Enabling NAND On-Die ECC for OMAP3
Using Linux/Android OS with YAFFS2

Introduction

Process shrinks on the newest generations of NAND Flash devices have lead to a gradual increase in the frequency of random data bit errors the longer a device is in use. To provide newest-generation devices with the same longevity as earlier NAND Flash devices, ECC enhancements to increase fault tolerance are required.

Most new processor designs anticipate coupling with NAND Flash devices, and include 4-bit and greater ECC engines within the hardware itself. Some existing processor designs only support 1-bit ECC. This document addresses applications using these existing 1-bit ECC processors. It includes instructions for enabling/disabling Micron® on-die ECC, describes mapping for the internal ECC and spare area, and provides an example for enabling the NAND Flash 4-bit on-die ECC feature on a TI OMAP3 platform with the Linux/Android operating system and the YAFFS2 file system (see “Appendix: Codebase Modification Example” on page 4).

Considerations

Micron NAND Flash MT29F1GxxABxDA, MT29F2GxxABxEA, and MT29F4GxxABxDA devices each include 4-bit on-die ECC, providing enhanced performance for applications using a board that supports only 1-bit ECC. There are two potential trade-offs associated with on-die ECC use; the first is speed. Processor-supported hardware ECC engines are typically very fast due to faster clock speeds compared to on-die ECC implementations.

The second potential trade-off is device longevity vs. speed. With NAND Flash memory, as a general rule for designs based on older processors, designers must make a choice between shorter usable life employing 1-bit ECC, or slower performance using 4-bit on-die ECC.
Enabling/Disabling On-Die ECC

Internal ECC can be enabled/disabled using SET FEATURES (EFh). The EFh command, followed by address 90h, followed by four data bytes (only the first data byte is used) will enable/disable internal ECC. To confirm the state of the internal ECC, issue the GET FEATURES (EEh) command.

<table>
<thead>
<tr>
<th>Action</th>
<th>Command Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable on-die ECC with EFh</td>
<td>EFh-90h-08h-00h-00h-00h-wait</td>
</tr>
<tr>
<td>Disable on-die ECC with EFh</td>
<td>EFh-90h-00h-00h-00h-00h-wait</td>
</tr>
<tr>
<td>Confirm the internal ECC state, then check the first data byte for enablement</td>
<td>EEh-90h-wait-read¹-read -read -read</td>
</tr>
</tbody>
</table>

Notes: 1. Only this data byte is used; the others are reserved. The on-die ECC state is defined as: 08h = enabled, 00h = disabled

Internal ECC and Spare Area Mapping for ECC

Internal ECC enables 5-bit detection and 4-bit error correction in 512 bytes (x8) or 256 words (x16) of the main area and 4 bytes (x8) or 2 words (x16) of metadata I in the spare area. The metadata II area, which consists of two bytes (x8) and one word (x16), is not ECC protected. During the busy time for PROGRAM operations, internal ECC generates parity bits when error detection is complete. See Figure 1 and Figure 2 on page 3 for x8 and x16 spare-area mapping.

During READ operations the device executes the internal ECC engine (5-bit detection and 4-bit error correction). When the READ operation is complete, read status bit 0 must be checked to determine whether errors larger than four bits have occurred.

Following the READ STATUS command, the device must be returned to read mode by issuing the 00h command.

Limitations of internal ECC include the spare area, defined in the figures below, and ECC parity areas that cannot be written to. Each ECC user area (referred to as main and spare) must be written within one partial-page program so that the NAND device can calculate the proper ECC parity. The number of partial-page programs within a page cannot exceed four.
# TN-29-56: Enabling On-Die ECC for OMAP3 on Linux/Android OS

