



OBJECTIVE ANALYSIS

Semiconductor Market Research

PCM BECOMES A REALITY

Memory Design will Never be the Same

Phase-Change Memory or PCM is a new technology being explored by several companies. This technology fits between today's volatile and nonvolatile memory technologies to provide features that appeal to system designers who for years have had to work around the idiosyncrasies of existing memory technologies.

PCM is simple enough to use that designers can begin to forget all the strange work-arounds they now must use to design NOR or NAND flash into their systems, recognizing some significant improvements in time to market that often are joined by related improvements in performance, cost, and code density. In certain cases designers will find it worthwhile to rework existing designs to convert them from flash to PCM. Many designs should also be able to reduce or even eliminate a RAM chip that was once required to compensate for

flash's slow and messy programming protocol.

This white paper goes into some depth about flash work-arounds, none of which are necessary or even useful in a PCM-based system. In the end readers will see that PCM is preferable to established memory technologies by its simplicity which can offer improvements in time-to-market, chip count, software and system complexity, and even power consumption.

To help illustrate the strengths and weaknesses of today's dominant memory technologies against PCM, Table 1 presents the performance of seven key metrics for each of three technologies: DRAM, NAND flash, and phase-change memory (PCM). Relative performance for each technology is given for sequential and random read and write, power consumption,

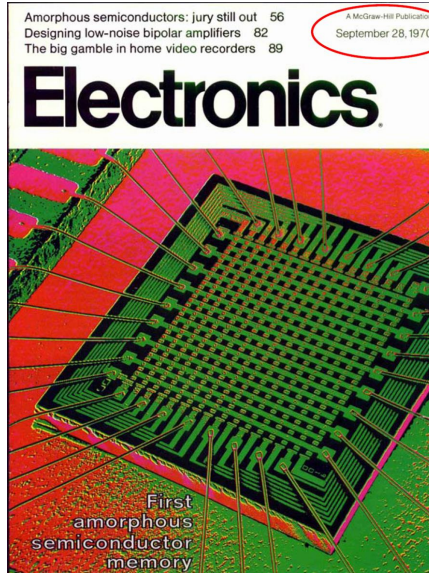


Table 1. PCM vs. DRAM & NAND

	DRAM	NAND	PCM
Sequential Read	Fast	Fast	Fast
Random Read	Fast	Slow	Fast
Sequential Write	Fast	Medium	Medium
Random Write	Fast	Slow	Medium
Power	High	Low	Low
Nonvolatile	No	Yes	Yes
Cost	Medium	Low	High

volatility, and cost per bit. This data will be discussed in depth over the course of this white paper.

Why PCM? Why Now?

In a 1970 Electronics magazine article titled *Nonvolatile and reprogrammable: The read-mostly memory is here* Gordon Moore explained Intel's prototype of a 128-bit memory based upon a phase-change material similar to that used in today's Numonyx devices. This device was featured in the article's cover photo and is shown in Figure 1.

It's been nearly 40 years since this paper was published – several lifetimes in the world of semiconductors. Why has this technology remained dormant for so long and why is it now finally reaching production? There are several reasons for both. The biggest reason that PCM has been unable to make its mark in the market today is that existing memories have proven to be far more economical than any new alternatives. This has been the case for quite a long time, shutting any newcomers out of the market. On any given process, these alternative memories have either suffered from having a larger die size than their entrenched competition, or the wafer processing costs have been significantly higher. Cost is everything in the memory market, so any chip with a higher manufacturing cost doesn't stand a chance of displacing any existing technology. This will change soon, as PCM costs close the gap with DRAM over the next few years.

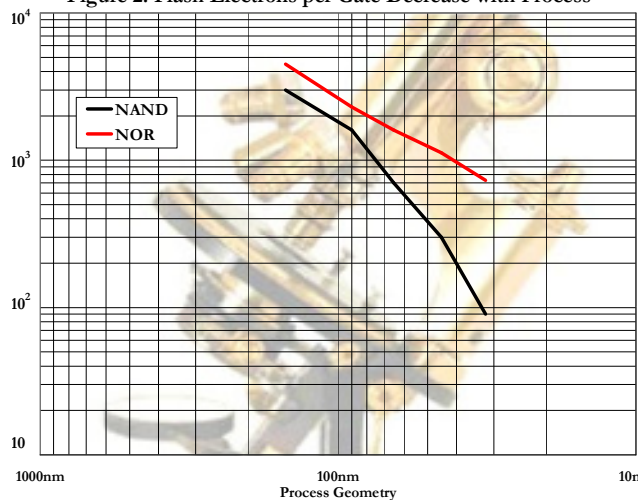
The PCM of 1970 also had a problem with power consumption. The PCM in the 1970 Electronics article required 25V at 200mA (5W) to drive the state change during a write! Today's PCM uses power levels that are a few orders of magnitude lower than that, and are similar to those required by today's NOR and NAND flash chips.

