MPC8379E Chip Errata

This document details all known silicon errata for the MPC8379E PowerQUICC II Pro family of integrated host processors. This document also applies to the MPC8378E and MPC8377E, as noted in each erratum description. The following table provides a revision history for this document.

Table 1. Revision history

Revision	Date	Significant changes
7	06/2014	Added eTSEC A-007207 Modified eTSEC69 Renamed eTSEC-A002 to A-006502
6	09/2013	Added the following errata: • SATA A-005255, A-005636, and A-006187 • USB A-003829 and A-003845 Modified the following errata: • eSDHC5 • IEEE1588-A001 Updated the Silicon Revision from 2.0 to 2.1 in Table 3, "Summary of Silicon Errata and Applicable Revision"
5	11/2012	• Renamed USB38 to USB erratum A-003817 • Added eSDHC errata eSDHC23, A-004577, A-005055 A-005286 and A-005287; SATA A-005035; and USB erratum A-003837. • Modified eSDHC16 workaround.
4	08/2011	• Added CPU-A022, USB-A005, USB-A007, eTSEC79 • Updated impact for SEC-A001 • Updated work around for PCI20
3	12/2010	• Added PCIe-A002, SEC-A001, USB-A003 • Updated eTSEC-A002, IEEE1588-A001, PEX7, SATA2, SATA-A002, USB-A002 • Updated work around for eSDHC19, USB32
2	05/2010	• Updated workaround for eTSEC20, eTSEC76 • Updated wording for eLBC2, eLBC5, eTSEC13, eTSEC16, SEC14 • Added eLBC-A001, eSDHC-A001, eTSEC78, eTSEC-A001, eTSEC-A002, IEEE1588-A001, PEX9, SATA-A002, USB-A001, USB-A002
1	07/2009	• Added General16, IEEE_1588_21, CPU6, eSDHC17-eSDHC20, USB32-USB38 eTSEC76-eTSEC77

Table continues on the next page...

Table 1. Revision history (continued)

Revision	Date	Significant changes
		• Updated General17 • Renamed eTSEC57 as eTSEC13.
0	01/2009	Initial public release.

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 2. MPC8379E Family Devices

	Mask (Silicon Rev 2.1)
MPC8379E	W01M68P
MPC8378E	
MPC8377E	

Table 3 summarizes all known errata and lists the corresponding silicon revision level to which they apply. A 'Yes' entry indicates the erratum applies to a particular revision level, and a 'No' entry means it does not apply.

Table 3. Summary of Silicon Errata and Applicable Revision

Errata	Name	Projected Solution	Silicon Rev.	
			1.1	2.1
ARB				
ARB3	Changing the value of ACR[PIPE_DEP] can cause the arbiter to generate false data time out	No plans to fix	Yes	Yes
CPU				
CPU6	DTLB LRU logic does not function correctly	No plans to fix	Yes	Yes
CPU-A022	The e300 core may hang while using critical interrupt	No plans to fix	Yes	Yes
DMA				
DMA2	Data corruption by DMA when destination address hold (DAHE) bit is used	No plans to fix	Yes	Yes
eLBC				
eLBC1	LTESR[CS] error issue	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
eLBC3	eLBC NAND Flash memory has an ECC syndrome that collides with the JFFS2 marker in Linux	No plans to fix	Yes	Yes
eLBC4	$\overline{\text{LGTA}}$/LUPWAIT assertion in PLL-bypass mode misrepresented	No plans to fix	Yes	Yes
eLBC5	LTEATR and LTEAR may show incorrect values under certain scenarios	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.	
			1.1	2.1
eLBC-A001	Simultaneous FCM and GPCM or UPM operation may erroneously trigger bus monitor timeout	No plans to fix	Yes	Yes
eSDHC				
eSDHC1	Data End Bit Error (DEBE, bit 22 in the interrupt status register) incorrectly set if Card INTerrupt is driven in SDIO 1 bit mode	No plans to fix	Yes	Yes
eSDHC2	Read/Write Multiple command when block count is 1	No plans to fix	Yes	Yes
eSDHC3	Force Event Register	No plans to fix	Yes	Yes
eSDHC4	CPU polling read mode when the burst length is one word	No plans to fix	Yes	Yes
eSDHC5	eSDHC reads more data than expected from the system bus	No plans to fix	Yes	Yes
eSDHC6	eSDHC indicates AUTO CMD12 interrupt later than expected	No plans to fix	Yes	Yes
eSDHC7	Data corruption during write with pause operation	No plans to fix	Yes	Yes
eSDHC8	Cannot initiate non-data command while data transfer is in progress	No plans to fix	Yes	Yes
eSDHC10	Incorrect data is written to a card after transfer paused	No plans to fix	Yes	Yes
eSDHC11	eSDHC Multiple Block Read Failures	No plans to fix	Yes	Yes
eSDHC12	During multi-write operation, unexpected Transfer Complete (IRQSTAT[TC] register) bit can be set	No plans to fix	Yes	Yes
eSDHC13	CRC might be corrupted for write data transfer if it is preceded by read transfer	No plans to fix	Yes	Yes
eSDHC14	Data End Bit Error (DEBE, bit 22 in the interrupt status register) incorrectly set if Card Interrupt is driven in SDIO 4-bit mode	No plans to fix	Yes	Yes
eSDHC15	Unable to issue CMD12 if SD clock shuts off due to slow system access	No plans to fix	Yes	Yes
eSDHC16	Manual Asynchronous CMD12 abort operation causes protocol violations	No plans to fix	Yes	Yes
eSDHC17	PRSSTAT[CIDHB] is not reliable for commands with busy (R1b)	No plans to fix	Yes	Yes
eSDHC18	Invalid generation of Block Gap Event	No plans to fix	Yes	Yes
eSDHC19	eSDHC cannot finish write operation after continuing from Block Gap Stop	No plans to fix	Yes	Yes
eSDHC20	Corrupted data read from eSDHC for certain values of WML[RD_WML] and BLKATTR[BLKSIZE]	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-A001	Data timeout counter (SYSCTL[DTOCV]) is not reliable for values of 0x4, 0x8, and 0xC	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-005286	CPU polling read mode when the burst length is one word	No plans to fix	Yes	Yes
A-005287	Read multiple block failure	No plans to fix	Yes	Yes
eTSEC				

Table continues on the next page...

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

Errata	Name	Projected Solution	Silicon Rev.	
			1.1	2.1
eTSEC11	VLAN extraction with shim header not supported	No plans to fix	Yes	Yes
eTSEC13	Fetches with errors not flagged, may cause livelock or false halt	No plans to fix	Yes	Yes
eTSEC16	Parsing of tunneled IP packets not supported	No plans to fix	Yes	Yes
eTSEC42	Frame is dropped with collision and HALFDUP[Excess Defer] = 0	No plans to fix	Yes	Yes
eTSEC45	eTSEC parser does not perform length integrity checks	No plans to fix	Yes	Yes
eTSEC46	eTSEC does not verify IPv6 routing header type field	No plans to fix	Yes	Yes
eTSEC48	L4 info passed to filer in L2/L3-only mode	No plans to fix	Yes	Yes
eTSEC58	VLAN Insertion corrupts frame if user-defined Tx preamble enabled	No plans to fix	Yes	Yes
eTSEC64	User-defined Tx preamble incompatible with Tx Checksum	No plans to fix	Yes	Yes
eTSEC67	ECNTRL[AUTOZ] not guaranteed if reading MIB counters with software	No plans to fix	Yes	Yes
eTSEC69	Magic packet sequence embedded in partial sequence not recognized	Partially fixed in Rev 2.0	Yes	Yes
eTSEC72	Receive pause frame with PTV = 0 does not resume transmission	No plans to fix	Yes	Yes
eTSEC73	TxBD polling loop latency is 1024 bit-times instead of 512	No plans to fix	No	Yes
eTSEC74	MAC: Rx frames of length MAXFRM or MAXFRM-1 are marked as truncated	No plans to fix	Yes	Yes
eTSEC75	Misfiled Packets Due to Incorrect Rx Filer Set Mask Rollback	No plans to fix	Yes	Yes
eTSEC76	Excess delays when transmitting TOE=1 large frames	No plans to fix	Yes	Yes
eTSEC77	SGMII receiver loss of signal threshold marginality	No plans to fix	Yes	Yes
eTSEC78	Controller may not be able to transmit pause frame during pause state	No plans to fix	Yes	Yes
eTSEC79	Data corruption may occur in SGMII mode	No plans to fix	Yes	Yes
eTSEC-A001	MAC: Pause time may be shorter than specified if transmit in progress	No plans to fix	Yes	Yes
A-006502	Incomplete GRS or invalid parser state after receiving a 1- or 2-byte frame	No plans to fix	Yes	Yes
A-007207	TBI link status bit may stay up after SGMII electrical idle is detected	No plans to fix	Yes	Yes
IEEE1588				
IEEE 1588_8	Writing Offset registers during use may yield unpredictable results	No plans to fix	Yes	Yes
IEEE 1588_12	1588 alarm fires when programmed to less than current time	No plans to fix	Yes	Yes
IEEE 1588_14	TxPAL timestamp uses TxBD snoop enable instead of Tx data	No plans to fix	Yes	Yes
IEEE 1588_16	Odd prescale values not supported	No plans to fix	Yes	Yes
IEEE 1588_19	Tx FIFO data parity error (DPE) may corrupt Tx timestamps if TMR_CTRL[TRTPE]=1	No plans to fix	No	Yes
IEEE 1588_21	IEEE 1588 accuracy can be adversely impacted in systems using multiple unsynchronized gigabit Ethernet ports	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.	
			1.1	2.1
IEEE1588-A001	Incorrect received timestamp or dropped/data corruption packet when 1588 time-stamping is enabled	No plans to fix	Yes	Yes
General				
General16	Enabling I^{2}C could cause I^{2}C bus freeze when other I^{2}C 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
General18	Machine Model (MM) Electrostatic Discharge (ESD) criteria not met by certain SerDes pins	No plans to fix	Yes	Yes
JTAG				
JTAG6	Boundary scan test on SerDes pins unreliable	No plans to fix	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
PCI20	PCI controller may hang when returning from "PCI pins low" state	No plans to fix	Yes	Yes
PCIe				
PEX1	No support of PCI Express completions with BCM bit set (PCIX bridge interface)	No plans to fix	Yes	Yes
PEX2	DMA Interrupt descriptor race condition (IDRC)	No plans to fix	Yes	Yes
PEX7	Recovery from hot reset or link down	No plans to fix	Yes	Yes
PEX9	Excess correctable errors in receiving DLLPs and TLPs on x2 link	No plans to fix	Yes	Yes
PCIe-A002	PCI Express Packets may be transmitted with excess data on x1 link	No plans to fix	Yes	Yes
RESET				
RESET3	External Soft reset functionality is not functional	No plans to fix	Yes	Yes
SATA				
SATA2	Reads of one sector from hard disk may return less data than requested	No plans to fix	Yes	Yes
SATA3	SATA controller hangs when handling data integrity errors	No plans to fix	Yes	Yes
SATA9	Fails the SATA InterOperability Sinusoidal Jitter test	No plans to fix	Yes	Yes
SATA12	SATA: DMAT handling in SATA not as expected	No plans to fix	Yes	Yes
SATA13	BIST-L mode is not enabled by BIST-L FIS	No plans to fix	Yes	Yes
SATA-A002	ATAPI commands may fail to complete	No plans to fix	Yes	Yes
A-005035	Possible data loss if PRD[DBA] or PRD[DWC] is not at least 16-byte aligned	No plans to fix	Yes	Yes
A-005255	Failure to Detect Single SYNC Primitive	No plans to fix	Yes	Yes
A-005636	Auto-activate feature enabled in DMA setup command causes timeout	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.	
			1.1	2.1
A-006187	Intermittent, non-recoverable, transient data integrity error seen when accessing SSC-enabled SATA drives	No plans to fix	Yes	Yes
SEC				
SEC9	Protocol descriptor hang conditions	No plans to fix	Yes	Yes
SEC10	AES-GCM IV Length Restriction	No plans to fix	Yes	Yes
SEC11	Performance counter register access requirement	No plans to fix	Yes	Yes
SEC12	TLS_SSL_Block_Inbound HMAC Error	No plans to fix	Yes	Yes
SEC14	Non-compliant implementation of deterministic pseudo-random number generator	No plans to fix	Yes	Yes
SEC15	Limitations on Custom Modes of CRC	No plans to fix	Yes	Yes
SEC16	Kasumi hardware ICV checking does not work	No plans to fix	Yes	Yes
SEC-A001	Channel Hang with Zero Length Data	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
USB37	OTG Controller as Host does not support Data-line Pulsing Session Request Protocol	No plans to fix	Yes	Yes
USB-A001	Last read of the current dTD done after USB interrupt	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.	
			1.1	2.1
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-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-003817	USB Controller locks after Test mode "Test_K" is completed	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

ARB3: Changing the value of ACR[PIPE_DEP] can cause the arbiter to generate false data time out

Description: Devices: MPC8379E, MPC8378E, MPC8377E

If the pipeline depth is changed to any value other than zero by writing to ACR[PIPE_DEP], and the next transaction is acknowledged or retried at the same cycle that the actual pipe depth value change is registered in the arbiter logic, the arbiter internal logic will be broken.

Impact: The arbiter generates a false data time out.

Workaround: If the system is configured by the e300 CPU:

- Make sure no other masters are active when pipe depth is configured.
- Add a loop of 100 NOP instructions following the store to ACR. In order to avoid an instruction fetch after the ACR write, make sure the instruction cache is on and that the ACR write and the loop of 100 NOPs are in the same cache line.This will ensure that no instruction is fetched after the ACR write and that, therefore, CSB is idle while the pipe depth is being updated.

For PCI agent mode, use an external PCI host to configure the system. Hold the e300 in core disable mode by setting the CORE_DIS bit in RCWH. Set ACR[PIPE_DEP] to the desired value and then wait for at least 1 μs before issuing further transactions to the device. At this point the e300 core can be enabled to fetch its boot code by clearing ACR[CORE_DIS].

If possible, use the I^2C boot sequencer to configure initial settings of the system, including ACR[PIPE_DEP].

