Freescale Semiconductor
Application Note

Document Number: AN3672
Rev 2, 01/2009

ST Micro and Micron 2 Kbytes/Page NAND Flash Connection to i.MX27 (MCIMX27) and i.MX31 (MCIMX31)

by: Florent Auger

The i.MX27 and i.MX31 NAND Flash controllers have the capability to support NAND Flash in both 512 bytes/page and 2 Kbytes per page. The Samsung NAND Flashes in 512 bytes/page are usually the preferred proven solution. Due to the increasing size of the flash density, most NAND Flashes are now mainly available in 2 Kbytes/page.

The aim of this application note is to show the compatibility of NAND Flash controllers from two manufacturers other than Samsung (ST Microelectronics and Micron) with the NAND Flash controller used in i.MX27 and i.MX31.

To that end, a Linux RedBoot was slightly modified to boot from the tested NAND Flashes, and that process and the results are described in this document.

Contents

1 Which NAND Flashes Were Tested?

The first NAND Flash tested was from ST Microelectronics. This is a common 8-bit bus size NAND Flash based on 2 Kbytes/page with a density of 1 Gbyte. The part number is NAND01GW3B2AN6, which is now obsolete and replaced by the equivalent NAND01GW3B2BN6.

The second NAND Flash tested was from Micron. This is a common 8-bit bus size NAND Flash based on 2 Kbytes/page with a density of 2 Gbyte. The part number is MT29F2G08A.

Table 1 provides details of the NAND Flashes tested.

Table 1. NAND Flashes Tested

Manufacturer / Part Number	Density	Bus Size	Page Size
ST Microelectronics/ NAND01GW3B2AN6	1 Gbyte	8-bit	2 Kbytes + 64 bytes spare
Micron/ MT29F2G08A	2 Gbytes	8-bit	2 Kbytes + 64 bytes spare

2 Hardware Setup:

You will need the following:

- Two reworked NAND Flash daughterboards compliant with i.MX31 ADS (only one connector)
- One i.MX31 ADS based on i.MX31 T02
- One i.MX27 EVB based on i.MX27 T02, which uses the same connector as the i.MX31 ADS for the NAND Flash daughterboard (the i.MX27 ADS uses a new daughterboard with two connectors)

3 Software Setup:

3.1 RedBoot:

For a default RedBoot package, only a few modifications are necessary to make the new NAND Flashes recognizable by RedBoot. The default NAND Flash driver can handle all NAND Flashes, because most of them follow the same quasi-standard commands and programming mechanism, unlike NOR Flashes which do not.

Refer to the manual to install the sources and rebuild RedBoot. This test was based on version 200749.

Because of limitations of the NAND Flash controller, the bad block indicator (BI) of the 2 Kbytes/page NAND should be handled correctly by the NAND Flash driver. By default, i.MX31 ADS RedBoot handles it fine. The 200749 version of the i.MX27 RedBoot for the ADS has to be slightly modified to handle the 2 Kbytes/page and the BI.

For both processors, the modifications related to the NAND details (page size, block size, number of blocks, identification) are made in this file:

./src/ecos/packages/devs/flash/arm/mxc/current/include/mxc_nand_parts.inl

Here is the added code:

```
    {
        device_id  : 0xf120, // ST Micro NAND01GW3B2AN6 (2 Kbytes/page 8-bit SLC
Nand)
        device_id2 : 0xFFFF,
        device_id3 : 0xFFFF,
        device_id4 : 0xFFFF,
        page_size  : 512*4,
        spare_size : 16*4,
        pages_per_block : 64,
        block_size : 64*2*1024,
        block_count: 1024,
        device_size: 128*1024*1024, // 128MB device =0x08000000
        port_size  : MXC_NAND_8_BIT,
        base_mask  : ~(0x08000000 - 1),
        type       : NAND_SLC,
        vendor_info: "ST Micro NAND01GW3B2AN6 8-bit 2K page 128MB SLC NAND",
    },
    {
        device_id  : 0xda2c, // Micron MT29F2G08AAC (2 Kbytes/page 8-bit SLC Nand)
        device_id2 : 0xFFFF,
        device_id3 : 0xFFFF,
        device_id4 : 0xFFFF,
        page_size  : 512*4,
        spare_size : 16*4,
        pages_per_block : 64,
        block_size : 64*2*1024,
        block_count: 2048,
        device_size: 256*1024*1024, // 256 MB device =0x10000000
        port_size  : MXC_NAND_8_BIT,
        base_mask  : ~(0x10000000 - 1),
        type       : NAND_SLC,
        vendor_info: "Micron MT29F2G08AAC 8-bit 2K page 256MB SLC NAND",
    },
```

