ANALOG DEVICES

SHARC®
Embedded Processor

Silicon Anomaly List

ADSP-21160M

# ABOUT ADSP-21160M SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC ADSP-21160M product(s) and the functionality specified in the ADSP-21160M 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 bits 31-25 of the **MODE2_SHDW** register either via JTAG or DSP code.

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

```
<UREG> = MODE2_SHDW;
```

| Silicon REVISION | MODE2_SHDW[31:25] |
|---|---|
| 1.2 | 0111001 |
| 1.1 | 0110001 |
| 0.1 | 0101001 |
| 0.0 | 0100001 |

## 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 |
|---|---|---|---|
| 09/16/2009 | K | 0 | Added anomalies: 02000069,Added common note on Tools action for the anomalies 02000014 and 02000065 |
| 05/10/2007 | J | 0 | Document Format Update. |
| 04/19/2006 | I | 0 | Modified anomalies: 02000066 |
| 02/23/2006 | H | 0 | Added anomalies: 02000066 |
| 08/12/2005 | G | 0 | Added anomalies: 02000065,Modified anomalies: 02000060 |
| 11/25/2004 | F | 0 | Added anomalies: 02000063,02000064 and updated document references |
| 12/2/2003 | E | 0 | Added anomalies: 02000060,02000061 and 02000062,Modified anomalies: 02000046 |

NR002534K



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-21160M anomalies and the applicable silicon revision(s) for each anomaly.

| No. | ID | Description | 0.0 | 0.1 | 1.1 | 1.2 |
|---|---|---|---|---|---|---|
| 1 | 02000001 | Non-complimentary destination conditional UREG-to-UREG transfers execute incorrectly | x | . | . | . |
| 2 | 02000002 | Conditional Long-Word and Extended Precision writes not flushed properly from the shadow write FIFO | x | . | . | . |
| 3 | 02000003 | SIMD mode explicit accesses of odd addresses in internal memory do not function correctly | x | . | . | . |
| 4 | 02000004 | $\overline{PA}$ incorrectly truncates non-priority burst transfers | x | . | . | . |
| 5 | 02000005 | ACK is not driven properly by synchronous slave | x | . | . | . |
| 6 | 02000006 | The JTAG Tap controller does not asynchronously reset | x | . | . | . |
| 7 | 02000007 | Host Direct Reads and Direct Writes may fail due to early assertion of $\overline{HBG}$ | x | . | . | . |
| 8 | 02000008 | Buffer Hang Disable (BHD) is set (=1) by default at reset | x | . | . | . |
| 9 | 02000009 | Core driven link port transfers may fail when overrunning the link buffer | x | x | . | . |
| 10 | 02000010 | Host accesses following an external port DMA transfer fail | x | . | . | . |
| 11 | 02000011 | MMS transfers may not function correctly | x | . | . | . |
| 12 | 02000012 | SPORT Multichannel Transmit failure | x | x | . | . |
| 13 | 02000013 | IMASKP register not updated correctly | x | x | . | . |
| 14 | 02000014 | RFRAME instruction does not function correctly | x | x | . | . |
| 15 | 02000015 | Slave Write FIFO may corrupt data | x | . | . | . |
| 16 | 02000016 | External Port arbitration failure during DMA burst transfer with core priority | x | . | . | . |
| 17 | 02000017 | Erroneous Host Bus Grant Assertion | . | x | . | . |
| 18 | 02000018 | SPORT Internally generated frame sync operates incorrectly under specific conditions | x | x | . | . |
| 19 | 02000019 | MODE2 bit 28 (Silicon Revision) Incorrect | . | x | . | . |
| 20 | 02000020 | Setting MAXBL[1:0] to 01 corrupts MMS write transmitted data | x | x | . | . |
| 21 | 02000021 | Idle cycle between even/even or odd/odd MMS banks not inserted | x | x | . | . |
| 22 | 02000022 | The PLL does not reliably reset on power-up | x | x | x | x |
| 23 | 02000023 | Power supply sequence (Core and I/O power supplies) can damage ESD diodes | x | x | x | x |
| 24 | 02000024 | Link port change in direction of transfer corrupts first data word | x | x | . | . |
| 25 | 02000025 | Simultaneous DMA/CORE access of EPBx | x | x | . | . |
| 26 | 02000026 | Host asynchronous writes may fail | x | x | x | x |
| 27 | 02000027 | Host asynchronous access overlapping DSP to DSP transfer fail | x | x | x | x |
| 28 | 02000028 | SPORT transfers fail at frequencies above 35 MHz | x | x | x | x |
| 29 | 02000029 | SPORT companding fails for first active transmit channel in multichannel frame | x | x | x | x |
| 30 | 02000030 | SIMD mode is adversely affecting data accesses to odd external memory addresses | . | x | x | x |
| 31 | 02000031 | Broadcast mode works improperly when source buffer is located in external memory | . | x | x | x |
| 32 | 02000032 | 64-bit data accesses in internal memory using the LW mnemonic do not function correctly with odd explicit address | . | x | x | x |
| 33 | 02000033 | Link port to link port transfers do not function reliably at all frequencies. At certain frequencies, the link port transmitter does not meet link port receiver data setup and hold timing requirements | x | x | x | x |
| 34 | 02000034 | Link port data corruption may occur when a particular alignment of LCLK and CK occurs | . | x | x | x |
| 35 | 02000035 | DMA handshake modes limited in throughput | . | x | x | x |
| 36 | 02000036 | MMS reads following MMS broadcast writes do not function properly | . | x | x | x |
| 37 | 02000037 | BMAX/BCNT counter expire corrupts synchronous burst transfers in progress during a BTC (bus transition cycle) | . | x | x | x |
| 38 | 02000038 | $\overline{SBTS}$ deassertion can cause a resumed external memory access to abort | . | x | . | . |
| 39 | 02000039 | Shadow Write FIFO Anomaly | x | x | x | . |

| No. | ID | Description | 0.0 | 0.1 | 1.1 | 1.2 |
|---|---|---|---|---|---|---|
| 40 | 02000040 | SIMD read from internal memory with Shadow Write FIFO hit does not function correctly | . | x | x | x |
| 41 | 02000041 | Under special conditions, glitching may occur on $\overline{HBG}$ in a system with multiple DSPs and a Host | . | . | x | . |
| 42 | 02000042 | The first Host asynchronous read after HTC may fail | . | x | x | x |
| 43 | 02000043 | Link port buffers and buffer status bits may not behave properly upon RESET affecting initial link port data transmission | . | x | x | x |
| 44 | 02000044 | IMASKP bits are left shifted by 1 bit for writes to bits 14 to 30 | . | . | x | x |
| 45 | 02000045 | Inactive EPBx DMA channel parameter registers (EIx, EMx, ECx) may be read incorrectly | . | x | x | x |
| 46 | 02000046 | Direct writes to IMASKP or LIRPTL may cause highest priority interrupt to be serviced twice | . | x | x | x |
| 47 | 02000047 | Asynchronous Host Reads can fail if tSADRDL is less than one external clock cycle | x | x | x | x |
| 48 | 02000048 | 32-bit wide link port transfers limited in throughput with 1:1 LCLK-to-CCLK ratio | x | x | x | x |
| 49 | 02000050 | Conditional type 10 instruction may fail in SIMD | x | x | x | x |
| 50 | 02000051 | Broadcast load fails in type 10 instruction | x | x | x | x |
| 51 | 02000052 | Rn=MANT Fx results will be rounded if RND32 is enabled | x | x | x | x |
| 52 | 02000053 | In Serial Port Multichannel mode, an external RFS one cycle early causes data corruption | . | . | x | x |
| 53 | 02000054 | Illegal DAG stalls can occur under certain circumstances | x | x | x | x |
| 54 | 02000055 | Execution of instructions that modify interrupt latch registers may cause incoming interrupts to be ignored | x | x | x | x |
| 55 | 02000056 | A breakpoint following a JUMP, CALL, RTI or RTS can trigger an early/improper emulator break | x | x | x | x |
| 56 | 02000057 | DAG register write followed directly by a DAG register read fails during context switch | x | x | x | x |
| 57 | 02000058 | System registers written with a PM access do not have a 1 cycle effect latency | x | x | x | x |
| 58 | 02000059 | Conditional RTI fails in SIMD mode | x | x | x | x |
| 59 | 02000060 | Single instruction loops can terminate early | x | x | x | x |
| 60 | 02000061 | Bit reversal fails with indirect jump | x | x | x | x |
| 61 | 02000062 | VIPD bit gets cleared when branching to ISR | x | x | x | x |
| 62 | 02000063 | MU (Multiplier underflow) flag gets set anomalously for some non-underflow cases | x | x | x | x |
| 63 | 02000064 | Top of the loop address stack is filled with zero when PUSH LOOP instruction is executed | x | x | x | x |
| 64 | 02000065 | DAG stall causes external port to malfunction | x | x | x | x |
| 65 | 02000066 | Read of slave DSP's EPBx by host or master DSP in a multiprocessing may return wrong data | x | x | x | x |
| 66 | 02000069 | Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers | x | 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-21160M including a description, workaround, and identification of applicable silicon revisions.

## 1. 02000001 - Non-complimentary destination conditional UREG-to-UREG transfers execute incorrectly:

**DESCRIPTION:**
In SIMD mode (PEYEN set in MODE1), a conditional ureg-to-ureg transfer with a non-complimentary ureg (non-SIMD pair, e.g. I0, MODE1) as the destination will not execute properly if the PEx condition fails and the PEy conditional passes.

**WORKAROUND:**
Avoid using the conditional ureg-to-ureg transfer instruction in code sections where SIMD will be enabled. Alternately, force the condition of PEy to match that of PEx before execution of this instruction in SIMD mode.

