Freescale Semiconductor
Errata

Document Number: MPC860CE
Rev. 1.9, 05/2008

MPC860/855T Family Device Errata Reference

This document is a compilation of all MPC855T, MPC860, MPC860SAR, and MPC860T device errata from Revision A.2 forward. Herein, the errata are classified and numbered, and each erratum is provided with a description and workarounds in the second part of the document. To determine which silicon revision and mask set are associated with which version, please refer to Table 2. To find which errata apply to a particular device, please refer to the errata listings for each revision provided in Table 4. Indications of schedules for errata fixes are provided in the errata listing of the latest revision of each part.

Table 1 provides a revision history for this document.

Table 1. Document Revision History

Rev. No.	Date	Significant Change(s)
1.9	5/2008	Added ATM 11 errata information for D4 to Table 3. Replaced MPC860CESUMM_Rev1_7.doc with this one.

Table 2. MPC860CE Silicon Revision, Mask Set, and Version Key

Silicon Revision	Mask Set	855T	860	860DC	860DE	860DH	860DP	860DT	860EN	860MH	860P	860SR	860T
A.2	2F84C		√	√	√	√			√	√			
A.3	4F84C		√	√	√	√			√	√			

Table 2. MPC860CE Silicon Revision, Mask Set, and Version Key (continued)

Silicon Revision	Mask Set	855T	860	860DC	860DE	860DH	860DP	860DT	860EN	860MH	860P	860SR	860T
B.1	1J24A, 0H86G											√	
	2J24A, 9J24A		√	√	√	√			√	√			
B.2	2H56S							√					√
B.3	1J21M							√					√
B.5	3J21M							√					√
C.0	0H96G											√	
	1H96G		√	√	√	√			√	√			
C.1	2H96G											√	
	3H96G		√	√	√	√			√	√			
D.4 (D.3)	3K20A (2K20A)	√			√		√	√	√		√	√	√

Table 3. MPC860CE Errata and Silicon Revision Key

Erratum	A.2	A.3	B.1	B.1 (860SR only)	B.2	B.3	B.5	C.0	C.0 (860SR only)	C.1	C.1 (860SR only)	D.4 (D.3)	Workaround
GLL1	√	√	√	√	√	√	√	√	√	√	√	√	Y
SIU1	√	√											N
SIU2	√	√											Y
SIU3	√	√	√	√	√	√	√						Y
SIU4	√	√	√	√	√	√	√	√	√	√	√	√	Y
SIU5	√	√	√	√	√	√	√	√	√	√	√	√	Y
SIU6	√	√	√	√	√	√	√	√	√	√	√	√	Y
SIU7	√	√	√	√	√	√	√	√	√	√	√		Y
SIU8	√	√	√	√	√	√	√						Y
SIU9	√	√	√	√	√	√	√	√	√	√	√	√	Y
SIU10	√	√	√	√	√	√	√	√	√	√	√	√	N
CPM1	√	√											Y
CPM2	√	√											Y
CPM3	√	√	√	√	√	√	√						Y
CPM4	√	√	√	√	√	√	√	√	√	√	√		Y

Table 3. MPC860CE Errata and Silicon Revision Key (continued)

Erratum	A.2	A.3	B.1	B.1 (860SR only)	B.2	B.3	B.5	C.0	C.0 (860SR only)	C.1	C.1 (860SR only)	D.4 (D.3)	Workaround
CPM5	√	√	√	√	√	√	√	√	√	√	√		Y
CPM6	√	√	√	√	√	√	√	√	√	√	√		Y
CPM7	√	√	√	√	√	√	√	√	√	√	√		Y
CPM8	√	√	√	√	√	√	√	√	√	√	√		N
CPM9	√	√	√	√	√	√	√	√	√	√	√		N
CPM10	√	√	√	√	√	√	√	√	√	√	√		N
CPM11	√	√	√	√	√	√	√	√	√	√	√		Y
CPM12												√	Y
G1	√	√											Y
G2	√	√	√	√	√	√	√	√	√	√	√	√	Y
G3	√	√	√	√	√	√	√						Y
G4	√	√	√	√	√	√	√						Y
G5			√	√	√	√	√						Y
G6			√	√	√	√	√						Y
G7	√	√	√	√	√	√	√						Y
G8								√	√	√	√		Y
G9	√	√	√	√	√	√	√	√	√	√	√	√	Y
G10												√	N
CPU1	√	√	√	√	√	√	√	√	√	√	√		N
CPU2	√	√	√	√	√	√	√						N
CPU3	√	√											Y
CPU4	√												N
CPU5	√	√	√	√	√	√	√	√	√	√	√	√	Y
CPU6	√	√	√	√	√	√	√						Y
CPU7	√	√	√	√	√	√		√	√				Y
CPU8	√	√	√	√	√	√	√	√	√				Y
CPU9										√	√		Y
CPU10	√	√	√	√	√	√	√	√	√	√	√	√	Y
CPU11	√	√	√	√	√	√	√	√	√	√	√	√	Y
CPU12	√	√	√	√	√	√	√	√	√	√	√	√	Y
CPU13	√	√	√	√	√	√	√	√	√	√	√	√	Y

Table 3. MPC860CE Errata and Silicon Revision Key (continued)

Erratum	A.2	A.3	B.1	B.1 (860SR only)	B.2	B.3	B.5	C.0	C.0 (860SR only)	C.1	C.1 (860SR only)	D.4 (D.3)	Workaround
CPU14	√	√	√	√	√	√	√	√	√	√	√	√	Y
CPU15	√	√	√	√	√	√	√	√	√	√	√	√	Y
ATM1				√									Y
ATM2				√									Y
ATM3				√									Y
ATM4				√									Y
ATM5				√					√		√	√	Y
ATM6				√									Y
ATM7				√					√		√		Y
ATM8				√					√		√		Y
ATM9				√					√		√		Y
ATM10				√					√		√	√	Y
ATM11				√					√		√	√	Y
FEC1					√	√	√						N
FEC2					√	√	√						Y
FEC3					√	√	√					√	N
FEC4					√								N
FEC5					√	√	√						N
FEC6					√	√	√						N
FEC7					√	√	√						N
FEC8					√								Y
FEC9					√								Y
FEC10					√								Y
FEC11					√	√	√						N
FEC12					√	√	√					√	Y
FEC13					√	√	√						N
FEC14					√	√							Y
FEC15					√	√	√					√	Y

Errata List

Global Errata

GLL1 Some registers are not initialized correctly during power-on $\overline{\text{RESET}}$, $\overline{\text{HRESET}}$, and $\overline{\text{SRESET}}$

Table 4 is provided to clarify/correct the power-on $\overline{\text{RESET}}$ value of many registers and lists whether each register is affected by $\overline{\text{HRESET}}$ and/or $\overline{\text{SRESET}}$. The table applies to the MPC850, the MPC855T, the MPC857T, the MPC860, and the MPC862.

Table 4. Power-On $\overline{\text{Reset}}$ Value of the Registers

Register	Value at Power-On $\overline{\text{RESET}}$	Affected by $\overline{\text{HRESET}}$	Affected by $\overline{\text{SRESET}}$
SIUMCR	01200000	Yes	No
SYPCR	FFFFFF07	Yes	No
SWSR	0	Yes	Yes
SIPEND	0000xxxx	Yes	Yes
SIMASK	0000xxxx	Yes	Yes
SIEL	0000xxxx	Yes	No
SIVEC	(xx11)(11xx)xxxxxx	Yes	Yes
TESR	XXXX0000	Yes	Yes
SDCR	0	Yes	No
PBR0	x	No	No
POR0	x	No	No
PBR1	x	No	No
POR1	x	No	No
PBR2	x	No	No
POR2	x	No	No
PBR3	x	No	No
POR3	x	No	No
PBR4	x	No	No
POR4	x	No	No
PBR5	x	No	No
POR5	x	No	No
PBR6	x	No	No
POR6	x	No	No
PBR7	x	No	No
POR7	x	No	No
PGCRA	0	Yes	No
PGCRB	0	Yes	No
PSCR	x	No	No
PIPR	??00??00	Yes	Yes
PER	0	Yes	Yes
BR0	XXXXX(??00)0(000?)	Yes	No
OR0	00000FF4	Yes	No
BR1	XXXXXX(xx00)0	Yes	No

Table 4. Power-On $\overline{\text{Reset}}$ Value of the Registers (continued)

Register	Value at Power-On $\overline{\text{RESET}}$	Affected by $\overline{\text{HRESET}}$	Affected by $\overline{\text{SRESET}}$
OR1	XXXXXXX(xxx0)	Yes	No
BR2	XXXXXX(xx00)0	Yes	No
OR2	XXXXXXX(xxx0)	Yes	No
BR3	XXXXXX(xx00)0	Yes	No
OR3	XXXXXXX(xxx0)	Yes	No
BR4	XXXXXX(xx00)0	Yes	No
OR4	XXXXXXX(xxx0)	Yes	No
BR5	XXXXXX(xx00)0	Yes	No
OR5	XXXXXXX(xxx0)	Yes	No
BR6	XXXXXX(xx00)0	Yes	No
OR6	XXXXXXX(xxx0)	Yes	No
BR7	XXXXXX(xx00)0	Yes	No
OR7	XXXXXXX(xxx0)	Yes	No
MAR	x	No	No
MCR	(xx00)0(x000)0(xxx0)X(00xx)X	Yes	No
MAMR	xx001000	Yes	No
MBMR	xx001000	Yes	No
MSTAT	0	Yes	No
MPTPR	0200	Yes	No
MDR	x	No	No
TBSCR	0	Yes	No
TBREFA	x	No	No
TBREFB	x	No	No
RTCSC	00(000x)(000x)	Yes	Yes
RTC	x	No	Yes
RTSEC	x	No	Yes
RTCAL	x	No	No
PISCR	0	Yes	No
PITC	x	No	No
PITR	x	N/A	N/A
SCCR	0(000?)(?000)(0??0)0000	Yes	No
PLPRCR	???0(0100)000	Yes	Yes
RSR	0	Yes	Yes
TBSCRK	x	Yes	Yes
TBREFAK	x	Yes	Yes
TBREFBK	x	Yes	Yes
TBK	x	Yes	Yes
RTCSCK	x	Yes	Yes
RTCK	x	Yes	Yes
RTSECK	x	Yes	Yes
RTCALK	x	Yes	Yes
PISCRK	x	Yes	Yes
PITCK	x	Yes	Yes
SCCRK	x	Yes	Yes
PLPRCRK	x	Yes	Yes
RSRK	x	Yes	Yes

Table 4. Power-On $\overline{\text{Reset}}$ Value of the Registers (continued)

