Technical Note

Design and Use Considerations for NAND Flash Memory

Introduction

NAND Flash devices have rapidly become the preferred choice for high-density, nonvolatile memory storage, particularly in mobile products. NAND Flash devices are frequently used with or supplant other types of memory in these applications.

NAND Flash device operation differs from that of other discrete memory devices. System designers must take these differences into account when interfacing the end system with the NAND Flash device. This technical note presents effective design and use practices to help avoid potential problems.

NAND Flash Memory Array Management

With use, memory cells that make up blocks of the NAND Flash memory array can wear out. This is true of all Flash technology. Robust systems take block failure into account and make appropriate adjustments.

Bad-Block Identification

Most NAND Flash devices, like all types of mass storage memory, include some initial bad blocks within the memory array. These blocks are typically marked as bad by the manufacturer, indicating that they should not be used in any system. NAND Flash device data sheets provide the location of bad-block markings.

Factory testing is performed under worst-case conditions, and those blocks that fail this testing are marked as bad. If a factory-marked bad block is used in a system, it may appear to operate normally but may cause other good blocks to fail or create additional system errors.

If a customer erases a factory-marked bad block, the marking is also erased. To avoid losing this information, it is best to do an initial read for bad blocks and create a bad-block table before issuing any PROGRAM or ERASE commands.

After the original bad-block table is created, it must be maintained and added to as additional blocks go bad with use.

Block Degradation and Tracking

Because good blocks in a NAND Flash device can degrade and wear out, it is important to track not only the initial, factory-marked bad blocks but also the blocks that go bad during normal device operation.

To detect whether a block has worn out and is bad, issue a READ STATUS command after any ERASE or PROGRAM operation. The READ STATUS command will report whether the previous ERASE or PROGRAM operation passed or failed.

Types of Failures

When failures occur during NAND Flash device operation, they are categorized as either permanent or temporary failures. A permanent failure is one from which the NAND Flash device cannot recover. Temporary failures can be corrected or avoided.

Permanent Failures

Permanent failures usually appear to the user as one or more bits in a NAND Flash page that are stuck in the 0 or 1 state and cannot be altered by PROGRAM or ERASE operations. When a permanent failure occurs, it is necessary to add the block to the bad-block table and avoid using it in the future.

Temporary Failures

Temporary failures can be minimized with proper NAND Flash design practices. If these failures occur, it is possible to recover the failing location of the memory array; the block need not be added to the bad-block table.

Temporary failures occur in multiple forms:

Program Disturb: A program disturb error occurs when one or more bits not intended to be programmed are changed during a PROGRAM operation. An increased number of partial-page programs to a page can exacerbate this error. Program disturb errors can show up on pages being programmed or on other pages within the same block.

To recover from this type of error, erase the block where the error occurred and reprogram the data to that block.

Read Disturb: A read disturb error occurs when one or more bits are changed during a READ operation. Read disturb errors occur within the block being read, but on a page or pages other than the page being read. Performing a high number (hundreds of thousands or millions) of READ operations on individual pages before an ERASE command for the block containing those pages can exacerbate this error.

To recover from this type of error, erase the block where the error occurred and reprogram the data to that block.

Over-programming: Over-programming can occur when a memory-cell threshold gate voltage on a bitline within a block goes too high. This prevents the memory cells from being read correctly and could result in incorrect programming of subsequent pages within a given block.

To recover from this type of error, erase the block where the error occurred and reprogram the data to that block.

Data Loss: During extended storage periods, some memory cells can lose data by changing via charge gain or charge loss. High ERASE/PROGRAM cycle counts can exacerbate data loss by inducing trapped charge in the gate oxide of the memory cells, effectively wearing out the gate oxide.

To recover from this type of error, erase the block where the error occurred and reprogram the lost data.

NAND Flash Controllers

Using the low-level commands (READ/PROGRAM/ERASE) to read and program data to NAND Flash devices is only one aspect of NAND Flash operation within a system. NAND Flash controllers can be implemented in a system to enhance NAND Flash performance. Controllers typically provide wear leveling, error correction code (ECC), and bad-block management.

