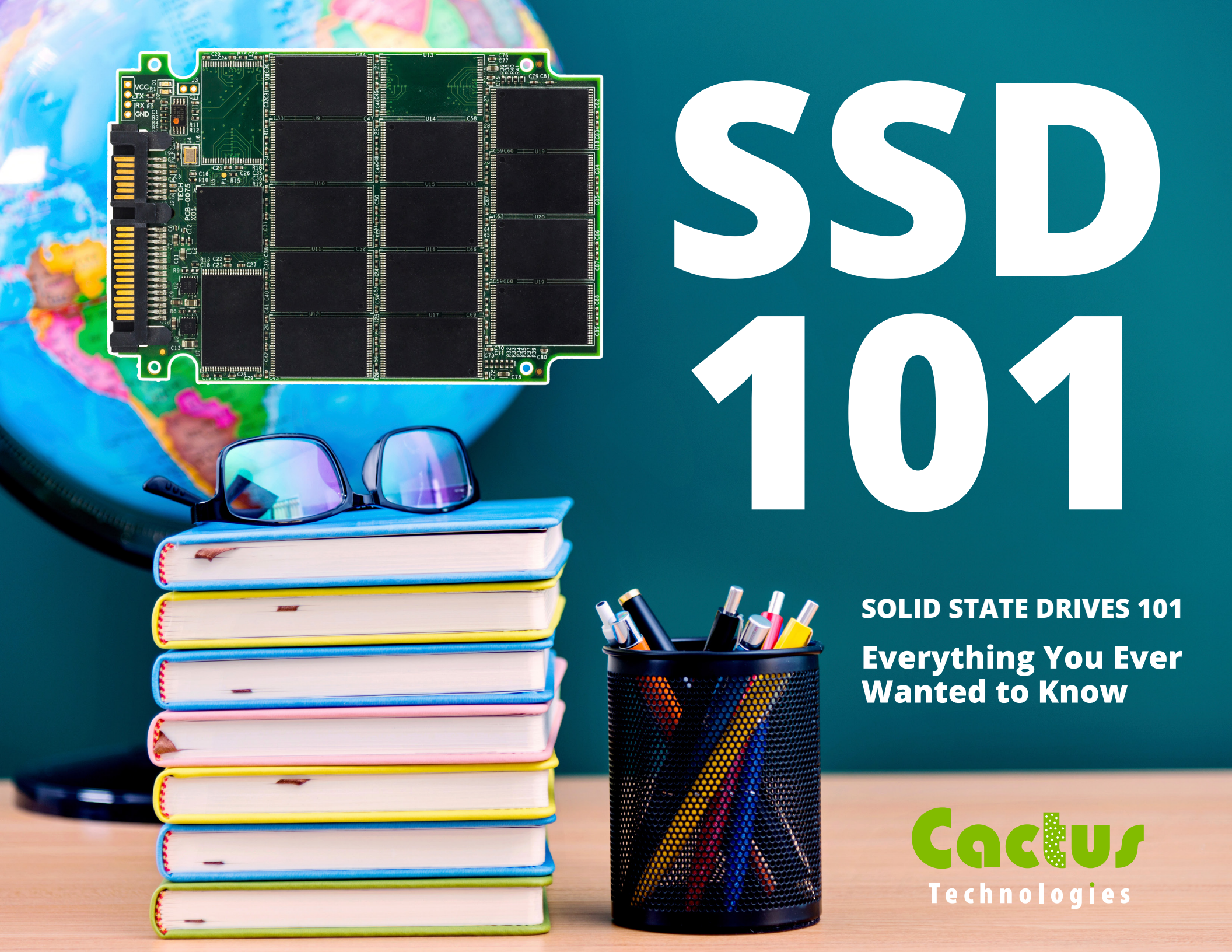


SSD 101

SOLID STATE DRIVES 101

Everything You Ever Wanted to Know

Cactus Technologies

About the Author

Steve Larrivee
VP Sales & Marketing
Cactus Technologies

Steve Larrivee has over 30 years' experience in the data storage market, including 5 years at Seagate Technology and 10 years at SanDisk. He joined Cactus Technologies Limited as an equity partner and Co-Founded Cactus USA in 2007 with partner Tom Aguillon.

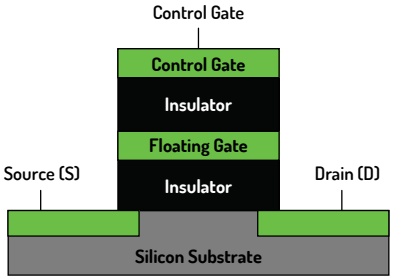
Index

01 Introduction - The Basic NAND Flash Cell

This section takes a look at the basics of a NAND flash cell, the building block of almost every solid state drive. It is the first of several sections describing the basics of Solid State Drives (SSD).

In order to store a single bit of data on a solid state drive, you need the smallest building block - a single NAND flash cell. The simplest NAND cell can be set to either a 0 or 1 state. It will continue to store that state even after power has been removed.

What does a NAND Cell look like?



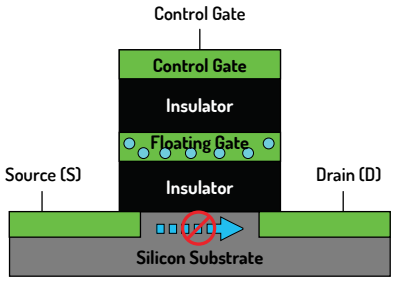
A simple NAND Flash Cell diagram is shown above. The NAND flash cell is made from a floating gate transistor. Electrical charge is stored on the floating gate which is isolated above and below by oxide insulating layers.

In its simplest form when the floating gate is charged, it is programmed and recognized as a binary 0. When the floating gate has no charge it is erased and recognized as a binary value of 1.

FLOATING GATE STATE	REFERRED TO AS	BINARY ASSIGNED VALUE
Charged	Programmed	Zero - 0
No Charge	Erased	One - 1

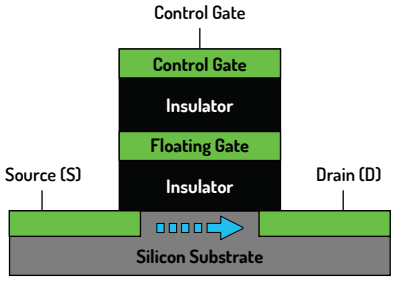
The floating gate remains in its charged or uncharged state until it is changed by surrounding circuitry. Removing power from the NAND device does not affect the state of the floating gate which is why it is such a valuable device for data storage.

How to Read a NAND Cell



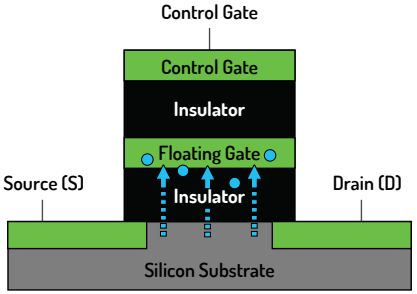
To read a cell, voltage is applied to the control gate and current flow from the source to drain is attempted.

If there is no current flow, it signifies the floating gate is charged (binary 0) - as in the diagram above. If there is current flow, the floating gate is not charged (binary 1) - as in the diagram below.



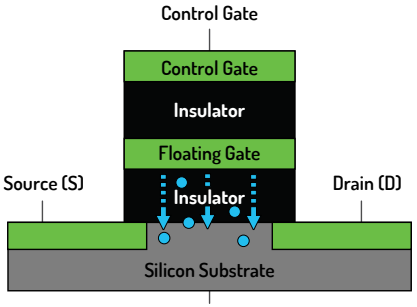
How to Write a NAND Cell

To write a cell, a high voltage is applied to the control gate and electrons move from the silicon substrate to the floating gate. This process is called tunneling since the electrons "tunnel" through the oxide insulator to reach the floating gate. See diagram below.



How to Erase a NAND Cell

To erase a NAND cell, a high voltage is applied to the silicon substrate and electrons move from the floating gate to the silicon substrate. This uses the same tunneling process as the writing process. See diagram below.



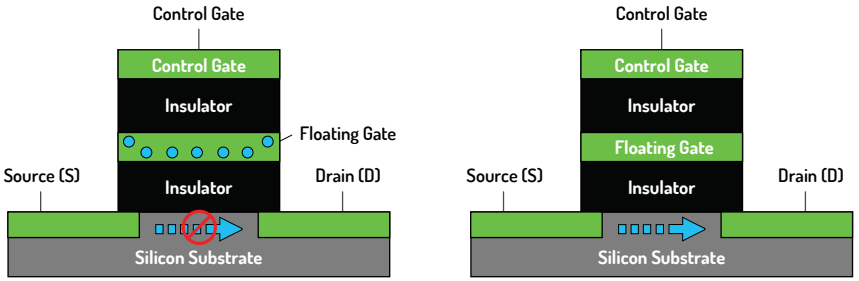
NAND Cell Life

The tunneling process described in the Write and Erase functions cause stress on the oxide insulator layer. Over time this stress breaks down the oxide layer and the floating gate becomes unable to maintain a charge. At some point the cell is no longer usable and must be retired. This is what is responsible for the finite number of writes/erases per cell of NAND flash.

Introduction - SLC, MLC and TLC NAND Flash

This section builds on the Basic NAND Flash Cell by showing the advances in technology from the original SLC to MLC, and TLC NAND Cells.

For a simple review of the Basic NAND Cell, charges are either stored or not stored on a floating gate which is sandwiched between two layers of oxide which act as an insulator.



On the original and simplest type of NAND flash, if no current flows between the Source and the Drain, it indicates the floating gate has a charge (blue dots represent electrons) and therefore is programmed, representing a binary 0. See diagram above left.