Register	Value at Power-On $\overline{\text{RESET}}$	Affected by $\overline{\text{HRESET}}$	Affected by $\overline{\text{SRESET}}$
I2MOD	0	Yes	Yes
I2ADD	x	No	No
I2BRG	FFFF	Yes	No
I2COM	0	Yes	Yes
I2CER	0	Yes	Yes
I2CMR	0	Yes	Yes
SDAR	x	No	No
SDSR	0	Yes	Yes
SDMR	0	Yes	Yes
IDSR1	0	Yes	Yes
IDMR1	0	Yes	Yes
IDSR2	0	Yes	Yes
IDMR2	0	Yes	Yes
CIVR	0	Yes	Yes
CICR	0	Yes	No
CIPR	0	Yes	Yes
CIMR	0	Yes	Yes
CISR	0	Yes	Yes
PADIR	0	Yes	No
PAPAR	0	Yes	No
PAODR	0	Yes	No
PADAT	x	No	No
PCDIR	0	Yes	No
PCPAR	0	Yes	No
PCSO	0	Yes	No
PCDAT	x	No	No
PCINT	0	Yes	No
PDDIR	0	Yes	No
PDPAR	0	Yes	No
PDDAT	x	No	No
TGCR	0	Yes	Yes
TMR1	0	Yes	Yes
TMR2	0	Yes	Yes
TRR1	FFFF	Yes	Yes
TRR2	FFFF	Yes	Yes
TCR1	0	Yes	Yes
TCR2	0	Yes	Yes
TCN1	0	Yes	Yes
TCN2	0	Yes	Yes
TMR3	0	Yes	Yes
TMR4	0	Yes	Yes
TRR3	FFFF	Yes	Yes
TRR4	FFFF	Yes	Yes
TCR3	0	Yes	Yes
TCR4	0	Yes	Yes
TCN3	0	Yes	Yes

Table 4. Power-On $\overline{\text{Reset}}$ Value of the Registers (continued)

Register	Value at Power-On $\overline{\text{RESET}}$	Affected by $\overline{\text{HRESET}}$	Affected by $\overline{\text{SRESET}}$
TCN4	0	Yes	Yes
TER1	0	Yes	Yes
TER2	0	Yes	Yes
TER3	0	Yes	Yes
TER4	0	Yes	Yes
CPCR	0	Yes	Yes
RCCR	0	Yes	No
RCTR1	NA	Yes	Yes
RCTR2	NA	Yes	Yes
RCTR3	NA	Yes	Yes
RCTR4	NA	Yes	Yes
RTER	0	Yes	Yes
RTMR	0	Yes	Yes
BRGC1	0	Yes	No
BRGC2	0	Yes	No
BRGC3	0	Yes	No
BRGC4	0	Yes	No
GSMR_L1	0	Yes	Yes
GSMR_H1	0	Yes	Yes
PSMR1	0	Yes	Yes
TODR1	0	Yes	Yes
DSR1	7E7E	Yes	Yes
SCCE1	0	Yes	Yes
SCCM1	0	Yes	Yes
SCCS1	0	Yes	Yes
GSMR_L2	0	Yes	Yes
GSMR_H2	0	Yes	Yes
PSMR2	0	Yes	Yes
TODR2	0	Yes	Yes
DSR2	7E7E	Yes	Yes
SCCE2	0	Yes	Yes
SCCM2	0	Yes	Yes
SCCS2	0	Yes	Yes
GSMR_L3	0	Yes	Yes
GSMR_H3	0	Yes	Yes
PSMR3	0	Yes	Yes
TODR3	0	Yes	Yes
DSR3	7E7E	Yes	Yes
SCCE3	0	Yes	Yes
SCCM3	0	Yes	Yes
SCCS3	0	Yes	Yes
GSMR_L4	0	Yes	Yes
GSMR_H4	0	Yes	Yes
PSMR4	0	Yes	Yes
TODR4	0	Yes	Yes
DSR4	7E7E	Yes	Yes

Table 4. Power-On $\overline{\text{Reset}}$ Value of the Registers (continued)

Register	Value at Power-On $\overline{\text{RESET}}$	Affected by $\overline{\text{HRESET}}$	Affected by $\overline{\text{SRESET}}$
SCCE4	0	Yes	Yes
SCCM4	0	Yes	Yes
SCCS4	0	Yes	Yes
SMCMR1	0	Yes	Yes
SMCE1	0	Yes	Yes
SMCM1	0	Yes	Yes
SMCMR2	0	Yes	Yes
SMCE2	0	Yes	Yes
SMCM2	0	Yes	Yes
SPMODE	0	Yes	Yes
SPIE	0	Yes	Yes
SPIM	0	Yes	Yes
SPCOM	0	Yes	Yes
PIPC	0	Yes	No
PTPR	0	Yes	No
PBDIR	xxx(xx00)0000	Yes	No
PBPAR	xxx(xx00)0000	Yes	No
PBODR	0	Yes	No
PBDAT	x	Yes	Yes
SIMODE	0	Yes	Yes
SIGMR	0	Yes	No
SISTR	0	Yes	No
SICMR	0	Yes	Yes
SICR	0	Yes	No
SIRP	0	Yes	Yes

Legend:

x or X = “don’t care” in either bits, nibbles, or the entire register.
0 = a single zero indicates the entire register is reset to zeros.
( ) = isolates bits of a nibble of the register.
? = a don’t care for POR, but if this register is affected by $\overline{\text{HRESET}}$ or $\overline{\text{SRESET}}$, indicates that the value will remain the same as what it was before the reset occurred.
NA = Not Applicable, indicates that this register has no POR value.

SIU Errata

SIU1 CLKOUT signal drive

Description:

The MPC860’s capability to change the strength of the output buffer drive of CLKOUT is not functional. The strength of the output buffer can be modified between the following two states:

- Full strength—COM = 00 or COM = 01 in the SCCR register
- Disabled—COM = 11

Workaround:

Do not use fractional drive capability.

SIU2 Data show cycle hang condition

Description:

If the DSHW bit in the SIUMCR register is set in order to allow data show cycles to take place on the external bus, and the CPM initiates a DMA transfer to/from an internal address (for example the dual port RAM), the chip will hang.

Workaround:

If DMA accesses are initiated to internal locations, reset the DSHW bit.

SIU3 Incorrect timing of PCMCIA slot B ALE signal

Description:

This erratum only affects designs that use latches for the address in their PCMCIA slot B interfaces.

The intended operation of the PCMCIA ALE signal is that ALE should assert with the address on the rising clock edge (plus the delay specified in Spec P52) and negate on the rising clock edge (plus the delay specified in Spec P53). However, for PCMCIA slot B, the ALE signal asserts and negates on the falling clock edge immediately prior to the intended edge.

This behavior shortens the address setup time for sampling in the external address latches used for PCMCIA slot B.

Workarounds:

Either of the following will serve as a workaround to this erratum:

- Verify address latch timing with this new timing information. If it is satisfied, there is no problem.
- Do not use external address latches for the PCMCIA slot B interface.

SIU4 Spurious external bus transaction following PLPRCR write

Description:

This erratum only affects designs that execute code from synchronous memories or bus slaves.

Spurious external bus transactions can occur after executing a store to the PLPRCR register, which changes the PLL multiplication factor (MF bits). This store causes the PLL to freeze the clocks while another external bus access is already visible on the pins of the chip. This appears externally as a transaction that begins, has its clocks frozen, and is abruptly aborted without following the bus protocol.

This behavior only affects systems with bus slaves that implement synchronous state machines that are sensitive to bus protocol violations. Synchronous DRAMs are not affected, and synchronous bus slaves that ignore bus signals when not selected (for example, Tundra QSPAN) are not affected.

This erratum will only cause problems in one of the following situations:

- The device is executing code from a slave that implements a state machine dependent on the 60x bus protocol, where that state machine might “get lost.”
- There is an external device that snoops the 60x bus and implements a state machine; this state machine might “get lost.”

The impact of this erratum has been deemed minimal, and it will therefore not be corrected.

Workaround:

The behavior exhibited in this erratum is a secondary symptom of the behavior of SIU9. See the description of SIU9 for the suggested workaround. The workaround described there will also avoid the above-described spurious external bus transaction.

SIU5 Bus monitor failure after $\overline{\text{TEA}}$ assertion

Description:

The bus monitor does not activate for the access immediately following an access that terminated with a TEA assertion (either internally or externally generated). Therefore, if the following access is to an unmapped address, the access will not be terminated by the MPC860. If this is allowed to occur, the cycle can potentially be endless, causing the system to hang.

The impact of this erratum has been deemed minimal, and it will therefore not be corrected.

Workaround:

Any of the following will serve as a workaround to this erratum:

- Avoid this situation (which should not occur in a normal system).
- Terminate the cycle externally (for example, implement an external bus monitor).
- Rely on the software watchdog timer to reset the system if this error occurs.

SIU6 Incorrect reporting of loss-of-lock reset status

Description:

The RSR[LLRS] bit is set by both unintentional and software-initiated loss-of-lock; it should be set only by an unintentional loss-of-lock. Software-initiated loss-of-lock (that is, changing the SPLL multiplication factor or entering low-power modes) should not set this bit.

The impact of this erratum has been deemed minimal, and it will therefore not be corrected.

Workaround:

The PLPRCR[SPLSS] functions as intended. Reference this bit instead.

SIU7 Missed DRAM refresh cycles with external masters

Description:

If the MPC860 is using internal arbitration (SIUMCR[EARB]=0), and the arbitration request level (SIUMCR[EARP]) for external masters is greater than zero, then if a request by an external master (signalled by $\overline{\text{REQ}}$) occurs simultaneously with a request from the DRAM refresh controller, the request from the DRAM refresh controller is cancelled. This results in a missed refresh cycle. In a system with many bus requests by external masters, this can potentially result in the cancellation of all DRAM refreshes.

Workaround:

Any of the following will serve as a workaround to this erratum:

- Program SIUMCR[EARP] to zero.
- Increase the refresh rate to compensate for the potential cancellation of refresh cycles. Treated probabilistically, it should be possible to keep the refresh rate above a minimum intended rate. This is difficult to model exactly but can be roughly estimated. For the following discussion:
  - N = proportion of bus bandwidth used by internal MPC860 masters (other than refresh)
  - E = proportion of bus bandwidth used by external masters
  - A = proportion of bus bandwidth available for refresh

  By definition, N + E + A = 1.

  The proportion of time that a refresh request can occur is (E+A).

  The probability that a refresh request will be cancelled is E / (E+A). If P is the probability that the refresh request will be successfully transacted, P = 1 - [E / (E+A)].

  Therefore, to compensate for the cancellation of requests, increase the refresh request rate by 1/P.

  For a numerical example, assume that internal and external masters each use 30% of the bus bandwidth. Thus, N = 0.3, E = 0.3, and A = 0.4. In this example, set the refresh rate to 1.75 times the intended rate.

  Note, however, that this workaround becomes impractical as A approaches zero.

- Implement a software-controlled refresh initiated by a periodic timer request. The user should program the PIT timer (or a CPM timer) to provide a periodic interrupt. The interrupt service routine should incorporate a software routine to refresh a memory block. This software refresh routine can consist of either reads from the appropriate DRAM page or, more simply, execution of the UPM's refresh routine through a RUN command to the MCR. The second method is recommended, as it is simpler and uses the DRAM's internal counter to keep track of the row to be refreshed. The user should choose the size of memory block to be refreshed per interrupt in order to minimize the impact of the interrupt overhead.

At one extreme, assume a system with two 4Mx32 DRAM banks controlled by $\overline{\text{CS2}}$ and $\overline{\text{CS3}}$. Each bank has 2048 pages (rows), and each page must be refreshed every 15.6 ms. If the UPM refresh pattern called by the software-refresh routine is set up to loop 16 times (and therefore can refresh 16 rows per call), the timer interrupt should occur every (16/2048)*15.6 ms, or approximately 120 µs. If one iteration of the UPM refresh pattern is 5 clocks, the total time required to execute the software-refresh routine (plus overhead for fetching instructions) for both banks is 5*16*2+20 = 180 clocks. Assuming an interrupt service routine entry/exit overhead of 1200 clocks, each refresh interrupt would take approximately 1400 clocks, or 28 µs (assuming a 50-MHz system clock). An ISR consuming 28 µs out of every 120 µs period would consume 23.3% of the CPU, with 8200 interrupts per second.

