# Mask Set Errata for Mask 1N03P

This report applies to mask 1N03P for these products:

- MK80FN256Vxx15
- MK82FN256Vxx15
- MK81FN256Vxx15

**Table 1. Errata and Information Summary**

| Erratum ID | Erratum Title |
|---|---|
| e8992 | AWIC: Early NMI wakeup not detected upon entry to stop mode from VLPR mode |
| e6939 | Core: Interrupted loads to SP can cause erroneous behavior |
| e9005 | Core: Store immediate overlapping exception return operation might vector to incorrect interrupt |
| e6940 | Core: VDIV or VSQRT instructions might not complete correctly when very short ISRs are used |
| e50117 | FAC: Execute-only access control feature has been deprecated |
| e9265 | FTM: Incorrect match may be generated if intermediate load feature is used in toggle mode |
| e9457 | Kinetis Flashloader/ ROM Bootloader: The peripheral auto-detect code in bootloader can falsely detect presence of SPI host causing non-responsive bootloader |
| e9274 | LTC: Data Size Register does not handle concurrent update requests for CCM or GCM |
| e9407 | LTC: Writing individual bytes of PKHA RAM will cause adjacent bytes within the same 32-bit word to be corrupted. |
| e7735 | MCG: IREFST status bit may set before the IREFS multiplexor switches the FLL reference clock |
| e9650 | QuadSPI: Not all QuadSPI implementations supported |
| e9651 | QuadSPI: QuadSPI SDR clock limitation when core clock is greater than 100MHz |
| e9461 | QuadSPI: Read data errors may occur with data learning in 4x sampling method |
| e9626 | ROM Bootloader: Aliased QuadSPI address space is not supported by the Kinetis Bootloader command APIs |
| e9627 | ROM Bootloader: Cannot boot into QuadSPI DDR mode |
| e3981 | SDHC: ADMA fails when data length in the last descriptor is less or equal to 4 bytes |
| e3982 | SDHC: ADMA transfer error when the block size is not a multiple of four |
| e4624 | SDHC: AutoCMD12 and R1b polling problem |
| e3977 | SDHC: Does not support Infinite Block Transfer Mode |

*Table continues on the next page...*

Table 1. Errata and Information Summary (continued)

| Erratum ID | Erratum Title |
|---|---|
| e4627 | SDHC: Erroneous CMD CRC error and CMD Index error may occur on sending new CMD during data transfer |
| e3984 | SDHC: eSDHC misses SDIO interrupt when CINT is disabled |
| e3983 | SDHC: Problem when ADMA2 last descriptor is LINK or NOP |
| e3978 | SDHC: Software can not clear DMA interrupt status bit after read operation |
| e9625 | System: Leakage is possible on some PORTE pins when VDD is greater than VDDIO_E |
| e8807 | USB: In Host mode, transmission errors may occur when communicating with a Low Speed (LS) device through a USB hub |

Table 2. Revision History

| Revision | Changes |
|---|---|
| 09 Sept 2015 | Initial revision |
| 09 JUL 2019 | The following errata were added.<br>• e50117<br>The following errata were revised.<br>• e6940<br>• e9005<br>• e6939<br>• e9650 |

## e8992: AWIC: Early NMI wakeup not detected upon entry to stop mode from VLPR mode

**Description:** Upon entry into VLPS from VLPR, if NMI is asserted before the VLPS entry completes, then the NMI does not generate a wakeup to the MCU. However, the NMI interrupt will occur after the MCU wakes up by another wake-up event.

**Workaround:** There are two workarounds:

1) First transition from VLPR mode to RUN mode, and then enter into VLPS mode from RUN mode.

2) Assert NMI signal for longer than 16 bus clock cycles.

## e6939: Core: Interrupted loads to SP can cause erroneous behavior

**Description:** Arm Errata 752770: Interrupted loads to SP can cause erroneous behavior

This issue is more prevalent for user code written to manipulate the stack. Most compilers will not be affected by this, but please confirm this with your compiler vendor. MQX™ and FreeRTOS™ are not affected by this issue.

Affects: Cortex-M4, Cortex-M4F

Fault Type: Programmer Category B

Fault Status: Present in: r0p0, r0p1 Open.