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

## 2. 02000002 - Conditional Long-Word and Extended Precision writes not flushed properly from the shadow write FIFO:

**DESCRIPTION:**
A conditional, long-word write to internal memory (either long-word address space, or long-word instruction option), which gets aborted (condition tests false), may not get flushed from the shadow write buffer correctly. The aborted write data will not be written to memory but if the conditional write is immediately followed by a read whose address corresponds to the same long-word address, then the aborted write data will be sourced, rather than the correct data from memory. The aborted write must be to a long-word address; however, the read can be to either a long-word address, a long-word instruction option, or a normal/short-word address which corresponds to the same physical space as the aborted data.

Also, extended-precision (that is, IMDWx bit set for the addressed memory block) reads may incorrectly hit against aborted extended-precision writes in the shadow write buffer, in the same manner as described above.

**WORKAROUND:**
If the programmer needs to use conditional, long-word writes, followed by reads that can be to the same address (including a short or normal-word read from either normal-word address of the long-word address), then two non-read operations must be inserted between the write and the read. A non-read operation can be a NOP, or any other instruction that does not read from the memory block addressed by the store.

Likewise, if a memory block is configured for extended-precision (IMDWx bit in SYSCON set for that block), then the same Work around must be applied for potentially aborted extended-precision writes.

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

## 3. 02000003 - SIMD mode explicit accesses of odd addresses in internal memory do not function correctly:

**DESCRIPTION:**
Normal-Word address accesses:
SIMD mode explicit accesses of odd Normal-Word addresses in internal memory do not function correctly. The implicit part of this SIMD mode transfer incorrectly accesses the previous sequential even address. For example, a SIMD mode explicit access to Normal-Word address 0x40001 will result in an implicit access to Normal-Word address 0x40000.
The following table illustrates:
**Incorrect:**

| | Explicit "R0" R0=dm(I0,M0); | | Explicit "S0" S0=dm(I0,M0); | |
|---|---|---|---|---|
| Explicit Address (I0) | R0 | S0 | R0 | S0 |
| 0x40001 | 32-bit word at 0x40001 | 32-bit word at 0x40000 | 32-bit word at 0x40000 | 32-bit word at 0x40001 |

This access type should result in an implicit access to the next sequential even address value. For example, a SIMD mode explicit access to Normal-Word address 0x40001 should result in an implicit access to Normal-Word address 0x40002.

The following table illustrates:
**Correct:**

| | Explicit "R0" R0=dm(I0,M0); | | Explicit "S0" S0=dm(I0,M0); | |
|---|---|---|---|---|
| Explicit Address (I0) | R0 | S0 | R0 | S0 |
| 0x40001 | 32-bit word at 0x40001 | 32-bit word at 0x40002 | 32-bit word at 0x40002 | 32-bit word at 0x40001 |

Short-Word:
SIMD mode explicit accesses of Short-Word addresses with ADDR1=1 in internal memory do not function correctly. The implicit part of this SIMD mode transfer incorrectly accesses a previous sequential Short-Word address. For example, a SIMD mode explicit access to Short-Word address 0x80002 will result in an implicit access to Short-Word address 0x80000.

The following table illustrates:
**Incorrect:**

| | Explicit "R0" R0=dm(I0,M0); | | Explicit "S0" S0=dm(I0,M0); | |
|---|---|---|---|---|
| Explicit Address (I0) | R0 | S0 | R0 | S0 |
| 0x80002 | 16-bit word at 0x80002 | 16-bit word at 0x80000 | 16-bit word at 0x80000 | 16-bit word at 0x80002 |
| 0x80003 | 16-bit word at 0x80003 | 16-bit word at 0x80001 | 16-bit word at 0x80001 | 16-bit word at 0x80003 |

This access type should result in an implicit access to the value at the Short-Word address equal to the explicit address + 2. For example, a SIMD mode explicit access to Short-Word address 0x80002 should result in an implicit access to Short-Word address 0x80004.

The following table illustrates:
**Correct:**

| | Explicit "R0" R0=dm(I0,M0); | | Explicit "S0" S0=dm(I0,M0); | |
|---|---|---|---|---|
| Explicit Address (I0) | R0 | S0 | R0 | S0 |
| 0x80002 | 16-bit word at 0x80002 | 16-bit word at 0x80004 | 16-bit word at 0x80004 | 16-bit word at 0x80002 |
| 0x80003 | 16-bit word at 0x80003 | 16-bit word at 0x80005 | 16-bit word at 0x80005 | 16-bit word at 0x80003 |

**WORKAROUND:**
None

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

## 4. 02000004 - $\overline{PA}$ incorrectly truncates non-priority burst transfers:

**DESCRIPTION:**
Asserting the $\overline{PA}$ signal to a ADSP-21160 that is performing a non-priority (configured by PRIO in DMACx) burst transfer on the external bus will truncate the burst transfer operation and force re-arbitration for the bus. The correct functionality is for the ADSP-21160 to complete the current (4-word) burst before allowing re-arbitration.

**WORKAROUND:**
$\overline{PA}$ and non-priority bursts should not be used together. Do not connect $\overline{PA}$ in systems where non-priority bursts will be enabled. Do not enable non-priority bursts in systems where $\overline{PA}$ is connected.

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

## 5. 02000005 - ACK is not driven properly by synchronous slave:

**DESCRIPTION:**
When the ADSP-21160 is accessed as a slave, with a synchronous bus operation (that is, either MP space, or a host access that does not use $\overline{CS}$), then the ADSP-21160 slave incorrectly only drives ACK if it must deassert ACK, and then assert ACK. For example, if several synchronous writes access a ADSP-21160's EPB0 buffer through MP space, that ADSP-21160 may never drive ACK asserted, since it may never have had to deassert ACK. Also, if there is no synchronous access on the bus, ACK will not be driven actively. In both these cases, ACK is pulled-up by the current supplied by weak pull-up device of the keeper latch, enabled for device ID= 000 and 001. A low glitch of sufficient magnitude and duration on ACK may flip its state at such times and hang the bus, since a subsequent bus operation will not start until ACK is sampled high.

The correct functionality is to have a pull-up device enabled for ADSP-21160 ID=000 or 001. Also the ADSP-21160 should actively drive ACK from the cycle after the bus transfer is started on the bus, until the cycle after the ADSP-21160 slave asserts ACK to the master, concluding the bus transfer (except for broadcast writes. See "ADSP-21160 SHARC DSP Hardware Reference" for more information on broadcast writes).

**WORKAROUND:**
Add an external pull-up resistor on ACK, to provide additional pull-up current source. The value of the resistor must be >= 2k ohm, recommend 5-10k ohm value.

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

## 6. 02000006 - The JTAG Tap controller does not asynchronously reset:

**DESCRIPTION:**
TCK must be low ("0") for $\overline{TRST}$ assertion to place the Tap controller in reset.

**WORKAROUND:**
TMS must be high ("1"), TCK must be low ("0"), and $\overline{TRST}$ must be asserted low ("0") in order to reset the tap controller.

**Note:** TMS has an internal pull-up.

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

## 7. 02000007 - Host Direct Reads and Direct Writes may fail due to early assertion of $\overline{HBG}$:

**DESCRIPTION:**
The ADSP-21160 asserts $\overline{HBG}$ in response to $\overline{HBR}$ too early to accept an immediate turnaround of $\overline{RDx}$ and $\overline{WRx}$. This may cause host direct reads and direct writes to fail.

**WORKAROUND:**
Assertion of $\overline{RDx}$ and $\overline{WRx}$ should be delayed a minimum of one CLKIN period (tck) from $\overline{HBG}$ sampled asserted.

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

## 8. 02000008 - Buffer Hang Disable (BHD) is set (=1) by default at reset:

**DESCRIPTION:**
The BHD bit in the SYSCON register is set by default at reset in revision 0.0 but is cleared by default at reset in revision 0.1 and later. Setting the BHD bit will cause the processor core to ignore a hold-off caused by reading an empty EPBx, RXx, or LBUFx buffer or by writing to a full EPBx, RXx, or LBUFx buffer. Core driven transfers that depend on the hold-off will not function reliably.

**WORKAROUND:**
Clear BHD (=0) in SYSCON before performing any core driven transfers with the EPBx, RXx, or LBUFx buffers that rely on the hold-off. The following code can be used to perform this function with no negative implications for other silicon revisions.

```
ustat1 = dm(SYSCON);  /* Clear Buffer Hang Disable */
bit clr ustat1 BHD;   /* for rev 0.0 */
dm(SYSCON) = ustat1;
```

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

## 9. 02000009 - Core driven link port transfers may fail when overrunning the link buffer:

**DESCRIPTION:**
Core driven link port transfers may not function correctly if the code relies on core hang when reading an empty link buffer or when writing to a full link buffer. The use of a polling routine to ensure that the core hang is not encountered may resolve the problem but is not guaranteed.

**WORKAROUND:**
Use DMA driven link port transfers.

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

## 10. 02000010 - Host accesses following an external port DMA transfer fail:

**DESCRIPTION:**
Host accesses of ADSP-21160 internal memory may not function correctly if they occur at any time after an ADSP-21160 external port DMA transfer has taken place. Reads of ADSP-21160 internal resources may return invalid data. Writes to ADSP-21160 internal resources may not complete correctly. The failure is due to a race condition on an internal DMA address bus that can result in an incorrect address latch for the slave transfer. Contention between the previous value on this bus and the new address can cause several lower address bits to not transition from a 1 to a 0 as they should.

**WORKAROUND:**
Whenever a master (host or another sharc) wants to do a direct read/write to a slave's internal memory. It should perform the following sequence:

1. Save EI10 of the slave.
2. Write 0x00000000 to EI10.
3. Read back EI10.
4. Perform all the direct reads/writes of internal memory.
5. Restore EI10 of the slave.