On the positive side, there are a few reasons that PCM has recently become attractive. For one, materials have progressed significantly over the past decade and it is now much more feasible to produce the high-purity thin films that are required by the phase-change material. Also, there have been numerous breakthroughs with the chalcogenide

materials used in PCM because they have been used in high volume to manufacture both CD-R and CD-RW disks. Along with this has been a vast increase in the understanding of the physics of these materials. Process shrinks have played their part: In the past the amount of material

to be heated was relatively large, requiring significant energy to achieve a phase change, an important point we explained above. As processes have shrunk, what once seemed like an ocean of material to heat has now plummeted to something more akin to a bathtub. Finally, a general acknowledgement that flash memory will soon reach its scaling limit has added impetus to develop follow-on technologies that will continue to scale past this limit. Although flash's scaling limit has been postponed for a number of years, all flash makers agree that there will soon come a time when flash can no

Figure 2. Flash Electrons per Gate Decrease with Process



longer be shrunk to the next process node and the industry will have to change technologies.

What's this? A limit to scaling? Yes indeed. As Figure 2 illustrates, the number of electrons stored in a flash bit is on a steady decline. It appears, from the shapes of the curves in the chart, that both NAND and NOR flash could store fewer than 10 electrons per bit well before either of these technologies reaches the 10nm process node in about ten years. Ten electrons is far too few to store multiple bits in MLC in a noisy environment, so the minimum required number of electrons is significantly higher than 10, closer to 100 per bit. Even at this level the low number of electrons makes it difficult to meet the reliability requirements of existing applications.

Today's common expectation is for NAND and NOR flash technologies to hit their scaling limit by 25nm or 20nm, which are only 2-3 process generations or 4-5 years from today.

Phase-Change Memory is here today. Samsung announced a PRAM prototype in 2004 that is a precursor to 2009 production. Shortly afterwards Numonyx (then Intel and STmicroelectronics) announced a prototype PCM device that started shipping in limited production toward the end of 2008. One other company – BAE Systems – has been shipping its C-RAM chips into aerospace market since 2006. This market is interested in PCM since it is immune to bit errors caused by alpha particle radiation.

Those designers who have been able to try out these parts report to us that they are pleased to find that the technology removes a number of obstacles they have had to work around when using older, more conventional memory technologies.

New Memory Structures

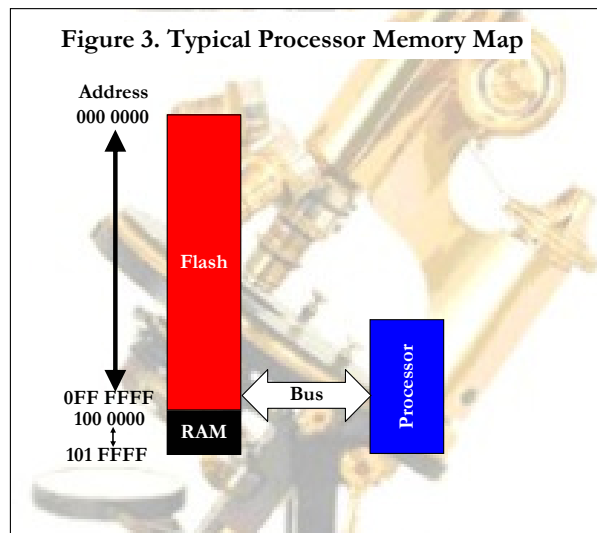
To understand the enthusiasm designers have for PCM we must consider how memory is used in the system. A typical system will use both some type of non-volatile memory and some type of RAM

(Figure 3). Even within microcontrollers there is an SRAM bank and a ROM bank, with the code residing in the ROM and the data residing in the RAM. As the systems increase in complexity external memory phases in, with NOR flash and SRAM for many systems, and NAND

flash (or a combination of NOR and NAND) plus DRAM for more complex systems. Even a PC or a server uses this model, with a NOR holding the BIOS that bootstraps the system, and a disk drive holding the bulk of the software, which is ported to DRAM for execution.

In order to keep the programmer from having to work around this mess, operating systems hide the differences between each type of memory and perform the task of managing the volatile and non-volatile memories in a way that is transparent to the other programs. This adds a considerable amount of complexity to the system.

Even with this help, the programmer is under some constraint to fit the code into the read-only code space and the data into the read-write data space. If either



code or data grows larger than the size of its memory, even by a single byte, the size of that portion of memory must be doubled at a significant price increase. We will soon see that in some cases this can be avoided in a PCM-based system.

Why is flash not used for the read/write or data portion of the memory? After all, it can be written to! The reason is simple, although the mechanism is complex. Flash memory by its very nature is organized to be written to in bytes (or pages in the case of NAND flash) but can only be overwritten if an entire block is erased. Put simply, this renders the chip unsuitable for the small random writes that a processor needs to perform. We will explore this in significant depth later in this white paper.

These challenges make it impossible to build a system using only one technology. A complement of nonvolatile memory and RAM must be used to assure reasonable operation. That is, this was the case until the introduction of PCM.