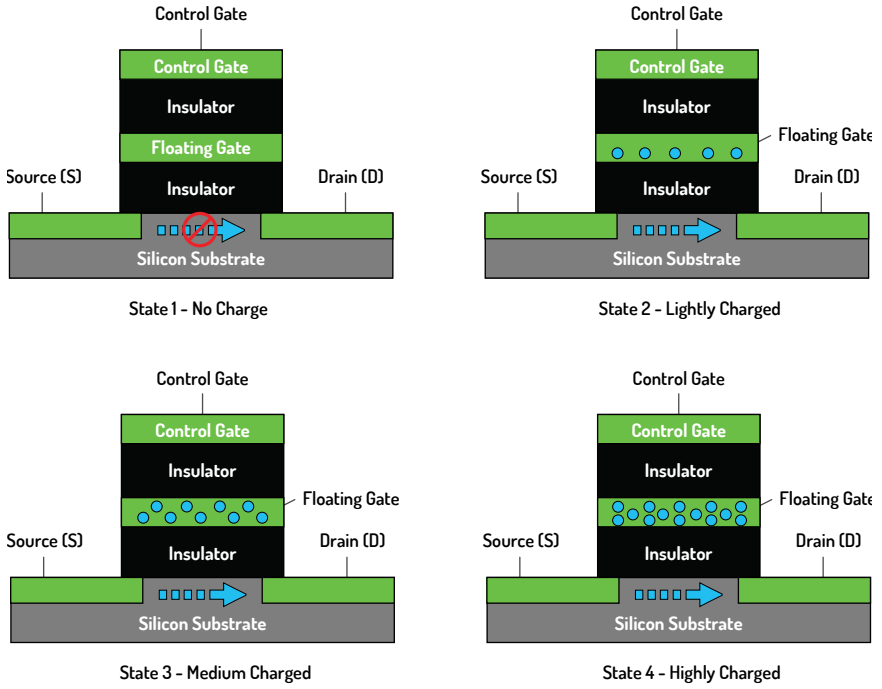
If current flow is detected, it indicates the floating gate does not have a charge and is erased, representing a binary 1. See diagram above right.

How MLC and TLC Store More than One Bit Per Cell

The previous example shows a SLC (Single Level Cell) NAND Cell. When any current is detected between the source and drain it can be concluded the cell is programmed. Since only two states, programmed or erased, are needed to represent one bit, that's all that is needed.

With MLC (Multi Level Cell) NAND, there is a need to store two bits of data, which requires 4 distinct states. In order to accomplish this, the MLC NAND cell must be able to apply charge to the floating gate at four different levels and later be able to detect which of the four levels is set.

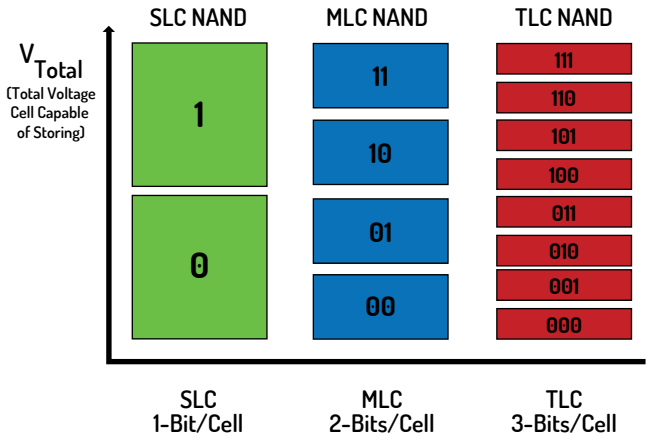
The diagram below represents the additional electrons in blue on the floating gate which must be set to precise levels so they can later be read accurately. This makes MLC more challenging and slower to write than their SLC NAND counterpart.



TLC (Tri Level Cell and also known as Triple Level Cell) NAND has an even more complicated mission. It must be able to store and recognize 3 bits per cell, requiring 8 distinct states.

Voltage Level in SLC, MLC and TLC NAND Cells

The maximum voltage in each cell is about the same. So SLC cells have plenty of guard band between their states. Because of this, SLC NAND is able to withstand temperature extremes and other adverse effects much better than MLC or TLC NAND.



Voltage Allocated to each State based on NAND Flash Technology

The Image above shows the levels of voltage thresholds required to store the multiple states in each of the memory technologies. Not counting guard band area, each SLC state is allocated 50% of the voltage range; MLC 25% and TLC 12.5%.

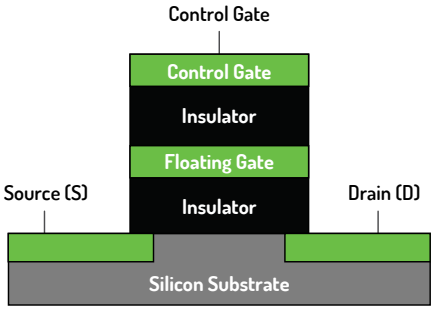
As you can see, MLC and TLC have much tighter tolerances and will be more susceptible to external environmental and circuit effects than SLC NAND. Their principal advantage is cost.

03 NAND Architecture - Strings and Arrays

The previous two sections focused on the individual NAND Cell, whether used to store one, two or three bits. This article focuses on the bigger picture of how numerous NAND cells are combined into strings and arrays.

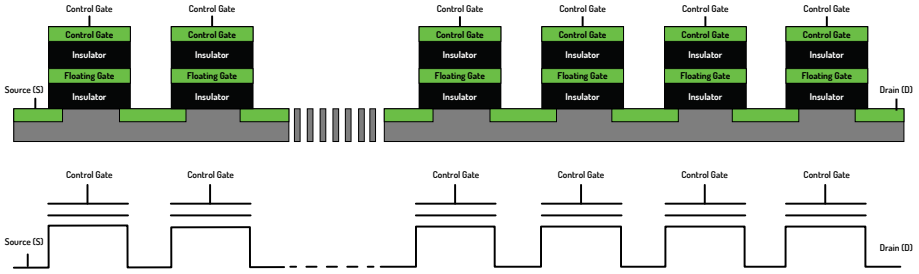
For a quick review, a single NAND flash cell stores an electrical charge on a floating gate which is isolated by oxide insulating layers above and below. In its simplest form when there is a charge on the floating gate it is programmed and recognized as a binary 0. When the floating gate has no charge it is erased and recognized as a binary value of 1.

Diagram of a Single NAND Flash Cell



Combining Individual NAND Flash Cells into a String

All by itself, a single flash cell would not be of much value. But combining many of them is what allows the storage of significant amounts of data. The first step in combining individual NAND cells is the NAND String.

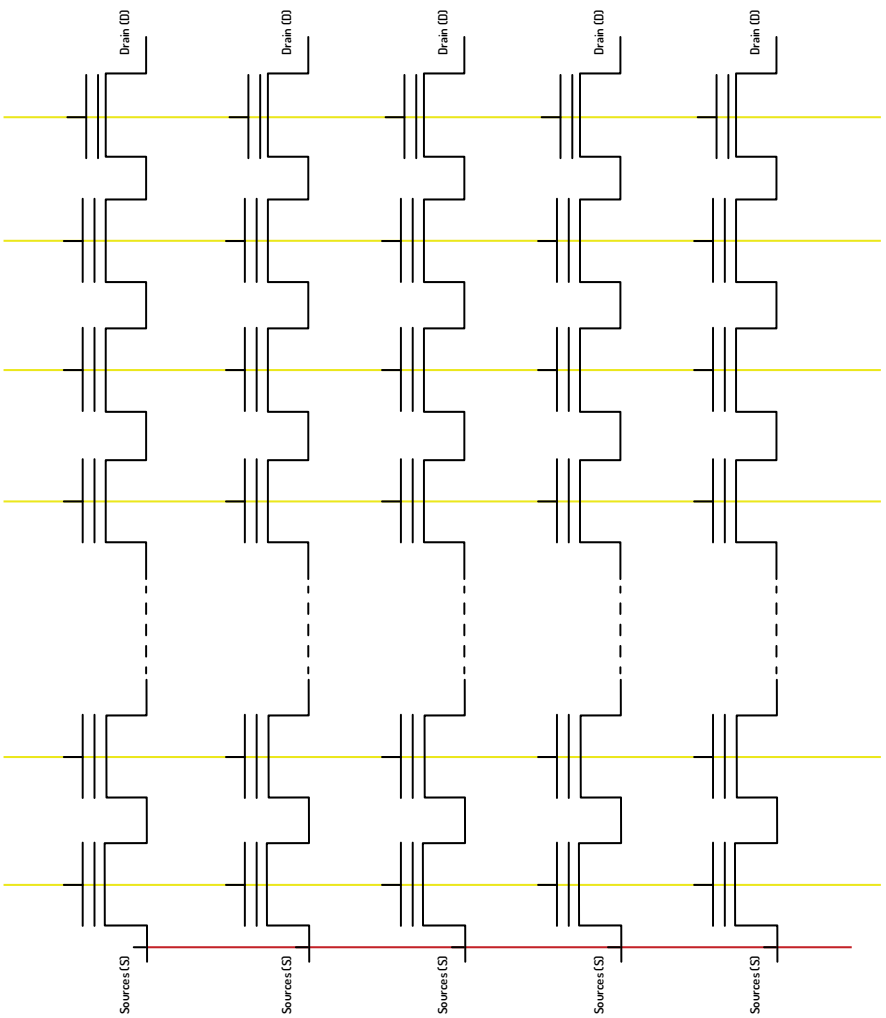


NAND String (Shown in Diagram and Schematic Versions)

The image above shows the NAND String depicted in both a diagram form and in schematic form. Schematic form is typically used to show much larger arrays.

NAND cells are connected end to end to form a string of cells. Typically 32 or 64 cells are connected together in series with each other, with each cell representing a bit of data (0 or 1).

Combining NAND Strings into Arrays



NAND Flash Array

While a NAND String can store 32 bits of data, this still only translates into 4 bytes of data or enough for 4 characters. So, strings are combined into larger arrays to achieve more useful amounts of storage.