For the i.MX27 RedBoot, only the following file had to be modified:

./src/ecos/packages/hal/arm/mx27/ads/current/include/hal_platform_setup.h

The 2 Kbytes/page NAND was not supported by the first-stage, low-level NAND driver in this RedBoot version. Everything related to the NAND Flash low-level driver should be modified as for the hal_platform_setup.h of the i.MX27 3DS.

The patch is available at the end of this document.

After the modifications are made, both RedBoots can be built and are ready to be flashed in the NAND.

3.2 Advanced Toolkit

To program these NAND Flashes, the Advanced Toolkit (ATK version 1.41 or later) is used, as it handles the bad block swap programming method.

The default ATK NAND Flash driver supports the 2kB page size, but has to be modified with the specifications of these new NAND Flashes: page size, block size, number of blocks, identification.

Refer to the manual to install the sources and rebuild the NAND Flash library of the Advanced ToolKit.

The only modification was made in the NAND Flash library for both i.MX31 and i.MX27. The following files are to be modified:

```
./device_program\flash\nand_flash\mx27_2kpage\src\mx_nand2k.c
```

```
./device_program\flash\nand_flash\mx31_2kpage\src\mx_nand2k.c
```

The nand_type[] table must be updated with the following:

```
{ 0x2C, 0xDA, 8, 0, 3, 2048, 64, "NAND MT29F2G08A" }, // Micron MT29F2G08A
```

```
{ 0x20, 0xF1, 8, 0, 3, 1024, 64, "NAND NAND01GW3B2AN6" }  // ST Micro NAND01GW3B2AN6
```

After both libraries are built and installed in the image directory of the Advanced ToolKit, the NAND Flash can be programmed by enabling the BI swap and selecting the flash model K9K2G08R0A. This calls the driver that has just been modified but that now recognizes NAND Flashes other than the Samsung ones.

First immediate result: This step simply shows that the driver can access the NAND Flash to read its ID, and that the driver successfully programmed the flash.

4 Results

With the modification of the Advanced ToolKit, the NAND Flashes can be read and programmed. The next step is to verify that the programming is correct and, most important, that the NAND Flashes can be used to boot the system.

Simply by connecting a serial cable, as it is usually done when using RedBoot, and then setting the boot mode to 'NAND 2 Kbytes/page 8-bit', it is easy to check whether the RedBoot displays its console on the host terminal.

Boot settings:

- i.MX31: BOOT0=ON, BOOT1=ON, BOOT2=ON, BOOT3=ON, BOOT4=OFF
- i.MX27: BOOT0=ON, BOOT1=OFF, BOOT2=ON, BOOT3=ON

Final result: RedBoot displays its console on the host terminal, which validates the programming as well as the boot capability for the ST Micro and Micron tested NAND Flashes.

NOTE

It is likely that if one NAND Flash model from a manufacturer can be used with the i.MX NAND Flash controller, all other models of this manufacturer will work as well. However, there are exceptions; for example, some NAND Flashes from Micron require a Reset command before any other access, which makes them not directly bootable by the i.MX27/31 NAND Flash controller.

NAND Flashes use the same basic set of commands and work in a similar way. Thus, users are offered features that are quasi-standard, making it easy to migrate from one manufacturer to an another without having to rewrite the whole drivers.