Phase-Change Memory has changed the rules of the game. No longer must code and data be split between two fixed-size banks of NVM and RAM. Code and data may be contained within a single component. For small system designers this reduces chip count and power consumption. Designers of both large and small systems will appreciate the fact that there is no longer a fixed delineation between read/write and read-only memory. This gives significantly more flexibility to the programmer, allowing different programs with different balances of code and data to be used within a single hardware design. If a pinch more data space is required, it can be taken as a pinch, rather than the doubling that is necessary with discrete components. Likewise, if code creeps slightly over the size of the preassigned program space, it

can move into some of the data area's unused space. Life is good!

One important factor is that this flexibility yields cost benefits in most systems, since memory need not be added in binary chunks, a limitation that has so far required significant incremental cost should either code or data exceed the size of a memory chip. Surprisingly, the cost of a PCM-based system is often lower than that of its NOR/DRAM or NAND/DRAM counterpart for this reason alone, even if the PCM chip is more expensive than a NOR or DRAM chip of the same density.

Intricacies of Flash

Flash memory is difficult to manage. One designer we interviewed referred to the process of managing flash as a "Very involved dance." Those who have designed with either NAND or NOR flash will attest to this. Flash management involves a number of considerations that make this task extraordinarily complex. Let's explore this. To keep it from becoming too complex at first, we will start with NOR, the more straightforward technology, then move to NAND.

NOR flash is very simple to read – you input an address and the data from that address comes out a set time later. This behavior is similar to the way an SRAM operates, and PCM behaves the same way. From the processor's viewpoint it's the simplest to manage of any memory interface.

A write to a NOR is more involved, and is called "Programming" a term that refers back to the technology's roots in EPROM. When a byte is written in flash, the programming process can only change bits from 1s to 0s, but not vice-versa. This is because flash is a cost-reduced version of its EEPROM predecessor. In EEPROM each memory cell (bit) uses two transistors: one to

store the data (0 or 1) and the other to isolate the storage transistor from the rest of the memory array and allow a small amount of data to be altered at one time. In flash, the memory cell is composed of a single transistor for storage of data and the isolation is provided at “block” level where a block is composed of tens of thousand or up to one million bits. This makes the chip significantly smaller and cheaper. A consequence of this is that the entire block must be erased at one time, so individual words cannot be reprogrammed until the block has been erased.

This can make life difficult for programmers. If only one word within a block needs reprogramming, then the entire contents of the block will need to be moved to temporary storage (RAM) so that the block can be erased. While the data is in RAM the new word will overwrite the word that needed changing, then the entire block will be re-written with the original data plus the modified word. This complex two-step is illustrated in Figure 4.

Since it takes a few seconds to erase a block, and another half second or so to program all the words within that block, you don’t want to move all the data into and out of the block every time a single word needs updating. This is clearly an issue that needs to be managed.

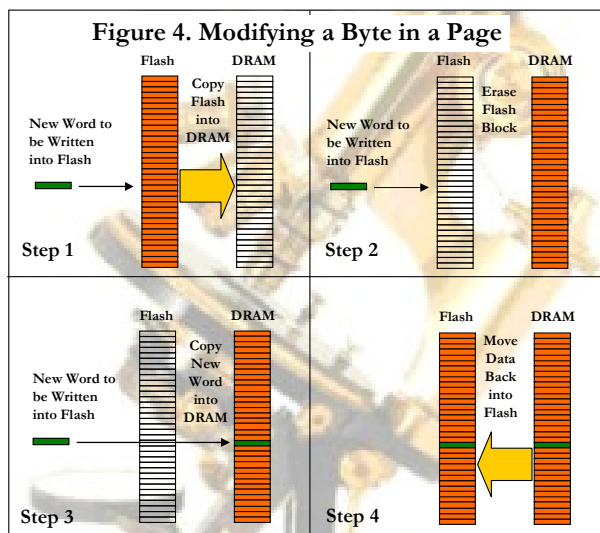
Furthermore, if a single block undergoes a significant number of erase/write cycles, it will wear out. We will explore this problem shortly.

A standard work-around for this problem is to make sure that a program separates its working data into two spaces: One that undergoes infrequent changes if it is changed at all, and one that undergoes frequent changes. Furthermore, to prevent the program from causing an entire block to be erased and re-written every time that a single byte needs to be changed, most coders use a block abstraction algorithm that re-maps the frequently-changed data to arbitrary locations within a pre-erased block that is set aside for frequent writes. Data that the code believes is at address 100 002A may actually be mapped to 101 F00D the first time it is reprogrammed, 101 F00E the second time, 101 F00F the third time, and so on. The older locations which have become invalid are no longer referred to by the address translation rendering them invisible to the application program.

The task of keeping track of this address translation can be complex and time-consuming for the programmer, so

utility packages have become generally available to manage the re-mapping of frequently-changed data. This tends to be a part of a “Flash File System” software package.