At the other extreme, the entire memory (2048 refresh cycles per bank) can be refreshed every 15.6 ms. In this case, the software refresh routine would require 1200+(2048/16*180) = 24240 clocks, or 485 µs. In this case, the ISR would consume 485 µs out of every 15.6 ms, or 3% of the CPU, and would require only 64 interrupts per second. However, system tasks would be stalled for 485 µs while waiting for the refresh task to complete.

The best compromise lies in between. For example, at 64 pages per interrupt, the software refresh routine consumes 1200+(64/16*18) = 1920 clocks, or 38.5 µs. The CPU bandwidth consumed would be 38.5 µs/(120 µs*4) or about 8%, with about 2000 interrupts per second.

Example code implementing this software refresh follows below:

```
#=============================================================

#This code initialize the PIT timers to interrupt (number 0) every ~24000 clocks

xor 10,0,0

ori 10,10,0xaa33

oris 10,10,0x55cc

stw 10,RTSECK_OFFSET(20)  # OPEN RTC KEY


stw 10,RTSEC_OFFSET(20)  # RESET RTC divider


addis 10,10,0x80

stw 10,PISCR_OFFSET(20)  # CLEAR PIT INT bit


lwz 7,SCCR_OFFSET(20)
```

```
andi. 8,7,0xffff
andis. 9,7,0xff7f
or 7,8,9
oris 7,7,0x0100
stw 7,SCCR_OFFSET(20)  # RTC_CLK = SYSCLK/512

xor 9,0,0
addis 9,9,0x2f
stw 9,PITC_OFFSET(20)  # Int every 24000 system clocks

xor 10,0,0
addis 10,10,0x85
stw 10,PISCR_OFFSET(20)  # PIT enable
sync
#=============================================================
#The interrupt routine should include the following code:
#=============================================================
INT0 :
lhz 9,PISCR_OFFSET(20)  #
sth 9,PISCR_OFFSET(20)  # CLEAR PIT INT bit
andi. 9,9,0x80
bc 0x4,2,INT0_L

xor 8,8,8
ori 8,8,0x0030
oris 8,8,0x8080   # Refresh CS_0 by MCR command
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)

ori 8,8,0x2000   # Refresh CS_1 by MCR command
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)
stw 8,MCR_OFFSET(20)  # MCR(UPMB,0x30)

INT0_L: .......
#==================================================================
```

SIU8 Lock/unlock function of RSR also locks/unlocks SCCR

Description:

When the RSR is locked or unlocked through the RSRK register, the same function is also performed on the SCCR.

Workaround:

This erratum should not affect user software as long as one is aware of it. In order to avoid possible software errors due to this (if, for example, the associated code statements were reordered by the user in a code revision), as a code convention one should always perform the unlock-modify-lock operations in immediate succession on individual registers. That is: unlock the register, modify it, then lock it.

SIU9 CPU receives machine check after writing to the PLPRCR

Description:

The CPU may receive a machine check after writing to the PLPRCR. This error is caused by an extra clock generated by the clock block after the SIU releases the bus.

The error occurs when the CPU begins a transaction after the SIU releases the internal bus. Due to the PLPRCR write, the CPU's clocks are stopped. However, the extra clock allows the CPU's transaction to begin before the clocks stop. Therefore, the CPU's clocks are stopped in mid-transaction, and its transaction never receives acknowledgement. When the CPU resumes operation, it receives a machine check.

The failure mechanism is due to an internal logic synchronization issue aggravated by memory refreshes performed by the UPM. The problem is most often exhibited when entering and exiting low-power modes frequently, such as when using doze mode to conserve power. The probability of encountering this problem is small but finite (approximately one in a million).

Workaround:

Prevent the CPU from getting the bus during the extra clock following the PLPRCR write. To do so, enable the instruction cache and create a software delay. To calculate how long of a delay is necessary, take the longest bus transaction in CPU clocks (including memory refresh or PCMCIA access). The resultant number of clocks must be executed using instructions such as NOP (1 clock), ISYNC (2 clocks), or DIVW (13 clocks). The instructions for the delay must either fit in one cache line, or some other mechanism must be used to make certain that the instructions for the delay are in the cache. The instructions of the delay must be executed after the PLPRCR write, which can be enforced by the ISYNC instruction. The length of the software delay must also be greater than or equal to the length of the longest bus transaction (in CPU clocks).

For example, if the longest transaction is eight bus clocks and the CPU is in 1:2 mode, the delay required is 16 clocks. This can be accomplished with two DIVW instructions. The instruction cache must be enabled to make certain that the code sequence is loaded into the cache as a unit, and the PLPRCR write should be aligned to a 16-byte boundary to make sure it is the first instruction in the cache line containing the sequence. The following workaround code is suggested:

```
.align 16

.global SetPLPRCR

SetPLPRCR:

    stw 4, PLPRCR_OFFSET(3) # burst aligned address

    isync                  # isync

    addi 3, 0, 1           # safeguard against divide-by-zero

    divw 4, 4, 3

    divw 4, 4, 3

    blr
```

SIU10 RTC/PIT does not count properly when the pre-divider is four

Description:

The periodic interrupt timer (PIT) consists of a 16-bit counter clocked by the PITRCLK clock supplied by the RTCLK clock (real time clock). The 16-bit counter counts down to zero when loaded with a value from the PITC. After the timer reaches zero, the PS bit is set and, if the PIE bit is a logic one, an interrupt is generated. The user can program the RTC and PIT clock to be divided by 4 or 512 (depending on SCCR[RTDIV]). When the RTC clock is divided by four, an interrupt will not occur due to a bug in the rtclk_sync_raw logic. The rtclk_sync_raw is the real time clock for the RTC timers, and its frequency is the same as rtclk_raw. If the PLL output clock is enabled and not in reset state, and the timer has not expired, the rtclk_sync_raw clock has a 25% duty cycle synchronous with a system T4 tick; otherwise this clock is the same as rtclk_raw.

From the ckpspcl schematics, rtclk_raw also selects rtclk_sync_raw. There is no issue with the above statement when the pre-divider is set to 256 clocks because the select line is slower then the selected clock source. However, when the pre-divider is set to four, there is supposed to be a rtclk_sync_raw edge every two clocks. The rising edge of this clock will disappear due to a race between rtclk_sync_raw (as the select line) and the ckp_rtclk_sysd (as the data for the mux).

At room temperature, this generates a spike signal, and at hot temperature, this degrades and disappears. When this happens, the RTC does not count properly, and no interrupt occurs.

CPM Errata

CPM1 Microcode bug in Async HDLC & IrDA protocol

Description:

If the first character received of a new buffer descriptor (BD) is either an unmapped (discardable) control character or the control escape character ($7D), erroneous operation will result.

Workaround:

Download and use the microcode patch available from Freescale.

CPM2 I2C receive problem in arbitration-lost state

Description:

If the MPC860 I2C master transmitter loses arbitration to another I2C master that is transmitting to the MPC860, the 860 receiver does not accept the message (address byte not acknowledged).

Workarounds:

Either of the following will serve as a workaround to this erratum:

- Avoid multiple master configuration.
- The other master should retry the operation through software.

CPM3 I2C error in FLT bit

Description:

An error occurs if the FLT bit is set to turn on the digital filter for the I2C. The digital filter is activated by setting the FLT bit in the I2C mode register and is turned off at reset.

(However, note that this digital filter is not required for normal operation. The MPC8xx I2C is fully compliant to the I2C specification even without this digital filter.)

Workaround:

Do not turn on the digital filter for the I2C clock filter.

CPM4 Shared flags in Async HDLC

Description:

If two Async HDLC frames are separated by a single shared flag, the second frame is discarded.

Workaround:

Download and use the microcode patch available from Freescale.

CPM5 I2C master fails to receive after executing write

Description:

If the I2C channel performs a transaction (read or write command) in master mode, it fails to receive a transmission from another master and responds with NACK.

Furthermore, if the I2C master then attempts to perform another transaction (read or write command) after the failed reception, the transaction fails with an underrun error.

Workaround:

After the master I2C channel completes its transmission, disable and re-enable the channel in the I2MOD register (thereby resetting it).

CPM6 I2C receives single-byte buffers after failed transaction

Description:

The following is true if the I2C channel is in master mode:

- If the I2C master attempts a transaction (read or write command) that receives a NACK and then attempts to execute a read to another slave, it receives the first byte of the slave’s message in one buffer, closes the BD, and continues to receive the rest of the message in the subsequent BDs. This reception of the first byte in a single-byte buffer will happen regardless of the MRBLR.

The following is true if the I2C channel is in slave mode:

- If the I2C slave responds to a read command (that is, performs a transmission) and then responds to a write command (that is, performs a reception), it receives the first byte of the master’s message in one buffer, closes the BD, and continues to receive the rest of the message in the subsequent BDs. This reception of the first byte in a single-byte buffer will happen regardless of the MRBLR.

Workaround:

After the I2C channel performs a transmission (master read or write, or slave response to read), disable and re-enable the channel in the I2MOD register (thereby resetting it).

CPM7 I2C receiver locks, holding SDA low

Description:

The I2C receiver may lock up, holding the SDA line low, in a system that has slow rise/fall time on the I2C clock (SCL) if the environment is noisy.

Workaround:

Set the I2C predivider to 32 (by setting I2MOD[PDIV]=00), and restrict the rise/fall time of SCL to 0.5 µs. In addition to this, for revision C.0 and later of the MPC860, enable the digital filter through the I2MOD[FLT]. (For previous revisions of the MPC860, the digital filter is not functional.)

CPM8 I2C master collision after ‘double start’

Description:

The following situation causes the I2C controller to collide with the transmission of another master:

1. Another I2C master performs a ‘master write’ to the I2C controller of the MPC860.

2. The I2C controller waits for the I2C bus to become idle in order to become the master and perform a transaction.
3. The other I2C master asserts a new ‘START’ condition without asserting a ‘STOP’ condition.

   In this situation, the I2C master of the MPC860 will incorrectly interpret the new ‘START’ condition as generated by itself and therefore drive the I2C bus concurrently with the other master.

Workaround:

Avoid performing back-to-back START conditions on the I2C bus.

CPM9 I2C short aborted transmission after NACK

Description:

The following situation causes the I2C controller of the MPC860 to send a short aborted transmission:

1. The I2C controller performs a transaction, transmitting a buffer that has no STOP condition at the end. The next buffer (not yet transmitted) issues a START condition, producing back-to-back transactions without an intervening STOP (also known as ‘double start’).
2. The I2C controller receives a NACK on the last or next-to-last byte of the buffer.

   In this situation, the I2C controller will assert a STOP condition (as expected by the I2C protocol). However, when software subsequently issues a new start command (I2COM = 0x81), the I2C master will begin its next transaction erratically. It will issue a START condition, drive one bit of the message, drive a new START condition, and restart the transmission (including the first bit).

Workaround:

Do not set up the I2C controller to perform ‘double start’.

CPM10 I2C split receive buffer between loopback and read

Description:

If the I2C master of the MPC860 performs a loopback transaction (that is, a master write to its own I2C address or a master write to the general call address with general call reception enabled) and then performs a master read transaction, the receive buffer used for the loopback transaction is closed after the first byte of the read transaction is received, instead of after the loopback transaction. Thus, the received data from the read transaction is split between the loopback buffer and the intended receive buffer.

