ANALOG DEVICES

# SHARC Processor

Silicon Anomaly List **ADSP-21368**

## ABOUT ADSP-21368 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC® ADSP-21368 product(s) and the functionality specified in the ADSP-21368 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] |
|---|---|
| 0.2 | 0010 |
| 0.1 | 0001 |
| 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 |
|---|---|---|---|
| 04/08/2011 | I | E | Modified anomaly: 08000025, Added Anomalies: 08000030 & 08000031 |
| 08/23/2010 | H | E | Added anomaliy: 08000029 |
| 06/16/2009 | G | D | Modified anomalies: 08000001, Added common note on Tools action for all the core related anomalies |
| 01/20/2009 | F | D | Added anomalies: 08000025,08000026 |
| 03/16/2007 | E | A | Modified anomalies: 08000001, 08000014, 08000015, 08000016, 08000017, 08000018, 08000019, 08000020, 08000024 |
| 01/05/2007 | D | A | Added anomaly: 08000024<br>Modified anomaly: 08000020 |
| 08/14/2006 | C | A | Added anomalies: 08000018, 08000019, 08000020<br>Modified anomalies: 08000014, 08000017 |
| 03/21/2006 | B | PrC | Added anomalies: 08000014, 08000015, 08000016, 08000017<br>Modified anomalies: 08000002, 08000003, 08000004, 08000005, 08000006, 08000008, 08000009, 08000010, 08000011, 08000012 |
| 09/13/2005 | A | PrC | Initial release |

NR003131I

**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 ©2011 Analog Devices, Inc. All rights reserved.**

## SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-21368 anomalies and the applicable silicon revision(s) for each anomaly.

| No. | ID | Description | 0.0 | 0.1 | 0.2 |
|---|---|---|---|---|---|
| 1 | 08000001 | Core Stall not executed properly | x | x | x |
| 2 | 08000002 | Memory write operations can fail under certain conditions while DMA to internal memory is in progress | x | . | . |
| 3 | 08000003 | SDRAM Refresh Specification violated after bus transfer | x | . | . |
| 4 | 08000004 | Slave DSP writes to external memory lost | x | . | . |
| 5 | 08000005 | Shared Memory system can hang under specific conditions | x | . | . |
| 6 | 08000006 | tDCAD specification violated | x | . | . |
| 7 | 08000008 | Bus Arbitration can fail if a slave arbitrates but does not receive the bus | x | . | . |
| 8 | 08000009 | SDRAM Timing Violations following bus transfer | x | . | . |
| 9 | 08000010 | Shared Memory signals fails for PLL ratio = 44 | x | . | . |
| 10 | 08000011 | SDRAM Precharge not issued after bus transition | x | . | . |
| 11 | 08000012 | SDRAM Autorefresh not working properly in shared memory system | x | . | . |
| 12 | 08000014 | Instruction fetch fails when there is a CORE-DMA conflict coupled with an external memory access | x | x | . |
| 13 | 08000015 | Elevated core voltage needed for operation above 266 MHz | x | x | . |
| 14 | 08000016 | SPI DMA issue when using SPIBAUD=2 | x | x | x |
| 15 | 08000017 | 166MHz SDRAM operation violates tSSDAT specification | x | x | . |
| 16 | 08000018 | TCK can effect external PM bus accesses under certain circumstances | x | x | . |
| 17 | 08000019 | PDAP must use DAI pins, external port pins not available for PDAP data | x | x | x |
| 18 | 08000020 | Input Shift Register Anomaly in ASRCs (Asynchronous Sample Rate Converters) impacts Daisy-Chained TDM mode | x | x | . |
| 19 | 08000024 | Parallel EPROM/FLASH boot following a Hard (Hot) Reset may fail if issued in the middle of external SDRAM reads | x | x | x |
| 20 | 08000025 | In a shared memory system, the processor core hangs, when the other processor requests for the bus during an AMI access | x | x | x |
| 21 | 08000026 | Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers | x | x | x |
| 22 | 08000028 | Conditional FLAG instructions involving DAG index registers must not be followed immediately by an instruction that uses the same index register | x | x | x |
| 23 | 08000029 | External port arbitration may not work as expected when the lower priority request is not continuous | x | x | x |
| 24 | 08000030 | VDSP Statistical profiler may provide incorrect profile information for idle instructions | x | x | x |
| 25 | 08000031 | After an emulator halt at the instruction before ‘idle' instruction, the Core Timer stops decrementing even after code execution restarts | x | x | 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-21368 including a description, workaround, and identification of applicable silicon revisions.

## 1. 08000001 - Core Stall not executed properly:

### 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 unexpected code operation.

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

```
Example1:
[1] PCSTK = DM(I0,M0);
[2] RTS;

Example2:
[1] PCSTK = DM(I0,M0);
[2] RTI;
```

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

-OR-
The memory access in instruction 1 is to External Memory.