This will ensure that the internal DMA address bus gets initialized to 0x00000000 before any direct reads/writes are performed. Saving and restoring EI10 (or any other EI register) will ensure that the when the slave gets back to master mode, if any DMA on channel 10 were pending, they would continue from where they had left off.

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

## 11. 02000011 - MMS transfers may not function correctly:

**DESCRIPTION:**
MMS transfers may not function correctly or reliably under all conditions.

**WORKAROUND:**
None

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

## 12. 02000012 - SPORT Multichannel Transmit failure:

**DESCRIPTION:**
In multichannel mode, if the frame sync period is set to be exactly equal to the frame data width, and if the first channel is selected for transmit, the first bit after the frame sync is not transmitted (i.e. tristated). There is an SLCK edge for this bit and all subsequent bits are transmitted correctly but the first bit after the frame is not driven.

**Example:**
Number of channels = 5
Number of bits per word = 16
Frame sync divisor = 80 = 5 * 16

**WORKAROUND:**
If the number of channels in the sequence is less than 32 (i.e. NCH < 31) and if the frame sync period is exactly equal to the frame width (i.e. FSDIV=((SLEN+1)*(NCH+1)-1)), then the MTCS register should be programmed as follows:

if NCH = x; (NCH is the number of channels - 1)
if MTCS[0] = 1;
then MTCS[x+1] = 1;

For the above example, MTCS[0] = 1 and MTCS[5] = 1

This will select to transmit one channel beyond the number of channels configured. It will have no side effects but will allow the first channel to be transmitted correctly.

**Note:** The failure does not occur when the number of channels is equal to 32 (NCH = 31).

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

## 13. 02000013 - IMASKP register not updated correctly:

**DESCRIPTION:**
The IRPTL[31:14] interrupts, when being serviced, will set the IMASKP bit that corresponds to the interrupt one priority level higher (or one bit position lower in the IMASKP register). Although the bit positions do not correlate interrupt nesting does function correctly.

**WORKAROUND:**
For interpreting the settings in IMASKP correctly, bit shifting the value in IMASKP[31:14] one position toward the msb will result in the appropriate value.

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

## 14. 02000014 - RFRAME instruction does not function correctly:

**DESCRIPTION:**
The RFRAME instruction is generated by the compiler to facilitate restoring of the stack and frame pointers when returning from a subroutine (function, isr, etc.). It is a special opcode which has the same functionality as "I7=I6, I6=dm(0,I6);". For the data access (dm(0,I6)) portion of the opcode only, the I6 register is not sourced properly. Instead of using the address contained in I6 for the load, the DSP uses the last address driven on the local DAG1 bus. Therefore it may appear to function correctly if the previous value on the DAG1 local bus coincidentally happens to be the same value in I6. This can happen if I6 is the last DAG1 register to be written, read, or used.

**WORKAROUND:**
Do not use the RFRAME instruction. This instruction should never be used in assembly programming in general. The ADSP-21160 C Compiler can be forced not to generate the RFRAME instruction.

**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

## 15. 02000015 - Slave Write FIFO may corrupt data:

**DESCRIPTION:**
The slave write FIFO is used to buffer data from direct writes or IOP writes at the external port before delivering the data to the actual destination. Data from these writes can back up into the slave write FIFO such that it is stored in the slave write FIFO for multiple cycles. For example, this can occur when writing to an external port buffer (EPBx) before a DMA has been configured. In the case of this event, the data stored in the slave write FIFO can be corrupted.

**WORKAROUND:**
Do not allow direct writes or IOP writes to back up into the slave write FIFO. Make sure that slave mode DMAs are configured before IOP writes occur. Close attention should also be paid to the DMA channel priorities as higher priority DMA activity can hold off external port DMAs.

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

## 16. 02000016 - External Port arbitration failure during DMA burst transfer with core priority:

**DESCRIPTION:**
When External Bus Priority is setup such that the core has higher priority (EBPR=01) and there is a DMA burst operation in progress, arbitration for the External Port bus will fail if the core attempts to perform an external access. The DMA burst believes that the core has priority but the core does not believe that it can break up the burst. This results in a deadlock which will never resolve. Note: burst transfer must be in progress at the time of the core access for this deadlock to occur.

The operation that locks up the EP bus is triggered under the following conditions.

EBPR = 01 (External Bus Priority: Core has higher-priority).
DMA is doing a burst operation.
CORE (through DAG) attempts to perform an external access.

**WORKAROUND:**
Program SYSCON to select even priority, alternating processor core and IOP accesses of the EP bus (EPBR = 00). With this Work around, deadlock will not occur. Selecting even priority will have an impact on core performance and should be taken into consideration.

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

## 17. 02000017 - Erroneous Host Bus Grant Assertion:

### DESCRIPTION:

Erroneous Host Bus Grant ($\overline{\text{HBG}}$) assertion can occur without Host Bus Request ($\overline{\text{HBR}}$) after a specific external port condition. If the ADSP-21160 bus master samples a deasserted ACK during an inactive bus cycle (no strobes active), the bus master will erroneously assert $\overline{\text{HBG}}$. The only situation when the ACK can be deasserted without an access is when the master is performing writes to a synchronous FIFO. A synchronous FIFO, such as the slave-write FIFO of an ADSP-21160 slave or a memory mapped synchronous FIFO, will deassert ACK if it becomes full during a valid write access in the previous cycle in order to prevent further writes. ACK will remain deasserted until the FIFO is no longer full, i.e. at least one word read out of the FIFO. **NOTE:** If ACK stays low for just one cycle, $\overline{\text{HBG}}$ will not assert.

There are three consequences of the erroneous $\overline{\text{HBG}}$ assertion:
1. If the $\overline{\text{HBG}}$ assertion while $\overline{\text{HBR}}$ is deasserted triggers some action on the host side which could have negative side effects for the host and/or DSP system, the host interface must employ a work around.
2. When $\overline{\text{HBG}}$ asserts erroneously, the DSP bus-master three-states all the strobes and bus pins. During this period where all the DSP's behave as a slave to a phantom host, all the EP signals on the board may be floating if the host interface hardware does not drive these pins to inactive states.
3. It will take the DSP bus master one extra cycle to recover from deassertion of ACK and resume its activity on the external bus. This has a minor impact on the throughput of the external bus.

It has been verified that should $\overline{\text{HBR}}$ assert before, during or after $\overline{\text{HBG}}$ has asserted, the accesses initiated by host still complete correctly.

### WORKAROUND:

**Software Work-around:**
This work around is based upon the fact that erroneous $\overline{\text{HBG}}$ assertion occurs only when ACK is deasserted during an inactive bus cycle if the external bus is not locked by the master DSP. BUSLK (defined in MODE2 register) should be set prior to making WRITE access(es). BUSLK will allow the bus master to lock the bus, thus preventing assertion of $\overline{\text{HBG}}$. To avoid ACK low on inactive bus cycles after BUSLK has been removed, a READ access should be launched following a WRITE to a synchronous FIFO. Now, even if ACK goes low, the DSP is not affected as it waits for ACK to assert, thus completing the READ access. For a series of back to back WRITE accesses, only the last access will have to be followed by a READ access and only if the last WRITE access was to a synchronous FIFO.

EXAMPLES:
To implement Software Work around for the CORE:
1. Set BUSLK.
2. Perform WRITE(s).
3. Perform READ.
4. Clear BUSLK.

To implement Software Work around for DMA:
1. Set BUSLK.
2. Initiate DMA WRITE(s) with interrupt enabled.
3. In ISR, perform READ and then clear BUSLK.

**Hardware Work-around:**
The assertion of $\overline{\text{HBG}}$ without accompanying $\overline{\text{HBR}}$ puts all the DSP's in slave mode - all the devices consistently recognize that the bus has transitioned to host. Consequently, all the ADSP-21160 devices three-state their ADDR/DATA bus and the EP strobes. The host interface hardware on the board should be built to take care of the situation.

EXAMPLES:
1. Qualify $\overline{\text{HBG}}$ with $\overline{\text{HBR}}$.
2. Implement a state machine which does not transition state(s) based upon $\overline{\text{HBG}}$ alone.

### APPLIES TO REVISION(S):

0.1

## 18. 02000018 - SPORT Internally generated frame sync operates incorrectly under specific conditions:

**DESCRIPTION:**
When the SPORT frame sync period setting is exactly equal to the data word width (i.e. frame sync period = slen + 1), the SPORT transmit circuitry ignores the state of the DITFS (Data Independent Transmit Frame Sync) bit in the STCTLx register. The SPORT operates as if the DITFS bit were set, regardless of the actual setting. If these conditions are true and DITFS = 0, the SPORT will operate incorrectly by generating the transmit frame sync on the appropriate periodic interval with or without new data to transmit. The behavior is exactly as if DITFS is set.

Under normal operation, when DITFS = 0 (default after reset), the transmit frame sync signal (TFS) is dependent upon new data being present in the TX buffer and the TFS signal is only generated when new data is present in the buffer. Setting DITFS = 1 selects data independent transmit frame syncs. This causes the TFS signal to be generated whether or not new data is present, transmitting the contents of the TX buffer (previous data word) regardless.

**WORKAROUND:**
Increase frame sync period so that the condition (frame sync period = slen + 1) will be false. If you deviate in any way to make the condition (frame sync period = slen + 1) false, the SPORT will work correctly. It is important to ensure that you do not violate the rule that frame sync width must be greater than data word length when adjusting your settings.

