MPC8309 Chip Errata

The following table provides a cross-reference to match the revision code in the processor version register to the revision level marked on the device.

Table 1. Revision Level to Part Marking Cross-Reference

Part	Revision	Processor Version Register Value	System Version Register Value	Mask
MPC8309	1.1	0x8085_0020	0x8110_0011	N83A
MPC8309	1.0	0x8085_0020	0x8110_0010	N83A

This document details all known silicon errata for the MPC8309. The following table provides a revision history for this document.

Table 2. Document Revision History

Revision	Date	Significant Changes
2	10/2015	Added the following errata: • eSDHC A-004577, A-005055, and A-009204 • QE A-004261 • USB A-003827, A-003829, A-003837 and A-003845 Modified the following errata: • eSDHC23, eSDHC16 • General 17 • USB-A003 • Renamed USB38 to USB erratum A-003817
1	09/2011	Added CPU-A022, USB-A007
0	01/2011	Initial release

Table 3 summarizes all known errata.

Table 3. Summary of Silicon Errata and Applicable Revision

Errata	Name	Projected Solution	Silicon Rev.	
			1.0	1.1
CPU				
CPU-A002	CPU may hang after load from cache-inhibited, unguarded memory	No plans to fix	Yes	Yes
CPU-A022	The e300 core may hang while using critical interrupt	No plans to fix	Yes	Yes
CPU6	DTLB LRU logic does not function correctly	No plans to fix	Yes	Yes
DDR				
DDR21	MCK/$\overline{\text{MCK}}$ AC differential crosspoint voltage outside JEDEC specifications	No plans to fix	Yes	Yes
eLBC				
eLBC-A001	Simultaneous FCM and GPCM or UPM operation may erroneously trigger bus monitor timeout	No plans to fix	Yes	Yes
eLBC2	UPM does not have indication of completion of a Run Pattern special operation	No plans to fix	Yes	Yes
eLBC5	LTEATR and LTEAR may show incorrect values under certain scenarios	No plans to fix	Yes	Yes
eSDHC				
eSDHC16	Manual Asynchronous CMD12 abort operation causes protocol violations	No plans to fix	Yes	Yes
eSDHC23	CMD CRC error or CMD index error may be set for CMD without data while CMD with data is in progress	No plans to fix	Yes	Yes
eSDHC-A002	BLKATTR[BLKCNT] does not return to 0 after Transfer Complete for multi-block transfer	No plans to fix	Yes	Yes
A-004577	PRSSTAT[DLA] bit does not reflect the data line state when any command with busy (R1b) is issued	No plans to fix	Yes	Yes
A-005055	A glitch is generated on the card clock with software reset or a clock divider change	No plans to fix	Yes	Yes
A-009204	System bus may hang if software register SYSCTL[RSTD] is set while DMA transactions are active	No plans to fix	Yes	Yes
General				
General16	Enabling I^2C could cause I^2C bus freeze when other I^2C devices communicate	No plans to fix	Yes	Yes
General17	DUART: Break detection triggered multiple times for a single break assertion	No plans to fix	Yes	Yes
General19	Boot from eSDHC may fail	Fixed in Rev 1.1.	Yes	Yes
PCI				
PCI15	Assertion of $\overline{\text{STOP}}$ by a target device on the last beat of a PCI memory write transaction can cause a hang	No plans to fix	Yes	Yes
PCI19	Dual-address cycle inbound write accesses can cause data corruption	No plans to fix	Yes	Yes

Table continues on the next page...

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

Errata	Name	Projected Solution	Silicon Rev.	
			1.0	1.1
QE				
QE_BISYNC1	RxBD in continuous mode is not closed after an Rx parity error	No plans to fix	Yes	Yes
QE_ENET4	RMON errors	No plans to fix	Yes	Yes
QE_ENET5	Rx MAC might accept dropped frame	No plans to fix	Yes	Yes
QE_ENET6	Half-duplex collision on FCS of short frame may cause transmit lockup	No plans to fix	Yes	Yes
A-004261	QE-ENET: After the reception of a PAUSE frame, an Ethernet frame may be transmitted during the PAUSE window	No plans to fix	Yes	Yes
QE_ENET-A001	MAC: Pause time may be shorter than specified if transmit in progress	No plans to fix	Yes	Yes
QE_General-A003	High Tx Virtual FIFO threshold size can cause UCC to halt	No plans to fix	Yes	Yes
QE_General-A005	8-bit and 16-bit accesses not supported	Fixed in Rev 1.1	Yes	Yes
QE_UART1	Overrun indication is not reported on the current BD	No plans to fix	Yes	Yes
QE_UART2	DRT mode (UPSMR[DRT] = 1) is not functional	No plans to fix	Yes	Yes
QE_UART3	Simultaneous loopback and echo mode is not functional	No plans to fix	Yes	Yes
USB				
USB15	Read of PERIODICLISTBASE after successive writes may return a wrong value in host mode	No plans to fix	Yes	Yes
USB19	USBDR in Host mode does not generate an interrupt upon detection of a CRC16/PID/Timeout error when a received IN data packet is corrupted	No plans to fix	Yes	Yes
USB21	SE0_NAK issue	No plans to fix	Yes	Yes
USB25	In host mode, when the software forces a port resume by writing into the FPR bit of the portsc register, the port change detect interrupt bit is falsely fired	No plans to fix	Yes	Yes
USB26	NackCnt field is not decremented when received NYET during FS/LS Bulk/Interrupt mode	No plans to fix	Yes	Yes
USB27	When an ACK or NAK is sent from the device in response to a PING, the CERR counter value is not being reset to the initial value	No plans to fix	Yes	Yes
USB28	In device mode, when receiving a Token OUT, if a Rx flush command is issued at the same time, to the same endpoint, the packet will be lost	No plans to fix	Yes	Yes
USB29	Priming ISO over SOF will cause transmitting bad packet with correct CRC	No plans to fix	Yes	Yes
USB31	Transmit data loss based on bus latency	No plans to fix	Yes	Yes
USB32	Missing SOFs and false babble error due to Rx FIFO overflow	No plans to fix	Yes	Yes
USB33	No error interrupt and no status will be generated due to ISO mult3 fulfillment error	No plans to fix	Yes	Yes
USB34	NAK counter decremented after receiving a NYET from device	No plans to fix	Yes	Yes
USB35	Core device fails when it receives two OUT transactions in a short time	No plans to fix	Yes	Yes
USB36	CRC not inverted when host under-runs on OUT transactions	No plans to fix	Yes	Yes

Table continues on the next page...

Table 3. Summary of Silicon Errata and Applicable Revision (continued)

Errata	Name	Projected Solution	Silicon Rev.	
			1.0	1.1
USB37	OTG Controller as Host does not support Data-line Pulsing Session Request Protocol	No plans to fix	Yes	Yes
A-003817	USB Controller locks after Test mode "Test_K" is completed	No plans to fix	Yes	Yes
USB-A001	Last read of the current dTD done after USB interrupt	No plans to fix	Yes	Yes
USB-A002	Device does not respond to INs after receiving corrupted handshake from previous IN transaction	No plans to fix	Yes	Yes
USB-A003	Illegal NOPID TX CMD issued by USB controller with ULPI interface	No plans to fix	Yes	Yes
USB-A005	ULPI Viewport not Working for Read or Write Commands With Extended Address	No plans to fix	Yes	Yes
USB-A006	READ operation on some USB registers returns an incorrect value.	No plans to fix	Yes	Yes
USB-A007	Host controller fails to enter the PING state on timeout during High Speed Bulk OUT/DATA transaction	No plans to fix	Yes	Yes
A-003827	DATA PID error interrupt issued twice for the same high bandwidth ISO transfer	No plans to fix	Yes	Yes
A-003829	Host detects frame babble but does not halt the port or generate an interrupt	No plans to fix	Yes	Yes
A-003837	When operating in test mode, the CSC bit does not get set to 1 to indicate a change on CCS	No plans to fix	Yes	Yes
A-003845	Frame scheduling robustness-Host may issue token too close to uframe boundary	No plans to fix	Yes	Yes

CPU-A002: CPU may hang after load from cache-inhibited, unguarded memory

Description: There is a one-core-clock-cycle window of opportunity for a snoop to collide with the data returned from cache-inhibited memory to a load that has been cancelled. This collision can hang the data cache in a busy state, prohibiting further data cache accesses. The snoop address is immaterial.

The load can be cancelled if it meets the following conditions: it was executing speculatively beyond a conditional branch that resolved unfavorably or was pre-empted by an external interrupt or decrementer interrupt that took priority. The load must be from a cache-inhibited, non-guarded memory block. A load from cacheable memory, even if it misses in the cache, does not hang the data cache in a busy state, and if the memory block is marked guarded, the load will not be executed out of order.