-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 RTS/RTI vectoring to some unknown location instead of the value from PCSTK.

### WORKAROUND:

Add a nop between instructions 1 and 2.

**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):

0.0, 0.1, 0.2

## 2. 08000002 - Memory write operations can fail under certain conditions while DMA to internal memory is in progress:

**DESCRIPTION:**
If an instruction modifies a register and the same register is the source to a memory write, and a DMA happens to same block as the memory write in the next cycle, under the following conditions, the modified value of the register is written to memory instead of the old one.

Examples of instructions that modify a register and the same register are written to memory are:

```
1. R0 = R1-R2, DM(I0,M0) = R0; // The memory access can be either DM/PM
2. DM(I0,M0) = R0, R0 = PM(I8,M8);
```

The conditions under which we see the issue are:
1. The failing instruction is the first instruction in a 1,2, or 4 instructions long loop.

Example:

```
lcntr = 8, do (pc,1) until lce;
DM(I0,M0) = R0, R0 = PM(I8,M8); //and DMA occurs in next cycle.
// The new value of R0 (fetched through PM) is written into DM memory
lcntr=0x7, do ST2_IN_BFLY_T2 until lce; //2 instr long loop
f4=f2+f4, dm(i3,m6)=r0, r0=pm(i11,m11);
ST2_IN_BFLY_T2: f4=pass f2, dm(i4,m6)=r4, r2=pm(i11,m11);
```

2. When the compute of the failing instruction generates both the operands of multiplier in the next instruction
Example:

```
F0=F0+F4, F1=F0-F4, DM(I0,M0) = R1; // and DMA occurs in next cycle.
// The new value of R1 (output of compute) is written into DM memory
F4=F0*F1;
```

3. When the failing instruction is followed by a conditional branch and any of the following two happens
a. A compute in the failing instruction affects the condition of the branch
Example:

```
F0=F0+F4, PM(I10,M10) = R0; // and DMA occurs in next cycle.
//The new value of R0 (output of compute) is written into PM memory
IF EQ JUMP(PC,0x12);
```

b. A compute in the instruction preceding the failing instruction affects the condition of branch
Example:

```
F0=F0+F4;
DM(I0,M0) = R0, R0 = PM(I8,M8); // and DMA occurs in next cycle.
//The new value of R0 (fetched through PM) is written into DM memory
IF EQ JUMP(PC,0x12);
```

4. When the failing instruction contains a floating point multiplication and is followed by compute operation involving any fixed point operand register executing in ALU or shifter.

Example:

```
F0=F0*F4, DM(I0,M0) = R0;
F5=FLOAT R1;
//The new value of R0 (output of multiply) is written into PM memory
```

**WORKAROUND:**
Move DMA to another block of memory OR,
1. For case1, unroll the loop to make the loop length more than 4.

2. For case 2, 3a, 3b and 4 insert an unrelated instruction between the failing instruction and the instruction following it.

**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):**

0.0

## 3. 08000003 - SDRAM Refresh Specification violated after bus transfer:

**DESCRIPTION:**

In a system with multiple ADSP-21368s sharing a bank of SDRAM, one processor is responsible for the SDRAM power up. It's SDRAM Controller will initiate and power-up the SDRAM and then refresh it periodically as expected. When another ADSP-21368 arbitrates for and receives the bus, it will assume responsibility for refreshing the SDRAM. However it is possible that the new bus master will not refresh the SDRAM in enough time to comply with the SDRAM's specification.

**WORKAROUND:**

1. The DSPs other than the one controlling the SDRAM powerup should wait for the SDRAM powerup to complete. GPIO flags or nop-loops can be used to do this.

2. After detecting the power up completion, and before doing any external accesses, the DSPs (exluding the DSP that powered up the SDRAM) should execute the following set of instructions to ensure that a refresh happens.

```
r0 = dm(sdram_addr);       // dummy access to grab the bus
if not BM jump(pc,0);      // wait for busmastership

                           //force auto refresh
ustat1 = dm(SDCTL);
bit set ustat1 FAR;        //FAR is defined as bit 20 of SDCTL.
dm(SDCTL) = ustat1;
```

After this step the DSP can start the external accesses.
These steps need to be repeated if a DSP is reset.

**APPLIES TO REVISION(S):**

0.0

## 4. 08000004 - Slave DSP writes to external memory lost:

**DESCRIPTION:**

If an ADSP-21368 attempts to write to SDRAM while it is a bus slave, the access should be held off until bus mastership is attained. In 0.0 silicon, the access is not held off, so the data will be lost. The data in SDRAM will not be corrupted however, as the DSP's pins are tristated when it is a bus slave.

