Atmel AT697E

Rad-Hard 32-bit SPARC v8 Processor

ERRATA SHEET

Active Errata List

1. Multiplier/Divider Failure on Negative Operands Treatments
2. Call Return Address Failure with Large Displacement
3. Byte and Half-word Write to SRAM Failure when Executing from SDRAM
4. Wrong PC stored during FPU exception trap
5. Single-stepping over SWAP and LDSTUB instruction locks AHB bus
6. Divide overflow will not clear zero flag
7. ~~Register file fault-injection incorrectly implemented~~
8. 'Data cache tag' Error Counter Counting Error
9. Wrong SDRAM chip-select asserted in 512MB SDRAM bank-size when SRAM and SDRAM are enabled
10. Power-down causes lock-up of processor
11. PCI arbiter erroneous reset
12. Address lead-out cycle on I/O read sequence does not always appear
13. Odd-numbered FPU register dependency not properly checked in some double-precision FPU operations
14. Meaningless PCI Class Code
15. Deadlock with delayed PCI reads during long AHB wait states
16. Memory Block Protection

Table 1. Errata History

Product Release	Errata List
All AT697E part-numbers	1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16

Errata Description

1. Multiplier/Divider Failure on Negative Operands Treatments

The embedded multiplier/divider block can generate wrong values when negative operands are used. The bug occurs with negative dividends and positive divisors. As the dividend is 64-bit including the y register, the y-register needs to be initialised to a negative value in order to reproduce this error.

In all cases observed, the absolute error is a maximum of 2, but in case of small quotients, this can be a large relative error, for example -2/1 = 0 instead of -2. Since the bug can also cause zero or sign change, the ICC flags can also be wrong.

This Multiplier/Divider failure is due to a LEON2 VHDL model error.

Workarounds

If you want to maintain the hardware multiplier functional, it is sufficient to avoid negative operands or to avoid SDIV operations.

If you do not need the multiplier to remain available, you can simply not use the -mv8 flag when compiling the project with LECCS / BCC / RCC compilers. This will disable use of both the divider and the multiplier.

2. Call Return Address Failure with Large Displacement

CALL instruction saves the return address in an FPU register rather than in an IU %o7 register when the call address is larger than 64Mbytes (forward) or 8Mbytes(backward). The return address is normally expected by the return operation in %o7 register at the end of a called function.

The bug is triggered exactly and only by opcodes matching the following condition:*(bit 31 == 0) & (bit 30 == 1) & (bit 24 == 1) & (bit 21 == 0)*

- Bit 31/30 == “01” indicate the call instruction
- Bit 24/21 are part of the relative call address (30 bit word displacement) which is coded in the opcode, corresponding to bits 26/23 of the offset in terms of byte addresses.

As a consequence, the failure will only occur if the displacement is >= 64 MByte for forward calls or >= 8 MByte for backward calls. Due to the specific condition however, not all calls above this limit will trigger the failure.

The called function is executed normally, but the return fails.

This “Call Return Address” failure is due to a LEON2 VHDL model error.

Workarounds

Use only calls with an address displacement smaller than 64 Mbytes (forward) or 8 Mbytes (backward).

If dynamic linking is used, the memory configuration should be constrained to avoid calls beyond the given limits. If the operating system is moved to the upper memory area, all OS calls are forward calls, allowing the use of up to 64 MByte program memory.

3. Byte and Half-word Write to SRAM Failure when Executing from SDRAM

If an application is executing in SDRAM, byte- and half-word writes to SRAM can fail if the EDAC or read-modify-write cycles are used.

This “SRAM Write access” failure is due to a LEON2 VHDL model error.

Workarounds

Do not perform byte and half-word write operations to SRAM when executing code from SDRAM (in assembler language, byte/half-word stores are performed by the instructions STB, STH, LDSTUB -- CLRB and CLRH also, they are

synthetic instructions for STB and STH -- and their alternate space equivalents, whereas in C-code, they are typically assignments to char and short data types).

4. Wrong PC stored during FPU exception trap

When a trap is taken by the processor, the program counter (PC) is stored into %l1 of the trap window and the next program counter (nPC) is stored into %l2. This operation works correctly for all traps except FPU exception (trap type 0x08). During FPU exception, the nPC is erroneously stored into both %l1 and %l2. This means that the exception handler can not return and re-execute the trapped FPU instruction. During normal operation, this is not a problem since re-executing the trapped instruction would just cause the instruction to trap again.

Workarounds

Return from FPU exception by restoring the execution address from %l2 (nPC) only, thus skipping the trapped FPU instruction (this is usually performed by a JMPL %l2, %g0 / RETT %l2 + 4 instruction sequence).

