

White Paper
Brian Skerry
Sr. Software Architect
Intel Corporation

Storage Solutions for Embedded Applications

December 2008



Executive Summary

Any embedded system needs reliable access to storage. This may be provided by a hard disk drive or access to a remote storage device. Alternatively there are many flash solutions available on the market today. When considering flash, there are a number of important criteria to consider with capacity, cost, and reliability being foremost. This paper considers hardware, software, and other considerations in choosing a storage solution.

Wear leveling is an important factor affecting the expected lifetime of any flash solution, and it can be implemented in a number of ways.

Depending on the choices made, software changes may be necessary. Solid state drives offer the most straight forward replacement option for Hard disk drives, but may not be cost-effective for some applications. The Intel® X-25M Mainstream SATA Solid State Drive is one solution suitable for a high performance environment.

For smaller storage requirements, CompactFlash* and USB flash are very attractive.

Downward pressure continues to be applied to flash solutions, and there are a number of new technologies on the horizon.

As a result of reading this paper, the reader will be able to take into consideration all the relevant factors in choosing a storage solution for an embedded system.

Intel® architecture can benefit the embedded system designer as they can be assured of widespread hardware and software support, including critical register level interfaces to their storage solution.

§



Contents

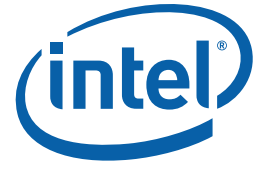
Flash Technology Background4

Flash Solutions5

Alternatives to Flash.....10

Making the Choice11

Conclusion17



Flash Technology Background

Flash technologies offer many compelling features to the designer of embedded systems. While speed and low power are obvious advantages, it is important to understand the basic types of flash and how they can be interfaced into a design.

The first two types of flash memory to consider are NOR flash and NAND flash. Though the terms NAND and NOR define how the memory cell itself are implemented, in general usage the terms refer to distinct product categories optimized for different purposes.

While NOR flash sizes are smaller than NAND flash, it has the benefit of being able to Execute in Place (XIP), meaning the code can be run directly from NOR flash as if it were RAM. This is of obvious benefit in the early stages of booting a system as the system RAM has not yet been initialized. Read access is thus random-access. However, writing to NOR flash requires that a block be erased first, with a typical block size being 64 Kbyte (Kb). Thus it cannot be used as a RAM for write operations.

NAND flash sizes are relatively much larger, but are accessed on a page basis. Page sizes of 512 bytes or 4096 bytes are typical. From a system designer's point of view, NAND flash is more equivalent to a Hard Disk Drive (HDD) in that XIP is not an option, but rather the code or data needs to be moved to RAM in order to be used.

Single Level Cell (MLC) technology refers to having one bit per memory cell. This tends to provide the fastest memory access, although at a relatively higher cost (in general). Multi-Level Cell technology in contrast, has more bits per cell thus providing a more cost effective solution. In general MLC have lower performance, although as we shall see it is important to consider the details of any implementation. MLC has at least two bits per cell, although four or more is possible. If you consider that silicon costs are related closely with die-area, it is obvious why MLC is cheaper per bit.

Erasing a block takes a similar amount of time on an SLC vs. an MLC flash. Read performance is twice as long for MLC, while write performance can be up to four times as much. Again, this is in general, and it is important to look at the specific implementation.

Aside from the technical differences between various flash implementations, it is also important to consider different operational aspects such as power, wear leveling, and reliability.

Any flash solution will have lower power than a traditional HDD. But it is important to look beyond just idle vs. active power requirements. What the system designer should bear in mind is how much power is required



to access a given amount of data. We will explore this in more detail in [Hardware Considerations](#).

Wear leveling is a key differentiator between competing flash solutions. Flash technology has a limit on the number of erase/write cycles it can undergo. To mitigate the effect of this on the overall system, different wear leveling techniques can be employed. From the designers point of view the easiest is if the flash device itself implements a robust wear leveling algorithm. Many Solid State Disks (SSDs) implement some form of wear leveling. Even some USB Flash devices have wear leveling included.

However, just having wear leveling as a feature is not enough. There are a few key questions to consider:

- How well does the wear leveling get spread across the device?
- What is the failure mode?
- Is there any indication of when the device has reached its design limit?

We will consider these and other questions when looking at different flash solutions. Another consideration is to use system software to implement wear leveling. This is discussed further in [Software Considerations](#).

