Device Errata
MPC821

MPC821Silicon Revision B.3 -- Mask Set 5J24A
March 30, 2000 (Version 3)

These errata are valid on Revision B.3 silicon. Please note that any errata listed in this document applies to the B revision of the silicon, unless otherwise stated. Changes to this errata are in *italics*.

CPU ERRATA

CPU1. Bus Error Not Fully Supported by the Data Cache on a Burst

The Data Cache does not support a bus error which might occur on the 2nd or 3rd data beat of a burst. (burt_232)

Workaround: Avoid using bus error in this case.

CPU2. Possible Data-Cache Corruption With Special Purpose Register Access Located In Data Cache, Data MMU or SIU

A write access to a special-purpose register located in the Caches, MMUs, or the SIU might corrupt the contents of the data-cache. The special-purpose registers are: IMMR, IC_CST, IC_ADR, IC_DAT, DC_CST, DC_ADR, DC_DAT, MI_CTR, MI_AP, MI_EPN, MI_TWC, MI_RPN, MI_DBCAM, MI_DBRAM0, MI_DBRAM1, MD_CTR, M_CASID, MD_AP, MD_EPN, M_TWB, MD_TWC, MD_RPN, M_TW, MD_DBCAM, MD_DBRAM0, MD_DBRAM1, DEC, TB, TBU, AND DPDR. (burt_292)

Workaround: 1. 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).

2a. If the contents of the TLBs are changed dynamically (pages are loaded on demand), then each “mtspr” instruction which access one of these registers must be preceded by a store word and load word instruction with the data 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 following procedure should be followed:

```
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
lwz   r3, some_address_lsb(r2)# between this
mtspr M_TW, r1                # 3 instructions
```

2b. Go to Normal Low Mode before any of the special purpose registers mentioned above are accessed.

CPU3. Case of I-Cache Using Address of Old Page When Fetching New Page

The Instruction Cache uses the address associated with the old page when fetching the first data from a new page, under the following circumstances.

1. There is a show cycle on a sequential instruction which crosses the page boundary.
2. The internal bus is busy during the IMMU request with the old page number.

Thus on the next cycle the I-Cache uses this incorrect address to access the external memory and internal cache. (burt_285)

Workaround: Do not run in “show all” mode or do not put a sequential instruction in the last address of an MMU page.

CPU4. Incorrect Data Breakpoint Detection on Store Instructions.

When a breakpoint on data occurs and you have programmed the size elements as byte or half-word, the following may occur:

- A breakpoint might be detected when it should not
- A breakpoint might not be detected when it should

Either of these two cases can occur if the data that is programmed to be detected, matches some other portion of the register that is currently stored to memory by the store byte or store half-word instruction.

For example:

- Assume that you have programmed a byte data breakpoint on a store instruction and you are looking for the byte element 0x55. Assume that register R1 has the value 0x00080000, R10 has the value 0x55443322, and the stb R10,0x3(R1) store instruction is performed.

  What occurs is that byte 0x22 from R10 is stored to address 0x00080003, and this should not generate a breakpoint since 0x22 does not equal 0x55, but, in some cases, it can and does (in this scenario, R10 does include the data 0x55). The result is a breakpoint is executed when it should not be.
- Assume that in the above case you are programming for byte element 0x22, maybe a breakpoint condition will not be detected, although it should. (burt_246)

Note: These fault cases depend on the previous Load-Store instruction address. If the previous Load-Store instruction address’ LSB is different from the current instruction address’ LSB, then an incorrect breakpoint detection might occur.