A controller can be implemented in software, hardware, or a combination of the two. Generally, the controller resides between the host and the NAND Flash device and controls access to the device.

Wear Leveling

Because NAND Flash memory cells can eventually wear out, a robust NAND Flash implementation should include a wear-leveling strategy. Wear leveling is used to translate a logical memory address to different physical memory addresses each time a file is programmed. This operation is monitored and implemented by the controller connected to the NAND Flash device.

Wear leveling spreads NAND Flash memory-cell use over the entire range of the memory array, equalizing use of all the memory cells, and helping to extend the life of the device.

Error Correction Code (ECC) Requirement

Bit errors can occur during NAND Flash device operation or during long periods of inactivity (see "Block Degradation and Tracking" on page 1), making ECC use mandatory. ECC is generally characterized in terms of the number of bits the code can correct per 528-byte data sector checked. (A data sector is typically 528 bytes, but it may vary in some devices.) For example, if a controller has 1-bit ECC, it is able to correct one 1-bit error in each data sector. For ECC implementation details, refer to the documentation for the controller used.

Many different ECC implementations are available, and the extent of ECC protection a system requires depends on the type of data to be stored and the type of NAND Flash technology used (single-level cell or multi-level cell). To determine the level of ECC protection necessary, review the NAND Flash device data sheet and the requirements of the end system using the device.

Power Loss during NAND Array Operations

Power loss during NAND array operations (especially Program/Erase) is a violation of the NAND voltage specifications, which is not supported and should be avoided. The power supply voltage at point of reference at the NAND package voltage inputs must remain within their specified operative range for the entire duration of NAND array operations, until the device returns to a non-busy state. Power loss during NAND array operations, program or erase busy for example may cause data corruption in un-selected erase blocks or un selected pages in worst case. Issuing NAND commands outside of specified voltages must be avoided. If power loss during a NAND array operation is unavoidable then system level mitigation is needed such as:

- Bulk capacitance to the NAND voltage supply inputs (separate from host controller voltage supply inputs) sufficient to complete any combination of ongoing NAND array operations from the last point in time where the host controller could issue NAND operations – if sufficient bulk capacitance is supplied in this system level mitigation that should be able to cover all cases of power-loss during ongoing NAND array operations.
- The host upon detecting a power loss condition should issue a RESET command or toggle WP_ low to the NAND in an attempt to stop ongoing NAND array operations in a controlled manner prior to violating NAND voltage specifications – Note: due to multiple variables in a host system and when a RESET command is issued or WP_ is toggled low, the operations may not have sufficient time to complete prior to violation of NAND voltage specifications and if so may not always be effective.

In summary, the best system mitigation is to never violate NAND voltage specifications during NAND array operations as per datasheet requirements. The next best system mitigation would be to have sufficient capacitance to the NAND voltage supply signals to complete any combination of

NAND array operations from the last point in time where the host controller could issue NAND operations. A third system level mitigation is for the host monitor its power supply and end ongoing NAND array operations by issuing a RESET command to the NAND and cease issuing NAND operations prior to violating NAND voltage specifications such that any combination of ongoing NAND array operations has sufficient time complete their RESET operation within the NAND operational voltage specifications.

Best Practices for NAND Flash Design

NAND Flash is usually designed into systems that require nonvolatile, solid-state memory. These systems include designs built from the ground up with NAND Flash, designs using NAND Flash to replace nonvolatile memory (NOR Flash or a hard drive), and designs that never included nonvolatile memory.

Adoption in any of these categories, but especially in the latter two, may inadvertently bypass best practices for NAND Flash system design. This section highlights examples of less-than-optimal NAND Flash use observed in commercial products and explains design implementations that are best avoided.