## Internal ECC and Spare Area Mapping for ECC

### Figure 1: Spare Area Mapping (x8)

<table>
<thead>
<tr>
<th>Max Byte Address</th>
<th>Min Byte Address</th>
<th>ECC Protected</th>
<th>Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1FFh</td>
<td>000h</td>
<td>Yes</td>
<td>Main 0</td>
<td>User data</td>
</tr>
<tr>
<td>3FFh</td>
<td>200h</td>
<td>Yes</td>
<td>Main 1</td>
<td>User data</td>
</tr>
<tr>
<td>5FFh</td>
<td>400h</td>
<td>Yes</td>
<td>Main 2</td>
<td>User data</td>
</tr>
<tr>
<td>7FFh</td>
<td>600h</td>
<td>Yes</td>
<td>Main 3</td>
<td>User data</td>
</tr>
<tr>
<td>801h</td>
<td>800h</td>
<td>No</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>803h</td>
<td>802h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>807h</td>
<td>804h</td>
<td>Yes</td>
<td>Spare 0</td>
<td>User metadata I</td>
</tr>
<tr>
<td>80Fh</td>
<td>808h</td>
<td>Yes</td>
<td>Spare 0</td>
<td>ECC for main/spare 0</td>
</tr>
<tr>
<td>811h</td>
<td>810h</td>
<td>No</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>813h</td>
<td>812h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>817h</td>
<td>814h</td>
<td>Yes</td>
<td>Spare 1</td>
<td>User metadata I</td>
</tr>
<tr>
<td>81Fh</td>
<td>818h</td>
<td>Yes</td>
<td>Spare 1</td>
<td>ECC for main/spare 1</td>
</tr>
<tr>
<td>821h</td>
<td>820h</td>
<td>No</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>823h</td>
<td>822h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>827h</td>
<td>824h</td>
<td>Yes</td>
<td>Spare 2</td>
<td>User metadata I</td>
</tr>
<tr>
<td>82Fh</td>
<td>828h</td>
<td>Yes</td>
<td>Spare 2</td>
<td>ECC for main/spare 2</td>
</tr>
<tr>
<td>831h</td>
<td>830h</td>
<td>No</td>
<td></td>
<td>User data</td>
</tr>
<tr>
<td>833h</td>
<td>832h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>837h</td>
<td>834h</td>
<td>Yes</td>
<td>Spare 3</td>
<td>User metadata I</td>
</tr>
<tr>
<td>83Fh</td>
<td>838h</td>
<td>Yes</td>
<td>Spare 3</td>
<td>ECC for main/spare 3</td>
</tr>
</tbody>
</table>

![Spare Area Mapping (x8)](image1)

### Figure 2: Spare Area Mapping (x16)

<table>
<thead>
<tr>
<th>Max word Address</th>
<th>Min word Address</th>
<th>ECC Protected</th>
<th>Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0FFh</td>
<td>000h</td>
<td>Yes</td>
<td>Main 0</td>
<td>User data</td>
</tr>
<tr>
<td>1FFh</td>
<td>100h</td>
<td>Yes</td>
<td>Main 1</td>
<td>User data</td>
</tr>
<tr>
<td>2FFh</td>
<td>200h</td>
<td>Yes</td>
<td>Main 2</td>
<td>User data</td>
</tr>
<tr>
<td>3FFh</td>
<td>300h</td>
<td>Yes</td>
<td>Main 3</td>
<td>User data</td>
</tr>
<tr>
<td>400h</td>
<td>400h</td>
<td>No</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>401h</td>
<td>401h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>403h</td>
<td>402h</td>
<td>Yes</td>
<td>Spare 0</td>
<td>User metadata I</td>
</tr>
<tr>
<td>407h</td>
<td>404h</td>
<td>Yes</td>
<td>Spare 0</td>
<td>ECC for main/spare 0</td>
</tr>
<tr>
<td>408h</td>
<td>408h</td>
<td>No</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>409h</td>
<td>409h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>408h</td>
<td>40Ah</td>
<td>Yes</td>
<td>Spare 1</td>
<td>User metadata I</td>
</tr>
<tr>
<td>40Fh</td>
<td>40Ch</td>
<td>Yes</td>
<td>Spare 1</td>
<td>ECC for main/spare 1</td>
</tr>
<tr>
<td>410h</td>
<td>410h</td>
<td>No</td>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td>411h</td>
<td>411h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>413h</td>
<td>412h</td>
<td>Yes</td>
<td>Spare 2</td>
<td>User metadata I</td>
</tr>
<tr>
<td>417h</td>
<td>414h</td>
<td>Yes</td>
<td>Spare 2</td>
<td>ECC for main/spare 2</td>
</tr>
<tr>
<td>418h</td>
<td>418h</td>
<td>No</td>
<td></td>
<td>User data</td>
</tr>
<tr>
<td>419h</td>
<td>419h</td>
<td>No</td>
<td></td>
<td>User metadata II</td>
</tr>
<tr>
<td>418h</td>
<td>41Ah</td>
<td>Yes</td>
<td>Spare 3</td>
<td>User metadata I</td>
</tr>
<tr>
<td>41Fh</td>
<td>41Ch</td>
<td>Yes</td>
<td>Spare 3</td>
<td>ECC for main/spare 3</td>
</tr>
</tbody>
</table>