**WORKAROUND:**

Use the 'IF BM mnemonic' to ensure writes are not made until bus mastership is attained.

**APPLIES TO REVISION(S):**

0.0

## 5. 08000005 - Shared Memory system can hang under specific conditions:

**DESCRIPTION:**
In a system with 3 ADSP-21368s in a shared-memory configuration, if both BMAX and BUSLOCK bits are set, it is expected that the DSP with bus mastership would continue it's external accesses un-interrupted, until it's BUSLOCK bit is cleared, irrespective of the bustimeout feature.

The issue is that when timeout occurs, the bus master keeps the bus, and it's external accesses go into a pending state, causing the shared memory bus to hang. No other DSP will be able to arbitrate for the bus until the bus master is reset.

**WORKAROUND:**
Do not use the BUSLOCK bit and the BUSTIMEOUT features together.

**APPLIES TO REVISION(S):**
0.0

## 6. 08000006 - tDCAD specification violated:

**DESCRIPTION:**
When using the SDRAM Controller with a 166 MHz SDCLK, the SDRAM Controller specification tDCAD (Data delay after SDCLK) is violated on specific data pins for the first access only. It is specified as 4.0 ns max, and may increase to as much as 6.0 ns. This could cause a spec violation for the SDRAM. This violation has not been seen to cause any SDRAM access failures.

**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.0

## 7. 08000008 - Bus Arbitration can fail if a slave arbitrates but does not receive the bus:

**DESCRIPTION:**
If a DSP arbitrates for the bus, and then deasserts it's bus request line before receiving bus mastership, the bus master can lose the bus. This can happen for instance if the slave DSP has a soft reset executed directly after it asserts it's bus request. The master DSP will assume that the bus is taken, and will not get the bus back until another DSP gets bus mastership and relinquishes it.

**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.0

## 8. 08000009 - SDRAM Timing Violations following bus transfer:

**DESCRIPTION:**
In a shared memory system, when a bus transition happens immediately after an autorefresh/selfrefresh command, there is a chance that the new master will violate the tRFC,tXSR,tRAS timing parameters.

```
(tRFC - Autorefresh to auto refresh minimum period)
(tXSR - Self refresh to auto refresh minimum period)
(tRAS - Activate to precharge minimum period)
```

After getting bus mastership, the new master immediately does an autorefresh.
It doesn't have any information about the last autorefresh/selfrefresh command done by the previous DSP master, and so it won't wait for the minimum timing parameters programmed.

This could result in violation of the SDRAM specification.

**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.0

## 9. 08000010 - Shared Memory signals fails for PLL ratio = 44:

**DESCRIPTION:**
In a shared memory system , the external bus request signals may fail to operate properly when the Power Management Control Register is configured to multiply the CLKIN by 44. (i.e. CLKIN ~= 9MHz, CClk = 400MHz) All other ratios will allow the shared memory functionality to work properly.

**WORKAROUND:**
Use a different ratio of CLKIN:CCLK.

**APPLIES TO REVISION(S):**
0.0

## 10. 08000011 - SDRAM Precharge not issued after bus transition:

**DESCRIPTION:**
In a system with multiple ADSP-21368s sharing a common SDRAM, following a transfer of bus mastership the new bus master must precharge the SDRAM, but this precharge is not issued in certain cases.

**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.0

## 11. 08000012 - SDRAM Autorefresh not working properly in shared memory system:

**DESCRIPTION:**
In a shared memory system using SDRAM, the bus master is responsible to maintain the SDRAM control signals and specifications. One of these is a periodic autorefresh command. If the DSP is no longer making accesses to SDRAM, the /BRx signal along with other memory control lines goes into tristate. However the DSP still has bus mastership and is expected to maintain the SDRAM. The issue is that with tristated control signals, the SDRAM is not selected, and therefore does not receive the autorefresh command from the ADSP-21368's SDRAM Controller.

**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.0

## 12. 08000014 - Instruction fetch fails when there is a CORE-DMA conflict coupled with an external memory access:

**DESCRIPTION:**
Instruction fetch may fail under certain conditions:

An instruction fetch may fail if following combination of external accesses, instruction pipeline stall & DMA block conflict occurs. The failure happens only if all three conditions mentioned below are met:

1. Two instructions executing consecutively make data accesses to/from external memory.

2. When the processor core is being stalled for the first access, there is another stall generated at the same time. The later stall can be any of the stalls mentioned in "PROGRAM SEQUENCER" chapter of "ADSP-2136x SHARC Processor- Programming Reference" (pages 3-11 through 3-26 in Revision 1.0, November 2005), with two exceptions:

a) Stalls due to any of the DAG register (index, modifier etc.) dependencies.
b) Stalls generated by emulator.