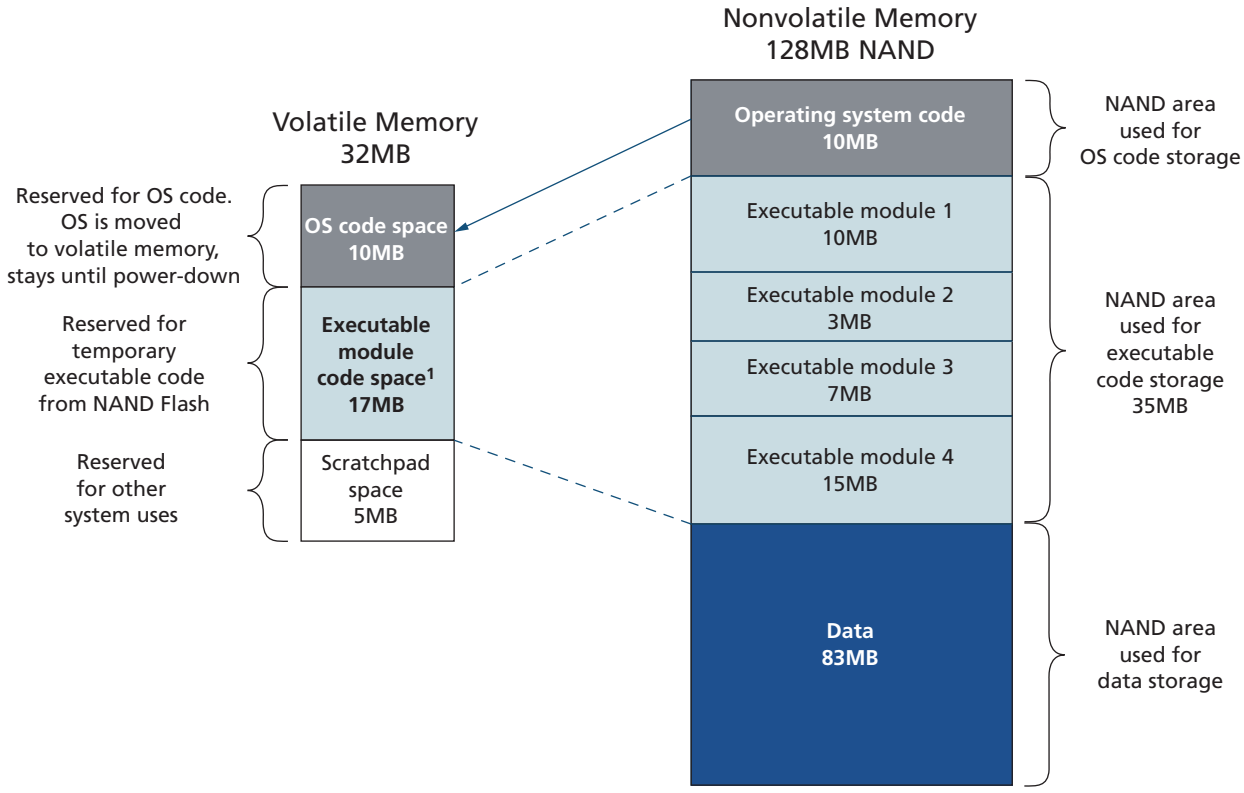
NAND Flash Not Always a Hard Drive Replacement

As higher-density NAND Flash devices become available, they are sometimes used to replace lower-density hard drives in mobile and embedded systems. This trend will likely continue as device densities increase and prices decrease.

Because NAND Flash is encroaching on the markets for mobile and embedded hard drives, designers tend to use NAND Flash in the same way they used hard drives. This usage model can be problematic because NAND Flash's susceptibilities differ from those of hard drives. One significant difference is that NAND Flash is susceptible to wear caused by repeated reads to the same physical area of the memory array.

In many applications, certain instruction sets are read over and over again. Demand paging is one means by which this is carried out with NAND Flash. Demand paging involves moving data (usually executable code) from the NAND Flash device to volatile memory (SDRAM or PSRAM), where it can be executed for system use. Generally, a system does not have enough volatile memory to hold all the needed executable code, so the execution code modules in the NAND Flash device must time-share the volatile memory (see Figure 1 on page 6).

Figure 1: Demand Paging



Notes: 1. This volatile memory area is used to execute temporary operational code stored in the NAND Flash device. This area is not large enough to store all the executable modules at once, so when one module is finished executing, another is copied in from the NAND Flash device, as needed.