An external master must put addresses on the bus with the attribute of memory coherency required (Global) so that the CPU allows snooping of the data cache. External masters have to be programmed to snoop.

Impact: CPU halts or stops executing on a subsequent load or store that finds the data cache busy. This load or store does not complete, and the data cache stays busy until a hardware or software reset ($\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$). Debug tools will reveal that the program counter is stopped and pointing to the load or store that cannot complete.

Workaround: Use one of the following options:

- Set bit 14 in the HID2 register. This bit disables a minor feature that attempted to take advantage of cache hits under a cancelled cache miss which was still waiting for data. There should be no performance loss from disabling this feature. HID2[14] is not implemented on any other e300 devices. Setting it will have no effect on other devices or any future revisions.
- Mark all data memory blocks from which loads could occur “cacheable” or “guarded.”

CPU-A022: The e300 core may hang while using critical interrupt

Description: If BOTH critical interrupt AND normal interrupt types are used in a system, the e300 core may hang.

Impact: The processor may stop dispatching instructions until a hardware reset($\overline{\text{HRESET}}$). Debug tools will not be able to read any register correctly except program counter IAR which points to a location in the critical interrupt vector.

Workaround: If both critical interrupt and normal interrupt types are used, then instead of using an **rfi** instruction at the end of every exception handler, replace the **rfi** with the following:

1. Disable critical interrupts by setting MSR[CE] to 0 with a **mtspr** instruction.
2. Copy SRR0 and SRR1 to CSRR0 and CSRR1, respectively.
3. Execute an **rfci** instruction. This enables MSR[CE] and any other bits that the original **rfi** would have set including the MSR[EE].

Sample Code:

```
// Disable MSR[CE]
mfmsr r2
lis r3, 0xffff
ori r3, r3, 0xff7f
and r2, r2, r3
sync
mtmsr r2
isync
// Copy SRR0, SRR1 to CSRR0 and CSRR1
mfspr r2, srr0
mfspr r3, srr1
mtspr csrr0, r2
mtspr csrr1, r3
...restore GPRs
rfci
```

Fix plan: No plans to fix

CPU6: DTLB LRU logic does not function correctly

Description: The DTLB is implemented as 2-way set associative with 32 entries per way. EA[15:19] is used to determine which one of the 32 entries of both ways. When a DTLB miss occurs, normally the CPU provides information (through SRR1 bit 14, DTLB replacement way) to indicate which of the two ways the software (DTLB exception handler or the software table walk routine) should use to bring in the new page. Presumably, the CPU provides the information, through SRR1 bit 14, based on the LRU (least recently used) algorithm. However, because of this bug, this is not the case.

In fact, for any given EA[15:19], when a DTLB miss occurs, SRR1 bit 14 always indicates that way1 should be used or replaced. (Except if it is the very first DTLB miss after hard reset. In this case, SRR1 bit 14 indicates way0 should be used.)

In other words, if the DTLB exception routine follows the SRR1 bit 14's suggestion to do the TLB replacement, it always replaces the one in way1. In addition, whatever has been loaded in way0 is effectively locked and is not replaced.

Impact: Performance degradation due to the reduction of usable DTLB entries.

Workaround: In the software, use one word (32 bits) to keep the record of the DTLB way that is Least Recently Written (LRW). Use this information to overwrite the SRR1 bit 14 (DTLB replacement way) when a Data Translation Miss exception occurs. Basically, the LRU (least recently used) hardware algorithm is changed to an LRW (least recently written) software algorithm.

For the system that has no secondary storage (such as a hard drive), it is highly recommended to set the C (Change) bit during the DTLB load exception to optimize the performance.

For a system that has a secondary storage and the OS does a page swap, the OS can choose whether to set the C bit in the DTLB load exception.

If the C bit is set in the DTLB load exception, it can preempt the subsequent DTLB store exception to the same page. However, since all the pages are marked as changed, during the page swap all pages must be written back to the secondary storage regardless of whether they have really been changed or not.

If the C bit is not set in the DTLB load exception with the LRW algorithm, a subsequent store to the same page as the previous load will use a separate entry. Therefore, one page occupies both ways. This causes inefficiency for the DTLB allocation but may save time during the page swap since the page change status is correctly marked.

Fix plan: No plans to fix

DDR21: MCK/$\overline{\text{MCK}}$ AC differential crosspoint voltage outside JEDEC specifications

Description: The crossover points of the MCK and $\overline{\text{MCK}}$ signals of the DDR controller are not meeting JESD79-2C specifications, which indicates that the crossover points should lie within ±125 mV range of reference voltage.

Impact: Disagreement with the JESD79-2C standard.

Workaround: To meet the JESD79-2C crosspoint voltage specifications, we recommend implementing the following connections between the DDR controller and DDR2 memory:

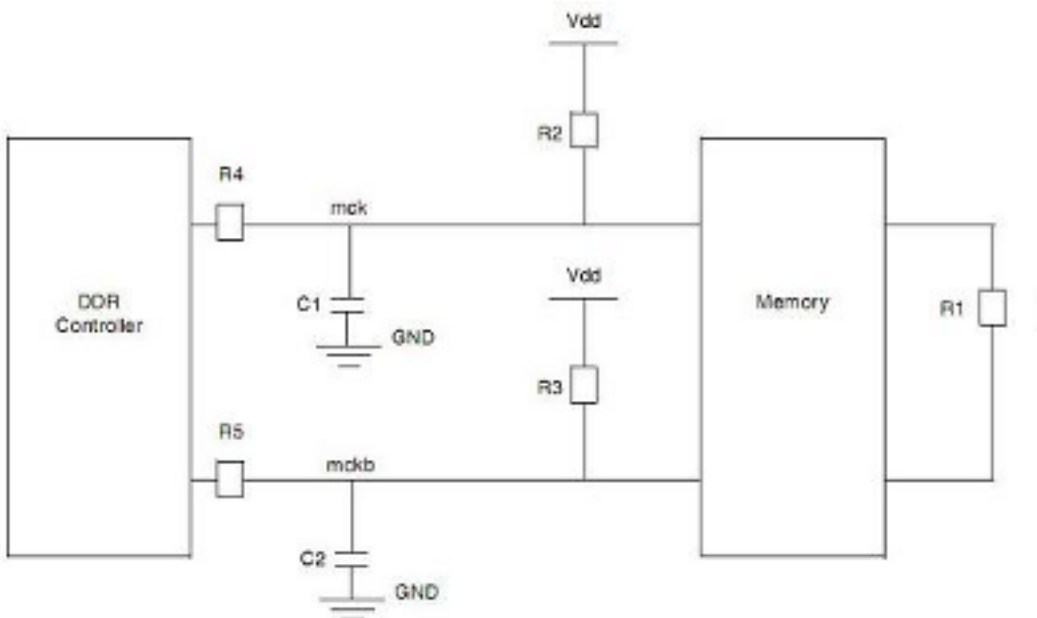

Figure 1. Recommended connections for DDR controller and memory

Effect of different circuit components on MCK and $\overline{\text{MCK}}$ signals:

1. Increasing R2 and R3 shifts signal levels up (used as pull up).
2. Increasing C1 increases rise/fall time of mck signal.
3. Increasing C2 increases rise/fall time of mckb signal.
4. Reducing R1 can help in shifting the signals up.
5. R4 and R5 can be used to make termination proper.

Component Placing:

R4, R5, C1, C2 near to SOC, R1, R2, and R3 have been placed at the end of clock transmission lines.

By considering these suggestions and choosing proper values of components for a particular board layout, one can keep crossover points within range.

We recommend varying termination resistor R1 first. Lowering R1 can help increase the voltage levels up. The only disadvantage is that it may add up more reflections. Therefore, the user must decide accordingly. If it does not give sufficient margin, the user can add the pull-up resistor (R3 and R4) option to further increase the voltage levels.

Exact values of components vary from design to design and some of them may not be required fpr all designs. We recommend that you make provisions for all these options in your design.

Fix plan: No plans to fix

eLBC-A001: Simultaneous FCM and GPCM or UPM operation may erroneously trigger bus monitor timeout

Description: When the FCM is in the middle of a long transaction, such as NAND erase or write, another transaction on the GPCM or UPM triggers the bus monitor to start immediately for the GPCM or UPM, even though the GPCM or UPM is still waiting for the FCM to finish and has not yet started its transaction. If the bus monitor timeout value is not programmed for a sufficiently large value, the local bus monitor may time out. This timeout corrupts the current NAND Flash operation and terminate the GPCM or UPM operation.

