using NAND flash (fast, tough, silent) outweigh the problem of only having 10,000 shots at using some memory. Consider it from the level of the whole disk drive. Many people never fill up their drive. There's always plenty of room on it. An SSD controller can take advantage of that fact by writing to every page on the SSD before taking the drastic step of erasing a block.

This, incidentally, is the reason for the observations in the early days that a brand new SSD was always faster than an SSD that had been used for a while: the new drive had plenty of empty pages, and so writes happened at full speed. After a while, every page on the drive had been used, so some writes went through the read-erase-write cycle and slowed the overall speed of the drive drastically.

However, there is another wrinkle to the whole SSD story. A drive controller doesn't know anything about a file system – that's the operating system's job. The OS thinks of the drive as a linear array of pages, with the page indexes known as LBAs (logical block addresses). It then builds a hierarchical file system over that array by having files as a sequence of pages (for example, LBA 17, followed by 42, then 167, then 23) and folders as special files that contain an array of 'file entries', with each entry containing a file name and the starting LBA. The SSD controller is in charge of maintaining the mapping between an LBA number and the actual page of flash memory in the SSD.

It seems simple enough, except that when you delete a file, the OS just marks it as deleted in the file system. That's why file undelete tools work – the file hasn't been physically deleted, the data is still on the drive. It's fast because no data gets written to the SSD. If the file had just one flash page, that page wouldn't be marked as invalid. As far as the SSD is concerned, the page is still being used.

At some later time the OS will reuse a page that it knows has deleted data and write to it. At that point the SSD will do its usual work and mark the page as invalid and use another empty page. This is where TRIM comes in. Solid state drive manufacturers recognized the problems of deleted files and having deleted pages hanging around. They came up with an API, known as TRIM, that allowed the OS to tell the drive that a file had been deleted and that such and such LBAs are now invalid and can be reused. The drive could then, at the time of the file delete operation, mark all affected pages as invalid and then perform a read-erase-rewrite operation on the blocks concerned. Maybe not every time, but only if the proportion of invalid pages in the block reached a certain critical threshold. This would mean that file deletions would take longer, but this extra work would help during file write operations, which is where the user would prefer the speed. Currently Windows 7 supports TRIM, whereas Mac OS X only does for certain Apple-badged drives.



▲ **Figure 2: Diagram of how an SLC cell responds to increasing voltage. An MLC cell will have four peaks.**

Idle garbage collection

Another option that some SSDs support is known as idle garbage collection. In essence, what happens is that during moments of little to no drive activity, the controller goes through the blocks on the SSD and performs the read-erase-rewrite cycle on sufficiently 'dirty' ones (again, provided that the number of invalid pages in a block has reached some threshold value). Since this happens during idle times, the SSD is, in essence, cleaning up the drive ready for those times that you really need lots of empty pages ready to use. It's believed that Apple MacBooks use this system in some form or other (apart from those that now support TRIM, of course).

As you've seen, SSDs are as fascinating as they are fast. By all means, get an SSD, but we would recommend making sure you're using Windows 7 first. That way, you know you're using the drive to its best advantage. PCP

Julian M Bucknall has worked for companies from TurboPower to Microsoft, and is now CTO for Developer Express.
feedback@pcplus.co.uk

Defragmentation

Hard disk drives require defragmentation in order to make them as efficient as possible. Consider the case when you are reading a file stored in contiguous sectors. The head only has to be positioned once, and it can then read the data in a continuous stream. If the data were split up amongst several pages, the head would have to be positioned for every page, and then the data read. You would therefore incur seek time and rotational delay for every page.

SSDs, on the other hand, don't care. Reading from this page or that page takes exactly the same time. There is no seek time or rotational delay to suffer. It doesn't matter if a file is fragmented or not. So there's no need to defrag an SSD.

Also, if you did, you would be incurring an incredible number of read-erase-rewrite cycles in doing so. You would in fact shorten the lifespan of your SSD drive. This is the reason why Windows 7 automatically turns off defragmentation when it detects the presence of an SSD drive. ■