CPU5. Program Trace Mechanism Error

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

The program is:

```
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 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 besides the first page fault. Because the second fault is sequential to the branch target it is not reported by the vf. This causes a wrong vf flush information to be reported when the unimplemented exception occurs. (burt_251)

CPU6. Possible Error in Instruction Execution with Branch Prediction and Sequential Branch Instructions.

IF there are three branches in sequence in the run-time program flow
AND the third branch is in the mis-predicted path of the second branch,
THEN although the third branch is part of a predicted path, it may be "issued" from the instruction queue. If this instruction issue in the mis-predicted path happens at the same time that the condition of the prediction is resolved (thereby causing mis-predicted instructions to be flushed from the instruction queue), then the resulting instruction cancellation will back up too far into the instruction queue. This will cause the instruction sequence starting from the instruction immediately preceding the first branch to be re-issued.

Note: 1) Other factors of the internal state of the core also affect the occurrence of this behavior. Therefore, not all occurrences of this instruction sequences 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) and Metaware. We are continuing to investigate other compilers with their vendors; their status is unknown at this time. We will update this list as our investigation progresses.

Workaround: For every conditional branch preceded in program order by another branch:

1) IF the two possible targets of the conditional branch consist of a branch instruction and a non-branch instruction
THEN force the prediction of the conditional branch to predict the non-branch instruction (using the y-bit in the opcode of the conditional branch).

2) IF both of the possible targets of the conditional branch are branch instructions
THEN insert a non-branch instruction before the branch on the predicted path OR insert an 'isync' instruction before the first branch.

CPU7. Missed Instruction After Conditional Branch.

IF the instruction cache is enabled
AND IF a conditional branch residing near the boundary of the current memory page is mis-predicted such that the CPU fetches beyond the page boundary
AND the branch target also resides on another memory page
THEN the instruction at the branch target address may not be executed.

[The boundary of the current memory page is as follows:

1) If the MMU is enabled (MSR[IR]=1), then it is as defined by the associated MMU page table entry

2) If the MMU is disabled (MSR[IR]=0), then it is at a 4KB boundary.]

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

Workarounds: 1) Disable the instruction cache. This will cause the instruction to be fetched from external memory, and 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.

CPM ERRATA

CPM1. see General Customer Information CI-102

CPM2. I^2C Receive Problem in Arbitration-Lost State

If the MPC821 I^2C master transmitter loses arbitration to another I^2C master which is transmitting to the MPC821, the 821 receiver will not accept the message (address byte not acknowledged).

Workaround: 1. Avoid multimaster configuration

2. The operation should be retried by the other master through software.

CPM3. I^2C Error in FLT Bit

An error will occur if the FLT bit is set to turn on the digital filter for the I^2C. The digital filter is activated by setting the FLT bit in the I^2C mode register and is default off at reset. Note, that this digital filter is not required for normal operation. The MPC821 I^2C is fully compliant to the I^2C specification without this digital filter option. (burt_282).

Workaround: Do not turn on the digital filter for the I^2C clock filter.

CPM4. I^2C Master Fails to Receive After Executing Read or Write

If the I^2C channel is in master mode, after the I^2C channel performs a transaction (read or write command), the I^2C channel will fail to receive a transmission from another master. It will respond with NACK.Furthermore, after the failed reception, if the I^2C master then attempts to perform another transaction (read or write command), the transaction will fail with an underrun error.

Workarounds: (1) After the master I^2C channel completes its transmission, disable and re-enable the channel in the I2MOD register (thereby resetting it).

(2) Don’t use multiple I^2C masters.

CPM5. I²C Receives Single-Byte Buffers After Failed Transaction

