



ABOUT ADSP-21371/ADSP-21375 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC® ADSP-21371/ADSP-21375 product(s) and the functionality specified in the ADSP-21371/ADSP-21375 data sheet(s) and the Hardware Reference book(s).

SILICON REVISIONS

A silicon revision number with the form "-x.x" is branded on all parts (see the data sheet for information on reading part branding). The silicon revision can also be electronically read by reading the REVPID register either via JTAG or DSP code.

The following DSP code can be used to read the register:

```
<UREG> = REVPID;
```

Silicon REVISION	REVPID[7:4]
0.0	0000

ANOMALY LIST REVISION HISTORY

The following revision history lists the anomaly list revisions and major changes for each anomaly list revision.

Date	Anomaly List Revision	Data Sheet Revision	Additions and Changes
06/30/2010	G	Rev C	Added anomalies: 09000025 .
06/07/2009	F	Rev B	Added anomalies: 09000023 and 09000024 .
02/02/2009	E	Rev B	Revised anomaly 09000018 . Added anomalies: 09000019 , 09000020 , 09000021 , and 09000022 .
06/05/2008	D	Rev A	Added anomalies: 09000014 , 09000015 , and 09000018 .
03/21/2008	C	Rev A	Consolidated anomaly lists for ADSP-21371/ADSP-21375. Added new anomalies 09000009 and 09000011 . Consolidated old anomalies 09000006 (for ADSP-21371) and 09000001 (for ADSP-21375) into 09000013 , and 09000007 (for ADSP-21371) and 09000002 (for ADSP-21375) into 09000007 .
01/03/2007	B	PrA	Removed anomalies: 09000008-This information is added to the HRM
11/06/2006	A	PrA	Initial release

SHARC and the SHARC logo are registered trademarks of Analog Devices, Inc.

NR003347G

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-21371/ADSP-21375 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.0
1	09000007	First bit of IDCODE register will not be a logical ONE when shifted out via boundary scan controller	x
2	09000009	BTC failure during external memory access	x
3	09000011	Indirect Branches from External to Internal Memory may corrupt the Instruction Cache, under Special Conditions	x
4	09000013	Parallel EPROM/FLASH boot following a Hard (Hot) Reset may fail if issued in the middle of external SDRAM reads	x
5	09000014	Incorrect Execution of Conditional External data accesses in a delayed branch (DB) slot	x
6	09000015	Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers	x
7	09000018	Specific Multiplier operations must not be part of the same Instruction as an External Memory access	x
8	09000019	EMUCLK/EMUCLK2 counters do not function correctly in presence of external core data accesses	x
9	09000020	Wrong instruction address may be cached when PMDA instruction executing from external memory is interrupted	x
10	09000021	IOP Register access immediately following an External Memory access may not work	x
11	09000022	Effect latency may be 2 cycles instead of 1 for external data accesses	x
12	09000023	Writes to LCNTR, CURLCNTR and LADDR from Internal Memory may fail if there is a DMA block conflict	x
13	09000024	Specific Core Stall may not execute as expected	x
14	09000025	External port arbitration may not work as expected when the lower priority request is not continuous	x

Key: x = anomaly exists in revision

. = Not applicable

DETAILED LIST OF SILICON ANOMALIES

The following list details all known silicon anomalies for the ADSP-21371/ADSP-21375 including a description, workaround, and identification of applicable silicon revisions.

1. 09000007 - First bit of IDCODE register will not be a logical ONE when shifted out via boundary scan controller:

DESCRIPTION:

The LSB of the newly introduced device identification register (IDCODE) is a 1 as per the IEEE 1149.1 standard.

Auto-detection of the existence of this register by the test-scan controller (which usually happens as soon as the first bit is shifted out) will fail because in this case, the first bit that will be shifted out is not the LSB but is instead the MSB.

WORKAROUND:

Please use the processor-specific BSDL files provided by Analog Devices, and ignore the syntax error that will occur for the LSB (bit 0) of the IDCODE register.

APPLIES TO REVISION(S):

0.0

2. 09000009 - BTC failure during external memory access:

DESCRIPTION:

The background telemetry channel (BTC) will not work if the processor core is accessing external memory data. External memory execution will not interfere with BTC operation.

WORKAROUND:

Do not use BTC if the processor is expected to fetch data from external memory.