5 Patch for i.MX27 hal_platform_setup.h:

```
--- packages/hal/arm/mx27/ads/current/include/hal_platform_setup.h      2008-02-06
10:52:33.000000000 +0100
+++ packages/hal/arm/mx27/ads/current/include/hal_platform_setup.h.new  2008-02-06
11:54:20.000000000 +0100
@@ -59,6 +59,8 @@

 #define CYGHWR_HAL_ROM_VADDR        0x0

+#define NFC_2K_BI_SWAP
+
 // This macro represents the initial startup code for the platform
 // r11 is reserved to contain chip rev info in this file
     .macro  _platform_setup1
@@ -69,6 +71,23 @@
     mcr 15, 0, r0, c8, c7, 0    /* invalidate TLBs */
     mcr 15, 0, r0, c7, c10, 4   /* Drain the write buffer */
```

```
+    /* Reload data from spare area to 0x400 of main area if booting from NAND */
+    mov r0, #NFC_BASE
+    add r1, r0, #0x400
+    cmp pc, r0
+    blo init_aipi_start
+    cmp pc, r1
+    bhi init_aipi_start
+#ifdef NFC_2K_BI_SWAP
+    ldr r3, [r0, #0x7D0]    // load word at addr 464 of last 512 RAM buffer
+    and r3, r3, #0xFFFFFF00 // mask off the LSB
+    ldr r4, [r0, #0x834]    // load word at addr 4 of the 3rd spare area buffer
+    mov r4, r4, lsr #8      // shift it to get the byte at addr 5
+    and r4, r4, #0xFF       // throw away upper 3 bytes
+    add r3, r4, r3          // construct the word
+    str r3, [r0, #0x7D0]    // write back
+#endif
+
 init_aipi_start:
     init_aipi

@@ -76,12 +95,25 @@
     ldr r1, AVIC_VECTOR0_ADDR_W
     str r0, [r1] // for checking boot source from nand, nor or sdram

+/* __FLO__ */
+/* It overwrites the FMS bit that is used later to know what is the size of the
NAND pages */
+#ifdef __FLO__
     // setup System Controls
     ldr r0, SOC_SYSCTRL_BASE_W
     mov r1, #0x03
     str r1, [r0, #(SOC_SYSCTRL_PCSR - SOC_SYSCTRL_BASE)]
     mov r1, #0xFFFFFFC9
     str r1, [r0, #(SOC_SYSCTRL_FMCR - SOC_SYSCTRL_BASE)]
+#endif
+/* Use instead the following from 3DS HAL code */
+    // setup System Controls
+    ldr r0, SOC_SYSCTRL_BASE_W
+    mov r1, #0x03
+    str r1, [r0, #(SOC_SYSCTRL_PCSR - SOC_SYSCTRL_BASE)]
+    ldr r1, [r0, #(SOC_SYSCTRL_FMCR - SOC_SYSCTRL_BASE)]
+    and r1, r1, #0xFFFFFFF0
+    orr r1, r1, #9
+    str r1, [r0, #(SOC_SYSCTRL_FMCR - SOC_SYSCTRL_BASE)]

 init_max_start:
     init_max
@@ -117,6 +149,7 @@
     blo Normal_Boot_Continue
     cmp pc, r2
     bhi Normal_Boot_Continue
+
 NAND_Boot_Start:
     /* Copy image from flash to SDRAM first */
     ldr r1, MXC_REDBOOT_ROM_START
@@ -137,33 +170,44 @@
     nop
     nop

+/* __FLO__ */
+/* For the NAND management, it uses instead the folling from 3DS HAL code */
+NAND_Copy_Main:
```

```
+    // Check if x16/2kb page
+    ldr r7, SOC_SYSCTRL_BASE_W
+    ldr r7, [r7, #0x14]
+    ands r7, r7, #(1 << 5)
+
     mov r0, #NAND_FLASH_BOOT
     ldr r1, AVIC_VECTOR0_ADDR_W
     str r0, [r1]
     mov r0, #MXCFIS_NAND
     ldr r1, AVIC_VECTOR1_ADDR_W