Impact: Local bus monitor may time out unexpectedly and corrupt the NAND transaction.

Workaround: Set the local bus monitor timeout value to the maximum by setting LBCR[BMT] = 0 and LBCR[BMTPS] = 0xF.

Fix plan: No plans to fix

eLBC2: UPM does not have indication of completion of a Run Pattern special operation

Description: A UPM or FCM special operation is initiated by writing to MxMR[OP] or FCM[OP] and then triggering the special operation either through the LSOR register or by performing a dummy access to the bank.

The UPM and FCM are expected to have different indications of when the special operation is completed. The FCM will see the LTESR[CC] bit set when such a special operation is completed. The UPM will see MxMR[MAD] increment when a write to or read from UPM array special operation completes. However, the UPM does not have any status indication of completion of the run pattern special operation.

The following two scenarios could be affected by this erratum:

1. A UPM Run Pattern special operation is initiated and a second UPM command is issued before the Run Pattern is completed.

   If the above scenario occurs the programmed mode registers could be altered according to the second operation and cause the current/first operation to encounter errors due to mode changes in the middle of the operation. Note that if the second command issued is a Run Pattern operation and it does not change the mode registers, the first operation should not encounter errors.

2. A write to LSOR initiates a UPM Run Pattern special operation and then LSOR is written too again, to initiate a second special operation that requires the mode registers to change while the first Run Pattern operation is in progress.

   If the second special operation issued does not change the programming mode of the first Run Pattern operation because the operation is either exactly the same as the first or it requires programming of an independent set of registers then the Run Pattern operation should not encounter errors.

The behavior of the eLBC is unpredictable if the Run Pattern special operation mode is altered between initiation of the operation and the relevant memory controller completing the operation.

Impact: Because of this erratum, when a UPM run pattern special operation is to be followed by any other UPM command for which the mode registers need to change, the Run Pattern operation may not be handled properly. Software does not have any means to confirm when the current run pattern special operation has completed so that register programming for the next operation can be done safely.

Workaround: None

Fix plan: No plans to fix

eLBC5: LTEATR and LTEAR may show incorrect values under certain scenarios

Description: The eLBC IP acks any transaction request when FCM special operation is in progress. In such a scenario when any one of the errors/events occur (such as Bus Monitor Timeout, Write Protect, Parity error, Atomic error, FCM command completion, or UPM command completion except for the CS error), the registers LTEAR and LTEATR capture the address and attributes of the most recently req-acked transaction instead of the FCM special operation that caused error. Hence indeterministic value may show up in those registers.

Impact: LTEAR and LTEATR cannot be used for debugging in this scenario.

Workaround: None

Fix plan: No plans to fix

eSDHC16: Manual Asynchronous CMD12 abort operation causes protocol violations

Description: There may be protocol violations if a manual (software) asynchronous CMD12 is used to abort data transfer. Due to this erratum, the eSDHC controller continues driving data after a manual (software) asynchronous CMD12 is issued. Therefore, it may cause a conflict on the data lines on the SD bus.

Impact: Manual asynchronous CMD12 to terminate data transfer cannot be used.

Workaround: Do not issue a manual asynchronous CMD12. Instead, use a (software) synchronous CMD12 or AUTOCMD12 to abort data transfer.

Due to erratum A-004577, CMD13 needs to be sent after TC of the data transfer command for which AutoCMD12 is enabled. See A-004577 for details.

For a manual synchronous CMD12, the following steps are required:

1. Set PROCTL[SABGREQ] = 1
2. Wait for IRQSTAT[TC] bit set, or Transfer Complete Interrupt
3. Set IRQSTAT[TC] = 1
4. Issue CMD12 after checking PRSSTAT[CIHB] = 0
5. Set both SYSCTL[RSTD] and SYSCTL[RSTC]

Fix plan: No plans to fix

eSDHC23: CMD CRC error or CMD index error may be set for CMD without data while CMD with data is in progress

Description: While a command with data is in progress and a command without data is issued, for example CMD13, an invalid command CRC error (IRQSTAT[CCE]) and/or command index error (IRQSTAT[CIE]) might be detected. This can happen when SDHC_CLK shuts off (due to buffer danger) exactly after START bit of response is detected by the eSDHC. While the clock is gated, the eSDHC samples an incorrect value from the command line thus sampling an incorrect command index in the response and eventually generating a command CRC and index error.

Impact: This issue occurs under a rare condition. The data transfer itself is not impacted.

Workaround: On detecting a command CRC error (IRQSTAT[CCE]) or command index error (IRQSTAT[CIE]), perform error recovery and re-issue the command without data. If Auto CMD12 is enabled for data transfer then Auto CMD12 won't be issued by hardware, so software needs to issue it after data transfer completion.

Fix plan: No plans to fix

eSDHC-A002: BLKATTR[BLKCNT] does not return to 0 after Transfer Complete for multi-block transfer

Description: For a multi-block transfer, when the XFERTYP[BCEN](block count enable) is 1, BLKATTR[BLKCNT] should decrement to 0 after Transfer Complete (TC)((IRQSTAT[TC] = 1). However, due to this erratum, BLKATTR[BLKCNT] returns to the initial value which was programmed while issuing the multi-block transfer command.

Impact: The value of BLKATTR[BLKCNT] is unreliable and cannot be read to confirm a Transfer Complete.

Workaround: For a multi-block transfer, after IRQSTAT[TC] = 1, software should not rely on the value of BLKATTR[BLKCNT].

Fix plan: No plans to fix

A-004577: PRSSTAT[DLA] bit does not reflect the data line state when any command with busy (R1b) is issued

Affects: eSDHC

Description: When an AutoCMD12 or any command with busy (R1b) is issued, PRSSTAT[DLA] bit should reflect the data line state. However, due to this erratum, PRSSTAT[DLA] is not applicable to detect data busy state. Furthermore, the corresponding transfer complete interrupt is not generated. However, the AutoCMD12 or any command with busy (R1b) can still be used with the restriction that busy needs to be de-asserted before sending new data command.

Impact: When an AutoCMD12 or any command with busy (R1b) is issued, PRSSTAT[DLA] bit does not reliably reflect the data line state.

Workaround: Software needs to wait for busy de-assertion before issuing any new data command. DAT0 line could be polled, but robust solution would be to keep sending CMD13(SEND_STATUS) until card reaches “trans" state.

- For AutoCMD12, CMD13 needs to be sent after TC of the data transfer command for which AutoCMD12 is enabled.
- For other command with busy, CMD13 needs to be sent after the command with busy completion(IRQSTAT[CC] = 1).

Fix plan: No plans to fix

A-005055: A glitch is generated on the card clock with software reset or a clock divider change

Affects: eSDHC

Description: A glitch may occur on the SDHC card clock when the software sets the SYSCTL[RSTA] bit (that is, performs a software reset). It can also be generated by setting the clock divider value. The glitch produced can cause the external card to switch to an unknown state. The occurrence is not deterministic and it happens rarely. The next command causes a timeout error(IRQSTAT[CTOE]) after this issue occurs.

Impact: Changing the frequency or performing a software reset for all may not work reliably.

Workaround: When the timeout error occurs for the command right after the SYSCTL[RSTA] is set or the clock divider value is changed, send CMD0 to bring the card to idle state, and perform re-initialization again. If the error occurs again, repeat this step until the initialization process completes.

Fix plan: No plans to fix

A-009204: System bus may hang if software register SYSCTL[RSTD] is set while DMA transactions are active

Affects: eSDHC

Description: In the event of that any data error (like, IRQSTAT[DCE]) occurs during an eSDHC data transaction where DMA is used for data transfer to/from the system memory, setting the SYSCTL[RSTD] register may cause a system hang. If software sets the register SYSCTL[RSTD] to 1 for error recovery while DMA transferring is not complete, eSDHC may hang the system bus. This happens because the software register SYSCTL[RSTD] resets the DMA engine without waiting for the completion of pending system transactions.

Impact: Causes system bus to hang.

Workaround: Add a 5 ms delay before setting SYSCTL[RSTD] to make sure all the DMA transfers are finished.

Fix plan: No plans to fix

General16: Enabling I²C could cause I²C bus freeze when other I²C devices communicate

Description: When the I²C controller is enabled by software, if the signal SCL is high and the signal SDA is low, and the I²C address matches the data pattern on the SDA bus right after the enabling, an ACK is issued on the bus. The ACK is issued because the I²C controller detects a START condition due to the nature of the SCL and SDA signals at the point of enablement. When this occurs, it may cause the I²C bus to freeze. However, it happens very rarely due to the need of two conditions occurring at the same time.