![Spare Area Mapping (x16)](image2)
Appendix: Codebase Modification Example

The balance of this technical note describes the necessary changes to the OMAP3530 Beagleboard codebase to enable NAND Flash 4-bit on-die ECC. Components used in this example are described in Table 2 on page 4.

The reader should have an understanding of the standard Linux/Android kernel and the bootloader building procedure, plus appropriate NAND Flash knowledge and the data sheet for the specific device(s) to be enabled. Micron technical note TN-29-45 provides instructions for enabling/disabling on-die ECC, and is available at www.micron.com.

Table 2: Example Configuration

<table>
<thead>
<tr>
<th>Platform</th>
<th>Micron NAND Flash</th>
<th>Linux Kernel Version</th>
<th>ARM Cross-Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMAP3530 Beagleboard1</td>
<td>MT29F4G16ABxDA</td>
<td>2.6.29</td>
<td>Date: 2008q3</td>
</tr>
<tr>
<td>Device Size</td>
<td>64 bytes OOB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
1. Subsequent testing on the OMAP3430SDP and the OMAP3430/OMAP 3630 Zoom platforms were also successful.
2. OOB = out-of-band; part of the spare area in NAND Flash reserved for error detection/correction code.

Device Code and Hardware Changes

On-die ECC enablement requires changes to X-Loader, U-Boot, and Kernel (uImage) code. This section provides a high-level view of necessary changes for each application. Also note that changes must be made to the OOB (spare) area layout of each page. This enables YAFFS2 to coexist with the ECC data in the OOB area (see Table 4 on page 13).

X-Loader (MLO) Change Sequence

1. **Step 1:** Add the NAND Flash SET/GET FEATURES commands.
2. **Step 2:** Use the NAND Flash SET FEATURES to enable the on-die ECC.
3. **Step 3:** Disable standard hardware and software ECC.
4. **Step 4:** Verify that NAND Flash timing requirements are met.

U-Boot Change Sequence

There are both mandatory and optional changes for U-Boot, presented in sequence as follows:

**Mandatory Changes:**

1. **Step 1:** Disable standard hardware and software ECC.
2. **Step 2:** Verify that NAND Flash timing requirements (tR::tR_ECC) are met. (In Micron testing/example, no changes were required; the default delay for tR was set to 100µs.)
3. **Step 3:** Check status for ECC fails after each READ operation.

**Optional Changes:**

1. **Step 4:** Add code to check whether on-die ECC was enabled by X-Loader and switch ECC mode accordingly.
**Step 5:** Modify the OOB layout to accommodate the new ECC map. (Required if the U-Boot will perform any NAND Flash WRITEs from within the U-Boot.)

 Kernel (uImage) Change Sequence

**Step 1:** Disable standard hardware and software ECC.

**Step 2:** Modify the OOB layout to accommodate the new ECC map.

**Step 3:** Poll for status after READ (required after every READ); check for errors beyond the 4 handled by the on-die ECC. When an error is logged, there is some form of data corruption.

**Step 4:** Verify that NAND Flash timing requirements are met, with sufficient time to complete the required status check.

Source Code Changes

Source Code Notes

The code samples and descriptions that follow present one method for enabling on-die ECC. While some of the items mentioned are mandatory, others are not. For instance, where the on-die ECC is turned on in X-load is irrelevant, as long as it is turned on. The reader is encouraged to use initiative when placing function calls; use the most efficient locations in the code for the individual customer application.