A. If the MPC821 I²C channel is in master mode, then:

IF the MPC821 I²C master attempts a transaction (read or write command) which receives a NACK, AND the I²C master then attempts to execute a read to another slave,
THEN the master will receive the first byte of the slave's message in one buffer and will close the BD, and then will continue to receive the rest of the message in the next BDs. This reception of the first byte in a single-byte buffer will happen regardless of the MRBLR

B. If the MPC821I²C channel is in slave mode, then:

IF the MPC821 I²C slave responds to a read command (i.e. performs a transmission), AND the I²C slave then responds to a write command
THEN the slave will receive the first byte of the master's message in one buffer and will close the BD, and then will continue to receive the rest of the message in the next BDs. This reception of the first byte in a single-byte buffer will happen regardless of the MRBLR.

Workaround: After the MPC821 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).

CPM6. Possible I²C receiver lock up, holding the I2CSDA line low

The I²C receiver may lock up, holding the I2CSDA line low, in a system that has slow rise/fall time on the I²C clock (I2CSCL) or if the environment is noisy.

Workaround: Set the I²C pre-divider to 32 (by setting I2MOD[PDIV]=00), and restrict the system such that the rise/fall time of I2CSCL no greater than 0.5 microseconds.

CPM7. Shared Flags in Asynchronous HDLC.

If two asynchronous HDLC frames are separated by a single shared flag, the second frame will be discarded.

Workaround: Download Motorola-supplied microcode patch.

URL: http://www.mot.com/SPS/ADC/pps/subpgs/etoolbox/821/ucodepA.html

CPM8. I²C Master Collision After 'Double Start'.

The following situation will result in the I²C controller colliding with the transmission of another master:

1) Another I²C master performs a 'master write' to the I²C controller of the MPC821.

2) The I²C controller of the MPC821is waiting 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 case, the I²C master of the MPC821 will incorrectly interpret the new 'START' condition as generated by itself, and will 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.

The following situation will cause the I²C controller of the MPC821 to send a short aborted transmission:

1) The MPC821's I^2C controller performs a transaction, transmitting a buffer which has no STOP condition at the end. The next buffer (not yet transmitted) will issue a START condition, producing back-to-back transactions without an intervening STOP (also known as 'double start').

2) The MPC821's I^2C controller receives a NACK on the last or next-to-last byte of the buffer. If this case occurs, then the MPC821's I^2C controller will assert a STOP condition (as expected by the I^2C protocol). However, when software subsequently issues a new start command (I2COM = 0x81), the I^2C master will begin its next transaction erratically. It will issue a START condition and drive one bit of the message, then drive a new START condition and restart the transmission (including the first bit).

Workaround: Do not set up the MPC821's I^2C controller to perform 'double start.'

CPM10. I^2C Split Receive Buffer Between Loopback and Read.

IF the MPC821's I^2C master performs a loopback transaction (i.e. a master write to its own I2C address or a master write to the General Call address with General Call reception enabled).
AND the MPC821's I^2C master then performs a master read transaction
THEN the receive buffer used for the loopback transaction will not be closed after the loopback transaction. Instead, it will be closed after the first byte of the read transaction is received. Thus, the received data from the read transaction will be split between the loopback buffer and the intended receive buffer.

Workaround: Avoid performing loopback transactions during normal operation.

CPM11. I^2C Spurious BUSY Errors After Reception in I^2C Master Mode.

IF the MPC821's I^2C controller is configured as an I2C master
AND the I^2C controller is the target of another master's write,
THEN after the MPC821 receives the data from the master (and thus closes the receive buffer appropriately), it 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 BUSY error.

Workaround: Ignore BUSY errors in this case.

CPM12. Aggressive Mode Frequency Limitation

The use of aggressive mode (as determined by the LAM, RAID and LAID fields of the SDCR, page 16-78 of the MPC821 UM) is not recommended for use above 50 MHz. If used above 50 MHz unexpected behavior will result. This behavior is caused by edge conflicts in the internal logic for this mode of operation.