Impact: Enabling the I²C controller may cause the I²C bus to freeze while other I²C devices communicate on the bus.

Workaround: Use one of the following options:

- Enable the I²C controller before starting any I²C communications on the bus.

  This is preferred solution.

- If the I²C is configured as a slave, implement the following steps:

  a. Software enables the device by setting I2CnCR[MEN] = 1, and start a timer.
  b. Delay for four I²C bus clocks.
  c. Check Bus Busy bit (I2CnSR[MBB])

```
if MBB == 0
         Jump to Step 6; (Good condition. Go to normal operation.)
else
         Disable device (I2CnCR[MEN] = 0)
```

  d. Reconfigure all the I²C registers if necessary.
  e. Go back to Step 1
  f. Normal operation.

Fix plan: No plans to fix

General17: DUART: Break detection triggered multiple times for a single break assertion

Description: A DUART break signal is defined as a logic zero being present on the UART data pin for a time longer than (START bit + Data bits + Parity bit + Stop bits).The break signal persists until the data signal rises to a logic one.

A received break is detected by reading the ULSRn and checking for BI=1. This read to ULSRn will clear the BI bit. Once the break is detected, the normal handling of the break condition is to read the URBR to clear the ULSRn[DR] bit. The expected behavior is that the ULSRn[BI] and ULSRn[DR] bits will not get set again for the duration of the break signal assertion. However, the ULSRn[BI] and ULSRn[DR] bits will continue to get set each character period after they are cleared. This will continue for the entire duration of the break signal.

At the end of the break signal, a random character may be falsely detected and received in the URBR, with the ULSRn[DR] being set.

Impact: The ULSRn[BI] and ULSRn[DR] bits will get set multiple times, approximately once every character period, for a single break signal. A random character may be mistakenly received at the end of the break.

Workaround: The break is first detected when ULSRn is read and ULSRn[BI]=1. To prevent the problem from occuring, perform the following sequence when a break is detected:

1. Read URBRn, which will return a value of zero, and will clear the ULSRn[DR] bit
2. Delay at least 1 character period
3. Read URBRn again

ULSR[BI] will remain asserted for the duration of the break. The UART block will not trigger any additional interrupts for the duration of the break.

This workaround requires that the break signal be at least 2 character-lengths in duration.

This workaround applies to both polling and interrupt-driven implementations.

Fix plan: No plans to fix

General 19: Boot from eSDHC may fail

Description: The boot ROM code uses 0000 as timeout value in SYSCTL[DTOCV] to read boot image from eSDHC interface. This might lead to time-out failure with some SD/MMC cards.

Impact: Device may fail to boot with some SD/MMC cards

Workaround: None.

Fix plan: Fixed in Rev 1.1.

PCI15: Assertion of $\overline{\text{STOP}}$ by a target device on the last beat of a PCI memory write transaction can cause a hang

Description: Devices: MPC8309

As a master, the PCI IP block can combine a memory write to the last PCI double word (4 bytes) of a cacheline with a 4 byte memory write to the first PCI double word of the subsequent cacheline.

This only occurs if the second memory write arrives to the PCI IP block before the deassertion of $\overline{\text{FRAME}}$ for the first write transaction. If the writes are combined, the PCI IP block masters a single memory-write transaction on the PCI bus. If for this transaction, the PCI target asserts $\overline{\text{STOP}}$ during the last data beat of the transaction ($\overline{\text{FRAME}}$ is deasserted, but $\overline{\text{TRDY}}$ and $\overline{\text{IRDY}}$are asserted), the transaction completes correctly. A subsequent write transaction other than an 8-byte write transaction causes a hang on the bus. Two different hang conditions can occur:

- If the target disconnects with data on the first beat of this last write transaction, the PCI IP block deasserts $\overline{\text{IRDY}}$ on the same cycle as it deasserts $\overline{\text{FRAME}}$ (PCI protocol violation), and no more transactions will be mastered by the PCI IP block.
- If the target does not disconnect with data on the first beat of this last write transaction, $\overline{\text{IRDY}}$ will be deasserted after the first beat is transferred and will not be asserted anymore after that, causing a hang.

Impact: This affects 32-bit PCI target devices that blindly assert $\overline{\text{STOP}}$ on memory-write transactions, without detecting that the data beat being transferred is the last data beat of the transaction. It can cause a hang.

If the PCI transaction is a one data beat transaction and the target asserts $\overline{\text{STOP}}$ during the transfer of that beat, there is no impact.

Workaround: Hardware workaround:

Ensure that the PCI target device does not assert $\overline{\text{STOP}}$ during the last beat of a PCI memory write transaction that is greater than one data beat and crosses a cacheline boundary. It could assert $\overline{\text{STOP}}$ during the last data beat of the 32-byte cacheline or not assert $\overline{\text{STOP}}$ at all.

Software workarounds:

Set bit 10, the master disabling streaming (MDS) bit, of the PCI bus function register (address 0x44) to prevent the combining of discrete outbound PCI writes or the recombining of writes that cross the 32-byte cacheline boundary into a burst.

Set the PCI latency timer register (offset 0x0D) to zero. A value of zero is the reset value for this register, so keeping this register unmodified after reset prevents the PCI IP block from ever combining writes. It is not necessary to set the PCI latency timer register to zero if the MDS bit is set.

Fix plan: No plans to fix

PCI19: Dual-address cycle inbound write accesses can cause data corruption

Description: Devices: MPC8309

When using a dual-address cycle (DAC) for inbound write accesses and when the IOS is full, (that is, a previous PCI request to the IOS is being retried), the PCI overwrites the address for the IOS with the new address from the bus, despite the transaction being retried on the PCI bus.

Impact: Dual address cycle (DAC) feature cannot be sustained by the device while working as PCI target. As PCI initiator, DAC is not supported.

Workaround:

- When operating in host mode, map inbound windows to 32-bit addressing (below 4G addressing space) where this type of application is allowed.
- When operating in agent mode, the above workaround is not valid. The host is mapping the agents according to the address type in configuration registers, which, in this case, is '10'. Thus, it could map anywhere in the 64-bit address space.

Fix plan: No plans to fix

QE_BISYNC1: RxBD in continuous mode is not closed after an Rx parity error

Description: RxBD in continuous mode (CM is set in the RxBD) is not closed after an Rx parity error. After receiving a parity error, the controller sets the PR status bit in the RxBD, but does not clear the Empty control bit as expected.

Impact: The receiver does not work as expected in continuous mode when an Rx parity error occurs.

Workaround: When using CM in the receiver, the software should poll PR status during the routine handling of RxBD.

Fix plan: No plans to fix

QE_ENET4: RMON errors

Description: RMON statistics are inaccurate under the following conditions:

- Transmit collision counter status increments after underrun even though no collision has occurred. This bug only shows up when CRC is not appended.
- The MAC (TX module) does not report CRC error on a frame that is transmitted if the previous frame had underrun.
- When BPNB (back pressure no backoff) is enabled the MAC may report a wrong value of collision count (a value that is less than the real collision count).
- If all the following conditions occur, Tx CRC error might not be reported by the Tx MAC:
  - Underrun on last data byte of previous frame
  - No pad or CRC is set on MACCFG2
  - No pad or CRC bit in TxBD is set on current frame (TxBD[TC,PAD/CRC])
  - There is a CRC error on the current frame

Impact: Inaccurate RMON statistics.

Workaround: None

Fix plan: No plans to fix

QE_ENET5: Rx MAC might accept dropped frame

Description: When a dropped frame contains data with 5d pattern, the MAC assumes that it is a new frame and cancels the drop event. The MAC effectively accepts this partial/dropped frame. However, the MAC receive statistic vector (receive previous packet dropped) is asserted to indicate that the received packet is dropped.

Impact: Partial/incomplete frame being received and a false count in the MIBs. However, this partial frame will be rejected by the receiver (due to CRC error).

Workaround: None

Fix plan: No plans to fix

QE_ENET6: Half-duplex collision on FCS of short frame may cause transmit lockup

Description: In half-duplex mode, if a collision occurs in the FCS bytes of a short (less than 64 bytes) frame, the Ethernet MAC may lock up and stop transmitting data or control frames. Only a reset of the controller can restore proper operation once it is locked up.

Impact: A collision on hardware-generated FCS bytes of a short frame in half-duplex mode may cause a transmit lockup.

Workaround: Use one of the following options:

- Set MACCFG2[PAD/CRC] = 1, which pads all short Tx frames to 64 bytes.
- Use software-generated CRC (MACCFG2[PAD/CRC] = 0, MACCFG2[CRC_EN] = 0 and TxBD[TC] = 0).