5. Single-stepping over SWAP and LDSTUB instruction locks AHB bus

During a debug session using the debug support unit (DSU), it is possible to perform singlestepping. If an attempt to single-step a SWAP or LDSTUB instruction is made, the AHB bus will be locked and further debugging will be impossible. The reason for this behaviour is that SWAP and LDSTUB instruction perform a read-modify-write cycle which locks the AHB bus to insure atomicity. When such an instruction is single-stepped, the lock signal will be kept active even after the processor enters debug mode, thereby preventing further bus arbitration. Since the communications with the DSU is done over the AHB bus, further debugging is impossible and the device is in principle dead-locked. This state can only be exited by deasserting the DSUEN signal (resuming execution) or asserting the RESET signal.

Workarounds

Do not single-step over SWAP and LDSTUB instructions. Instead, set a breakpoint on the instruction right after the SWAP/LDSTUB instruction and resume normal execution.

6. Divide overflow will not clear zero flag

The divide instructions SDIVCC and UDIVCC set the integer condition codes (negative, overflow and zero) with respect to the final result. When a divide overflow occurs, a pre-defined non-zero value is returned, the overflow bit is set and the zero bit is cleared. However, under certain overflow conditions, the zero bit is wrongly set even though the result is always nonzero.

Workarounds

If direct control over assembly language is possible, simply do not rely on the zero bit flag on a divide overflow. If direct control over assembly language is not possible (high-level programming language such as C), activate the appropriate compiler options to prevent the compiler from using the divide instructions (if using LECCS / BCC / RCC compilers, do not use the -mv8 compiler flag).

7. Register file fault-injection incorrectly implemented

Caution: The faulty behaviour raised here is related to a LEON2 VHDL model error that is not applicable in AT697E configuration of the model.

8. ‘Data cache tag’ Error Counter Counting Error

The data tag error counter doesn’t show the correct number of errors.

If a data tag error is detected on a write cycle, the data cache is not updated but the tag error is counted. Since the tag is not written the error remains, and on sub-sequent writes to the same address the error will be counted again.

It is also possible that a tag error occurs on the same address offset as an on-chip register. A tag error will then be reported every time the register is accessed, since the access is not cacheable and the tag will not be written. If the application software does not use this particular location of the cache, the error will never be removed but continously reported.

Workarounds

Applications that do not use all locations of the cache should monitor the error counters and perform a flush operation when an error has been registered.

9. Wrong SDRAM chip-select asserted in 512MB SDRAM bank-size when SRAM and SDRAM are enabled

If the SRAM and the SDRAM are enabled in MCFG2 (the SDRAM is mapped starting at address 0x60000000) with an SDRAM bank size of 512 Mbytes, then the *SDCS*[1]* signal will be asserted instead of the *SDCS*[0]* signal when accessing the SDRAM.

This error does not occur when the SDRAM bank size is smaller than 512 Mbytes, or when the SRAM is disabled (the SDRAM is mapped starting at address 0x40000000).

Workarounds

When SRAM and a SDRAM bank size of 512 MB are used, only one SDRAM bank is supported. So connect SDCS*[1] to the chip-select pin of the SDRAM device when an SDRAM bank size of 512 Mbyte is configured and the SRAM is enabled too.

10. Power-down causes lock-up of processor

If an asynchronous interrupt occurs between the store/load sequence of power-down, the processor might enter power-down inside the interrupt handler. Since the traps are then disabled, the processor will not exit the power-down mode on the next interrupt and hang infinitely.

Workarounds

Do not use the power-down functionality in an application with unpredictable asynchronous interrupts.

11. PCI arbiter erroneous reset

The AHB clock domain in the PCI arbiter is erroneously reset by the PCI reset, while the PCI clock domain is reset by the AHB reset. If a PCI reset is issued, then the PCI arbiter registers will be reset to their default value.

Workarounds

Assert the PCI reset simultaneously with the processor reset to avoid possible side-effects.

12. Address lead-out cycle on I/O read sequence does not always appear

During an I/O read sequence, the address is specified to be stable one clock cycle after the de-assertion of the *IOS*[x]* I/O select signal. Under certain conditions, the address is de-asserted at the same clock edge as the I/O select signal. The occurence of the missing address lead-out cycle cannot be (easily) predicted.

Workarounds

Design the external I/O devices to operate correctly without the need for an address lead-out cycle.

13. Odd-numbered FPU register dependency not properly checked in some double-precision FPU operations

Data dependency is not properly checked between a load singleword floating-point instruction (`LDF`) involving an odd-numbered floating-point register as a destination of the load and an immediately following double-precision floating-point instruction (`FADDd`, `FSUBd`, `FMULd`, `FDIVd` or `FSQRTd`) that satisfies all of the following conditions:

- the odd-numbered floating-point register is used as (part of) a source operand
- the destination floating-point register is also a source operand
- in an `FSUBd` or `FDIVd`, the two source operands are different registers

In this case, the final result of the double-precision floating-point instruction will be wrong.