Flash Solutions

In this section we explore a number of flash solutions on the market today as well as some upcoming technologies.

SPI Flash

The Serial Peripheral Interface (SPI) bus is a full duplex serial interface that is typically used for system BIOS flash devices. While the interface can be run at a variety of speeds, 33 MHz is common, with some devices, such as the Intel[®] Serial Flash Memory (S33) offering a FastRead mode (at 68 MHz).

As the system BIOS is the code that initializes the rest of the system, SPI flash is NOR-based so it can take advantage of XIP. This allows the system BIOS to configure system memory.

Being NOR-based, SPI flash devices are relatively small [note when comparing datasheets that SPI flash devices are typically specified in Mbit (Mb), not MByte (MB)]. For example, the Intel[®] Serial Flash Memory (S33) comes in 16 Mb to 64 Mb sizes (2 MB to 8 MB). One other constraint on SPI flash sizes would be the platform itself, refer to the datasheet of the chosen platform, although typically this would not be the limiting factor.



Another feature to be aware of when choosing a platform is the ability to protect the SPI flash from accidentally being written over. With the SPI flash being necessary for every system boot, having this code being corrupted is potentially catastrophic. The Intel® EP80579 Integrated Processor provides two locking mechanisms: one global, the other to specific regions.

At this point it is also worth pointing out that another potential boot device is available on Intel® platforms, specifically the FWH (Firmware Hub) operating off the LPC (Low Pin Count) bus. FWH devices are generally smaller than SPI flash devices, thus we will not consider them further other than as an alternative boot device.

Given the relatively small size of SPI flash devices, its serial interface, and its use by the system BIOS, it is not generally used as a storage device for embedded applications. If the application is small enough (e.g. 4 to 6 MB) and write requirements are minimal then it may suffice for some applications.

From a performance viewpoint, SPI Flash is not the fastest type of access, being serial. However when considering the overall time to boot a system, this is not the bottleneck. Customized boot solutions can be made quite fast (less than a second to the bootloader) with SPI Flash. The speed of the BIOS boot is due to other factors, such as time-outs while scanning the system. For embedded systems where the hardware configuration is known a priori, this time can be considerably reduced.

Note: For the rest of this white paper, we will assume that the system BIOS resides on the SPI flash unless otherwise noted.

Parallel Flash

As an alternative to the SPI serial interface, NOR flash devices are also available with parallel interfaces. This can be a good fit in a system with an available parallel interface, such as the LEB (Local Expansion Bus) on the Intel® EP80579 Integrated Processor.

The advantage over a serial interface is of course speed. Another advantage is not having to share the flash with the system BIOS. This can allow the flash to be accessed independently without unlocking the SPI flash.

Of course the available capacities tend to be rather small compared to NAND flash alternatives.

USB Flash Devices

Although USB flash devices are widely available, they are not often considered as a candidate for embedded applications. While this may be



a case of "familiarity breeds contempt", it is worthwhile to consider this option.

USB 2.0 is now widely available and, with a top speed of 480 Mb/s, the interface itself provides significant bandwidth. Most system BIOSs already support booting to a USB device, thus if using an off the shelf OS there should be little to no software porting effort.

A common complaint heard about USB flash is that the user does not want to see a USB flash dongle hanging off their embedded device. This, of course, is not the way USB flash should be interfaced to an embedded device. The device can either be on the board itself, or, if desired for maintenance, could be on a USB connector within the enclosure.

USB flash devices are available in a wide array of sizes, which should suffice for most embedded applications (with the exception of those with large data sets, i.e. more than 16 GB). As a consumer end product the prices continue to decline.

While the interface itself is not a bottleneck, the USB flash devices vary considerably in the performance they offer. Throughput for reads is of the order of the low tens of MB/s, with write performance somewhat less. Note that the performance is highly dependent on the data profile (file size, read/write mix, etc.)

Where USB devices may fall short is in the feature set. Consumer devices typically do not have the features required in an embedded application, such as wear leveling. Fortunately there are some devices that have been designed specifically for the embedded market. Consider the Intel® Z-U130 Value Solid State Drive. It includes wear leveling and ECC. Sequential read performance is 28 MB/s and write performance is 20 MB/s.