Workaround:

Avoid performing loopback transactions during normal operation.

CPM11 I2C spurious BSY errors after reception in I2C master mode

Description:

This erratum occurs when the I2C controller of the MPC860 is configured as an I2C master and is the target of another master’s write. After the MPC860 receives data from the master (and thus closes the receive buffer appropriately), the controller will attempt to open the next receive buffer (even though there is no receive data). If there is no buffer available, it will generate a BSY error.

Workaround:

Ignore BSY errors in this case.

CPM12 RCCR must be written as a word

Description:

RCCR[ERAM4K] (located at address IMMR+0x9C7) is cleared if the RCCR is written as a byte or halfword (to addresses IMMR+0x9C4, IMMR+0x9C5).

Workaround:

To change the RCCR bytes at address IMMR+(0x9C4, 0x9C5), write the whole word (addresses IMMR+0x9C4 to IMMR+0x9C7).

General Errata

G1 Termination of open drain and active pull-up pins

Description:

Open drain pins and active pull-up pins drains "high" current when the input voltage to them is higher than VDDH. The excess current at the pin in question is approximately 150 microamps.

Active pull-up pins include: $\overline{\text{BB}}$, $\overline{\text{TS}}$, $\overline{\text{TA}}$, and $\overline{\text{BI}}$

Open drain pins include: $\overline{\text{TEA}}$, $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$, and any general-purpose I/O programmed as open drain

Workaround:

Connect the external pull-up resistor on these pins to the VDDH power supply.

G2 Higher than expected Keep-Alive Power (KAPWR) current when main power (VDDH & VDDL) is removed

Description:

Four nodes within the MPC860 float when VDDH and VDDL power is not supplied to the device. When this condition, which is typical in power down mode, occurs, the current drain on the keep-alive power rail is greater than expected. (10 - 20 mA versus 10 µA)

Workaround:

Provide adequate current source for the KAPWR pin in power down mode.

G3 EXTCLK and CLKOUT clocks may not be in phase in half-speed bus mode

Description:

When the MPC860 uses EXTCLK as an input clock source, MF=001 in PLPRCR (that is, the frequency of EXTCLK is half that of the internal clock), and half-speed bus mode is used (EBDF=01 in SCCR), the output clock from CLKOUT can be 90 or 180 degrees out of phase from the input clock. This will affect synchronous designs where the same clock source is used as an input to EXTCLK or to an external synchronous device (for example, a peripheral or ASIC).

Workarounds:

Case 1. Multiple external devices need to operate synchronously with the MPC860.

- Use the CLKOUT pin of the MPC860 as the clock source for all external, synchronous devices (that is, CLKOUT is the effective system master clock to be used for distribution).

Case 2. It is necessary to synchronize an external master clock (for example, from a backplane), an MPC860, and external peripherals, to allow data transfer in all three directions.

- There is no known workaround for this case. Use full-speed bus operation.

G4 Potential problems caused by skew between EXTCLK and CLKOUT

Description:

In correct operation, the PLL of the MPC860 will lock on the rising edge of the input clock. However, on these revisions of the MPC860, the PLL locks on the falling edge of the input clock. This affects the skew between EXTCLK and CLKOUT at the rising edge. The skew is dependent on the duty cycle of the input clock (but for a 50% duty cycle will not exceed 2nS). This affects synchronous designs where the same clock source is used as an input to EXTCLK, as well as to an external synchronous device (for example, a peripheral or ASIC).

Workarounds:

Case 1. Multiple external devices need to operate synchronously with the MPC860.

Use the CLKOUT pin of the MPC860 as the source of clock for all external, synchronous devices (that is, CLKOUT is the effective system master clock to be used for distribution).

Case 2. It is necessary to synchronize an external master clock (for example, from a backplane), an MPC860, and external peripherals, to allow data transfer in all three directions.

This workaround is a concept only. It has not been verified in hardware.

Insert a PLL between the external master clock and the EXTCLK pin of the MPC860. Connect the phase comparison pin of the PLL to the CLKOUT pin of the MPC 860. Also use the CLKOUT signal as the reference clock for distribution to the local external peripherals.

NOTE

The PLL must be capable of operating with a permanent offset of -2nS, therefore the range of lock should extend to about -4nS.

A diagram of this concept is given in Figure 1.

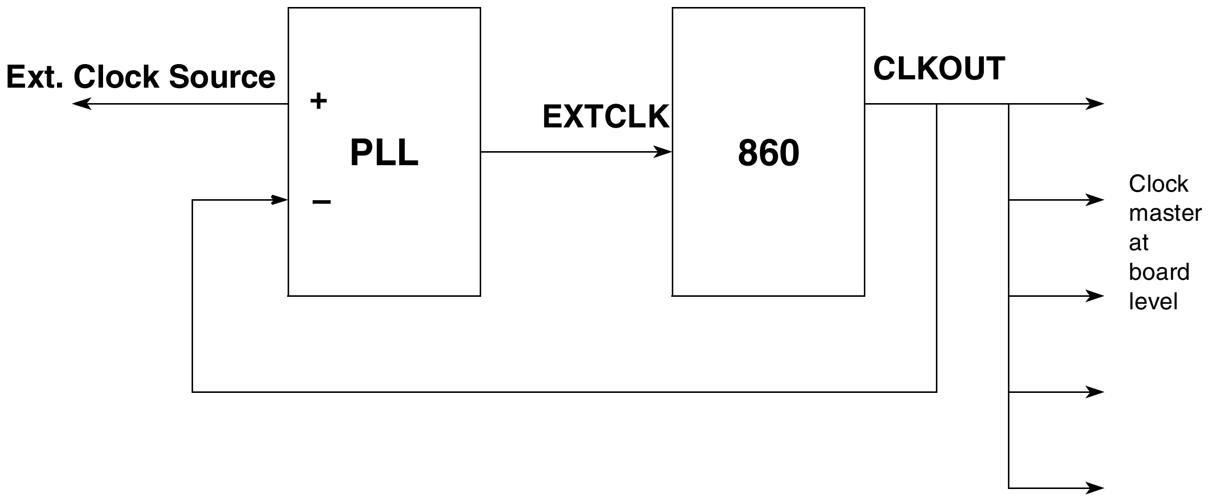

Figure 1. Workaround for Case 2

G5 ESD breakdown voltage for XFC pin less than Freescale-imposed requirements

Description:

The XFC pin (T2) of this version of the MPC860 silicon fails Freescale's XC qualification of 1 Kvolt for the electrostatic discharge (ESD) breakdown voltage test. The maximum ESD voltage that can be applied to this pin on this silicon without damage is 750 volts.

Workaround:

Ensure that devices are not exposed to greater than 750 volts of electrostatic discharge.

G6 Possible spikes on $\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, and $\overline{\text{IRQ7}}$

Description:

For MPC860 Rev B.1 silicon with date codes beginning with 'Q':

If $\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, or $\overline{\text{IRQ7}}$ is pulled up to 3.3 V, the data bus has a light capacitive load, the $\overline{\text{IRQ}}$ pin(s) in question are pulled up weakly and not driven high with an active driver, the $\overline{\text{IRQ}}$ pin(s) in question have a light capacitive load, and the associated interrupt is enabled in SIMASK (except for $\overline{\text{IRQ0}}$, which is always enabled), ground bounce caused by the data bus switching can potentially couple to the $\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, or $\overline{\text{IRQ7}}$ pin, resulting in a narrow spike being driven to the interrupt logic and possible subsequent erratic operation.

Workaround:

The following steps make up the workaround for this erratum.

1. Pull up the $\overline{\text{IRQ0}}$, $\overline{\text{IRQ1}}$, and $\overline{\text{IRQ7}}$ pins to 5 V.
2. Disable these $\overline{\text{IRQ}}$s by either masking the associated interrupt in SIMASK or connecting the $\overline{\text{IRQ}}$ signal directly to VCC. Note that disabling the interrupts in SIMASK will only work for $\overline{\text{IRQ1}}$ and $\overline{\text{IRQ7}}$ as $\overline{\text{IRQ0}}$ is non-maskable.
3. Actively drive these $\overline{\text{IRQ}}$ signals.

G7 Active pull-up drivers switch to high-impedance too early

Description:

The active pull-up drivers (which include $\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{BI}}$, and $\overline{\text{BB}}$) switch to high-impedance at a threshold voltage that is lower than the specified minimum output voltage level VOH. Thereafter, the pull-up resistor must pull the signal beyond the specified output voltage level. Depending on the pull-up resistor value and the capacitive load of the signal, this can result in a longer negation time than specified.

Workaround:

Use a 1K pull-up resistor for these drivers.

Notes:

The long rise times do not cause a problem to the processor. Nor will the longer rise times for these signals present a problem for other devices in most systems.

1. $\overline{\text{TS}}$ is normally sampled at the beginning of a bus cycle and is thereafter a ‘don’t-care’ until the cycle is terminated with $\overline{\text{TA}}$. Thus, a $\overline{\text{TS}}$ that extends into the subsequent clock cycles will be ignored.
2. $\overline{\text{BI}}$ must only be in its negated state when sampled concurrently with $\overline{\text{TA}}$ when a cycle is to a burstable target. In these systems, typically the only burstable target is the UPM, which will drive the $\overline{\text{BI}}$ actively throughout cycles in which it is in control of the target. Therefore, this behavior will not affect operation of the memory controller. Furthermore, for burstable targets that are not in control of the memory controller, (A) the pull-up resistor should have plenty of time to complete the signal negation before the $\overline{\text{TA}}$ of the cycle, and (B) the worst that can result from a falsely asserted $\overline{\text{BI}}$ is that the master would break the burst into four accesses, resulting in a performance degradation but not a system failure.
3. For a non-burst cycle, $\overline{\text{TA}}$ is normally sampled only once after $\overline{\text{TS}}$ is driven. $\overline{\text{TA}}$ is then a ‘don’t-care’ until after the next $\overline{\text{TS}}$ is driven. Therefore, there should be sufficient time for the pull-up resistor to complete the signal negation of $\overline{\text{TA}}$ before termination conditions for the next cycle are sampled. For burst cycles, typically the only burstable target in the system is the UPM, which drives the $\overline{\text{TA}}$ signal actively until the completion of the entire burst cycle, thus avoiding the problem during the burst. And for other burstable targets, it is the responsibility of the target to meet the appropriate assertion/negation timing for $\overline{\text{TA}}$.
4. If this condition results in a long negation time for $\overline{\text{BB}}$, the only effect is increased latency between bus cycles as the bus is handed off between bus masters. That is, the bus would falsely appear busy for a short period after the on-chip master actually released the bus.
5. $\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{BI}}$, and $\overline{\text{BB}}$ will typically be lightly loaded.

G8 $\overline{\text{TS}}$ driver switches to high-impedance too early

Description:

The active pull-up driver on $\overline{\text{TS}}$ switches to high-impedance at a threshold voltage that is lower than the specified minimum output voltage level VOH. Thereafter, the pull-up resistor must pull the signal beyond the specified output voltage level. Depending on the pull-up resistor value and the capacitive load of the signal, this can result in a longer negation time than specified.

Workaround:

Use a 1K pull-up resistor for this driver.

Notes:

The long rise times do not cause a problem to the processor. Nor will the longer rise times for these signals present a problem for other devices in most systems.