SIU ERRATA

SIU1. Incorrect Timing of PCMCIA Slot B ALE Signal.

The Address Latch Enable (ALE) pin should change with external address phase (t3) of the SIU; but it actually changes at time t1.

This erratum only effects designs which use latches for address in their PCMCIA slot B interfaces.

The intended operation of the PCMCIA ALE signal is that ALE should assert with address on the rising clock edge (plus delay specified in spec P52) and should negate on the rising clock edge (plus 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:

1) Verify address latch timing with this new timing information. If it is satisfied, there is no problem.

2) Do not use external address latches for the PCMCIA slot B interface.

SIU2. Elimination of DRAM Refresh Request or MCR Run Command Execution

External master access may cause elimination of a DRAM refresh request. Also if refresh is enabled, external master access may cause elimination of MCR run command execution

Workarounds:

For refresh: If the system includes an external master on the system bus, DRAM refresh should be done by software. This can be accomplished by using the Periodic Interrupt Timer with an interrupt service routine which activates a DRAM refresh pattern from the UPM via a RUN command in Memory Command Register (MCR) of the memory controller. In this case, the hardware refresh mechanism of the memory controller should be disabled.

For other MCR run commands (e.g. initializing SDRAM):

If the system includes an active external master on the system bus, then make sure hardware refresh is disabled.

GENERAL ERRATA

G1. Core Operation Is Limited to a 3.0V Minimum

The current versions of the MPC821 silicon are only tested and verified at 3.0V–3.6V power. Because of this, low voltage operation at 2.2V cannot be guaranteed to power the core.

Workaround: None.

G2. ESD Breakdown Voltage for XFC Pin Less Than Motorola Imposed Requirements.

The XFC pin (T2) of the B.1 (3J24C) version of the 821 silicon fails Motorola's XC qualification of 1K Volt 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 devices on not exposed to greater than 750 volts of electrostatic discharge.

G3. EXTCLK and CLKOUT Clocks May Not Be In Phase In Half Bus Speed Mode

When the MPC821 uses EXTCLK as an input clock source and MF=2 in the PLPRCR (i.e., the frequency of EXTCLK is 1/2 of the internal clock) and the half bus speed mode is used (EBDF=1 in the SCCR), the output clock from CLKOUT could be 180 degrees out of phase from the input clock. This

will effect synchronous designs where the same clock source (i.e., EXTCLK) is shared between the MPC821 and another synchronous device. (burt_293)

Workaround: Use the CLKOUT as the only source clock to all synchronous devices.

G4. PLL May Lock On The Falling Edge Of EXTCLK

The PLL of the MPC821 can lock on either the rising or the falling edge of the input clock (clock at EXTCLK pin). If it locks on the falling edge, this will effect the skew between EXTCLK and CLKOUT at the rising edge. Effectively, the skew at the rising edge will depend on the duty cycle of the input clock. This will effect synchronous designs where the same clock source (i.e., EXTCLK) is shared between the MPC821 and any other synchronous devices.

Workaround: Use the CLKOUT as the only source clock to all synchronous devices.

G5. LCD Controller Off Sequence When LAM Bit Is Set May Cause the CPU to Lockup

Starting with Revision B, an additional bit was added to provide more aggressive arbitration for the LCD Controller when doing DMAs to system memory. This bit is bit 25, called the LCD Aggressive Mode bit or LAM bit, of the SDCR Register. If this bit is set in the SDCR and the LCD Controller is turned off, the LCD Controller generates a spurious request to the SDMA that may cause the CPU to lockup.

Workaround: Clear the LAM bit of the SDCR before turning the LCD Controller off.

G6. LCD Off then On Sequence With a Pending SDMA Cycle Causes Wrong Data Fetch

If the LCD Controller is turned off and there is a pending SDMA cycle for the LCD Controller, and then the LCD is turned on again before the SDMA cycle completes, the LCD Controller will start fetching from the start address + 16 instead of the start address.

Workaround: Ensure that the pending SDMA cycle is completed before turning on the LCD Controller. This can be done by performing an access to external memory before turning the LCD Controller on .

G7. Lock/Unlock Command of RSR Also Locks/Unlocks SCCR

When the Lock or Unlock mechanism of the KAPWR is used on the RSR the same function is performed on the SCCR. (burt_283)