It is worth pointing out here another important metric to consider related to wear leveling, specifically the write rate associated with the useful product life. Again with reference to the Intel® Z-U130 Value Solid State Drive, it is rated at a write rate of 12 GB/day for 6 days/week, 52 weeks a year, for 5 years. If one wanted to extend that to a 7 year life cycle, with 7 days/week, this would work out to more than 7 GB/day. This is sufficient for many embedded applications, but software must be designed to minimize unnecessary writes to ensure this limit is not exceeded.

CompactFlash*

CompactFlash* was based originally on NOR flash, but now uses NAND. It was one of the first widely available formats for digital cameras, and thus has had continuing price reductions over the years.



CompactFlash has a parallel interface and can present itself as an IDE storage device. While IDE is not as available on chipsets as it once was, some systems allow it to be connected to an expansion bus, such as the LEB (Local Expansion Bus) on the Intel® EP80579 Integrated Processor. Implementation details are included in [Utilizing CompactFlash* on the Intel® EP80579 Integrated Processor Product Line Application Note](#).

The CompactFlash* standard supports speeds up to 133 MB/s. Direct Memory Access (DMA) is supported by some cards in addition to Programmed I/O (PIO). The system designer needs to ensure that their platform also supports these speeds and features.

CompactFlash cards are available in a variety of sizes, and from some vendors are available with more advanced features such as ECC and wear leveling.

As the form factor has been standard for almost 15 years, CompactFlash is ideally suited to an environment where the card may be removed for maintenance or upgrading.

Future revisions to the CompactFlash standard are described in the [CFast](#) section.

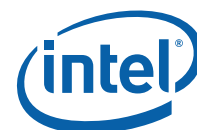
Solid State Drives

Of all the flash technologies being discussed, none has progressed as rapidly in the past few years as Solid State Drives (SSDs). Not only have capacities continued to increase exponentially, but features critical to embedded (as well as non-embedded) applications continue to be added.

Compared to the other solutions discussed thus far, SSDs have the highest capacity, but of course they are also the highest cost.

SSDs are now available in sizes that reach up to Hard Disk Drive (HDD) capacities of only a few years ago. The Intel® X25-M Mainstream Solid State Drive is an example of an MLC drive that is on par with SLC drives. The key differentiator is no longer just the underlying flash technology, but also the flash controller on the SSD. It is the flash controller that provides the gains in performance over other solutions, in some cases bridging the gap between MLC and SLC implementations. Of course the same flash controller will always provide better performance with SLC over MLC.

An advantage that all SSDs share is easy system integration, in that they are a drop-in replacement for SATA HDDs. Generally all SSDs have at least a minimal wear leveling implementation, but the system designer must pay attention to the details here. One of the important metrics for wear leveling is how well writes are distributed across each of the blocks. Here the Intel® X-25M Mainstream Solid State Drive offers industry leading performance. This directly leads to the next metric, how many



GB/day can be guaranteed. The Intel® X-25M Mainstream Solid State Drive provides 100 GB/day over 5 years.

Performance is another important factor to consider. Here one must be careful to not just look at sequential reads and writes. A useful set of benchmarks was run by Anandtech* <http://www.anandtech.com/cpuchipsets/intel/showdoc.aspx?i=3403>. As they demonstrated a real-world performance test can show some SSDs can have performance issues. This demonstrates again the importance of a well optimized flash controller.

With current SSD sizes most embedded applications should not have a capacity problem, with the exception of those with large data sets, e.g. video streaming.

Physical form factors have been following HDD standards, with 1.8" and 2.5" models available.

Future Trends

In addition to the flash technologies covered previously, it is worthwhile to discuss a few technologies coming to the market in the near term.

ONFI

The Open NAND Flash Interface (ONFI) is an open standard that defines a new interface to have interoperable NAND flash devices. Conceptually, it is similar to what JEDEC has done for DRAM interfaces. The standard defines the pin-out, form factor, command sets, and initialization procedures for a flash module. The ONFI 1.0 specification <http://www.onfi.org/> was ratified in December 2006.

ONFI has gained momentum in the server segment on Intel® architecture, and may appear in future embedded devices.

The form factor makes it well suited to embedded applications. The intent is that a chipset would support an ONFI interface, and system software could access the flash as if it were a SATA device. With a standard connector on the motherboard the system would be able to have the same design support multiple capacities.

CFast

As IDE has been widely supplanted by SATA, the CompactFlash* Association <http://www.compactflash.org/> has come up with a new standard that provides for a CompactFlash card with a SATA interface, called CFast. The CFast standard was published in September 2008. Products can be expected 12 to 18 months from that date.