What about that comment a while back about wear-out? Each block in a flash memory is guaranteed to withstand a limited number of erase-write cycles. After that number of cycles is surpassed, individual bits are likely to become stuck at a 0 level and cannot be erased. The number of cycles guaranteed by the manufacturer is called a part’s “endurance” and today ranges from



10,000 erase/write cycles for 2-bit MLC flash to 100,000 cycles for SLC flash. In the future the number of bits stored on a cell will increase to 3, then 4 bits. With each increase comes a reduction in endurance, so 3-bit and 4-bit MLC are expected to have endurance well below 10,000 cycles.

To counter this problem either the same Flash File System software or a separate controller chip (in the case of NAND) performs a function called “Wear Leveling”. We present Figures 5 and 6 to help illustrate how this concept works. To simplify the picture we have only shown 16 erase blocks, but

a typical part is likely to have hundreds of such erase blocks. Figure 5 indicates the wear that might be incurred if a single block were to be hit with 90,000 erase/write cycles of its total 100,000-cycle life. The chart shows that this block will be within 10% of its usable life, where all the other blocks show no signs of wear at all.

If that block wears out, then the flash chip might become unusable, causing the system to fail.

In order to keep any one block from wearing out, “wear leveling” algorithms are used to spread the wear among all blocks of the device. This is illustrated in Figure 6. With the

fictitious 16-block device we have used, all blocks are allotted 1/16th of the wear that would have hit the single block of Figure 5. Rather than having one block with only 10% of its life left, threatening the system with failure, all blocks will still have over 94% of their useful life left.

Once again we have an algorithm that must re-map all blocks from a physical address seen by the system to virtual blocks within the flash chip, requiring address mapping to work both the byte reallocation mechanism and the wear-leveling part. Clearly, flash file systems are very complex programs!

But that’s not all that must be considered when working with NOR flash! One final difficulty is that a write to any single word address, without considering a block erase, consumes around 15 microseconds, or about 200 times as long as a read. Were the system to have to stop every time a byte was written to the flash chip overall performance would slow to a crawl.

Up through the middle 1990s cell phone designers who were saddled with this problem would add yet another memory chip – an EEPROM – to absorb the more frequent writes. This would add a third memory zone to the memory map of Figure 3, increasing the phone’s size, chip count,

Figure 5. Wear on a Single Block

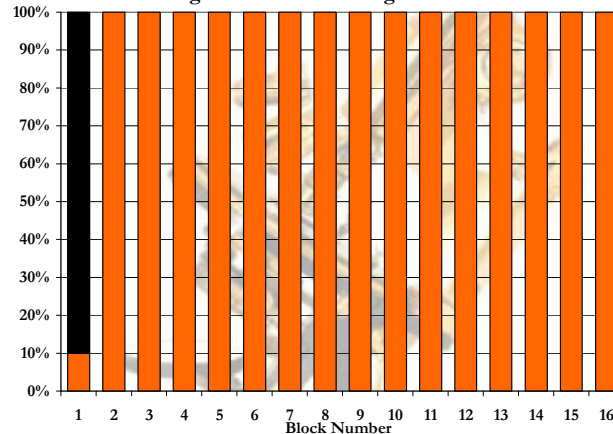
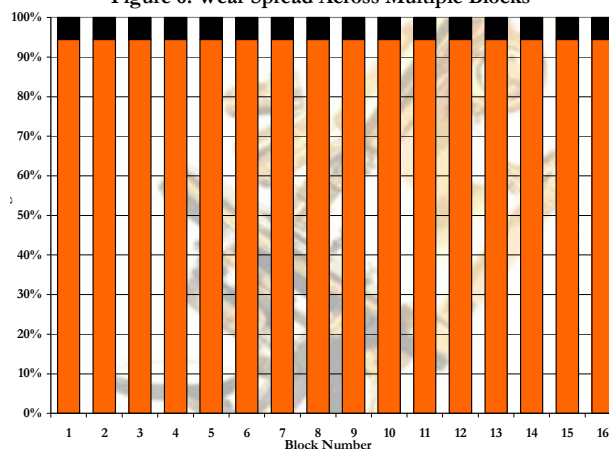


Figure 6. Wear Spread Across Multiple Blocks



cost, and power consumption while making the programmer's job even more complex. Leading manufacturers devised a new type of NOR chip with a read-while-write capability that supported reads from one bank while a word was being written to another bank. This behaved like multiple discrete NOR chips in a single package. It was an elegantly simple solution to most of the problems, but it didn't make the programmer's job any easier. The programmer is still responsible for seeing that the writes are mapped to a different block than the reads that will occur near the time that the write is invoked. If that sounds complicated it's because it is. From a speed standpoint this was considered to be a worthwhile trade-off, giving up ease of programming to gain reduced chip count and cost.

Now this has been a great long dissertation about the complexities of handling NOR, precautions that are generally not required with PCM, but before we move to the glories of PCM let us examine the additional problems posed by NAND.

NAND is Worse

NAND adds a number of further complexities to the long list of those presented by NOR above.

NAND flash is designed to minimize chip area through significant concessions. The technology uses bits that are largely reliable, but are not guaranteed to consistently contain the data that was written into them. This means that error correction must be used for anything

programmed into or read from the NAND chip.

