

Technical Note

Adapting the Linux Kernel for Micron® P30, P33, and J3 Flash Memory

Introduction

The Linux environment differs from other operating environments in several ways. The Micron® P30, P33, and J3 family of parallel NOR Flash memory devices can easily be adapted for the Linux environment via the memory technology device (MTD), and the Linux environment can be modified for correct operation with the devices.

This technical note provides a guide for modifying the MTD device-layer software for correct use with Micron P30, P33, and J3 devices. It also describes the modifications required in the Linux environment.

Enabling Buffered Programming in 2.4.x Kernels

Buffered programming features are not included in the MTD for kernel versions prior to 2.4.6. To take advantage of enhanced performance provided by the buffered programming feature, a proprietary `do_write_buffer()` function must be implemented and included in the Linux Flash driver.

Enabling P30, P33, and J3 1KB Buffered Programming

Compared to devices with smaller buffer sizes, the 1KB buffer sizes found in P30, P33, and J3 devices provide significant performance increases. For kernel versions 2.6.13 and later, the MTD driver automatically supports the 1KB buffer size (when the Flash memory is used in 16-bit mode). Enabling the larger buffer size in older versions of the kernel requires the code modification shown in the following example.

Apply this patch in the `mtd/cfi.h` file:

```
- static inline map_word cfi_build_cmd(u_char cmd, struct  
map_info *map, struct cfi_private *cfi)
```

```
+ static inline map_word cfi_build_cmd(u_long cmd, struct  
map_info *map, struct cfi_private *cfi)
```



Enabling J3 256Mb Buffered Programming – x8 Mode

J3 Flash memory can work in two modes: x8 mode and x16 mode. The modes refer to the Flash data bus sizes of 8 bits and 16 bits, respectively. The behavior of the two modes is similar; the buffer size for the x8 mode is set to 256 bytes instead of 1024 bytes. This causes a problem because in the CFI the value related to the buffer size is set to 1024 bytes independent of the data bus size. This means that when the J3 device is used in x8 mode, the Linux probe function reads the buffer size from the CFI and sets the internal structures to perform a PROGRAM operation that fills 1024 bytes of buffer. As a result, the PROGRAM fails. A code modification is required to avoid this failure.

Micron provides several patches for different kernels that are available to customers on the Micron Web site. The logic behind the modification is similar for each kernel, and it is often possible to port a patch designed for a specific kernel version into a different kernel. The patch is also submitted to the Linux communities, and upon approval it is made available for all future kernel deliveries. The following code defines a patch developed and validated for the 2.6.33 Linux kernel:

```
--- a/drivers/mtd/chips/cfi_probe.c    2011-05-11 15:21:24.000000000 +0200
+++ b/drivers/mtd/chips/cfi_probe.c    2011-05-11 15:22:14.000000000 +0200
@@ -213,6 +213,17 @@
     cfi->cfiq->InterfaceDesc = le16_to_cpu(cfi->cfiq->InterfaceDesc);
     cfi->cfiq->MaxBufWriteSize = le16_to_cpu(cfi->cfiq->MaxBufWriteSize);

+   /* Correct buffer size when Micron/J3-256Mb in 8-bit mode (1024 to 256
+      bytes) */
+   if ( cfi->mfr == CFI_MFR_NMX || cfi->mfr == CFI_MFR_ST ) {
+       if ((cfi->id & 0xff) == 0x1d) {
+           if ((cfi->cfiq->MaxBufWriteSize > 0x8) && (le16_to_cpu(cfi->cfiq->
+>P_ID) == 0x0002)) {
+               cfi->cfiq->MaxBufWriteSize = 0x8;
+               pr_warning("Adjusted buffer size on Micron Flash J3-256Mb");
+               pr_warning("in 8-bit mode (1024 to 256 bytes)\n");
+           }
+       }
+   }

#ifdef DEBUG_CFI
    /* Dump the information therein */
    print_cfi_ident(cfi->cfiq);
```

Resolving ERASE SUSPEND Hang Ups

Some revisions of the P3x suffer from ERASE SUSPEND hang ups. In particular, this can occur when the *ERASE CONFIRM* -> *SUSPEND* -> *PROGRAM* -> *RESUME* sequence causes a lockup due to internal timing issues. The consequence is that the ERASE cannot be resumed without inserting a dummy command after programming and prior to resuming. If the ERASE SUSPEND is not required, the user can enqueue a READ or PROGRAM operation that is required when the Flash device is erasing a block. This is done by applying the following code modification into the *chip_ready* function in the *cfi_cmdset_001.c* file:

```
case FL_ERASING:
- if (mode == FL_WRITING)
+ if ((mode == FL_WRITING) || (mode == FL_READY))
goto sleep;
```

This patch can be applied to all 2.6.x versions of the kernel. The erase suspend feature is not enabled when using a 2.4.x version of the kernel.