Fix plan: No plans to fix

CPU6: DTLB LRU logic does not function correctly

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

DMA2: Data corruption by DMA when destination address hold (DAHE) bit is used

Description: Devices: MPC8379E, MPC8378E, MPC8377E

There can be corruption of the DMA data under the following conditions:

- DMAMR[DAHE] = 1 (destination address hold)
- DMAMR[DAHTS] = 10 (4 bytes) or 11 (8 bytes)
- DMA source address is not aligned to the transaction size specified by DAHTS
- The source port width is smaller than the destination transaction size or the source port returns valid read data only in the valid byte lanes

Examples of error condition are as follows:

- DAHTS is 8 bytes and the source port is a 32-bit PCI bus
- The source memory space is on the PCI bus and is not prefetchable

Impact: Corrupted data written to the destination peripheral or memory.

Workaround: Use one of the following options:

- Use a source address aligned to the destination transaction size
- Do not access any DMA registers while this type of DMA transfer is active

Fix plan: No plans to fix

eLBC1: LTESR[CS] error issue

Description: Devices: MPC8379E, MPC8378E, MPC8377E

LSOR is used as a special operation by reporting chip select errors in LTESR[CS]. This special operation initiates a correct response as expected by the OP field. For example, when OP = 01, that is, the UPM RAM Array is to be written and LSOR is written, then correct data from MDR is written into the UPM RAM word, but this gives a chip select error. This works fine when JTAG is used as the master but gives a CS error when the core is the master. As an alternative, if a dummy write is used instead of a write to LSOR, it does not report a LTESR[CS] error.

Impact: A write to LSOR performs the correct intended operation, but this causes an LTESR[CS] error. So, software will report error if it is monitoring the LTESR register.

Workaround: A dummy transaction can be used as an alternative to writing to LSOR, which performs the same operation.

Fix plan: No plans to fix

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

eLBC3: eLBC NAND Flash memory has an ECC syndrome that collides with the JFFS2 marker in Linux

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The current NAND Flash memory ECC location collides with the JFFS2 marker in Linux. There is no industry specification about where the ECC syndrome should be placed.

Impact: Cannot boot from NAND flash if flash includes Linux JFFS2 markers.

Workaround: Disable hardware ECC and use software ECC instead. But hardware ECC is enabled by default during 4K boot phase and can only be disabled once boot is over for remaining data transfers. If the software can handle this, by using hardware ECC for the 4K boot block and software ECC for the rest of the NAND Flash, this workaround would work well.

Fix plan: No plans to fix

eLBC4: $\overline{\text{LGTA}}$/LUPWAIT assertion in PLL-bypass mode misrepresented

Description: Devices: MPC8379E, MPC8378E, MPC8377E

In the Local Bus section of the Hardware Specification document, the timing diagram figures for PLL-bypass mode show that the $\overline{\text{LGTA}}$/LUPWAIT signal is latched on the falling edge of the internal launch/capture clock signal. This is a misrepresentation of the device functionality as $\overline{\text{LGTA}}$/LUPWAIT is actually latched on the subsequent rising edge of the internal launch/ capture clock signal.

Impact: Asserting $\overline{\text{LGTA}}$/LUPWAIT for only the falling edge of the internal launch/capture clock signal in PLL-bypass mode will result in the local bus controller not registering the $\overline{\text{LGTA}}$/LUPWAIT input.

Workaround: Asserting the $\overline{\text{LGTA}}$/LUPWAIT signal for an additional LCLK cycle in PLL-bypass mode will guarantee the local bus controller registering the $\overline{\text{LGTA}}$/LUPWAIT input correctly.

Fix plan: No plans to fix

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

eSDHC1: Data End Bit Error (DEBE, bit 22 in the interrupt status register) incorrectly set if Card INTerrupt is driven in SDIO 1 bit mode

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the eSDHC transfers data from/to the card and the interrupt line is asserted by the SDIO card, the eSDHC indicates data end bit error after each block transfer, although the data end bit in the data packet is correct.

The problem occurs only under the following conditions:

1. The eSDHC operates in SDIO 1-bit SD data transfer mode.

2. The eSDHC transfers data from/to the card, and the interrupt line is asserted by the SDIO card (SD_DAT1 signal is driven to 0) at the same time that the end bit of the data or CRC is transferred.

With these conditions, Data End Bit Error (DEBE, bit 22 in the interrupt status register) and Card INTerrpt (CINT, bit 8 in the interrupt status register) are set.

Impact: None, if workaround is implemented.

Workaround: When DEBE and CINT status bits in the interrupt status register are set, the software should ignore DEBE status, but it must not ignore the other status bits (e.g. CRC error).

The software should also clear this status bit by writing 1 to this field. It is highly recommended to clear this bit before the next transfer.

Fix plan: No plans to fix

eSDHC2: Read/Write Multiple command when block count is 1

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the driver sends multiple read/write commands to the card and the block count is 1, with auto CMD12 enabled, eSDHC does not send auto CMD12 at the end of the transfer.

Impact: In this state, the driver should send CMD12 and only then can it send another read/write command to the device.

Workaround: For a single block write, the driver should send the WRITE_BLOCK command (CMD24). For a single block read, the driver should send the READ_SINGLE_BLOCK command (CMD17).

Fix plan: No plans to fix

eSDHC3: Force Event Register

Description: Devices: MPC8379E, MPC8378E, MPC8377E

If any of the events in the interrupt status register are cleared in the previous writing of the interrupt status register, the event cannot be forced in the next (first) access to the force event register. This event will be forced only in the second access to the force event register.

Impact: There is an overhead for the host driver to write twice to the force event register.

Workaround: If the driver uses this feature, the IPGEN bit in the system control register should be high or the host driver should write twice to the force register to force the event.

Fix plan: No plans to fix

eSDHC4: CPU polling read mode when the burst length is one word

Description: Devices: MPC8379E, MPC8378E, MPC8377E

For CPU polling read mode, when burst length (RD_WML) is 1 or the remaining block size is 1, the last data read by software, from the buffer, may be invalid.

Impact: For cases when burst length (RD_WML) is 1 or the remaining block size is 1, the buffer read reasy bit will be set even if there is only one word available. In this case the software read may be invalid.

Workaround: If the burst length is never 1 during a read operation (the burst legnth is the smaller of RD_WML and remaining block size of the current block), then this issue will not occur.

Fix plan: No plans to fix

eSDHC5: eSDHC reads more data than expected from the system bus

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When using the internal DMA to write data to the card, it is possible for the eSDHC to read more data than expected from the system address. This depends on the value of the WR_WML field in the watermark level register and on the DVS and SDCLKFS fields in the system control register.

Impact: After the DMA has stopped, the value of the DMA system address register should be the next system address of the next contiguous data position. Due to this issue, sometimes the value of this register is the next system address of the next contiguous data position plus the write burst length (WR_WML x 4).

This behavior can be problematic when the next contiguous address is not an accessible address. In this case, the eSDHC may have a transfer error and indicate a DMA error event.

Workaround: If the eSDHC DMA is used, ensure that the next contiguous address is an accessible address if possible. Ignore transfer error when DMA is used for SD write.

Fix plan: No plans to fix

eSDHC6: eSDHC indicates AUTO CMD12 interrupt later than expected

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the IPGEN bit in the system control register is cleared, and an error occurs during the response to AUTO CMD12, the eSDHC does not indicate an AUTO CMD12 interrupt until the IPGEN bit is set to 1 or until the host driver sends another command to the card. At this time, the AUTO CMD12 error status register gets the correct value, but the eSDHC does not indicate an AUTO CMD12 interrupt in the interrupt status register.

Impact: After sending multiple read/write commands, when AUTO CMD12 is enabled, the host driver does not know if an error occurred in response to the stop transmission command (CMD12).

Workaround: Do not clear the IPGEN bit in data transfer with the AUTO CMD12 bit set.

Fix plan: No plans to fix

eSDHC7: Data corruption during write with pause operation

Description: Devices: MPC8379E, MPC8378E, MPC8377E

In the write with pause operation, after data transfer is resumed, the eSDHC corrupts one word (32 bits) in the middle of the transfer and writes the word to the card. This behavior occurs when the HCKEN bit in the system control register is cleared.

Impact: When the above situation occurs, a wrong value is written to the card.

Workaround: Do not clear HCKEN bit during a DMA transfer. Always set HCKEN bit to 1.

Fix plan: No plans to fix

eSDHC8: Cannot initiate non-data command while data transfer is in progress

Description: Devices: MPC8379E, MPC8378E, MPC8377E

According to the SD physical specification, the host driver can issue CMD0, CMD12, and CMD13 when the data lines are busy during data transfer. The host driver should then send these commands during a block gap and afterwards the host driver is able to resume the transfer. However, the related command bits are not protected during the block gap and thus data transfer cannot be resumed.

Impact: The host driver cannot initiate non-data transfer commands while a data transfer is in progress.

Workaround: None

Fix plan: No plans to fix

eSDHC10: Incorrect data is written to a card after transfer paused

Description: Devices: MPC8379E, MPC8378E, MPC8377E

For a write operation, if the host controller pauses the transfer, and there is no more data in the internal FIFO when transfer stops, an intermediate channel to write data to the card, loads dirty data and corrupts the beginning bytes of the next block when data transfer is resumed.

Due to this defect, the software may not reliably pause the write transfer if the write from system side is slow. That is, when the request to stop at block is sent out, if the system side is not writing data for the next block comparing the card side, data corruption occurs.

Impact: This issue only occurs for a write pause, and if suspend command is sent out to the card, this issue also disappears. In real applications, the requirement to pause and resume a write transfer is rare, so such an issue is not critical.

Workaround: The software workaround is to check the system side to confirm the system side has already written all the data blocks that are to write to the card prior to write pause, and then it is allowed to stop the transfer.

Fix plan: No plans to fix

eSDHC11: eSDHC Multiple Block Read Failures

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The eSDHC drives and samples data at the falling edge of the SD_CLK. The SD Card, in High Speed mode, drives the data at the rising edge of the SD_CLK.

Due to this implementation there are limitations to the clock and data delay propagations on the board (both min and max), and the data driven by the SD card on a Read operation should be sampled after 1.5 clock cycles by the eSDHC host.

If the internal data buffer is in danger, the eSDHC may gate off the SD clock to avoid buffer over/under-run. The eSDHC stops the SD clock after the rising edge while it is asserted.

The problem is that this last rising edge of the clock causes the SD card to drive the next data, but the previous data was never sampled by the eSDHC because at sample time the previous data's clock was already off. When the clock is re-enabled the data on the SD lines ends up being sampled twice.

Impact: Multiple block Read operations cannot be used in DMA mode.

Workaround: The 100% data-corruption-free workaround is to use only small DMA transfers, equal or less than the internal buffer size (512 bytes). This solution will have a great impact on performance.

The most practical way to handle this situation is to work normally, and in case of a CRC error, request the same transfer again.

There are a few programming options that decrease the probability of data corruption:

- The internal clock (ipg_clk) should be set to the highest possible frequency.
- The priority of the eSDHC on the CSB bus should be high.
- The watermark level of the eSDHC DMA should be set to a small value.

It is also important to note the following:

- CPU multiple block read operations must not be used
- A CRC error may occur during DMA read operations due to an internal error

Fix plan: No plans to fix

eSDHC12: During multi-write operation, unexpected Transfer Complete (IRQSTAT[TC] register) bit can be set

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Invalid Transfer Complete (IRQSTAT[TC]) bit could be set during multi-write operation even when the BLK_CNT in BLKATTR register has not reached zero. Therefore, Transfer Complete might be reported twice due to this erratum since a valid Transfer Complete occurs when BLK_CNT reaches zero.

Impact: The IRQSTAT[TC] status bit is not reliable during the multiple block transfer. The software driver needs to send additional command to check the card status to ensure transfer complete, in case false TC is reported.

Workaround: Ignore the IRQSTAT[TC] register bit if BLKATTR[BLK_CNT] register field is confirmed to be non-zero.

If block count is zero, it is more robust to poll card status using CMD13 for MMC/SD cards or CCSR[Ready Flags] for SDIO.

Fix plan: No plans to fix

eSDHC13: CRC might be corrupted for write data transfer if it is preceded by read transfer

Description: Devices: MPC8379E, MPC8378E, MPC8377E

CRC might be corrupted for a write data transfer if it is preceded by a read transfer. Software writing to the Transfer Type configuration register (system clock domain) can cause a setup/hold violation in the CRC flops (card clock domain), which can cause write accesses to be sent with corrupt CRC values. This issue occurs only for write preceded by read.

Impact: A write data transfer that follows read may have corrupted CRC content. Software needs to reset the data path or shut off the card clock for such a scenario.

Workaround: Use one of the following options:

- Set SYSCTL[RSTD] (software reset) bit after each read operation is done (transfer complete) and reconfigure all related registers.
- After a read operation, use the following steps:
  a. Clear bits 2–0 of the system control register (PEREN, HCKEN, IPGEN).
  b. Wait for bit 7 or 6 of the present state register to be set, which means either the card clock or the internal baud rate clock stops.
  c. Send either CMD13 to poll status or CMD24 or CMD25 to launch write operation.

Fix plan: No plans to fix

eSDHC14: Data End Bit Error (DEBE, bit 22 in the interrupt status register) incorrectly set if Card Interrupt is driven in SDIO 4-bit mode

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Data End Bit Error (DEBE, bit 22, in the interrupt status register) is incorrectly set if Card Interrupt is driven in SDIO 4 bit mode. eSDHC monitors the end bit at CRC status. For 4-bit mode and 8-bit mode (MMC card only), if any of the data lines is “0” at the end bit cycle, the end bit error occurs. This assumption is correct for MMC/SD cards, but it does not work for SDIO cards. For a single block write or a predefined number of write blocks, the interrupt may occur during the CRC status stage. The “0” on DAT1 (which may occur due to SDIO interrupt) will then lead to a false end bit error.

Impact: False DEBE interrupt may be issued to the core, so core interrupted due to false error which is a software overhead.

Workaround: For SDIO card block write, if both card interrupt status and end bit error status are set, the software needs to ignore the end bit error.

Fix plan: No plans to fix

eSDHC15: Unable to issue CMD12 if SD clock shuts off due to slow system access

Description: Devices: MPC8379E, MPC8378E, MPC8377E

If the eSDHC issues a data transfer, but system side fails to access its internal buffer as quickly as the card, and the buffer fills up for a read transfer or empties for a write transfer, the eSDHC will stop the card clock to avoid buffer overrun (for read) or underrun (for write). If CMD12 (with XFERTYP[CMDTYP] = 3, abort) is issued to stop the transfer, the CMD12 never makes it onto the SD/MMC interface due to this erratum.