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

## 19. 02000019 - MODE2 bit 28 (Silicon Revision) Incorrect:

**DESCRIPTION:**
The Silicon Revision field of the MODE2 register is used for differentiating between silicon revisions.
Bit 28 of this field is incorrectly cleared (=0) such that the silicon revision field reads identically for both revision 0.0 and 0.1.

```
INTENDED BIT PATTERN
 MODE2[31:25]
 31 30 29 28 27 26 25
  0  1  0  0  0  0  1  Rev 0.0
  0  1  0  1  0  0  1  Rev 0.1

ACTUAL BIT PATTERN
 MODE2[31:25]
 31 30 29 28 27 26 25
  0  1  0  0  0  0  1  Rev 0.0
  0  1  0  0  0  0  1  Rev 0.1
```

**Note:** MODE2[31:25] are read only bits of the MODE2 register.

**WORKAROUND:**
Read MODE2_SHDW instead of MODE2 for silicon revision number.
MODE2_SHDW is a memory mapped register whose address is 0x11.

In silicon revisions prior to revision 1.2, the upper 7 bits of the MODE2_SHDW register were documented to mirror corresponding bits within the MODE2 register. In silicon revision 1.2 (and any later revisions which may follow), MODE2_SHDW has been modified to no longer mirror the upper 7 bits of MODE2 and will contain information unique to the silicon revision.

```
MODE2_SHDW[31:25] shadow register
31 30 29 28 27 26 25  (bits)
 0  1  0  0  0  0  1  Rev 0.0 (mirrors MODE2)
 0  1  0  1  0  0  1  Rev 0.1 (unique to MODE2_SHDW)
 0  1  1  0  0  0  1  Rev 1.0/Rev 1.1 (mirrors MODE2)
 0  1  1  1  0  0  1  Rev 1.2 (unique to MODE2_SHDW)
```

Alternatively, it is also possible to differentiate between revision 0.0 and 0.1 through software by examining SYSCON. Due to Anomaly #02000008 which existed in revision 0.0 and is fixed in revision 0.1, the BHD bit in the SYSCON register is set by default at reset in revision 0.0 but is cleared by default at reset in revision 0.1. This BHD bit state may be used to indicate the silicon revision when differentiating between 0.0 and 0.1.

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

## 20. 02000020 - Setting MAXBL[1:0] to 01 corrupts MMS write transmitted data:

**DESCRIPTION:**
Setting MAXBL[1:0] to 01 (enabling bursting) corrupts data transmitted during MMS writes. DSP to DSP writes are single cycle transfers and therefore bursting does not improve the throughput and should not be used. The DSP should ignore the setting of MAXBL for writes in MMS space. This anomaly indicates that MAXBL settings are not ignored and in fact have an adverse affect during MMS writes.

It has been observed that DMA master writes from one ADSP-21160 to another with MAXBL set specifically to 01 causes the transmitting ADSP-21160 to assert BRST and corrupt the data transferred.

**WORKAROUND:**
Clear MAXBL (MAXBL[1:0] = 00 ) during MMS writes. DSP to DSP writes are single cycle transfers and therefore bursting does not improve throughput and should not be used.

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

## 21. 02000021 - Idle cycle between even/even or odd/odd MMS banks not inserted:

**DESCRIPTION:**
Idle cycle between even/even or odd/odd MMS banks not inserted. An IDLE cycle is supposed to be inserted between any two accesses from different MMS banks. Due to this anomaly, an IDLE cycle is not inserted if the MMS bank changes from even ID to even ID or from odd ID to odd ID. For example, if ID#1 makes successive accesses to ID#4 and ID#6, an IDLE cycle will not be inserted as expected. This happens for both READ and WRITE accesses.

**WORKAROUND:**
Insert dummy READ or WRITE operation from external memory space.

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

## 22. 02000022 - The PLL does not reliably reset on power-up:

**DESCRIPTION:**
It has been found in bench level testing within ADI that the power supply sequencing affects this issue. Results indicate that the 2.5V supply must be powered before the 3.3V supply by a minimum of 0ns and a maximum of 200ms and this will allow the PLL to properly reset. If this power supply sequence cannot be met, then the workaround provided below must be implemented to ensure proper PLL reset. Also, Anomaly #02000023 must also be taken into consideration in order to avoid damage to the DSP.

How to recognize this problem exists. The following is a list of some (but not necessarily all) known symptoms of this anomaly:

1. The first and easiest symptom to detect is that CLKOUT is not running.
2. The next symptom which may manifest is that CLKOUT runs but the DSP fails to function properly. It has been observed that booting does not complete successfully (no $\overline{\text{HBG}}$ after $\overline{\text{HBR}}$, failure to attempt to read from EPROM, and link port fails to accept data).

**WORKAROUND:**
Perform all of the following steps while $\overline{\text{RESET}}$ is asserted:

1. Allow CLKIN to run for a minimum of 10 us.
2. Stop CLKIN (CLKIN can be held high or low) for a minimum of 10 us.
3. Restart CLKIN and allow to run for a minimum of 2000 cycles before deasserting $\overline{\text{RESET}}$.

This will reliably reset the PLL on power-up under all conditions.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 23. 02000023 - Power supply sequence (Core and I/O power supplies) can damage ESD diodes:

**DESCRIPTION:**
The I/O pads have internal diodes to protect the DSP from damage by electrostatic discharge (ESD diodes). These diodes connect the 2.5V to the 3.3V supplies internally, therefore they can be damaged any time the 2.5V supply is present without the 3.3V supply being present. Damage can occur if there is a significant amount of loading on the I/O due to the fact that I/O will be powered from the 2.5V supply through the ESD diodes.

**WORKAROUND:**
**Solution #1:**
The best recommendation to prevent damage to ESD diodes involves the addition of a Schottky power diode between the 2.5V and 3.3V supply to protect the ADSP-21160 from partially powering the 3.3V supply. The anode of the diode must be connected to the 2.5V supply. The diode must have a forward biased voltage of 0.6V or less and must have a current rating sufficient to supply the 3.3V rail of the system.

**Solution #2 (if you cannot implement Solution #1):**
The following two situations need to be addressed:
1. Power Up:
Always power up the 3.3V supply before, or at the same time as the 2.5V supply to prevent damage to ESD diodes.

2. Power Supply drop out:
In the event that the 3.3V supply fails (or is somehow removed from the system) the 2.5V supply should be powered down to prevent damage to ESD diodes.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 24. 02000024 - Link port change in direction of transfer corrupts first data word:

**DESCRIPTION:**
When a link port changes direction of transfer from transmitter to receiver, the first data word received is corrupted. The first byte of the first word is corrupted, all other data is received correctly.

The problem exists on all link ports and may be seen to manifest itself only on certain link ports depending upon various factors present in your system. The work around described below should be applied to all link ports to ensure proper communication.

**WORKAROUND:**
Whenever the direction of transfer for a link port is changed from transmitter to receiver, the first data word sent to this link port should be a dummy word. This dummy word can be discarded and subsequent words can be considered valid.

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

## 25. 02000025 - Simultaneous DMA/CORE access of EPBx:

**DESCRIPTION:**
If the core attempts to access an EPBx using a 32-bit transfer while any other EPBx has a 64-bit internal DMA transfer in progress, the core transfer will function as a 64-bit access. To clarify further, the core access happens as a 32-bit access but read pointers are incorrectly updated as if it was a 64-bit access.

Therefore, the core will hang when only one 32-bit word is available. Also, half of the intended data will be lost whenever a transfer occurs. It is important to understand that under normal operation the DSP will attempt to optimize throughput by performing 64-bit DMA transfers if certain conditions are met even if the programmer only sets up for 32-bit transfers. The DMA controller will automatically attempt to perform 64-bit DMA transfers if all of the following are true:

```
DMA parameters:
IIx is 64-bit aligned (even normal-word address)
IMx = 1 or -1
Cx = an even number of 32-bit words

DMACx control register:
PMODE = 000 (No-Packing), 001 (16-32/64), 100 (32-32/64)
EPBx buffer depth is > 1 (more than one 32-bit word available to transfer)
DTYPE = 0  (Data type is data not instructions)
INT32 = 0  (Do not force 32-bit internal transfers)
```

If any of these conditions are not met, the DSP will essentially be forced to perform only 32-bit transfers. So, the user should be aware that even if 32-bit transfers have been programmed, the DSP will attempt to optimize throughput by performing 64-bit transfers if these conditions are met. Further details may be found in chapter 6 of the ADSP-21160 Hardware Reference Manual.

**WORKAROUND:**
Set INT32 bit in the respective DMAC. However, this will reduce the bandwidth on IOD to half the maximum capability of the ADSP-21160.

**Additional ideas for work arounds considered less robust than the above solution:**

Only perform 64-bit accesses of EPB via the core. With this option, a potential pitfall for users would be the inability to determine if at least two 32-bit words are present in the EPB since the EPB status only describes the states FULL, PARTIALLY FULL, and EMPTY. None of these states tells the user if at least two 32-bit words are available.

One suggestion would be to have the core only perform 64-bit reads which could either be polling or interrupt driven but should only be done when the buffer is FULL. This could work but would be tedious for single word transfers or the last seven 64-bit reads as the FIFO would have to be filled with up to seven 64-bit (or fourteen 32-bit) dummy writes. It would also not be very efficient for interrupt driven transfers as the ISR would also require a polling routine to be certain that the FIFO is FULL.

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

## 26. 02000026 - Host asynchronous writes may fail:

**DESCRIPTION:**
Host asynchronous writes fail if the rising edge of the $\overline{WRH}$ strobe is relatively coincident with the rising edge of the internal core clock. The write data may be written to the wrong address.

