Technical Note

Enabling Software BCH Error Correction Code (ECC) on a Linux Platform

Introduction

Many current designs are moving to NAND Flash memory to take advantage of its higher densities and lower cost for high-performance applications. However, a NAND Flash device may have bad blocks that require error correction code (ECC) to maintain data integrity.

Most new processor designs anticipate coupling with NAND Flash devices and include 4-bit and greater ECC engines within the hardware itself. However, some existing processor designs only support 1-bit ECC. This document addresses applications using these existing 1-bit ECC processors to enable Micron® MT29F1GxxABxDA, MT29F2GxxABxEA, MT29F4GxxABxDA, and MT29F1GXXABXEA NAND Flash memory devices with software BCH ECC.

Micron Single-Level Cell (SLC) NAND Error Management Requirements

NAND Flash memory requires ECC to ensure data integrity. Error management requirements for Micron's MT29F1GxxABxDA, MT29F2GxxABxEA, and MT29F4GxxABxDA NAND Flash memory devices are described in Table 1. Requirements for MT29F1GxxABxEA NAND are described in Table 2. Each SLC device includes 4-bit on-die ECC.

Table 1: ECC Requirements for MT29F1GxxABxDA, MT29F2GxxABxEA, and MT29F4GxxABxDA

Description	Requirement
Minimum ECC required	4-bit ECC per 528 bytes
Minimum ECC with internal ECC enabled	4-bit ECC per 516 bytes (user data) plus 8 bytes (parity data)
Minimum ECC required for block 0 if PROGRAM/ERASE cycles are less than 1000	1-bit ECC per 528 bytes

Table 2: ECC Requirements for MT29F1GxxABxEA

Description	Requirement
Minimum ECC required	4-bit ECC per 528 bytes

Introduction to ECC Algorithms

Simple Hamming codes provide the easiest hardware implementation for error correction. However, they can only correct 1-bit errors. Reed-Solomon codes provide more robust error correction capability and are used in many controllers currently on the market. BCH codes are also becoming popular due to their improved efficiency over Reed-Solomon codes.

BCH is the recommended ECC algorithm because:

- An arbitrary level of error correction is possible
- It includes efficient code for un-correlated error patterns (such as 1-bit errors, which are more typical for NAND devices)
- Is more space efficient when compared to other ECC algorithms' requirements for fitting into the NAND page spare area

The spare area for Micron NAND devices is set up with BCH ECC algorithm space requirements for a given ECC protection level. A comparison between the different types of ECC is provided in Table 3.

Table 3: ECC Algorithm Comparison

ECC Selection Factor	NAND Flash Error Characteristics	Hamming	Reed-Solomon	Binary BCH
Error Correction	Some error detection beyond the correction power of the code improves system performance.	Error detection is not inherent in the code, but can be added by increasing overhead.	An arbitrary level of additional error detection is possible.	An arbitrary level of additional error detection is possible.
Length of error patterns	Bit errors are un-correlated. When given a bit error, the probability of any other bit being in error is not increased.	Not applicable since it only corrects 1-bit errors.	Efficient code for correlated error patterns (such as burst errors).	Efficient code for un-correlated error patterns (1-bit errors).
Distribution of error patterns	Errors are randomly distributed within a page.	Good for single-bit random errors.	Good for randomly distributed error patterns.	Good for randomly distributed error patterns.
Frequency of errors	Raw error rates vary by technology, but errors are relatively rare.	Unacceptable when the raw error rate is greater than 10^{-9}.	Applies when error rates are less than approximately 10^{-4}.	Applies when error rates are less than approximately 10^{-4}.

Note: MT29F1GxxABxDA, MT29F2GxxABxEA, and MT29F4GxxABxDA devices support on-die ECC. MT29F1GxxABxEA devices do not support on-die ECC.