Impact: There is software overhead to initiate dummy write/read accesses to buffer for switching on the SD clock to issue CMD12.

Workaround: A software workaround may be implemented to check the card clock status on issuing CMD12 to abort the data transfer. If the clock is stopped, several accesses need to be applied to the buffer to activate the clock.

Upon sending CMD12 to abort the data transfer, poll the card clock status to see if it is stopped. If so, write XFER_TYPE register to issue the CMD12 and make several accesses (write or read depending on the current transfer direction) to the buffer to change the buffer state. Doing this will restore the clock, then send the command.

Fix plan: No plans to fix

eSDHC16: Manual Asynchronous CMD12 abort operation causes protocol violations

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

eSDHC17: PRSSTAT[CIDHB] is not reliable for commands with busy (R1b)

Description: Devices: MPC8379E, MPC8378E, MPC8377E

PRSSTAT[DLA] (Data Line Active) is not reliable for commands, (such as CMD6, CMD7, CMD12, CMD28, CMD29, or CMD38), with busy signal. DLA affects PRSSTAT[CIDHB] (Command with Data Inhibit). Therefore, a software driver may not know the busy status in DLA/CIDHB and if it issues next command with data or R1b, there might be contention on SD bus and this might create data CRC error.

Impact: The PRSSTAT[CIDHB] and PRSSTAT[DLA] bit may not be used for commands with busy.

Workaround: Driver should read the card status, using SEND_STATUS command(CMD13), to check busy de-assertion before issuing next command which uses data line.

Fix plan: No plans to fix

eSDHC18: Invalid generation of Block Gap Event

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The block gap event (IRQSTAT[BGE]) may be wrongly generated if the stop at block gap request is asserted during the transfer of last block while performing a read or write operation.

Impact: IRQSTAT[BGE] may not be correct when PROCTL[SABGREQ] is set.

Workaround: When IRQSTAT[BGE] is set, check the block count (BLKATTR[BLKCNT]). If it is zero, then ignore the IRQSTAT[BGE] bit.

Fix plan: No plans to fix

eSDHC19: eSDHC cannot finish write operation after continuing from Block Gap Stop

Description: Devices: MPC8379E, MPC8378E, MPC8377E

After stop at block gap in write operation, when the transfer is continued (by setting PROCTL[CREQ]), the data transfer cannot finish and the transfer complete status bit cannot be set.

When PROCTL[SABGREQ]is set, transfer stops at current block completion on the SD interface and IRQSTAT[BGE] and IRQSTAT[TC] event for the stop request are generated.

When the Continue request is asserted after Stop at block gap, the host starts to transfer remaining blocks. Since the host pre-fetches one extra word from the buffer at stop at block gap, it either transfers corrupted data to the card and thus generates Write CRC status error (IRQSTAT[DCE]) or shuts off the SD clock, which prevents transfer complete(IRQSTAT[TC]) generation and creates deadlock.

This occurs because one extra word is pre-fetched after the block transfer is stopped at block gap and when Continue is requested—the buffer is left with one word less than actual block size (because of one pre-fetch after the previous block).

Impact: Continue cannot be used after Stop at block gap.

Workaround: Software should not set PROCTL[SABGREQ] as this enables stop at block gap request.

Fix plan: No plans to fix

eSDHC20: Corrupted data read from eSDHC for certain values of WML[RD_WML] and BLKATTR[BLKSIZE]

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Corrupted data may be read from the internal eSDHC buffer if previous buffer read access and buffer prefetch occurs at the same time. This is valid for both CPU polling and DMA modes. This issue is observed only when WML[RD_WML] or ((BLKATTR[BLKSIZE] + 3) ÷ 4) is set to an odd value.

Impact: Restriction on using certain values of WML[RD_WML] and BLKATTR[BLKSIZE].

Workaround: BLK_SIZE_IN_WORDS and WML[RD_WML] should not be odd; where BLK_SIZE_IN_WORDS = ((BLKATTR[BLKSIZE] + 3) ÷ 4).

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: Devices: MPC8379E, MPC8378E, MPC8377E

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-A001: Data timeout counter (SYSCTL[DTOCV]) is not reliable for values of 0x4, 0x8, and 0xC

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The data timeout counter (SYSCTL[DTOCV]) is not reliable for DTOCV values 0x4(2^17 SD clock), 0x8(2^21 SD clock), and 0xC(2^25 SD clock). The data timeout counter can count from 2^13–2^27, but for values 2^17, 2^21, and 2^25, the timeout counter counts for only 2^13 SD clocks.

Impact: SYSCTL[DTOCV] is not reliable for values 0x4, 0x8, and 0xC. These values cannot be used.

Workaround: Program one more than the affected value for SYSCTL[DTOCV]. Instead of programming the values of 4, 8, and 12, the SYSTCTL[DTOCV] should be 5, 9, and 13, respectively.

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: Devices: MPC8379E, MPC8378E, MPC8377E MPC8306 MPC8309

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: Devices: MPC8379E, MPC8378E, MPC8377E

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-005286: CPU polling read mode when the burst length is one word

Affects: eSDHC

Description: Devices: MPC8379E, MPC8378E, MPC8377E

For CPU polling read mode, there is one flip-flop to latch the data read from the internal buffer. However, the pre-read latch is only performed after there are at least two words available in the buffer.

When the burst length (RD_WML) is 1, or the remaining block size is 1, the buffer read ready bit is set even if there is only one word available; in this case, the flip-flop may not have been updated and the software read may be invalid.

Impact: Read watermark SDHC_WML[RD_WML] cannot be set to 1.

Workaround: If the burst length is never 1 in read operation (the burst length is the smaller of RD_WML and the remaining block size of the current block), the issue will not occur. In other words, BLK_SIZE_IN_WORDS % WML[RD_WML] should not be equal to 1, where BLK_SIZE_IN_WORDS = ((BLKATTR[BLKSIZE] + 3) / 4).

This means that WML[RD_WML] should be programmed such that the block size is in number of words (4 bytes) when divided by the read watermark level, the remainder should not be 1.

Fix plan: No plans to fix

A-005287: Read multiple block failure

Affects: eSDHC

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The eSDHC drives and samples data at the falling edge of the SD_CLK. In high-speed mode, the SD card drives the data at the rising edge of the SD_CLK. This implementation results in minimum and maximum limitations to the clock and to data delay propagation on the board. Data driven by the SD card on a read operation should be sampled by the host (eSDHC) after 1.5 clock cycles. When the internal data buffer is in danger, the eSDHC may gate off the SD clock to avoid buffer overrun/underrun. The eSDHC stops the SD clock after the rising edge, while it is asserted. The problem occurs when the last rising edge of the clock causes the SD card to drive the next data. However, because the eSDHC clock was already off at sample time, the previous data was never sampled by the eSDHC. When the clock is re-enabled, the data on the SD lines is sampled twice.

Impact: Multiple-block read operations cannot be used in CPU high-speed mode.

DMA mode is the main operation mode and usually works with multiple-block read operations. Depending on the SoC architecture and load, the data may be occasionally corrupted, causing a CRC error. This invalidates the entire chunk of data that was intended to be transferred by the DMA operation.

According to theoretical calculations as well as testing, the probability of data corruption is very low. This is due to the difference between the SoC overall throughput and the SD throughput; that is, the DMA is able to empty the internal buffer before the bug scenario is reached. It has not yet been possible to reproduce the failure in silicon under this condition.

Workaround: In DMA mode, perform the following:

To ensure no data is corrupted, use only small DMA transfers, equal to or less than the internal buffer size (512 bytes). However, this may not be practical and it could impact performance. A more practical way is to work normally and, in the case of a CRC error, request the same transfer again.

The following are a few programming options that decrease the probability of data corruption:

- Set the internal clock at the eSDHC-SoC interface to the highest possible frequency.
- Ensure the eSDHC priority on the internal bus is high.
- Set the watermark level of the eSDHC DMA to a small value.

Fix plan: No plans to fix

eTSEC11: VLAN extraction with shim header not supported

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Shim header shifts the eTSEC header, including the VLAN ID, by 2 to 254 bytes. The VLAN extraction feature of the controller does not take that shift into account, and examines bytes from the wrong offset of the header. In most cases, the data at the unshifted offset does not match the VLAN ID, so no extraction occurs and nothing is forwarded to the filer.

If the data at the unshifted offset happens to match the VLAN ID by coincident (0x8100 or the value in DFVLAN), then those bytes are incorrectly extracted from the frame and forwarded to the filer and the actual VLAN ID, if any, will be left in the frame.

Impact: VLAN extraction cannot be used if shim headers are enabled.

Workaround: If shim headers are enabled (RCTRL[L2OFF] ≠ 0), disable VLAN extraction (by setting RCTRL[VLEX] = 0).

Fix plan: No plans to fix

eTSEC13: Fetches with errors not flagged, may cause livelock or false halt

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The error management for address (for example, unmapped address) and data (for example, multi-bit ECC) errors in the Ethernet controller does not properly handle all scenarios. The behavior is as follows:

Scenario 1

- Address error on Tx data fetch

  The Ethernet controller does not detect errors on Tx data fetches. IEVENT[EBERR] is not set and the queues are not halted. For address errors, no data is returned to eTSEC and the controller hangs. The error may still be detected at the platform level, via an interrupt from the source of the error (e.g. ECM/MCM address mapping error).

Scenario 2

- Data error on Tx data fetch

  The Ethernet controller does not detect errors on Tx data fetches. IEVENT[EBERR] is not set and the queues are not halted. For data errors, the frame is transmitted as if data is good, with good FCS. The error may still be detected at the platform level, via an interrupt from the source of the error (e.g. DDRC multibit ECC error).

Some fetch errors are handled correctly. The correct behavior is as follows:

- Non-first TxBD fetch for queue 0 OR TxBD fetch for queues 1-7

  The Ethernet controller will set IEVENT[EBERR] and halt all Tx queues (TSTAT[THLTn]=1, n=0-7). EDIS[EBERRDIS] must be 0.

- RxBD fetch

  The Ethernet controller will set IEVENT[EBERR] and halt the queue with the error (RSTAT[QHLTn]=1). EDIS[EBERRDIS] must be 0.

Impact: The Ethernet controller may stop transmitting packets without setting IEVENT[EBERR] if a buffer descriptor or data fetch has an uncorrectable error.

The transmit scheduler may halt queues without setting IEVENT[EBERR] if a buffer descriptor fetch has an uncorrectable error.

In case of platform errors, the controller will transmit corrupted system data without an error indicator.

Workaround: All scenarios:

1. Make sure all eTSEC BD and data addresses map to valid regions of memory.
2. Ensure EDIS[EBERRDIS] = 0.

Transmit buffer descriptor work around:

For recovery from error scenarios 1:

If error interrupt handlers cannot resolve address or data errors without changing Tx state (e.g. BD address), execute a Tx reset to recover from Tx livelock condition.

The Tx reset sequence is:

1. Set DMACTRL[GTS]
2. Poll IEVENT[GTSC] until set or 10,000 byte times elapse
3. Clear MACCFG1[Tx_Flow]
4. Wait 256 TX_CLK cycles
5. Clear MACCFG1[Tx_EN]
6. Wait 3 TX clocks
7. Set MACCFG1[Reset Tx MC] and MACCFG1[Reset Tx fun]
8. Wait 3 TX clocks
9. Clear MACCFG1[Reset Tx MC] and MACCFG1[Reset Tx fun]
10. Set TBPTRn to next available BD in the TX ring.
11. Set MACCFG1[Tx EN] and, if desired, MACCFG1[Tx Flow]
12. Clear DMACTRL[GTSC]

For error scenario 2:

Data errors can be flagged by platform for additional software processing, but no workaround exists to prevent the transmission of corrupted data.

Fix plan: No plans to fix

eTSEC16: Parsing of tunneled IP packets not supported

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Encapsulation of IP in IP in either TCP or UDP packets is not supported by eTSEC parser. This applies to both IPv4 and IPv6.

A tunneled IP packet is an IP/TCP or IP/UDP packet and one of the following:

1. IPv4 header with a value of either 4 or 41 in the Protocol field, indicating that the next header is either another IPv4 header or IPv6 header, respectively
2. IPv6 header with a value of either 4 or 41 in the Next Header field, indicating that the next header is either a IPv4 header or another IPv6 header, respectively

When the parser encounters a tunneled IP packet, it terminates its parsing operation at the end of the outer IP header.

Impact: Validly encapsulated tunneled IP packets may cause a false parser error or false TCP/UDP checksum error.

Malformed tunneled packets may be received without a parser error.

Tunneled packets with an actual TCP/UDP checksum error may fail to report a checksum error.

Workaround: If L3 or L4 parsing is enabled and tunneled packets are expected, software must examine each packet header to see if it is a tunneled IP packet. If the packet is a tunneled IP packet, software should ignore any parser or checksum error.

Fix plan: No plans to fix

eTSEC42: Frame is dropped with collision and HALFDUP[Excess Defer] = 0

Description: Devices: MPC8379E, MPC8378E, MPC8377E

eTSEC drops excessively deferred frames without reporting error status when HALFDUP[Excess Defer] = 0. This erratum affects 10/100 Half Duplex modes only.

Impact: The eTSEC does not correctly abort frames that are excessively deferred. Instead it closes the BD as if the frame is transmitted successfully. This results in the frame being dropped (because it is never transmitted) without any error status being reported to software.

Workaround: Do not change HALFDUP[Excess Defer] from its default of 1.Thus eTSEC always tries to transmit frames regardless of the length of time the transmitter defers to carrier.

Fix plan: No plans to fix

eTSEC45: eTSEC parser does not perform length integrity checks

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The eTSEC currently only uses the total length reported in the IPv4 or IPv6 header when calculating checksums. This checksum calculation includes the length used in the pseudoheader, and the actual length of the data in the payload. Proper operation when parsing a subsequent L4 header that has a length field (be it payload and/or header) should be to check for consistency against what is reported in the outer IP header. If there is a mismatch, the eTSEC should signal a parse error and not perform the UDP or TCP payload checksum check (for example, RxFCB[PERR] = 10 and RxFCB[CTU] = 0).

One simple example of this is that UDP has its own payload length. If eTSEC encounters a simple IPv4/UDP packet it should take the IP total length field, subtract IP header length and that should equal the UDP payload length. If it doesn't then this packet is malformed.