**WORKAROUND:**
Only deassert the $\overline{WRH}$ signal in the following window:

| CLK ratio | Allowed range for $\overline{WR}$ rising edge after initial rising edge of CLKIN |
|---|---|
| 2:1 | [(n(tCK/2)+tCK/4)- Delta1 to (n(tCK/2)+tCK/4) + Delta2] |
| 3:1 | [(n(tCK/3)+tCK/6)- Delta1 to (n(tCK/3)+tCK/6) + Delta2] |
| 4:1 | [(n(tCK/4)+tCK/8)- Delta1 to (n(tCK/4)+tCK/8) + Delta2] |

where:
Delta1 is 1.5ns
Delta2 is 4.5ns
n is any # of core cycles

**Note:**
This work around is intended to provide a window in which it is safe for the write rising edge to occur. This window has been verified over frequency and temperature. In order to determine the range for $\overline{WR}$ after any rising edge of CLKIN use n=0 in your calculation.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 27. 02000027 - Host asynchronous access overlapping DSP to DSP transfer fail:

**DESCRIPTION:**
Assertion of $\overline{CS}$ by the host to a DSP which is currently the slave in a DSP to DSP transfer can cause the DSP to DSP transfer to fail and as a result cause this particular chip which is selected (selected via $\overline{CS}$ asserted) to fail to respond properly for subsequent host transfers.

Example:
DSP A performs writes to DSP B. During one of these writes the host attempts to perform a transfer to DSP B by asserting $\overline{HBR}$ and $\overline{CS}$ (of DSP B). This causes the DSP A to DSP B write transfer to fail and may cause subsequent host accesses to fail with REDY deasserted forever.

**WORKAROUND:**
Qualify all assertions of $\overline{CS}$ with the assertion of $\overline{HBG}$. In other words, do not assert $\overline{CS}$ until after $\overline{HBG}$ is asserted.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 28. 02000028 - SPORT transfers fail at frequencies above 35 MHz:

**DESCRIPTION:**
At SCLKx frequencies above 35MHz both SPORT0 and SPORT1 fail to transfer and receive (latch) data correctly. This frequency related issue affects the serial ports both when they transmit and when they receive.

**WORKAROUND:**
Do not exceed 35 MHz SCLK frequencies to ensure proper SPORT operation.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 29. 02000029 - SPORT companding fails for first active transmit channel in multichannel frame:

**DESCRIPTION:**
When the SPORT is configured for multichannel mode and A-law/μ-law companding is enabled, companding will not work for the first enabled transmit channel in any frame beyond the first active frame after the SPORT is enabled.

This companding failure occurs only in the first enabled transmit channel, regardless of which channel is "the first enabled transmit channel".

For example, if slots 0 to 31 are enabled and set to be compressed by the companding logic, slot 0 data will not be compressed after the tx 1st frame. If, however, slots 0-4 are disabled and slots 5-31 are enabled and set to be compressed, then slot 5 data will not be compressed after the 1st tx frame.

This failure arises only in transmission of compressed data, not reception (expansion) of compressed data.

**WORKAROUND:**
**Solution #1:**
Do not use SPORT companding (compression) in multichannel mode for the first transmit channel when multichannel operation is used.

**Solution #2:**
It is possible to implement A-law or μ-law compression in software for the first tx channel. The estimated software overhead of compressing a 16-bit data word to compressed 8-bit data is approximately 18 to 19 DSP instruction cycles. For information on implementing software A/μ-Law compression, refer to Chapter 11 in the Digital Signal Processing Applications Using the ADSP-2100 Family manual, Volume 1 (Prentice Hall, 1992: No longer available in hardcopy).

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 30. 02000030 - SIMD mode is adversely affecting data accesses to odd external memory addresses:

**DESCRIPTION:**
Example:
i9 = dm(address) does not yield correct results when 'address' is an odd address in external memory space.

i9 = dm(0x1,i1) with i1 = 0x800 000 tested and observed to work incorrectly as detailed below.

With SIMD mode enabled, i9 gets loaded with contents at 0x800 000, not 0x800 001 as expected due to the pre-modify operation.

With SIMD mode disabled, i9 gets loaded correctly with contents at 0x800 001.

Internal memory accesses always work correctly with both even and odd addresses being accessed. For example, for instruction i9 = dm(0x1,i1) with i1 = 0x50000 (internal memory), i9 gets loaded correctly (contents of 0x50001), regardless of SISD or SIMD mode.

**WORKAROUND:**
Disable SIMD mode if the following types of accesses are required in your application code:
DEST = dm(address), where 'DEST' is any register and 'address' is an odd address in external memory space.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 31. 02000031 - Broadcast mode works improperly when source buffer is located in external memory:

**DESCRIPTION:**
Correct Broadcast Mode loading is defined as follows: The DSP's BDCST1 and BDCST9 bits in the MODE1 register control broadcast register loading. When broadcast loading is enabled, the DSP writes to complementary registers or complementary register pairs in each processing element on writes that are indexed with DAG1 register I1 (if BDCST1 =1) or DAG2 register I9 (if BDCST9 =1). Broadcast load accesses are similar to SIMD mode accesses in that the DSP transfers both an explicit (named) location and an implicit (un-named, complementary) location, but broadcast loading only influences writes to registers and writes identical data to these registers.

When the source for the Broadcast Mode loads is located in external memory, the explicit data register is loaded with the correct value however the implicit data register is loaded with an incorrect value.

**Note:** When Broadcast Mode loads are performed using a buffer in internal memory as the source, the DSP works as expected.

**WORKAROUND:**
Do not use Broadcast Mode when source is located in external memory.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 32. 02000032 - 64-bit data accesses in internal memory using the LW mnemonic do not function correctly with odd explicit address:

**DESCRIPTION:**
64-bit data accesses in internal memory using the LW mnemonic (using either direct or indirect addressing) do not function correctly with odd explicit address. The explicit part of the normal word address using LW mnemonic access will incorrectly access the next sequential even address. In other words, the LW mnemonic access crosses 64-bit boundary for internal memory access with odd explicit address.

Example of incorrect operation:

```
r12 = dm(0x00050003)(LW);
r12 gets data from location 0x50004
r13 gets data from location 0x50003
```

The above example illustrates anomalous behavior present in silicon revisions noted in the Anomaly Summary Table detailed at the beginning of this document.

This access type should result in DSP operation described as follows: All Long word accesses load or store two consecutive 32-bit data values. The register file source or destination of a Long word access is a set of two neighboring data registers in a processing element. In a forced Long word access (uses the LW mnemonic), the even (Normal word address) location moves to or from the explicit register in the neighbor-pair, and the odd (Normal word address) location moves to or from the implicit register in the neighbor-pair. For example, here is a description of proper operation for the ADSP-21160:
Example of correct operation:

```
r12 = dm(0x00050003)(LW);
r12 gets data from location 0x50002
r13 gets data from location 0x50003
(r12 neighbors r13)
```

**Note:** Use of the LW option for 64-bit accesses to external memory works correctly and as expected.

Example:

```
r14 = dm(0x00800003)(LW);
r14 gets data from location 0x800002
r15 gets data from location 0x800003
```

**Note:** The above description applies to both direct and indirect memory addressing. For more information on types of memory addressing, refer to the ADSP-21160 Instruction Set Reference.

**WORKAROUND:**
Ensure that all LW accesses to internal or external memory are even address aligned for consistent and correct functionality.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 33. 02000033 - Link port to link port transfers do not function reliably at all frequencies. At certain frequencies, the link port transmitter does not meet link port receiver data setup and hold timing requirements:

**DESCRIPTION:**
Maximum throughput varies across link port transmit/receive pairs. Complete details of maximum throughput for link port to link port transfers can be found in the ADSP-21160M data sheet. Link port transmit/receive pairs do not function reliably for Link Clock (LxCLK) frequencies above data sheet maximum throughput values.

If a transmit/receive device other than a link port (i.e., an FPGA, etc.) is used to communicate with a ADSP-21160M link port, higher data throughput can potentially be attained. Please refer to the link port timing specifications in the ADSP-21160M data sheet to determine timing requirements.

**WORKAROUND:**
Do not exceed maximum throughput when operating link port transmit/receive pairs as detailed in the ADSP-21160M data sheet, revision 0 and later, in order to ensure reliable link port operation.

**Alternatives:**
By examining the maximum throughput values detailed in the ADSP-21160M data sheet, revision 0, it is evident that several options exist:
1. If 80MHz DSP core frequency is to be used on all DSPs in a system, then the link ports may be operated at 1/2x core clock frequency (40MHz) in order to ensure reliable operation.
2. If all six link ports on every DSP in a system are to be used, then all of the link ports may be operated at link port clock frequency up to 62.5MHz (worst case throughput shown in data sheet table) and the DSP core clock frequency must also be set up to 62.5MHz in order to obtain a LxCLK:CCLK ratio of 1:1.
3. If you wish to only use a subset of the link ports, operate the DSPs and their link ports at appropriate frequencies as indicated by the maximum throughput values shown in the data sheet table.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 34. 02000034 - Link port data corruption may occur when a particular alignment of LCLK and CK occurs:

**DESCRIPTION:**
Data corruption on the link port receiver occurs under particular alignment of Link Clock (LxCLK) to Core Clock (CK) and only when Link Port Clock Divisor (LxCLKD) is set to 1. Data corruption does not occur on transmission, only on reception and can occur at any frequency and on any link port. Data corruption does not occur when Link Port Clock Divisor is set to a value other than 1, i.e., LxCLKD = 2, 3, or 4.
If the DSPs that are communicating via link ports share the same CLKIN signal, data corruption can only occur if the CLKIN and LxCLK skew is on the order of 4ns and only when Link Port Clock Divisor (LxCLKD) is set to 1. This skew value is quite large and so it is highly unlikely that the failing clock signal alignment will occur in this case.