The form factor will be similar to current CompactFlash cards. CFAST will emulate a SATA device, with a 3 Gb/s interface. As with CompactFlash there will likely be a wide variety of products available, some with features such as wear leveling.

From a system design point of view, with CFAST having a SATA interface it would be easy to design and test a platform with an SSD then transition to CFAST if that is a better fit (from a cost/size point of view).

Alternatives to Flash

Overview

Thus far we have considered a variety of flash technologies for embedded applications. Depending on the system requirements, other alternatives may be more suitable.

Hard Disk Drives

Hard Disk Drives (HDDs) have the advantage of ubiquitous software support, large capacity sizes, and well-understood performance trade-offs.

For embedded devices however, there are a number of disadvantages. First among these is power. While it is important to consider active and idle power measurements, HDDs also have considerable power draw on start-up. This may result in power supplies having to be significantly larger than necessary to account for HDD spin-up.

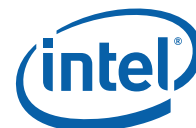
Certain HDDs have been designed with power reductions in mind. These will still have higher power requirements than SSDs, but will be less than non-optimized drives. Of course this comes at a performance penalty.

Another alternative for embedded systems would be a notebook HDD. These generally are slower (lower RPM) and have additional power saving features.

Other factors are covered in the [Hardware Considerations](#) section.

Remote Storage

Remote Storage solutions offer an interesting trade-off for embedded devices. By centralizing the storage requirements of multiple devices, each device can have lower power and cost. Note that for the purposes of this discussion we are using the term Remote Storage as a generic term, encompassing Network Attached Storage (NAS) and Storage Attached Networks (SAN).



This comes at a price however. Latency is obviously higher the farther away the physical storage device is. Reliability must also be considered. In a chassis with multiple embedded CPUs, centralizing the storage can result in overall savings in power and cost, while minimizing the latency/reliability potential downsides.

It is worth mentioning that any network storage implies at least a minimal flash solution on-board for the system BIOS at least. We take this as a given for any embedded application.

With the system BIOS in place, there are a few options for how the network storage is implemented. By far the most common is to use PXE (Pre-boot Execution Environment). The PXE client resides on the embedded device as part of the system BIOS. It uses DHCP and TFTP to boot from a remote PXE server.

Once booted, the NAS solution can then be implemented with a file based protocol such as NFS (Network File System) or SMB (Server Message Block).

An alternative to PXE/NAS would be to boot from iSCSI. This would be a form of Storage Attached Network (SAN). With the iSCSI protocol, the NIC on the embedded computer acts as a SCSI client to work with a remote SCSI storage device. It essentially allows block access to the remote device, which gives the embedded device more control over the storage device than the PXE method. This comes at a price, in that the clients need to coordinate access to the shared storage. iSCSI is widely supported by general purpose OSs as well as RTOSs.

An alternative to iSCSI is becoming more widely available, Fibre Channel over Ethernet (FCoE). While there is nothing preventing the use of FCoE in embedded applications, it is primarily being used in enterprise storage.

Making the Choice

Thus far we have considered many of the features of flash based solutions and alternatives in isolation. Here we look across the potential solutions to help guide system designers to the right solution.

Software Considerations

Embedded applications cover a broad range of potential designs, so here we will try to address the most common issues generically. Generally speaking an embedded application can use a custom software stack entirely, can be based on a commercial RTOS, or can be based on open source software. We will use the term "OS" to keep it generic, even though in some cases there might not be an OS per se. The "pain" points



will be different for each of these, thus the system designer must consider each carefully.

Note: The following assumes a flash solution has been chosen, and thus software needs to be aware of these issues.

The first point to consider is the amount of writing that the embedded system needs to do (based on application needs) versus what is imposed on it by the OS. Consider for example log files. While these are very useful in development, in deployed systems one needs to be more careful in what gets written to storage. Certainly critical failures should have enough information to enable recovery and fault isolation. But open source solutions often have many more log files than necessary. Fortunately these are easily disabled (e.g. send to /dev/null). RTOSs tend to be more optimized to begin with, as they are more aware of embedded system limitations.

The use of swap files is also something that needs to be optimized (or eliminated). In Linux* there is a parameter for this:

```
/proc/sys/vm/swappiness
```

Again, RTOSs would have these in place already.