If the erase suspend feature is required, the workaround is to issue a dummy write cycle that writes an "FF" command code before the RESUME command.

The following code, which is applied to the *cfi_cmdset_0001.c* file, is a patch validated for kernel version 2.6.33:

```
--- a/drivers/mtd/chips/cfi_cmdset_0001.c 2011-05-13 12:10:22.000000000 +0200
+++ b/drivers/mtd/chips/cfi_cmdset_0001.c 2011-05-18 15:15:54.000000000 +0200
@@ -813,6 +813,17 @@
```

```
if (time_after(jiffies, timeo)) {
    /* Urgh. Resume and pretend we weren't here. */
+    /* before resume, insert a dummy 0xFF cycle for Micron P3x-65nm
       devices*/
+    if ((cfi->mfr == 0x0089) &&
+        ((cfi->id == 0x8919) || (cfi->id == 0x8960) ||
+         (cfi->id == 0x8962) || (cfi->id == 0x891c) ||
+         (cfi->id == 0x8961) || (cfi->id == 0x8963) ||
+         (cfi->id == 0x8999) || (cfi->id == 0x899a) ||
+         (cfi->id == 0x891f) || (cfi->id == 0x8964) ||
+         (cfi->id == 0x8966) || (cfi->id == 0x8922) ||
+         (cfi->id == 0x8965) || (cfi->id == 0x8967) ||
+         (cfi->id == 0x899e) || (cfi->id == 0x899f)) )
+        map_write(map, CMD(0xFF), adr);
    map_write(map, CMD(0xd0), adr);
    /* Make sure we're in 'read status' mode if it had finished */
    map_write(map, CMD(0x70), adr);
```

```
@@ -1007,6 +1018,17 @@
    sending the 0x70 (Read Status) command to an erasing
    chip and expecting it to be ignored, that's what we
    do. */
+
+   /* before resume, insert a dummy 0xFF cycle for Micron P3x-65nm
+      devices*/
+
+       if ((cfi->mfr == 0x0089) &&
+
+         ((cfi->id == 0x8919) || (cfi->id == 0x8960) ||
+
+         (cfi->id == 0x8962) || (cfi->id == 0x891c) ||
+
+         (cfi->id == 0x8961) || (cfi->id == 0x8963) ||
+
+         (cfi->id == 0x8999) || (cfi->id == 0x899a) ||
+
+         (cfi->id == 0x891f) || (cfi->id == 0x8964) ||
+
+         (cfi->id == 0x8966) || (cfi->id == 0x8922) ||
+
+         (cfi->id == 0x8965) || (cfi->id == 0x8967) ||
+
+         (cfi->id == 0x899e) || (cfi->id == 0x899f)) )
+
+         map_write(map, CMD(0xFF), adr);
    map_write(map, CMD(0xd0), adr);
    map_write(map, CMD(0x70), adr);
    chip->oldstate = FL_READY;
@@ -1168,6 +1190,17 @@
    local_irq_disable();

    /* Resume the write or erase operation */
+
+   /* before resume, insert a dummy 0xFF cycle for Micron P3x-65nm
+      devices*/
+
+       if ((cfi->mfr == 0x0089) &&
+
+         ((cfi->id == 0x8919) || (cfi->id == 0x8960) ||
+
+         (cfi->id == 0x8962) || (cfi->id == 0x891c) ||
+
+         (cfi->id == 0x8961) || (cfi->id == 0x8963) ||
+
+         (cfi->id == 0x8999) || (cfi->id == 0x899a) ||
+
+         (cfi->id == 0x891f) || (cfi->id == 0x8964) ||
+
+         (cfi->id == 0x8966) || (cfi->id == 0x8922) ||
+
+         (cfi->id == 0x8965) || (cfi->id == 0x8967) ||
+
+         (cfi->id == 0x899e) || (cfi->id == 0x899f)) )
+
+         map_write(map, CMD(0xFF), adr);
    map_write(map, CMD(0xd0), adr);
    map_write(map, CMD(0x70), adr);
    chip->state = oldstate;
```