System Architecture

Software-Based BCH ECC Systems

There are primarily two solutions when using NAND to store code:

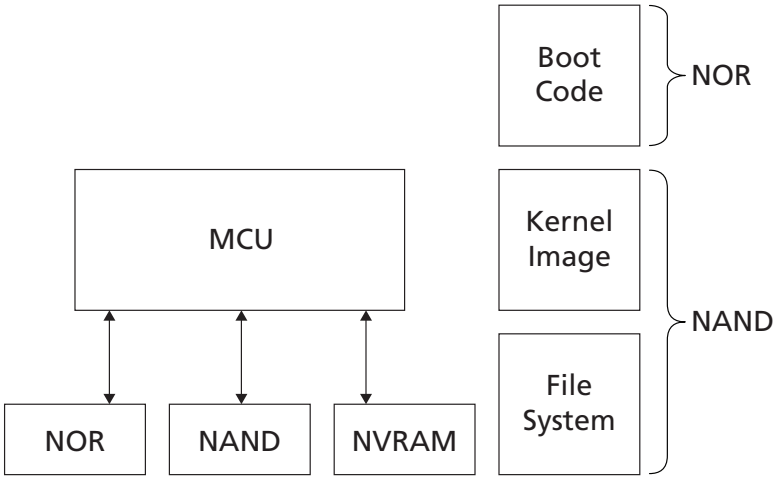
- NOR plus NAND plus RAM architecture
- NAND plus RAM boot method

These solutions are described in the following sections.

NOR Plus NAND Plus RAM Architecture

One solution when using NAND to store code is NOR plus NAND plus RAM architecture, which is illustrated in Figure 1.

Figure 1: NOR, NAND, and RAM Solution



In this solution, NOR is used for boot code and NAND is used for the kernel and file system. When the system boots up, boot code is first run directly from NOR or copied to nonvolatile RAM (NVRAM) by the MCU. Then, when boot code is running, it copies the NAND kernel image to RAM. Finally, the system is ready. For this solution, the MCU should support NOR boot (here, NOR is a low-density device; sometimes SPI NOR is used).

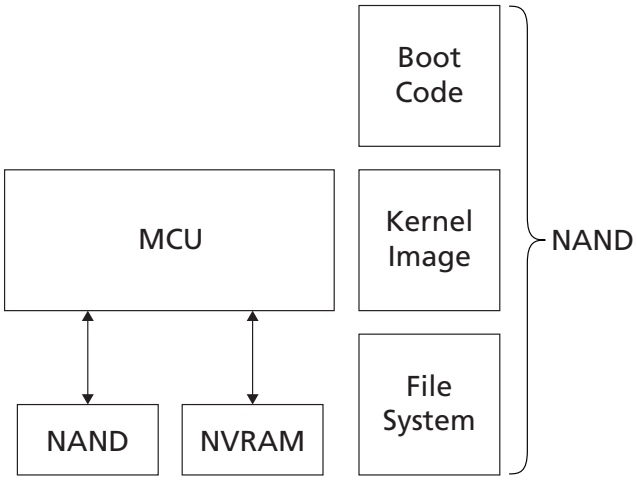
The advantages of this solution include:

- The reliability of NOR is better than NAND, so the system stability of boot segment is excellent and is most likely one-time programmed for users.
- It is easy to update software ECC to support new NAND technology, which requires more ECC bits.

NAND Plus RAM Booth Method

Another solution when using NAND to store code solution is the NAND plus RAM boot method, which is illustrated in Figure 2.