Frequently, systems that use demand paging store the code to be accessed in a single place on the NAND Flash device. Accessing this code repeatedly, without erasing and reprogramming the data, causes errors to occur over time (see “Temporary Failures” on page 2). Many factors influence the point at which errors begin to appear, including how often the same physical area of the NAND Flash device is read and the extent to which ECC is used to maintain data integrity in the system.

Correcting Errors Due to Frequent Data Access

When errors accumulate to the point that ECC can no longer correct them, the system fails. Often, fundamental operating code is loaded into a system only once (as with a hard drive), and is expected to operate correctly during the operational life of the system. This expectation overlooks the inherent differences between a hard drive and NAND Flash. In many systems originally

designed for a hard drive, no mechanism is provided for reloading the affected data, which is necessary to support continued proper operation using NAND Flash. Three methods can be used to help avoid this problem:

Mimic a Hard Drive: Provide enough volatile memory space (SDRAM or PSRAM) to store all the executable code. With this approach, the code in the NAND Flash is read only once for each system power-on.

Build in Redundancy: Provide two or more copies of code that will be read repeatedly; store a master copy in a dedicated block of the NAND Flash device and a working copy in a different block. After a designated number of reads from the working copy (as determined by the system), erase and replace it with a fresh copy from the master block.

Use More Powerful ECC: Use a more robust ECC algorithm than is specified by the NAND Flash data sheet and set a threshold for the maximum number of bits allowed to go bad under the ECC correctable limit. When the threshold is met, move the data to another block within the NAND Flash device and begin reading from the new location.

Performance Enhancement Options

As mobile systems grow to support greater functionality and a wider variety of applications, the performance demands on individual components also increase. Application demands include longer battery life, smaller form factors, new features, and support for emerging technologies. Component device demands continue to increase with each subsequent generation of mobile systems. This rule applies to NAND Flash memory devices, which are finding their way into more and more mobile systems. The following two sections explain manufacturer and end-user performance enhancements for NAND Flash devices.

Manufacturer Enhancements

Two timing parameters, data programming time to the memory array (^{t}PROG) and data read time from the memory array (^{t}R), exert a major influence on NAND Flash performance. System designers cannot control either of these parameters, so timing improvements in these two areas are typically among the leading customer requests. Each new NAND Flash design effort focuses on reducing the time required for these two processes.

End-User Enhancements

Two timing parameters system designers *can* control, READ cycle time (^{t}RC) and WRITE cycle time (^{t}WC), can also boost NAND Flash performance.

It is common to find systems running ^{t}RC and ^{t}WC with timing values set three or more times longer than the published timings for the NAND Flash device in use. This is due in part to enhanced NAND Flash devices arriving in the marketplace ahead of the new controllers designed to take advantage of faster NAND Flash timing.

Using correct ^{t}RC and ^{t}WC values can improve NAND Flash performance as much as manufacturer enhancements to ^{t}PROG and ^{t}R. In the quest for overall program and read performance improvement, customers can realize significant performance gains with correct ^{t}RC and ^{t}WC timing.

Creating an Initial Bad-Block Table

Creating an initial bad-block table is a relatively simple process. However, the bad-block table must be created correctly to realize the benefits. A flowchart for creating an initial bad-block table is shown in Figure 2.