**Caution**

Disclaimer: There many variations in Linux source code. The code changes are generally subtle and may be difficult to identify because the end result for each version is the same. This can make integrating code somewhat challenging. The changes noted for the source code—file names, and other indicated differences—are for the Linux version shown in Table 2 on page 4; the reader may not encounter exactly the same paths or files when using a different version. Note that some porting may be required.

In the following code references necessary changes in the snippets are indicated by red text.

1 X-Loader (MLO):

1.1 File: .../include/common.h (Add #defines where other NAND_CMDs are located for the GET/SET FEATURES commands.)

1.1.1 Add the following definitions:

```c
#define NAND_CMD_READID 0x90
#define NAND_CMD_GET_FEATURES 0xEE /* ONFI */
#define NAND_CMD_SET_FEATURES 0xEF
#define NAND_CMD_RESET 0xff
```

**Note:** GET FEATURES and SET FEATURES commands are defined by ONFI as optional commands in the ONFI 1.0 specification. For ONFI definitions and specifications, see the ONFI Web site at www.onfi.org.
1.2 File: ...
drivers/k9f1g08r08.c

1.2.1 Comment out the line: \#define ECC_CHECK_ENABLE

1.2.2 Check in the function: nand_read_block() If the action in 1.2.1 disabled the bad block check, force it back on. Use a new \#define or remove these directives:

\#ifdef/\#endif

1.2.3 Add in nand_chip(); after READID and before NAND_DISABLE_CE():

set_on_die_ecc(); /* MICRON */

1.2.4 Add new function: set_on_die_ecc() with prototype somewhere in File: ...
drivers/k9f1g08r08.c
The GET FEATURES command call can also be added here to validate that on-die ECC is activated.

void set_on_die_ecc() /* MICRON */
{
  Int i, addr = 0x90;
  u16 data_out = 0x8;
  /* MICRON Send CMD #EF ADDR 08 00 00 00 */
  if (NanD_Command(NAND_CMD_SET_FEATURES)) {
    printf("ERROR: On-Die ECC\n");
  }
  NanD_Address(1, addr);
  delay(100);
  for(i=0; i<8; i++, data_out=0x0)
    WRITE_NAND(data_out, NAND_ADDR);
  delay(1000);

  return;
}

1.2.5 Increase delay in nand_read_page() and nand_read_oob() to account for tR_ECC; the delay is placed immediately following NAND_WAIT_READY():

- Old: delay(10000);
- New: delay(20000);

In this example, 20000 measured ≈ 80µs, exceeding the minimum required 70µs tR_ECC.
2 U-Boot

2.1 Mandatory Changes to File: .../drivers/mtd/nand/omap_gpmc.c

2.1.1 In function `board_nand_init()`, change `ecc.mode` to `NONE`

```c
nand->ecc.mode = NAND_ECC_NONE;
```

2.1.2 In function `board_nand_init()`, verify that the chip delay satisfies the new `tR_ECC` value:

```c
nand->chip_delay = 100;
```

2.2 File: .../drivers/mtd/nand/nand_base.c (also found at .../drivers/mtd/nand.c in some code). In function `nand_command_lp()`, add the following code to check the status after each READ operation.

```c
case NAND_CMD_READ0:
    chip->cmd_ctrl(mtd, NAND_CMD_READSTART,
                   NAND_NCE | NAND_CLE | NAND_CTRL_CHANGE);
    chip->cmd_ctrl(mtd, NAND_CMD_NONE,
                   NAND_NCE | NAND_CTRL_CHANGE);

    udelay(chip->chip_delay);

    chip->cmdfunc(mtd, NAND_CMD_STATUS, -1, -1);

    if(chip->read_byte(mtd) & 0x01){/* MICRON: check error bit */
        printf("ERROR: ON-DIE ECC READ FAILURE\n");
        // do something here to handle the failure.
    }

    /* MICRON: restore to data out */
    chip->cmd_ctrl(mtd, NAND_CMD_READ0,
                   NAND_NCE | NAND_CLE | NAND_CTRL_CHANGE);
    chip->cmd_ctrl(mtd, NAND_CMD_NONE,
                   NAND_NCE | NAND_CTRL_CHANGE);

    return;

/* This applies to read commands */
default:
```
2.3 Optional Changes to File: .../drivers/mtd/nand/nand_base.c (also found at .../drivers/mtd/nand.c in some code). To intelligently switch the ECC status based on the on-die ECC, add this group:

2.3.1 Define value for GETFEATURES NAND_CMD. Optionally, this can also be added to '.../include/linux/mtd/nand.h' with other NAND_CMD defines.

```
#define NAND_CMD_GET_FEATURES 0xEE /* MICRON */
```

2.3.1.1 To enable the GETFEATURES command, add the following snip into nand_command_lp().

```
if (column != -1 || page_addr != -1) {
    int ctrl = NAND_CTRL_CHANGE | NAND_NCE | NAND_ALE;

    /* MICRON: Single ADDRESS Sequence for GET_FEATURES */
    if(command == NAND_CMD_GET_FEATURES) {
        chip->cmd_ctrl(mtd, column, ctrl);
    }
    /* Serially input address */
    else {
        if (column != -1) {
            /* Adjust columns for 16 bit buswidth */
            if (chip->options & NAND_BUSWIDTH_16)
                column >>= 1;
            chip->cmd_ctrl(mtd, column, ctrl);
            ctrl &= ~NAND_CTRL_CHANGE;
            chip->cmd_ctrl(mtd, column >> 8, ctrl);
```

2.3.1.2 To test for on-die ECC enabled, add function and prototype as needed. Place function call where the test can be utilized; i.e., in nand_read().

```
/* ############################################## */
/* MICRON: Test on-die ECC status; return: 1/On or 0/Off */
int test_on_die_ecc(struct mtd_info *mtd)
{
    struct nand_chip *chip = mtd->priv;
    int addr = 0x90;
    uint8_t data;

    chip->cmdfunc(mtd,NAND_CMD_GET_FEATURES,addr,-1);
```
ndelay(1000);

data = chip->read_byte(mtd);

return((data & 0x08) ? 1 : 0 );

2.3.1.3 To provide ECC disabling, add external function prototype:

    extern void omap_nand_switch_ecc(int32_t);

2.3.1.4 To disable ECC, add this code somewhere after NAND initialization and before READ; i.e., before the start of nand_read(). This is also the place to add the optional ECC test. Note that value passed to omap_nand_switch_ecc() is a new case added to that function.

    if(chip->ecc.mode != NAND_ECC_NONE &\& test_on_die_ecc(mtd)) {
        omap_nand_switch_ecc(2);
    }

2.4 File: ../drivers/mtd/nand/omap_gpmc.c In function board_nand_init(), undo the changes made to ecc.mode in section 2.1.1. The mode will be selected later.

2.4.1 In function omap_nand_switch(), add an “if” clause for a new mode, “NONE.”

    /* Setup the ecc configurations again */
    /* MICRON */
    if (hardware == 2) {   // noecc using on-die
        nand->ecc.mode = NAND_ECC_NONE;
        nand->ecc.layout = NULL;
        printf("INFO:: NO ECC selected\n");
    }
    else if (hardware) {
        nand->ecc.mode = NAND_ECC_HW;
        nand->ecc.layout = &hw_nand_oob;
2.5 File: .../drivers/mtd/nand/nand_base.c. If writing to NAND while in U-Boot, the OOB layout must change. Replace the existing nand_oob_64 layout structure with the following new one:

```c
/* MICRON: modified OOB Layout */

static struct nand_ecclayout nand_oob_64 = {
    .eccbytes = 32,

    .eccpos = {
        8, 9, 10, 11, 12, 13, 14, 15,
        24, 25, 26, 27, 28, 29, 30, 31,
        40, 41, 42, 43, 44, 45, 46, 47,
        56, 57, 58, 59, 60, 61, 62, 63
    },

    .oobfree = {
        { .offset =  4, .length = 4 },
        { .offset = 20, .length = 4 },
        { .offset = 36, .length = 4 },
        { .offset = 52, .length = 4 },
    },
};
```
3 Kernel (uImage):