Other double-precision floating-point instructions (`FCMPd`, `FCMPEd`, `FdTOi` and `FdTOs`) are not affected by this issue and will operate as expected.

The error case appears when any of the six following sequences of instructions is present (n in [0:31], x and y as different even numbers in [0:30]):

Case 1:

```
LD [%rn], %fx+1
FPOPd(1) %fx, %fy, %fx
```

Case 2:

```
LD [%rn], %fx+1
FPOPd(1) %fy, %fx, %fx
```

Case 3:

```
LD [%rn], %fx+1
FPOPd(1) %fx, %fy, %fy
```

Case 4:

```
LD [%rn], %fx+1
FPOPd(1) %fy, %fx, %fy
```

Case 5:

```
LD [%rn], %fx+1
FPOPd(2) %fx, %fx, %fx
```

Case 6:

```
LD [%rn], %fx+1
SQRTd %fx, %fx
```

Notes: 1. *FPOPd* is one of FADDd, FSUBd, FMULd or FDIVd.

2. *FPOPd* is one of FADDd or FMULd (FSUBd and FDIVd operate as expected).

Workarounds

If direct control over assembly language is possible, simply insert a NOP before the double-precision floating-point instruction (case 1 to 6):

LD [%r_n], %f_{x+1}
NOP
FPOPd <same registers set as described above>

If direct control over assembly language is not possible (high-level programming language such as C), checking the SPARC binary code against any of the six above mentioned faulty sequences of instructions shall be done using the code-checker program provided by Atmel (*search for* doc7787 on Atmel web site).

Although there is a very low likelihood of occurrence with high-level programming languages, customers facing this problem should contact the SPARC hotline (sparc-applab.hotline@nto.atmel.com).

14. Meaningless PCI Class Code

The PCI Class Code value (0xB) in the PCIID2 register (0x80000108) of the AT697E is meaningless and leads to the device not being properly recognized by PC BIOS's.

Workarounds

Do not rely on the PCI Class Code.

15. Deadlock with delayed PCI reads during long AHB wait states

When a read request arrives at the AT697E PCI target while the internal AHB bus is in a wait state, the read is not immediately registered into the AHB state machine but is stored in an intermediate register. If during that same AHB wait-state a PCI target write occurs, it will be propagated through the FIFO ('delayed read' feature allowing a write to overtake a read), and it will overwrite the previous read request in the intermediate register.

The PCI target state machine remains locked into the read transaction, but data will never be delivered, and the PCI remote master (who requested this read) also remains locked into the read request, since it is supposed to retry the read request until it obtains data, unless it has a retry timeout mechanism which terminates the request.

In conclusion, the AT697E PCI target does neither deliver the requested data, nor does it accept any new read. Meanwhile, all further write requests coming from remote PCI masters are correctly processed by the AT697E PCI target.

Note there is no problem, if the PCI target read request itself, once started on AHB, is subject to long wait states.

The error is only applicable to systems in which several remote master devices on the PCI bus can access the AT697E PCI target interface. Systems with no or only a single remote PCI master are not affected.

Workarounds

In a system in which several remote master devices on the PCI bus can access the AT697E PCI target interface:

- Reduce the probability for the above mentioned condition to occur by reducing the wait-states on the internal AHB bus. This means for example avoid accesses to high-wait state memory such as PROM, or IO accesses controlled by BRDYN while several external PCI masters may be accessing the AT697E.
- Implement a recovery mechanism: The external PCI master whose read request is not satisfied should time out after a certain number of retries. For example if the external PCI master is also an AT697, then after a number of retries defined by bit 15:8 of the PCIRT register (0x80000140), retry is abandoned, and a system error is generated. This timeout information should be forwarded to the AT697E CPU, e.g. by an interrupt from the external master or by polling a status bit. The CPU should then reset the AT697E target interface by writing "all-ones" to the PCITSC register (0x80000160). One should note that due to this reset, the above mentioned read request is lost and also other external write requests pending in the PCI FIFO might be lost.

16. Memory Block Protection

When both Memory Block Protection units in the AT697E operate in non-overlapping segment mode it is not possible to write to any memory areas.

Workarounds

Do not enable the 2 Memory Block Protection units together in non-overlapping segment mode.

Enabling Unlimited Possibilities™

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan G.K.
16F Shin-Osaki Kangyo Building
1-6-4 Osaki
Shinagawa-ku, Tokyo 141-0032
JAPAN
Tel: (+81)(3) 6417-0300
Fax: (+81)(3) 6417-0370