An instruction making an external data access may get stalled when it is in "address" stage of the instruction pipeline, in order to wait until the external memory device acknowledges the access completion. For details on other stalls, refer to the Programming Reference mentioned above.

3. An internal memory block conflict happens between a DMA and instruction fetch of the 3rd instruction following the last external access instruction. This instruction fetch fails, leading to corruption of the corresponding opcode and in turn producing unpredictable results.

For example, consider following sequences of instructions:

a) Multiplier dependency stall:

```
I1: r12=r5+r4, r0=dm(ext1);
I2: r12=r12*r12, r1=dm(ext2);
I3: <any_instruction>;
I4: <any_instruction>;
I5: <any_instruction>;
```

Here, I1 and I2 are consecutive instructions which perform external memory accesses. Both of them get stalled in their "address" stages in the pipeline due to extra time required for external accesses. When I1 is being stalled, there is another core generated dependency stall happening at the same time. This stall is due to both operands of multiplier in I2 (i.e. R12) depend upon execution of I1 where R12 is getting written.When both these stalls vanish, the pipeline moves ahead where instruction fetch of I5 begins. While I5 is in "fetch1" stage, if there happens to be a DMA to the same block of internal memory where the code is residing, then an internal memory block conflict occurs. Therefore, due to this anomaly, the instruction fetch of I5 fails due to its opcode getting corrupted.

b) The above accesses inside a loop:

```
lcntr = 100 , do (PC,3) until lce;
I1: r12=r5+r4, r0=dm(ext1);
I2: r12=r12*r12, r1=dm(ext2);
I3: <any_instruction>;
```

Here too, the problem will occur as per the same reasoning. The difference is that instead of I5, I2 of the 2nd loop iteration fails.

c) Conditional branch dependency on compute:

```
I1: R0=R5+R8,R2 = DM(I0,M0) ;
I2: IF eq JUMP (pc, 3), ELSE R1=DM(I5,M5);
I3: <any_instruction>;
I4: <any_instruction>;
I5: <any_instruction>;
```

In this case processor core creates a stall due to an another reason, i.e. conditional jump in I2 depending on compute in I1. Fetch of I5 will fail.

Examples of DAG dependency stalls which will NOT have any problems:

```
a)
  I0:    I5 = buffer1;  // buffer1 in external memory.
  I1:    r12=r5+r4,   r0=DM(I0,M1);
  I2:    R1 = DM(I5,M5);

b)
  I0:    R2=R3-R4
  I1:    IF EQ DM(I1,M1) = R15;
  I2:    R0 = DM(I1,M2);
```

**WORKAROUND:**
1. Place code and DMA data in different blocks of internal memory.

2. Separate two instructions making external memory accesses by adding a NOP instruction between them. Any other instructions which do not access external memory can also be used instead of NOP. Note that using this workaround will result in degradation of data throughput to/from external accesses, resulting in slower execution of the code.

**APPLIES TO REVISION(S):**
0.0, 0.1

## 13. 08000015 - Elevated core voltage needed for operation above 266 MHz:

**DESCRIPTION:**
A memory speed path in the internal memory of revision 0.0 and 0.1 material limits the speed at which the memory can be used at the nominal core voltage. This results in a need to raise the core voltage supply. This issue only effects operation above 266 MHz in 0.0 and 0.1 revision silicon. The 266 MHz products in the MQFP package are not effected and should be operated at their datasheet-specified voltages.

Functional testing of 0.2 revision material has shown that this issue has been fixed. While datasheet timing requirements and characteristics for 0.2 silicon are currently being characterized at 1.2v nominal core voltage, revision 0.2 material may be operated at Vddint of 1.2v +/-5% . Note that operation at 400 MHz in all revisions will still require a 1.3v +/-5% Vddint.

The ADSP-21367/8/9 datasheet will be revised to include the updated timing information at 1.2v nominal and 333 MHz core clock rate.

**WORKAROUND:**
1. If using revision 0.0 or 0.1 silicon at or above 333 MHz, adjust the vddINT voltage to 1.3 volts (+/-5%)

**APPLIES TO REVISION(S):**
0.0, 0.1

## 14. 08000016 - SPI DMA issue when using SPIBAUD=2:

**DESCRIPTION:**
When using the SPI peripheral in DMA master receive mode, when the SPI is programmed for back to back DMA access,and when the SPI port is used without being disabled in between the accesses, the SPI can generate an extra clock for a 32bit word even after the FIFO and the receive buffer are full. This extra clock can cause a receive buffer overflow error which will result in a 32bit word getting lost. This occurs only with SPIBAUD=2 setting.

**WORKAROUND:**
1) Do not disable DMA between back to back DMA access.
-OR-
2) If DMA must be disabled between back to back access, diable SPI as well and before reenabling the SPI clear the FIFO and the receive buffer and then enable SPI and then configure the DMA descriptors and enable the DMA.