1. $\overline{\text{TS}}$ is normally sampled at the beginning of a bus cycle and is thereafter a ‘don’t-care’ until the cycle is terminated with $\overline{\text{TA}}$. Thus, a $\overline{\text{TS}}$ that extends into the subsequent clock cycles will be ignored.
2. $\overline{\text{TS}}$ will typically be lightly loaded.

G9 Conflict between data show cycles and SDMA burst writes

Description:

If data show cycles are enabled through SIUMCR[DSHW], and an internal register or dual-port RAM access is made immediately after a SDMA burst write, the SDMA burst write may be corrupted. The observed phenomenon is that a burst write with four operands will hold the second operand into the third and fourth burst beats. For example, a burst write of A-B-C-D will be observed on the bus as A-B-B-B.

This behavior can also occur when the SDMA burst is to burst-inhibited memory. Setting the memory to burst-inhibited will not solve the problem.

Workaround:

Do not use data show cycles in a system that performs SDMA bursts. This includes systems that use ATM, fast Ethernet, and memory-to-memory IDMA.

G10 Restriction of open collector pull up

Description:

The open collector signal may not be able to be pulled to greater than 3.5 V.

Workaround:

Use an external buffer if an open collector signal needs to be pulled to greater than 3.5 V.

CPU Errata

CPU1 Wrong data breakpoint detection on store instructions

Description:

The data breakpoint mechanism comparison of operand data and operand size is faulty. If used, it can cause breakpoints where they should not occur, and conversely it can miss breakpoints where they should occur.

NOTE

The instruction and address portions of the data breakpoint mechanism operate correctly. It is therefore still possible to use the data breakpoints to break on a store to a particular address and/or on a store instruction in a particular address range. Only the operand comparison portion of the data breakpoints does not function properly at all. However, see CPU11 for further minor limitations.

Workaround:

Do not use the operand comparison function of the data breakpoints for store instructions.

CPU2 Program trace mechanism error

Description:

In the following case there is an error in the program trace mechanism:

```
Program
0x00004ff0: divw. r25,r27,r26
0x00004ff4: divw. r28,r27,r26
0x00004ff8: unimplemented
0x00004ffc: b 0x00005010
```

where 0x00005010 belongs to a page with a page fault.

The divide takes a long time to complete, so the instruction queue gets filled with the unimplemented instruction, the branch, and the branch target (page fault).

When the sequencer takes the unimplemented instruction, it releases the fetch that was blocked by the MMU error. This causes the queue to get another instruction in addition to the first page fault. Since the second fault is sequential to the branch target, it is not reported by the queue flush (VF). This causes an incorrect value to be present in the VF flush information when the unimplemented exception occurs.

CPU3 Incorrect value used for POW bit of MSR register

Description:

The value of the POW bit in the MSR register was taken with opposite polarity by the clock control logic. Correct operation of the logic will assure that: if PRQEN in the SCCR register is '1', the system clock will switch/remain in high frequency as long as the POW bit in the MSR register is reset ('0'). The POW bit should be previously set (‘1') to allow to the system to switch to "low" frequency. Under this definition if the source of an interrupt request is reset right after jumping to the software routine that handles it, the clock will continue to run in high frequency because the POW bit is reset whenever an interrupt service routine is run by the processor. Because of this problem, the clock will switch to low frequency if the interrupt source is reset.

Workaround:

Reset the source of the interrupt request *before* the RFI instruction is executed; or manipulate the CSRC bit in the PLPRCR register such that it is reset when entering the interrupt software routine and set before the RFI instruction is executed.

CPU4 Instruction cache replacement policy bug

Description:

The I-cache replacement policy is not optimized. This does not affect the correctness of program execution but will affect performance by an average of 10-20%. Once a new silicon is available, performance should improve without any software changes required.

CPU5 Instruction MMU bug at page boundaries in show-all mode

Description:

The wrong instruction address is driven by the core when all of the following conditions occur:

- MPC860 works in 'show all' mode (that is, ISCT_SER bits = 000 in ICTRL
- Sequential instruction crosses IMMU page boundary
- Instruction cache fails to get ownership of the internal U-bus on the first clock

In this case the address driven by the core will be that of the previous page and not the current one.

The impact of this erratum has been deemed minimal, and it will therefore not be corrected.

Workaround:

Either of the following can serve as a workaround to this erratum:

- Disable show all mode.
- Invalidate the page next to current (by using the tlbie instruction) when performing the TLB reload operation.

CPU6 Possible data cache corruption when writing SPRs

A write access to a special-purpose register located in the caches, MMUs, or SIU might corrupt the contents of the data-cache.

This may happen regardless of whether the cache is currently enabled or disabled (by either writing a disable command to the DC_CST or by setting all regions to cache-inhibited through MD_CTR[CIDEF]). Thus, it is not possible to work around this problem simply by temporarily disabling the data cache.

NOTE

This is a probabilistic effect caused by an internal race condition and therefore does not occur in all cases. However, since it is due to a race condition, it is affected by all parameters that affect the speed of the silicon (for example, silicon revision, temperature, and voltage). Therefore, if a system exhibits behavior that varies due to these factors, it is advisable to check for occurrence of this erratum.

Table 5 shows the affected special-purpose registers.

Table 5. Affected Special-Purpose Registers

SPR	spr_address
IMMR	0x3d30
IC_CST	0x2110
IC_ADR	0x2310
IC_DAT	0x2510
DC_CST	0x3110
DC_ADR	0x3310
DC_DAT	0x3510
MI_CTR	0x2180
MI_AP	0x2580
MI_EPN	0x2780
MI_TWC	0x2b80
MI_RPN	0x2d80
MI_DBCAM	0x2190
MI_DBRAM0	0x2390
MI_DBRAM1	0x2590
MD_CTR	0x3180
M_CASID	0x3380
MD_AP	0x3580
MD_EPN	0x3780

Table 5. Affected Special-Purpose Registers (continued)

SPR	spr_address
M_TWB	0x3980
MD_TWC	0x3b80
MD_RPN	0x3d80
M_TW	0x3f80
MD_DBCAM	0x3190
MD_DBRAM0	0x3390
MD_DBRAM1	0x3590
DEC	0x2c00
TB Write	0x3880
TBU Write	0x3a80
DPDR	0x2d30

Either of the following will serve as a workaround to this erratum:

- If the contents of the TLBs are not changed dynamically (fixed-page structure), any access to the above-mentioned registers should be avoided (except for initialization).
- If the contents of the TLBs are changed dynamically (pages are loaded on demand), then each "mtspr" instruction which accesses one of these registers must be preceded by a store word and a load word instruction of a data operand equal to the spr_address of the respective register. As an example, to write the data from the general purpose register r1 to the special purpose register M_TW, the procedure in Table 6 should be followed.

Table 6. Writing Data from r1 to M_TW

Instructions	Description
lis r2, some address_msb	an address in RAM
li r3, 0x3f80	the spr_address of the M_TW from the table
stw r3, some address_lsb(r2)	no interrupts between these three instructions
lwz r3, some address_lsb(r2)	
mtspr M_TW, r1	

CPU7 Branch prediction with sequential branch instructions

Description:

If there are three branches in sequence in the run-time program flow, and the third branch is in the mispredicted path of the second branch, the third branch may be “issued” from the instruction queue even though it is part of a predicted path. If this instruction issue in the mispredicted path happens while the condition of the prediction is resolved (thereby causing mispredicted instructions to be flushed from the instruction queue), the resulting instruction cancellation backs up too far into the instruction queue. This causes the instruction sequence starting from the instruction immediately preceding the first branch to be re-issued.

Notes:

1. Other factors of the internal state of the core also affect the occurrence of this behavior. Therefore, not all occurrences of this instruction sequence necessarily exhibit this behavior.
2. This behavior is not necessarily harmful to the user application. For example, the instruction preceding the first branch could be a simple move between registers.
3. Not all compilers generate this instruction sequence. The following compilers are known never to generate code that is susceptible to this erratum:
   — Diab Data (all versions)
   — Metaware

   Other compilers with their vendors continue to be investigated; their status is unknown at this time. This list will be updated as the investigation progresses.

Workarounds:

The following applies for every conditional branch preceded in program order by another branch:

- If the two possible targets of the conditional branch consist of a branch instruction and a non-branch instruction, force the prediction of the conditional branch to predict the non-branch instruction (using the y-bit in the opcode of the conditional branch).
- If both of the possible targets of the conditional branch are branch instructions, (1) insert a non-branch instruction before the branch on the predicted path, or (2) insert an 'isync' instruction before the first branch.

CPU8 Missed instruction after conditional branch

Description:

If the instruction cache is enabled, a conditional branch residing near the boundary of the current memory page is mispredicted such that the CPU fetches beyond the page boundary, and the branch target also resides on another memory page, the instruction at the branch target address may not be executed.

The boundary of the current memory page is as follows:

- If the MMU is enabled (MSR[IR]=1), it is as defined by the associated MMU page table entry.
- If the MMU is disabled (MSR[IR]=0), it is at a 4 KB boundary.

This erratum also depends on the internal state of the core (instruction queue cancellation and MMU page swap), so it does not occur in all cases.

Workarounds:

Any of the following will serve as a workaround for this erratum:

1. Disable the instruction cache. This will cause the instruction to be fetched from external memory. Therefore, the instruction queue will not be filled until the branch is resolved.
2. Run the CPU in serialized mode (by programming the ICTRL[ISCT_SER] bits). This mode will keep the predicted instructions from executing until the branch is resolved.
3. Avoid conditional branches with predicted paths that cross page boundaries.

CPU9 Instruction sequencer error when modifying MSR with interrupts enabled

Description:

If the instruction sequence below occurs, and 1–4 are true, the sequencer takes op2 as the first instruction in the interrupt handler. The sequencer and instruction cache are also out of sync in subsequent instructions fetched in the interrupt handler until a change of flow is executed. (“Change of flow” can also be isync and mtmsr commands.)

```
mtmsr Rx# change IR (Instruction Relocate) bit or PR (Problem
        # State) bit in MSR
op1
op2
```

1. External interrupts were previously enabled (or are being enabled by this mtmsr instruction).
2. An external interrupt or decrementer interrupt occurs (or is already pending).
3. Op1 is not in the instruction cache.
4. The first instruction in the interrupt handler is fetched at the same clock that the op2 instruction is prefetched from external memory (as seen on the internal bus).

Workarounds:

Either of the following will serve as a workaround to this erratum:

- Do not execute mtmsr that changes the IR or PR bits when an external interrupt (and decrementer interrupt) are enabled (that is, when MSR[EE]=1). Allow at least two sequential instructions after the mtmsr that changes IR or PR before enabling interrupts.
- One of the conditions for this problem to occur is that the exception routine executes entirely from cache, including any load/stores. Therefore, another workaround to prevent this situation is to force one cache miss in each exception routine. This can be done by inserting a dcbf instruction prior to a lwz instruction in the exception routine. This workaround should cause minimal system performance impact.

CPU10 Possible excess current consumption in deep sleep mode

Description:

Certain nodes of the multiplier hardware are not initialized at reset and may thus result in non-destructive internal contention. As a result, if the processor is put into deep sleep mode without the multiplier first being put into a known state, current consumption in this mode may be higher than expected.

The impact of this erratum has been deemed minimal, and it will therefore not be corrected.

Workaround:

Execute a mullw instruction at any point after reset; this will put the internal nodes in an orderly state. Deep sleep mode may then be entered at any time thereafter.