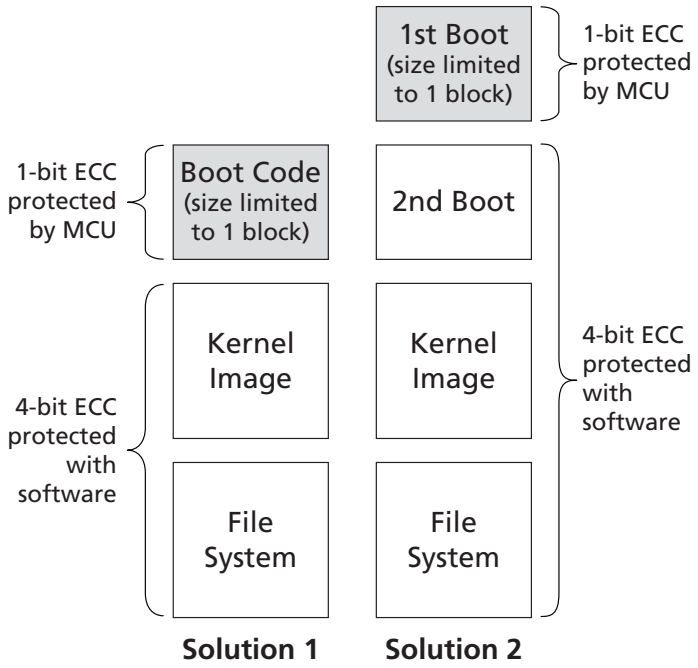
Figure 2: NAND Plus RAM Boot Method



In this solution, NOR is removed to reduce cost and all images are stored in NAND. In this case, the MCU must support a NAND boot solution where during system power up, ROM code in the MCU is run and the first few NAND blocks are copied. As previously noted, only the first block of MT29F1GxxABxDA, MT29F2GxxABxEA, and MT29F4GxxABxDA devices require 1-bit ECC, while all remaining blocks require 4-bit ECC. This means that when the MCU only supports 1-bit hardware ECC, Micron's on-die ECC or software ECC must be considered for all blocks except block one.

Figure 3 illustrates how to arrange the layout of the software ECC solution for only 1-bit hardware ECC supported by an MCU.

Figure 3: Layout of Software ECC for 1-Bit Hardware ECC



Whichever solution is implemented in the system, the boot code size or the first boot code size must be limited to 1 block and be stored on the first block of the NAND Flash device.

Choosing Between Hardware ECC, On-Die ECC, and Software ECC

For NAND error management, the following solutions can be used:

- Hardware ECC on the MCU
- NAND on-die ECC
- Software ECC

A comparison of each ECC method is provided in Table 4 to help you determine which method to use:

Table 4: ECC Type Comparison

ECC Type	Speed	Hardware Resources	Compatibility
Hardware ECC	Fast	Requires the MCU to support the related ECC engine.	Depends on host controller. If the MCU is not changed, the type of ECC and layout of ECC bytes are not changed. Different types of NAND Flash can be compatible.
On-Die ECC	Normal	Requires Micron Flash to support on-die ECC.	The compatibility between different types of NAND Flash is not optimal.
Software ECC	Slow	No special hardware limitation, but it needs to occupy much more CPU resources.	Provides the best compatibility, even between different types of MCU and NAND Flash.

If MCU hardware ECC can support enough bits of ECC, then hardware ECC should be selected. Otherwise, if compatibility is a concern but speed is not, software ECC can be used. However, if speed is more important than compatibility, then on-die ECC is recommended.

Performance Considerations

Software ECC is slower than hardware ECC, and as a result, it is important to consider the performance differences when using 4-bit BCH software ECC.

There are two factors that contribute to the performance impact:

- A CPU with a fast clock frequency will speed up the BCH calculation and reduce the overhead. Turning on D-cache/I-cache will also improve performance.
- Tuning the NAND controller to a tighter t_{RC}/t_{WC} timing will reduce the time spent during data transfers from the internal cache to the host. Even though this will not increase the software overhead, it will increase the software overhead rate.

Figure 4 and Figure 5 illustrate the performance impact data collected by Micron on an ARM9™ CPU at 400 MHz with I/D cache on.