**APPLIES TO REVISION(S):**
0.0, 0.1, 0.2

## 15. 08000017 - 166MHz SDRAM operation violates tSSDAT specification:

**DESCRIPTION:**
While SDRAM operations up to a 133 MHz SDCLK rate work correctly, due to a timing issue, the SDRAM controller's tSSDAT timing requirement does not adequately correlate with the SDRAM's setup time characteristic (tAC in many SDRAM vendor's specifications) when used at a 166 MHz SDCLK rate. This issue only exists at the 166 MHz SDCLK rate. Using the SDRAM controller at a 166 MHz SDCLK rate could result in incorrect data accesses.

**WORKAROUND:**
Use the SDRAM controller at the 133 MHz rate.

**APPLIES TO REVISION(S):**
0.0, 0.1

## 16. 08000018 - TCK can effect external PM bus accesses under certain circumstances:

**DESCRIPTION:**
In some processors, it is possible to experience an issue with External Memory accesses using the PM data bus while TCK is being driven. In some cases the address value driven may change inappropriately. This issue effects PM bus accesses to external memory only, and is dependent on the TCK jtag input being driven. No issues have been observed while TCK is inactive. TCK is typically only driven by an In-Circuit-Emulator or a boundary-scan controller. During normal non-debug operation of the processor, TCK is not driven, and so this issue can easily be avoided.

**WORKAROUND:**
During normal processor operation, without an emulator or boundary-scan controller in use, the TCK input is not driven, and this issue will not appear. In scenarios where the processor application is being debugged with an emulator, or a boundary-scan controller is active (driving TCK), and an PM bus write to external space is observed to be incorrect, use the DM bus to execute the external write. Note that this is only necessary during debug, as this issue has not been observed while TCK is inactive.

**APPLIES TO REVISION(S):**
0.0, 0.1

## 17. 08000019 - PDAP must use DAI pins, external port pins not available for PDAP data:

**DESCRIPTION:**
The SHARC user has the choice of using the PDAP input by either routing the PDAP DATA signals over the External Port Data pins or using the SRU to route these signals to DAI pins. Due to a silicon issue, the PDAP MUST have it's data signals routed to DAI pins via the SRU, and CANNOT be used with it's data signals using the External Port Data pins. This issue does not effect other External Port or SDRAM usage. It is a restriction on PDAP usage only.

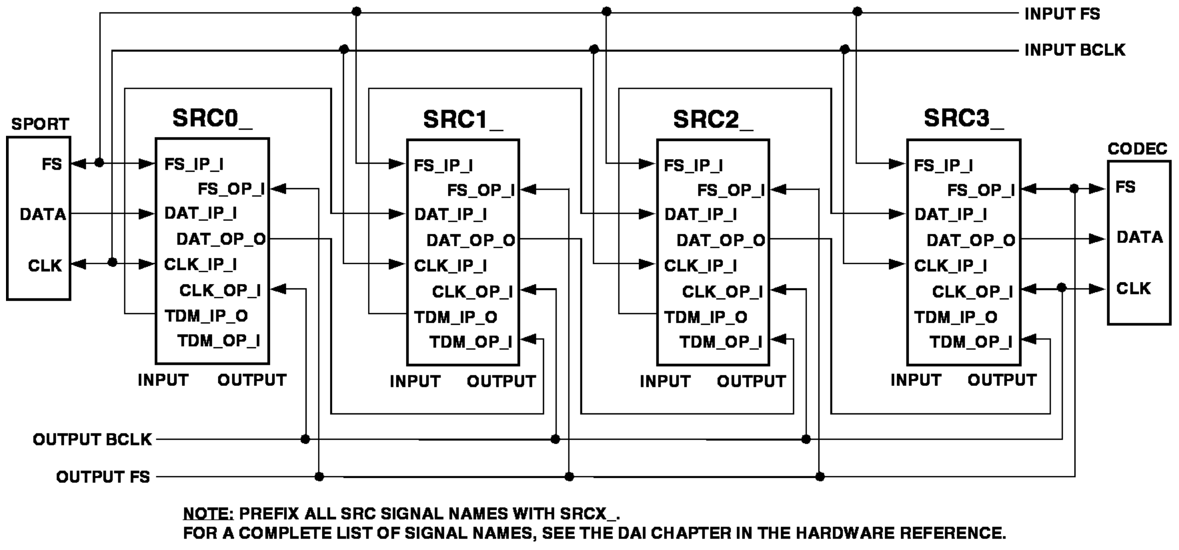
**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.0, 0.1, 0.2

## 18. 08000020 - Input Shift Register Anomaly in ASRCs (Asynchronous Sample Rate Converters) impacts Daisy-Chained TDM mode:

**DESCRIPTION:**

In the TDM Daisy chain mode the ASRCs can be connected as follows to achieve TDM mode:



Each ASRC has two 64-bit shift registers (one for the input side, and the other for the output side). When the ASRCs are daisy-chained, these 64-bit registers can be thought of as a single long shift register, which holds an entire frame of data.