APPLIES TO REVISION(S):

0.0

3. 09000011 - Indirect Branches from External to Internal Memory may corrupt the Instruction Cache, under Special Conditions:

DESCRIPTION:

When executing code from external memory with the cache enabled, if the following conditions are all true:

1. An indirect branch instruction is encountered (JUMP/CALL with branch address provided by a pair of modifier and index registers).
2. The target address of the indirect branch instruction points to a location in internal memory address space.
3. A redirection of instruction fetch sequence occurred just prior to the indirect branch (this could either be by an explicit instruction such as a JUMP, CALL, RTS, etc., or implicitly by servicing an interrupt, looping back within a loop, etc.).

then the third instruction following the indirect branch instruction is incorrectly cached by the program sequencer. The following examples illustrate scenarios where these conditions are satisfied:

```
// following code resides in external memory
jump _some_location;
jump(m12,i12); // target points to internal memory
nop;
nop;
r4=5; // <-- this instruction is improperly cached
```

Another example:

```
// following code resides in external memory
lcntr=<count>, do (pc, end) until lce;
...
...
call(m12,i12); // target points to internal memory
nop;
end: nop;
r4=5; // <--- this instruction is improperly cached
```

WORKAROUND:

1. Ensure that 3rd instruction following any indirect branch instruction in external memory never gets executed. An example code sequence as shown below would accomplish this:

```
// following code resides in external memory
jump _some_location;
call (m12,i12); // target is in internal memory
nop;
jump(pc,2);
<instruction_3>; // <-- this instruction is improperly cached
<instruction_4>; // start of useful code.
```

Note that for the above workaround to work, <instruction_3> must not be a target of any other branch.

2. Avoid any indirect branch in code residing in external memory with target in internal memory. If the target address of the branch does not point to an internal memory location, this anomaly will not be experienced.
3. Add a "**FLUSH CACHE;**" instruction at any one of the following locations:
 - All possible target addresses of concerned indirect branch instructions.
 - Just before every concerned indirect branch instruction.
 - Convert all concerned indirect branches to delayed branches using (DB)modifier, and place the "**FLUSH CACHE;**" instruction in one of the two delayed-branch slots.
4. Disable cache functionality by setting CADIS bit in **MODE2** register. Please note that performance hit will be observed in this case, because entire instruction caching functionality (internal as well as external memory) gets disabled.

APPLIES TO REVISION(S):

0.0

4. 09000013 - Parallel EPROM/FLASH boot following a Hard (Hot) Reset may fail if issued in the middle of external SDRAM reads:

DESCRIPTION:

If the processor is in the process of fetching data from external SDRAM memory, or executing code from external SDRAM memory, initiating a Hard (hot) reset during this process may cause the SDRAM device to continue to drive the data bus during the entire duration of $\overline{\text{RESET}}$ being asserted, and also failing to relinquish the external data bus even when the DSP comes out of $\overline{\text{RESET}}$.

If the DSP had been configured to boot via parallel external FLASH or EPROM (which shares part of the same data bus with the external SDRAM), the resulting bus contention on the external data bus between the DSP trying to boot from the FLASH memory and the SDRAM continuing to drive the data bus causes incorrect boot data to be loaded in, and the booting fails. Once the above state is hit, subsequent DSP Resets will not correct the situation. A power-cycling sequence is required to recover from this state.

If a microcontroller or host processor controls the $\overline{\text{RESET}}$ pin of the ADSP-21375 and is responsible for performing the Reset of the DSP, a simple communication protocol between the microcontroller/host processor and the DSP, that ensures that the DSP's program execution jumps to internal memory (in case of external code execution) or that any DMA transfer/external data fetch is completed (in case of data accesses from external SDRAM) before $\overline{\text{RESET}}$ is asserted, will avoid this problem.

Note that the probability of encountering this problem is greatly reduced during normal mode of DSP operation, i.e.,

- when the cache is enabled (CADIS bit of the MODE2 register is not set)
- if Bit 31 of the SDCTL register (NO_BSTOP bit) is set to disable burst-mode accesses from the SDRAM.

The above conditions reduce the probability of encountering this problem, but are not suggested as workarounds. For this, please refer to the techniques listed in the workaround section. This anomaly is not observed during writes to SDRAM memory. It is only seen on reads from external SDRAM, and sequencer fetches from external SDRAM. The anomaly is also not observed during other Reset scenarios such as Power-on Reset, software Reset, or Running Reset.