If an interrupt occurs during the data-phase of a single word load to the stack-pointer (SP/R13), erroneous behavior can occur. In all cases, returning from the interrupt will result in the load instruction being executed an additional time. For all instructions performing an update to the base register, the base register will be erroneously updated on each execution, resulting in the stack-pointer being loaded from an incorrect memory location.

The affected instructions that can result in the load transaction being repeated are:

1) LDR SP,[Rn],#imm

2) LDR SP,[Rn,#imm]!

3) LDR SP,[Rn,#imm]

4) LDR SP,[Rn]

5) LDR SP,[Rn,Rm]

The affected instructions that can result in the stack-pointer being loaded from an incorrect memory address are:

1) LDR SP,[Rn],#imm

2) LDR SP,[Rn,#imm]!

Conditions:

1) An LDR is executed, with SP/R13 as the destination.

2) The address for the LDR is successfully issued to the memory system.

3) An interrupt is taken before the data has been returned and written to the stack-pointer.

Implications:

Unless the load is being performed to Device or Strongly-Ordered memory, there should be no implications from the repetition of the load. In the unlikely event that the load is being performed to Device or Strongly-Ordered memory, the repeated read can result in the final stack-pointer value being different than had only a single load been performed.

Interruption of the two write-back forms of the instruction can result in both the base register value and final stack-pointer value being incorrect. This can result in apparent stack corruption and subsequent unintended modification of memory.

**Workaround:** Most compilers are not affected by this, so a workaround is not required.

However, for hand-written assembly code to manipulate the stack, both issues may be worked around by replacing the direct load to the stack-pointer, with an intermediate load to a general-purpose register followed by a move to the stack-pointer.

If repeated reads are acceptable, then the base-update issue may be worked around by performing the stack pointer load without the base increment followed by a subsequent ADD or SUB instruction to perform the appropriate update to the base register.

## e9005: Core: Store immediate overlapping exception return operation might vector to incorrect interrupt

**Description:** Arm Errata 838869: Store immediate overlapping exception return operation might vector to incorrect interrupt

Affects: Cortex-M4, Cortex-M4F

Fault Type: Programmer Category B Rare

Fault Status: Present in: r0p0, r0p1 Open.

The Cortex-M4 includes a write buffer that permits execution to continue while a store is waiting on the bus. Under specific timing conditions, during an exception return while this buffer is still in use by a store instruction, a late change in selection of the next interrupt to be taken might result in there being a mismatch between the interrupt acknowledged by the interrupt controller and the vector fetched by the processor.

Configurations Affected

This erratum only affects systems where writeable memory locations can exhibit more than one wait state.

**Workaround:** For software not using the memory protection unit, this erratum can be worked around by setting DISDEFWBUF in the Auxiliary Control Register.

In all other cases, the erratum can be avoided by ensuring a DSB occurs between the store and the BX instruction. For exception handlers written in C, this can be achieved by inserting the appropriate set of intrinsics or inline assembly just before the end of the interrupt function, for example:

ARMCC:

```
...
__schedule_barrier();
__asm{DSB};
__schedule_barrier();
}
```

GCC:

```
...
__asm volatile (“dsb 0xf” ::: “memory”);
}
```

## e6940: Core: VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

**Description:** Arm Errata 709718: VDIV or VSQRT instructions might not complete correctly when very short ISRs are used

Affects: Cortex-M4F

Fault Type: Programmer Category B

Fault Status: Present in: r0p0, r0p1 Open.

On Cortex-M4 with FPU, the VDIV and VSQRT instructions take 14 cycles to execute. When an interrupt is taken a VDIV or VSQRT instruction is not terminated, and completes its execution while the interrupt stacking occurs. If lazy context save of floating point state is enabled then the automatic stacking of the floating point context does not occur until a floating point instruction is executed inside the interrupt service routine.

Lazy context save is enabled by default. When it is enabled, the minimum time for the first instruction in the interrupt service routine to start executing is 12 cycles. In certain timing conditions, and if there is only one or two instructions inside the interrupt service routine, then the VDIV or VSQRT instruction might not write its result to the register bank or to the FPSCR.

**Workaround:** A workaround is only required if the floating point unit is present and enabled. A workaround is not required if the memory system inserts one or more wait states to every stack transaction.

There are two workarounds:

1) Disable lazy context save of floating point state by clearing LSPEN to 0 (bit 30 of the FPCCR at address 0xE000EF34).

