Mask Set Errata for Mask 1N29D

This report applies to mask 1N29D for these products:

- MPC5645S

Errata ID	Errata Title
4168	ADC: Abort switch aborts the ongoing injected channel as well as the upcoming normal channel
6026	DSPI: Incorrect SPI Frame Generated in Combined Serial Interface Configuration
7026	FLASH: Continuously cycling from STANDBY to RUN mode may disturb flash array content.
6082	LINFlexD : LINS bits in LIN Status Register(LINSR) are not usable in UART mode.
4340	LINFlexD: Buffer overrun can not be detected in UART Rx FIFO mode
7274	LINFlexD: Consecutive headers received by LIN Slave triggers error interrupt
7394	MC_ME: Incorrect mode may be entered on low-power mode exit.
6976	MC_ME: SAFE mode not entered immediately on hardware-triggered SAFE mode request during STOP0 mode
5099	MC_ME: possibility of the machine check on low power mode exit (MPC5645S only)
7953	ME: All peripherals that will be disabled in the target mode must have their interrupt flags cleared prior to target mode entry
4114	MPC5645S RM does not correctly describe slew rate control bits
6726	NPC: MCKO clock may be gated one clock period early when MCKO frequency is programmed as SYS_CLK/8.and gating is enabled
6481	NZ4C3/NZ7C3: Erroneous Resource Full Message under certain conditions
7120	NZxC3: DQTAG implemented as variable length field in DQM message
9066	PAD RING: The Device may lock-up into reset loop while exiting from standby mode or at power up
3624	RLE: Decode operation freezes when configured for very small data transfers

e4168: ADC: Abort switch aborts the ongoing injected channel as well as the upcoming normal channel

Errata type: Errata

Description: If an Injected chain (jch1,jch2,jch3) is injected over a Normal chain (nch1,nch2,nch3,nch4) the Abort switch does not behave as expected.

Expected behavior:

Correct Case (without SW Abort on jch3): Nch1- Nch2(aborted) -Jch1 - Jch2 - Jch3 - Nch2(restored) - Nch3 - Nch4

Correct Case(with SW Abort on jch3): Nch1 - Nch2(aborted) -Jch1 - Jch2 - Jch3(aborted) - Nch2(restored) - Nch3 - Nch4

Observed unexpected behavior:

Fault1 (without SW abort on jch3): Nch1 - Nch2(aborted) - Jch1 - Jch2 - Jch3 - Nch3 - Nch4 (Nch2 not restored)

Fault2 (with SW abort on jch3): Nch1- Nch2 (aborted) - Jch1 - Jch2 - Jch3(aborted) - Nch4 (Nch2 not restored & Nch3 conversion skipped)

Workaround: It is possible to detect the unexpected behavior by using the CEOCFRx register. The CEOCFRx fields will not be set for a not restored or skipped channel, which indicates this issue has occurred. The CEOCFRx fields need to be checked before the next Normal chain execution (in scan mode). The CEOCFRx fields should be read by every ECH interrupt at the end of every chain execution.

e6026: DSPI: Incorrect SPI Frame Generated in Combined Serial Interface Configuration

Errata type: Errata

Description: In the Combined Serial Interface (CSI) configuration of the Deserial Serial Peripheral Interface (DSPI) where data frames are periodically being sent (Deserial Serial Interface, DSI), a Serial Peripheral Interface (SPI) frame may be transmitted with incorrect framing.

The incorrect frame may occur in this configuration if the user application writes SPI data to the DSPI Push TX FIFO Register (DSPI_PUSHR) during the last two peripheral clock cycles of the Delay-after-Transfer (DT) phase. In this case, the SPI frame is corrupted.

Workaround: Workaround 1: Perform SPI FIFO writes after halting the DSPI.

To prevent writing to the FIFO during the last two clock cycles of DT, perform the following steps every time a SPI frame is required to be transmitted:

Step 1: Halt the DSPI by setting the HALT control bit in the Module Configuration Register (DSPI_MCR[HALT]).

Step 2: Poll the Status Register's Transmit and Receive Status bit (DSPI_SR[TXRXS]) to ensure the DSPI has entered the HALT state and completed any in-progress transmission. Alternatively, if continuous polling is undesirable in the application, wait for a fixed time interval such as 35 baud clocks to ensure completion of any in-progress transmission and then check once for DSPI_SR[TXRXS].

Step 3: Perform the write to DSPI_PUSHR for the SPI frame.

Step 4: Clear bit DSPI_MCR[HALT] to bring the DSPI out of the HALT state and return to normal operation.