WORKAROUND:

Software:

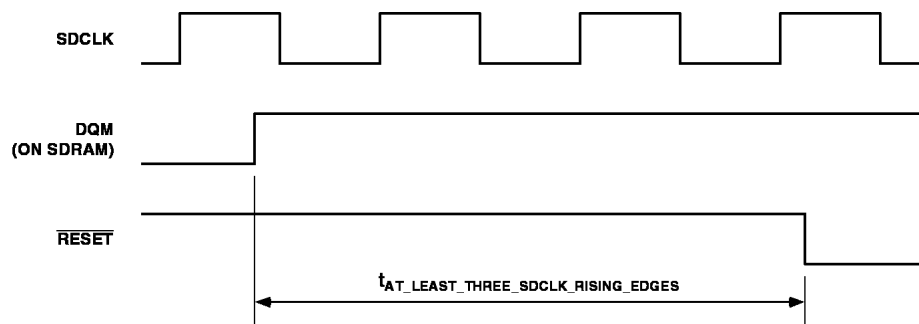
Configure the DSP to boot via SPI port (master or slave) and perform the following sequence within the loader kernel:

- Initialize the SDCTL and SDRRC registers to the appropriate values
- Perform a dummy read access of external SDRAM memory.

This has the effect of resetting and re-initializing the states of the SDRAM controller as well as that of the external SDRAM memory, and avoid hitting the problem state.

Hardware:

Providing an external delay of at least 3 SDCLK clock cycles between the SDRAM's DQM signal and the $\overline{\text{RESET_IN}}$ to the DSP will also cause the problem to be avoided (see attached Figure).



APPLIES TO REVISION(S):

0.0

5. 09000014 - Incorrect Execution of Conditional External data accesses in a delayed branch (DB) slot:

DESCRIPTION:

Conditional external data access instructions in the delay slots of a branch instruction (such as JUMP/CALL/RTS/RTI) maybe incorrectly executed because the evaluation of the condition maybe wrong. This applies to both internal and external memory execution.

For example:

```
// Code residing in either internal or external memory
jump (some_addr) (DB);
<instruction_1>;
if eq r0=dm(ext_addr); //Condition evaluation may be wrong
```

WORKAROUND:

Do not use conditional external data access instructions in the delay slots of any branch instruction.

APPLIES TO REVISION(S):

0.0

6. 09000015 - Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers:

DESCRIPTION:

If a delayed branch modifier (DB) is used to return from the interrupt service routines of any of IRQx (hardware) or timer interrupts, the automatic popping of ASTATx/ASTATy/MODE1 registers from the status stack may go wrong.

The specific instructions affected by this anomaly are "RTI (DB);" and "JUMP (CI) (DB);".

This anomaly affects only IRQx and Timer Interrupts as these are the only interrupts that cause the sequencer to push an entry onto the status stack. This anomaly applies to both internal and external memory execution.

WORKAROUND:

Do not use (DB) modifiers in instructions exiting IRQx or Timer ISRs. Instructions in the delay slots should be moved to a location prior to the branch.

APPLIES TO REVISION(S):

0.0

7. 09000018 - Specific Multiplier operations must not be part of the same Instruction as an External Memory access:

DESCRIPTION:

Specific Multiplier operations where Multiplier Results Registers (MRF/MRB) are used as a destination, must not be part of the same instruction as an External Memory access.

If such an operation is combined with external data access, the MR* registers will not get updated with expected value. This applies to both internal and external memory code execution.

Further, if any multiplier operation is part of the same instruction as an external memory data access, the corresponding status flags ASTATx/y and STKYx/y registers may not be correctly updated and interrupts associated with them may also not get triggered.

WORKAROUND:

Ensure that these specific Multiplier operations are on a separate instruction from external memory accesses.

APPLIES TO REVISION(S):

0.0

8. 09000019 - EMUCLK/EMUCLK2 counters do not function correctly in presence of external core data accesses:

DESCRIPTION:

EMUCLK and EMUCLK2 counters do not increment correctly for core accesses involving external memory. As a result, they count lesser than actual intervals in presence of external data accesses. This anomaly is specific only to core driven data accesses involving external memory. DMA accesses and/or instruction accesses are not affected.

WORKAROUND:

1. Use other counters if core accesses to external memory are required.
2. Avoid core accesses involving external memory while using EMUCLK and EMUCLK2 counters.

APPLIES TO REVISION(S):

0.0

9. 09000020 - Wrong instruction address may be cached when PMDA instruction executing from external memory is interrupted:

DESCRIPTION:

While executing a PMDA instruction (one that contains a PM() access) from external memory, if the core is interrupted then the instruction being cached at that time may not get cached properly. Note that the anomaly is limited to instructions containing PMDA instructions. It does not affect all instructions being executed from external memory.

WORKAROUND:

1. If a user application contains PMDA instructions that are executed from external memory and interrupts may occur and need to be serviced during this interval, then insert a "flush cache" instruction as the first instruction within affected Interrupt service routines (ISRs).
2. Disable cache using CADIS bit in MODE2. This will disable both internal as well as external memory cache, thereby reducing performance.
3. If possible, the user may disable and then re-enable global interrupts around PMDA instructions executing from external memory. Care must be taken to account for any latencies involved when disabling the interrupts.