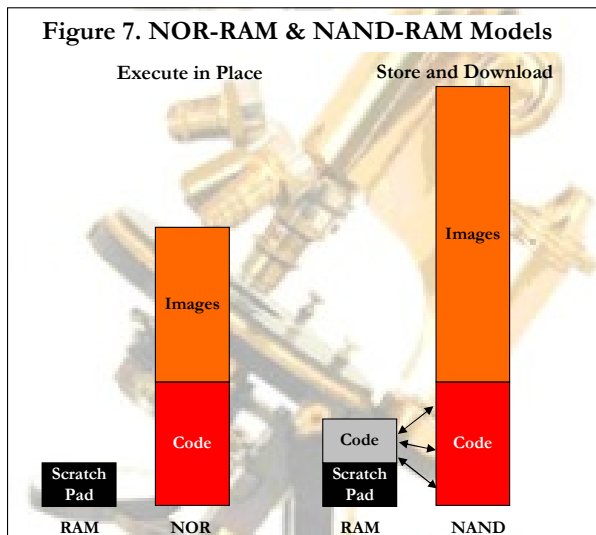
Error correction is performed on a whole page, so the NAND is set up to read or write an entire page at one time. Each page has its own syndrome bits which are used to store error correction information. A page read has a relatively long set-up time of about 25 microseconds, but then data is read out sequentially at a very rapid rate of one byte or word every 50 nanoseconds. That would be great except that it has absolutely no bearing to the way that processors ask for data and instructions. Processors demand data in a seemingly random sequence, so even if a NAND

chip did not require error correction it could not be used effectively for code storage.

To make up for NAND's sequential nature and its need for error correction designers use an architecture like the one shown in Figure 7, where a small RAM

(usually a DRAM) is used to store portions of the code in the NAND. The DRAM is small, because if it were large, then the DRAM/NAND array would be costlier and would consume more power than the NOR alternative.

In general designers use the smallest DRAM they can get away with in order to reduce power as much as possible, and often use mobile SDRAM, which has features to allow half or even $\frac{3}{4}$ of the array to be powered down. This forces these designs to use "Demand Paging" software, a more exotic kind of operating system that allows portions of programs to be moved into and out of



the DRAM as required, fooling the application into thinking that everything it needs permanently resides in a very large DRAM.

A further problem with NAND is its sensitivity to multiple read cycles. If a bit is read several times then neighboring bits tend to lose their contents due to a mechanism called “Read Disturb”. We won’t go into that mechanism in this white paper, but will note that it causes the system to require something like a DRAM refresh cycle from time to time to push the bits back toward their original state of a 1 or a 0. This would not be such a big problem if except that the refresh process requires the use of an Erase/Write cycle in the process. The Erase/Write counts against the chip’s endurance specification the same way that a standard Erase/Write would. An Erase/Write is also extremely time consuming, taking the chip offline for up to a full second. Taken to the extreme, if a system only read from a NAND chip and never wrote to it for a significant number of times it would still suffer endurance failures over an extended period, since Erase/Writes would be performed to clean up the bits.

Many NAND-based systems use a chip devoted to the management of the NAND chip’s wear, error correction, and other housekeeping. In a system using PCM this chip (and its throughput delays) would be completely unnecessary, providing a path to reduced chip count, cost, and power consumption.

DRAM Issues

We should mention something about DRAM, since this technology also brings some challenges. We already mentioned above that DRAM consumes a lot of power – typically a DRAM will consume twice the power of the same size NOR during standard operation. Part of this stems from the fact that a

DRAM needs to be continually refreshed. This refreshing adds complexity to the memory control logic and consumes power, so DRAM is more problematic to design with, especially in battery-operated systems.

DRAM is considered a useful alternative to SRAM, though, since it is available in significantly higher densities at a considerably lower price.

DRAM’s high power consumption is a problem that will only worsen over time, since power consumption increases as a DRAM’s process technology shrinks. This sort of power issue is not the case with PCM, whose power consumption actually decreases as its process technology shrinks, so the power consumption gap between DRAM and PCM will grow over time.

Keep in mind, too, that since PCM has relatively fast write cycles and is less sensitive to wear, DRAM or SRAM can be pared down in a PCM-based system to reduce cost and power consumption. In some cases the high-speed RAM requirements can be judiciously filled by the internal memory of the processor – either a cache or a scratchpad memory. In such systems the use of an external RAM can be avoided altogether, saving chip count, power, and cost.

Interestingly, with the exception of its slower writes PCM specifications look very similar to those of a DRAM. Sequential reads are equally fast, and random reads come close to the same speed. If the PCM is regarded as a nonvolatile DRAM with a slower write speed, then new approaches to a system design will present themselves to the designer.

PCM to the Rescue

Phase Change Memories have considerably reduced the long list of problems explained over the past several paragraphs. PCM is byte-alterable so

there are no erase blocks. This vastly simplifies writes. Writes do not need to be preceded by an erase cycle – in PCM ones can be changed to zeros and zeros can be changed to ones. A PCM's write is more like a write to a RAM than to either NAND or NOR flash.