If using an open source solution (Linux*/BSD) then the use of a file system optimized for flash is recommended. YAFFS (Yet Another Flash File System) and JFFS (Journaling Flash File System) are two prominent examples. Both of these include wear leveling in the file system itself.

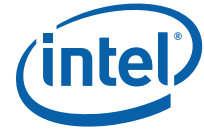
If a commercial RTOS is being used, a flash file system may be included, or can be licensed (e.g. YAFFS makes a provision for licensing in commercial RTOSs).

In the case of a custom OS, the file system is unlikely to be easily swapped out for a "flash friendly" one. In this case, the following factors need to be considered:

- Wear leveling
- Bad block management
- Track usage (i.e. to determine when approaching write limits of the device)
- Cache/coalesce writes
- Garbage collection

Some flash file systems also support ECC in software.

Another factor to consider is the failure mode. Ideally the flash device would fail in a predictable way, such that no data is ever corrupted.



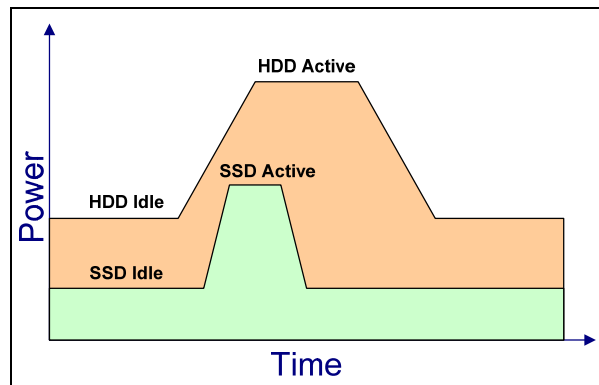
Note that the use of an SSD such as the Intel® X-25M Mainstream State Drive includes these functions in the SSD itself, thus no changes to software are required. This includes failing without data corruption. Upon failing the SSD becomes essentially a read-only device.

Hardware Considerations

For hardware the choice of storage solution will have a major impact on the power, thermal, and reliability of the embedded system.

When considering power and thermal issues, it is important not to just look at active and idle power ratings. The design must also take into account the time that the system spends in each of these states. Consider the following example in [Figure 1](#):

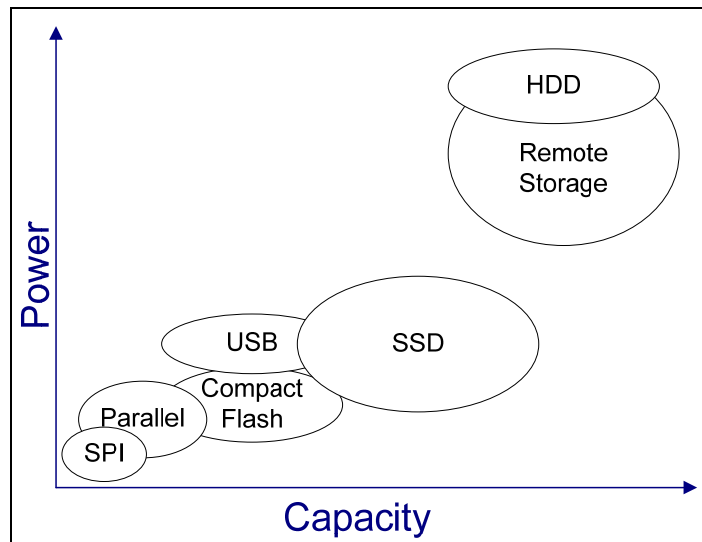
Figure 1. Power Comparison HDD vs SSD



HDD idle and active powers are greater than the SSDs. When considering the time dimension the differences are even greater. Because of the higher performance of the SSD, it spends less overall power than the HDD. This would be true even if the SSD and HDD had the same idle/active power requirements. HDDs spend considerable amount of time seeking the data, this results in much longer time being spent in the active power state.

[Figure 2](#) summarizes how the storage solutions stack up against each other with power and capacity traded off. The reason Remote Storage is shown with a broad range in power is that it is effectively shared among multiple embedded devices, thus it can be more power effective than a non-shared HDD. Also note that SSDs have a fairly wide variance in power depending on how aggressively they have been optimized.