APPLIES TO REVISION(S):

0.0

10. 09000021 - IOP Register access immediately following an External Memory access may not work:

DESCRIPTION:

If an instruction making an access to an IOP register immediately follows another instruction that performs an access to external memory, the IOP register access may not occur correctly.

WORKAROUND:

Separate the two instructions by inserting another instruction in between them, such as a NOP.

APPLIES TO REVISION(S):

0.0

11. 09000022 - Effect latency may be 2 cycles instead of 1 for external data accesses:**DESCRIPTION:**

The following registers that have an effect latency of 1 (the maximum number of instructions it takes for a write to these registers to take effect) will instead have an effect latency of 2 if any of their bits impact an instruction containing an external data access:

`MODE1, MODE2, MMASK, ASTATx, ASTATy, STKYx, and STKYy.`

For example, consider the following sequence of instructions:

```
bit set MODE1 BR8;
nop;           //Sufficient if not immediately followed by external memory access instruction
nop;           //Extra NOP needed if following instruction accesses external memory
pm(i8,m12)=f9; //i8 is pointing to an address in external memory
```

Registers other than the ones listed above are not affected by this anomaly.

Note that the anomaly is independent of whether the instruction itself resides in internal or external memory. Rather, the anomaly is encountered if there are external memory data accesses within the two instructions immediately following the register modification.

WORKAROUND:

If any of the above registers with an effect latency of 1 is modified, it is recommended that no accesses involving external memory (over either PM or DM bus) are performed in the two instructions immediately following the register modification. It is recommended to insert two NOPs after such register modifications.

APPLIES TO REVISION(S):

0.0

12. 09000023 - Writes to LCNTR, CURLCNTR and LADDR from Internal Memory may fail if there is a DMA block conflict:**DESCRIPTION:**

Write to LCNTR, CURLCNTR and LADDR from internal memory (either as a DM access or as a PM access) fails when a DMA transfer to/from the same block occurs in the same cycle.

For example, consider the following instruction:

```
CURLCNTR = dm (i0,m0);
```

Now consider any DMA access involving the same memory block as pointed to by address (I0 + M0). If the DMA and the above write align in such a way that DMA transfer happens in the same cycle as the above instruction, then the above write will fail.

Also note that the anomaly also occurs if (I0 + M0) points to a memory-mapped I/O register.

WORKAROUND:

- 1) Change the DMA to source/target a different internal memory block thereby avoiding any DMA block conflict.
- 2) Instead of loading these registers directly from memory, they can be loaded indirectly as a 2-step process as shown below:

```
r0 = dm (i0,m0);
CURLCNTR = r0;
```

APPLIES TO REVISION(S):

0.0

13. 09000024 - Specific Core Stall may not execute as expected:

DESCRIPTION:

Under certain specific conditions outlined below, one type of core stall that is normally incurred does not get executed. In certain cases this can cause incorrect code operation.

When PCSTK is loaded, and an RTI/RTS instruction is executed immediately afterward, there is a stall as the return waits for a writeback of PCSTK before the return.

Example:

```
[1] PCSTK = DM(I0,M0);  
[2] RTS;
```

If: The memory access in instruction 1 is to a memory-mapped IOP register.

-OR-

DMA is simultaneously accessing the same bank as the memory access in instruction 1.

Then: Stall does not occur, and this results in the RTI/RTS vectoring to some undefined location instead of the value from PCSTK.

WORKAROUND:

Add a nop between instructions 1 and 2.

APPLIES TO REVISION(S):

0.0

14. 09000025 - External port arbitration may not work as expected when the lower priority request is not continuous:

DESCRIPTION:

The external port arbitration logic between core, EPDMA0, and EPDMA1 (configurable using EPBRx and DMAPRx bits of the EPCTL register), may behave unexpectedly when the lower priority request is single/non-sequential. Some scenarios where the external port request may be single/non-sequential are:

a) Single core external access

```
Non-external access instruction..
R0= dm (<external memory address>);
Non-external access instruction..
```

b) Interrupted back to back core accesses

```
r0=dm(<external memory address>);
Interrupt occurs here....
r1=dm(<external memory address>);
```

c) A DMA transfer with count =1

d) A DMA transfer with count greater than one, but the external port access requests not continuous because of some internal memory conflicts.

In general, the lower priority access should complete only when there is no pending higher priority request. But, because of the anomalous behavior, the lower priority access may still be completed even if there is a pending higher priority request. This behavior is applicable for both AMI and SDRAM accesses.

WORKAROUND:

There is no workaround for this anomaly.

APPLIES TO REVISION(S):

0.0