ANALOG DEVICES

SHARC Processor

Silicon Anomaly List — ADSP-21483/21486/21487/21488/21489

ABOUT ADSP-21483/21486/21487/21488/21489 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC® ADSP-21483/21486/21487/21488/21489 product(s) and the functionality specified in the ADSP-21483/21486/21487/21488/21489 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]
0.1	0000*

* - See anomaly 15000017

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
03/17/2010	B	PrA	Title of the document is updated to reflect correct part numbers
02/24/2010	A	PrA	Initial release

NR004008B

One Technology Way, P.O.Box 9106, Norwood, MA 02062-9106 U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.461.3113

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-21483/21486/21487/21488/21489 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.1
1	15000002	Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifier	x
2	15000003	IOP Register access immediately following an External Memory access may not work	x
3	15000004	Effect latency of some System Registers may be 2 cycles instead of 1 for External data accesses	x
4	15000005	Writes to LCNTR, CURLCNTR and LADDR from Internal Memory may fail if there is a DMA block conflict	x
5	15000010	Incorrect value when the results of Enhanced Modify/BITREV Instruction are used in the very next Instruction	x
6	15000012	Latency with external FLAG-based Conditional instructions involving DAG register post-modify operation	x
7	15000014	Special PLL Initialization Sequence required if MediaLB interface is used in DMA-driven transfer mode	x
8	15000016	When PM accesses are used, some Instructions may get corrupted under specific conditions	x
9	15000017	Incorrect Silicon revision number in REVPID register	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-21483/21486/21487/21488/21489 including a description, workaround, and identification of applicable silicon revisions.

1. 15000002 - Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifier:

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 not work correctly.

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 IRQ or Timer ISRs. Instructions in the delay slot should be moved to a location prior to the branch.

Note: This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and Operating Systems supported by ADI, such as VisualDSP++ and VDK please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

APPLIES TO REVISION(S):
0.1

2. 15000003 - 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):
0.1

3. 15000004 - Effect latency of some System Registers 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, SYSCTL, BRKCTL, 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):
0.1

4. 15000005 - Writes to LCNTR, CURLCNTR and LADDR from Internal Memory may fail if there is a DMA block conflict:

DESCRIPTION:
Writes to LCNTR, CURLCNTR and LADDR from internal memory (either as a DM access or as a PM access) may fail 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):
0.1

5. 15000010 - Incorrect value when the results of Enhanced Modify/BITREV Instruction are used in the very next Instruction:

DESCRIPTION:
In the following specific sequence of instructions, the memory or the register load in **INSTR3** will not contain the correct updated value of the DAG register **Ia** from **INSTR2**, but rather its value from **INSTR1**:

```
INSTR1: Ia = <immediate load | register load | memory load>;
INSTR2: Ia = MODIFY|BITREV (Ib, Mc);
INSTR3: <memory load | register load>  = Ia;
```

Note that this anomaly is only applicable in the case where **Ia** and **Ib** are unique and different.

WORKAROUND:
The user must avoid the above exact sequence of instructions which might produce an incorrect result.

Note: This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and Operating Systems supported by ADI, such as VisualDSP++ and VDK please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

APPLIES TO REVISION(S):
0.1

6. 15000012 - Latency with external FLAG-based Conditional instructions involving DAG register post-modify operation:

DESCRIPTION:
External FLAG-based Conditional instructions involving DAG register post-modify operation must not be followed immediately by an instruction that uses the same index register.

For example, in the following instruction sequence shown below:

```
INSTR1: IF COND dm(Ia,Mb); //any instruction that involves post-modify operation
INSTR2: dm(Ia,Mc); //any instruction that depends on updated Ia value
```

The value of the DAG index register in **INSTR2** will either be **Ia** or **(Ia+Mb)** depending on whether **INSTR1** was aborted or executed.

In the unique case where **COND** is an external **FLAG** condition (for example, say **FLAG2_IN**) which is set asynchronously by an external source or event, the necessary internal stalls which would result in the DAG index register getting the correct value do not take effect, and consequently the value of the DAG index register in **INSTR2** may not contain the correct and expected value.

WORKAROUND:
Separate the instructions in the above sequence by at least two NOPs.

APPLIES TO REVISION(S):
0.1

7. 15000014 - Special PLL Initialization Sequence required if MediaLB interface is used in DMA-driven transfer mode:

DESCRIPTION:
The MediaLB interface's DMA clock may lose synchronization with the processor's internal clock if the procesor's PLL is placed in bypass mode as part of the initial initialization sequence.

WORKAROUND:
Use the following instruction sequence while initializing the PLL if using MediaLB interface in DMA-driven transfer mode:

1. Disable clock to the MediaLB interface.

```
ustat4 = dm(PMCTL1);
bit set ustat4 MLBOFF;
R2=dm(MLB_VCCR); // any "dummy" read of an IOP register outside of the core to force
                 // clock synchronization between core and peripheral clock domains
dm(PMCTL1) = ustat4;
```

2. Place PLL in bypass mode.
3. Proceed with programming PLL parameters as per default guidelines. Provide sufficient delay in order for the changes to take effect, again as per default guidelines.
4. Bring PLL out of bypass mode and re-enable the clock to the MediaLB interface:

```
ustat1 = dm(PMCTL);
bit clr ustat1 PLLBP;
ustat4 = dm(PMCTL1);
bit clr ustat4 MLBOFF;
dm(PMCTL) = ustat1;
dm(PMCTL1) = ustat4;
```

Ensure that the above instruction sequence is executed from within internal memory, is not interrupted, and that there is no background DMA activity.