If the DSPs that are communicating via link ports do not share the same CLKIN signal, and the link port clock divisor is set to a value of 1, the failing alignment between the two clock signals can occur causing data corruption. If the link port clock divisor (LxCLKD) is set to a value of 2, 3 or 4, data corruption will not occur. This is the case regardless of whether the core or DMA performs data transfers over the link ports.

With link port clock divisor set to 1, MSB 8-bits (nibble mode) or MSB 16-bits (byte mode) may be corrupted depending on which transfer mode is in use. Data corruption may be seen only in odd words when transferring 32-bit words in byte mode using link port clock divisor set to 1 (see solution #3 below). Data corruption may be seen in all transferred words when transferring in all other modes with link port clock divisor set to 1.

In systems where a device other than a SHARC is acting as a transmitter to communicate with an ADSP-21160M link port receiver and these devices do not share the same CLKIN, it may be possible to avoid the problematic clock signal alignment by manipulating the phase of the transmitting device's clock signal. Further details describing the problematic clock signal alignment when LxCLKD is set to 1 (i.e., "the window of relative coincidence") are provided in the table shown below in the Work Around section.

**WORKAROUND:**
**Solution #1:**
Ensure that all devices which are communicating via link ports share the same CLKIN signal and CLKIN to LxCLK skew does not place the clock signals within the window of relative coincidence. If a device other than a DSP is communicating with a DSP via link ports, take steps to ensure that the alignment of LxCLK and CLKIN edges on the receiver does not fall within the window of relative coincidence by manipulating the phase of the transmitting device's clock signal for example.

Window of Relative Coincidence:
The following equations describe the "keep out" zones for receiver LxCLK falling edge relative to the receiver DSP CLKIN rising edge & period (tCK). This information applies only when Link Port Clock Divisor (LxCLKD) is set to 1:

| Core Clock to CLKIN ratio | zone 1 | zone 2 | zone 3 | zone 4 | x value |
|---|---|---|---|---|---|
| 2:1 | (tCK/2 -5.5)+/- x | (tCK -5.5)+/- x | na | na | .5(tCK/4) |
| 3:1 | (tCK/3 -5.5)+/- x | (tCK/(3/2) -5.5)+/- x | (tCK -5.5)+/- x | na | .5(tCK/6) |
| 4:1 | (tCK/4 -5.5)+/- x | (tCK/2 -5.5)+/- x | (tCK/(4/3) -5.5)+/- x | (tCK -5.5)+/- x | .5(tCK/8) |

If the transmitter clock frequency is equal to CK (ie 1:1 link port clock divisor ratio) and failing phase relation between the clocks occurs, then there can be a failure. If the Transmitter clock frequency is lesser say 1/2, 1/3, or 1/4 of DSP's CK then there will be no data corruption at all.

There will be a LCLK to CK frequency ratio above which corruption may happen and below which it can't and we have verified that ratio to be greater than .5 and less than 1.

The following recommended solutions apply only if your system employs DSPs that do not share the same CLKIN signal and are communicating via link ports.
**Solution #2:**
Operate link ports using link port clock divisor (LxCLKD) equal to 2, 3, or 4 yielding Core Clock:LxCLK ratio of 1:2, 1:3, or 1:4 respectively. Data corruption will not occur in these link port configurations.
**Solution #3:**
This solution only applies when transferring 32-bit data words in byte mode using DMA over the link ports. It does not apply to 48-bit word transfers or transfers using the core.

Since data corruption occurs only in odd word transfers, it is possible to transfer twice as much data and discard every odd word where

data corruption occurs. This will affect link port throughput since every other word is discarded.
For example, if the intended 32-bit data pattern to be sent is as follows:

```
0x11111111
0x22222222
0x33333333
0x44444444
0x55555555
```

Then send the following pattern instead and discard every odd word that is received beginning with the first word as shown:

```
0x11111111 (discard)
0x11111111 (keep)
0x22222222 (discard)
0x22222222 (keep)
0x33333333 (discard)
0x33333333 (keep)
0x44444444 (discard)
0x44444444 (keep)
0x55555555 (discard)
0x55555555 (keep)
```

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 35. 02000035 - DMA handshake modes limited in throughput:

**DESCRIPTION:**
Characterization data reveals that nonsynchronous timing specifications tWDR ($\overline{\text{DMARx}}$ width low) and tDMARH ($\overline{\text{DMARx}}$ width high) limit throughput for three DMA handshake modes; paced master mode, handshake mode and external handshake mode.

The sampling rate of $\overline{\text{DMARx}}$ signal by the internal circuitry of the ADSP-21160 prohibits maximum throughput at core clock to CLKIN ratio of 2:1.

DMA transfers should be supported at the full CLKIN rate for all clock configurations. However, full bandwidth at 2:1 core clock to CLKIN ratio is not possible. Core clock to CLKIN ratios of 3:1 and 4:1 will support full speed throughput at the CLKIN frequency.

**WORKAROUND:**
If full CLKIN rate is required, synchronize the assertions/deassertions of $\overline{\text{DMAR}}$. See ADSP-21160M data sheet for timing details.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 36. 02000036 - MMS reads following MMS broadcast writes do not function properly:

### DESCRIPTION:

For example, in a multiprocessor system, if a DSP (or a host device) performs a broadcast write in MMS space, subsequent reads in MMS space by any device will fail. If the same device which performed the MMS broadcast write performs one simple write in MMS space then subsequent MMS reads will operate correctly.

### WORKAROUND:

**Solution #1:**
The device which performed the MMS broadcast write should perform one MMS write operation (at any time after the original MMS broadcast write) before MMS reads are attempted by any other processor. This MMS write operation will clear the problem such that all subsequent MMS read operations will be performed correctly.

**Solution #2:**
When any other device (other than the device which performed the broadcast write) performs a non-broadcast MMS write, it can clear the problem of failing MMS reads if the transfer of bus mastership i.e., BTC (Bus Transition Cycle) happens not in the very next cycle after broadcast write ends but after at least one CLKIN cycle. In this case, the delay of the BTC for at least 1 CLKIN cycle is absolutely necessary in order to clear the problem and allow subsequent MMS read operations to operate correctly. The following example should help to illustrate this further:
1. In cycle 1, device ID1 performs broadcast write.
2. Cycle 2 is the required idle cycle.
3. In cycle 3, bus mastership transition takes place.
4. In cycle 4, device ID2 performs non-broadcast MMS write.

The problem is now cleared.

If the CLKIN delay cycle in step #2 shown above is not present and BTC occurs in cycle 2 instead of in cycle 3, the state machine in device ID2 (the new bus master) will not be properly cleared. When device ID2 (the new bus master) performs a non-broadcast MMS write, then the state machines inside all devices except device ID2 (the new bus master) will be cleared of the problematic state. MMS reads from the new bus master device ID2 will not operate properly. The problematic state can be cleared on this device, which did not wait long enough to obtain bus mastership and perform a non-broadcast MMS write, by any other device by performing a non-broadcast MMS write. There is NO minimum delay required in this case. The BTC here can occur in the very next CLKIN cycle.

**Solution #3:**
Do not use MMS broadcast writes and instead use single MMS writes to each DSP. This will completely avoid the problematic state where subsequent MMS read failures occur. This work around will incur additional execution time since more than one single MMS write operation will be required in place of a single MMS broadcast write operation in order to write to multiple DSP slaves.

### APPLIES TO REVISION(S):

0.1, 1.1, 1.2

## 37. 02000037 - BMAX/BCNT counter expire corrupts synchronous burst transfers in progress during a BTC (bus transition cycle):

**DESCRIPTION:**
BMAX countdown expire corrupts burst reads and burst write transfers in the middle of a burst transfer, instead of forcing a bus transition cycle after the burst transfer sequence completes at the burst boundary.

**WORKAROUND:**
**Solution #1:**
Disable bursting during external port DMA transfers by clearing the MAXBL[1:0] bits in DMACx if the bus timeout feature of the SHARC is critical to your system.

**Solution #2:**
If the synchronous bursting protocol is required in your system to increase throughput for DMA transfers, then do not use the bus timeout feature by programming BMAX. If the slave requests the bus using core transfers, then the $\overline{PA}$ pin can be used by the slave to gain bus mastership.

**Solution #3:**
A less efficient workaround would be to reduce the size of background DMA transfers to smaller blocks to allow a slave processor to gain control of the bus.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 38. 02000038 - $\overline{SBTS}$ deassertion can cause a resumed external memory access to abort:

**DESCRIPTION:**
The condition occurs when $\overline{SBTS}$ interrupts a core transfer of data to/from external memory or when the program sequencer is executing instructions from external memory. $\overline{SBTS}$ has the capability to interrupt external memory core accesses without their completion. When any banked external memory access is resumed following an $\overline{SBTS}$ deassertion, the $\overline{MSx}$ line does not reassert, hence the current external memory access is aborted.

**WORKAROUND:**
Do not assert $\overline{SBTS}$ during a banked external memory access by the DSP.
In order for a host processor to gain control of the system during a deadlock resolution, consider the following to force the DSP to halt core data accesses or execution of instructions from external memory:

The host can use a programmable I/O pin to assert an $\overline{IRQx}$ pin which would cause the DSP to vector to internal memory to the $\overline{IRQx}$ interrupt vector address, which then causes the DSP to halt external accesses, allowing the host to gain control of the system.

The host can use a programmable I/O pin to drive a flag input pin. The user must then insert polling instructions during the critical portions of code where the host will access the bus for a long period of time.

If large amounts of code are executed from external memory, then the user would need to insert JUMP instructions periodically in the external code to force the DSP to internal memory.

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

## 39. 02000039 - Shadow Write FIFO Anomaly:

### DESCRIPTION:

This anomaly has been identified in the shadow write FIFOs that exist between the internal memory array of the ADSP-21160M and core /IOP busses that access the memory. (See pg 7-73 of ADSP-21160M SHARC DSP Hardware Reference for more details on shadow register operation). A particular sequence of a core write followed by a read of the same internal memory address, in conjunction with a certain type of IOP activity can cause the core read to return incorrect data.
Under the circumstances described below, the Read from Addr 1 will incorrectly return the data for Addr 2.