     str r0, [r1]
-NAND_Copy_Main:
+
     ldr r0, NFC_BASE_W   //r0: nfc base. Reloaded after each page copying
     mov r1, #0x800       //r1: starting flash addr to be copied. Updated constantly
-    add r2, r0, #0x200   //r2: end of 1st RAM buf. Doesn't change
+    add r2, r0, #0x800     //2K Page:: r2: end of 1st RAM buf. Doesn't change
+    addeq r2, r0, #0x200   //512 Page:: r2: end of 1st RAM buf. Doesn't change
     add r12, r0, #0xE00  //r12: NFC register base. Doesn't change
-    ldr r14, MXC_REDBOOT_ROM_START
-    add r13, r14, #REDBOOT_IMAGE_SIZE //r13: end of SDRAM address for copying. Doesn't
change
-    add r14, r14, r1     //r14: starting SDRAM address for copying. Updated constantly
+    ldr r11, MXC_REDBOOT_ROM_START
+    add r13, r11, #REDBOOT_IMAGE_SIZE //r13: end of SDRAM address for copying. Doesn't
change
+    add r11, r11, r1     //r11: starting SDRAM address for copying. Updated constantly

     //unlock internal buffer
     mov r3, #0x2
     strh r3, [r12, #0xA]

 Nfc_Read_Page:
-//  writew(FLASH_Read_Mode1, NAND_FLASH_CMD_REG);
-    mov r3, #0x0;
-    strh r3, [r12, #NAND_FLASH_CMD_REG_OFF]
-    mov r3, #NAND_FLASH_CONFIG2_FCMD_EN;
-    strh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
-    do_wait_op_done
-
+//  NFC_CMD_INPUT(FLASH_Read_Mode1);
+    mov r3, #0x0
+    nfc_cmd_input
+
+    // Check if x16/2kb page
+    ldr r7, SOC_SYSCTRL_BASE_W
+    ldr r7, [r7, #0x14]
+    ands r7, r7, #(1 << 5)
+    bne nfc_addr_ops_2kb
 //    start_nfc_addr_ops(ADDRESS_INPUT_READ_PAGE, addr, nflash_dev_info->base_mask);
     mov r3, r1
     do_addr_input       //1st addr cycle
@@ -171,28 +215,75 @@
     do_addr_input       //2nd addr cycle
     mov r3, r1, lsr #17
     do_addr_input       //3rd addr cycle
-
```

```
     mov r3, r1, lsr #25
     do_addr_input       //4th addr cycle
+    b end_of_nfc_addr_ops

+nfc_addr_ops_2kb:
+//    start_nfc_addr_ops(ADDRESS_INPUT_READ_PAGE, addr, nflash_dev_info->base_mask);
+    mov r3, #0
+    do_addr_input       //1st addr cycle
+    mov r3, #0
+    do_addr_input       //2nd addr cycle
+    mov r3, r1, lsr #11
+    do_addr_input       //3rd addr cycle
+    mov r3, r1, lsr #19
+    do_addr_input       //4th addr cycle
+    mov r3, r1, lsr #27
+    do_addr_input       //4th addr cycle

-//    NFC_DATA_OUTPUT(buf, FDO_PAGE_SPARE_VAL);
-//        writew(NAND_FLASH_CONFIG1_ECC_EN, NAND_FLASH_CONFIG1_REG);
-    mov r3, #(NAND_FLASH_CONFIG1_ECC_EN)
-    strh r3, [r12, #NAND_FLASH_CONFIG1_REG_OFF]
+//    NFC_CMD_INPUT(FLASH_Read_Mode1_2K);
+    mov r3, #0x30
+    nfc_cmd_input

-//        writew(buf_no, RAM_BUFFER_ADDRESS_REG);
-    mov r3, #0
-    strh r3, [r12, #RAM_BUFFER_ADDRESS_REG_OFF]
-//        writew(FDO_PAGE_SPARE_VAL & 0xFF, NAND_FLASH_CONFIG2_REG);
-    mov r3, #FDO_PAGE_SPARE_VAL