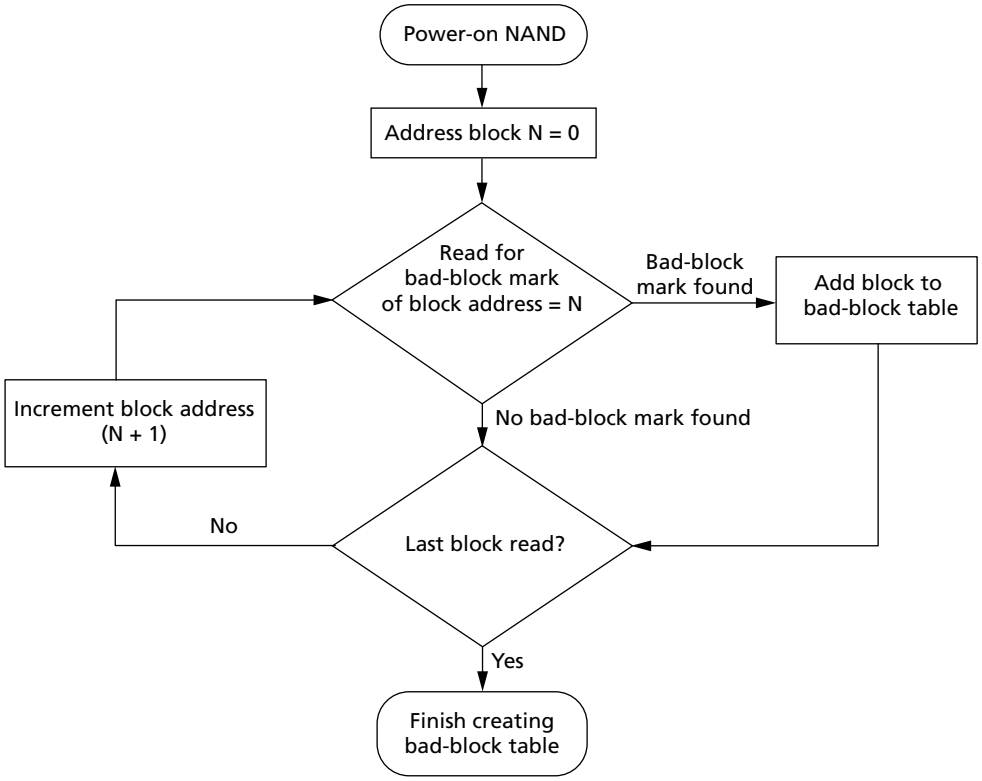
Each NAND Flash manufacturer uses a different method to mark bad blocks; the method can even vary among different devices from the same manufacturer. This means that the algorithm used to read bad-block information must be able to read the information correctly from any NAND Flash device that is used.

An algorithm capable of reading bad-block information for 2K-page, 2Gb, SLC Micron NAND Flash devices is provided in “Sample C Code Algorithm to Read for Bad Blocks” on page 10. This file and supporting files can be downloaded from the Micron NAND Flash Software Resources page under the NAND Flash I/O Drivers link at:
www.micron.com/products/nand/software/

After an initial bad-block table is created, new bad-block information must be appended as other blocks in the array wear out. The host device should reserve a marking location within each NAND Flash block and mark it if the block becomes bad. If the list of bad blocks becomes corrupted, it can be re-created by reading all the marked bad blocks directly from the blocks in the NAND Flash array.

8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-3900
www.micron.com/productsupport Customer Comment Line: 800-932-4992

Figure 2: Flowchart for Creating an Initial Bad-Block Table



Sample C Code Algorithm to Read for Bad Blocks

The following algorithm applies to a typical 2K-page, 2Gb, SLC NAND Flash device.

```
**************************************************************************
/*Read for Bad blocks and set up bad block table. */
  printf( Read Bad Blocks test.\n");
  for (i=0; i<NAND_BLOCK_COUNT; i++)
  {
    rc = NAND_ReadPage(i*64, 2048, 1, ucPageReadBuf);/* Read Block i, Page 0, Byte 2048 */
    if(ucPageReadBuf[0] == 0xFF)
    {
      rc = NAND_ReadPage(i*64+1, 2048, 1, ucPageReadBuf);/* Read Block i, Page 1, Byte 2048
*/
      if(ucPageReadBuf[0] == 0xFF)
          bb[i]=1;/*block is good */
      else
      {
          bb[i]=0;/*block is bad */
          printf("Block %d is bad!\n",i);
      }
    }
    else
    {
      bb[i]=0;/*block is bad */
      printf("Block %d is bad!\n",i);
    }
  }
**************************************************************************
```

Conclusion

The design considerations discussed in this technical note can help designers more fully exploit the capabilities of NAND Flash devices, ensure consistent data integrity, and deliver better performance in their embedded systems. Using these methods, designers can realize cost savings by using NAND Flash devices in place of other products.

For full Micron NAND Flash data sheets, visit the Micron Web site at: www.micron.com/products/nand/