CPU11 Wrong data breakpoint detection using address range function

Description:

Erroneous detection can occur when data breakpoint/watchpoint logic is used to trap on load/store accesses to addresses less than or greater than a given address. This error occurs for byte-size loads/stores where the address accessed and the compare address (in comparator E or F) differ only in the two least-significant bits, and the two least-significant bits of the address accessed are not 00.

The impact of this erratum has been deemed minimal, and it will therefore not be corrected.

Workaround:

Select the address in the compare register (E or F) such that it will differ by more than the two least-significant bits from the address range to be detected. For example, if the address range to detect are addresses less than or equal to 0xA0000006, use address 0xA0000008 instead.

CPU12 Non-maskable interrupts are unrecoverable

Description:

When an interrupt occurs during a RFI instruction, the RFI partially executes before the exception is taken. The partial execution includes copying the contents of SRR1 to the MSR. Therefore, the MSR value seen in the exception handler are the MSR value of the previous machine state, not of the machine state when the RFI occurred. The contents of the MSR are therefore unreliable upon entry into this ‘second’ exception routine.

Workaround:

This erratum only affects exceptions that occur during a RFI instruction. For a normal application, the only exceptions that can occur during a RFI instruction are non-maskable interrupts or breakpoints. (External interrupts will be masked during the machine-state recovery portion of an exception handler.) Therefore, only workarounds for the above two cases are required:

Either of the following will serve as a workaround for this erratum:

- Do not put an instruction breakpoint on a RFI instruction.
- Treat all non-maskable interrupts as non-recoverable since the state of the MSR[RI] bit is not reliable.

CPU 13 First instruction in decrementer or external interrupt handler may be skipped if instruction fetch show cycles are enabled in ICTRL[IST_SER]

Background:

The 860 IMMU works in the following way: Each access from the core reaches the cache using the page from the previous access on the assumption that the page has not changed. Usually that assumption is correct, so there is no stall. If the page has changed, a retry is done with the correct page. The MMU also signals the cache to disregard the previous access since the page was incorrect, and the results are useless. When a TLB miss occurs, there is no retry, but the previous access is normally aborted since it used the wrong page value. The MMU shuts itself off until there is an abort signal from the core that specifies a change in flow. Normally, this change of flow is the ITLB miss exception, in which case the instruction pipeline is cleared before taking the exception. However, if the decrementer or external interrupt is asserted, this exception can take precedence over the ITLB miss exception.

Description:

An ITLB miss is detected that is immediately followed by a decrementer or external interrupt exception. The first address of the exception routine is then presented by the core. Since the previous page was not valid (ITLB miss), the MMU provides an incorrect address (ff800500). As a result of the internal state of the MMU and the fact that it is a program trace show cycle, this cycle is (mistakenly) not cancelled, although it is retried. This causes the bad address to appear on the external bus as a show cycle. Although the show cycle is incorrect, the program flow usually continues correctly.

However, if the internal U-Bus pipeline happens to be full on the cycle discussed above, the retry is not handled correctly (as a result of the same issue mentioned above), and the second address of the routine (504) is used for the fetch instead of the first address (500). This causes the instructions of the first cache line to be fetched in the wrong order.

Workarounds:

Either of the following will serve as a workaround for this erratum:

- Since the first instruction of the decrementer and external interrupt routines might not be executed before the second instruction, it should not do anything useful, so it will not matter if it is skipped. Therefore, a nop or isync instruction should be placed in this location. An isync instruction should be placed in the second location of these routines to resynchronize the core with the IMMU. If the bug occurs, the isync instruction at 504 will be executed as if it is at location 500 and cause another fetch of itself from address 504. At this point, the program flow is resynchronized and continues as usual.
- Disable the instruction fetch show cycle in ICTRL[IST_SER].

CPU14 RCPU: do not execute "o" form integer instructions before "." form integer multiply/divide complete

Description:

When an "o" form integer instruction starts to execute before a previously started "." form integer multiply or divide instruction completes, condition register bit 3 (SO) may be wrongly updated by the XER[SO] bit changed by "o" form instruction. Instruction sequence "divw. Rz,Rx,Ry, subfo Rt,Rv,Ru" may cause this problem. This does not happen if the "o" form instruction is also in "." form or register dependencies exist.

Workarounds:

Any of the following will serve as a workaround for this erratum:

- Keep the "o" form instruction four integer or six other instructions apart from integer divide "." form instructions in the code or more than one instruction in the case of an integer multiply instruction.
- Use both instructions with the "." form.
- Run RCPU in serialized mode.
- Place the "sync" instruction between multiply/divide and "o" form integer instruction.
- Do not use "." form of integer divide or multiply instructions.

CPU15 Incorrect code execution after branch on MMU page boundary

Description:

An incorrect or invalid instruction is executed if all of the following are true:

- A conditional branch or branch to LR/CTR is executed in the last word of MMU page <n> (branches with primary opcodes 16 or 19).
- The target address of the branch points to the last cache line of MMU page <n>+1.
- The branch remains unresolved long enough for a prefetch of the first cache line of MMU page <n>+1 to start.
- Code execution continues after the branch beyond the end of MMU page <n>+1 into the first cache line of page <n>+2, which must already be in the cache.
- Both ICACHE and IMMU are enabled.

An incorrect or invalid instruction will be executed instead of the correct one out of the first cache line of MMU page <n>+2.

NOTE

Absolute branches (bx with primary opcode 18) are not the cause of the problem, as they are always resolved implicitly. Every MMU page containing a “risky” branch as described above is called a “risky” page below.

Workarounds:

A variety of workarounds are available:

- Do not put conditional branches or branches to LR/CTR into the last word of an MMU page, that is, generate and load the code image appropriately.
- If running code out of RAM, replace the conditional branch with an absolute branch, to patch a function containing the “true” branch code out of RAM, or replace each “risky” branch with a trap to a patch function if an absolute branch is not possible due to code size.
- Specifically for blr, add enough nops or other statements between mtlr and blr so that blr is definitely resolved during execution according to execution timing. This is only a partial workaround for blr.
- In the ITLB miss exception code, when loading the TLB for an MMU page, also invalidate any TLB referring to the next and previous page using tlbie. This intentionally forces an ITLB miss exception on every execution across sequential MMU page boundaries.
- Turn off ICACHE completely.

Below is a simple example ITLB miss handler showing how a tlbie-based workaround can be implemented. The actual code needed depends on how precisely the MMU is used in a system.

```
asm void os_inst_tlb_miss(void)
{
    mtspr        M_TW,r2     /* save r2 into M_TW */
    mfspr        r2,SRR0     /* SRR0 */
    mtspr        MD_EPN,r2   /* save instruction miss effective address in MD_EPN */
    mfspr        r2,M_TWB    /* load r2 with level one pointer from M_TWB*/
    lwz          r2,0(r2)    /* load level one page entry */
    mtspr        MI_TWC,r2   /* save level one attributes into MI_TWC */
    mtspr        MD_TWC,r2   /* save level two base pointer into MD_TWC */
    mfspr        r2,MD_TWC   /* load r2 with level two pointer while taking */
                          /* into account the page size */
    lwz          r2,0(r2)    /* load level two page entry */
    mtspr        MI_RPN,r2   /* Write TLB entry via MI_RPN */

    // The following code assumes that under all circumstances, instruction pages are
    // consecutive physical pages, i.e., the Physical Addresses (PA) of sequential pages
         // are sequential.
    // A page size of 4KB is also assumed.
    // If this is not the case, modifications are needed to translate the
    // address appropriately
    rlwinm      r2,r2,0,0,31-12 // Eliminate lower twelve bits (4K page!)
                            // extract "true" PA
    ori         r2,r2,0x1000-4  // Point to end of current page (4K page!)
    subi        r2,r2,0x1000    // Move to end of previous page
                            // (4K page! PA translation!)
    mtspr       SPRG0,r3
    mtspr       SPRG1,r4
    mfcr        r4
    lwz         r3,0(r2)        // Read instruction word at end of page using PA
                            //(This should work for LE)
    rlwinm      r3,r3,6,26,31   // Isolate primary opcode in lower 6 bits
    cmpwi       r3,16           // Conditional branch?
    beq         @tlbieprev      // Invalidate previous page
    cmpwi       r3,19           // Possibly conditional branch to register address?
    beq         @tlbieprev      // Invalidate previous page

    addi        r2,r2,0x1000    // Move back to current page address
                            //  (4K page! PA translation!)
    lwz         r3,0(r2)        // Read instruction word at end of page using PA
                            //(This should work for LE)
    rlwinm      r3,r3,6,26,31   // Isolate primary opcode in lower 6 bits
    cmpwi       r3,16           // Conditional branch?
    beq         @tlbienext      // Invalidate next page
    cmpwi       r3,19           // Possibly conditional branch to register address?
    beq         @tlbienext      // Invalidate next page
    mtcr        r4
    mfspr       r4,SPRG1
    mfspr       r3,SPRG0

    mfspr        r2,M_TW     /* Restore r2 previously saved in M_TW */
    rfi

@tlbienext:
    addi        r2,r2,0x1000    // Move to next page address (4K page! PA translation!)
@tlbieprev:
    tlbie       r2              // Make sure we refetch the ITLB on the dangerous page
    mtcr        r4
    mfspr       r4,SPRG1
    mfspr       r3,SPRG0

    mfspr        r2,M_TW     /* Restore r2 previously saved in M_TW */
    rfi
} /* os_inst_tlb_miss */
```