In TDM chaining mode:

1. 64-bit shift register of each ASRC is divided into two 32-bit channels.
2. TDM output of the one ASRC input is connected to the data input of the next ASRC in the chain.
3. Data output of one ASRC is passed to the TDM input of the next ASRC in the chain.

The following diagram shows the channel allocation in the input and output side of the TDM Daisy chain.

| 32-BIT CHANNEL | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| BIT | | 0 | | | | | | | 255 |
| ASRC | IN SIDE | ASRC 3 LEFT | ASRC 3 RIGHT | ASRC 2 LEFT | ASRC 2 RIGHT | ASRC 1 LEFT | ASRC 1 RIGHT | ASRC 0 LEFT | ASRC 0 RIGHT |
| | OUT SIDE | ASRC 0 LEFT | ASRC 0 RIGHT | ASRC 1 LEFT | ASRC 1 RIGHT | ASRC 2 LEFT | ASRC 2 RIGHT | ASRC 3 LEFT | ASRC 3 RIGHT |

When the ASRCs are used in TDM Daisy chain mode, the MSB of the input data is lost at the output of each ASRC in the chain. This anomaly is applicable for the TDM mode in both the bypass and non-bypass modes. The following example provides the expected failure pattern for the bypass mode, since in bypass mode the incoming data is not modified by the ASRC.

For a TDM daisy chain with 8 channels in BYPASS mode, the output of the final ASRC in the chain will result in three-bit left shift for the channels 1 and 2, two bit left shift for the channels 3 and 4 and one bit left shift for the channels 5 and 6. The channels 7 and 8 will not have any data shift. At the output end, three MSBs will be lost for the channels 1 and 2, two MSBs will be lost for the channels 3 and 4 and one MSB will be lost for the channels 5 and 6.

For the set of the following 8-channel input data,

```
0xF1111100      0xF1111100      // Channels 1 and 2
0xF1111100      0xF1111100      // Channels 3 and 4
0xF1111100      0xF1111100      // Channels 5 and 6
0xF1111100      0xF1111100      // Channels 7 and 8
```

The output data will be as follows,

```
0x888888FF      0x888888FF      //3-bit left shift and 3 MSBs are lost
0xC44444FF      0xC44444FF      //2-bit left shift and 2 MSBs are lost
0xE22222FF      0xE22222FF      //1-bit left shift and one MSB is lost
0xF11111FF      0xF11111FF
```

In the above output data the 8-LSBs of each word has the ratio information.

For a TDM daisy chain in NON-BYPASS mode, the output of the TDM Daisy chain will be corrupted completely because of the MSB bit loss in each ASRC in the chain.

**WORKAROUND:**
The workaround for this anomaly must be carried out in software only on the device providing input data to the ASRCs in TDM daisy chain mode. It can not be implemented on the output data from the ASRCs. The workaround implementation in software adds some core MIPS to the application. For devices like ADCs/CODECs, the workaround may not be implemented directly. For such systems, two serial ports of the SHARC should be dedicated for this purpose. One serial port should be used for receiving the data from the ADC/CODEC and the other one should be used for transmitting the modified input data to the ASRC.

When the ASRCs are used in TDM daisy chain mode, only the 24-MSBs of the input 32-bit data is used and the 8-LSBs are not used by the ASRC. The anomaly can be worked around by shifting the 24-MSBs of the data to the 8-LSBs as follows:

1. Right shift the input data for channels 1 and 2 by three bits. Now the 24-MSBs which has the data are placed in the bits from 28 - 5.
2. Right shift the input data for channels 3 and 4 by two bits. Now the 24-MSBs which has the data are placed in the bits from 29 - 6.
3. Right shift the input data for channels 5 and 6 by one bit. Now the 24-MSBs which has the data are placed in the bits from 30 - 7.
4. The input data for channels 7 and 8 need not be modified.

When the modified input data is provided to the TDM daisy chain, one MSB of the modified data will be lost at each ASRC in the chain due to the anomaly. Since the 24-MSBs which has the data is shifted to the LSBs of the input, the actual MSB is not lost at each ASRC in the chain. Because of the anomaly the data will be shifted left by the ASRC and received correctly at the output end.

This workaround is applicable for both the BYPASS and NON-BYPASS mode of the ASRC. In both the cases the 24-bit audio data will be preserved and there will be no loss of resolution. The workaround also has no impact on the matched phase mode of the ASRC TDM daisy chain mode, since the phase information lies only in the 8-LSBs of the output SRC shift registers which are not affected by this anomaly.