ERASE SUSPEND Following ERASE RESUME

Some revisions of the P30, P33, and J3 Flash memory devices can hang when an ERASE SUSPEND command is issued following an ERASE RESUME without waiting for the minimum delay time to elapse. The result is that when the ERASE appears to be complete (no bits are toggling), the contents of the Flash memory block on which the ERASE was executing could be inconsistent with the expected values (typically, the array value is stuck to the 0xC0, 0xC4, 0x80, or 0x84 values). This causes the ERASE operation to fail.

When file system operations on the Flash are intensive, the incidences of this failure could be high. To preclude such failures, applying a patch is recommended. Intensive file system operations can cause excessive calls to the garbage routine to free Flash space (erasing physical Flash blocks may also help). When these failures occur, many consecutive SUSPEND and RESUME commands can follow.

This situation is resolved when an appropriate delay is inserted after the RESUME command by using the `udelay (...)` function available in Linux.

The applied delay value must be tuned based on the customer's platform. In all cases, the maximum delay value is 500µs. In most cases, a delay of 30µs to 50µs is sufficient.

If a customer suspects that inadequate delay may be causing failures, this approach is recommended:

1. Set the delay to the maximum value.
2. Check for reoccurrence.
3. If the failure does not occur again, try a lower delay value.

The following code defines the P30, P33, and J3 device patch for kernel 2.6.35. The changes are in the `put_chip` function.

```
*** a/drivers/mtd/chips/cfi_cmdset_0001.c 2011-02-24
11:13:21.000000000 +0100
--- b/drivers/mtd/chips/cfi_cmdset_0001.c 2011-02-24
11:15:21.000000000 +0100
*****
*** 1006,1011 ****
--- 1006,1015 ----
        do. */
        map_write(map, CMD(0xd0), adr);
        map_write(map, CMD(0x70), adr);
+
+     /* P3x Family Suspend-Resume-Suspend Issue */
+     udelay(DELAY);
+
        chip->oldstate = FL_READY;
        chip->state = FL_ERASING;
        break;
```



Reading the Extended Query Table in Linux 2.6.14 and Earlier

The P3x's extended query table is not read correctly in Linux versions 2.6.14 and earlier. The following patch is an example of code for correcting this for Linux 2.6.14.