Fix plan: No plans to fix

A-004261: QE-ENET: After the reception of a PAUSE frame, an Ethernet frame may be transmitted during the PAUSE window

Affects: QE-ENET

Description: The Ethernet MAC might transmit a frame while being in a PAUSE state and this frame could consume the intended PAUSE duration. This is a violation of the specification which states in the QEIWRM "Receiving a Flow control Frame" section that "transmission stops for the time specified in the control frame."

The indications of UCCE[CBPR] and UCCS[BPR] are incorrect as well; the PAUSE indication should have been set only when the UCC Ethernet controller (UEC) transmitter is in PAUSE state and is not transmitting on the line, but the actual behavior is that the PAUSE indication is set while the UEC is transmitting on the line.

Impact: The UCC Ethernet controller may (occasionally) violate the pause duration.

Workaround: None

Fix plan: No plans to fix

QE_ENET-A001: MAC: Pause time may be shorter than specified if transmit in progress

Description: When the Ethernet controller receives a pause frame with PTV!=0, and MACCFG1[Rx Flow] = 1, it completes transmitting any current frame in progress and should then pause for PTV × 512 bit times. The MAC, however, does not take the full transmission time of the current frame into account when calculating the Tx pause time and may pause for 1–2 pause quanta (512–1024 bit times) less than the PTV value.

Impact: The Ethernet controller's transmitter may pause transmission for up to 1024 bit times less than requested in a receive pause frame. If the PTV value does not contain at least 2 pause quanta worth of margin, this may lead to receive buffer overflows in the link partner.

Because the transmit pause does not take effect until after the current frame completes transmitting, the link partner's pause frame generator must already include the maximum frame size as margin when calculating the pause time value to use to prevent overflow of the receiver's buffers.

Workaround: Add 2 pause quanta to the pause time value used when generating pause frames to prevent receive buffer overflow.

Fix plan: No plans to fix

QE_General-A003: High Tx Virtual FIFO threshold size can cause UCC to halt

Description: Due to the structure of Tx Virtual FIFO, it is possible for the Tx Virtual FIFO to contain part of a frame when there is no room for the remainder of the frame (i.e. when the frame size is on the scale of magnitude of the Tx Virtual FIFO size).

If the Tx Virtual FIFO contains a partial frame, the transmission of the frame may start even if fewer than UTFTT (UCC Tx Virtual FIFO Threshold) bytes of data reside in the Tx Virtual FIFO. The only reason the frame will not start transmission is if it meets a specific sequence in which the UCC transmit-start condition is not met for the frame. This is rare but possible.

The erroneous behavior is a result of incorrect recovery after the pausing data-retrieve phase from the BD buffer to the Tx Virtual FIFO in a certain internal stage.

Reaching the UTFTT watermark in the Tx Virtual FIFO for each transmit frame prevents this sequence. UTFTT has an upper limit value, which is less than the UTFS (UCC Tx Virtual FIFO Size). For each UTFS value there is a maximum value of UTFTT that corresponds to it. Setting UTFTT higher than the suggested value can result in a UCC halt, and Ethernet transmission stops.

NOTE

1. As defined in the reference manual, UTFTT refers to the payload of a single Ethernet frame.
2. Transmission may start before surpassing the threshold if there is no space for additional data in Tx Virtual FIFO. This is the correct behavior.

Impact: The UCC may stop transmitting when a large frame is partially stored in the Tx Virtual FIFO before the transmission of the frame starts.

Note that the Tx BD ring buffer configuration (data allocation to BD buffers) may adversely affect Tx Virtual FIFO utilization, thus preventing the Tx Virtual FIFO from reaching the threshold for transmission.

Workaround: Use one of the following options:

- To recover from a possible failure, the application code should configure UTFTT to 0x40 and then set it back to the original value. This sequence triggers transmission from the point it stopped. Detection of a UCC halt can be done by monitoring Tx BD ring vacancy. A full Tx BD ring may indicate the UCC has halted.
- To prevent the failure, configure UTFS and UTFTT so that it is possible to store UTFTT data (payload) bytes in Tx VFIFO.
  a. For the usage of a single buffer per frame, here are the maximum UTFTT values that can be used:
     - Ethernet Controller: Set UTFTT < [(0.9375 x UTFS) – 128]
     - HDLC Controller: Set UTFTT < [(0.7500 x UTFS) – 32]
     - For example, for an Ethernet controller with UTFS = 1024, set UTFTT < 832.
  b. For the usage of a multiple buffer per frame, here are the maximum UTFTT values that can be used: (M is the smallest buffer size used)
     - Ethernet Controller: Set UTFTT < [(UTFS x (M – 8) ÷ M) – 128]
     - HDLC Controller: Set UTFTT < [(UTFS x (M – 8) ÷ M) – 32]
     - For example, for an Ethernet Controller with UTFS = 1024 and M = 64, set UTFTT < 768.

Fix plan: No plans to fix

QE_General-A005: 8-bit and 16-bit accesses not supported

Description: 8 or 16 bit read/write access to QUICC Engine configuration space (IMMRBAR space) does not return/modify the register with the correct data.

Impact: Software will not be able to perform 8/16 bit accesses to QUICC Engine configuration registers.

Workaround: Perform only 32-bit accesses for accessing QUICC Engine registers.

Fix plan: Fixed in Rev 1.1

QE_UART1: Overrun indication is not reported on the current BD

Description: The overrun indication is not reported on the current Rx buffer descriptor but on first buffer descriptor related to the received data after the overrun condition ended.

Impact: The user will not have any knowledge about the overrun event until the event ends and data is received. Not backward compatible with the PowerQUICC II family.

Workaround: Use a microcode patch.

Fix plan: No plans to fix

QE_UART2: DRT mode (UPSMR[DRT] = 1) is not functional

Description: Disable receive while in transmitting mode (UPSMR[DRT] = 1) is not functional.

Impact: DRT mode cannot be used.

Workaround: Use a microcode patch.

Fix plan: No plans to fix

QE_UART3: Simultaneous loopback and echo mode is not functional

Description: Simultaneous loopback and echo diagnostic mode (GUMR_L[DIAG] = 11) is not functional.

Impact: Issue only in diagnostic mode.

Workaround: Test echo mode and loopback mode separately.

Fix plan: No plans to fix

USB15: Read of PERIODICLISTBASE after successive writes may return a wrong value in host mode

Description: In the USB controller a new feature (hardware assist for device address setup) was introduced. This feature allows presetting of the device address in DEVICEADDR register before the device is enumerated, using a shadow register, to assist slow processors. The problem is that this mechanism, which is supposed to be functional only in device mode, is not blocked in host mode. DEVICEADDR register serves as PERIODICLISTBASE in host mode.

If PERIODICLISTBASE was set to some value, and later it is modified by software in such a way that bit 24 is set to 1, then wrong (previous) value is read back. However the USB controller will always read the correct value written in the register. ONLY the software initiated read-back operation will provide the wrong value.

Impact: No impact, if the software driver does not rely on the read-back value of the PERIODICLISTBASE register for its operation (practically there is no reason to do that).

Workaround: Write 0 twice to PERIODICLISTBASE before setting it to the desired value. This will clear the shadow register.

Fix plan: No plans to fix

USB19: USBDR in Host mode does not generate an interrupt upon detection of a CRC16/PID/Timeout error when a received IN data packet is corrupted

Description: USB dual role controller (USBDR) in Host mode does not generate an interrupt and does not set the respective bit in the USBSTS register upon detection of a CRC16/PID/Timeout error when a received IN data packet is corrupted. The error information, however, is written into the status field of the corresponding data structure and is not lost.

Impact: None

Workaround: The software driver should always check the status field of the transfer descriptor.

Fix plan: No plans to fix

USB21: SE0_NAK issue

Description: When put into SE0_NAK test mode in device configuration, the USB controller may occasionally miss an IN token (not respond with a NAK token), if it was issued exactly on 125 microsec micro-frame boundary, when SOF is expected in functional mode.

Impact: None

Workaround: None

Fix plan: No plans to fix

USB25: In host mode, when the software forces a port resume by writing into the FPR bit of the portsc register, the port change detect interrupt bit is falsely fired

Description: In host mode, a false "port change detect" interrupt is fired when the HCD (Host controller driver) resumes a suspended port by writing "1" to PORTSC[FPR] bit.

Impact: An interrupt is falsely fired when the software forces a port resume. There is an extra overhead to deal with the mis-fired interrupt.

Workaround: After setting PORTSC[FPR] and subsequent interrupt, the software should check the interrupt source, and clear USBSTS[PCI] bit, which corresponds to "port change detect" in Host mode.