2) Ensure that every interrupt service routine contains more than 2 instructions in addition to the exception return instruction.

## e50117: FAC: Execute-only access control feature has been deprecated

**Description:** The FAC feature is no longer recommended for use.

**Workaround:** Do not program the XACCn registers to use the FAC feature.

## e9265: FTM: Incorrect match may be generated if intermediate load feature is used in toggle mode

**Description:** When a channel (n) match is used as an intermediate reload, an incorrect second match may occur immediately following the correct match. The issue is problematic only if channel (n) is configured for output compare with the output configured to toggle mode. In this scenario, channel (n) toggles on the correct match and again on the incorrect match. The issue may also occur if a certain channel has a match which is coincident with an intermediate reload point of any other channel.

**Workaround:** If any channel is configured for output compare mode with the output set for toggle mode, the intermediate reload feature must not be used.

## e9457: Kinetis Flashloader/ ROM Bootloader: The peripheral auto-detect code in bootloader can falsely detect presence of SPI host causing non-responsive bootloader

**Description:** During the active peripheral detection process, the bootloader can interpret spurious data on the SPI peripheral as valid data. The spurious data causes the bootloader to shutdown all peripherals except the “falsely detected" SPI and enter the command phase loop using the SPI. After the bootloader enters the command phase loop using the SPI, the other peripherals are ignored, so the desired peripheral is no longer active.

The bootloader will not falsely detect activity on the I2C, UART, or USB interfaces, so only the SPI interface is affected.

**Workaround:** Ensure that there is an external pull-up on the SPI chip-select pin or that the pin is driven high. This will prevent the bootloader from seeing spurious data due to activity on the SPI clock pin.

## e9274: LTC: Data Size Register does not handle concurrent update requests for CCM or GCM

**Description:** In CCM and GCM AES modes, it is possible that if the AESA tries to decrement the data size register, LTC0_DS, in the same cycle in which software writes the IV size, AAD size or data size registers, then the LTC will update the data size with that value rounded up to the next 16 bytes. This will cause the AESA to become out of sync and not recognize when the last block is being processed.

The CCM and GCM modes are the only ones that utilize the IV/AAD data type.

**Workaround:** There two possible workarounds for this issue.

1) Write all IV, AAD and MDATA sizes to LTC0_IVSZ, LTC0_AADSZ and LTC0_DS registers respectively before any data is written to LTC0_IFIFO.

2) After the IV size is written and IV is written to LTC0_IFIFO in GCM mode, then poll the LTC0_DS register until it reads 16 before the AAD or MDATA sizes are written. Likewise, after the AAD size is written and AAD written to LTC0_IFIFO, poll the LTC0_DS register until it reads 16 before the MDATA size is written to LTC0_DS register. The reason 16 should be waited for instead of 0 is that in these modes AESA will stall processing until it has at least 2 blocks (32 bytes to process), or until bit 31 is written in LTC0_IVSZ or LTC0_AADSZ registers indicating that only IV/AAD are being processed in this job. This enables special actions to be taken in case the next block to be processed is the last one.

## e9407: LTC: Writing individual bytes of PKHA RAM will cause adjacent bytes within the same 32-bit word to be corrupted.

**Description:** In LTC containing PKHA, the PKHA RAM is written from a 32-bit interface. Normally, each write consists of 4 bytes of data to be written. However, for writes of only 1-3 bytes, the non-written bytes within the same word will be overwritten with incorrect data.

**Workaround:** Always write all 32-bits of any word within PKHA RAM. If modifying an individual byte within a word of PKHA RAM is required, first read the full word, merge in the byte(s) to be written, then write back the entire new word.

## e7735: MCG: IREFST status bit may set before the IREFS multiplexor switches the FLL reference clock

**Description:** When transitioning from MCG clock modes FBE or FEE to either FBI or FEI, the MCG_S[IREFST] bit will set to 1 before the IREFS clock multiplexor has actually selected the slow IRC as the reference clock. The delay before the multiplexor actually switches is:

2 cycles of the slow IRC + 2 cycles of OSCERCLK

In the majority of cases this has no effect on the operation of the device.