For the set of the following 8-channel actual input data,

```
0xF1111100 0xF1111100 // Channels 1 and 2
0xF1111100 0xF1111100 // Channels 3 and 4
0xF1111100 0xF1111100 // Channels 5 and 6
0xF1111100 0xF1111100 // Channels 7 and 8
```

The data modified according to the workaround will be as follows:

```
0xFE222220 0xFE222220 // Channels 1 and 2 with three bit right shift
0xFC444440 0xFC444440 // Channels 3 and 4 with two bit right shift
0xF8888880 0xF8888880 // Channels 5 and 6 with one bit right shift
0xF1111100 0xF1111100 // Channels 7 and 8
```

The data at the output end will be as follows for the BYPASS mode:

```
0xF11111FF 0xF11111FF // Channels 1 and 2
0xF11111FF 0xF11111FF // Channels 3 and 4
0xF11111FF 0xF11111FF // Channels 5 and 6
0xF11111FF 0xF11111FF // Channels 7 and 8
```

**APPLIES TO REVISION(S):**
0.0, 0.1

## 19. 08000024 - Parallel EPROM/FLASH boot following a Hard (Hot) Reset may fail if issued in the middle of external SDRAM reads:

**DESCRIPTION:**
If the SHARC processor is in the process of fetching data 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 /RESET being asserted, and also failing to relinquish the external data bus even when the DSP comes out of /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 operation 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.

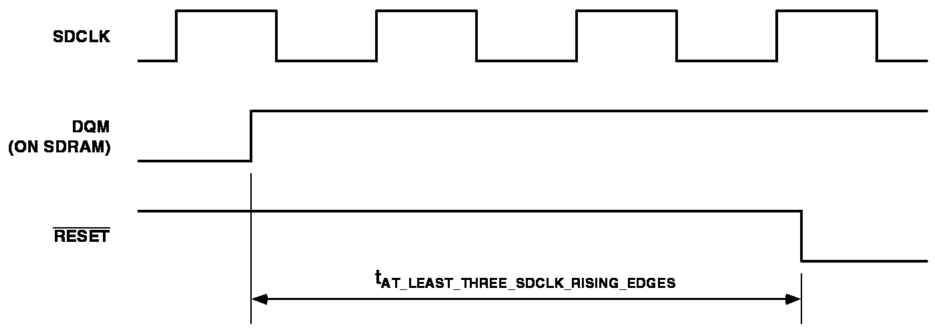
If a microcontroller or host processor controls the /RESET pin of the SHARC, a simple communication protocol between the microcontroller/host processor and the DSP that ensures that any DMA transfer/external data fetch is completed before /RESET is asserted will avoid this problem.

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. The anomaly is also not observed during other Reset scenarios such as Power-on Reset, or software 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 /RESET_IN to the DSP will also cause the problem to be avoided (see attached Figure).


SDCLK
DQM (ON SDRAM)
$\overline{\text{RESET}}$
$t_{\text{AT\_LEAST\_THREE\_SDCLK\_RISING\_EDGES}}$

**APPLIES TO REVISION(S):**
0.0, 0.1, 0.2

## 20. 08000025 - In a shared memory system, the processor core hangs, when the other processor requests for the bus during an AMI access:

### DESCRIPTION:
In a shared memory system, the processor core may hang when the following sequence occurs:

1. The processor is the current bus master which accesses the AMI interface either using core or DMA access.
2. Another processor requests for the bus at the same time.

At this time, the current shared bus master de-asserts the bus request ($\overline{BRx}$) before AMI finishes its transfer. This causes the AMI internal logic to go into a dead-lock state and never recovers from that state. This is applicable to both Core (only AMI) as well as EPDMA0/1 (only AMI) accesses. Both AMI reads and AMI writes will be affected by this. The processor which has requested the bus will not have any issues. The processor making the request gets the bus and accesses the external bus in the following cycle. This problem is not applicable to SDRAM accesses.

When the problem occurs, the bus mastership is lost before the AMI access is completed. The core hangs, waiting for the AMI transfer to complete and enters a dead lock condition. The processor can not be recovered from the dead lock condition with hardware interrupts.

### WORKAROUND:
1. Bus lock:
Lock the external bus before accessing the AMI interface. This will make sure that the core does not hang. This can be achieved by using the following sequence:

```
ustat1 = dm(SYSCTL);
bit set ustat1 BUSLK;//Set bit to lock the bus
dm(SYSCTL) = ustat1;
nop;nop;nop;nop;nop;nop;nop;//Effect latency of the BUSLK bit*
IF NOT BM JUMP(PC,0);// Wait to get the bus mastership
R0 = dm(0x8000000);// Do the AMI access
ustat1 = dm(SYSCTL);// Unlock the bus after access
bit clr ustat1 BUSLK;
dm(SYSCTL) = ustat1;
nop;nop;nop;nop;nop;nop;nop; //Effect latency of the BUSLK bit*
```

This workaround will affect the performance of the application when the processor is not the current bus master and one of the other processor is accessing the external bus. In this case the core has to stall until the other processor finishes its external port access.