The image to the left shows the NAND String schematic repeated several times in an array. Notice the additional connections made to the NAND strings which serve to tie the array together. The red line connects the Sources (S) of the individual strings.

The yellow lines connect the Control Gates of the NAND strings. In the array, the control gates are connected horizontally, but not vertically. In addition, the Drain (D) lines are not showing connections since they will be used separately in the array.

04 NAND Architecture - Pages and Blocks

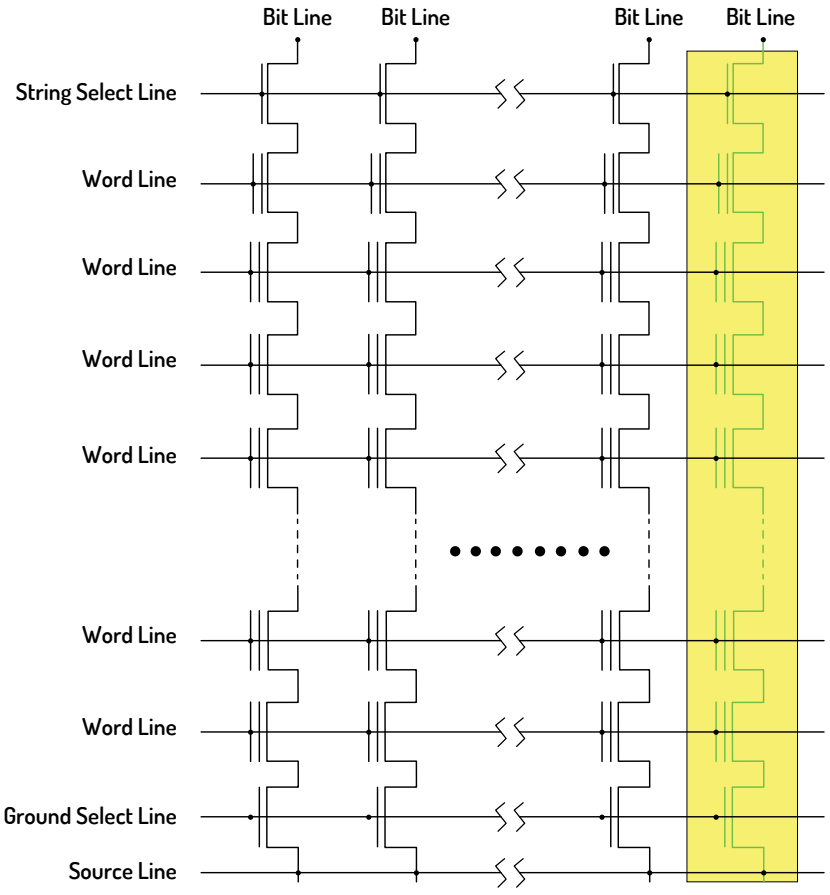
The previous sections focused on the individual NAND Cell, NAND strings and arrays. This section digs deeper into arrays and builds up to the page and blocks of NAND flash.

For a quick review, single NAND flash cells individually storing a single bit of 0 or 1 are joined together in strings and arrays to form much larger data storage structures. These strings are connected to allow storage and retrieval of data from selected cells. Very large data storage devices are made possible by adding more and more NAND cells to the array.

The NAND Flash String

The next Image shows the much larger array with control and data lines as well. Highlighted in green with a yellow background is the NAND String discussed in the previous article. There are many strings in between depicted as dots.

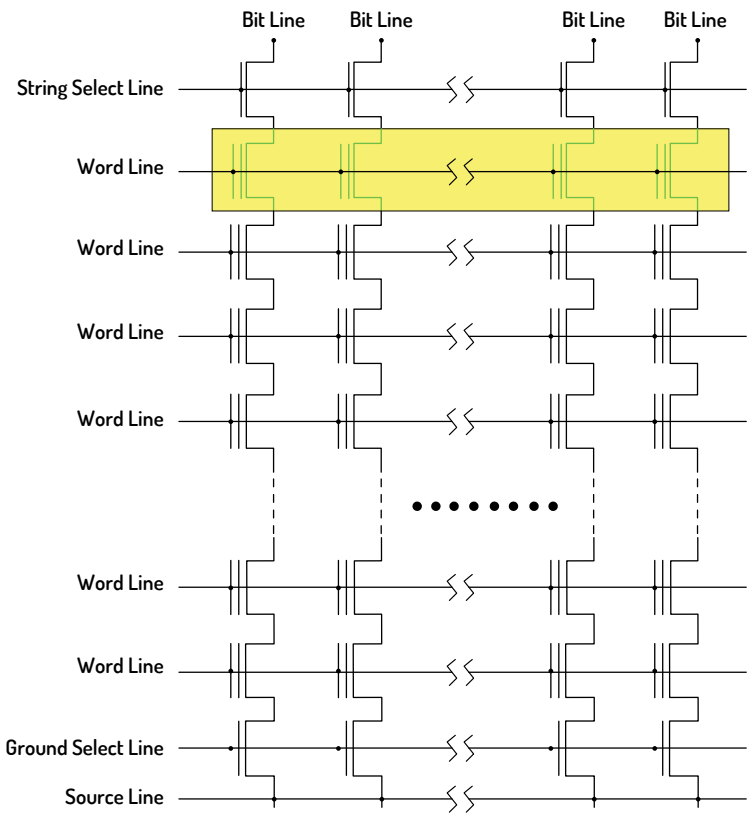
Strings (shown as columns) are the minimum unit to read and are typically comprised of 32 or 64 NAND cells. All strings in the array are connected at one end to a common Source Line (SL) and at the other end to the Bit Line (BL).



NAND Block

Each string also contains two control mechanisms in series with the NAND cells. String and ground select transistors are connected to the String Select Line (SSL) and Ground Select Line (GSL).

The NAND Flash Page



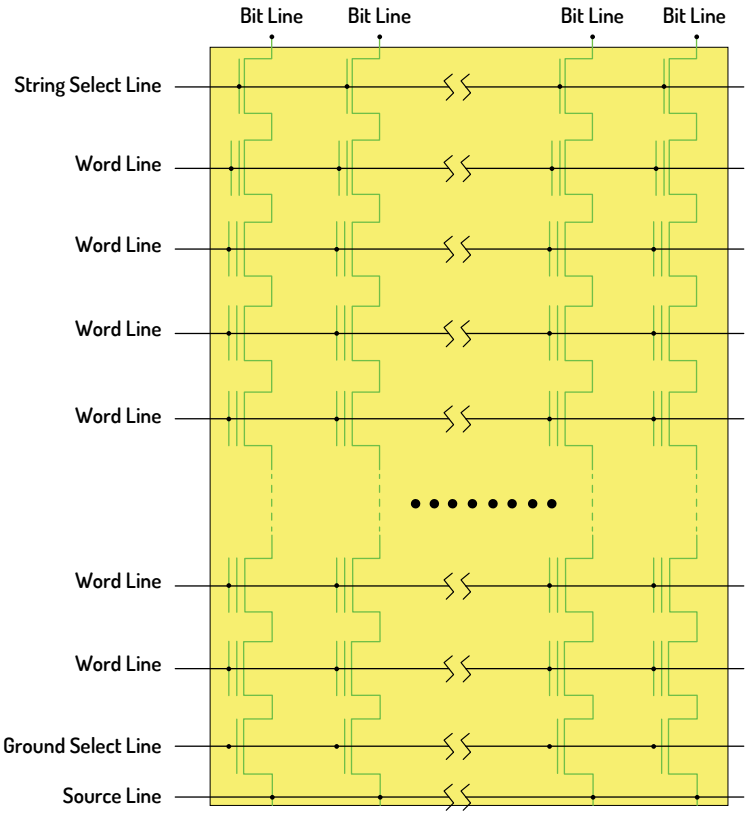
NAND Block

The image above shows the NAND Page with green lines and yellow highlighting.

Pages (shown as rows) share the same word line and are the minimum unit to program. They are typically comprised of at least 32,768 NAND cells, with many of the newer NAND devices have page sizes of 64K or 128K cells.

Most page sizes are referred to as 2K, 4K, 8K, etc. This signifies the page size in bytes. So if the page size has 32,768 NAND Cells (bits), this equates to 4096 bytes or 4K Page size.

The NAND Flash Block



NAND Block

This image above shows the NAND Block with green lines and yellow highlighting.

A block is a 2-dimensional matrix comprised of pages (rows) and strings (columns). The total number of bits in a block can be calculated by multiplying the number of strings by the number of pages.

From a Micron presentation at the 2014 Flash Memory Summit, maximum Pages per Block are approaching 512 and Block sizes are reaching up to 8 Mbytes.

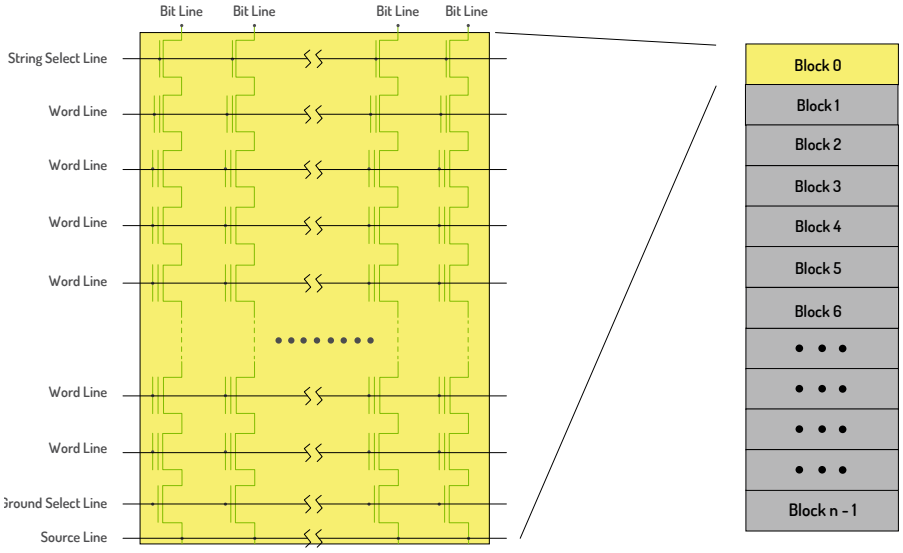
05 NAND Architecture - Planes and Die

Previous sections ranged from the basic NAND cell up to the block level. In this section we move up to discuss NAND planes and die.