Figure 4: Overhead Rate Versus Raw Speed

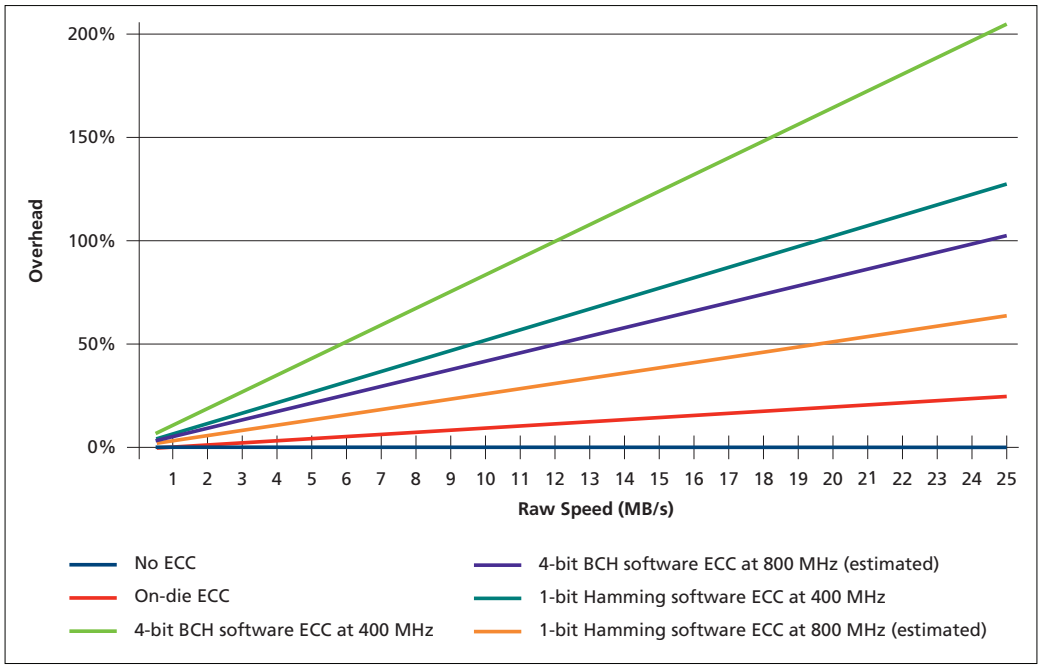
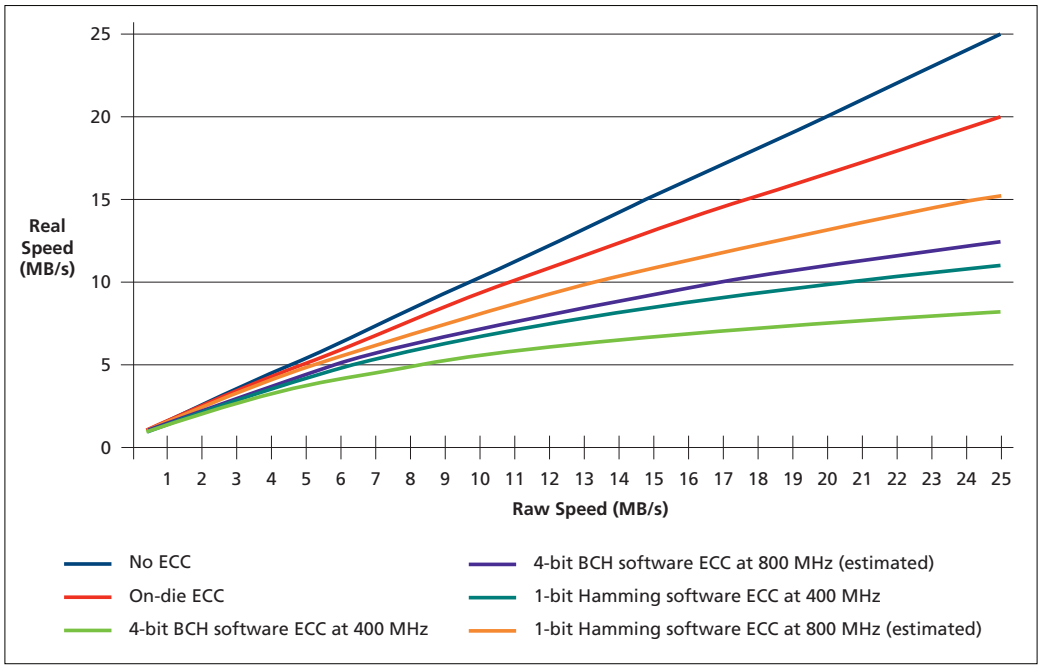


Figure 5: Real Speed Versus Raw Speed



Source Code Changes

This section provides an example of software BCH ECC patches on a Linux platform. This example assumes that a particular 1-bit processor is incorporated into the platform. Both the Uboot and Linux kernel code must be modified to support 4-bit BCH ECC. The following source code changes modify the Linux kernel code only. Similar changes are required for Uboot.

In the following Linux kernel code changes:

- Code in red indicates code that must be added
- Code in blue indicates code that must be removed
- Code in black indicates code that requires no change

MTD NAND Driver Modifications

1. Add the related macro's definition in */drivers/mtd/nand/Kconfig*:

```
config MTD_NAND_BCH
    tristate
    select BCH
    depends on MTD_NAND_ECC_BCH
    default MTD_NAND

config MTD_NAND_ECC_BCH
    bool "Support software BCH ECC"
    default n
    help
        This enables support for software BCH error
        correction. Binary BCH codes are more powerful
        and cpu intensive than traditional Hamming
        ECC codes.They are used with NAND devices requiring
        more than 1bit of error correction.
```