**Case 1**

| Core Cycle | Core Activity | IOP Activity |
|---|---|---|
| x | Core write to address 1 in block X | IOP read from block X |
| y | Core write to address 2 in block X | IOP read from block X |
| z | Core read from address 1 in block X | No access from IOP to block X |

**Case 2**

| Core Cycle | Core Activity | IOP Activity |
|---|---|---|
| x | Core write to address 1 in block X | IOP write address 2, block X |
| y | Core read from address 1 in block X | No access from IOP to block X |

**Case 3**

| Core Cycle | Core Activity | IOP Activity |
|---|---|---|
| x | Core write to address 1 in block X | IOP read from block X |
| y | Core read from block X | IOP write address 2, block X |
| z | Core read from address 1 in block X | No access from IOP to block X |

Core cycles x, y, and z are not necessarily consecutive. Multiple continuous reads from both the core and IOP hold the state of the shadow write FIFO. Therefore multiple core and IOP reads could be inserted between Core cycles x,y and y,z thus delaying the occurrence of the failure relative to the original write.

**Note:** for the purposes of this anomaly, instruction fetches are considered core reads. The program sequencer will attempt to fetch an instruction every cycle. Thus, if program instruction code and data reside in the same internal memory block, the program sequencer will be performing reads from internal memory every cycle and will hold the contents of the shadow write FIFO static. Only write accesses to this block of internal memory where program instruction code resides will flush the contents of the shadow write FIFO. Two writes will be required to fully flush the FIFO.

This anomaly may potentially be encountered if overlays are being implemented. Typically, two dummy writes will be required to flush the FIFO if the overlay data is brought into a block of internal memory from which the program sequencer is fetching instructions. However, the most appropriate workarounds necessary to avoid anomalous behavior will depend upon how the user has implemented overlays via their own overlay manager.

This problem is caused by the shadow write FIFO erroneously returning data for a core read when data should have been returned from internal memory. During write operations, data is placed in the 1st stage of a 2 stage shadow write FIFO. Data is moved from 1st to 2nd stage when a second write is performed (by either DSP core or IOP) or if there is no internal memory read (by either DSP core or IOP) in subsequent cycles. Similarly data is moved from the 2nd stage of the FIFO to internal memory. On read operations, address compare logic allows data to be fetched either from internal memory or the fifos. Note there is one Shadow Register fifo per memory block and all core and IOP accesses to internal memory use this fifo. The internal memory clock (not visible to the user) runs at twice the core clock frequency. So, each core cycle consists of 2 memory cycles with one of the two memory cycles dedicated to the core and the other dedicated to the IOP.

### WORKAROUND:

**Work Around 1:**
Place 2 non-read cycles between a core write and read of the same address. This example avoids the failure:

```
Core write to Addr 1 in Block X
Core write to Addr 2 or compute or nop
Core write to Addr 3 or compute or nop
Core read from Addr 1 in Block X
```

*Ensure code is not running from Block X.

**Work Around 2:**
Ensure that IOP reads or writes of internal memory (either for TCB loading or for transferring data to/from external world via link, serial, or external ports including host accesses) occurs from one memory block while core accesses (writes and reads) use the other memory block.

**Work Around 3:**
Allow only one DMA channel to be active at a time and ensure that TCB location follows Work Around 1. Alternatively instead of a single active DMA channel allow host access.

**Transmit Case**
If the DMA channel is a transmit (internal to external), ensure that the code immediately after enabling the DMA does not perform this sequence in the 10 cycles following the enable:

```
Core Cycle n: Core write to Addr1 in Block X
              ***
Core Cycle n+1: Core write to Addr2 in Block X
              ***
Core Cycle n+2: Core read Addr1 from Block X
```

*** denotes multiple, continuous reads by the Core from Block X.

**Note:** A single transmit DMA channel will perform IOP reads every core cycle after the channel is enabled until the port's (serial, link, or external port) buffer is full. Thereafter, the DMA channel will not perform IOP accesses every core cycle.

**Receive Case**
If the DMA Channel is a receive (external to internal), ensure that the code that is running while the DMA is active does not have a sequence such as:

```
Core Cycle n: Core write to Addr1 in Block X
Core Cycle n+1: Core read Addr1 from Block X
```

**Note:** A single receive DMA channel will not perform IOP accesses every core cycle.

Similarly, if a host is performing writes to internal memory, this sequence must be avoided.

### APPLIES TO REVISION(S):

0.0, 0.1, 1.1

## 40. 02000040 - SIMD read from internal memory with Shadow Write FIFO hit does not function correctly:

**DESCRIPTION:**
This anomaly has been identified in the Shadow Write FIFOs that exist between the internal memory array of the ADSP-21160M and core /IOP busses that access the memory. (See pg 7-73 of ADSP-21160M SHARC DSP Hardware Reference for more details on shadow register operation).

If performing SIMD reads which cross Long Word Address boundaries (i.e. odd Normal Word addresses or non-Long Word boundary aligned Short Word addresses) and the data for the read is in the Shadow Write FIFO, the read will result in rev 0.0 behavior for the read.

Please refer to the description shown in Anomaly #02000003: SIMD mode explicit accesses of odd addresses in internal memory do not function correctly for details on the differences between SIMD data accesses between silicon revisions. The data access examples describing incorrect behavior in Anomaly #02000003 also apply to this anomaly when the above conditions are met. The examples describing correct behavior in Anomaly #02000003 apply when there is no Shadow register hit in the case of this anomaly.

**WORKAROUND:**
Align all variables and arrays in memory to Long Word Address boundaries via the use of the .ALIGN assembler directive. Do not explicitly access odd Normal Word addresses or non-Long Word boundary aligned Short Word addresses in SIMD Mode.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 41. 02000041 - Under special conditions, glitching may occur on $\overline{\text{HBG}}$ in a system with multiple DSPs and a Host:

**DESCRIPTION:**
Glitching of $\overline{\text{HBG}}$ may occur in systems constituted by a host device and at least two ADSP-21160 DSPs in cluster configuration. Under special conditions, the $\overline{\text{HBG}}$ signal may be simultaneously driven by more than one DSP in response to a host device request for bus mastership which will result in glitching on the $\overline{\text{HBG}}$ signal due to bus contention.

The glitch on $\overline{\text{HBG}}$ occurs when the current bus master relinquishes bus ownership but continues to assert $\overline{\text{HBG}}$ for 1 extra cycle. The new bus master may deassert $\overline{\text{HBG}}$ and cause contention that results in a glitch.

If ALL of the following conditions are true, then $\overline{\text{HBG}}$ glitching can occur:
1. A 21160 is the current bus master and deasserts its bus request signal ($\overline{\text{BRx}}$) in CLKIN cycle N.
2. Another 21160 is requesting the bus via assertion of its $\overline{\text{BRx}}$ at the same time.
3. $\overline{\text{HBR}}$ is asserted low in CLKIN cycle N-2 and N-1.

Both DSPs will drive $\overline{\text{HBG}}$ in N+1 cycle with opposite logic levels.
Only New bus master will drive $\overline{\text{HBG}}$ "high" in cycle N+2.
Only New bus master will start driving $\overline{\text{HBG}}$ "low" from cycle N+3 and $\overline{\text{HBG}}$ will be stable.

If $\overline{\text{HBR}}$ is asserted before or after this time window then this problem will not occur. If $\overline{\text{HBR}}$ is asserted low earlier, then the current bus master will keep asserted its $\overline{\text{BRx}}$ and will drive $\overline{\text{HBG}}$. If $\overline{\text{HBR}}$ is asserted after this window then bus mastership will change and the new bus master will drive $\overline{\text{HBG}}$. In the problem window (CLKIN cycles N-2 and N-1), one part of the current bus master DSP's internal logic thinks that it is still the bus master and processes $\overline{\text{HBR}}$ request and drives $\overline{\text{HBG}}$. Another part of the DSP's internal logic goes ahead and deasserts $\overline{\text{BRx}}$. Another DSP processor takes this deasserted $\overline{\text{BRx}}$ from old master and assumes that it has got the bus mastership and starts driving $\overline{\text{HBG}}$. This results in both Old and New bus master driving $\overline{\text{HBG}}$ with opposite logic level for some time. Eventually Old bus master stops driving $\overline{\text{HBG}}$. Because of this contention we see glitching on $\overline{\text{HBG}}$.

**WORKAROUND:**
Delay launch of any host accesses until $\overline{\text{HBG}}$ is in stable asserted state.

**For synchronous host:**
Sample $\overline{\text{HBG}}$ asserted low for a minimum of two consecutive CLKIN cycles prior to launching any host access.

**For asynchronous host:**
Delay launch of host access for at least two CLKIN cycles after $\overline{\text{HBG}}$ asserted.

$\overline{\text{HBG}}$ will have stabilized by the time indicated above.

**APPLIES TO REVISION(S):**
1.1

## 42. 02000042 - The first Host asynchronous read after HTC may fail:

**DESCRIPTION:**
The failure is manifested as corruption of the first data value read immediately following the Host Transition Cycle (HTC). Subsequent reads operate correctly and no data corruption occurs.

**WORKAROUND:**
Do not begin an asynchronous host read access in the cycle immediately following an HTC. Instead, delay the beginning of the read by one CLKIN period.

**For example:**
If an HTC occurs in CLKIN cycle N, host asynchronous reads should begin in CLKIN cycle N+2. If the read begins in cycle N+1, corruption may occur on the first data value transferred. Subsequent reads, while the host retains bus ownership, are correct and no data corruption occurs.