As discussed in earlier, individual NAND cells are combined on Strings and Pages which are configured as columns and rows of an array. The overall array is called a Block. Some of the latest NAND components have block sizes as high as 8Mbytes.

This section will build from the Block level to show the pieces that make up a NAND die.

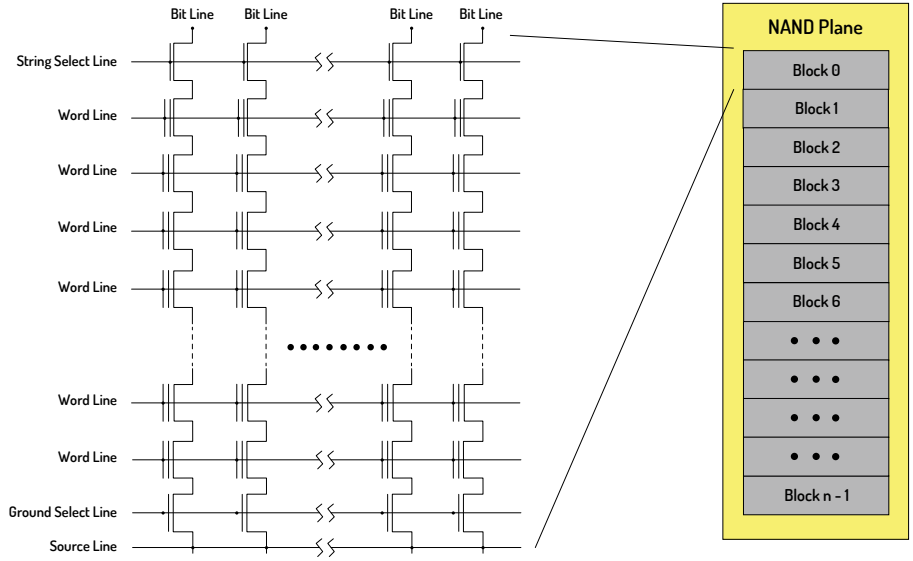
The NAND Block Structure



NAND Block NAND Architecture - Block Structure

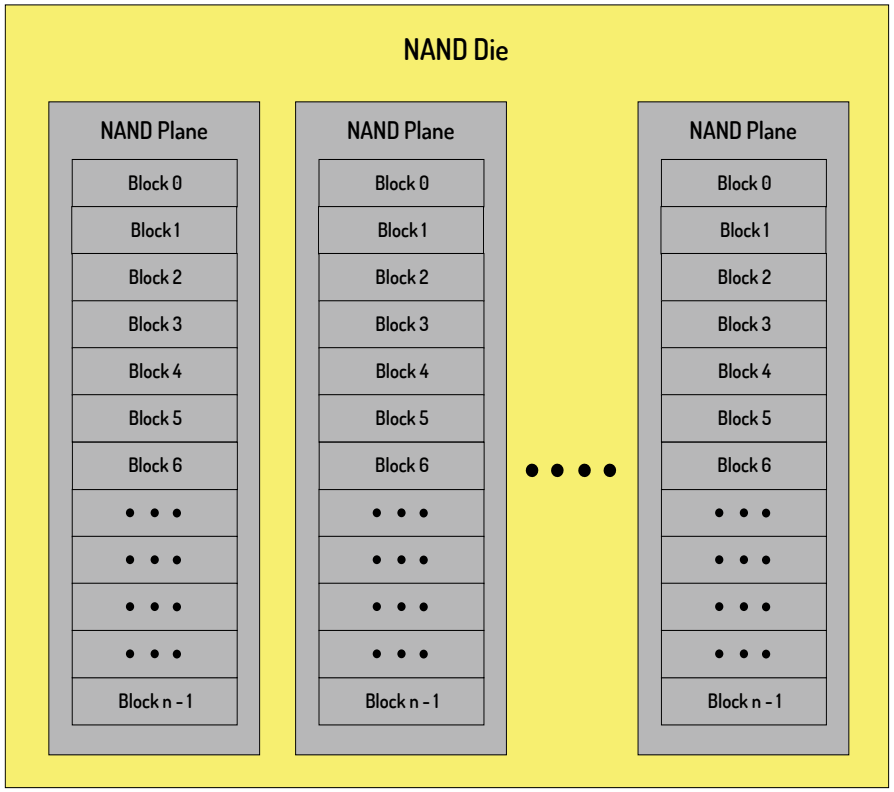
The NAND Block which is made up of a matrix of Strings and Pages is a building block for larger data structures. A single block is grouped together in a bank of many other blocks as shown in the illustration above.

The NAND Plane and Die



NAND Architecture - Block Structure

This bank of Blocks highlighted in yellow above is referred to as a Plane. One or many planes are grouped together to form a NAND die highlighted in the illustration on the next page. There are many configurations of die to meet many different design needs of OEMs.



NAND Die with Multiple Planes

A single die or multiple die stacked on top of each other are packaged into a usable form in popular JEDEC standard TSOP, BGA and other packages.

Over time as the requirements for additional storage continue, new ways of increasing density arise. One of the new technologies for high capacity SSD making its way to the forefront is 3D Memory.

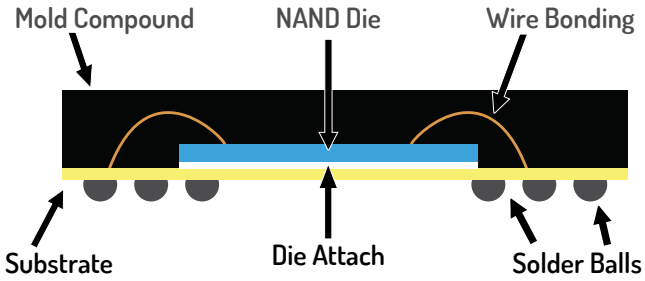
06 NAND Architecture - Component Packaging

Previous sections started at the basic NAND cell and built up to the NAND die level. We go up one more level in this section to discuss common NAND component packaging options.

The NAND die themselves are relatively fragile and require special equipment for placement and bonding. Typically NAND die are placed inside a protective component package as opposed to directly on a circuit board. Most follow an open industry standard defined by JEDEC.

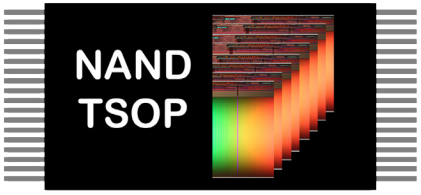
These components allow a manufacturer to place a single or several NAND die inside one package with a standard pin-out of typically a TSOP or BGA package. The standard packages are easily handled by the pick and place systems as they adhere the parts to printed circuit boards (PCBs) prior to being run through soldering machines.

BGA Packaged NAND



A simple single NAND die BGA component package image is shown above. There is a substrate which has a NAND die attached to it. Wire bonding machines connect the NAND die to the substrate which has via connections to the balls on the bottom side. After the wiring is complete, a molding compound encapsulates the top of the substrate providing for a rugged physical package.

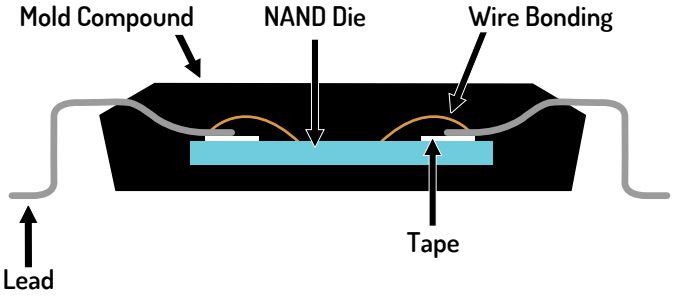
The illustration below shows a cutaway of a NAND component with multiple layers of die stacked on top of each other to create a single large capacity NAND memory device.



There is an insulating layer between each NAND die and connections are made from each NAND die to the substrate using a wire bonding machine. The substrate is like a very thin PCB (Printed Circuit Board) which is the base for the stacked die. As with the single die cutaway shown earlier, the entire top of the component is encapsulated.

If the parts share a JEDEC standard form factor, the parts appear physically identical regardless of the number of die inside.

TSOP Packaged NAND



This illustration shows a NAND die in a TSOP package. There are a couple of difference with the TSOP package from the BGA package. First, there are leads as opposed to balls that take the connections from the NAND die to the outside world. Second the entire assembly is encapsulated, not just the top. The only exiting connection is the end of the lead frame. As with the BGA package, multiple die can be stacked inside of a TSOP package.

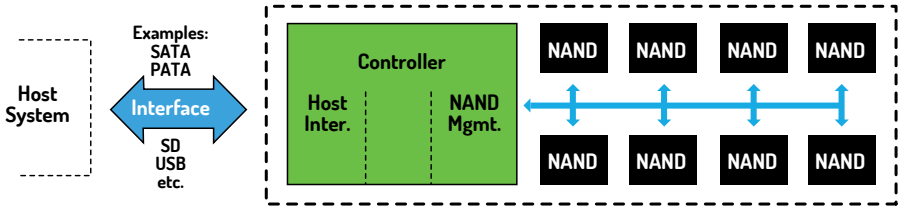
Future sections will look at putting the NAND components and a controller together to create a Solid State Drive as well as the challenges required of the SSD controller technology.

07 SSD Controller Architecture - Basic Overview

Previous sections described the NAND architecture from the basic NAND cell to a packaged component. This section begins to integrate the controller into the picture. Without a controller, the NAND is a relatively unintelligent storage device.