Figure 2. Comparisons of Storage Solutions Power vs Capacity



Another power related factor mentioned previously is the drive spin-up power draw. This can end up requiring a larger power supply to cover this peak case. In general terms, this can lead to a less efficient power supply solution for the embedded system.

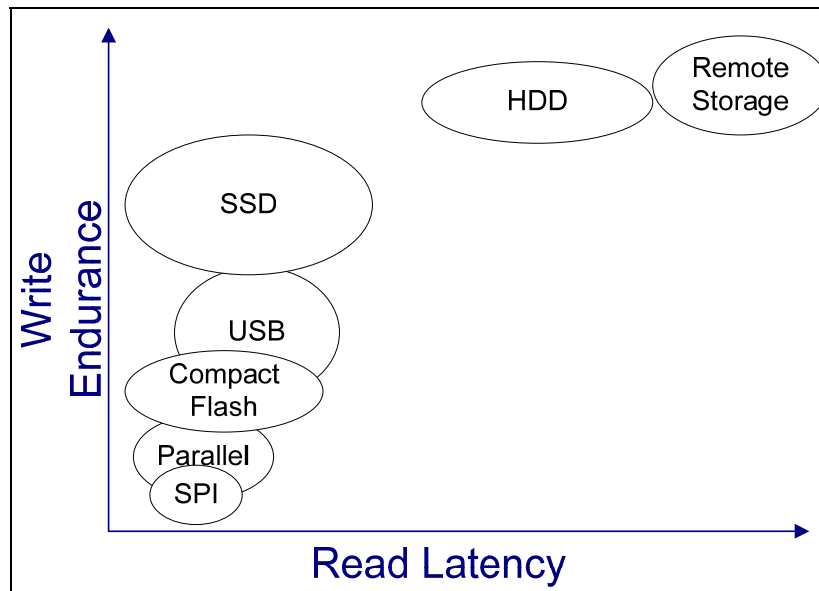
From a thermal and airflow point of view, the HDD will suffer compared to any of the alternatives. A well managed SSD can be quite power efficient, especially as many of them have been optimized to work in notebook computers.

Finally there is reliability to consider, of which there are two vectors. The first is how well the system can handle vibrations or sudden movement. For some embedded systems (e.g. automotive) this is a primary design criterion. Other applications may be more tolerant. Here the clear winner would be flash-based. The other factor to consider is the lifespan of the system. This is highly dependent on the characteristics of the application. For example, a system with a lot of write activity may pre-maturely degrade the flash system, where the HDD would not have any issues. In favor of the flash system is that a failure is more predictable (assuming adequate logging of writes is maintained), whereas HDD failure has much less predictability.

[Figure 3](#) summarizes how the solutions compare relative to one another when considering write endurance and read latency. Flash dominates for read latency compared to HDDs or Remote Storage, but suffers in terms of write endurance. The availability of wear leveling on SSDs, USB, and some Compact Flash devices helps them be better Parallel or SPI flash. SSDs in particular have put a considerable effort into closing the gap with HDDs (with wide variations between vendors).



Figure 3. Comparison of Write Endurance and Read Latency

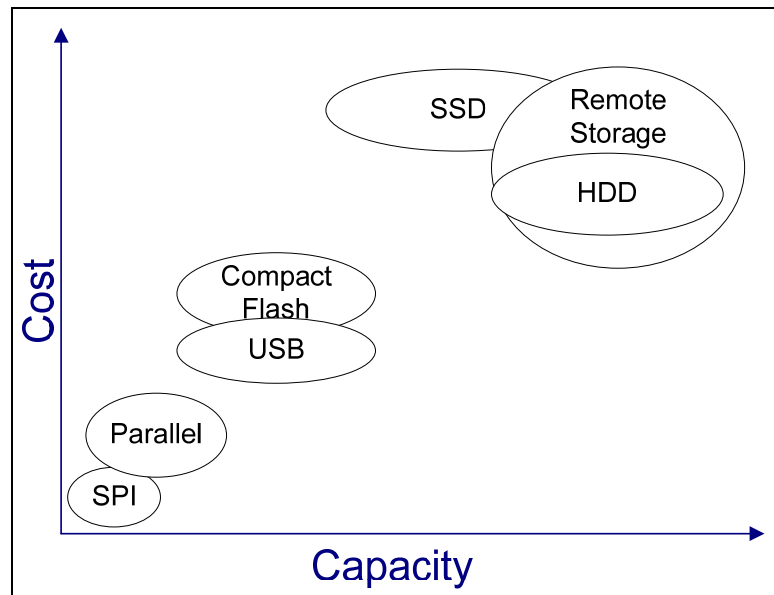


Other Considerations

We did not focus much on performance variations as this is another area where there are great differences between system needs. Any software can benefit from the lower read latency of flash, while the write performance will depend on the particulars of the flash implementation. SSDs have been optimized for fast write access, whereas many other flash types are significantly slower on writes.