Note that the above procedure does not need to be followed if the MediaLB interface is not used in a system, or if the MediaLB interface is expected to operate only with core-driven data transfers.

APPLIES TO REVISION(S):
0.1

8. 15000016 - When PM accesses are used, some Instructions may get corrupted under specific conditions:

DESCRIPTION:
When specific PM accesses are used, either the PM instruction or some Instructions which follow this instruction may get corrupted. The problem is seen when the PM accesses has any of the below conditions:

1. Conflicts with another core/DMA access to the same memory block
2. Accesses any of the memory mapped(IOP) registers
3. Accesses the external memory space

CASE 1 (applicable for both VISA and NON-VISA mode):
The single instruction loop which has the PM access instruction described above as part of the instruction in the loop, may not work as expected. The PM instruction in the loop while being fetched from the cache may get corrupted. This is applicable for both counter based and non-counter based loops. For counter based loops the corruption occurs only if the count value is greater than four.

Example1:

```
lcntr=x, do (pc,1) until lce;    // x > 4
dm(I0,M0)=R10, pm(I12,M10)=R10;  //PM access meets one of the conditions described
```

Example2:

```
lcntr=x, do (pc,1) until lce;    // x > 4
R10 = pm(I12,M10);               //PM access meets one of the conditions described
```

Example3:

```
do (pc,1) until forever;
R10 = pm(I12,M10);              //PM access meets one of the conditions described
```

CASE 2 (applicable only for VISA mode):
A code sequence which includes two successive compressed PM instructions and the second PM access instruction meets the one of the conditions described above, may not work as expected. The instruction(s) following the second PM access (while getting fetched from the cache) may get corrupted. The particular instruction(s) which get corrupted is dependent on the size of the instructions following the above sequence and their alignment in the Instruction Alignment Buffer (IAB). This anomaly is not applicable for the pm sequence which is formed by the first and last instruction of a hardware loop containing pm accesses.

Example1:

```
pm(I12,M10)=0x5c226;             //PM access 1
dm(I0,M0)=R10, pm(I12,M10)=R10;  //PM access 2, meets one of the conditions described
...
...
```

Instruction(s) following this sequence may get corrupted.

Example2:

```
pm(I12,M10)=0x5c226;     //PM access 1
R10 = pm(I12,M10);       //PM access 2, meets one of the conditions described
...
...
```

Instruction(s) following this sequence may get corrupted.

Note that this anomaly is seen especially at low operating temperatures, but it is not limited to any particular operating temperature range.

WORKAROUND:

Applicable for CASE 1:

1. Avoid memory block conflict stalls for these scenarios by moving either the core/DMA access or the PM access to different memory block.

2. Avoid single instruction loops having program memory access by unrolling the loop. This can be done by one of the following ways:
 a. Replicating the instruction and correspondingly reducing the loop count
 b. Adding another instruction or NOP instruction to the loop
 c. Breaking the instruction which contains pm access into two separate instructions.

3. Disable the cache for the problematic PM accesses.

```
BIT SET MODE2 CADIS;
nop;
nop;
lcntr=x, do (pc,1) until lce;   // x > 4
dm(I0,M0)=R10, pm(I12,M10)=R10; // PM access meets one of the conditions described
BIT CLR MODE2 CADIS;
nop;
nop;
```

Note that disabling the cache may affect the performance of the hardware loop. The PM instruction in the loop will always stall and the loop will take double the time than the cache enabled case.

Applicable only for CASE 2:

1. Avoid memory block conflict stalls for these scenarios by moving either the core/DMA access or the PM access to different memory block.

2. Add a "nop" or a non-PM access instruction between the two problematic PM accesses.

3. All sequences of PM accesses with length more than 1 should remain uncompressed. This can be done by using the assembler directive ".NOCOMPRESS".

```
.NOCOMPRESS;                    //Disable compression
pm(I12,M10)=0x5c226;            //PM access 1
dm(I0,M0)=R10, pm(I12,M10)=R10; //PM access 2, meets one of the condition described
.COMPRESS;                      //Enable compression
```

4. Replace the pm access with the dm access.

5. Disable the cache for the problematic PM accesses.

```
BIT SET MODE2 CADIS;
nop;
nop;
pm(I12,M10)=0x5c226;            //PM access 1
dm(I0,M0)=R10, pm(I12,M10)=R10; //PM access 2, DM access points to the same
                                //memory block resulting in a block conflict
BIT CLR MODE2 CADIS;
nop;
nop;
```

Note: Some of these workarounds may be built into the development tool chain and/or into the operating system source code. For tool chains and Operating Systems supported by ADI, such as VisualDSP++ and VDK please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

APPLIES TO REVISION(S):

0.1

9. 15000017 - Incorrect Silicon revision number in REVPID register:

DESCRIPTION:
The REVPID register bits 4-7 do not contain the correct silicon revision information.

WORKAROUND:
None.

APPLIES TO REVISION(S):
0.1