Workaround 2: Do not use the CSI configuration. Use the DSPI in either DSI-only mode or SPI-only mode.

Workaround 3: Use the DSPI's Transfer Complete Flag (TCF) interrupt to reduce worst-case wait time of Workaround 1.

Step 1: When a SPI frame is required to be sent, halt the DSPI as in Step 1 of Workaround 1 above.

Step 2: Enable the TCF interrupt by setting the DSPI DMA/Interrupt Request Select and Enable Register's Transmission Complete Request Enable bit (DSPI_RSER[TCF_RE])

Step 3: In the TCF interrupt service routine, clear the interrupt status (DSPI_SR[TCF]) and the interrupt request enable (DSPI_RSER[TCF_RE]). Confirm that DSPI is halted by checking DSPI_SR[TXRXS] and then write data to DSPI_PUSHR for the SPI frame. Finally, clear bit DSPI_MCR[HALT] to bring the DSPI out of the HALT state and return to normal operation.

e7026: FLASH: Continuously cycling from STANDBY to RUN mode may disturb flash array content.

Errata type: Errata

Description: Repetitively power cycling the flash with STANDBY mode may disturb flash bits, causing them to flip from a 1 (erased state) to a 0 (programmed state) when switching from STANDBY back to RUN mode when 80mV < VDD12 < 150mV. This voltage range is determined by characterization on a small sample size for typical devices.

For reference, a minimum 2.95M wakeups cycles from STANDBY to RUN with 80mV < VDD12 < 150mV are needed to disturb the flash contents.

Workaround: When exiting STANDBY mode, ensure that the VDD12 pins are below or above the defined range, from 80mV to 150mV, before entering a RUN mode.

Based on typical application conditions, Freescale recommends that VDD12 pins are below the lowest range value (80mV) with the following options:

1. Increase the time the MCU remains in STANDBY

2. Use a pull down resistor on VDD12

These options are recommendations or guidelines that each customer needs to analyze and adjust to their specific application.

Refer to Engineering Bulletin EB795, for a detailed analysis on the workaround options.

e6082: LINFlexD : LINS bits in LIN Status Register(LINSR) are not usable in UART mode.

Errata type: Errata

Description: When the LINFlexD module is used in the Universal Asynchronous Receiver/Transmitter (UART) mode, the LIN state bits (LINS3:0]) in LIN Status Register (LINSR) always indicate the value zero. Therefore, these bits cannot be used to monitor the UART state.

Workaround: LINS bits should be used only in LIN mode.

e4340: LINFlexD: Buffer overrun can not be detected in UART Rx FIFO mode

Errata type: Errata

Description: When the LINFlexD is configured in UART Receive (Rx) FIFO mode, the Buffer Overrun Flag (BOF) bit of the UART Mode Status Register (UARTSR) register is cleared in the subsequent clock cycle after being asserted.

User software can not poll the BOF to detect an overflow.

The LINFlexD Error Combined Interrupt can still be triggered by the buffer overrun. This interrupt is enabled by setting the Buffer Overrun Error Interrupt Enable (BOIE) bit in the LIN Interrupt enable register (LINIER). However, the BOF bit will be cleared when the interrupt routine is entered, preventing the user from identifying the source of error.

Workaround: Buffer overrun errors in UART FIFO mode can be detected by enabling only the Buffer Overrun Interrupt Enable (BOIE) in the LIN interrupt enable register (LINIER).

e7274: LINFlexD: Consecutive headers received by LIN Slave triggers error interrupt

Errata type: Errata

Description: As per the Local Interconnect Network (LIN) specification, the processing of one frame should be aborted by the detection of a new header sequence.

In LINFlexD, if the LIN Slave receives a new header instead of data response corresponding to a previous header received, it triggers a framing error during the new header's reception. The LIN Slave still waiting for the data response corresponding to the first header received.

Workaround: The following three steps should be followed -

1) Set the MODE bit in the LIN Time-Out Control Status Register (LINTCSR[MODE]) to '0'.

2) Set Idle on Timeout in the LINTCSR[IOT] register to '1'.

3) Configure master to wait until the occurrence of the Output Compare flag in LIN Error Status Register (LINESR[OCF]) before sending the next header. This flag causes the LIN Slave to go to an IDLE state before the next header arrives, which will be accepted without any framing error.

e7394: MC_ME: Incorrect mode may be entered on low-power mode exit.

Errata type: Errata

Description: For the case when the Mode Entry (MC_ME) module is transitioning from a run mode (RUN0/1/2/3) to a low power mode (HALT/STOP/STANDBY*) if a wake-up or interrupt is detected one clock cycle after the second write to the Mode Control (ME_MCTL) register, the MC_ME will exit to the mode previous to the run mode that initiated the low power mode transition.