Impact: Could get false checksum failures or false checksum passes.

Workaround: False checksum fails can be worked around by rechecking them in software running on the host.

Fix plan: No plans to fix

eTSEC46: eTSEC does not verify IPv6 routing header type field

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The RFC2460 (current referenced standard for IPv6 operation) states that when encountering a packet with an unrecognized Routing Type Value, and the field “segments left” is non-zero, the node must discard the packet and return an ICMP Parameter problem, Code 0, message to the packet’s source address. The eTSEC only recognizes type0 routing headers, but incorrectly interprets all Routing Type fields as type0 (for example, ignores the type field and continues parsing the packet including upper layer protocol checksums). The correct behavior is to signal parser error, and not check upper layer checksums

Impact: eTSEC parser/checksum engine incorrectly interprets non-type0 IPv6 routing headers. Functionally, this is a future-proof issue because there are currently no other type-fields defined.

Workaround: If this device is operating in a network that is using non-type0 IPv6 routing headers, then the upper layer processing (IPv6 extension headers, and payload checksumming operations) must be performed in software.

Fix plan: No plans to fix

eTSEC48: L4 info passed to filer in L2/L3-only mode

Description: Devices: MPC8379E, MPC8378E, MPC8377E

RCTRL[PRSDEP] has parse control for L2 and L3 (10) and L2, L3 and L4 (11). In the case of L2 and L3, the eTSEC goes ahead and continues to parse into L4 protocols and update both RxFCB and filer PID = 1 fields for TCP and UDP.

Impact: The L4 protocols are parsed and status bits are set even when the eTSEC is not programmed to include L4 parsing.

Workaround: User can simply ignore the bits associated with L4 protocols. In the case of filer PID = 1, user must mask bits associated with TCP and UDP.

Fix plan: No plans to fix

eTSEC58: VLAN Insertion corrupts frame if user-defined Tx preamble enabled

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When TCTRL[VLINS] = 1, the VLAN is supposed to be inserted into the Tx frame 12 bytes after start of the Destination Address (after DA and SA). If user-defined Tx preamble is enabled (MACCFG2[PreAmTxEn] = 1), the VLAN ID is inserted 12 bytes after the start of the preamble (4 bytes after start of DA), thus overwriting part of DA and SA.

Impact: If VLAN insertion is enabled with user-defined Tx preamble, the VLAN ID corrupts the Tx frame destination and source addresses.

Workaround: Use one of the following workarounds:

- Disable user-defined Tx preamble by setting MACCFG2[PreAmTxEn] = 0.
- Disable VLAN insertion by setting TCTRL[VLINS] = 0.

Fix plan: No plans to fix

eTSEC64: User-defined Tx preamble incompatible with Tx Checksum

Description: Devices: MPC8379E, MPC8378E, MPC8377E

If user-defined Tx preamble is enabled (by setting MACCFG2[PreAmTxEn]=1), an extra 8 bytes of data is added to the frame in the Tx data FIFO. IP and TCP/UDP checksum generation do not take these extra bytes into account and write to the wrong locations in the frame.

Impact: Enabling both user-defined Tx preamble and IP or TCP/UDP checksum causes corruption of part of the corresponding header.

Workaround: Use one of the following workarounds:

- Disable user-defined Tx preamble by setting MACCFG2[PreAmTxEn] = 0.
- Disable IP and TCP/UDP checksum generation by setting TCTRL[IPCSEN]=0 and TCTRL[TUCSEN] = 0.

Fix plan: No plans to fix

eTSEC67: ECNTRL[AUTOZ] not guaranteed if reading MIB counters with software

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The MIB function of the Ethernet controller has a feature to automatically zero out the registers when reading them if ECNTRL[AUTOZ] = 1. If the register read occurs in the same cycle as a hardware update of the register, then the register clear will not occur. Any software periodically reading MIB registers would expect to read A the first time, B the second, and C the third, with each value representing only the events that occurred in the interval between reads. If the first read collides with a hardware update, the second read would return A + B instead of B.

Hardware updates for MIB registers occur once per frame. For streaming 64-byte frames, the update would be every 84 Rx or Tx clocks (8 bytes of preamble, 64 bytes of data and 12 cycles of IPG).

Impact: Software polling of MIB counters with ECNTRL[AUTOZ] = 1 will over an extended period read a larger number of events than actually seen by the controller.

Workaround: Disable automatic clearing of the MIB counters by writing ECNTRL[AUTOZ] = 0. Software routines which periodically read MIB counters and accumulate the results should accumulate only when an MIB counter overflows, as in the description that follows: Assuming a 32-bit MIB counter (MIB_VALUE), a 64-bit accumulator consisting of two 32-bit registers (ACCUM_HI, ACCUM_LO), and a Carry Out bit (ACCUM_LO_CO), change the 64-bit accumulator update as follows:

Previous accumulate method (with ECNTRL[AUTOZ] = 1):

```
// Accumulate the MIB_VALUE into the lower half of the accumulator
 {ACCUM_LO_CO,ACCUM_LO} = {1'b0,ACCUM_LO} + {1'b0,MIB_VALUE};
// Accumulate the Carry Out from the step above, as well as the MIB register
OVFRFLW, which is detected through the CARn register.
{ACCUM_HI_CO,ACCUM_HI} = {1'b0,ACCUM_HI} + ACCUM_LO_CO + OVRFLW;
```

New accumulate method (with ECNTRL[AUTOZ]=0):

```
// Read instead of accumulate since we are not clearing MIB_VALUE
 ACCUM_LO = MIB_VALUE;
// Accumulate the MIB register OVRFLW, which is detected through the CARn
register
 {ACCUM_HI_CO,ACCUM_HI} = {1'b0,ACCUM_HI} + OVRFLW;
```

Fix plan: No plans to fix

eTSEC69: Ethernet controller does not exit from Magic Packet mode when an oddly formed Magic Packet is received

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The Ethernet MAC should recognize as a Magic Packet any Ethernet frame with the following contents:

- A valid Ethernet header (Destination and Source Addresses)
- A valid FCS (CRC-32)
- A payload that includes the specific MagicPacket byte sequence at any offset from the start of data payload.

The specific byte sequence comprises an unbroken stream of 102 bytes, the first 6 bytes of which are 0xFFs followed by 16 copies of the MAC's unique IEEE station address in the normal byte order for Ethernet addresses.

However, if a complete Magic Packet sequence (including 6 bytes of 0xFF) immediately follows a partial Magic Packet sequence the complete sequence will not be recognized and the MAC will not exit Magic Packet mode.

The following are examples of partial sequences followed by the start of a complete sequence for a station address 01_02_03_04_05_06:

- FF_FF_FF_FF_FF_FF_FF_01_02_03_04_05_06_01...

  Seventh byte of 0xFF does not match next expected byte of Magic Packet sequence (01). Pattern search restarts looking for 6 bytes of FF at byte 01.
- FF_FF_FF_FF_FF_FF_01_FF_FF_FF_FF_FF_FF_01_02_03_04_05_06_01...

  First FF byte following 01 does not match Magic Packet sequence.

  Pattern search restarts looking for 6 bytes of FF at second byte of FF following 01.

The following is an example partial sequence followed by the start of a complete sequence that is erroneously not recognized for station address 01_02_03_04_FF_06:

- Sequence c) FF_FF_FF_FF_FF_FF_01_02_03_04_FF_FF_FF_FF_FF_FF_01...

  11th byte (0xFF) is seen as the 11 byte of the partial pattern and is not recognized as the start of a complete sequence.

  Pattern search restarts looking for 6 bytes of 0xFF at 12th byte, but sees only 5.

Impact: The Ethernet controller does not exit Magic Packet mode if the Magic Packet sequence is placed immediately after other frame data that partially matches the Magic Packet sequence.

Workaround: There is no on-chip software or hardware workaround that can be performed to avoid or recover from this behavior. The controller does not wake up if one of these oddly formed magic packets is received, but any subsequent, properly formatted magic packet does wake up the controller.

If the received magic packet is properly formed, this erratum is avoided.

Fix plan: Partially fixed in Rev 2.0

Sequences a) and b)—fixed

Sequence c)—no plans to fix

eTSEC72: Receive pause frame with PTV = 0 does not resume transmission

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The Ethernet controller supports receive flow control using pause frames. If a pause frame is received, the controller sets a pause time counter to the control frame's pause time value, and stops transmitting frames as long as the counter is non-zero. The counter decrements once for every 512 bit-times. If a pause frame is received while the transmitter is still in pause state, the control frame's pause time value replaces the current value of the pause time counter, with the special case that if the pause control frame's pause time value is 0, the transmitter should exit pause state immediately. The controller does use the frame's pause time value to set the current pause time counter, but it then decrements the pause time counter before performing the compare to zero. As a result an XON (pause frame with PTV = 0), actually causes the transmitter to continue in pause state for 65,535 pause quanta, or 33,553,920 bit times.

Impact: A received pause frame with PTV = 0 causes the transmitter to pause for 65,535 pause_quanta. The expected behavior is for the controller to continue, or resume, transmission immediately. Note that the Ethernet controller always uses the value of the PTV register when generating pause frames. It never automatically generates a pause frame with pause time value of 0 when the receiver recovers from being above the RxFIFO threshold or below the free RxBDs threshold.

Workaround: To force an exit of pause state, use a pause frame with PTV value of 1 instead of 0.

Fix plan: No plans to fix

eTSEC73: TxBD polling loop latency is 1024 bit-times instead of 512

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Register bit DMACTRL[WOP] defines the use of wait on poll when transmit ring scheduling algorithm is set to single polled ring mode. (TCTRL[TXSCHED]=00). When the use polling is selected by setting DMACTRL[WOP]=0, the poll to TxBD on ring 0 should occur every 512 bit-times. Due to the errata the poll occurs every 1024 bit-times.

Impact: The duration of the polling is twice as long as originally specified.

Workaround: None

Fix plan: No plans to fix

eTSEC74: MAC: Rx frames of length MAXFRM or MAXFRM-1 are marked as truncated

Description: Devices: MPC8379E, MPC8378E, MPC8377E

If MACCFG2[Huge Frame]=0 and the Ethernet controller receives frames which are larger than MAXFRM, the controller truncates the frames to length MAXFRM and marks RxBD[TR]=1 to indicate the error. The controller also erroneously marks RxBD[TR]=1 if the received frame length is MAXFRM or MAXFRM-1, even though those frames are not truncated.

No truncation or truncation error occurs if MACCFG2[Huge Frame]=1.

Impact: If MACCFG2[Huge Frame]=0, Rx frames of length MAXFRM or MAXFRM-1 are received normally, but RxBD[TR] is set to 1.

Workaround: Option 1:

Set MACCFG2[Huge Frame]=1, so no truncation occurs for invalid large frames. Software can determine if a frame is larger than MAXFRM by reading RxBD[LG] or RxBD[Data Length].

Option 2:

Set MAXFRM to 1538 (0x602) instead of the default 1536 (0x600), so normal-length frames are not marked as truncated. Software can examine RxBD[Data Length] to determine if the frame was larger than MAXFRM-2.

Fix plan: No plans to fix

eTSEC75: Misfiled Packets Due to Incorrect Rx Filer Set Mask Rollback

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the eTSEC Rx filer exits a cluster or AND chain that does not produce a match, it should roll back the mask to the value at the beginning of the chain or cluster. If that cluster or AND chain does not contain a Set Mask rule, however, the mask rolls back to the value prior to the previous Set Mask rule due to this erratum.

The following list provides an example of how a rule sequence should maintain a Mask for filer frame matching (the mask value is as it should be starting evaluation of the rule):

Rule #1: <Mask = default, 0xFFFF_FFFF>

Set Mask to 0x0000_8000 to search for frame data pattern that would match, as programmed by RQFPR's property field, and RQFCR[PID, Property ID].

Rule #2: <Mask = 0x0000_8000>

Look for a frame with status of having Broadcast address as the destination address.

Rule #3: <Mask = 0x0000_8000>

Start a Cluster of rules, grouped together for a special match.

Rule #4: <Mask = 0x0000_8000>

Set Mask to 0x0000_0210 inside Cluster, to search for frame data pattern that would match, as programmed by RQFPR's property field, and RQFCR[PID, Property ID].

Rule #5: <Mask = 0x0000_0210>

Inside Cluster, exit cluster and look for frame with status IPv4 header and UDP. Roll back mask to value at start of cluster.

Rule #6: <Mask = 0x0000_8000>

Set Mask (outside of cluster) to new value 0xFF00_FFFF.

Rule #7: <Mask = 0xFF00_FFFF>

Start of AND chain of 2 rules, the 1st rule, look for Destination Address 24-bits & Mask greater than RQPROP

Rule #8: <Mask = 0xFF00_FFFF>

End of AND chain, the 2nd rule, look for Destination Address low 24-bits & Mask greater than RQPROP. Roll back mask to value at start of chain.

Rule #9: <Mask = 0xFF00_FFFF>

Start of AND chain of 2 rules, the 1st rule, look for Source Address high 24-bits & Mask less than RQPROP

Rule #10: <Mask = 0xFF00_FFFF>

End of AND chain, the 2nd rule, look for Source Address low 24-bits & Mask less than RQPROP

Rule #11: <Mask = 0xFF00_FFFF>

etc.

The incorrect behavior in this example starts after rule #8:

Rule #8: <Mask = 0xFF00_FFFF>

End of AND chain, the 2nd rule, look for Destination Address low 24-bits & Mask greater than RQPROP. Roll back mask to value at start of chain.

After rule #8, the mask should roll back to the mask set by the last Set Mask rule outside the cluster (rule #6), but instead rolls back to the previous mask value (set by rule #1, and restored after cluster exit between rule 5 and 6).

Rule #9: <Mask = 0x0000_8000>

Start of AND chain of 2 rules, the 1st rule, look for Source Address high 24-bits & Mask less than RQPROP. Mask should be 0xFF00_FFFF.

Rule #10: <Mask = 0x0000_8000>

End of AND chain, the 2nd rule, look for Source Address low 24-bits & Mask less than RQPROP. Mask should be 0xFF00_FFFF.

Rule #11: <Mask = 0x0000_8000>

Mask should be 0xFF00_FFFF.

etc.

The AND chain of rule #9 and 10, or any rule that follows it, could falsely reject or misfile an incoming frame because the wrong mask is being applied.