In some cases a system's entire memory can be economically made using only PCM, often relieving the programmer of concerns about using one component for code and a different component for data. The programmer is now empowered to determine exactly how much of the memory space will be assigned to program storage and how much will be used for data.

PCM is in an early phase of its evolution so it tends to be conservatively specified. Even so PCM's million-write endurance is already a full order of magnitude better than the erase endurance of even SLC flash (10^5 Erase/Writes). PCM holds the promise of improving its endurance specification over time, a fact that should render complex wear-leveling algorithms unnecessary. Since these chips have not yet been put into high-volume production we fully expect PCM makers to confidently tighten endurance specifications far beyond today's numbers.

Furthermore, all PCM bit failures occur on write cycles, so if a verification protocol is employed, then a bit failure will immediately be detected, and the word to be written can be re-mapped to a different address.

With these significantly lowered concerns about block management or wear leveling there is a far lower need to reassign blocks and map addresses – in the vast majority of cases the data

resides in the same PCM address that the processor believes it is in. This simplifies software significantly. A Flash File System is simply not required.

PCM has fast write cycles with no need for an erase. This removes the need for a concurrent read/write capability and programmers will rarely, if ever, need to write code to prevent reads from

Table 2. Comparing Attributes of NAND, NOR, and PCM

	NOR	NAND	RAM	PCM
Byte Alterable	No	No	Yes	Yes
Write Page	32B	1KB	32-64B	32-64B
Erase Page	256KB	512KB	N/A	N/A
Read	Random	Sequential	Random	Random
Endurance	10^{4-5} erases	10^{3-5} erases	Unlimited	10^{6-8} writes
Data Integrity	Absolute	Corrected	Absolute	Absolute

occurring near a recent write.

When compared to NAND, PCM has random NOR-like or SRAM-like addressing which perfectly matches what the processor wants to see. Furthermore, PCM does not require error correction since all bits are guaranteed to contain the same data that was written into them. Future parts will use a standard DRAM interface, allowing a DRAM and a PCM to share the same bus, and for other systems to remove their DRAM altogether and use a PCM without changing the interface.

Finally, since code can be executed directly out of the PCM, there is no need to use a demand paging operating system. With neither a Flash File System nor a demand paging operating system, the software support for PCM becomes much simpler than that required by NAND flash.

Not only that, but without any of this software support – wear leveling, bad block management, error correction, or demand paging – code becomes smaller,

simpler, and more manageable, and many system delays evaporate. Chips dedicated to managing flash can be cut out of the system, and in some cases the improved speed of a PCM system will allow the use of a cheaper system processor thanks to the elimination of all those delays once required to manage the NAND or NOR flash.

Table 2 compares NAND, NOR, and RAM to PCM. All those lengthy explanations above about the processes that are required to manage flash in general, and NAND flash in particular, are quite simply unnecessary in a PCM environment. Some call PCM: “A firmware/software engineer’s perfect nonvolatile memory.”

Fits All Applications

In the earlier sections of this report we briefly touched on the point that each memory application requires its own unique memory topology: Systems with modest code requirements and little data can use NOR and SRAM. For large data sets a combination of NOR plus NAND and SRAM or PSRAM can be used. To save costs, this second solution can cut out the NOR, but the RAM must grow, so DRAM will be used, unless the system has power constraints, in which case it will gravitate toward using mobile SDRAM with demand paging software to manage power consumption.

With PCM this challenge dissolves. A single PCM chip or a PCM-only array

will store both code and data, removing any need for a solution based on an external RAM chip plus a nonvolatile chip.

As an added bonus, the programmer need now worry only about the size of the code plus data, rather than about the code space and the data space as two separate areas. If the data space increases by a couple of bytes, perhaps there will be room that can be “borrowed” from the code space. This luxury is simply unavailable in a non-PCM topology.

How PCM Works

So what is this new PCM and how does it work? Phase-Change Memory uses the changing state of a special material to determine whether a bit is a “1” or a “0”. This is very different from other memory types that store a charge. While DRAM and all floating gate memories (NAND, NOR, EEPROM, and even EPROM) use a stored charge to indicate whether a bit is high or low, the phase-change memory’s two states are

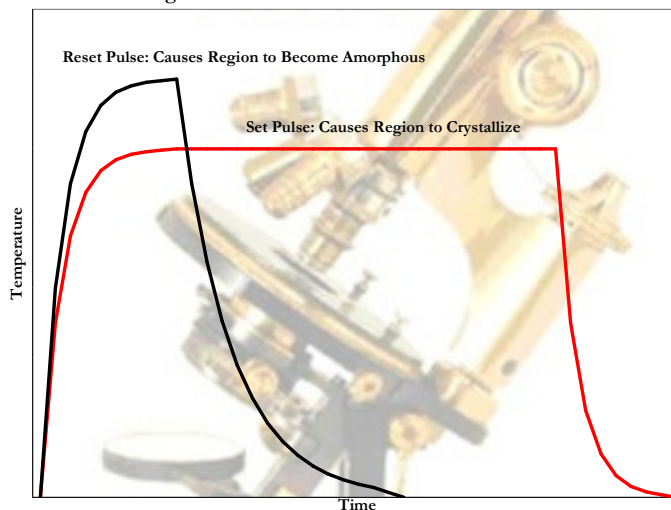
“crystallized” and “amorphous”.