**Workaround:** In the majority of applications no workaround is required. If there is a requirement to know when the IREFS clock multiplexor has actually switched, and OSCERCLK is no longer being used by the FLL, then wait the equivalent time of:

2 cycles of the slow IRC + 2 cycles of OSCERCLK

after MCG_S[IREFST] has been set to 1.

## e9650: QuadSPI: Not all QuadSPI implementations supported

**Description:** The following QuadSPI implementation is not supported:

- two separate QuadSPI/Dual Die flash in DDR mode with DQS.

**Workaround:** Use one of the following QuadSPI implementations which are supported:

- two separate QuadSPI/Dual Die flash in SDR or DDR mode without DQS

- Spansion HyperFlash™ NOR memory

- Octal Flash (SDR or DDR)

- Single Die Flash (SDR or DDR)

## e9651: QuadSPI: QuadSPI SDR clock limitation when core clock is greater than 100MHz

**Description:** The MCGPLL 2x clock cannot be used as the QuadSPI source clock (selected by QuadSPIx_SOCCR[QSPISRC]) when the MCGPLL clock is over 100MHz and QuadSPIx_SOCCR[SCLKCFG] is non-zero.

This means that when running the core clock at above 100MHz, the QuadSPI SDR clock cannot be 100MHz because it requires the use of the MCGPLL 2x clock to derive that QuadSPI clock speed.

**Workaround:** To run the QuadSPI SDR clock at 100MHz, the MCGPLL/core clock will also need to run at 100MHz.

If the core clock is run above 100MHz, the SDR clock can be generated by dividing the MCGPLL clock by 2, meaning that a 75MHz maximum SDR clock speed is possible with the core clock at 150MHz.

## e9461: QuadSPI: Read data errors may occur with data learning in 4x sampling method

**Description:** Data learning using 4x Sampling method may select a sampling point which is marginal. A marginal sampling point occurs when the sampling point is located on the edge of the valid sampling window. A marginal sampling point may return a positive comparison of the data learning pattern but small variations in voltage and temperature during the same read transaction may result in data errors, since the sampling point is not properly located inside the valid sampling window.

**Workaround:** There are two options:

- Internal DQS method allows to perform data learning as described on the Reference Manual.
- If 4x Sampling method is used, data learning should not be used and a fixed sampling point must be selected.

## e9626: ROM Bootloader: Aliased QuadSPI address space is not supported by the Kinetis Bootloader command APIs

**Description:** The Kinetis ROM Bootloader does not recognize QuadSPI alias space, starting at 0x0400_0000, as a valid address space. This means that API commands used to program data into this space will not succeed, and attempts to boot directly to an address in the alias space will return status error code ‘kStatus_OutOfRange'.

**Workaround:** To program the alias memory space, use the normal QuadSPI address space starting at address 0x6800_0000.

To begin executing code from the aliased space, the bootloader must first jump to the normal QuadSPI address space starting at 0x6800_0000, or to internal Flash address space, and then the code located in that address space should then jump to an address in the aliased space.

## e9627: ROM Bootloader: Cannot boot into QuadSPI DDR mode

**Description:** Certain fields required to configure QuadSPI for DDR mode are not able to be set by the ROM Bootloader. Thus a workaround is required for the application image for the ROM to boot into DDR mode.

**Workaround:** When writing an application image to QuadSPI, a piece of code must first be loaded and executed from RAM to configure QuadSPI DDR mode before using the ROM Bootloader to write the image to QuadSPI. When booting from QuadSPI, the QuadSPI configuration block must be located in internal Flash memory and the application must start executing from internal Flash in order to configure QuadSPI DDR mode before jumping to a QuadSPI address. The KBLQSPIUG has more information on this setup.

## e3981: SDHC: ADMA fails when data length in the last descriptor is less or equal to 4 bytes

**Description:** A possible data corruption or incorrect bus transactions on the internal AHB bus, causing possible system corruption or a stall, can occur under the combination of the following conditions:

1. ADMA2 or ADMA1 type descriptor

2. TRANS descriptor with END flag

3. Data length is less than or equal to 4 bytes (the length field of the corresponding descriptor is set to 1, 2, 3, or 4) and the ADMA transfers one 32-bit word on the bus

4. Block Count Enable mode

**Workaround:** The software should avoid setting ADMA type last descriptor (TRANS descriptor with END flag) to data length less than or equal to 4 bytes. In ADMA1 mode, if needed, a last NOP descriptor can be appended to the descriptors list. In ADMA2 mode this workaround is not feasible due to ERR003983.