Impact: The Filer can accept or reject a frame based on filer rule matching using incorrect mask.

Workaround: Use one of the following workarounds:

- Have all clusters or AND chains contain a Set Mask rule.
- After a stand alone Set Mask rule, the next cluster or AND chain in a sequence of filer rules should have immediately following it a repeat of the previous stand alone Set Mask rule.

Fix plan: No plans to fix

eTSEC76: Excess delays when transmitting TOE=1 large frames

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The Ethernet controller supports generation of TCP or IP checksum in frames of all sizes. If TxBD[TOE]=1 and TCTRL[TUCSEN]=1 or TCTRL[IPCSEN]=1, the controller holds the frame in the TxFIFO while it fetches the data necessary to calculate the enabled checksum(s). Because the checksums are inserted near the beginning of the frame, transmission cannot start on a TOE=1 frame until the checksum calculation and insertion are complete.

For TOE=1 huge or jumbo frames, the data required to generate the checksum may exceed the 2500-byte threshold beyond which the controller constrains itself to one memory fetch every 256 eTSEC system clocks. This throttling threshold is supposed to trigger only when the controller has sufficient data to keep transmit active for the duration of the memory fetches. The state machine handling this threshold, however, fails to take large TOE frames into account. As a result, TOE=1 frames larger than 2500 bytes often see excess delays before start of transmission.

Impact: TOE=1 frames larger than 2500 bytes may see excess delays before start of transmission.

Workaround: Limit TOE=1 frames to less than 2500 bytes to avoid excess delays due to memory throttling.

When using packets larger than 2700 bytes, it is recommended to turn TOE off.

Fix plan: No plans to fix

eTSEC77: SGMII receiver loss of signal threshold marginality

Description: Devices: MPC8378E

The SerDes receiver in SGMII mode may identify a false Loss Of Signal threshold, slightly out of the specified range.

The specified range for loss of signal threshold (VLOS) is 100 mV. In rare cases, the SerDes receiver may detect a loss of signal also in the amplitude range between 100 mV and 150 mV. When the threshold condition is met, the receiver is disabled.

NOTE

The concept of the VLOS parameter is equivalent to the Electrical Idle Detect Threshold parameter in PCI Express. Refer to the PCI Express Differential Receiver (RX) Input Specifications section of the hardware specification for further explanation.

Impact: The MPC8378E as an SGMII receiver may detect a false loss of signal condition in systems with very large signal attenuation or noise.

Workaround: Disable the loss of signal circuit in the SerDes by setting the following fields in the SerDes Control Register 2 (SRDS*n*CR2):

SEICA, SEICE = 000

PEICA, PEICE = 11

When the loss of signal (idle detect) circuit of the SerDes receiver is disabled, the receiver is always turned on and is able to detect the signal within the specified amplitude. However, due to signal integrity issues, noise may be recognized as signal by the receiver. In this case, the software should be able to filter out any unexpected received data.

In applications where the SerDes receiver should be idle for long period of times, it is recommended to disable the receiver by software during these periods if the loss of signal (idle detect) circuit of the SerDes receiver is disabled.

Fix plan: No plans to fix

eTSEC78: Controller may not be able to transmit pause frame during pause state

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the Ethernet controller pauses transmit of normal frames after receiving a pause control frame with PTV!=0, it should still be able to transmit pause control frames. The Ethernet controller, however, does not check whether the MAC is paused before initiating a start-of-frame request to the MAC. Once it has initiated a start-of-frame request, the Ethernet controller cannot initiate a pause control frame request until the normal frame completes transmission. Since the MAC will not transmit the normal frame until the pause time expires, this means the controller may be unable to transmit a pause frame while it is in pause state if there is a normal frame ready to transmit.

Impact: The Ethernet controller may be unable to transmit a pause frame during pause state if a normal frame is ready to transmit.

This applies to pause frame generation as a result of RxFIFO over threshold (ordinary flow control), free BDs below threshold (lossless flow control), or software-generated pause frame (TCTRL[TFC_PAUSE]).

Workaround: None

Fix plan: No plans to fix

eTSEC79: Data corruption may occur in SGMII mode

Description: Devices: MPC8378E

When operating the Ethernet controller in SGMII mode (ECNTRL[SGMIIM] = b'1), the device's SerDes's clock and data recovery may not accurately track the data if the incoming bit stream's data rate is slower than the expected data rate derived from the SD*n*_REF_CLK. This may cause data corruption in the received packet. The probability of failure is higher if SGMII is operating in 10-Mbps or 100-Mbps modes.

For example, if the SGMII SD*n*_REF_CLK is created by a 125 MHz ± 25 ppm oscillator, and the SGMII link partner reference clock is created by a separate 25 MHz ± 25 ppm oscillator, the incoming data rate may be slower, and this erratum applies.

Note that if the SGMII SD*n*_REF_CLK and the SGMII link partner reference clocks are created by a single clock oscillator, this erratum does not apply.

Impact: Customer systems that have an SGMII incoming bit stream data rate that is slower than the expected data rate derived from the SD*n*_REF_CLK may receive corrupted data in a packet that results in a CRC error that sets RxBD[CR] = 1 and increments the MIB counter RCDE.

When the packet is corrupted by this erratum and the CRC error is posted, one or more of the following may also occur:

- The packet may be truncated. This packet truncation may occur when an 8b10b symbol is incorrectly interpreted as a control character which forces end of packet.
- The SGMII link may go down until the next interpacket gap. While the link is down, eTSEC will transmit idles. If auto-negotiation is turned on (TBI MIIM Control Register [AN Enable] = 1), the eTSEC will attempt to auto-negotiate the SGMII link.
- The subsequent packet may be silently dropped.

After these events, normal data reception resumes.

Workaround: Use one of the following options:

- Use one clock oscillator to drive both the device SGMII SD*n*_REF_CLK and the link partner reference clock. There is an option to use a single ended clock for the SD*n*_REF_CLK. Refer to the device hardware specifications for details on single ended versus differential SD*n*_REF_CLK.
- Use a clock oscillator for the device's SGMII SD*n*_REF_CLK that has a frequency offset in the range of 10 to 200 ppm less than the SGMII link partner clock oscillator.

  For example: use a clock oscillator for SD*n*_REF_CLK which is specified at {125 MHz – 75 ppm} ± 25ppm = 124.990625 MHz ± 25 ppm, with a separate clock oscillator for the SGMII link partner, which is specified at 25 MHz ± 25 ppm. This combination creates a frequency offset of 25 to 125 ppm, with the incoming bit stream's data rate guaranteed to be faster than the expected data rate derived from the SD*n*_REF_CLK.

- Set bit 17 in SerDes registers at offset 0x80 and 0x88. This can be accomplished by ORing the existing values of these registers with 0x0000_4000.

Fix plan: No plans to fix

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the Ethernet controller receives a pause frame with PTV!=0, and MACCFG1[Rx Flow]=1, it completes transmitting any current frame in progress, then should 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 eTSEC 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.

Since 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

A-006502: Incomplete GRS or invalid parser state after receiving a 1- or 2-byte frame

Affects: eTSEC

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Ethernet standards define the minimum frame size as 64 bytes. The eTSEC controller also supports receiving short frames less than 64 bytes, and can accept frames more than 16 bytes and less than 64 bytes if RCTRL[RSF] = 1. Frames shorter than 17 bytes are supposed to be silently dropped with no side-effects. There are, however, two scenarios in which receiving frames <= 2 bytes cause erroneous behavior in the controller.

In the first scenario, if the last frame (such as an illegal runt packet or a packet with RX_ER asserted) received prior to asserting graceful receive stop (DMACTRL[GRS]=1) is <= 2 bytes, then the controller fails to signal graceful receive stop complete (IEVENT[GRSC]) even though the GRS has successfully executed and the receive logic is completely idle. Any subsequent receive frame that is larger than 2 bytes resets the state so the graceful stop can complete (IEVENT[GRSC] = 1). A MAC Rx reset also resets the state.

In the second scenario, the parser and filer are enabled (RCTRL[PRSDEP] = 01,10,11). If a 1- or 1.5-byte frame is received, the controller carries over some state from that frame to the next, causing the next frame to be parsed incorrectly. This, in turn, may cause incorrect parser results in RxFCB and incorrect filing (accept versus reject, or accept to wrong queue) for that following frame. The parser state recovers itself after receiving any frame >= 2 bytes in length.

Impact: If software initiates a graceful receive stop after a 1- or 2-byte frame is received, the stop may not complete until another frame has been received.

A frame following a 1 or 1.5B frame may be parsed and filed incorrectly.

Workaround: For GRS scenario:

After asserting graceful receive stop (DMACTRL[GRS] = 1), initiate a timeout counter. The wait time is system and memory dependent, but a reasonable worst-case time is the receive time for a 9.6 Kbyte frame at 10/100/1000 Mbps. If IEVENT[GRSC] is still not set after the timeout, read the eTSEC register at offset 0xD1C. If bits 7-14 are the same as bits 23-30, the eTSEC Rx is assumed to be idle and the Rx can be safely reset. If the register fields are not equal, wait for another timeout period and check again.

MAX Rx reset procedure:

1) Clear MACCFG[RX_EN].

2) Wait three Rx clocks.

3) Set MACCFG2[RX_EN].

Fix plan: No plans to fix

A-007207: TBI link status bit may stay up after SGMII electrical idle is detected

Affects: eTSEC

Description: Devices: MPC8378E

The TBI Status register (SR) contains a Link Status bit (TBI SR [Link Status]) that represents the current state of the SGMII link. If Auto-Negotiation (AN) is disabled, the TBI Link Status bit should be 1 (indicating the link is up) after recognizing IDLE sequences, and stay at 1 as long as valid data is received and the TBI is not reset. The TBI Link Status bit should be 0 (indicating the link is down) after several invalid characters are received or the TBI is reset. If AN is enabled, the TBI Link Status bit is not set to 1 until auto-negotiation is complete (TBI CR [AN DONE] = 1), but the same conditions as AN disabled then apply for the TBI Link Status bit to be cleared to 0.

An electrical idle (common mode) condition on the SGMII link results in the reception of invalid data and should cause the TBI Link Status bit to get cleared. If the transition from active to common mode takes enough time that the Rx is able to recognize at least 4 more K28.5 characters (for IDLE sequences, 70-80 UI), the portion of the design intended to detect the link down condition may shut off before the link down condition is actually reflected in the TBI.

This premature shutdown may cause the TBI Link Status to remain set to 1, indicating the link is up. This 'stuck at 1' condition persists until valid K28.5 characters are received again.

Impact: If the system never enters SGMII electrical idle, or if the transition from active to common mode takes less than 40 UI (~32 ns), then there is no impact and the false link up scenario does not occur.

If the system can generate an SGMII electrical idle condition as described above, then the TBI status may stay stuck at 1 while the link is down and does not transition to 0 until valid K28.5 characters are received again.

Workaround: If TBI SR[Link Status] = 0, the link is down.

For affected systems, there is no access to electrical idle detection circuitry in SGMII mode and there is no bit replacement for TBI SR[Link Status] to monitor.

When the TBI link status is set, the SW can periodically poll the state of the Ethernet Controller by reading RBYT (receive byte counter). If the controller is expected to receive data packets, RBYT should increment. If RBYT has not incremented over a period of time it could indicate the link is down. However, there are various reasons why RBYT may not increment (controller configuration errors, SerDes PLL issues, or MIB Counter RCDE incremented). Examination of system conditions may be necessary to determine why RBYT has not incremented.

Fix plan: No plans to fix

IEEE 1588_8: Writing Offset registers during use may yield unpredictable results

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The current system time, which is used to timestamp packets and events, is composed of the sum of the 64-bit time counter and the offset register. The offset register can be updated (written) by the CPU. The erratum is that CPU writes to the OFFSET register may result in temporarily unstable values on the register outputs, which can result in an incorrect calculation of the current system time, as well as incorrect timestamps.

Impact: Applications that require the CPU to periodically update the OFFSET register while it is being actively used may not work correctly due to unpredictable current time values.

Workaround: None.

Although the OFFSET register can be updated during initialization, care must be taken to ensure that it is not updated during active use. If the register is updated, ensure that any timestamps recorded during that time are ignored and not used as they may be in error.

Fix plan: No plans to fix

IEEE 1588_12: 1588 alarm fires when programmed to less than current time

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The 1588 alarm2 mechanism operates purely on a less-than-or-equal-to comparison with the current time. If the alarm is programmed to a value less than the current time, it fires immediately, rather than waiting for the current time to wrap around and cross the alarm time.

Impact: Alarm2 may fire before the intended time if armed when current time value is greater than alarm value.

Workaround: Ensure that the current time is less than the intended alarm time when programming TSEC_TMR_ALARMn_H to arm the event.

Fix plan: No plans to fix

IEEE 1588_14: TxPAL timestamp uses TxBD snoop enable instead of Tx data

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The DMACTRL register contains two snoop enable bits for Tx: TBDSEN for buffer descriptors and TDSEN for frame data. Tx timestamp writes should use TDSEN to determine transaction snoop enable, but use TBDSEN instead.

Impact: If DMACTRL[TBDSEN] = 0, Tx timestamp writes to memory will not be snooped by the core regardless of the setting of DMACTRL[TDSEN].

If DMACTRL[TBDSEN] = 1, Tx timestamp writes to memory will be snooped by the core even if DMACTRL[TDSEN] = 0.

Workaround: Set DMACTRL[TBDSEN] = 1 if snooping of Tx timestamp writes to memory is desired.

Fix plan: No plans to fix

IEEE 1588_16: Odd prescale values not supported

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The 1588 timer prescale register (TMR_PRSC) defines the timer prescale as follows: PRSC_OCK: Output clock division/prescale factor. Output clock is generated by dividing the timer input clock by this number. Programmed value in this field must be greater than 1. Any value less than 1 is treated as 2.

The output pulse (TSEC_TMR_PPn) width should be 1x the output clock width. For odd prescale values, the pulse width is 1.5x the output clock width instead.

Impact: Odd 1588 timer prescale values are not supported.

Workaround: Use only even timer prescale values.