Like a liquid crystal display blocks light or allows it to pass through, depending on a crystal’s orientation, the chalcogenide glass that is used for bit storage in a PCM either

allows current to pass (when crystallized) or impedes its flow (when amorphous).

How does the chip switch the material from a crystalline state to an amorphous

Figure 8. Set and Reset Mechanisms of PCM



one? Each bit location has a tiny heater that melts the glass, then cools it in either a way to allow crystals to grow or to keep crystals from growing.

Figure 8 illustrates the temperature profiles to set and reset such a bit. For the set pulse, the temperature is raised to roughly 600°C to melt the glass held at that level for a time, then ramped down once crystallization has begun. The reset pulse increases to a higher level of about 700°C then immediately ramps down, before the melted material can form any crystals. This results in an amorphous, or nonconducting, organization of the material at that location.

Since very small heaters are used, they can heat the material very rapidly for a tiny location – on the order of nanoseconds. This allows fast writes to be performed and prevents adjacent bits from being disturbed. Furthermore, these heaters shrink

along with process geometry shrinks, which will make parts produced with a smaller processes easier to program than their large-geometry predecessors. All in all, this is a technology that should shrink well beyond the limits of NAND and NOR flash.

PCM write speeds are comparable with NAND flash now and will increase by an order of magnitude on future products. Add to this the fact that PCM writes require no erasing nor the generation of any syndrome bits for error correction and the net result is that system-level write speeds will be one or two orders of magnitude faster than those of NAND

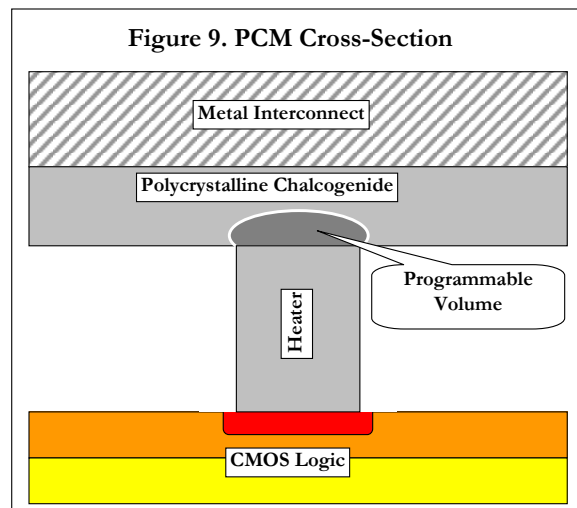
flash. This in itself should save system designers from having to employ all the work-arounds we listed earlier that have been devised to accommodate NAND's and NOR's very slow erase/programming protocol.

PCM read speeds are comparable with NOR flash now and will approach DRAM-like speeds with DDR interfaces and lower latency product designs.

From a system architecture viewpoint, one beauty of PCM is that there is no block erase – each bit can be set or reset at any time independently of all the other bits. This gets past the limitations that block erase imposes on both NAND and NOR flash. Recall that block erase was

a necessary evil for flash, since it is the reason that these chips can be manufactured more economically than their more expensive predecessor technology, the EEPROM.

Figure 9 roughly illustrates a cross section of a PCM bit. Standard CMOS



logic is used to switch and sense, with a few additional layers patterned on top to create the heater, the phase-change element, and the top metal sense line.

The phase-change layer consists of a material called a chalcogenide glass which has properties that allow it to be converted back and forth between an amorphous and a crystalline state. The particular glass used by today's PCMs is dubbed GST, a short-hand form of $\text{Ge}_2\text{Sb}_2\text{Te}_5$ a fast phase-change compound made up of germanium, antimony, and tellurium. The same material has been in high volume production for well over a decade as the data layer in CD-R and CD-RW media.

This glass is layered atop the heating elements.

Diodes are used to control individual bits in today's PCM, as opposed to the MOSFET approach used by DRAM, NAND, and NOR. These diodes are made from bipolar transistors, and can be packed more tightly than can their MOS counterparts. This approach is commonly referred to as a "cross point array", with some using the term "X-point." Diode access dramatically shrinks the area of the cell, which is already only $5f^2$ (five times the square of the smallest feature that can be printed on the chip) as opposed to the $6f^2$ cell size used by today's leading-edge DRAM and the $8f^2$ cell used by many DRAM makers. NAND cells are typically around $5f^2$, which shrinks to a virtual cell size of $2.5f^2$ through the use of 2-bit MLC. We expect to see further shrinks of PCM cells over time from the use of more aggressive design.