Cost is of course a main driver in any choice of system components. SSDs currently command a price premium over HDDs and are not available in as large a size either. If capacity requirements are lower, then USB or Compact Flash would typically be more cost effective. Any of the flash solutions including more features (such as wear leveling or bad block management) will also have a higher price tag. These should be traded off against the effort required to implement these features in software.

[Figure 4](#) summarizes relative prices against capacities. Again, Remote Storage will depend on how many units are sharing the cost.

Figure 4. Cost vs. Capacities Comparison

Future price competition can be expected to level the playing field between SSDs and HDDs. Future technologies such as a CFAST will provide further alternatives.

Benefits of an Intel® Architecture Embedded System

Embedded systems typically have a much longer lifecycle compared to other computer systems. It is thus important that the system as a whole can continue to be supported and, if desired, incrementally improved upon throughout its operating life with minimal changes to hardware and software. It is here that an Intel® architecture-based solution offers many benefits.

Consider, for example, the transition from IDE to SATA. To ease the transition to SATA drives, Intel chipsets continued to support a legacy IDE mode so that no software changes were required (if that was desired). Similarly, with new interfaces, such as the ONFI described earlier, a backwards compatibility mode will continue to exist so it can be accessed as a SATA drive.

Because of the widespread availability of Intel® architecture and its ubiquitous support amongst third party software vendors, whatever storage solution is chosen will be made easier by choosing an Intel® architecture system. This applies to all of the storage solutions discussed in this paper.



Another example would be CompactFlash*. The CompactFlash Association chose to emulate an IDE drive when the standard was first adopted. In their latest update they have chosen SATA. Both of these standards have long been part of the Intel® architecture ecosystem.

In the case of SATA, Intel has supported the Advanced Host Controller Interface (AHCI), which standardizes the register interface for software to use SATA. Similarly, Intel chairs the Non-Volatile Memory Host Controller Interface (NVMHCI), which is the register interface for accessing ONFI.

The Preboot Execution Environment (PXE) is another standard supported by Intel. It makes booting to a remote storage solution possible/

Intel continues to support standardizing storage interfaces, which an embedded system designer can take advantage of to ensure hardware and software support of their storage solution.

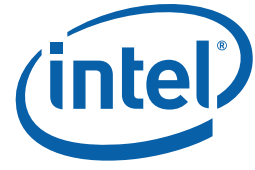
Conclusion

We considered a number of different storage solutions for embedded systems. While flash based solutions were explored more in-depth, we also looked at alternatives.

It is impossible to make a single recommendation on what is the "best" storage solution for embedded applications. Important factors to evaluate are:

- Capacity required
- Write requirements
- Software changes
- Power
- Performance (latency/throughput)
- Cost

The storage landscape is continually evolving, making choices difficult for long lifecycle designs. Solutions that are extremely costly at the beginning of a product's life are likely to be obsolete by the end of its lifecycle.

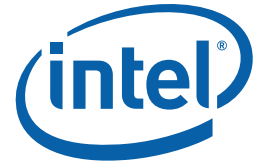


Author

Brian Skerry is a Senior Software Architect with the Embedded Components Group at Intel Corporation.

Acronyms

AHCI	Advanced Host Controller Interface
HDD	Hard Disk Drive
LEB	Local Expansion Bus
MLC	Multi-Level Cell
NVMHCI	Non-Volatile Memory Host Controller Interface
ONFI	Open NAND Flash Interface
PXE	Preboot Execution Environment
RAM	Random Access Memory
SLC	Single Level Cell
SSD	Solid State Drive
XIP	Execute In Place



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This paper is for informational purposes only. THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Intel, the Intel logo, Intel. leap ahead. and Intel. Leap ahead. logo, Intel X25-M SATA State Drive, Intel EP80579 Integrated Processor, Intel Z-U130 Value Solid State Drive are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2008 Intel Corporation. All rights reserved.

§