Fix plan: No plans to fix

IEEE 1588_19: Tx FIFO data parity error (DPE) may corrupt Tx timestamps if TMR_CTRL[TRTPE]=1

Description: Devices: MPC8379E, MPC8378E, MPC8377E

If TMR_CTRL[TRPTE] = 1, the Ethernet controller writes Tx timestamp information for frames with TxFCB[PTP] = 1 to external memory in a PAL region. There is a queue of TxPAL addresses in the Ethernet controller that doesn't get cleared upon recovery from a Tx FIFO data parity error. As a result, if there were any frames with TxFCB[PTP]=1 in the Tx FIFO at the time the data parity error occurred (other than the frame with the error), a latent timestamp write for those frames may occur any time within the transmission of the first 8 TxBDs after DPE recovery. There may be up to three frames in the Tx FIFO in addition to the one with the parity error.

Example:

Suppose there are 4 frames in the Tx FIFO at time of the DPE, each with TxFCB[PTP] = 1 and each therefore using 2 BDs: BD0–BD7. When the DPE occurs, BD0 and BD1 are closed normally with an underflow indication, and BDs 2–7 are closed with an underrun indication, but TxPAL addresses for BD2, BD4 and BD6 are saved to a TxPAL queue and left valid. After the DPE recovery, the first 8 BDs transmitted trigger a write of invalid timestamp state to the TxPAL addresses of BD2, BD4 and BD6.

Impact: If TMR_CTRL[TRTPE]=1, data parity error may cause corruption of Tx timestamp data for PTP frames flushed due to the DPE.

Workaround: Transmit 8 or more non-TxPAL BDs which do not use any of the previously flushed PTP BDs, before allowing retransmission of those PTP BDs. These BDs must either be non-PTP frames (TxFCB[PTP]=0), or TMR_CTRL[TRTPE] must be 0. This can be done for example by:

- Keeping the ring(s) containing the flushed PTP BDs in halt while enabling other ring(s) containing at least 8 ready BDs. The 8 BDs can be for frames already programmed for transmission, or new ones for dummy frames in a queue enabled just for the purpose of clearing the TxPAL state.

OR

- Reordering the PTP BDs in the ring—moving them at least 8 entries down—so that least 8 other BDs in the ring are transmitted first.

In both cases, if the 8 BDs to be transmitted cannot be guaranteed to have TxFCB[PTP] = 0, then TMR_CTRL[TRTPE] must be set to 0 until those 8 BDs have been transmitted. Note that while TMR_CTRL[TRTPE] = 0, there will be no true Tx timestamp writes to memory for PTP frames, so it is preferable to ensure that only non-PTP frames are transmitted for the first 8 BDs.

Either workaround guarantees the true Tx timestamp write to memory for the flushed PTP frames (if enabled) will occur after the corrupted Tx timestamp write. Disabling TxPAL by setting TMR_CTRL[TRTPE] = 0 does not prevent the data corruption.

Fix plan: No plans to fix

IEEE 1588_21: IEEE 1588 accuracy can be adversely impacted in systems using multiple unsynchronized gigabit Ethernet ports

Description: Devices: MPC8379E, MPC8378E, MPC8377E

In RGMII Ethernet interface mode, all MACs use the GTX_CLK125 reference clock input to create the transmit clock that is used to transmit data from the MAC to the PHY. The MAC and PHY should be operated synchronously using a common clock reference although it is possible for the PHYs attached to these interfaces to transmit data across the Ethernet link at a slightly different frequency than they receive data from the MAC. This can occur if a PHY is configured as a slave PHY either manually or during the 1000Base-T Auto negotiation process. When configured as a slave, a PHY recovers the timing reference from the received link signal and uses this timing reference for transmit operations.

If the slave PHY's recovered timing reference has any frequency variations compared to the GTX_CLK125 clock that is used to transfer data from the MAC to the PHY, then the PHY transmitter will likely exhibit variations in delay due to the different clock frequencies. In systems that use any combination of two or more RGMII interfaces, it is possible for all of the PHYs attached to these individual interfaces to be configured as slave PHYs that have different reference frequencies. Although using the PHY's recovered output clock to drive the MAC's GTX_CLK125 reference clock enables one of the PHYs to operate synchronously with the MAC interface, it is not generally possible to synchronize all slave PHYs to their attached MACs since all MACs use a common GTX_CLK125 reference clock. This issue does not affect MACs configured in MII, RMII, or SGMII interface modes.

Impact: IEEE 1588 accuracy can be adversely impacted in systems using multiple unsynchronized gigabit Ethernet ports.

Workaround: Use one of the following workarounds to minimize delay variations within a PHY attached to the MAC interface:

- Use only one MAC configured in RGMII mode for passing IEEE 1588 messages and synchronize the MAC transmit interface to the PHY transmitter by using the PHY’s recovered 125 MHz output clock as the GTX_CLK125 clock input. This allows the PHY to be configured as either a MASTER PHY or SLAVE PHY during the Auto negotiation process.
- Use one or more MAC interfaces in RGMII mode and force the attached MACs to be in MASTER PHY mode by writing to the appropriate PHY control registers. This will prevent the PHYs from Auto negotiating into SLAVE mode, and it allows a common 125 MHz timing reference to be supplied to all MACs and PHYs on the board.
- Use one or more MAC interfaces configured in MII, RMII, or SGMII modes for passing IEEE 1588 messages.

Fix plan: No plans to fix

IEEE1588-A001: Incorrect received timestamp or dropped/data corruption packet when 1588 time-stamping is enabled

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When timestamping is enabled for all packets arriving on an Ethernet port (TMR_CTRL[TE] = 1 and RCTRL[TS] = 1), the port may fail to properly recognize the timestamp point for received frames. As a result, the timestamp value for the frame may be larger than the correct value, or the frame may be dropped or data corruption may occur without error indication.

Impact: For all modes except RMII, the timestamp value for a received frame may be larger than the correct value, or the frame may be dropped.

This erratum does not affect the operation of the TRIGGER_IN inputs, ALARM outputs, or FIPER outputs.

This erratum does not affect the operation of an Ethernet port when time-stamping is disabled (TMR_CTRL[TE]=0 and RCTRL[TS]=0); HRESET default is disabled.

Workaround: There is no workaround for the issue other than disabling receive time-stamping (RCTRL[TS]=0).

Fix plan: No plans to fix

General16: Enabling I^{2}C could cause I^{2}C bus freeze when other I^{2}C devices communicate

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When the I^{2}C controller is enabled by software, if the signal SCL is high, the signal SDA is low, and the I^{2}C address matches the data pattern on the SDA bus right after enabling, an ACK is issued on the bus. The ACK is issued because the I^{2}C 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 I^{2}C bus to freeze. However, it happens very rarely due to the need for two conditions to occur at the same time.

Impact: Enabling the I^{2}C controller may cause the I^{2}C bus to freeze while other I^{2}C devices communicate on the bus.

Workaround: Use one of the following workarounds:

- Enable the I^{2}C controller before starting any I^{2}C communications on the bus. This is the preferred solution.
- If the I^{2}C controller is configured as a slave, implement the following steps:
  a. Software enables the device by setting I2CnCR[MEN] = 1 and starts a timer.
  b. Delay for 4 I^{2}C bus clocks.
  c. Check Bus Busy bit (I2CnSR[MBB])

```
if MBB == 0
     jump to Step f; (Good condition. Go to Normal operation)
else
     Disable Device (I2CnCR[MEN] = 0)
```

  d. Reconfigure all I^{2}C registers if necessary.
  e. Go back to Step a.
  f. Normal operation.

Fix plan: No plans to fix

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

General18: Machine Model (MM) Electrostatic Discharge (ESD) criteria not met by certain SerDes pins

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Machine Model (MM) testing shows that some SerDes pins do not meet the 200/175 V MM ESD criteria. All pin combinations pass machine model stress at 150 V with no failures. The list of affected pins is limited to the following:

- L1_SD_IMP_CAL_RX
- L1_SD_IMP_CAL_TX
- L1_SD_REF_CLK
- L1_SD_REF_CLK_B
- L1_SD_RXA_N
- L1_SD_RXA_P
- L1_SD_RXE_N
- L1_SD_RXE_P
- L1_SD_TXA_N
- L1_SD_TXA_P
- L1_SD_TXE_N
- L1_SD_TXE_P
- L2_SD_IMP_CAL_RX
- L2_SD_IMP_CAL_TX
- L2_SD_REF_CLK
- L2_SD_REF_CLK_B
- L2_SD_RXA_N
- L2_SD_RXA_P
- L2_SD_RXE_N
- L2_SD_RXE_P
- L2_SD_TXA_N
- L2_SD_TXA_P
- L2_SD_TXE_N
- L2_SD_TXE_P

Impact: None

Workaround: Ensure equipment used in handling of the device is properly grounded. Ensure parts are handled in an environment that is compliant with current ESD standards.

Fix plan: No plans to fix

JTAG6: Boundary scan test on SerDes pins unreliable

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The boundary scan registers related to the SerDes pins may be unreliable.

Impact: When performing the IEEE 1149.1 boundary scan operations, the SerDes boundary registers can drive different value than the data shifted into these registers in rare cases. The wrong data will appear on TDO while shifting out the information and on the SerDes pins when executing EXTEST or CLAMP commands.

Workaround: Do not rely on the value of the SerDes boundary scan registers and output signals.

Fix plan: No plans to fix

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: MPC8379E, MPC8378E, MPC8377E

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: MPC8379E, MPC8378E, MPC8377E

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

PCI20: PCI controller may hang when returning from "PCI pins low" state

Description: Devices: MPC8379E, MPC8378E, MPC8377E

When PCI_GCR[PPL] is set, the output and bidirectional PCI bus signals are forced low to allow other PCI bus clients to switch off their power supply, putting the PCI controller into the “PCI pins low” state. When returning to an operational state (clearing the PCI_GCR[PPL] bit), the PCI controller may remain in a PCI “non-idle” state and not be able to perform any further PCI transactions.

The sequence of the failure is as follows:

1. While the PCI bus is inactive, all external bus requests to the PCI arbiter are blocked (by setting PCI_GCR[BBR]), and PCI pins are pulled-low (by setting PCI_GCR[PPL]) to enter “PCI pins low” state.
2. In “PCI pins low” state, the PCI controller state-machine still remains active. It is actively tracking the bus signals and is expecting them to meet the AC timing specifications.
3. When PCI_GCR[PPL] = 0, the PCI control signals start to rise to “1” logic, pulled by the bus pull-up resistors only.
4. The AC timing specifications of the rising signals at this moment may not always be met, causing a timing violation to occur and, in turn, causing the PCI controller to hang.

Impact: The PCI controller may hang when attempting to return from the “PCI pins low” state.

Workaround: Use one of the following options:

- Before Setting PCI_GCR[BBR] and PCI_GCR[PPL] to enter “PCI pins low” state, Clear PCI command bit1 to disable PCI Memory Space. While coming back from “PCI pins low” state, after clearing PCI_GCR[PPL] = 0, read back the PCI_GCR[PPL] to ensure the operation was completed. Immediately afterward, turn off the clocks to the PCI controller (clearing SCCR[PCICM]) to ensure that the PCI controller will stop during the slow rise time of the PCI signals. After waiting up to 10 microseconds, turn the PCI clocks ON and resume the operations to enable the full functionality of PCI bus. Resume the sequence to enable the PCI bus to return to full functionality. At last, Set PCI Command bit1 to enable PCI memory space.
- Connect a wire between an unused GPIOn[x] pin and the FRAME signal. Make sure that this GPIOn[x] is an input while HRESET is active and after its negation (GPnDIR[x] = 0). During the resume procedure, before clearing PCI_GCR[PPL], set the GPIOn[x] data register to '0' (GPnDAT[x] = 0), and then set GPIOn[x] to be an output signal (GPnDIR[x] = 1). Only now, clear PCI_GCR[PPL]. Wait up to 10 microseconds, allowing the PCI signals (except FRAME) to return to a steady state. Set the GPIOn[x] to '1' (GPnDAT[x] = 1), then set GPIOn[x] as an input (GPnDIR[x] = 0). Resume the sequence to enable the PCI bus to return to full functionality.

Fix plan: No plans to fix

PEX1: No support of PCI Express completions with BCM bit set (PCIX bridge interface)

Description: Devices: MPC8378E, MPC8377E

To satisfy certain PCI-X protocol constraints, a PCI-X Bridge or PCI-X Completer for a PCI-X burst read in some cases will set the Byte Count field in the first PCI-X transaction of the Split Completion sequence to indicate the size of just that first transaction instead of the entire burst read. When this occurs, the PCI-X Bridge/PCI-X Completer will also set the BCM bit in that first PCI-X transaction, to indicate that the Byte Count field has been modified from its normal usage. A PCI Express Memory Read Requester needs to correctly handle the case when a PCI-X Bridge/PCI-X Completer sets the BCM bit. PCI Express Completers will never set the BCM bit.

The device does not expect the BCM to be set. A packet received with the BCM bit will cause unexpected behavior.

Impact: The device does not support the BCM behavior described in the PCI Express standard, and may not support PCI Express to PCI-X bridges.

Workaround: None

Fix plan: No plans to fix

PEX2: DMA Interrupt descriptor race condition (IDRC)

Description: Devices: MPC8378E, MPC8377E

The DMA DONE bit is set and issues an interrupt before the buffer descriptor (BD) is written to memory.

Impact: After detecting the interrupt, the software may read wrong data from the descriptor status.

Workaround: When serving the interrupt, Software must check that the done bit in the chain descriptor is set. If it is not set, software should poll the DONE bit until it is set.

Fix plan: No plans to fix

PEX7: Recovery from hot reset or link down

Description: Devices: MPC8378E, MPC8377E

The PCI Express CSB bridge enters a suspend mode as a result of a hot reset or link down event in either endpoint (EP) or root complex (RC) mode. When this happens, the PCI Express controller flushes all outstanding CSB transactions and does not accept new inbound transactions involving CSB and ATMU translation until the CSB bridge is unlocked, even though the link is automatically retrained properly and is stable in L0.

The CSB bridge cannot be unlocked without a soft reset. Software running on the local CPU must wait until the PCI Express CSB bridge is idle before performing a soft reset as well as re-enabling and reconfiguring the CSB bridge registers.

Further, even in RC mode, the PCI configuration space also gets reset as a result of a link down event (and not a hot reset event). This space needs to be reprogrammed during recovery.

A link down scenario can be caused by various reasons, and could occur even during the link negotiation phase of the initial link training coming out of a power-on reset. In this case, the LTSSM can temporarily go to L0, re-train after a quick link down, and then reach a stable link-up L0 state.