3.1 File: .../drivers/mtd/nand/omap2.c

3.1.1 In function, omap_nand_probe(), set the ECC to NONE.

    info->nand.ecc.mode = NAND_ECC_NONE;

3.2 File: .../drivers/mtd/nand/nand_base.c Replace the old OOB layout data structure with the new configuration:

    static struct nand_ecclayout nand_oob_64 = {
        .eccbytes = 32,

        .eccpos = {
            8, 9, 10, 11, 12, 13, 14, 15,
            24, 25, 26, 27, 28, 29, 30, 31,
            40, 41, 42, 43, 44, 45, 46, 47,
            56, 57, 58, 59, 60, 61, 62, 63
        },

        .oobfree = {
            { .offset =  4, .length = 4 },
            { .offset = 20, .length = 4 },
            { .offset = 36, .length = 4 },
            { .offset = 52, .length = 4 },
        },
    };

3.2.1 In function nand_command_lp(), in the switch(command) statement, declare variables to be used while polling for status.

    switch (command) {
        int status,cnt;    /* MICRON: local vars used */

        case NAND_CMD_CACHEDPROG:
3.2.1.1 Following **NAND_CMD_READSTAR** (after CMD #30 is issued), start polling for a READY status. Poll at least as long as the tR_ECC value. After READY is returned, check the ECC ERROR status. Readers will have to decide what to do for their specific applications when an ERROR is present.

```c
  case NAND_CMD_READ0:
    chip->cmd_ctrl(mtd, NAND_CMD_READSTART,
                    NAND_NCE | NAND_CLE | NAND_CTRL_CHANGE);
    chip->cmd_ctrl(mtd, NAND_CMD_NONE,
                    NAND_NCE | NAND_CTRL_CHANGE);

    /* MICRON: wait until part RDY; tR_ECC max */
    for(cnt=0; cnt<70; cnt++) {
      ndelay(1000);  /* 1 usec delay */
      chip->cmdfunc(mtd, NAND_CMD_STATUS, -1, -1);
      status = chip->read_byte(mtd);
      if( (status & NAND_STATUS_READY) )
        break;
    }

    /* MICRON: Look for STATUS bit ERR */
    if( (status & NAND_STATUS_FAIL) ) {
      printk(KERN_WARNING
             "WARNING :: READ Operation ECC Error: 0x%02x delay %d usec\n",
             status,cnt+1);
      /* do something on fail */
    }

    /* MICRON: re-issue CMD0 after STATUS Check */
    chip->cmd_ctrl(mtd, NAND_CMD_READ0,
                   NAND_NCE | NAND_CLE | NAND_CTRL_CHANGE);
    chip->cmd_ctrl(mtd, NAND_CMD_NONE,
                   NAND_NCE | NAND_CTRL_CHANGE);

    /* This applies to read commands */
    default:
```
Configuration Requirements

On-Die ECC Requirements

As noted previously, a modified out-of-band (OOB) layout is required for YAFFS2 to work properly with 4-bit on-die ECC. On-die ECC allocation requirements are listed in Table 3 and a graphic representation is provided in Table 4.

Table 3: OOB Layout Requirements for On-Die ECC (per 2KB Page)

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Required Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOB available bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>OOB bytes reserved for bad-block marking</td>
<td>8 bytes</td>
</tr>
<tr>
<td>OOB bytes not covered by ECC</td>
<td>8 bytes</td>
</tr>
<tr>
<td>OOB parity bytes required for on-die ECC</td>
<td>32 bytes</td>
</tr>
<tr>
<td>OOB bytes available, covered by on-die ECC</td>
<td>16 bytes</td>
</tr>
<tr>
<td>OOB bytes required by YAFFS2</td>
<td>14 bytes</td>
</tr>
</tbody>
</table>

Notes: 1. See Table 5 on page 14.