Fix plan: No plans to fix

USB26: NackCnt field is not decremented when received NYET during FS/LS Bulk/ Interrupt mode

Description: The spec says that NakCnt should be decremented, whenever Host receives a NYET response to the Bulk CSPLIT and it should be reloaded when a start event is detected in the asynchronous list. This can happen in each micro-frame, or when the asynchronous schedule comes back from sleeping after an empty asynchronous schedule is detected.

The idea of the NAK counter is to keep trying until the counter reaches 0. When this happens, the controller just stops from issuing CSPLITS, until next micro-frame, where it reloads the counter and retries the CSPLIT again.

In the current implementation the controller does not decrement the NAK counter in this situation. If it receives a NYET, Host controller just advances to next Queue Head (QH) in the list and do not update the overlay area, later it will return and issue a new CSPLIT.

This could result in the host Controller thrashing memory, repeatedly fetching the queue head and executing transaction to the Hub, which will not complete until after the transaction on the classic bus completes.

The specification indicates that on receipt of a NYET handshake, the host controller should only adjust Cerr if it is the last CSPLIT transaction scheduled and halt the pipe when Cerr field transitions to zero. Since the Host controller hardware set the Cerr to the maximum value (3) every time it receives a NYET and stays in CSPLIT state so that the Cerr will never reach a zero value and hence the pipe will not be halted by the host controller.

Setting the Cerr to 3 every time it receives a NYET is a minor deviation from the specification. It will not cause a functional problem as a normal functioning hub. And it will eventually return something other than NYET and the transaction will complete.

The reclamation bit was being set to zero every time the host fetched the queue head with H bit set giving rise to empty list detection and the asynchronous list was not empty yet. This situation was leading the host to the asynchronous sleep state repeated times because the host was not paying any attention to NackCnt and RL.

Impact: The impact in the system is almost none. The Host will continuously do retries, instead of aborting when the NAK counter reaches zero. This may have a small penalty in the data bandwidth between Host and remaining attached Devices to the HUB.

Workaround: There is no direct software workaround, but the system software can cancel the transfer that is not reaching to the end after some time, and retry it later.

Fix plan: No plans to fix

USB27: When an ACK or NAK is sent from the device in response to a PING, the CERR counter value is not being reset to the initial value

Description: When the Controller is acting as a Host, the host state machine needs to update the ping status in response to a PING. There are two situations which are successful packet handshaking: ACK and NAK. In these two cases, the Controller should reset the CERR field to its initial value. As the CERR field is not updated, there can be a situation where the qTD will be halted by this value reaching 0. Under this condition, the qTD token will be retired, even though at least one PING packet was successfully responded with ACK or NAK.

Impact: Some PING packets are retried, even though at least one PING packet was successfully responded with ACK or NAK.

Workaround: A software workaround for this issue is possible. If a value of 0 is used in field CERR of the dTD when building the data structures, the controller will retry the transaction continuously, and will not be retired due to consecutive errors. This change actually increases the chance for the transaction to complete.

Fix plan: No plans to fix

USB28: In device mode, when receiving a Token OUT, if a Rx flush command is issued at the same time, to the same endpoint, the packet will be lost

Description: When receiving a Token OUT, the packet is lost if an Rx flush command is issued at the same time to the same endpoint. It reports ACK but the data is not copied to memory. Additionally, if the controller is in the stream disable mode (SDIS bit set a '1'), the endpoint is also blocked and NAKs any token sent to that endpoint forever. It is only applied to a USB device.

If the USB is configured as a peripheral, the controller normally does the flush when the software writes to the ENDPTFLUSH register of a Rx endpoint. The flush on the Rx buffer consists of DMA (drain side) reading all data remaining in the buffer until the Rx buffer is empty. The data is then thrown away.

If the Rx flush command is done while a packet is being received, the controller waits for the packet to finish and only then does the flush. As soon as the flush starts, the endpoint is unprimed, making the protocol engine (PE) NAK all incoming packets.

The protocol engine informs the endpoint control state machine that a packet has started by writing a TAG in the Rx buffer. As soon as the token is captured (knowing the endpoint and address), the protocol engine checks if the endpoint is primed or not. At this point, the protocol engine decides if the incoming packet will be ACKed or NAKed, even if the endpoint is unprimed during the packet reception.

The issue appears when the Rx flush command is set at the same time that the protocol engine writes the packet start TAG into the Rx buffer. At this moment, the endpoint control state machine does not have the packet because it has not read the packet start token yet. Thus, it starts the Rx flush sequence. At the same time by writing the packet start TAG into the Rx buffer, the endpoint is primed, so the protocol engine ACKs the packet at the end.

In this situation, the endpoint control state machine reads the Rx buffer at the same rate as the protocol engine writes the incoming data because all data is ignored. The state machine empties the Rx buffer and unprimes the endpoint. But the protocol engine ACKs the packet anyway because it only cares about prime when receiving an incoming token.

At the end of the packet, the protocol engine replies with ACK and writes the end of packet TAG into the Rx buffer. The endpoint is blocked because the protocol engine blocks it at the end of a non-setup transaction. The endpoint is unblocked by the endpoint control state machine as soon as all the data has been transfered into the system memory. As the endpoint control state machine never saw the start of packet, it never unblocks the endpoint. In stream disable mode, the only way to unblock the endpoint in this scenario is to switch to streaming mode.

Impact: Rx flush should not be used to clear an endpoint when acting as peripheral. When receiving a Token OUT, if an Rx flush command is issued at the same time to the same endpoint, the packet will be lost. For stream disable mode, the only way to unblock the endpoint in this scenario is switch to streaming mode.

Workaround: Do not use Rx flush feature when acting as peripheral.

Fix plan: No plans to fix

USB29: Priming ISO over SOF will cause transmitting bad packet with correct CRC

Description: In device mode, a priming operation performed by software while an IN token is being received (usually just after the SOF) can lead to an invalid transfer in the next frame or an abortion of a transfer that should not be canceled. This can only happen for Isochronous IN transfers.

This scenario can happen when the Device controller receives an IN token from the host and the protocol engine has not been primed yet, but the software has already performed the priming operation. This can occur due to latency between the setting of the prime bit for the specific endpoint and the exact moment when the protocol engine recognizes that it has been primed. The root cause is that in the current implementation, the priming operation inside the protocol engine only occurs at the time of SOF reception.

Impact: Two problems can arise as a result of this issue:

- The data that is being written into the Tx RAM is read while the device sends a zero length packet (since it is not primed). Therefore, the data cannot be transferred to the next frame, and a short packet is sent the next time the host asks for data.
- After sending the zero length packet, the protocol engine informs the DMA state machine that the transfer has been completed, so the respective dTD is updated. This also causes an error because the total number of bytes transferred is not equal to zero. Therefore, this transfer is also lost.

Workaround: There is no workaround.

Fix plan: No plans to fix

USB31: Transmit data loss based on bus latency

Description: When acting as a Device, after receiving a Token IN, the USB controller will reply with a data packet. If the bus memory access is not fast enough to backfill the TX fifo, it will cause an under-run. In this situation a CRC error will be introduced in the packet and the Host will ignore it. However, when an underrun happens, the TX fifo will get a flush command. This situation may cause an inconsistence in the TX fifo controls, leading to a possible data loss (a complete packet or sections of a packet can be never transmitted). This situation may also happen if the software issues a TX flush command.

Impact: When the USB controller is configured as a device, it can not be used in the stream mode due to this erratum. Therefore, the USB external bus utilization is decreased.

Workaround: A valid software workaround is to disable the stream mode by setting USBMODE[SDIS] bit. This can avoid the issue at the expense of decreased USB external bus utilization.

Fix plan: No plans to fix

USB32: Missing SOFs and false babble error due to Rx FIFO overflow

Description: When in Host mode, if an Rx FIFO overflow happens close to the next Start-of-Frame (SOF) token and the system bus (CSB) is not available, a false frame babble is reported to software and the port is halted by hardware. If one SOF is missed, the Host controller will issue false babble detection and SOFs will no longer be sent. If more than 3.125 ms are elapsed without SOFs, the peripheral will recognize the idle bus as a USB reset.

Impact: If this scenario occurs, it will degrade performance and have system implications. The Host will have to reset the bus and re-enumerate the connected device(s).

Workaround: Reset the port, do not disable the port, on which the babble is detected.

Fix plan: No plans to fix

USB33: No error interrupt and no status will be generated due to ISO mult3 fulfillment error

Description: When using ISO IN endpoints with MULT = 3 and low bandwidth system bus access, the controller may enter into a wait loop situation without warning the software. Due to the low bandwidth, the last packet from a mult3 sequence may not be fetched in time before the last token IN is received for that microframe/endpoint.