Note: If the MPC8378E/MPC8377E initiated the link recovery, this is not considered a link down event and the PCI Express CSB bridge does not enter suspend mode. In this case, the device is fully operational when the link is back to L0 state and there is no need for a soft reset.

Impact: When the system is in EP mode, the system will fail if the link goes up and the RC issues transactions involving CSB and ATMU translation to the endpoint before the PCI Express CSB bridge is unlocked.

Additionally, reconfiguring the CSB bridge registers and PCI configuration space (when in RC mode during a link down event) can complicate and slow down the software.

Finally, standard software drivers (such as Linux) do not expect that the PCI Express interface to be locked upon link retraining and therefore require special consideration.

Workaround: To recover from the above mentioned conditions, the following steps should be taken by the software running on the device.

When in EP mode:

1. In order to be interrupted when detecting the hot reset or link down, set the PEX_CSMIER[RSTIE] bit.
2. When the controller identifies a hot reset or link down event, the PEX_CSMISR[RST] bit gets set, and an interrupt is generated if the PEX_CSMIER[RSTIE] bit was set.
3. The software should immediately clear the CFG_READY bit.
4. Write one to clear PEX_CSMISR[RST].
5. Poll IBPIORP and WDMARP bits in PEX_CSB_STAT register (0x81c) to complete all outstanding transactions.
6. Perform a soft reset to the PCI Express CSB bridge by clearing and re-setting the PECRn[CBRST] bit (100ns between clearing and re-setting is sufficient).
7. Reset to the PCI Express CSR space by clearing and re-setting the PECRn[CSR_RST] bit (100ns between clearing and re-setting is sufficient).
8. Reprogram the PCI Express CSB bridge registers. All configuration register bits that are non-sticky are reset and need to be reprogrammed (offset 0x800-0xFFC).
9. Set the CFG_READY bit.

When in RC mode after a link down event occurs:

1. In order to be interrupted when detecting the link down, set the PEX_CSMIER[RSTIE] bit.
2. When the controller identifies a link down event, the PEX_CSMISR[RST] bit gets set, and an interrupt is generated if the PEX_CSMIER[RSTIE] bit was set. Note that this bit is not set for a hot reset event when the MPC8378E/MPC8377E is in RC mode.
3. Write one to clear PEX_CSMISR[RST].
4. Poll IBPIORP and WDMARP bits in PEX_CSB_STAT register (0x81c) to complete all outstanding transactions.
5. Perform a soft reset to the PCI Express CSB bridge by clearing and re-setting the PECRn[CBRST] bit (100ns between clearing and re-setting is sufficient).
6. Reprogram the PCI Express CSB bridge registers. All configuration register bits that are non-sticky are reset and need to be reprogrammed (offset 0x800-0xFFC).
7. Reconfigure the PCI configuration space.

When in RC mode after a hot reset event occurs:

1. Poll IBPIORP and WDMARP bits in PEX_CSB_STAT register (0x81c) to complete all outstanding transactions.
2. Perform a soft reset to the PCI Express CSB bridge by clearing and re-setting the PECRn[CBRST] bit (100ns between clearing and re-setting is sufficient).
3. Reprogram the PCI Express CSB bridge registers. All configuration register bits that are non-sticky are reset and need to be reprogrammed (offset 0x800-0xFFC).

The PCI Express controller is now ready for normal transactions.

Fix plan: No plans to fix

PEX9: Excess correctable errors in receiving DLLPs and TLPs on x2 link

Description: Devices: MPC8378E, MPC8377E

The PCI Express specification allows for graceful recovery from bit errors on the interface through packet CRC error detection and recovery. The PCI Express controller, however, may flag a correctable error ("CED" in PCI Express device status register and "bad DLLP" or "bad TLP" in the PCI Express correctable error status register) based on an internal state error when receiving DLLPs and/or TLPs in x2 link width. When "bad TLP" correctable error status bit is flagged, the PCI Express controller NAKs the valid TLP received, which causes the remote PCI Express link partner to replay the TLP affected. In some cases, the frequency of these internal errors may be high enough to cause replay timeouts in the remote link partner. However, there is no link down being noticed.

Because DLLPs as well as TLPs can be affected, the errors may occur even if the controller is not receiving or transmitting TLPs (for example, during flow control initialization or updates).

This problem may occur when the PCI Express Controller is running at frequencies above 200 MHz. Note that there is no direct dependency on the CSB frequency. The problem only applies to the PCI Express controller 1, as the PCI Express controller 2 can only work in x1 link width.

Impact: PCI Express controller may observe excess correctable errors when trained as a x2 link. A PCI Express controller performance degradation of up to 15% may be observed.

Workaround: Use the PCI Express controller at frequencies of 200 MHz or below.

Fix plan: No plans to fix

PCIe-A002: PCI Express Packets may be transmitted with excess data on x1 link

Description: Devices: MPC8378E, MPC8377E

An internal error in the PCI Express controller may cause 8 bytes of packet header or data payload to be duplicated on transmit. Depending on the location of the duplicate bytes, this may cause a malformed packet error or CRC error detected in the link partner.

Impact: A PCI Express link partner may see excess correctable errors or malformed packets in x1 links.

Note that any PCI Express specification-compliant recipient at the ultimate end of the transaction may only recognize the erroneous packet as a Bad TLP and flag the error as correctable due to the LCRC error. It should respond with a NAK and then wait for the device to re-try the packet. The ultimate recipient should not recognize the erroneous packet as a Malformed TLP, since before reaching the transaction layer the Bad TLP should have been discarded by the ultimate recipient's link layer.

Workaround: None

Fix plan: No plans to fix

RESET3: External Soft reset functionality is not functional

Description: Devices: MPC8379E, MPC8378E, MPC8377E

External Soft reset is defined such that the DDR controller and the local bus controller will not be reset in the event of an external soft reset. However, the internal bus masters and the internal bus components will be held in reset immediately and as long as external soft reset event is valid and the soft reset sequence is in progress. As a result, if the soft reset event happens in the middle of an outstanding write transaction which is targeted to the DDR main memory or to one of the local bus slaves, the internal bus component will transition to reset state while it needs to supply the data to be written and as a result wrong data could be written to the main memory or to the local bus slave.

External Soft reset is defined such that it has no effect on system configuration registers, including IMMRBAR. As a result, software should behave different in regards to IMMRBAR initialization when recovering from soft reset as opposed to recovering from hard reset (in which the IMMRBAR is guaranteed to be located in its default mapping). Since software cannot tell if it is recovering from soft reset or hard reset without reading the platform's RSR register, and since software has to know the IMMRBAR value in order to read the RSR register, there is no way to recover from soft reset (unless the IMMRBAR is kept in its default value at all time).

Impact: Recovery from a soft reset event is not guaranteed.

Workaround: Do not use soft reset at all. Use hard reset or power on reset. SRESET signal must be used as an output-only signal. Software must not write 1 to the RCR[SWSR] bit.

Fix plan: No plans to fix

SATA2: Reads of one sector from hard disk may return less data than requested

Description: Devices: MPC8379E, MPC8377E

When the SATA controller issues a read DMA command and initiates a read of 512 bytes of data (one sector), a faulty hard disk may send less than the requested amount of data with a valid CRC and end the transmission of the data FIS, instead of sending the entire 512 bytes.

In this scenario, the SATA controller should perform a check for the total number of bytes it receives, as its DMA is working to write 512 bytes into memory but has received fewer bytes than that. However, it does not perform a check.

Instead, the SATA controller sets the command n completed (CCn) bit in the CCR (Command Completed Register) corresponding to the command n queue (CQn) bit in the CQR (Command Queue Register). This results in the HSTatus[CC] bit being set. No error bits are set in the HStatus register.

Impact: This would be of particular concern when more than one sector is being read from the device and the device sends fewer sectors than it should have sent.

Workaround: If the hard disk provides less data than requested by a command and then terminates that command with a good status FIS, then the command would be marked as complete.

In this, there are two situations:

1. If the hard disk indicates that the frame is transferred with error, the SATA controller does not have a mechanism to detect the length mismatch. In this scenario, the error detect mechanism will have to be implemented in customer's software.
2. If the hard disk indicates that the frame is transferred with good status:
   a. If the data contain correct CRC, the SATA controller does not have a mechanism to determine the transfer length error. In this scenario, the error detect mechanism will have to be implemented in customer's software.
   b. If the data contain incorrect CRC, then SError[T] should be set along with corresponding Command Error bit.

Fix plan: No plans to fix

SATA3: SATA controller hangs when handling data integrity errors

Description: Devices: MPC8379E, MPC8377E

The SATA controller hangs when it faces data integrity errors due to CRC/Disparity/Decoding errors that cause changes in the host controller‘s internal state either due to DMA or through the PIO. When this happens, it must be brought offline and then online again to resume normal operation.

When a read/write DMA command is issued to the device and the driver/responder portions of the verification environment are configured to respond with a CRC/Decode/Disparity error, the command is issued in any one of the sixteen available command queue slots. The host detects the error and indicates the device error by setting bit 0 (DE0) of the DER (device error register) because only one device is connected to it. The host also sets the command error bit (CEn) for that particular command by setting the appropriate bit of the CER (command error register). However, the command complete bit setting always points to the slot zero command, even if command zero passed successfully.After this SATA controller hangs and does not issue any new commands.

Impact: The expected impact is expected to be minor as this is a rare scenario; under such a scenario, the SATA host will reset the link between the disk and the device, and software will just issue the commands again.This may impact performance if the software keeps track of the commands that were successfully completed and only issues bad commands again.

Workaround: None

Fix plan: No plans to fix

SATA9: Fails the SATA InterOperability Sinusoidal Jitter test

Description: Devices: MPC8379E, MPC8377E

SATA standard does not provide any specifics on Sinusoidal Jitter testing.

Sinusoidal Jitter test interpretation at several specific frequencies was added only to the SATA InterOp testing revisions 1.2 and above. The SATA ports will not be error free when presented with high levels of sinusoidal Jitter done in the SATA InterOp testing.

Impact: The SATA port on this device is not compliant to the SATA InterOperability testing revisions 1.2 and higher.

Workaround: None

Fix plan: No plans to fix

SATA12: SATA: DMAT handling in SATA not as expected

Description: Devices: MPC8379E, MPC8377E

According to the SATA specification, during a DMA write to a device, the device may send a DMA Terminate (DMAT) primitive back to the host to abort the transmission. Upon receiving a DMAT primitive, the host should cease transfer by deactivating its DMA engine and completing the frame by sending a valid CRC and an EOF primitive. The host DMA engine will save its state at the point it was deactivated and at a later time, the device may choose to resume the transfer by transmitting another DMA Activate FIS or close it entirely.

However, when the device reissues the DMA Activate FIS to resume operation, the host does not resume normal operation.

Impact: On the reception of the DMA Activate FIS to resume the operation, the Host does not resume the DMA transfer from the point it stopped the last DMA operation. Instead, the Host is in a hang state as it does not have any context saved from previous transactions to resume the DMA transfer. It waits for either an intervention from the device or from software. If the device sends a Status FIS signaling Error, then the host will terminate the transaction. If the device does not send any additional FIS after the DMA Activate FIS, the host will wait until software interferes due to command timeout. The Host waits for the intervention of the software to service the interrupt and then reissues the pending commands after resetting the host-device link. This may have an impact on the performance of the Host Controller provided excessive DMAT primitives are issued by the device.

Workaround: As the software stores the context of all the issued commands, it can reset the link and resume the operation from the point of interrupt.

Fix plan: No plans to fix

SATA13: BIST-L mode is not enabled by BIST-L FIS

Description: Devices: MPC8379E, MPC8377E

The SATA controller cannot be put into BIST-L mode by BIST-L FIS.

Impact: The impact is not significant as alternative methods can be used to put SATA controller into BIST-L mode.

Workaround: Use the internal PHY control register to put SATA controller in BIST-L mode.

In order to configure the PHY Control Register, software must do the following:

1. Set the SATA in BIST-L mode by writing 0x0000_2902 to offset 0x15C (PHY Control Configuration Register1). Note that the value is shown from the register perspective (byte-swapped).
2. Reset and set the SerDes to SATA operations.
3. Enable the SATA controller by writing 0x8000_0000 to offset 0x02C (Host Control Register). Note that the value is shown from the register perspective (byte-swapped).

Fix plan: No plans to fix

SATA-A002: ATAPI commands may fail to complete

Description: Devices: MPC8379E, MPC8377E

When ATAPI drives, such as DVD or CD-ROM drives, are connected, commands that require setting the ATAPI ‘A’ bit (bit 5 of Word3) in the command header may fail to complete in the following scenario.

1. The drive is connected directly to SATA port and command slot other than 0 is used to issue ATAPI command
2. The drive is connected via PM (Port Multiplier) port and the command slot number is different from the PM port number on which ATAPI device is connected

This is because SATA controller misassigns the command number

Impact: Controller may not be able to detect or access ATAPI drives.

Workaround: For drive connected directly to SATA port:

- Use command slot 0 to issue ATAPI commands to ATAPI drives

For drives connected through PM Port:

- Use the command slot that matches the port number on which the ATAPI device is attached for command execution. For example, if an ATAPI device is attached to port 3 of the PM ports, use command slot 3 to send commands to this device.

Fix plan: No plans to fix

A-005035: Possible data loss if PRD[DBA] or PRD[DWC] is not at least 16-byte aligned

Affects: SATA

Description: Devices: MPC8379E, MPC8377E

SATA controller by design always tries to reach 64 byte alignment. If the PRD[DBA] or PRD[DWC] is not 64 byte aligned, then in the beginning or the end of transaction, it has to use 32-byte, 16-byte, 4-byte size transaction.

Impact: SATA controller may lose data when it tries to do 4-byte transaction.

Workaround: Software must make sure both PRD[DBA] and PRD[DWC] are at least 16-byte aligned so that the SATA controller would avoid 4-byte access.

Fix plan: No plans to fix

A-005255: Failure to detect single SYNC primitive

Affects: SATA

Description: When a single SYNC primitive is sent between the WTRM and XRDY primitives, the host controller may fail to detect this primitive. This can cause the host controller link state machine to remain in the "ending" state. As a result, the controller will not be able to return to the "idle“ state, will not respond to the XRDY primitive stream and will not set corresponding command completion event.