-    strh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
-//        wait_op_done();
-    do_wait_op_done
+end_of_nfc_addr_ops:
+//    NFC_DATA_OUTPUT(buf, FDO_PAGE_SPARE_VAL);
+//        writew(NAND_FLASH_CONFIG1_INT_MSK | NAND_FLASH_CONFIG1_ECC_EN,
+//               NAND_FLASH_CONFIG1_REG);
+    mov r8, #0
+    bl nfc_data_output
+    bl do_wait_op_done
+    // Check if x16/2kb page
+    ldr r7, SOC_SYSCTRL_BASE_W
+    ldr r7, [r7, #0x14]
+    ands r7, r7, #(1 << 5)
+    beq nfc_addr_data_output_done_512
+
+// For 2K page - 2nd 512
+    mov r8, #1
+    bl nfc_data_output
+    bl do_wait_op_done
+
+// 3rd 512
+    mov r8, #2
+    bl nfc_data_output
+    bl do_wait_op_done
+
+// 4th 512
```

```
+    mov r8, #3
+    bl nfc_data_output
+    bl do_wait_op_done
+// end of 4th
+#ifdef NFC_2K_BI_SWAP
+    ldr r3, [r0, #0x7D0]    // load word at addr 464 of last 512 RAM buffer
+    and r3, r3, #0xFFFFFF00 // mask off the LSB
+    ldr r4, [r0, #0x834]    // load word at addr 4 of the 3rd spare area buffer
+    mov r4, r4, lsr #8      // shift it to get the byte at addr 5
+    and r4, r4, #0xFF       // throw away upper 3 bytes
+    add r3, r4, r3          // construct the word
+    str r3, [r0, #0x7D0]    // write back
+#endif
+    // check for bad block
+    mov r3, r1, lsl #(32-17)    // get rid of block number
+    cmp r3, #(0x800 << (32-17)) // check if not page 0 or 1
+    b nfc_addr_data_output_done

+nfc_addr_data_output_done_512:
     // check for bad block
-    mov r3, r1, lsl #(32-5-9)
-    cmp r3, #(512 << (32-5-9))
+    mov r3, r1, lsl #(32-5-9)    // get rid of block number
+    cmp r3, #(512 << (32-5-9))   // check if not page 0 or 1
+
+nfc_addr_data_output_done:
     bhi Copy_Good_Blk
     add r4, r0, #0x800  //r3 -> spare area buf 0
     ldrh r4, [r4, #0x4]
@@ -203,21 +294,45 @@
     cmp r3, #0x0
     beq Skip_bad_block
     // even suckier since we already read the first page!
-    sub r14, r14, #512  //rewind 1 page for the sdram pointer
-    sub r1, r1, #512    //rewind 1 page for the flash pointer
+
+    // Check if x16/2kb page
+    ldr r7, SOC_SYSCTRL_BASE_W
+    ldr r7, [r7, #0x14]
+    ands r7, r7, #(1 << 5)
+
+    subeq r11, r11, #512  //rewind 1 page for the sdram pointer
+    subeq r1, r1, #512    //rewind 1 page for the flash pointer
+
+    // for 2k page
+    subne r11, r11, #0x800  //rewind 1 page for the sdram pointer
+    subne r1, r1, #0x800    //rewind 1 page for the flash pointer
+
 Skip_bad_block:
-    add r1, r1, #(32*512)
+    // Check if x16/2kb page
+    ldr r7, SOC_SYSCTRL_BASE_W
+    ldr r7, [r7, #0x14]
+    ands r7, r7, #(1 << 5)
+
+    addeq r1, r1, #(32*512)
+    addne r1, r1, #(64*2048)
```

```
+
     b Nfc_Read_Page
 Copy_Good_Blk:
     //copying page
 1:  ldmia r0!, {r3-r10}
-    stmia r14!, {r3-r10}
+    stmia r11!, {r3-r10}
     cmp r0, r2
     blo 1b