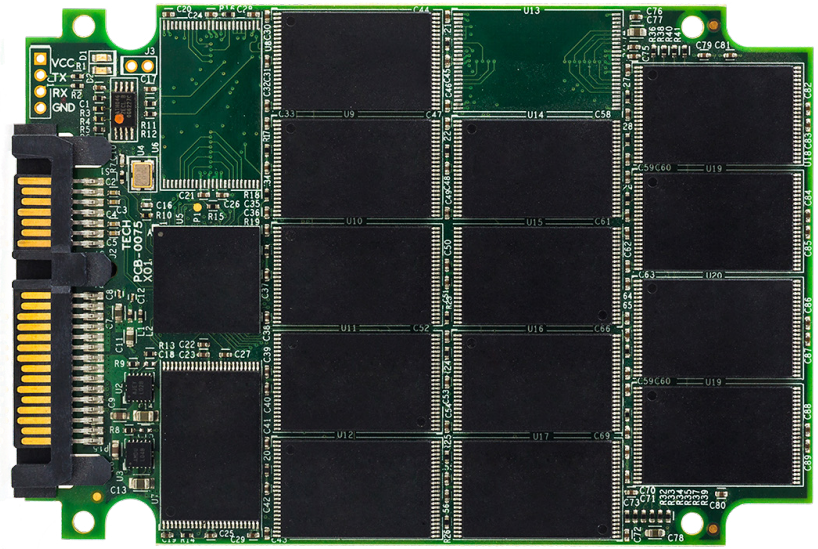
The reason for the controller function is to manage the NAND components and create a standard interface which communicates well with host systems. There are many popular interfaces today such as Serial ATA (SATA), SD, MMC, USB, PCIe as well as Parallel ATA (PATA, aka IDE).

All of these SSD interfaces have a common controller architecture design in which a controller resides between the NAND memory and the host system. In future articles we will look at the tasks a controller handles, but here we focus on the basic architecture of a generic Solid State Drive (SSD).



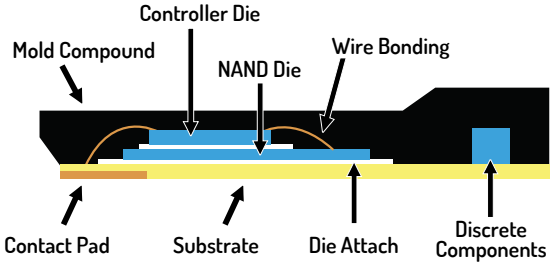
Basic Solid State Drie (SSD) Architecture

As shown to the left, the basic SSD consists of a controller chip which manages one or more NAND components, each of which could be comprised of multiple NAND die. The diagram is generic in the sense that it doesn't matter what final host interface is used.



As a real life example, see the image of a SATA SSD's internal circuit board above. All the rectangular components are NAND chips with the square component as the single controller for this SSD. The controller does not necessarily need to be in a square package, it just happens to be in this case.

Something you'll notice about the previous image is how tightly packed the NAND components are to each other. The limitation to an SSD's storage capacity is how many NAND die can be integrated into the industry standard package along with a controller's ability to address (read/write) each die.



Cutaway Illustration of microSD Card

For very small packaged SSD such as microSD cards, there is not sufficient physical space for packaged NAND and controller to be used.

In these cases, the controller die and NAND die are stacked on top of each other and connections are made with wire bonding.

The diagram above only shows a single NAND die in the microSD package, but multiple NAND die can be stacked with the controller to make higher density/capacity parts. The size of the die in the cross section is not proportional to the actual size for clarity. With wafer backgrinding techniques as many 16 NAND die have been successfully integrated into a single microSD card.

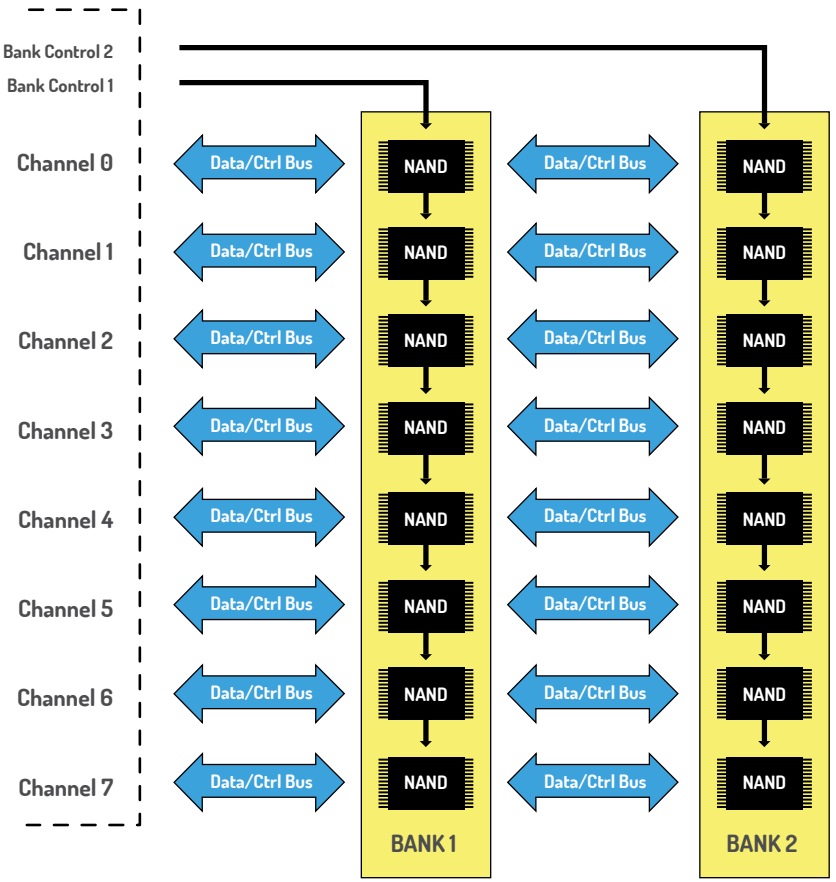
The next section will focus more on multiple channels, external RAM and other features common in today's Solid State Drive devices.

08 SSD Controller Architecture - Channels and Banks

This section focuses on the connection between the SSD controller and the NAND flash. There are many NAND configurations in SSD design and it makes a large difference to the SSD's overall power, performance and cost.

The illustration to the right shows a common 2.5" SATA III SSD NAND configuration. In this example, there are 8 Channels connected to the NAND chips. For each channel there are 2 Banks of NAND components.

There is a control line which selects either Bank 1 or Bank 2 to be active on the Data/Control Bus for a specific channel. This control line is connected to the Chip Select of each NAND component to enable or disable the component.



NAND Banks and Channels Illustration

NAND Channels

Channels refer to the number of flash chips the controller can talk to simultaneously. Low end SSDs usually have 2 or 4 channels; high end SSDs usually have 8 channels, some have 10 channels.

SSD manufacturers can trade off performance vs power consumption by stuffing less channels at time of manufacture. The limitation on more channels is added die size, pin count and power consumption, which all increase the cost.

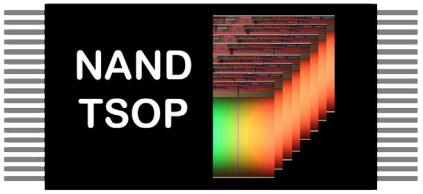
NAND Banks

Each flash chip at the same location in a channel together constitutes a bank. Refer to the diagram on the previous page. Each channel can have multiple chips. The limitation on maximum number of chips is a result of pin count, die size and cost considerations.

Additional SSD Performance Techniques

To further increase performance, controllers can take advantage of interleaving. Each NAND flash component can have multiple die in it, this is particularly so for high density parts. 2, 4 and 8 die packs are common.

The illustration below shows a cutaway of a TSOP NAND component with multiple layers of die stacked on top of each other to create a single large capacity NAND flash chip.



For a multi-die package, it is possible for each die to carry out a command; this is refer to as interleaving and can significantly increase device performance. The ability to interleave is dependent on flash, controller and firmware support.

Another mechanism to improve performance is multi-plane operation. A flash chip is internally organized in planes, with low density devices being single plane and higher density devices with 2, 4 or more planes.

In a multi-plane devices, it is possible for all planes to carry out a command in parallel (this is like interleaving but for a single die). Multi-plane operation, when available, can substantially improve device performance.

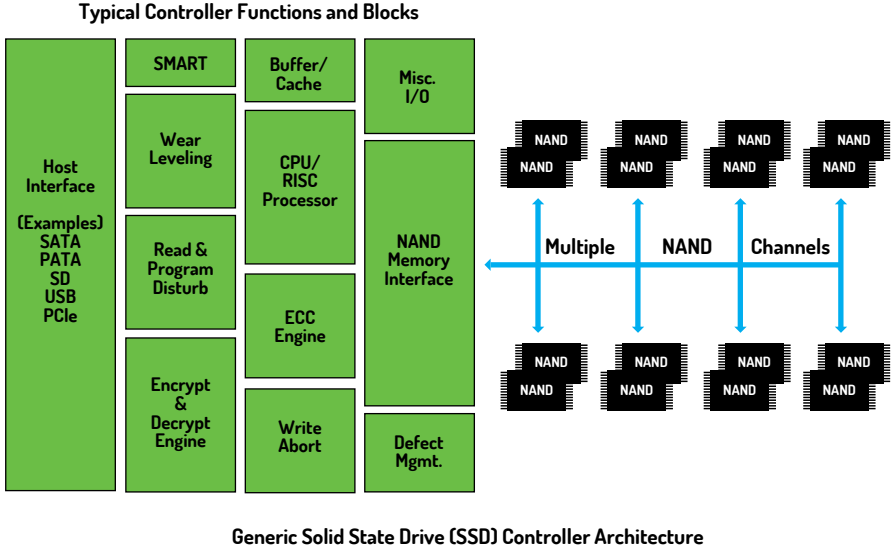
This section provided a basic understanding of the Channel and Bank architecture as well as performance enhancement techniques in an SSD. There are other more advanced techniques, such as copyback writes, cache reads, etc. not covered in this eBook.