2. Modified Bus lock:
For this case, the bus needs to be locked only when the processor is the current bus master. If the processor is not the bus master then this problem will not be seen. This can be achieved by using the following sequence:

```
IF NOT BM JUMP Label; // Check for bus mastership
ustat1 = dm(SYSCTL); // If the processor is the master lock the bus
bit set ustat1 BUSLK ; //Set bit to lock the bus
dm(SYSCTL) = ustat1;
nop;nop;nop;nop;nop;nop;nop; //Effect latency of the BUSLK bit*
Label:
R0 = dm(0x8000000);// Do the AMI access ( For the slave case anyway the core
                   // will automatically stall to get the bus).
ustat1 = dm(SYSCTL); // Unlock the bus after access
bit clr ustat1 BUSLK ;
dm(SYSCTL) = ustat1;
nop;nop;nop;nop;nop;nop;nop; //Effect latency of the BUSLK bit*
```

This workaround will avoid the time taken in waiting to gain the bus mastership with the first workaround.

*Note: BUSLK bit has an effect latnecy of 1 CCLK + 1 PCLK + 1 SDCLK clock cycles. Seven "nops" have been added for the worst case i.e. maximum CCLK:SDCLK ratio of 4:1. The number of "nops" can be modified for a different CCLK:SDCLK ratio e.g. only five "nops" are required a CCLK:SDCLK ratio of 2:1.

### APPLIES TO REVISION(S):
0.0, 0.1, 0.2

## 21. 08000026 - 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.

**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.

**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):**
0.0, 0.1, 0.2

## 22. 08000028 - Conditional FLAG instructions involving DAG index registers must not be followed immediately by an instruction that uses the same index register:

**DESCRIPTION:**
In the following instruction sequence shown below:

```
INSTR1: Compute;
INSTR2: If COND DM (Ia,Mb); //COND maybe based on the INSTR1, or any other condition
INSTR3: DM (Ia,Mc);
```

The value of the DAG index register in **`INSTR3`** will either be **`Ia`** or **`(Ia+Mb)`** depending on whether **`INSTR2`** was aborted or executed.

If **`COND`** is based on 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 may not contain the correct and expected value.

**WORKAROUND:**
Separate the second and third instructions in the above sequence by a NOP.
**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):**
0.0, 0.1, 0.2

## 23. 08000029 - 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, 0.1, 0.2

## 24. 08000030 - VDSP Statistical profiler may provide incorrect profile information for idle instructions:

**DESCRIPTION:**
The VisualDSP++ statistical profiler may indicate incorrect processor time expenditure when idle instructions are used. This occurs if the EMUPC register does not get updated when the program sequencer is executing the idle instruction.

In the example code below, the ISR_func1 is executed at the ISR's interrupt rate and most of the processor time is spent in executing the idle instruction. However, the statistical profiler may show incorrectly that most of the processor time is spent executing instructions in the ISR_func1. This occurs when the program sequencer branches to execute the IDLE instruction from the RTI instruction. When this occurs, the EMUPC register may still retain the previously executed ISR_func1's Program Counter value.

```
_main:
……
Test:
Idle;
Jump Test;
_main.end:

ISR_func1:
……
……
Instruction_x;
Instruction_y;
RTI;
```

Note: This issue impacts only the Statistical Profiling information.

**WORKAROUND:**
NOP instruction can be used instead of IDLE instruction wherever possible to get correct statistical profile information.

**APPLIES TO REVISION(S):**
0.0, 0.1, 0.2

## 25. 08000031 - After an emulator halt at the instruction before 'idle' instruction, the Core Timer stops decrementing even after code execution restarts:

**DESCRIPTION:**
When the processor is halted at a breakpoint in an emulator session, the core timer correctly stops decrementing and restarts when code execution is resumed. However, if the emulator breakpoint is placed at an instruction just before an idle instruction, the core timer remains halted even after the code execution is resumed.

In the example code below, the core timer remains halted after code execution is resumed following the core halt at Instruction1. The same behavior is seen if Instruction1 is executed by 'single stepping'.

```
Enable_Core_timer:
……
Bit set MODE1 TIMEN;
……
……
Instruction1;                // Placing breakpoint here causes the anomaly
Idle;
Instruction2;
Instruction3;
```

Note: This issue impacts only Emulator session debug.

**WORKAROUND:**
None

**APPLIES TO REVISION(S):**
0.0, 0.1, 0.2

This page intentionally left blank.