Table 4: 64B Per 2KB Page, OOB (Spare) Area Usage

<table>
<thead>
<tr>
<th>OOB/Spare Area: Standard Use</th>
<th>OOB/Spare Area: with On-Die ECC Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 ...</td>
<td>0 1 2 ...</td>
</tr>
<tr>
<td>Reserved bytes per data sheet</td>
<td>ECC parity bytes used by NAND.</td>
</tr>
<tr>
<td>ECC protected YAFFS2 metadata</td>
<td>Unused bytes not covered by ECC.</td>
</tr>
</tbody>
</table>
TN-29-56: Enabling On-Die ECC for OMAP3 on Linux/Android OS

NAND Loading Considerations:

YaFFS2 Requirements

YaFFS2 requires 14 bytes of metadata per 2KB page. (ECC bytes are omitted in this example because the device provides the ECC (see Table 5.) YaFFS2 metadata requirements are provided in Table 5.

Using a modified OOB layout, YaFFS2 can store metadata in the 16 bytes covered by the on-die ECC. This makes it possible to omit YaFFS2-specific ECC bytes. The available metadata area is interlaced with the on-die ECC bytes to enable YaFFS2 functionality.

Table 5: YaFFS2 Requirements for On-Die ECC

<table>
<thead>
<tr>
<th>Required Allocation</th>
<th>Required Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit chunk ID</td>
<td>4 bytes</td>
</tr>
<tr>
<td>32-bit object ID</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Number of data bytes in YaFFS2 chunk</td>
<td>2 bytes</td>
</tr>
<tr>
<td>Sequence number for this block</td>
<td>4 bytes</td>
</tr>
<tr>
<td>ECC on tags</td>
<td>Former 3 bytes not required</td>
</tr>
<tr>
<td>ECC on data</td>
<td>Former 12 bytes not required</td>
</tr>
</tbody>
</table>

NAND Loading Considerations:

Care should be taken when programming X-Loader, U-Boot, etc. into the NAND Flash device. The loader code used to program the NAND Flash may require modifications, depending on the method used to program the device. Table 6 lists the necessary partitions and settings for proper use.

Loading requirements are as follows:
- X-Loader must be programmed with 1-bit hardware ECC enabled.
- The 4-bit on-die ECC must be enabled prior to any programming. (The loader must be configured to enable/disable the on-die ECC feature.)

Table 6: Partition Settings and Loading Requirements

<table>
<thead>
<tr>
<th>Flow</th>
<th>Name</th>
<th>NAND Addr</th>
<th>Typical MTD Partition</th>
<th>NAND State During</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X-Loader</td>
<td>0x0 (Block 0)</td>
<td>Hardware 1-bit ECC</td>
<td>Hardware 1-bit ECC</td>
</tr>
<tr>
<td>2</td>
<td>U-Boot</td>
<td>0x00800000</td>
<td>On-die 4-bit ECC</td>
<td>On-die 4-bit ECC</td>
</tr>
<tr>
<td>2.1</td>
<td>U-Boot ENV</td>
<td>0x01C0000</td>
<td>On-die 4-bit ECC</td>
<td>On-die 4-bit ECC</td>
</tr>
<tr>
<td>3</td>
<td>Kernel</td>
<td>0x0200000</td>
<td>On-die 4-bit ECC</td>
<td>On-die 4-bit ECC</td>
</tr>
<tr>
<td>3.1</td>
<td>File System</td>
<td>0x2000000</td>
<td>On-die 4-bit ECC</td>
<td>On-die 4-bit ECC</td>
</tr>
<tr>
<td>3.2</td>
<td>User Data</td>
<td>0xC000000</td>
<td>On-die 4-bit ECC</td>
<td>On-die 4-bit ECC</td>
</tr>
<tr>
<td>3.2</td>
<td>Cache</td>
<td>0xE000000</td>
<td>On-die 4-bit ECC</td>
<td>On-die 4-bit ECC</td>
</tr>
</tbody>
</table>
Summary

This technical note describes one method of enabling 4-bit on-die ECC in Micron NAND Flash MT29F1GxxABxDA, MT29F2GxxABxEA, and MT29F4GxxABxDA devices for designers using processors that normally provide only 1-bit ECC. Using 4-bit on-die ECC in this way can help to extend the life of the NAND Flash device.

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 4-bit on-die ECC can give designers greater flexibility in their designs, along with increased longevity for the device.
Revision History

Rev. A ................................. 12/10

- Initial release