Example correct operation DRUN->RUN1-> RUN3->STOP->RUN3

Example failing operation DRUN->RUN1-> RUN3->STOP->RUN1

*Note STANDBY mode is not available on all MPC56xx microcontrollers

Workaround: To ensure the application software returns to the run mode (RUN0/1/2/3) prior to the low power mode (HALT/STOP/STANDBY*) it is required that the RUNx mode prior to the low power mode is entered twice.

The following example code shows RUN3 mode entry prior to a low power mode transition.

```
ME.MCTL.R = 0x70005AF0; /* Enter RUN3 Mode & Key */

ME.MCTL.R = 0x7000A50F; /* Enter RUN3 Mode & Inverted Key */

while (ME.GS.B.S_MTRANS) {} /* Wait for RUN3 mode transition to complete */

ME.MCTL.R = 0x70005AF0; /* Enter RUN3 Mode & Key */
```

```
ME.MCTL.R = 0x7000A50F; /* Enter RUN3 Mode & Inverted Key */
while (ME.GS.B.S_MTRANS) {} /* Wait for RUN3 mode transition to complete */
/* Now that run mode has been entered twice can enter low power mode */
/* (HALT/STOP/STANDBY*) when desired. */
```

e6976: MC_ME: SAFE mode not entered immediately on hardware-triggered SAFE mode request during STOP0 mode

Errata type: Errata

Description: If a SAFE mode request is generated by the Reset Generation Module (MC_RGM) while the chip is in STOP0 mode, the chip does not immediately enter SAFE mode if STOP0 is configured as follows in the STOP0 Mode Configuration register (ME_STOP0_MC):

- the system clock is disabled (ME_STOP0_MC[SYSCLK] = 0b1111)

- the internal RC oscillator is enabled (ME_STOP0_MC[IRCON] = 0b1)

In this case, the chip will remain in STOP0 mode until an interrupt request or wakeup event occurs, causing the chip to return to its previous RUNx mode, after which the still pending SAFE mode request will cause the chip to enter SAFE mode.

Workaround: There are two possibilities.

1. Configure the internal RC oscillator to be disabled during STOP0 mode (ME_STOP0_MC[IRCON] = 0b0) if the device supports it.

2. Prior to entering STOP0 mode, configure all hardware-triggered SAFE mode requests that need to cause an immediate transition from STOP0 to SAFE mode to be interrupt requests. This is done in the MC_RGM's ‘Functional' Event Alternate Request register (RGM_FEAR).

e5099: MC_ME: possibility of the machine check on low power mode exit (MPC5645S only)

Errata type: Information

Description: When executing from the flash and entering a Low-Power Mode (LPM) where the flash is in low-power or power-down mode, 2-4 clock cycles exist at the beginning of the RUNx to LPM transition during which a wakeup or interrupt will generate a checkstop due to the flash not being available on RUNx mode re-entry. This will cause either a checkstop reset or machine check interrupt.

Workaround: Workaround 1

The premise of Workaround 1 is to deal with the issue as it occurs: that is, enter and exit LPM as normal executing application code from flash memory. However, in the event the MCU exits LPM and the flash memory is not available, the next instruction fetch to flash memory will result in the core checkstop state as a result of a bus error. The checkstop state will generate a checkstop reset unless the machine check exception is enabled at the core. In the case of this problem occurring and the machine check exception being enabled, the application software can handle this event if the exception handler code is placed in RAM.

Workaround 2

The application can be configured to avoid the checkstop reset or machine check interrupt. To do this the code initiating the LPM transition has be executed in RAM. By this means, if the LPM mode is exited prior to the flash memory being returned to normal mode, any instruction fetch will be from the available RAM rather than the unavailable flash memory. The application software can then wait until flash is ready by only checking flash status flags in RAM. When the flash memory is ready, code execution can then return to the flash memory.

Workaround 3

For MPC5645S devices it is not necessary to run a work around from RAM, when the machine check is present due to a premature low power mode abort, user may prepare code to verify the flash status in the machine check exception subroutine. MPC5645S devices will cause a few attempts until the flash is successfully restarted and the user code is executed, after this user may return to run from flash again.

Refer to engineering bulleting EB770 on the web, for test case and work around examples.

e7953: ME: All peripherals that will be disabled in the target mode must have their interrupt flags cleared prior to target mode entry

Errata type: Errata

Description: Before entering the target mode, software must ensure that all interrupt flags are cleared for those peripheral that are programmed to be disabled in the target mode. A pending interrupt from these peripherals at target mode entry will block the mode transition or possibly lead to unspecified behaviour.