Once the patch has been applied, during the phase of probing the P3x by the Linux operating system, the pointer `extra[0]` (the last field of the `extp` structure returned by the function `read_pri_intelext` in the `drivers/mtd/chips/cfi_cmdset_0001.c` file) will be updated to point to a vector containing all extended query table fields, starting with the address `(P+13)h`. For example, `extp->extra[11]` will contain the number of synchronous mode read configuration fields, which is address `(P+1E)h` of the extended query table.

```
diff -rupN a/drivers/mtd/chips/cfi_cmdset_0001.c b/drivers/mtd/chips/
cfi_cmdset_0001.c
--- a/drivers/mtd/chips/cfi_cmdset_0001.c 2011-10-03 15:01:44.000000000 +0200
+++ b/drivers/mtd/chips/cfi_cmdset_0001.c 2011-10-10 10:56:53.000000000 +0200
@@ -105,6 +105,7 @@ static struct mtd_chip_driver cfi_intele
static void cfi_tell_features(struct cfi_pri_intelext *extp)
{
int i;
+ printk(" Extended Query version %c.%c\n", extp->MajorVersion, extp->Minor-
Version);
printk(" Feature/Command Support: %4.4X\n", extp->FeatureSupport);
printk(" - Chip Erase: %s\n", extp->FeatureSupport&1?"supported":"unsup-
ported");
printk(" - Suspend Erase: %s\n", extp->FeatureSupport&2?"sup-
ported":"unsupported");
@@ -116,8 +117,9 @@ static void cfi_tell_features(struct cfi
printk(" - Page-mode read: %s\n", extp->FeatureSupport&128?"sup-
ported":"unsupported");
printk(" - Synchronous read: %s\n", extp->FeatureSupport&256?"sup-
ported":"unsupported");
printk(" - Simultaneous operations: %s\n", extp->FeatureSupport&512?"sup-
ported":"unsupported");
- for (i=10; i<32; i++) {
-     if (extp->FeatureSupport & (1<<i))
+ printk(" - Extended Flash Array: %s\n", extp->FeatureSupport&1024?"sup-
ported":"unsupported");
+ for (i=11; i<32; i++) {
+     if (extp->FeatureSupport & (1<<i))
printk(" - Unknown Bit %X: supported\n", i);
}
@@ -130,8 +132,14 @@ static void cfi_tell_features(struct cfi
```



```
printk(" Block Status Register Mask: %4.4X\n", extp->BlkStatusRegMask);
printk(" - Lock Bit Active: %s\n", extp->BlkStatusRegMask&1?"yes":"no");
- printk(" - Valid Bit Active: %s\n", extp->BlkStatusRegMask&2?"yes":"no");
- for (i=2; i<16; i++) {
+ printk(" - Lock-Down Bit Active: %s\n", extp->BlkStatusReg-
+ Mask&2?"yes":"no");
+ for (i=2; i<3; i++) {
+     if (extp->BlkStatusRegMask & (1<<i))
+         printk(" - Unknown Bit %X Active: yes\n",i);
+ }
+ printk(" - EFA Lock Bit: %s\n", extp->BlkStatusRegMask&16?"yes":"no");
+ printk(" - EFA Lock-Down Bit: %s\n", extp->BlkStatusReg-
+ Mask&32?"yes":"no");
+ for (i=6; i<16; i++) {
+     if (extp->BlkStatusRegMask & (1<<i))
+         printk(" - Unknown Bit %X Active: yes\n",i);
+ }
@@ -252,12 +260,21 @@ read_pri_intelxt(struct map_info *map,
    if (!extp)
        return NULL;

+ if (extp->MajorVersion != '1' ||
+     (extp->MinorVersion < '0' || extp->MinorVersion > '4')) {
+     printk(KERN_ERR " Unknown Intel/Sharp Extended Query "
+         "version %c.%c.\n", extp->MajorVersion,
+         extp->MinorVersion);
+     kfree(extp);
+     return NULL;
+ }

+
+ /* Do some byteswapping if necessary */
+ extp->FeatureSupport = le32_to_cpu(extp->FeatureSupport);
+ extp->BlkStatusRegMask = le16_to_cpu(extp->BlkStatusRegMask);
+ extp->ProtRegAddr = le16_to_cpu(extp->ProtRegAddr);

- if (extp->MajorVersion == '1' && extp->MinorVersion == '3') {
+ if (extp->MajorVersion == '1' && extp->MinorVersion >= '3') {
+     unsigned int extra_size = 0;
```

```
int nb_parts, i;

@@ -266,7 +283,10 @@ read_pri_intelext(struct map_info *map,
                                sizeof(struct cfi_intelext_otpinfo);

    /* Burst Read info */
-   extra_size += 6;
+   extra_size += 2;
+   if (extp_size < sizeof(*extp) + extra_size)
+       goto need_more;
+   extra_size += extp->extra[extra_size-1];

    /* Number of hardware-partitions */
    extra_size += 1;
@@ -274,6 +294,10 @@ read_pri_intelext(struct map_info *map,
                                goto need_more;
    nb_parts = extp->extra[extra_size - 1];

+   /* skip the sizeof(partregion) field in CFI 1.4 */
+   if (extp->MinorVersion >= '4')
+       extra_size += 2;
+
    for (i = 0; i < nb_parts; i++) {
        struct cfi_intelext_regioninfo *rinfo;
        rinfo = (struct cfi_intelext_regioninfo *)&extp->extra[extra_size];
@@ -285,6 +309,9 @@ read_pri_intelext(struct map_info *map,
                                * sizeof(struct cfi_intelext_blockinfo);
    }

+   if (extp->MinorVersion >= '4')
+       extra_size += sizeof(struct
+           cfi_intelext_programming_regioninfo);
+
    if (extp_size < sizeof(*extp) + extra_size) {
        need_more:
        extp_size = sizeof(*extp) + extra_size;
@@ -481,7 +508,7 @@ static int cfi_intelext_partition_fixup(
    * arrangement at this point. This can be rearranged in the future
```

```

    * if someone feels motivated enough.  --nico
    */
-   if (extp && extp->MajorVersion == '1' && extp->MinorVersion == '3'
+   if (extp && extp->MajorVersion == '1' && extp->MinorVersion >= '3'
        && extp->FeatureSupport & (1 << 9)) {
            struct cfi_private *newcfi;
            struct flchip *chip;
@@ -493,12 +520,16 @@ static int cfi_intelext_partition_fixup(
            sizeof(struct cfi_intelext_otpinfo);

            /* Burst Read info */
-           offs += 6;
+           offs += extp->extra[offs+1]+2;

            /* Number of partition regions */
            numregions = extp->extra[offs];
            offs += 1;

+           /* skip the sizeof(partregion) field in CFI 1.4 */
+           if (extp->MinorVersion >= '4')
+               offs += 2;
+
            /* Number of hardware partitions */
            numparts = 0;
            for (i = 0; i < numregions; i++) {
diff -rupN a/drivers/mtd/chips/cfi_util.c b/drivers/mtd/chips/cfi_util.c
--- a/drivers/mtd/chips/cfi_util.c 2011-10-03 15:01:44.000000000 +0200
+++ b/drivers/mtd/chips/cfi_util.c 2011-10-07 08:54:09.000000000 +0200
@@ -70,15 +70,6 @@ __xipram cfi_read_pri(struct map_info *m
    local_irq_enable();
#endif

-   if (extp->MajorVersion != '1' ||
-       (extp->MinorVersion < '0' || extp->MinorVersion > '3')) {
-       printk(KERN_WARNING " Unknown %s Extended Query "
-               "version %c.%c.\n", name, extp->MajorVersion,
-               extp->MinorVersion);