2. Modify the make file in */drivers/mtd/nand/Makefile*:

```
obj-$(CONFIG_MTD_NAND)          += nand.o
obj-$(CONFIG_MTD_NAND_ECC)      += nand_ecc.o
obj-$(CONFIG_MTD_NAND_BCH)      += nand_bch.o
```

3. Modify code in */drivers/mtd/nand/nand_base.c*:

```
#include <linux/mtd/mtd.h>
#include <linux/mtd/nand.h>
#include <linux/mtd/nand_ecc.h>
#include <linux/mtd/nand_bch.h>
```

In the nand_scan_tail function:

```
if (!chip->ecc.layout) {  //removed
if (!chip->ecc.layout && (chip->ecc.mode !=
NAND_ECC_SOFT_BCH)) {

        switch (mtd->oobsize) {
        case 8:
            chip->ecc.layout = &nand_oob_8;
        ...
    case NAND_ECC_SOFT:
            chip->ecc.calculate = nand_calculate_ecc;
            chip->ecc.correct = nand_correct_data;
            chip->ecc.read_page = nand_read_page_swecc;
            chip->ecc.read_subpage = nand_read_subpage;
            chip->ecc.write_page = nand_write_page_swecc;
            chip->ecc.read_oob = nand_read_oob_std;
            chip->ecc.write_oob = nand_write_oob_std;
```

```
            chip->ecc.size = 256;
            chip->ecc.bytes = 3;
            break;
        case NAND_ECC_SOFT_BCH:
            if (!mtd_nand_has_bch()) {
                printk(KERN_WARNING "CONFIG_MTD_ECC_BCH not
                enabled\n");
                BUG();
            }
            chip->ecc.calculate = nand_bch_calculate_ecc;
            chip->ecc.correct = nand_bch_correct_data;
            chip->ecc.read_page = nand_read_page_swecc;
            chip->ecc.read_subpage = nand_read_subpage;
            chip->ecc.write_page = nand_write_page_swecc;
            chip->ecc.read_page_raw = nand_read_page_raw;
            chip->ecc.write_page_raw = nand_write_page_raw;
            chip->ecc.read_oob = nand_read_oob_std;
            chip->ecc.write_oob = nand_write_oob_std;
        /*
         * Board driver should supply ecc.size and ecc.bytes
         * values to select how many bits are correctable; see
         * nand_bch_init() for details.
         * Otherwise, default to 4 bits for large page devices
         */
        if (!chip->ecc.size && (mtd->oobsize >= 64)) {
            chip->ecc.size = 512;
            chip->ecc.bytes = 7;
        }
        chip->ecc.priv = nand_bch_init(mtd,
                                       chip->ecc.size,
                                       chip->ecc.bytes,
                                       &chip->ecc.layout);
        if (!chip->ecc.priv) {
            printk(KERN_WARNING "BCH ECC initialization
            failed!\n");
            BUG();
        }
        break;
```

In the *nand_release* function:

```
void nand_release(struct mtd_info *mtd)
{
    struct nand_chip *chip = mtd->priv;

    if (chip->ecc.mode == NAND_ECC_SOFT_BCH)
```

```
nand_bch_free((struct nand_bch_control *)
chip->ecc.priv);
```

4. Modify */include/linux/mtd/nand.h* and add the *NAND_ECC_SOFT_BCH* definition:

```
typedef enum {
    NAND_ECC_HW,
    NAND_ECC_HW_SYNDROME,
    NAND_ECC_HW_OOB_FIRST,
    NAND_ECC_SOFT_BCH,
} nand_ecc_modes_t;
```

Then modify the *nand_ecc_ctrl* structure:

```
struct nand_ecc_ctrl {
    int prepad;
    int postpad;
    struct nand_ecclayout   *layout;
    void *priv;
    void (*hwctl)(struct mtd_info *mtd, int mode);
    int (*calculate)(struct mtd_info *mtd, const
    uint8_t *dat,
            uint8_t *ecc_code);
```

5. Add the header file */include/linux/mtd/nand_bch.h* and add the C source file *drivers/mtd/nand/nand_bch.c*.

Add the BCH Algorithm Library

1. Add the header file */include/linux/bch.h* and add the C source file */lib/bch.c*.
2. Modify lib/makefile:

```
obj-$(CONFIG_ZLIB_INFLATE) += zlib_inflate/
obj-$(CONFIG_ZLIB_DEFLATE) += zlib_deflate/
obj-$(CONFIG_REED_SOLOMON) += reed_solomon/
obj-$(CONFIG_BCH) += bch.o
```