Workaround: For those peripherals that are to be disabled in the target mode the user has 2 options:

1. Mask those peripheral interrupts and clear the peripheral interrupt flags prior to the target mode request.

2. Through the target mode request ensure that all those peripheral interrupts can be serviced by the core.

e4114: MPC5645S RM does not correctly describe slew rate control bits

Errata type: Information

Description: The fast pads in MPC5645S do not have the slew rate control bits in the same physical location as the medium and slow pads. Instead of the SRC[1:0] control being in bits PCR[12:13] they are in PCR[8:9] instead. PCR[12:13] are writable but have no effect on the behaviour of the pad. The following fast pads exist on MPC5645S: GPIO[127], MCKO, GPIO[160], GPIO[97], GPIO[119] & GPIO[85].

Workaround: Use bits PCR[8:9] for configuring slew rate for fast pads.

e6726: NPC: MCKO clock may be gated one clock period early when MCKO frequency is programmed as SYS_CLK/8.and gating is enabled

Errata type: Errata

Description: The Nexus auxiliary message clock (MCKO) may be gated one clock period early when the MCKO frequency is programmed as SYS_CLK/8 in the Nexus Port Controller Port Configuration Register (NPC_PCR[MCKO_DIV]=111) and the MCKO gating function is enabled (NPC_PCR[MCKO_GT]=1). In this case, the last MCKO received by the tool prior to

the gating will correspond to the END_MESSAGE state. The tool will not receive an MCKO to indicate the transition to the IDLE state, even though the NPC will transition to the IDLE state internally. Upon re-enabling of MCKO, the first MCKO edge will drive the Message Start/End Output (MSEO=11) and move the tool's state to IDLE.

Workaround: Expect to receive the MCKO edge corresponding to the IDLE state upon re-enabling of MCKO after MCKO has been gated.

e6481: NZ4C3/NZ7C3: Erroneous Resource Full Message under certain conditions

Errata type: Errata

Description: The e200zx core Nexus interface may transmit an erroneous Resource Full Message (RFM) following a Program Trace history message with a full history buffer. The History information for both of the messages are the same and the RFM should have not been transmitted. This occurs when the instruction following the indirect branch instruction (which triggers the Program Trace History message) would have incremented the History field. The instructions must be executed in back to back cycles for this problem to occur. This is the only case that cases this condition.

Workaround: There are three possible workarounds for this issue.

(1) Tools can check to see if the Program Trace History message and proceeding Resource Full Message (RFM) have the same history information. If the history is the same, then the RFM is extraneous and can be ignored.

(2) Code can be rewritten to avoid the History Resource Full Messages at certain parts of the code. Insert 2 NOP instructions between problematic code. Or inserting an "isync" or a indirect branch some where in the code sequence to breakup/change the flow.

(3) If possible, use Traditional Program Trace mode can be used to avoid the issue completely. However, depending on other conditions (Nexus port width, Nexus Port speed, and other enabled trace types), overflows of the port could occur.

e7120: NZxC3: DQTAG implemented as variable length field in DQM message

Errata type: Errata

Description: The e200zx core implements the Data Tag (DQTAG) field of the Nexus Data Acquisition Message (DQM) as a variable length packet instead of an 8-bit fixed length packet. This may result in an extra clock ("beat") in the DQM trace message depending on the Nexus port width selected for the device.

Workaround: Tools should decode the DQTAG field as a variable length packet instead of a fixed length packet.

e9066: PAD RING: The Device may lock-up into reset loop while exiting from standby mode or at power up

Errata type: Errata

Description: The device may lock-up (core will not execute code) while exiting Standby mode or during power up of the VDD12 supply if the TCK pin is in floating or high state. The watchdog timer will continuously trigger a reset signal at the default 15 ms interval.

A destructive reset triggered by the watchdog or by any other reset source will not recover the device from the lock-up state.

Workaround: If a lock-up occurs, the device operation can be recovered by driving the TCK pin at logic level low and the device will recover at next reset de-assertion.

Use 4.7K ohm resistor to pull TCK pin down to prevent any occurrence of the failure.

e3624: RLE: Decode operation freezes when configured for very small data transfers

Errata type: Errata

Description: When the RLE decode module is configured to decode small blocks of encoded data it may enter a state where an operation either never completes or completes the current operation but causes the next decode operation to fail. The fail condition depends on the configuration of the RLE module and the eDMA channel used to copy data into the module.

Workaround: Use at least 32 byte images (compressed and non-compressed) when using DMA modes on RLE.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.