## e3982: SDHC: ADMA transfer error when the block size is not a multiple of four

**Description:** Issue in eSDHC ADMA mode operation. The eSDHC read transfer is not completed when block size is not a multiple of 4 in transfer mode ADMA1 or ADMA2. The eSDHC DMA controller is stuck waiting for the IRQSTAT[TC] bit in the interrupt status register.

The following examples trigger this issue:

1. Working with an SD card while setting ADMA1 mode in the eSDHC
2. Performing partial block read
3. Writing one block of length 0x200
4. Reading two blocks of length 0x22 each. Reading from the address where the write operation is performed. Start address is 0x512 aligned. Watermark is set as one word during read. This read is performed using only one ADMA1 descriptor in which the total size of the transfer is programmed as 0x44 (2 blocks of 0x22).

**Workaround:** When the ADMA1 or ADMA2 mode is used and the block size is not a multiple of 4, the block size should be rounded to the next multiple of 4 bytes via software. In case of write, the software should add the corresponding number of bytes at each block end, before the write is initialized. In case of read, the software should remove the dummy bytes after the read is completed.

For example, if the original block length is 22 bytes, and there are several blocks to transfer, the software should set the block size to 24. The following data is written/stored in the external memory:

4 Bytes valid data

4 Bytes valid data

4 Bytes valid data

4 Bytes valid data

4 Bytes valid data

2 Bytes valid data + 2 Byte dummy data

4 Bytes valid data

4 Bytes valid data

4 Bytes valid data

4 Bytes valid data

4 Bytes valid data

2 Bytes valid data + 2 Byte dummy data

In this example, 48 (24 × 2) bytes are transferred instead of 44 bytes. The software should remove the dummy data.

## e4624: SDHC: AutoCMD12 and R1b polling problem

**Description:** Occurs when a pending command which issues busy is completed. For a command with R1b response, the proper software sequence is to poll the DLA for R1b commands to determine busy state completion. The DLA polling is not working properly for the ESDHC module and thus the DLA bit in PRSSTAT register cannot be polled to wait for busy state completion. This is relevant for all eSDHC ports (eSDHC1-4 ports).

**Workaround:** Poll bit 24 in PRSSTAT register (DLSL[0] bit) to check that wait busy state is over.

## e3977: SDHC: Does not support Infinite Block Transfer Mode

**Description:** The eSDHC does not support infinite data transfers, if the Block Count register is set to one, even when block count enable is not set.

**Workaround:** The following software workaround can be used instead of the infinite block mode:

1. Set BCEN bit to one and enable block count

2. Set the BLKCNT to the maximum value in Block Attributes Register (BLKATTR) (0xFFFFfor 65535 blocks)

## e4627: SDHC: Erroneous CMD CRC error and CMD Index error may occur on sending new CMD during data transfer

**Description:** When sending new, non data CMD during data transfer between the eSDHC and EMMC card, the module may return an erroneous CMD CRC error and CMD Index error. This occurs when the CMD response has arrived at the moment the FIFO clock is stopped. The following bits after the start bit of the response are wrongly interpreted as index, generating the CRC and Index errors.

The data transfer itself is not impacted.

The rate of occurrence of the issue is very small, as there is a need for the following combination of conditions to occur at the same cycle:

• The FIFO clock is stopped due to FIFO full or FIFO empty

• The CMD response start bit is received

**Workaround:** The recommendation is to not set FIFO watermark level to a too small value in order to reduce frequency of clock pauses.

The problem is identified by receiving the CMD CRC error and CMD Index error. Once this issue occurs, one can send the same CMD again until operation is successful.

## e3984: SDHC: eSDHC misses SDIO interrupt when CINT is disabled

**Description:** An issue is identified when interfacing the SDIO card. There is a case where an SDIO interrupt from the card is not recognized by the hardware, resulting in a hang.

If the SDIO card lowers the DAT1 line (which indicates an interrupt) when the SDIO interrupt is disabled in the eSDHC registers (that is, CINTEN bits in IRQSTATEN and IRQSIGEN are set to zero), then, after the SDIO interrupt is enabled (by setting the CINTEN bits in IRQSTATEN and IRQSIGEN registers), the eSDHC does not sense that the DAT1 line is low. Therefore, it fails to set the CINT interrupt in IRQSTAT even if DAT1 is low.