A variant that is further optimized can often be generated to avoid memory reloads. However, this specifically depends on MMU usage in the system. The following example handler improves performance by relying on three bits of a L2 TLB part not being used by other software.

```
asm void os_inst_tlb_miss_l2_based(void)
{
    mtspr       SPRG0,r3
    mtspr       SPRG1,r4
    mfcr        r4

    mtspr        M_TW,r2     /* save r2 into M_TW */
    mfspr        r2,SRR0     /* SRR0 */
    mtspr        MD_EPN,r2   /* save instruction miss effective address in MD_EPN */
    mfspr        r2,M_TWB    /* load r2 with level one pointer from M_TWB*/
    lwz          r2,0(r2)    /* load level one page entry */
    mtspr        MI_TWC,r2   /* save level one attributes into MI_TWC */
    mtspr        MD_TWC,r2   /* save level two base pointer into MD_TWC */

    // We abuse the SPS/SH/CI bits as indicator of our state in terms of the erratum.
    // We know that they are normally 0 in most applications, which is why we chose them.
    // SPS/SH/CI==0b000: We have never before encountered this code page. We need to decide
    // SPS/SH/CI==0b100: This page does not contain a "bad" branch.
    // SPS/SH/CI==0b001: This page contains a bad branch. tlbie the next page
    // SPS/SH/CI==0b010: The previous page contains a bad branch. tlbie the previous page
    mfspr        r2,MD_TWC   /* load r2 with level two pointer while taking into account
                                the page size */
    lwz          r2,0(r2)    /* load level two page entry */
    rlwinm       r3,r2,0,31,27 // Clear SPS/SH/CI before updating TLB
    mtspr        MI_RPN,r3   /* Write TLB entry via MI_RPN */

    // Now the magic starts. We automagically learn if pages need to be treated.
    // Fast case, i.e., nothing special comes first.
@recheckstatus:
    rlwinm.     r2,r2,28,0,2    // Extract the SPS/SH/CI field left justified
    bge         @checkipage     // This page has not been checked before or is known
                                // to be "bad"

    mtcr        r4
    mfspr       r4,SPRG1
    mfspr       r3,SPRG0
    mfspr       r2,M_TW     /* Restore r2 previously saved in M_TW */
    rfi


@checkipage:
    beq         @evalpagestatus // This page has not been tested before, so we need to
                                // update its info

    // When we get to this point, we know that we need to tlbie a page.
    // Depending on the SPS/SH/CI bits, we tlbie the next or previous page.
    // The initial assumption is that we have to tlbie the next page.
    // This code works only for 4KB pages.
    mfspr       r3,SRR0         // The address pointing into the current page
    addi        r3,r3,0x1000    // Move to next page (4K page! PA translation!)
    andis.      r2,r2,0x2000    // This maps to 0b001 in the left justified value
    bne         @tlbienext
    subi        r3,r3,0x2000    // Move back to prev page (4K page! PA translation!)
@tlbienext:
    tlbie       r3              // Make sure we refetch the ITLB on the dangerous page

@pagecheckdone:
    mtcr        r4
    mfspr       r4,SPRG1
    mfspr       r3,SPRG0
    mfspr       r2,M_TW     /* Restore r2 previously saved in M_TW */
    rfi

// The code below probably does not need to be locked in the cache. It won't run a lot.
@evalpagestatus:
    // The following code assumes that under all circumstances, instruction
    // pages are mapped to consequtive
    // physical pages, i.e., the Physical Addresses of sequential pages are sequential.
    // A page size of 4KB is also assumed.
    // If this is not the case, modifications are needed to translate the address
    // appropriately
    rlwinm      r2,r3,0,0,31-12 // Eliminate lower twelve bits (4K page!) to extract
                                // "true" PA
    ori         r2,r2,0x1000-4  // Point to end of current page (4K page!)
    lwz         r3,0(r2)        // Read instruction word at end of page using PA
                                // (This should work for LE)
    rlwinm      r3,r3,6,26,31   // Isolate primary opcode in lower 6 bits
    cmpwi       r3,16           // Conditional branch?
    beq         @riskypage      // This is a risky page
    cmpwi       r3,19           // Possibly conditional branch to register address?
    beq         @riskypage      // This is a risky page

    subi        r2,r2,0x1000    // Move to the previous page (4K page! PA translation!)
    lwz         r3,0(r2)        // Read instruction word at end of page using PA
                                // (This should work for LE)
    rlwinm      r3,r3,6,26,31   // Isolate primary opcode in lower 6 bits
    cmpwi       r3,16           // Conditional branch?
    beq         @prevrisky      // We are on the page after a risky one
    cmpwi       r3,19           // Possibly conditional branch to register address?
    beq         @prevrisky      // We are on the page after a risky one

    // This page is not risky and not near a risky page. So we mark it as safe!
    mfspr       r3,MD_TWC       /* load r3 with level two pointer while taking into account
                                   the page size */
    lwz         r2,0(r3)        /* load level one page entry */
    ori         r2,r2,0x0008    // Set SPS/SH/CI to 0b100 for an innocent page
    stw         r2,0(r3)        /* store level one page entry */
    b           @pagecheckdone

@prevrisky:
    mfspr       r3,MD_TWC       /* load r3 with level two pointer while taking into account
                                   the page size */
    lwz         r2,0(r3)        /* load level one page entry */
    ori         r2,r2,0x0004    // Set SPS/SH/CI to 0b010 for a page next to a risky one
    stw         r2,0(r3)        /* store level one page entry */
    b           @recheckstatus

@riskypage:
    mfspr       r3,MD_TWC       /* load r3 with level two pointer while taking into account
                                   the page size */
    lwz         r2,0(r3)        /* load level one page entry */
    ori         r2,r2,0x0002    // Set SPS/SH/CI to 0b001 for a risky page
    stw         r2,0(r3)        /* store level one page entry */
    b           @recheckstatus

} /* os_inst_tlb_miss_l2_based */
```

ATM Errata

ATM1 Problematic reporting of BSY errors

Description:

Modes affected: Extended Channel, Serial, or Parallel (UTOPIA)

In the intended operation of the MPC860SAR, if cells are received with no receive buffer descriptors available, the receiver will issue a BSY interrupt after the first cell is received and stop reception. However, due to this erratum, the receiver will issue a BSY interrupt for every received cell, not just for the first cell, when the MPC860SAR is in extended channel mode (EXT=1 in SRSTATE). This will continue until the condition is corrected (that is, until the host provides an empty receive buffer descriptor).

Also, the first new AAL5 frame after the BSY interrupt may contain only a partial AAL5 frame (as indicated by the Length and CRC errors).

Workaround:

Respond quickly to BSY errors by preparing empty receive buffer descriptors. Ignore the first frame that arrives with a LEN or CRC error.

ATM2 Incorrect channel number in exception queue

Description:

Modes affected: Extended Channel, Serial

The channel number field (CHNUM) in the interrupt table entries of the exception queue may be faulty when the MPC860SAR is in extended channel mode (EXT=1 in SRSTATE) and an RXB, TXB, RXF, UN or BSY exception occurs.

Workaround:

Do not rely on the CHNUM field of the interrupt table entry. Poll the buffer descriptor status of all the receive or transmit channels when receive or transmit interrupts are reported. However, because the CHNUM has an approximately 50% chance of being correct, the channel indicated by CHNUM should be polled first.

ATM3 Incorrect operation of APC_BYPASS command

Description:

Modes affected: Parallel (UTOPIA)

The inserted channel is not immediately transmitted after the APC_BYPASS command is issued. It is delayed until the next cell is scheduled by the APC. If the APC table is empty, the inserted channel is never transmitted.

Workaround:

Insert at least one channel or a disabled (dummy) channel to the APC. The channel issued by APC_BYPASS will be transmitted as soon as the next channel (or dummy channel) is scheduled by the APC.

ATM4 Incorrect operation of expanded cell mode

Description:

Modes affected: Expanded cell, Parallel (UTOPIA)

The following is true if the MPC860SAR is in expanded cell mode (EC=1 in SRSTATE and STSTATE):

1. The expanded cell header transmits the last four bytes of the expanded cell header in place of the ATM cell header. For example, with a 12-byte expanded cell header (ECSIZE=3), the cells are transmitted as:

   [Cell Header Expansion 1 || Cell Header Expansion 2 || Cell Header Expansion 3 || Cell Header Expansion 3 || 48 byte Cell Payload]
2. The Tx SOC is asserted on the first byte of the cell header instead of the first byte of the expanded cell. For example, on a 64-byte cell transmit, the SOC signal is asserted on the 13th byte.

Workarounds:

For AAL0: Use external logic to sync on the TX SOC, and generate the correct SOC in reference to that. Also, insert the ATM cell header in the last word of the expanded cell header.

For AAL5: There is no workaround because the cell header must be manipulated by the microcode, specifically, the PTI[2] bit.

ATM5 APCO interrupts cannot be masked

Description:

Modes affected: All

APCO interrupts cannot be masked with the IMASK field of the receive connection table entry (RCT).

Workarounds:

Generally, if the APC is programmed well, there should not be any APCO. However, if they do occur and the user wants to mask them, either of the following methods are acceptable:

- Implement a software workaround that will ignore the specific APCO interrupts in the interrupt table or mask all interrupts globally by using the GINT mask in IDMR1 or SCCM.
- Download the RAM microcode package for enhanced UBR support. An enhancement supporting APCO masking has been integrated into this package.

ATM6 The OAM cell screener (with CUMB=0 and PTI[3]=0 in FLMASK) is inoperative

Description:

Modes affected: Extended channel using address compression

This erratum applies only if the MPC860SAR is in extended channel mode (EXT=1 in SRSTATE) and is using address compression tables (ACP=1 in SRSTATE), with CUMB=0 and PTI[3]=0 in FLMASK.

In this mode, the MPC860SAR should use the PTI[3] bit of the received cell to perform ‘OAM cell screening’ if CUMB of FLMASK is set to 0 and PTI[3] is not included in the address compression algorithm (that is, PTI[3]=0 in FLMASK). When performing OAM cell screening, the MPC860SAR checks the PTI[3] bit of the incoming cell. If the cell is set it is sent to the raw cell queue (connection number 0) to be processed as an OAM cell. This is possible because if address compression is used and CUMB is not set, it is assumed that the MPC860SAR receives no misinserted cells and any cell with PTI[3] set must be an OAM cell. CUMB must be set to 1 if it is possible in the system for the MPC860SAR to receive misinserted cells.

However, the OAM screener function does not currently work; the OAM screener does not check PTI[3] when CUMB=0. If the PTI[3] bit (the most significant bit of the cell header PTI field) is not included in FLMASK, the OAM and RM cells are treated as data cells instead of being inserted in the raw cell queue. In the case of AAL5, the OAM and RM cells are inserted into the AAL5 CPCS_PDU. This results in the OAM and RM cells being lost and the AAL5 data frame being corrupted.

A more detailed explanation of the OAM screener bug is that it checks the PTI[3] bit in the FLMASK to see if it should activate the OAM screener. If PTI[3]=0 in the FLMASK, then it is supposed to activate the screener and thus throw all received cells with PTI[3] set into the global raw cell queue. However, because of this erratum, it looks at the wrong bit when the OAM screener is activated; it throws cells with PTI[2] set into the global raw cell queue.

The workaround described below keeps the OAM screener deactivated.

Workarounds:

Treat the OAM cells as a separate AAL0 channel. Keep the PTI[3] bit set in the FLMASK (thereby using the PTI[3] bit to identify the connection). Assign the connection number for the OAM cells to something other than the connection number for the data cells. Route all OAM cells to the same queue by using the same connection number for all OAM connections or route them to individual queues by giving them unique connection numbers.

ATM7 CPM lockup when issuing APC_BYPASS when TX queue full

Description:

Modes affected: all

The CPM locks up if a cell is scheduled for transmission through the APC_BYPASS command when the transmit queue is full, causing immediate failure of all channels.

In the case of a CPM lockup, the CPM must be reset. This can be accomplished either through the CPCR[RST] or by issuing a $\overline{\text{SRESET}}$.

This should not occur during optimal operation. An overflow of the TX queue indicates that more transmit traffic has been scheduled than the physical layer can transmit, which is an error condition. Software should avoid this situation.

To fix this, the operation will be changed so that this condition does not cause lockup, and an additional semaphore bit will be provided to assist in avoiding this situation.

Workaround:

Monitor the transmit queue status, and do not issue the APC_BYPASS command if the number of empty entries in the transmit queue is less than (NCITS+2).

ATM8 Incorrect operation in presync state of cell delineation

Description:

Modes affected: Serial Receive

Incorrect operation occurs if a HEC error occurs during the Presync state of the serial receive cell delineation state machine. Instead of moving back to the Hunt state, the receiver decrements Alpha by one, receives the cell into the global raw cell queue, and remains in the Presync state. The cell delineation state machine will move back to the Hunt state only when the Alpha parameter reaches zero. This erroneous operation can result in long receive startup times, as decrementing Alpha can cause it to overflow back to 65535. The most common occurrence of this problem occurs when the lock is lost due to a line going down. There are occurrences of both good and bad HECs in the received cell sequence during restart.

Workaround:

Program Alpha = 1 and Delta = 6IF if the lock state is lost at restart or the cell delineation state machine is not locked and the global raw cell queue contains more than seven cells with HEC errors.