09 SSD Controller Architecture - Block Diagram

This section focuses on the main blocks of a generic SSD controller and its connection to the NAND flash. Controller functionality varies with the type of product for which it is intended.

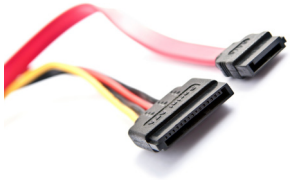
A simple consumer SD card controller is designed for cost and in some cases performance. For this application, it would be an overkill and unneeded expense to add an encryption & decryption engine to the silicon.

In other cases, such as secure military grade SSD, encryption & decryption may be an absolute necessity. Other applications rely on SMART data to predict an imminent failure looming in the future so the SSD can be replaced prior to an unexpected failure.



Generic Solid State Drive (SSD) Controller Architecture

The illustration above shows the basic blocks of a SSD. A brief description of each block follows.

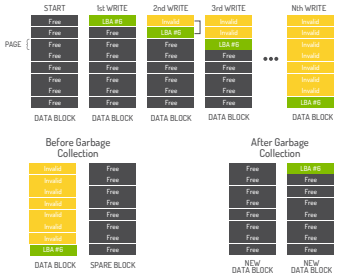


Host Interface

The host interface of a controller is typically designed to one industry standard interface specification. There are several interfaces made to address different system and design requirements. The most popular are SATA, SD, USB, PATA/IDE and PCIe.

SMART (Self-Monitoring, Analysis and Reporting Technology)

The SMART function, available in some controllers, monitors and records data regarding many attributes of the SSD and memory. An example is the ability to monitor the percentage of endurance cycles remaining in the SSD since this is a key determining factor of the life remaining.



Wear Leveling

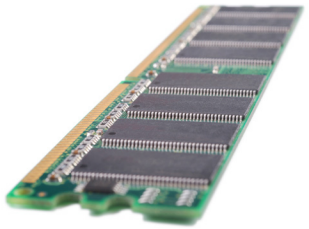
Wear Leveling is the ability to even out the number of write cycles throughout the available NAND. Since each NAND block has a limited number of erase/write cycles, if only one physical block is written continuously, it will quickly deplete its endurance cycles. A controller's Wear Leveling algorithm monitors and spreads out the writes to different physical NAND blocks.

Read & Program Disturb

With the finer and finer trace widths of the NAND flash, more issues arise to maintain the data contents of the NAND cells. Read & Program Disturb occur when cells are read or written causing cross coupling to adjacent cells and occasionally changing their values. Controllers need algorithms and in some cases circuitry to compensate for this phenomena.

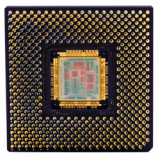
Encrypt & Decrypt Engine

For higher security applications, a hardware encryption and decryption engine is generally built into the silicon of the controller. The encryption engine is typically implemented in hardware to ensure speed for encrypting/decrypting on the fly. The most popular encryption method for SDDs today is AES256.



Buffer/Cache

Controllers generally have a high speed SRAM/DRAM cache buffer used for buffering the read and/or write data of the SSD. Since this cache uses volatile memory, it subjects data to loss if power is unexpectedly removed. It is common to see both internal caches in the controller chip itself as well as external RAM cache chips.



CPU/RISC Processor

The heart of every SSD is the main processing core. This can be a CPU or RISC processor. The size and performance of the CPU/RISC processor determines how capable the controller can be.

Error Check Correct

ECC Engine

Error Checking & Correction are a key part of today's SSD. ECC will correct up to a certain number of bits per block of data. Without ECC, many of the low cost consumer flash cards using very inexpensive memory would not be possible.

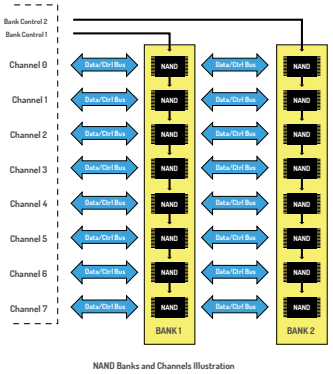
Write Abort

Write Abort is the when power to the SSD is lost during a write to the NAND flash. Without a battery or SuperCap backed cache, it is likely this data in transit will be lost. The more important aspect of this is to ensure the SSD's internal metadata and firmware remain uncorrupted. This is the function of Write Abort circuitry mainly found in Industrial Grade products.

Misc. Input Output

Miscellaneous I/O

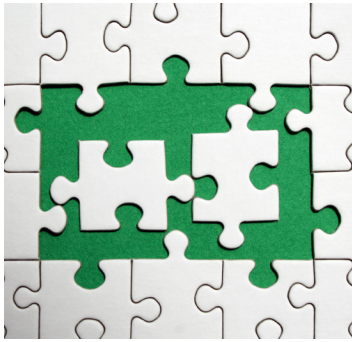
Simple functions such as chip select pins for the NAND components are handled with several input/output pins. There are also a number IO functions required for initial programming and production.



NAND Banks and Channels Illustration

NAND Memory Interface

The NAND memory interface was covered in the previous article on NAND banks and channels. Depending on the controller there can be a single NAND channel up to 10 or more. Each channel can have one or more NAND chips.



Defect Management

Every controller needs a method to deal with bad blocks of memory and new defects. At the point a NAND block becomes unusable, some action on the SSD controller's part must happen. In some cases a spare sector replaces the failed block. In a poor controller design, the SSD fails. Each controller has its method to deal with defects.

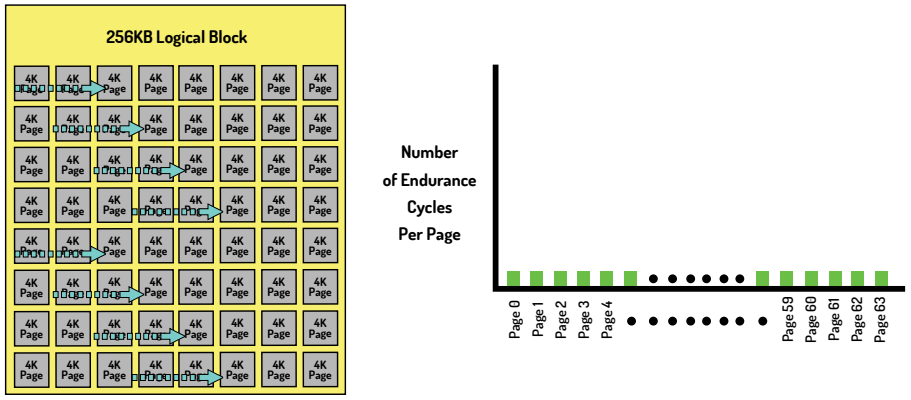
10 SSD Controller Functions - Wear Leveling

This section takes a look at how SSD controllers use Wear Leveling algorithms to compensate for the finite number of erase cycles inherent in NAND flash blocks.

A SSD controller receives commands from the host system which tell it where to read or write a piece of data. For simplicity of this article on Wear Leveling we will make two assumptions: 1) Each piece of data is 4KB and; 2) The NAND pages are also 4KB. In real world situations, the sizes of the data and page sizes can vary depending on host system and NAND memory used.

The host system provides the Logical Block Address (LBA) of the data it would like to read or write. It would be relatively straight forward for the SSD controller to simply read or write the LBA to the exact same Physical Block Address. Let's take a look at what would occur.

Wear Leveling and Sequential Writing to NAND Memory



Sequentially Written Flash Block

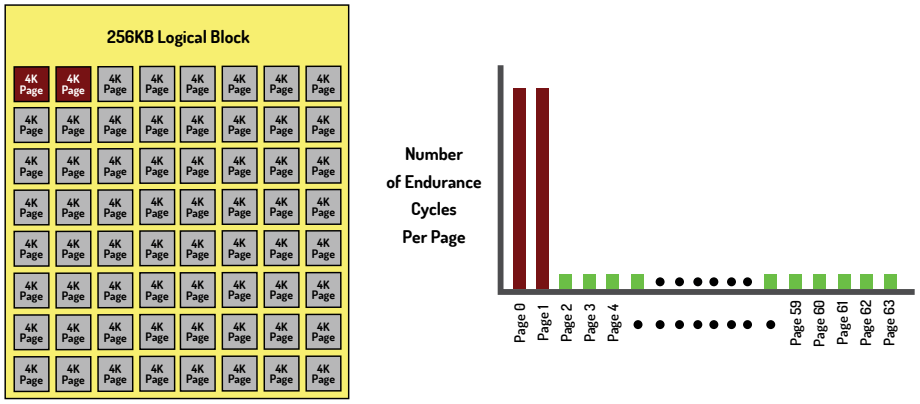
The image to the left shows a 256KB NAND block which is comprised of 64 Pages containing 4KB of storage capacity each. If the host system writes 256KB of data in sequential LBAs and the SSD controller stores this data in sequential Physical Block Addresses - starting at the first 4K Page and ending at the 64th in the block - then there has effectively been 1 endurance cycle used for the entire block.

The chart to the right of the 256KB block shows how sequential data is the ideal method of storing data in a NAND flash device. It evenly distributes all of the write cycles to the NAND Pages and Blocks so that one individual NAND Page/Block is not worn out prior to other Pages/Blocks.

Before continuing a couple clarifications:

- An Endurance cycle only occurs when an erase occurs, so on the first write, there is actually not an endurance cycle.
- Reading a NAND cell does not affect the endurance cycles of the NAND cell.