Generally, CINTEN bit is disabled in interrupt service.

The SDIO interrupt service steps are as follows:

1. Clear CINTEN bit in IRQSTATEN and IRQSIGEN.

2. Reset the interrupt factors in the SDIO card and write 1 to clear the CINT interrupt in IRQSTAT.

3. Re-enable CINTEN bit in IRQSTATEN and IRQSIGEN.

If a new SDIO interrupt from the card occurs between step 2 and step 3, the eSDHC skips it.

**Workaround:** The workaround interrupt service steps are as follows:

1. Clear CINTEN bit in IRQSTATEN and IRQSIGEN.

2. Reset the interrupt factors in the SDIO card and write 1 to clear CINT interrupt in IRQSTAT.

3. Clear and then set D3CD bit in the PROCTL register. Clearing D3CD bit sets the reverse signal of DAT1 to low, even if DAT1 is low. After D3CD bit is re-enabled, the eSDHC can catch the posedge of the reversed DAT1 signal, if the DAT1 line is still low.

4. Re-enable CINTEN bit in IRQSTATEN and IRQSIGEN.

## e3983: SDHC: Problem when ADMA2 last descriptor is LINK or NOP

**Description:** ADMA2 mode in the eSDHC is used for transfers to/from the SD card. There are three types of ADMA2 descriptors: TRANS, LINK or NOP. The eSDHC has a problem when the last descriptor (which has the End bit ‘1’) is a LINK descriptor or a NOP descriptor.

In this case, the eSDHC completes the transfers associated with this descriptor set, whereas it does not even start the transfers associated with the new data command. For example, if a WRITE transfer operation is performed on the card using ADMA2, and the last descriptor of the WRITE descriptor set is a LINK descriptor, then the WRITE is successfully finished. Now, if a READ transfer is programmed from the SD card using ADMA2, then this transfer does not go through.

**Workaround:** Software workaround is to always program TRANS descriptor as the last descriptor.

## e3978: SDHC: Software can not clear DMA interrupt status bit after read operation

**Description:** After DMA read operation, if the SDHC System Clock is automatically gated off, the DINT status can not be cleared by software.

**Workaround:** Set HCKEN bit before starting DMA read operation, to disable SDHC System Clock auto-gating feature; after the DINT and TC bit received when read operation is done, clear HCKEN bit to re-enable the SDHC System Clock auto-gating feature.

## e9625: System: Leakage is possible on some PORTE pins when VDD is greater than VDDIO_E

**Description:** There is current leakage observed from VDD to ground on a particular PORTE pin if all of the following conditions are met:

1) VDD is greater than VDDIO_E.

2) The pin is set as an input or as bidirectional (as in the case of a QuadSPI data pin).

3) The pin is being driven to a VDDIO_E level (logic high).

Only the following 8 pins are affected: PTE1, PTE2, PTE4, PTE6, PTE9, PTE10, PTE17, and PTE18.

**Workaround:** 1) Setting VDD equal to VDDIO_E avoids this issue.

2) If VDD is greater than VDDIO_E, then when possible, an affected pin should be pulled to ground either by an internal or external pull-down. This is particularly important before entering low power modes to avoid unnecessary leakage.

## e8807: USB: In Host mode, transmission errors may occur when communicating with a Low Speed (LS) device through a USB hub

**Description:** In Host mode, if the required 48 MHz USB clock is not derived from the same clock source used by the core, transmission errors may occur when communicating with a Low Speed (LS) device through a USB hub. A typical example that causes this issue is when an external 48 MHz clock is used for the USB module via the USB_CLKIN pin, and a separate external clock on XTAL/EXTAL is used to generate the system/core clock.

This issue does not occur when in USB Device mode or if the LS device is not connected through a USB hub.

**Workaround:** In Host mode, ensure the 48 MHz USB clock is derived from the same clock source that the system clock uses. The two clocks, while they do not need to be the same frequency, both need to come from the same source so that they are in sync. For example, generate the 48 MHz USB clock by dividing down the PLL clock used by the core/system via the SIM_CLKDIV2[USBFRAC] and SIM_CLKDIV2[USBDIV] bit fields.