3. Add the macro's definition in */lib/Kconfig*:

```
#
# BCH support is selected if needed
#
config BCH
    tristate
config BCH_CONST_PARAMS
    boolean
    help
        Drivers may select this option to force specific
        constant values for parameters 'm' (Galois field
        order) and 't' (error correction capability). Those
        specific values must be set by declaring default
        values for symbols BCH_CONST_M and BCH_CONST_T.
```

```
            Doing so will enable extra compiler optimizations,
            improving encoding and decoding performance up to 2x
            for usual (m,t) values (typically such that m*t <
            200). When this option is selected, the BCH library
            supports only a single (m,t) configuration. This is
            mainly useful for NAND flash board drivers requiring
            known, fixed BCH parameters.
config BCH_CONST_M
    int
    range 5 15
    help
        Constant value for Galois field order 'm'. If 'k' is
        the number of data bits to protect, 'm' should be
        chosen such that (k + m*t) <= 2**m - 1.
        Drivers should declare a default value for this sym
        bol if they select option BCH_CONST_PARAMS.
config BCH_CONST_T
    int
    help
        Constant value for error correction capability in
        bits 't'. Drivers should declare a default value for
        this symbol if they select option BCH_CONST_PARAMS.
```

NAND Host Controller Code Modification

Note: The following NAND host controller code modification assumes that S3C6410 is used as the hardware platform.

1. Modify code in the *s3c_nand_probe* function:

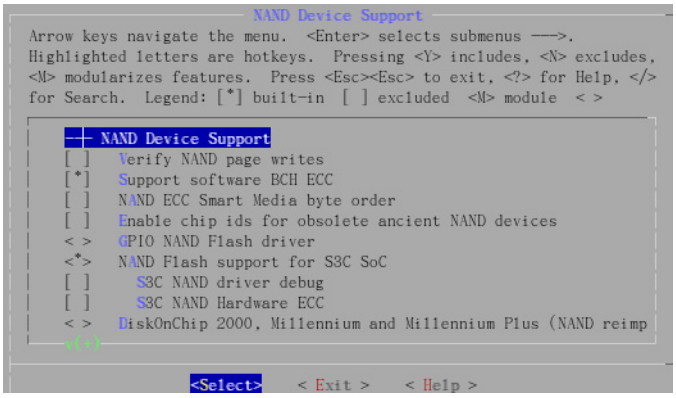
```
#if defined(CONFIG_MTD_NAND_S3C_HWECC)
    nand->ecc.mode  = NAND_ECC_HW;
...
        } else {
            nand_type = S3C_NAND_TYPE_SLC;
            nand->ecc.size = 512;
            nand->cellinfo = 0;
            nand->ecc.bytes = 4;
            nand->ecc.layout = &s3c_nand_oob_16;
        }

        printk("S3C NAND Driver is using hardware ECC.\n");
#else
        nand->ecc.mode = NAND_ECC_SOFT;  //removed
        nand->ecc.mode = NAND_ECC_SOFT_BCH;
        nand->ecc.size = 512;
        nand->ecc.bytes = 7;
        printk("S3C NAND Driver is using software ECC.\n");
#endif
```

2. Make *menuconfig* to define *CONFIG_MTD_NAND_BCH* and undefined *CONFIG_MTD_NAND_S3C_HWECC*.

3. The *S3C NAND Hardware ECC* option was originally selected. Deselect this option and select the *Support software BCH ECC* option instead, as shown in Figure 6.

Figure 6: NAND Device Support



Summary

This technical note describes one method for enabling 4-bit software BCH ECC when using Micron's MT29F1GxxABxDA, MT29F2GxxABxEA, MT29F4GxxABxDA, and MT29F1GxxABxEA NAND Flash memory devices in designs using processors that normally provide only 1-bit ECC. Because designs differ, each application may require some unique code development. In applications where the processor only supports 1-bit ECC, using Micron NAND Flash with software ECC can give greater flexibility in a design if system performances can be accepted.

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

Revision History

Rev. B 04/12

- Added “Performance Considerations” section
- Added MT29F1GxxABxEA devices

Rev. A 01/12

- Initial draft of document