Wear Leveling and Non-Sequential Writing to NAND Memory



Non-Sequentially Written Flash Block

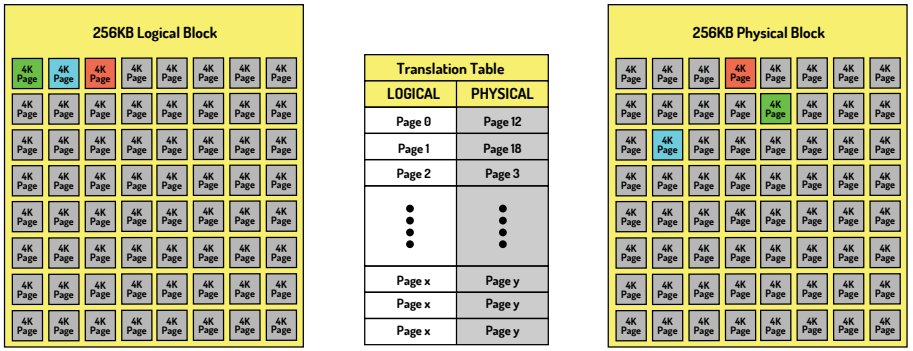
Unfortunately, real life usage of the SSDs rarely have the ideal sequential usage patterns. There are File system directories which need to be updated after any file is altered. There are mismatches in the data sizes from the host and the Page/Block sizes of the NAND on the SSD. There is the need to reclaim previously written blocks and many other factors.

The Non-Sequential image above shows a worst case situation where data is constantly written to only the first two 4K Pages. If the SSD controller just continued writing this data to these physical pages of the NAND, it would quickly exhaust the total number of endurance cycles of these cells.

What Does Wear Leveling Do?

Wear Leveling algorithms in SSD controllers attempt to evenly distribute host system writes throughout the entire SSD. Since NAND flash has a finite number of writes per block, wear leveling attempts to use every endurance cycle of the SSD prior to the end of its useful life.

There are many different schemes used by the different SSD controller designers, but they all share a couple characteristics.



Logical to Physical Translation

As shown above, they store the host data written for a Logical Block Address (LBA) to a physical location that has the least amount of endurance cycles used. Host data written to the same LBA is typically not stored in the same physical location of NAND. The controller must keep track of the translation from Logical to Physical block in a table or other method.

Another common occurrence is when static data on the SSD never moves - such as Operating System and Application data. It's stored once on the SSD and thereafter left alone. For these situations most new controllers will automatically move this static data to other physical NAND locations so they can take advantage of the endurance cycles of these NAND cells.

All alone, Wear Leveling cannot solve all an SSD's tasks, but it is an important part to creating a more reliable SSD which efficiently uses the limited endurance available in NAND memory.

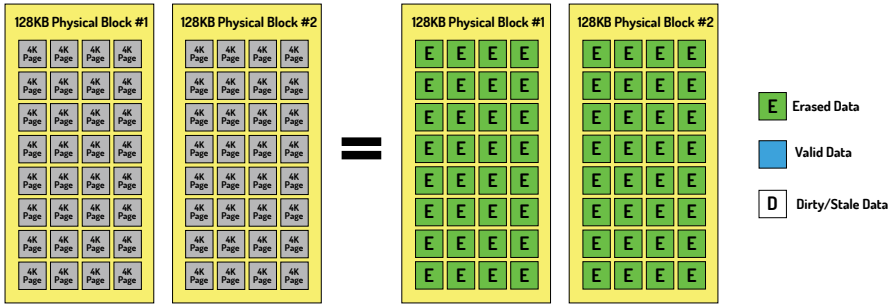
11 SSD Controller Functions - Garbage Collection

This section covers SSD controller's Garbage Collection algorithms. Garbage Collection is the process to reclaim previously written blocks of data so they can be rewritten with new data.

The reason there is a need for Garbage Collection is NAND's requirement to be erased prior to being written. A block is the smallest erasure unit of NAND flash and it is typically made up of 32 to 64 Pages.

A page of data can be written as long as it is already erased, but it is very inefficient to erase 32 or 64 pages in a block just to reclaim a single Page. There are many challenges related to Garbage Collection algorithms and efficiency, reliability and performance can be affected by its design.

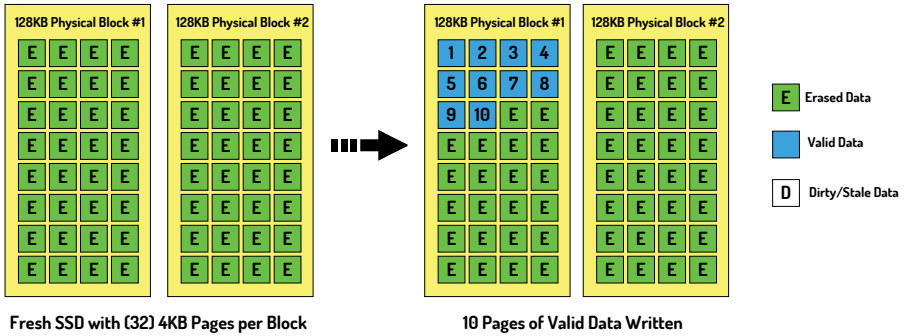
What Does Wear Leveling Do?



The diagram above shows two blocks of a fresh SSD which contain 32 Pages capable of storing 4KB of data each. This is a typical configuration of NAND and we will use it for the remainder of this section.

As you can see all of the Pages are erased from the factory and ready to be written. We will cover the other states of Valid Data and Dirty/Stale Data as we move forward.

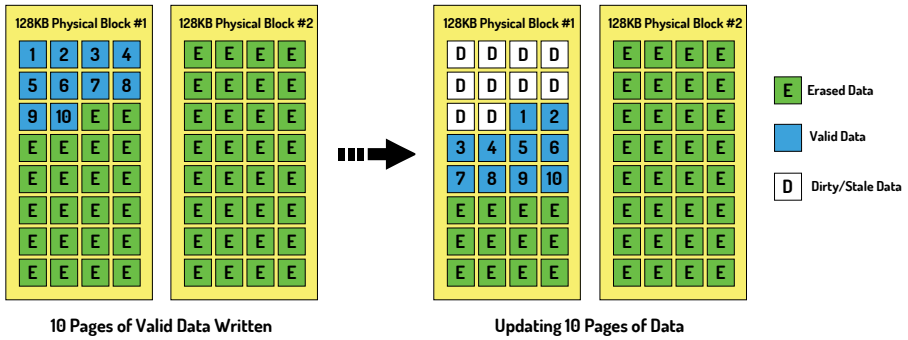
First Page Writes to Fresh SSD



On the first write to a fresh NAND block, things are easy. All pages are erased and it is just a matter of writing sequentially to these erased Pages.

Shown above is how a typical Garbage Collection algorithm would perform the initial task of writing 10 new pages of data on an already erased block.

Updating Page Data



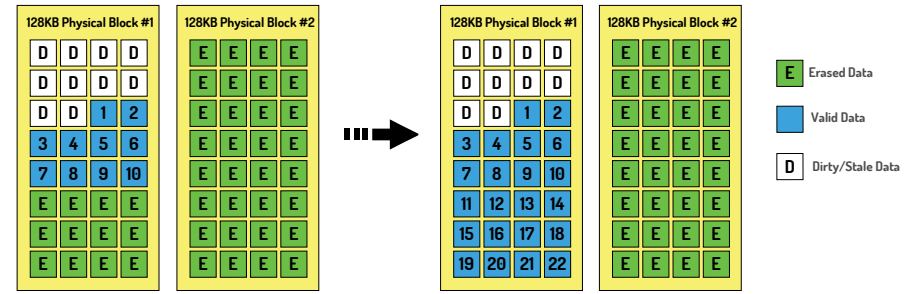
10 Pages of Valid Data Written

Updating 10 Pages of Data

We take it a step further in the above diagram by updating the 10 pages we originally wrote to the NAND. Since we would have to erase the entire block to erase the pages we want to update, we instead copy and write the updated data in the next available erased pages and mark the previous pages as dirty or stale.

At this point, the 10 pages are stored with valid data and we have 10 pages marked as dirty which cannot be written until the entire NAND Block is erased.

Filling the NAND Block



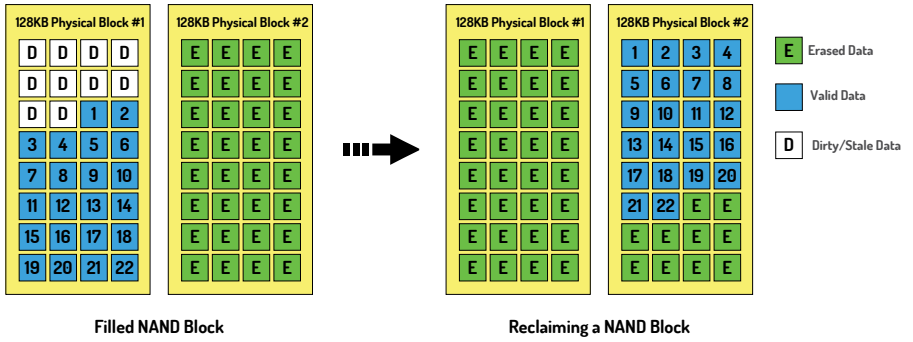
Mixed Block with Valid, Dirty and Erased Pages

Filling the NAND Block

This latest diagram above shows an additional 12 pages of data written to the NAND block. These new items 11 - 22 were placed in the final 12 erased pages at the end of NAND Block #1.

At this point, the entire NAND Block has been used. There are no longer erased pages to store data since there is only stored data or dirty data which needs to be erased to reclaim these pages.

What happens next - Garbage Collection?



Filled NAND Block

Reclaiming a NAND Block

Now the moment you've all been waiting for. What happens when all of the pages of the NAND block are either occupied with good data or previously written data which is no longer valid?

This is where Garbage Collection comes into play. To reclaim the NAND pages to an erased state, first any valid data in that block needs to be copied and written to the erased pages in a new NAND block. As you can see from the latest diagram, 22 valid pages of data were copied from the full Block #1 and written to the fully erased Block #2.