```

```
-         kfree(extp);  
-         extp = NULL;  
-     }  
-  
out:     return extp;  
}
```

```
diff -rupN a/include/linux/mtd/cfi.h b/include/linux/mtd/cfi.h  
--- a/include/linux/mtd/cfi.h 2011-10-03 15:02:20.000000000 +0200  
+++ b/include/linux/mtd/cfi.h 2011-10-03 11:54:45.000000000 +0200  
@@ -173,6 +173,15 @@ struct cfi_intelext_regioninfo {  
     struct cfi_intelext_blockinfo BlockTypes[1];  
} __attribute__((packed));  
  
+struct cfi_intelext_programming_regioninfo {  
+    uint8_t ProgRegShift;  
+    uint8_t Reserved1;  
+    uint8_t ControlValid;  
+    uint8_t Reserved2;  
+    uint8_t ControlInvalid;  
+    uint8_t Reserved3;  
+} __attribute__((packed));  
+  
/* Vendor-Specific PRI for AMD/Fujitsu Extended Command Set (0x0002) */  
  
struct cfi_pri_amdstd {
```



Summary of Available Patches

The available patches for using P30, P33, and J3 devices in a Linux environment are provided in Table 1.

Table 1: Available Patches

Objective	Kernel Affected	Patch Availability
Enable buffered programming	2.4.x	–
Enable 1KB buffered programming	Earlier than 2.6.13	2.6.12.6 2.4.30 2.4.25
Enable x8 mode	2.6.30 and earlier	2.6.33
ERASE SUSPEND hang ups	2.4.x and 2.6.x	2.6.33
Add delay after ERASE RESUME command	2.4.x and 2.6.x	2.6.35
Reading the extended query table	2.6.14 and earlier	2.6.14

Device Documentation

For additional information, refer to the component data sheets for the P30, P33, and J3 Flash memory devices available at <http://www.micron.com/products/nor-flash/parallel-nor-flash>.

Conclusion

Micron recognizes the value of open source compatibility and the importance of supporting the open source community. Micron regularly contributes patches and updates for the Linux MTD and Linux file systems to help ensure open source compatibility with Micron Flash memory devices.

To request specific Linux software or compatibility support with Micron Flash memory devices, contact your Micron representative or submit a request form at www.micron.com.

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

Micron and the Micron logo are trademarks of Micron Technology, Inc. All other trademarks are the property of their respective owners.



Revision History

Rev. C01/12
• Updated hyperlinks	
Rev. B10/11
• Added the “Reading the Extended Query Table in Linux 2.6.14 and Earlier” section	
• Updated the “Available Patches” table	
Rev. A05/11
• Initial release	