Impact: This will cause the controller to reply with a zero length packet (ZLP), thus breaking the prime sequence. The DMA state machine will not be warned of this situation and the controller will send a ZLP to all the following IN tokens for that endpoint. The transaction will not be completed because the DMA state machine will be waiting for the unprimed TX complete command to come from the Protocol Engine.

Workaround: If this scenario occurs, use MULT = 2.

Fix plan: No plans to fix

USB34: NAK counter decremented after receiving a NYET from device

Description: When in host mode, after receiving a NYET to an OUT Token, the NAK counter is decremented when it should not.

Impact: The NAK counter may be lower than expected.

Workaround: None

Fix plan: No plans to fix

USB35: Core device fails when it receives two OUT transactions in a short time

Description: In the case where the Controller is configured as a device and the Host sends two consecutive ISO OUT (example sequence: OUT - DATA0 - OUT - DATA1) transactions with a short inter-packet delay between DATA0 and the second OUT (less than 200 ns), the device will see the DATA1 packet as a short-packet even if it is correctly formed. This will terminate the transfer from the device's point -of-view, generating an IOC interrupt. However, DATA0 is correctly received.

Impact: If this scenario occurs, the clear command from the protocol engine state machine to the protocol engine data path is not sent (and internal byte count in the data path module is not cleared). This causes a short packet to be reported to the DMA engine, which finishes the transfer and the current dTD is retired.

Workaround: None

Fix plan: No plans to fix

USB36: CRC not inverted when host under-runs on OUT transactions

Description: In systems with high latency, the HOST can under-run on OUT transactions. In this situation, it is expected that the CRC of the truncated data packet to be the inverted (complemented), signaling an under-run situation.

Impact: Due to this erratum, the controller will not send this inverted CRC. Instead, it sends only one byte of the inverted CRC and the last byte of payload.

It is unlikely but remotely possible that this sent CRC to be correct for the truncated data packet and the device accepts the truncated packet from the host.

Workaround: Setting bigger threshold on TXFILLTUNING[TXFIFOTHRES] register might solve the under-run possibility and thus avoiding truncated packets without the expected inverted CRC.

However, this would not solve the inverted CRC issue by itself.

Fix plan: No plans to fix

USB37: OTG Controller as Host does not support Data-line Pulsing Session Request Protocol

Description: An OTG core as a Host must be able to support at least one Session Request Protocol (SRP) method (VBUS or Data-line Pulsing), but OTG as Device must support and use both when attempting SRP.

As our OTG controller as a Host fully supports VBUS pulsing, the SRP will always be successful and the impact of this issue is minor. However, the recent OTG 2.0 specification removes the VBUS pulsing SRP method, making the Data-line Pulsing a mandatory SRP detection for host controllers.

Impact: When the OTG core is acting as a Host, and VBUS is turned off, and the attached Device attempts to perform a Session Request Protocol (SRP) by using Data-line Pulsing, it will not be recognized by the Host.

Also, when doing role switching (HNP) and becoming a Host, a SE0 is forced in the line causing the OPT TD5.4 test to fail.

Workaround: The termsel will be changed from '0' to '1' when in reset and in the state after reset (PORT_DISABLE). As this signal is the same if the controller is host or device, the termsel was changed in both cases.

Software workaround is possible for the HNP situation only. With this workaround, it is possible to pass the OPT TD5.4 test. The software must assert core mode to device (USBMODE.CM) and Run/Stop bit (USBCMD.RS) to '1' just after the controller ends reset (wait until USBCMD.RST is '0' after setting it to '1').

This issue does not prevent OTG 1.3 SRP, since it is possible to do VBUS pulsing. This correction extends to OTG 2.0 support, since Data-line Pulsing is mandatory.

Fix plan: No plans to fix

A-003817: USB Controller locks after Test mode "Test_K" is completed

Affects: USB

Description: Previously known as USB38

When using the ULPI interface, after finishing test mode "Test_K," the controller hangs. A reset needs to be applied.

Impact: No impact if reset is issued after "Test K" procedure (it should be issued according to the standard).

Workaround: None

Fix plan: No plans to fix

USB-A001: Last read of the current dTD done after USB interrupt

Description: After executing a dTD, the device controller executes a final read of the dTD terminate bit. This is done in order to verify if another dTD has been added to the linked list by software right at the last moment.

It was found that the last read of the current dTD is being performed after the interrupt was issued. This causes a potential race condition between this final dTD read and the interrupt handling routine servicing the interrupt on complete which may result in the software freeing the data structure memory location, prior to the last dTD read being completed. This issue is only applied to a USB device controller.

Impact: Two different situations may occur:

- The case of a single dTD with the Interrupt on Completion (IOC) bit set. In this case, if the interrupt handling routine has a lower latency than the bus arbitration of the dTD read after the interrupt is posted (at this time the software will find the Active bit cleared - dTD retired by hardware), the software may clear or re-allocate the data structure memory location to other applications, before the last read is performed.
- The case of multiple dTDs with the IOC bit set. In this case, if the latency handling the interrupt is too long, the following might occur:
  a. The core could assert the interrupt when it completes dTD1.
  b. Proceed to execute the transfer for dTD2.
  c. By the time the core has just updated dTD2 Active field to inactive, the interrupt handling routine finishes processing the interrupt for dTD1, checks dTD2 and finds that it has completed and so re-allocates both data structures.
  d. The core then re-reads dTD2 and finds corrupted data. If the T bit is zero or the Active field in non active then the core will not re-prime.

Workaround: None

Fix plan: No plans to fix

USB-A002: Device does not respond to INs after receiving corrupted handshake from previous IN transaction

Description: When configured as a device, a USB controller does not respond to subsequent IN tokens from the host after receiving a corrupted ACK to an IN transaction. This issue only occurs under the following two conditions:

1. An IN transaction after the corrupted ACK
2. The time gap between two IN tokens are smaller than the bus time out (BTO) timer of the USB device controller

Under this case, for every IN token that arrives, the bus timeout counter is reset and never reaches 0 and a BTO is never signaled. Otherwise, the USB device times out and the device controller goes to an idle state where it can start to respond normally to subsequent tokens. .

Impact: There are two cases to consider:

1. CERR of the Queue Element Transfer Descriptor (qTD) is initialized to a non-zero value by the host driver

   This is what happens in the majority of use cases. The maximum value for CERR is 3. A host driver is signaled/interrupted after CERR is decremented to 0 due to the transaction errors. In this case, the host driver has to process the transaction errors. Most likely, the host driver will reset this device. Furthermore, the BTO timer of the USB device could time out due to time needed for the USB host to process the transaction errors.

2. CERR of the qTD is initialized to zero by the host driver

   In this case, the host driver does not process any transaction error. However, based on USB 2.0/EHCI specification, the host controller is required to maintain the frame integrity, which means that the last transaction has to be completed by the EOF1 (End Of Frame) point within a (micro) frame. Normally, there should be enough time for a USB device to time out.

Workaround:

1. CERR of the qTD is initialized to a non-zero value

   No workaround is needed since the USB host driver will halt the pipe and process the transaction errors

2. CERR of the qTD is initialized to zero and if
   a. The USB controller is configured as a full/low-speed device.

      No workaround is needed since USB 2.0 specification requires a longer idle time before a SOF (Start of Frame) than the BTO timer value. The USB device times out at the end of the next (micro) frame at most.

   b. The USB controller is configured as a high-speed device.

      No workaround is needed if the data length of the next transaction after the corrupted ACK is 32 bytes or longer. The USB device times out at the end of the next (micro) frame at most.

   c. The USB controller is configured as a high-speed device.

      No workaround is needed as long as the Host_delay in the USB host system is 0.6 us or longer. The USB device times out at the end of the next (micro) frame at most.

   d. The USB controller is configured as a high-speed device.

A workaround is needed if the data length of the next transaction after the corrupted ACK is less than 32 bytes and the Host_delay in the USB host system is less than 0.6 us. Under this condition, a transfer in a device controller does not progress and the total bytes field in the dTD (device transfer descriptor) remains static. The software on a device controller could implement a timer routine for an IN endpoint to monitor whether a transfer is progressing. If it is not, the timer routine can cancel the transfer and reset the device by setting the USBCMD[RST] bit. However, this is a rare case. No such case has been reported.