Although multilevel cell PCM technology has not yet been brought out of the research lab, many feel that the same

techniques that are so successful in NAND and NOR flash can be employed with PCM. Should this be successfully brought to market, then PCM prices, which are already roughly equivalent to DRAM prices, will come much closer to those of NAND, today's lowest-cost memory technology.

In the future researchers expect to be able to stack phase change layers to

produce 3D memory arrays, cutting costs significantly as more layers are added.

Price and Cost

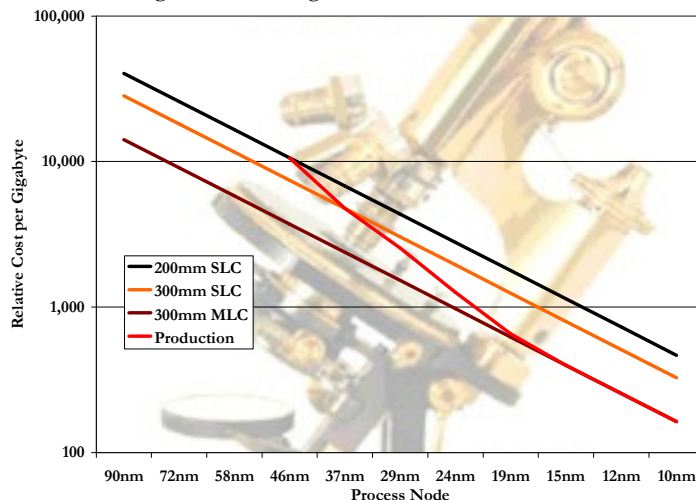
Memory chip prices are highly dependent on costs, and Objective Analysis projects that PCM makers will take a number of steps to bring their costs down to those of competing technologies, steps similar to those NAND makers used to drive their costs faster than could be achieved through simple lithography progressions. One is to move from a lagging process node to a leading one. Another is to convert from 200mm to 300mm wafers to realize an automatic 30% savings. Finally, the move from SLC to MLC will drive further cost reductions.

Figure 10 illustrates how these changes are likely to drive significant cost structure reductions, most of which are likely to be passed to the OEM. Each of the parallel lines represents the cost of a PCM for a particular wafer size and number of bits/cell. Costs for each of these lines steadily decrease as processes

shrink according to Moore's Law at a rate of about 32% per year. The slope of these parallel lines corresponds to this rate of decline. By progressing from one line to the next lower line the actual cost decrease can be made to

move faster than any of the parallel lines, as the red line illustrates. Here, the "Production" device not only follows process generations, but it also moves from 200mm to 300mm wafers, and from SLC to 2-bit MLC. In this case

Figure 10. Reducing Costs Faster than Moore's Law



costs decline nearly 40% per year, imitating the cost structure declines of NAND flash.

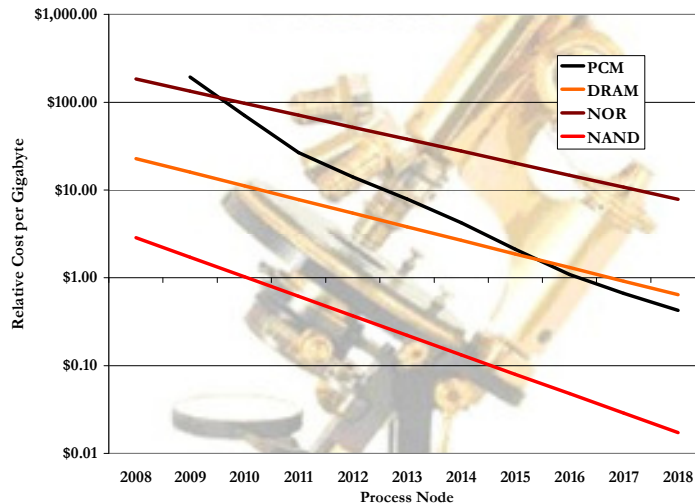
Figure 11 illustrates how this accelerated price decrease compares against our expectations for the manufacturing cost decreases of other memory technologies. The chart shows Objective Analysis' projected manufacturing cost per gigabyte for four key memory technologies: DRAM, NAND, NOR, and PCM. We expect PCM's price per gigabyte to fall below that of NOR within the next year.

Today PCM sells for roughly 25 times as much per gigabyte as does DRAM. Since PCM uses a $5f^2$ cell, which is tighter than the $6f^2$ used by leading-edge DRAMs, once wafer and process equality is reached,

PCM manufacturing costs should be able to fall below that of DRAM. As the technology gains parity with DRAM in process geometry and wafer diameter, and as unit volume grows large enough

to impact economies of scale, PCM will cross below DRAM's average price per gigabyte, a situation the market should expect in 2015-16. Further cost reductions will become possible as PCM migrates to MLC, when the technology's cost (and price) can be expected to fall to less than half that of DRAM, making it the second least expensive technology after NAND.

Figure 11. Likely PCM Price Decreases Over Time



Jim Handy, August 2009