Workaround: The user should modify the RSR and SCCR registers in the following sequence.

1. The initial state is both registers are locked.
2. Unlock one register, then modify it and then lock it.
3. Unlock the next register, then modify it and lock it.

G8. Active Pull-up Pins De-assertion Drive Ends Too Soon

All active pull-up signal pin's output buffers stop driving the signal to its negated state earlier than they should. This causes the output signal to be negated slower than specified. (burt_288)

Workaround: Use a smaller value pull-up resistor on the following pins: TS*, TA*, BB*, BI*.

G9. The External Bus Transaction May Hang After a PLPRCR Write Access

An endless external bus transaction can occur on the next external bus access after executing a PLPRCR write command. The PLPRCR write command causes the PLL to freeze the clocks until it is locked again. The failure

mechanism occurs because the clock unit indicates operation complete before all necessary tasks are actually completed. The next external bus request is driven by the system interface unit and suddenly all clocks are stopped.

Workaround: The store instruction to the PLPRCR register should be in a burst-aligned address (cache line) followed by an isync instruction. The instruction cache should be enabled while executing this code sequence.

G10. Possible External Bus Hang Occurs Under Certain Error Conditions

The external bus cycle may hang when the following sequence of events occur:

1. The transaction on the external bus ends as a result of $\overline{\text{TEA}}$ assertion OR a bus monitor timeout occurs.

AND

2. The next transaction also ends with a $\overline{\text{TEA}}$ assertion or a bus monitor timeout. (burt_300)

G11. Higher Than Normal Current Consumption Without Executing a MULLW instruction

In the integer multiply module in the core, there are internal latches that do not get initialized properly during reset. Due to this improper initialization there is a 1:8 chance of a driver contention which may create higher than normal current consumption in various modes. This current consumption is most noticeable in the lower power modes. (burt_361)

Workaround: Adding a mullw instruction puts the two control lines into a valid state, eliminating the contention.

G12. Incorrect Reporting of Loss-of-Lock Reset Status.

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

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

G13. Possible Spikes on IRQ0, IRQ1, and IRQ7.

For MPC821 Rev B.x silicon with date codes beginning with ‘Q’:

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

Workarounds: 1) Pull up the IRQ0, IRQ1, and IRQ7 pins to 5V.

2) Disable these IRQs, by either masking the associated interrupt in SIMASK or connecting the IRQ signal directly to VCC. [Note that disabling the interrupts in SIMASK will only work for IRQ1 and IRQ7; IRQ0 is non-maskable.]

3) Actively drive these IRQ signals.

G14. Conflict Between Data Show Cycles and SDMA Burst Writes.

IF data show cycles are enabled via SIUMCR[DSHW]
AND an internal register or dual-port RAM access is made immediately following an SDMA burst write, THEN 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.

Note: *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.*

Workarounds:

Do not use data show cycles in a system that performs SDMA bursts. This includes memory-to-memory IDMA.

G15. CPU Receives Machine Check After Writing to the PLPRCR

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. When the internal bus is released the CPU begins a transaction. The CPU's clocks are then stopped mid-cycle and it never receives the acknowledge from the bus. The failure mechanism occurs due to an internal logic synchronization issue aggravated by memory refreshes performed by the UPM. The problem is only evident when entering and exiting doze mode frequently, such as when using doze to conserve power. The possibility of encountering this problem is small but finite (1 in a million entries).

Workaround: Prevent the CPU from getting the bus during the extra clock. To do so you must enable the instruction cache and insert a delay. To calculate how long of a delay is necessary, take the longest bus transaction including memory refresh and PCMCIA (in CPU clocks). The resultant number of clocks must be executed using instructions such as NOP (1 clock), ISYNC (2 clocks) or DIVW (13 clocks). If you use the DIVW instruction, then divide the resultant number by 13 and round up. Then insert this many DIVW instructions (dividing by one) after the isync (see errata G9).

For example, If your longest transaction is 16 CPU clocks, then you must add 2 DIVW instructions:

```
      .align 16
      .global SetPLPRCR
SetPLPRCR:
      stw r4, PLPRCR(r3)
      isync
      addi r3, r0, 1
      divw r4, r4, r3
      divw r4, r4, r3
      blr
```

GENERAL CUSTOMER INFORMATION

Although not generally considered to be errata the following items are provided as guidelines in the appropriate use of the device.

CI-100. External Interrupt Handling

For external interrupt pins, if a request signal is a pulse, the interrupt request pin should be configured to “edge detect mode”. This makes sure that the interrupt will be recognized even if interrupts are temporarily blocked or disabled via software. The interrupt service routine (ISR) should clear the edge status flag after the ISR is entered and prior to setting the MSR (EE) bit [if it waits until after the EE bit is set, a 2nd interrupt may be taken].

If a request signal is a “standard handshake”, the assertion is asynchronous, but the negation occurs upon request from the ISR. This ensures that the interrupt is taken, and the source of the interrupt is known. The timing with respect to the EE bit is the same as above.

In order to avoid spurious interrupts, any interrupt masks should not be set while interrupts might be sent to the core. Likewise, no interrupts should be disabled while the interrupt might be pending at the core. That way, when the core responds to the interrupt request, the request will still be pending, and the core can determine what the source of the interrupt is. To accomplish all of the above, the EE bit should be disabled when masks are set, or when interrupt enables are cleared.

CI-101. Move To Special Register (mtspr) Access to ICTRL Register

Setting the ICTRL(IFM) bit (IFM=1), which is the Ignore First Match, by mtspr together with setting an instruction breakpoint on this instruction causes unpredictable behavior of the chip.

Workaround: Disable instruction breakpoints when setting the IFM bit of the ICTRL Register.

CI-102. Concurrent Operation Of Ethernet & I²C or SPI has Overlapping Parameter RAM Tables.

When concurrent operation for the Ethernet protocol and either I2C or SPI is set up and used at the same timer, there is an overlap in the parameter RAM.

Workaround: There is microcode available that moves the I^2C/SPI parameter RAM entries to another location in the dual port RAM. To use this, download the description of the change and the object code file from the website at:

http://www.mot.com/SPS/ADC/pps/_subpgs/_etoolbox/8XX/i2c_spi.html. This package is called the MPC8XX I^2C/SPI Microcode Package.

GENERAL DOCUMENTATION ERRATA ASSOCIATED WITH SILICON OPERATION

The following items reflect operation of the MPC821 and how this operation may be currently misrepresented in the MPC821 User's Manual (MPC821UM/AD). Please refer to the sections mentioned, in the UM, for clarification and/or replacement by the following information.

DOC1. Cache Inhibit Operation

In some cases, the last instruction executed from a certain page gets the caching inhibited attribute of the NEXT page when the page change occurs between the time the fetch request was issued to the Instruction Cache and the time the Instruction Cache delivers the instruction to the sequencer. Since Instruction Cache inhibit is used only for performance reasons (mostly for not caching very fast memories or pages that include non real-time programs), the performance effect of this feature is negligible. See Section 9, Instruction Cache. (burt_237)

DOC2. DAR and DSISR Updating with Debug Counter Operation

If a load/store breakpoint occurs as a result of debug counter expiration when one of the following three interrupts occur, Machine Check interrupt due to an error in a load/store cycle, Data Storage interrupt, and Alignment interrupt; the DAR & DSISR is set to the effective address associated with the interrupting instruction. In some cases, when a load/store breakpoint occurs due to one of the debug counters expiration; just before one of the above interrupts occur, the value of the DAR & DSISR is changed. Although the interrupt is after the breakpoint; and, therefore should be ignored by the processor, the DAR and DSISR are updated. The value of the DAR & DSISR is normally used by the software inside these interrupt routines and therefore this anomaly may influence program flow only if these interrupts are nested one inside the other and load/store breakpoint is used inside one of these interrupt routines. See Section 6.6.13, Core, and Section 19, Development Support. (burt_253)

DOC3. Ethernet Is Only Supported on SCC1

All references to Ethernet support on SCC2 is in error. The MPC821 only supports the Ethernet protocol on SCC1. Disregard all references to Ethernet on SCC2 in Section 16 of the MPC821 User's Manual.

DOC4. Aggressive Mode Bit Added to the SDCR

Referring to Section 16.10.2.1 (page 16-78) of the MPC821 User's Manual, a new feature has been added to the Revision B and subsequent silicon. Bit position 25 is no longer reserved and is now referred to as the LCD Aggressive Mode bit, or LAM. When this bit is cleared, LAM=0, the SDMA arbitration operates as described in the User's Manual. When the LAM=1, then the LAID and RAID fields must be set equal to '00'. This bit affects all SDMA transfers including LCD and Video Controller transfers.

For example, when LAM=1, LAID=00, and RAID=00, the SDMA arbitration function is set to provide the LCD Controller with the highest possible priority, to allow for improved display memory to LCD panel data transfer, thus minimizing the chance for an underrun condition.

Bit 25 of the $SDCR_{LAM}$:

LAM - LCD(Video) Aggressive Mode
0 = Disable LCD/Video aggressive mode. Priority depend on the LAID field.
1 = Enable LCD/Video aggressive mode. The LAID and RAID fields must be equal to 00.

DOC5. Reserved Bits in the PDPAR

Referring to Section 16.19.12.3 (page 16-466) of the MPC821 User's Manual, Port D Pin Assignment Register; note that bits 0-2 are marked with a '-' and these bits should be considered "reserved" on the MPC821. Anytime this register is written to, care should be taken to ensure that those three bits are always written as '0', the reset value. Prior revisions of the silicon allowed these bits to be 0 or 1, but with Revision B of the MPC821 these bits are used internally and when set to 1, improper operation may occur.

DOC6. I2C Address Register Not Cleared on Reset

The I2C Address Regsiter (I2ADD) is not cleared by reset. This may cause a nonzero value to be present on reset. If this value matches the address of a slave device on the I2C bus, it will cause reception problems over I2C. This errata has no other effect on operation.

Workaround: If using I2C Master Mode, reset I2ADD to 0x0 during your I2C initialization routine.