Fix plan: No plans to fix

USB-A003: Illegal NOPID TX CMD issued by USB controller with ULPI interface

Description: During the USB reset process (speed negotiation and chirp), if the protocol engine sends Start of Frame (SOF) commands to the port control, the port control filters out those SOFs. However, at the end of reset (end of chirp back from Host), when the protocol engine sends a SOF, the ULPI port control sends the SOF to the PHY before sending the update OpMode command. This results in an invalid packet being sent on the line. The invalid packet in ULPI protocol is a NOPID transmit command immediately followed by a STP pulse. This failure happens less than 0.1% of time.

Impact: Due to this erratum, some ULPI PHY's lock up and do not accept any additional data from USB host controller. As PHY is locked up, there cannot be any communication on the USB Interface.

Workaround: Do not enable USBCMD[RS] for 300 uSec after the USB reset has been completed (after PORTSCx[PR] reset to 0). This ensures that the host does not send the SOF until the ULPI post reset processing has been completed.

Fix plan: No plans to fix

USB-A005: ULPI Viewport not Working for Read or Write Commands With Extended Address

Description: It is not possible to read or write the ULPI PHY extended register set (address >0x3F) using the ULPI viewport.

The write operation writes the address itself as data, and a read operation returns corrupted data. A Controller lock up is not expected, but there is no feedback on this failed register access.

Impact: Read or Write Commands With Extended Address does not work through ULPI Viewport register.

Workaround: None

Fix plan: No plans to fix

USB-A006: READ operation on some USB registers returns an incorrect value.

Description: When the USB registers in the address space 0x100-0x1C8 with respect to the USB Base address are read, an incorrect value is returned. This happens for those read operations that require more than one bus clock cycle. This is because of incorrect handling of an internal wait signal to allow for the completion of the Read operation. However, a second read operation following the first read will return a correct value.

Impact: Not able to read the USB registers (0x100 -01C8) in single read operation.

Workaround: Software must perform two consecutive read operations for the affected USB registers as follows

a. Read register(0x100 as an example)

b. Incorrect value (discard the value)

c. Read again 0x100 (no read should be done in between)

d. Returns 0x40000001 (Correct value)

Double read has to be done for USB registers from 0x100 -0x1C8. Please note that the two reads must be done atomically without allowing for an interrupt handling between them.

Fix plan: No plans to fix.

USB-A007: Host controller fails to enter the PING state on timeout during High Speed Bulk OUT/DATA transaction

Description: For High-speed bulk and control endpoints, a host controller queries the high-speed device endpoint with a special PING token to determine whether the device has sufficient space for the next OUT transaction. The mechanism avoids using bus time to send data until the host controller knows that the endpoint has space for the data.

If a timeout occurs after the data phase of an OUT transaction, the host controller should return to using a special PING token. However, due to this erratum, the host controller fails to enter the PING state and instead retries the OUT token again.

Impact: The PING flow control for the high-speed devices does not work under this condition. Therefore, some USB bandwidth could be wasted. However, a timeout response to Out/Data or PING transactions is an unexpected event and should only occur if the device has detected an error and so should be rare.

Workaround: None

Fix plan: No plans to fix

A-003827: DATA PID error interrupt issued twice for the same high bandwidth ISO transfer

Affects: USB

Description: When receiving an Isochronous OUT transfer for a High Bandwidth endpoint (MULT > 0), if one of the DATA PIDs is corrupted, the controller issues two interrupts for that transaction error, one in the current microframe to signal the DATA PID error, and one fulfillment error in the next microframe.

On the beginning of a new microframe (upon receiving a SOF), the DMA checks and reports the status of the ISO transfer completed in the previous microframe. If the number of data packets correctly received for an ISO RX transfer is greater than 0 but less than MULT, the controller issues a fulfillment error by setting the transaction error bit. When a DATA PID error occurs, this error interrupt is triggered.

Both the bad PID and fulfillment error set the transaction error bit in the dTD. This is an ambiguous situation for the application that may then stop the endpoint from receving valid data in the next microframe (if there is another one from the Host).

Impact: This issue results in a correct ISO data transaction getting discarded.

There is no impact for the application software because data delivery in ISO transfers is not guaranteed. If there is a data delivery failure because of errors, no retries are attempted.

In ISO transfers, there is no dependency of the data between data packets and if a transaction is discarded due to a transaction error, the ISO stream can continue to the next dTD. This is transparent for the application, as the application is required to be capable of handling transaction errors in ISO streams.

The only impact of this issue is that the application discards not only the data transaction for which the DATA PID error occurred but also the next transaction.

Workaround: None

Fix plan: No plans to fix

A-003829: Host detects frame babble but does not halt the port or generate an interrupt

Affects: USB

Description: A high speed ISO Device, connected downstream to a high speed hub connected to the USB host, babbled in to the uframe boundary EOF1 time and the hub disabled the propagation of traffic to the upstream root host.

Inside the host controller, the ehci_ctrl state machine issues a request to the protocol engine to initiate the next transaction but this transaction is not sent to the USB as the port enable bit has been cleared. The result is that the ehci_crl state machine waits for the transaction to complete (which does not occur).

Eventually the software application times out without any frame babble error information. Just the iTD transaction error is issued.

The failure was seen with a high speed ISO device but a device babble error could occur on bulk or control or interrupt transactions for a failing device.

The final state is that the port has transitioned to full speed and the port enable bit is deasserted while the DMA does not know that there is a problem and the data structure shows only the occurrence of a transaction error.

This bug can occur for host IN transaction where the sending device under runs and error injects on the data packet. In this case the host may retry the IN immediately and it does not consider that the protocol engine may not be ready to issue the IN to the USB. The protocol engine issues the IN up to 5 useconds later than the EHCI Control state machine has issued the request to send the packet so it could result in a frame babble.

Impact: The host port is disabled, the halt bit is set, and the frame babble error is set in the associated data structure. This behavior complies with the EHCI specification for handling a frame babble error. The host controller driver handles the recovery by clearing the error conditions and re-queuing the transfer which should occur normally. When a frame babble occurs, there may be a loss of bandwidth because the application has to intervene and re-queue the transfer and re-enable the port.

Workaround: There is no workaround because the control of the retry is under hardware control so for non ISO transfers the hardware will retry so long as it determines that there is enough time but it does not account for the added delay due to the host protocol state machine being in the bus timeout state. Having a large TX FIFO and a good fill level (TXFIFOTHRES) will mean that there will be no under runs to host OUT transactions. This will significantly reduce the probability of occurrence of this issue for OUT Transactions. However please note that this bug can occur for host IN transaction where the sending device under runs and error injects on the data packet. In this case the host may retry the IN immediately.

Recovery:

The host will not get any response. The recovery from this condition will depend on the software, but host it will eventually time out and reset the device. This is not critical as this issue will only occur if the device downstream of a HS HUB is out of spec. and generates a frame babble.

Fix plan: No plans to fix

A-003837: When operating in test mode, the CSC bit does not get set to 1 to indicate a change on CCS

Affects: USB

Description: When in test mode (PORTSCx[PTC] != 0000), the Connect Status Change bit (PORSTCx[CSC]) does not get set to 1 to indicate a change in Current Connect Status (PORTSCx[CCS]).

Impact: This only affects the compliance test mode. There is no functional impact.

Workaround: Perform software polling for changes in the CCS bit (instead of using CSC) when test mode is enabled.

Fix plan: No plans to fix

A-003845: Frame scheduling robustness-Host may issue token too close to uframe boundary

Affects: USB

Description: When the USB host encounters an under-run while sending a Bulk OUT packet, it issues a CRC error according to the specification. However, the retry never occurs on the USB and the host appears to hang; it does not send any further transactions including SOF packets. The device ultimately detects a suspend condition and defaults to full speed mode. This can also happen for IN transactions where the device encountered an under-run and sent BAD CRC. The host will retry in this case without checking for the time left in the current uFrame. The response from the device will cause frame babble in this case.

Impact: The host appears to be hang as it does not send any further packets.

System Impact: The host port is disabled, the halt bit is set, and the frame babble error is set in the associated data structure. This behavior complies with the EHCI specification for handling a frame babble error. The host controller driver handles the recovery by clearing the error conditions and re-queuing the transfer which should occur normally. When a frame babble occurs, there may be a loss of bandwidth because the application has to intervene and re-queue the transfer and re-enable the port.

Workaround: For OUT transactions: If the host controller TX under-runs can be avoided then the problem will not occur for OUT transaction. Using a larger value for TXSCHOH can avoid this issue for OUT transactions.

For IN transactions: Insure the USB device side does not into an under-run condition. There is no workaround from the host side. The host controller driver (software) should handle the recovery by clearing the error conditions and re-queuing the transfer which should occur normally and re-enabling the port. The software driver can get the system restarted in this case. However, it cannot prevent the frame babble from occurring.

Fix plan: No plans to fix

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.