-    cmp r14, r13
+    cmp r11, r13
     bge NAND_Copy_Main_done
-    add r1, r1, #0x200
-    ldr r0, NFC_BASE_W
+    // Check if x16/2kb page
+    ldr r7, SOC_SYSCTRL_BASE_W
+    ldr r7, [r7, #0x14]
+    ands r7, r7, #(1 << 5)
+
+    addeq r1, r1, #0x200
+    addne r1, r1, #0x800
+    mov r0, #NFC_BASE
     b Nfc_Read_Page

 NAND_Copy_Main_done:
@@ -303,6 +418,24 @@

     .endm                       // _platform_setup1

+do_wait_op_done:
+    1:
+        ldrh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
+        ands r3, r3, #NAND_FLASH_CONFIG2_INT_DONE
+        beq 1b
+    bx lr     // do_wait_op_done
+
+nfc_data_output:
+    mov r3, #(NAND_FLASH_CONFIG1_INT_MSK | NAND_FLASH_CONFIG1_ECC_EN)
+    strh r3, [r12, #NAND_FLASH_CONFIG1_REG_OFF]
+
+    // writew(buf_no, RAM_BUFFER_ADDRESS_REG);
+    strh r8, [r12, #RAM_BUFFER_ADDRESS_REG_OFF]
+    // writew(FDO_PAGE_SPARE_VAL & 0xFF, NAND_FLASH_CONFIG2_REG);
+    mov r3, #FDO_PAGE_SPARE_VAL
+    strh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
+    bx lr
+
 #else // defined(CYG_HAL_STARTUP_ROM) || defined(CYG_HAL_STARTUP_ROMRAM)
 #define PLATFORM_SETUP1
 #endif
@@ -519,9 +652,18 @@
         ldr r1, [r1]
         ands r1, r1, #0xF0000000
         // add Latency on CAS only for TO2
-        ldreq r1, SDRAM_0x00795729
-        ldrne r1, SDRAM_0x00795429
+-       ldreq r1, SDRAM_0x00795729
+-       ldrne r1, SDRAM_0x00795429
-
```

```
+       // TO 1.0's ID = 0x0 ==>> CAS = 3
+       bne 2f
+        ldr r1, SDRAM_0x00795729
+       b 3f
+       // now handles TO 2.x
+    2:
+       ands r1, r1, #0xE0000000
+       // TO 2.0's ID = 0x1 => CAS = 4 due to the MPEG4 issue
+       ldreq r1, SDRAM_0x00795429
+        // subesquent TO's are OK w/ CAS = 3
+        ldrne r1, SDRAM_0x00795729
+    3:
         str r1, [r0, #0x4]
         ldr r1, SDRAM_0x92200000
         str r1, [r0, #0x0]
@@ -539,21 +681,19 @@
         str r1, [r0, #0x0]
     .endm   // setup_sdram_ddr

-    .macro do_wait_op_done
-    1:
-        ldrh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
-        ands r3, r3, #NAND_FLASH_CONFIG2_INT_DONE
-        beq 1b
-        mov r3, #0x0
+    .macro nfc_cmd_input
+        strh r3, [r12, #NAND_FLASH_CMD_REG_OFF]
+        mov r3, #NAND_FLASH_CONFIG2_FCMD_EN;
         strh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
-    .endm   // do_wait_op_done
+        bl do_wait_op_done
+    .endm   // nfc_cmd_input

     .macro do_addr_input
         and r3, r3, #0xFF
         strh r3, [r12, #NAND_FLASH_ADD_REG_OFF]
         mov r3, #NAND_FLASH_CONFIG2_FADD_EN
         strh r3, [r12, #NAND_FLASH_CONFIG2_REG_OFF]
-        do_wait_op_done
+        bl do_wait_op_done
     .endm   // do_addr_input

 #define PLATFORM_VECTORS         _platform_vectors
```

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3672
Rev. 2, 01/2009