Once the valid data has been successfully written to the new block, then the entire Block #1 is erased. This is a one step process and it deletes any valid or dirty/stale data. Now Block #1 can be written to as if it were a fresh block.

The design of the Garbage Collection algorithm has a lot to do with other factors such as Write Amplification, so it is an important part of an overall SSD design.

12 SSD Controller Functions - TRIM Command

This section covers the TRIM command supported by some SATA, SCSI and other SSD controllers. The TRIM command is related to a SSD's Garbage Collection process described in the previous section.

With Garbage Collection, when the Operating System replaces LBA (Logical Block Addresses) which already contain data, such as during a file update/ overwrite, the SSD stores the updated data on fresh pages and marks the existing pages as Dirty (or Stale).

At a later time, Garbage Collection reclaims these dirty pages when it erases an entire block, which is the smallest area of erasure in NAND flash. This freshly erased block is now available to be written with new data.

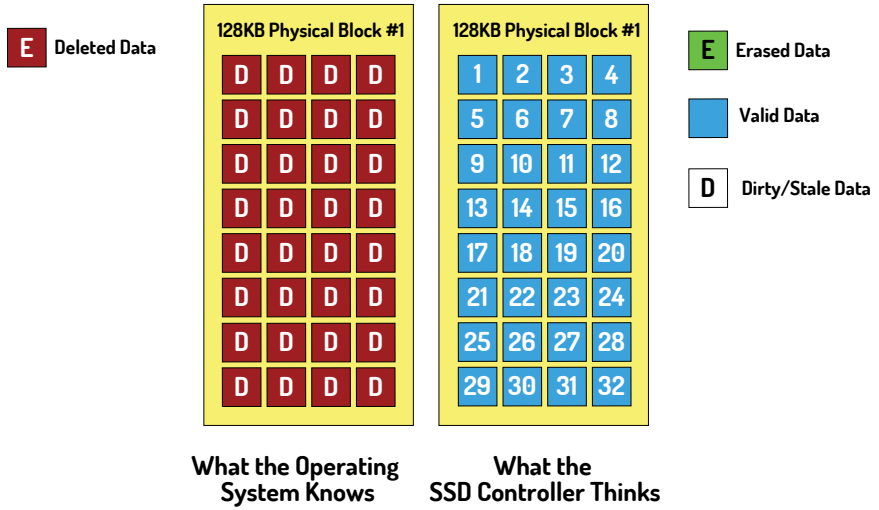
So Why the TRIM Command?

When the Operating system overwrites the same LBAs, and the SSD controller marks them as dirty for future erasure, it is clear there is no longer valid data in these pages.

But when a file is erased in the operating system, many times only the operating system's directory is updated. In these cases, there is no erase command sent to the SSD for the LBAs no longer considered valid by the operating system.

In comes the TRIM command. By issuing the TRIM command when a file is permanently deleted, it notifies the SSD that the associated LBAs no longer have valid data and can be marked as dirty for the next round of data collection.

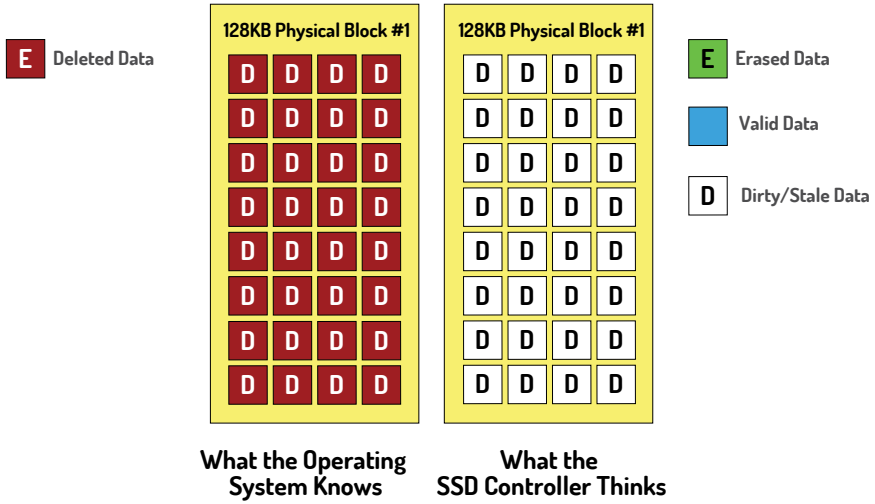
Example of Operating System and/or SSD without TRIM Command



For TRIM to be functional, the command must be supported by both the Operating System and the SSD. The newer versions of the Windows, Mac and Linux OS along as well as others support the TRIM command.

The illustration above shows a simplified example of a system without the TRIM command support. In this example, the host system has a file which is 32 pages in size (128KB), which has been deleted. The operating system knows this has been deleted and knows these are free areas (LBAs) it can use to write to.

But since the operating system simply marks these areas as available in the directory, there is no mechanism to tell the SSD controller these pages have been deleted. On the physical blocks of the SSD in the image on the right, you can see the SSD controller still believes there is valid data in these blocks. Therefore, it does not mark them as dirty and does not perform garbage collection on them until they are later overwritten by the operating system.



Example: Operating System & SSD with TRIM Command

For systems having TRIM support by both the Operating System and the SSD, the operating systems and SSD are in sync. After deleting a file, the operating system issues a TRIM command to the SSD with the deleted LBAs. The SSD then marks these LBAs as dirty and is able to reclaim them efficiently during its garbage collection process.

The illustration above shows a simplified example of the TRIM command. As in the previous example, the host system has a file which is 32 pages in size (128KB), which has been deleted. The operating system marks these areas as available in the directory, then issues the TRIM command with LBAs to the SSD. The SSD controller then marks these pages as dirty for garbage collection process.

13 SSD Controller Functions - Over-Provisioning

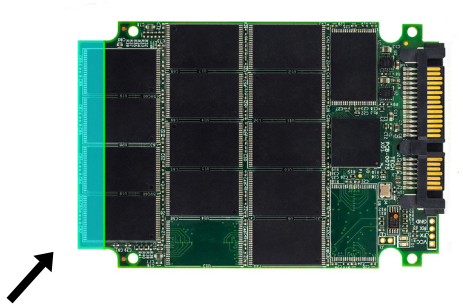
This section describes Over-Provisioning supported by many Solid State Drives, whether advertised on not. It is basically data space not available to the operating system which is used by the SSD to perform internal tasks such as bad block mapping, wear leveling and garbage collection.

The amount of Over-Provisioned area is generally set at the factory during the final low level formatting of the SSD. The typical percentages of Over-Provisioning are 0%, 7% and 28%.

An example of these percentages seen in the market would be a 128GB SSD which can be marketed as a 128GB with 0% Over-Provisioning, 120GB at 7% and 100GB at 28%. It is the same SSD with different amounts of "user-space" available.

Over-Provisioning Which is Typically Not Counted

One item that is not readily apparent to most users is the fact that NAND components are sold in binary capacity points versus SSD which are sold in decimal capacities. As shown to the right, there is a little over 7% of the SSD's NAND capacity which is not available to the user.



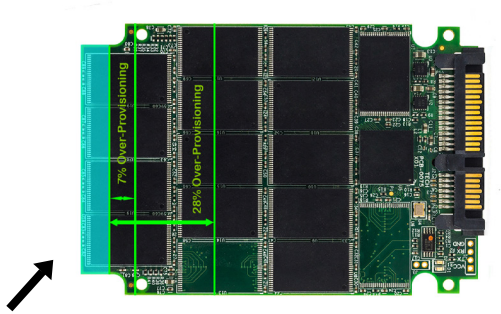
7% Difference in Binary to Decimal SSD Capacity used internally by SSD Controller

MARKETED CAPACITY	IDEMA DECIMAL CAPACITY (BYTES)	BINARY CAPACITY (BYTES)	DIFFERENCE
128GB	128,035,676,160	137,438,953,472	7.34%
512GB	512,110,190,592	549.755,813,888	7.35%

The International Disk Drive Equipment and Materials Association (IDEMA) has published a standard (LBA 1-03) for drive capacities that most HDD and SSD manufacturers follow. The 128GB and 512GB capacities are shown in the table above.

This binary to decimal difference is typically not counted as Over-Provisioning. The space is used internally by the controller for firmware storage, spare sectors, bad block mapping, wear leveling and other tasks similar to Over-Provisioning.

Over-Provisioning at 0%, 7% and 28%



7% Difference in Binary to Decimal SSD Capacity used internally by SSD Controller

0% OVERPROVISIONING	7% OVERPROVISIONING	28% OVERPROVISIONING
128GB	120GB	100GB
512GB	480GB	400GB

For this discussion, the 7% binary to decimal capacity is already removed. The table above shows the decimal 128GB and 512GB at the three most common Over-Provisioning percentages.

Over-Provisioning typically come in 0%, 7% or 28% chunks. Most of the lower capacity products have 0% Over-Provisioning in addition to many of the Industrial Grade devices built with the extremely high endurance SLC NAND.

For many client SSD built with MLC or TLC NAND on the market today, 7% Over-Provisioning is used and Enterprise SSD typically use 28%.

What are the Advantages and Disadvantages of Over-Provisioning?

The advantages of Over-Provisioning are to reduce Write Amplification while increasing endurance and performance. The key disadvantage of Over-Provisioning is loss of user capacity.

A user is trading higher reliability and more endurance and in some cases better performance for less usable space.

Cactus
Technologies

eBook

SSD 101

Cactus Technologies Limited
Suite C, 15/F, Capital Trade Center
62 Tsun Yip Street, Kwun Tong
Kowloon, Hong Kong
Tel: +852-2797-2277
Email: sales@cactus-tech.com

Cactus USA
3112 Windsor Road
Suite A356
Austin, Texas 78703
Tel: +512-775-0746
Email: americas@cactus-tech.com

Cactus-Tech.com