After programming these parameters, the user must check after at least four system clocks to confirm that these values were actually written to these parameters and were not overwritten by the CPM.

NOTE

A SYNC interrupt is issued the lock state is lost; see the description of the SYNC interrupt in the *MPC860 PowerQUICC User's Manual*.

ATM9 Intermittent errored cell transmission at high frequencies

Description:

Modes affected: UTOPIA

The UTOPIA interface may intermittently transfer cells that are not 53 bytes in length with a CPM frequency of greater than 50 MHz. The phenomenon exhibited is that the header transfer of the cell is foreshortened, changing from a four-byte transfer to a one- or three-byte transfer. This problem occurs because the SOC signal corrupts an internal counter. This phenomenon is not affected by the UTOPIA clock rate. The problem is more prevalent at high temperatures but may also occur at room temperature.

Workaround:

Download the Freescale-supplied microcode patch. The patch changes the mechanism of header transfer from a single burst of four bytes to two bursts of two bytes, and in so doing avoids counter corruption. Using the patch does not significantly affect ATM performance.

ATM10 Incorrect raw cell queue mapping in UTOPIA multi-PHY mode

Description:

Modes affected: UTOPIA multi-PHY receive in extended channel mode using address compression

In UTOPIA multi-PHY mode, the address mapping mechanism directs the received cells to the incorrect receive channel number when received cells are sent to the global raw cell queue (in the case of unidentified cell header, OAM cell screening, and so on). Correct mapping of the global raw cell queues would direct the cells such that RCT_entry=PHY#. Instead, the mechanism directs the cells such that RCT_entry = PHY# * 4.

Workaround:

Be aware of the error in the mapping mechanism, and assign the RCT entries accordingly. When configuring the global raw cell queues in the RCT for each PHY, use the following configuration:

PHY0 global raw cell queue --> RCT entry 0

PHY1 global raw cell queue --> RCT entry 4

PHY2 global raw cell queue --> RCT entry 8

PHY3 global raw cell queue --> RCT entry 12

ATM11 Errored cell transmission in UTOPIA clock 1:4 mode

Description:

Modes affected: UTOPIA

In UTOPIA mode, errored cell transmission can occur for AAL0 cells if the system clock to UTOPIA clock ratio selected is 1:4 (through the SCCR). The result of the failure is that the first four bytes of the payload may be dropped. The failure is intermittent.

Workaround:

Use other UTOPIA clock ratios (for example, 1:2, 1:3, 1:5).

FEC Errata

FEC1 Multicast address recognition in 7-wire interface mode

Description:

When in 7-wire interface mode, RECV.R_CNTRL.MII_MODE==0, the destination address and the first nibble of the source address are used to determine the hash value. Only using the destination address causes the hash tables to function improperly. Address recognition (both for multicast and unicast addresses) works correctly in MII transceiver interface mode.

Workaround:

None. The hash table should not be used if running in 10-Mbps serial mode.

FEC2 Re-enabling FEC with Ether_EN

Description:

The FEC does not properly reset all of its state machines with the negation of CSR.ECNTRL.ETHER_EN.

Workaround:

The user must assert CSR.ECNTRL.RESET in addition to negating CSR.ECNTRL.ETHER_EN to reinitialize the FEC.

FEC3 Maximum frame length enforcement

Background:

Receive frames of length >= 2 Kbytes (2048) should be truncated at 2047 bytes since the receive length counter is 11 bits in length. The TR bit should be set in the receive buffer descriptor for such frames.

Description:

This truncation may or may not occur correctly depending on the relative phase and frequency between the MII_RX_CLK and the 860T system clock. The problem has been observed with 10-Mbps traffic.

If the truncation does not occur correctly, one or both of the following behaviors may result:

- The FEC receiver locks up, preventing any further frames from being received. The only way to restart is to reset the FEC by negating ECNTRL.ETHER_EN (this will not reset configuration control registers but will reset the data path and the dma pointers to the descriptor rings) or by asserting ECNTRL.RESET (requires complete initialization).
- The FEC will write bogus status/length information into the receive buffer descriptor for the truncated frame. It has been observed that all status and length bits = 0; however, other values can occur.

The only way to prevent this problem from occurring is to ensure there are no frames on the network >= 2 Kbytes in length.

FEC4 BABT interrupt missing

Description:

The babbling transmit condition may not be detected correctly under all conditions. If the FEC is appending the CRC, and the frame length is in the 1519-1522 range, the BABT interrupt does not occur even though it should. If software appends the CRC, the BABT interrupt will always occur properly at 1519 bytes.

FEC5 Byte order

Description:

The DATA_BO and DESC_BO fields in the DMA.FUN_CODE register do not function as specified. The current implementation gives the following results:

BO[0:1] = 0X results in byte swapping within a 32-bit word.

BO[0:1] = 11 does not byte swap.

FEC6 PowerPC little-endian mode

Description:

The DMA block in the FEC does not implement the PowerPC little-endian mode correctly.

This erratum will not be corrected. This feature has been cancelled.

FEC7 Port D and port A programming

Description:

Port D takes programming precedence for alternative functionality over port A. In other words, where the same functionality exists on a port D pin and a port A pin such that one of which will be programmed for an alternative function and the other as general purpose I/O, the port D pin must be used for the alternative function, and the port A pin must be used as the general purpose I/O.

For example, PD11 and PA11 both have the ability to be programmed as RXD3. If PD11 were programmed as its other alternative function of MII_TX_ER, PA11 could be used for RXD3. If however, the FEC functionality is not to be used, so that either PD11 or PA11 could be RXD3, PD11 must be used for RXD3, and PA11 must be used as general purpose I/O. If PA11 were programmed as RXD3 and PD11 as general purpose I/O, the RXD3 would not be functional.

FEC8 Receive FIFO overflow

Description:

Under normal operating conditions, when a receive FIFO overflow occurs, the fifo_r_space_avail signal negates, the OV bit is set in the receive buffer descriptor, and software discards these frames. However, if the receiver is doing back-to-back writes to the receive FIFO at the time of overflow, the fifo_r_space_avail may not negate quickly enough, allowing the write pointer to go beyond the read pointer, the OV bit will not be set, the remainder of the frame will be corrupted, and the receive buffer descriptor status/length will be incorrect. Testing to date indicates that the receive FIFO does recover on the next frame.

Workaround:

Avoid overflow. This requires adequate system memory bandwidth and software that never falls behind in producing empty receive buffers.

FEC9 Address offset A40 (GSMR_L3) aliases to 840

Description:

Location 840 is also written when A40 is written.

This does not occur if SCC3 is in the HDLC, UART, BISYNC, or Ethernet modes.

Workaround:

If SCC3 is programmed to any other mode, the software workaround is to follow the following steps whenever the SCC3 mode register needs to be written:

1. Disable interrupts
2. Write SCC3 mode register
3. Write 0 into Eppc+0840 (a production test register)
4. Re-enable interrupts.

Note: There can be *no* FEC access between 2 and 3 above.

FEC10 Address offset C40 aliases to E40

Description:

When location C40 (within the SIRAM) is written, the FEC CSR.ECNTRL register (location E40) is also written. CSR.ECNTRL contains the ETHER_EN and RESET bits. If location C40 is written after the FEC has been enabled, the FEC may become disabled, and other erratic behavior may occur.

Workaround:

Initialize the SIRAM prior to initializing the FEC. Once the FEC is enabled, the SIRAM location C40 must not be read or written.

FEC11 Certain multicast destination addresses are mistakenly seen by the FEC as the broadcast address, ff:ff:ff:ff:ff:ff

Description:

Specifically there are 15 different multicast addresses that are incorrectly treated as being broadcast addresses. These are ff:ff:ff:ff:ff:*f where, * can be 0–E.

These multicast addresses are in the "locally administered" category, meaning that they are not controlled by the IEEE 802.3 standard.

FEC12 CRC error reported if MII_RX_ER asserts while in internal loopback mode

Description:

External inputs on the MII interface should be ignored when in internal loopback mode, but the receive logic currently mistakenly reports a crc error if MII_RX_ER asserts when a frame is being looped internally.

Workaround:

Hold the MII_RX_ER signal low when running an internal loopback test.

FEC13 Aggressive mode non-functional

Description:

The impact of this erratum is that to run the FEC in full-duplex mode, the processor must be running at 50 MHz, and the system must utilize SDRAM. (See the application note *MPC860T Design Advisory* for loading, latency, and system calculations in standard arbitration mode, http://www.mot.com/netcomm/aesop/mpc8XX/860/860t3.pdf.)

This erratum will not be corrected. This feature has been cancelled.

FEC14 Data or instruction corruption due to early bus access termination

Description:

Data or instruction corruption may occur in the following situation:

1. A write to a FEC register occurs while ECNTRL[ETHER_EN] is set.
   (Typically, only R_DES_ACTIVE, X_DES_ACTIVE, and IEVENT are written while the device is in normal operation. Other registers are written only during initialization.)
2. The write occurs at a point in time when the FEC is busy with transmit, receive, or DMA activity, which will cause the write to act as a “posted” write until the FEC can service it.
3. The following internally-initiated bus access (to either the internal or external bus) completes before this posted FEC write actually completes.

If the above situation occurs, then the next bus access may be terminated early with data that may or may not be valid.

The failure rate is very low (once every one or two days). It is worse at higher frequencies and has been noticed only under heavy traffic load, although it could theoretically occur at any time.

Workaround:

If only one UPM is being used (and GPL_x4 and GPL_x5 are not being used), the second UPM may be used to implement a software workaround. The software should initialize both UPMs. The first UPM, used in normal operation, should be programmed for optimum operation. The second UPM should be programmed similarly, but for slower memory access for all read, write, burst read, and burst write accesses. The UPM should be programmed such that bus accesses (from $\overline{\text{TS}}$ to the last $\overline{\text{TA}}$) complete in a minimum of fourteen clocks. For example, for SDRAM, the second UPM should insert nop commands to the SDRAM at the start of the bus accesses to delay the SDRAM cycles; for EDO or fast page mode DRAM, the second UPM should extend the RAS phase of the bus accesses.

Having programmed this, the user should follow the following procedure when writing to an FEC register:

1. Change BR[MS] so that the memory is controlled by the ‘slow’ UPM. (This can be changed on the fly.)
2. Perform the FEC register write.
3. Execute out of cache for a minimum of fourteen clocks (that is, perform no instruction fetches or load/store accesses from external memory). This can be accomplished by executing a delay loop which is locked in the instruction cache.
4. Change BR[MS] back, so that the memory is controlled by the ‘fast’ UPM.

FEC15 7-wire problem on the fast Ethernet controller at the end of an Ethernet frame

Description:

The FEC uses the receive clock to clock in data and run the FEC as the data is comes in. When the message has ended (signified by RENA being negated), the external clock discontinues. The FEC switches over to use an internal clock to finish flushing the FIFO and whatever else it needs to do at the end of the received message. The problem is the FEC was designed with the assumption that the external clock would quit immediately after RENA is negated. The MC68160 actually gives one more clock pulse after RENA is negated, and the level-one transceiver gives up to five more pulses. If a clock pulse is input while the FEC is switching internal clock sources, a glitch may occur on the internal clock line, which may cause "unpredictable results.”

Workaround:

Use an external AND gate to gate the receive clock with RENA.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

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

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

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

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

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

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

Document Number: MPC860CE
Rev. 1.9
05/2008