Impact: SATA devices that send a single SYNC primitive after a read transaction may fail proper operation. However, there are many SATA devices will send more than one SYNC primitive after a read transaction and are not impacted by this erratum.

Workaround: There is no workaround. If the device sends a single SYNC primitive and the host controller fails to detect it, the software will time out waiting for command completion event. To recover, the software can re-initialize the SATA Link by clearing HCONTROL[HC_ON] to 0x0 and then setting it to 0x1.

Fix plan: No plans to fix

A-005636: Auto-activate feature enabled in DMA setup command causes timeout

Affects: SATA

Description: When NCQ is enabled, the SATA controller does not support DMA setup FIS with auto-activate enabled from the device. The SATA host may timeout without finishing the transaction.

Impact: This will have a minor performance impact as disabling the auto-activate feature requires the device to send a DMA setup as well as a DMA activate FIS to enable reception of the first data FIS.

Workaround: Software can work around this with one of the following options:

- Disable the DMA setup auto-activate feature by a set features command.
- Or, disable NCQ by setting the queue depth to one.

Fix plan: No plans to fix

A-006187: Intermittent, non-recoverable, transient data integrity error seen when accessing SSC-enabled SATA drives

Affects: SATA

Description: The SATA standard allows for an optional spread spectrum clocking (SSC) of the transmitter clock to lower EMI emission. Intermittently, when reading from SATA drives with SSC enabled, the host controller may report that a data integrity error has occurred that is not recovered by the interface. The following bit fields in the SATA error register may be set:

- SATA interface error register, SError[bits CRC, D, B, E, T]
- Command error register, CER[bit CEn]
- Device error register, DER[bit DEn]

Impact: SATA read transactions fail intermittently. This error may cause the SATA controller to hang.

Workaround: Disable SSC or hard reset the device when encountering the error.

Fix plan: No plans to fix

SEC9: Protocol descriptor hang conditions

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The SEC is designed to perform single pass encryption and integrity checking as required for security protocols including IPSec and SSL/TLS. When using the descriptor types required for single pass IPSEC and SSL/TLS, where a primary EU is selected, and no secondary EU is selected, the channel hangs rather than producing an illegal descriptor header error.

Impact: Channel hangs if no secondary EU is selected for single pass IPSEC and SSL/TLS. There are two specifically observed hang conditions, as follows:

1. IPSec inbound without a secondary EU (Outbound, with or without a secondary EU is fine; inbound with a secondary EU is fine). The channel behavior is not predictable and may hang.
2. TLS_SSL_STREAM inbound if the primary EU is AFEU and there is No secondary EU, the channel hangs.

Workaround: Single pass protocol descriptors are designed for simultaneous encryption and integrity checking, and operation which required two EUs. For IPSec or SSL with integrity only, other descriptor types should be used. Note that this errata does not impact IPSec with AES-GCM (a use of IPSec with a single primary EU). IPSec with AES-GCM uses a different descriptor type.

Fix plan: No plans to fix

SEC10: AES-GCM IV Length Restriction

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Like most other modes of AES, AES-GCM uses a key and an IV to transform plaintext to ciphertext. The length of the IV is standardized by multiple independent bodies. The base AES-GCM specification allows the use of any size of IV or AAD or plaintext/ciphertext.

Other standards bodies, including the IEEE and IETF, define security protocols which use AES-GCM, such as MACSec (IEEE Std 802.1ae™-2005), and IPSec. The specifications for the use of AES-GCM within these security protocols restricts the size of the IV, AAD, and plaintext/ciphertext.

The AESU in the SEC performs AES-GCM without any issue whenever the IV is 96 bits, however the AESU fails under the following conditions:

1. The IV length is not 96 bits.
2. The initial IV Ghash operation produces a value with the 32 least significant bits all 1's

Impact: Both IPSec and MACSec (IEEE 802.1ae) specify an IV length of 96 bits, which allows the SEC to support these protocols without restriction. The only known concern associated with this errata comes from the current draft standard for P1619.1 (Draft Standard for Authenticated Encryption with Length Expansion for Storage Devices), which permits IVs other than 96 bits in length. Because there is no way to predict whether the initial IV Ghash operation produces a value with the 32 least significant bits all 1's, the SEC should not be used for AES-GCM in P1619.1, or any other protocol which does not limit IV length at 96 bits.

Workaround: For protocols specifying 96b IVs, no work around is necessary. For protocols specifying other IV lengths, none is possible.

Fix plan: No plans to fix

SEC11: Performance counter register access requirement

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The SEC Poly Channel contains four registers that allow software to quickly obtain statistics related to SEC performance. All four of these registers can only be read with a single 64-bit read.

Table 4. SEC Poly Channel Registers

Fetch FIFO Enqueue Count
Descriptor Finished Count
Data Bytes In Count
Data Bytes Out Count

Impact: Separate 32-bit reads of the upper and lower portions of the registers will cause the operation to fail.

Workaround: When accessing these registers, only use 64-bit reads.

Fix plan: No plans to fix

SEC12: TLS_SSL_Block_Inbound HMAC Error

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The SSL/TLS protocol allows for the use of several different cipher suites to encrypt and integrity check payload data. The encryption cipher can be a stream cipher like RC-4, or a block cipher like 3DES or AES.

The SSL/TLS processing steps for out-bound operations with a block cipher are as follows:

1. Calculate HMAC over the SSL record header and payload.
2. Add HMAC to the end of the payload.
3. Calculate the number of bytes of payload and HMAC, plus a Pad Length byte, and add padding bytes to the record until the total number of bytes (payload||HMAC||padding||Pad Length) is an integral number of cipher blocks.
4. Encrypt the record, starting at the first byte of payload, ending at the Pad Length (Pad Length is included in the encryption).
5. Change the Record Header Length field to match the length of payload||HMAC||padding|| Pad Length

This order of operations is opposite to most other security protocols, but does not present any special challenges to single pass hardware accelerators.

For inbound, the order of operations is almost the opposite:

1. Decrypt the record, starting at the first byte of payload, ending at the Pad Length (Pad Length is included in the decryption).
2. Calculate the number of bytes of payload by subtracting the length of the HMAC, padding, and pad length from the Length field received with the record header.
3. Change the length field to match the payload length only. Remove the padding and pad length bytes.
4. Calculate HMAC over the SSL record header and decrypted payload.
5. Compare the received HMAC with the calculated HMAC, and if different, drop the record.

This order of operations presents a significant challenge to single pass hardware. It isn't possible to change the length field in the record header until the Pad Length byte has been decrypted, and if the length field isn't changed prior to inbound integrity checking, the calculated HMAC are wrong. The SEC implementation did not comprehend that the Length field would not be modified by software prior to launching the descriptor, as is done for out-bound processing.

Impact: TLS_SSL_Block_Inbound operations produce incorrect HMAC values.

Workaround:

- Option 1: For TLS_SSL_Outbound operations, use the SEC's single pass descriptor type 1000_1 for maximum performance. For TLS_SSL_Inbound, use two descriptors:
  - First descriptor: Use type 0001_0 'common non-snoop', with header set for the appropriate block cipher algorithm, to decrypt the payload through Pad Length. Upon completion of this descriptor, software changes the value of the Length field in the record header and launches the second descriptor.
  - Second Descriptor: Type 0001_0 'common non-snoop' set for the appropriate HMAC algorithm (for example, HMAC-SHA-1). The pointers in this descriptor tell the SEC to calculate an HMAC over the SSL Record Header||Payload. The SEC can be set to automatically compare the generated HMAC against the received HMAC, or the descriptor can write the generated HMAC to memory for a comparison by software.

- Option 2: For TLS_SSL_Outbound operations, use the SEC's single pass descriptor type 1000_1 for maximum performance. For TLS_SSL_Inbound, use a two-descriptor method that minimizes the total amount of data the SEC must read to generate a decrypted and authenticated record:
  - First descriptor: Use type 0001_0 'common non-snoop', with header set for the appropriate block cipher algorithm. Assuming CBC mode, decrypt the final block of the record, using as the IV the second-to-last block of the record. Do not write the decrypted data back to the memory location of the record. The purpose of decrypting the final block is to recover the Pad Length field, which is the last byte of the output generated by this descriptor. After you have the Pad Length, discard the decrypted data. Upon completion of this descriptor, software subtracts the length of the HMAC (a known constant value), padding, and pad length from the Length field, updates the Length field in the record header and launches the second descriptor.
  - Second Descriptor: Type 1000_1 TLS_SSL_Block. Because the Length field is set properly, the single pass TLS_SSL_Inbound descriptor properly decrypts and authenticates the record.

Fix plan: No plans to fix

SEC14: Non-compliant implementation of deterministic pseudo-random number generator

Description: Devices: MPC8379E, MPC8378E, MPC8377E

There are two types of random number generators: true random number generators (True-RNG) and deterministic pseudo-random number generators (Pseudo-RNG).

- True-RNGs use an enviromental stimulus, such as the impact of temperature and voltage fluctuations on a ring oscillator, and combine it with feedback circuitry to produce a truly unpredictable random number.
- Deterministic Pseudo-RNGs use cryptographic algorithms, such as RSA or SHA, to produce a stream of random values from an initial seed value. Even if an attacker knows the method being used in a deterministic Pseudo-RNG, he cannot predict the next value because he does not know the seed value.

Although both types of RNGs can produce random numbers for a variety of cryptographic uses, the United States National Institute of Standards & Technology (NIST) only certifies deterministic Pseudo-RNGs. This is due to the inherent difficulty in proving that a True-RNG never outputs non-random data.

The RNG-B implements both a True-RNG and a deterministic Pseudo-RNG, based on SHA as described in FIPS 186-2, Appendix 3.1. This errata is due to the deterministic Pseudo-RNG implementation reversing the byte ordering associated with the XKEY operation described in FIPS 186-2, Appendix 3.1. Consequently, starting from a known seed, the RNG-B generates an equivalent-strength random output; however, this output is not what is defined for FIPS 182-2, Appendix 3.1 certification.

Impact: The impact of this errata is the system's inability to pass NIST RNG certification using the direct output of the RNG-B. Whenever NIST certification or use of NIST-compliant methods are required, the user must rely on a software implementation of a NIST approved deterministic Pseudo-RNG, with the associated software overhead. Generally NIST-compliant random numbers are only required for key generation/key exchange operations, which are relatively infrequent. The RNG-B can be used to seed the software deterministic Pseudo-RNG.

Workaround: As mentioned above, the RNG-B can be used to seed a software deterministic Pseudo-RNG when NIST-compliant random numbers are required. In the more common use cases, such as generation of random values to be used as Initialization Vectors (IVs) for security protocols such as IPsec, there is no requirement for NIST-compliant random numbers. In these cases, the RNG-B output can be used directly, with lower software overhead.

Fix plan: No plans to fix

SEC15: Limitations on Custom Modes of CRC

Description: Devices: MPC8379E, MPC8378E, MPC8377E

The CRCU supports 2 standard CRCs, as well as custom CRCs in which the user defines the polynomial and bit manipulations associated with the polynomial. These bit manipulations are bit swapping, byte swapping, and complementing the output of the polynomial. Using the RAW bit in the CRCU Mode Register, the user controls whether all or none of the manipulations are performed.

The CRCU supports generation of correct CRC output for any polynomial that uses either the raw polynomial output, or an output that is bit swapped, byte swapped, and complemented. It does not support CRCs in which some bit manipulations are performed, but not others.

Impact: The OFDMA CRC used in Wimax is not supported, due to the fact that it requires some, but not all, bit manipulations.

Workaround: Perform the OFDMA CRC in software.

Fix plan: No plans to fix

SEC16: Kasumi hardware ICV checking does not work

Description: Devices: MPC8379E, MPC8378E, MPC8377E

SEC's execution units (EU) that generate integrity check values (ICVs) are also capable of comparing a received ICV with a calculated ICV. In the case of the KEU (Kasumi) F9 function, the SEC is not able to properly compare a received F9 MAC (another acronym for an ICV) with the calculated MAC.

Impact: The Kasumi algorithm produces a 128-bit integrity check value, which is shortened in the 3G protocol to a 64-bits message authentication code (MAC). To verify a received MAC, the user copies the received MAC to the KEU's IV data, which the SEC fetches (along with other parameters) in order to generate the calculated MAC. The KEU is supposed to compare the upper 64 bits of the calculated MAC with the 64 bits received MAC; however, it attempts to compare the full 128 bits and consequently always reports an ICV comparison failure, for example, integrity check comparison result (ICCR) returns binary "10."

Workaround: Users are not required to use the hardware ICV comparison feature. Rather than copying the 64 bits received MAC to the KEU's IV data, the user can merely have the SEC output the 128-bit MAC generated by the KEU, and software can perform the comparison of the upper 64 bits. Performing the MAC comparison in software takes only a few more CPU cycles than copying the received MAC to the IV data and reading the SEC's comparison result from the descriptor header.

Fix plan: No plans to fix

SEC-A001: Channel Hang with Zero Length Data

Description: Devices: MPC8379E, MPC8378E, MPC8377E

Many algorithms have a minimum data size or block size on which they must operate. The SEC EUs detect when the input data size is not a legal value and signal this as an error. For most EUs, a zero byte length data input should be considered illegal, however the EUs do not properly notify the crypto-channel using the CHA of a data size error. Instead, the EUs wait forever for a valid data length, leading to an apparent channel hang condition.

Impact: When EUs detect illegal input data size, EUs wait forever for a valid data length, leading to an apparent channel hang condition.

Workaround: Option 1: Ensure that software does not create SEC descriptors to encrypt or decrypt zero length data.

Option 2: Use the SEC crypto-channel watchdog timer to detect hung channels. This is accomplished by enabling each channel's watchdog timer via the Channel Configuration Register CCRx[WGN]. This is a one time configuration. The timer stops when the channel completes a descriptor and restarts at zero each time the channel fetches a new descriptor. The default setting for the watchdog timer is 2^27 SEC clock cycles. If the timer expires, the channel will generate an interrupt and the Channel Status Register will show the cause as a Watchdog Timeout [WDT]. The channel hang is cleared by resetting the channel via CCR[RST]. The offending EU is reset automatically by the channel.

Fix plan: No plans to fix

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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: Devices: MPC8379E, MPC8378E, MPC8377E

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

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

Affects: USB

Description: Devices: MPC8379E, MPC8378E, MPC8377E

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

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:	Devices: MPC8379E, MPC8378E, MPC8377E 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