For asynchronous host reads, which begin in CLKIN cycle N+2, no anomalous behavior occurs and read operations execute correctly.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 43. 02000043 - Link port buffers and buffer status bits may not behave properly upon RESET affecting initial link port data transmission:

**DESCRIPTION:**
The link port buffers (LBUFx) will not be flushed and the link buffer status bits will not be set properly upon RESET if the link port buffer status is "not empty" at the time a RESET (software or hardware) is applied to the DSP. "Not empty" status indicates that there is one or more words in the LBUFx. Under these conditions unintended transmission of data values present in LBUFx prior to RESET will occur when the DSP's link port is configured to transmit and is enabled following the application of reset. Note that polling the LxSTAT bits of the LCOM register after a software reset to check the buffer status is not reliable because these status bits also fail during this particular condition.

This problem can be seen in all link port buffers, in all the link port clock modes of operation and under the following link port data modes:
1. DMA or Non DMA (core or interrupt driven) transfers.
2. 32-bit (LxEXT=0) or 48-bit (LxEXT=1) LBUFx word width.
3. 4-bit (LxDPWID = 0) or 8-bit (LxDPWID = 1) data transfers.
4. LxCLKD = 1,2,3,4.

This anomaly affects link ports configured as transmitters and does not affect link ports configured as receivers. If no data was present in the LBUFx prior to application of RESET this anomalous behavior will not occur.

**WORKAROUND:**
To prevent this from affecting your system, you must set up a small dummy transfer to clear the LBUFx FIFO status. Two methods of implementing dummy transfers to flush the link buffer data are described below.
**Note:** This work around will have no adverse impact if status was already clear.

**Method 1:**
The execution of a simple core driven link port loopback transfer between any two link port buffers will properly clear the LBUFx status for the two link buffers. Internal loopback operation also prevents the transmitting link port from driving data out to another connected external link port receiver. After the link port loopback completes, the link port can then be reprogrammed to operate in the required mode of transfer.

In order to flush all six link port buffers, the following set of steps can be implemented using internal loopback mode:
a. Enable 3 buffers as transmitters and 3 as receivers without any DMA or Interrupt service routines.
b. Establish *loopback mode transfer between them (i.e., assign one transmit buffer and one receive buffer to the same link port).
c. Write 2 dummy words to each transmit buffer.
d. Give enough time for data transfer to complete.(**15 link clock cycles will be enough).
e. Read two words from each receiver.
f. After the internal loopback mode transfer completes, disable the link ports.
g. Reconfigure selected link port as a transmitter in the mode required in your application code.

* - Details of link port loopback mode can be found in the ADSP-21160 Hardware Reference manual.
** - 15 clock cycles will be enough to ensure that buffer has became empty if the transmitter is enabled for clock ratio = 1:1 and 8bit data path.

One implementation of the above work around which may help save wasted clock cycles on unnecessary initialization of link port buffers following any reset could involve the use of one memory location (i.e. a semaphore) to indicate that link port transfer is ongoing prior to the occurrence of a reset. If reset aborts the ongoing link port transfer, then during the reset service routine that memory location can be checked, if the memory location indicates that a link port transfer was aborted, then the sequence described above should be followed otherwise there is no need to run this part of code.

**Method 2:**
Filling LBUFx with the first two pieces of valid data to be transmitted by the link port will overwrite any erroneous data that may have been left in the LBUFx due to the anomaly. When the core writes the data the transmitter link port should be disabled. After writing the two data words, the transmitter can then be enabled and transmission will start with the newly written data instead of with data that may have been stranded in the FIFO from a previous transfer due to the anomaly.

If DMA transfers will be used the programmer should manually read from the DMA buffer and use core instructions to copy the first two words from the DMA buffer. The DMA internal index register (IIx) should be offset initially by 2 locations. The DMA will take the responsibility for transferring data properly from the third word on.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 44. 02000044 - IMASKP bits are left shifted by 1 bit for writes to bits 14 to 30:

**DESCRIPTION:**
Direct writes via the DSP core to IMASKP[30:14] will result in a value being stored in IMASKP that is shifted left by one bit. When IMASKP is updated by normal operation of the DSP (i.e., during interrupt service), the correct value will be stored and read from IMASKP. The problem occurs during the direct write to IMASKP. Setting IMASKP[13:0] will not result in a left shift by 1 bit as these bits are not affected by this anomaly.

The following example illustrates anomalous behavior:

```
IMASKP = 0x04000000; // write to IMASKP register
R1 = IMASKP;  // read from IMASKP register
R1 will get the value 0x08000000
```

Note that the LPISUM (bit 14) is a read only bit and cannot be modified, however attempting to set bit 14 will result in bit 15 being modified. When the DSP is executing a link port interrupt, bit 14 is correctly set to indicate that a link port interrupt is being serviced. IMASKP bit 31 is a reserved bit and cannot be modified.

**WORKAROUND:**
In order to store the intended value in IMASKP correctly, users must first bit shift the value to be written to IMASKP[31:14] one position toward the LSB, then OR this value with the lower 14 bits before writing the 32-bit value to IMASKP. This will result in the appropriate interrupt bits being set.

**APPLIES TO REVISION(S):**
1.1, 1.2

## 45. 02000045 - Inactive EPBx DMA channel parameter registers (EIx, EMx, ECx) may be read incorrectly:

**DESCRIPTION:**
Active DMA channel parameter registers can always be read reliably. Reading inactive External Port Buffer DMA channel parameter registers (associated with the DMA external address generation circuitry EIx, EMx, ECx) will not reveal correct results when there is a DMA pipeline stall. The contents of the inactive EPBx external DMA channel registers are not corrupted; the read values simply do not reflect the actual contents. EPBx DMA functionality works properly and is not affected by this reporting error.

A DMA pipeline stall occurs when the DMA controller is unable to write to a peripheral buffer. Typical situations where DMA controller stalls can occur are when core external accesses are configured to have a higher priority than DMA and DMA is held off to allow a core access to compete, or when a peripheral transmitting data sends data out at a slower rate than the DMA controller writes the peripheral buffers.

An active EPBx DMA channel is one that is enabled and currently performing an internal<-> external memory access, while an inactive EPBx channel is enabled but is not currently performing an access.

**WORKAROUND:**
The DMASTAT register should be used to check the status of external port DMA channel activity on channels 10 to 13. Code should not test DMA status based on the EPBx DMA channel parameter registers. This reporting error for inactive EIx, EMx, ECx DMA channel parameter registers should be considered when viewing the DMA addressing window in the debugger interface.

**APPLIES TO REVISION(S):**
0.1, 1.1, 1.2

## 46. 02000046 - Direct writes to IMASKP or LIRPTL may cause highest priority interrupt to be serviced twice:

### DESCRIPTION:

The problem is observed when LIRPTL or IMASKP register is being written to and simultaneously an interrupt servicing starts, irrespective of the source of interrupt generation (Interrupt could be caused by writing to IRPTL, LIRPTL or it could be a normal interrupt). If beginning of an interrupt servicing overlaps with the execute phase of LIRPTL/IMASKP load, the problem occurs.

This will problem will occur only if the following two conditions are met:
1. User performs direct write to LIRPTL or IMASKP register.
2. An interrupt has just started servicing. This interrupt could be caused by direct write to IRPTL or could be a normally occurring interrupt.

For example, if we have the following instructions and condition #2 shown above is true:

```
bit set IRPTL  0x7FFFFDFF; //user direct write to register
bit set LIRPTL 0x003F003F; //user direct write to register
```

Then the highest priority interrupt (IICD in this example) is executed twice, once in the beginning of interrupt servicing and once after all the interrupts are serviced.

If multiple interrupts, which are all unmasked, are latched, the processor starts servicing the highest priority interrupt and then services the other interrupts in order of their priorities. When all the interrupts are serviced, it jumps to the ISR of the highest priority latched unmasked interrupt and services it again. Also during this interrupt service the bit in IMASKP corresponding to this highest priority interrupt is not set.

This behavior does not occur when multiple interrupts are latched in IRPTL without latching any interrupt in LIRPTL. Also when multiple interrupts are latched in LIRPTL without latching any interrupt in IRPTL this behavior will not occur.

Thus, anomalous behavior will occur with a sequence of instructions of the form shown below if an interrupt has just started servicing upon completion of the two instructions (note the instruction order)

```
bit set IRPTL
bit set LIRPTL
```

Furthermore, this anomalous behavior does not occur if we have the following set of instructions and an interrupt has just started servicing upon completion of the two instructions (note the change in instruction order)

```
bit set LIRPTL
bit set IRPTL
```

Two 'bit set' instructions are not necessary to reproduce the problem. For example, the instruction, "bit set IRPTL ...;", can be replaced by a normally occurring interrupt. Note that a software breakpoint is restricted immediately after bit manipulation instructions of LIRPTL/IMASKP as software breakpoint triggers an emulator interrupt.

### WORKAROUND:

If artificial latching of interrupts is required via direct user writes to LIRPTL/IMASKP, first mask all interrupts before writing to LIRPTL/IMASKP and then unmask them upon completion of write. This will preclude the overlap between LIRPTL loading and beginning of an interrupt service.

### APPLIES TO REVISION(S):

0.1, 1.1, 1.2

## 47. 02000047 - Asynchronous Host Reads can fail if tSADRDL is less than one external clock cycle:

**DESCRIPTION:**
The DSP is not meeting the tSADRDL specification of 0ns and as a result the address of a host read may not latch properly causing the data from the previous address read to be driven on the bus. If consecutive reads are occurring from the same address, for example when reading subsequent times from the external port buffer, all but the first host reads will be performed correctly.

**WORKAROUND:**
Guarantee that the address is valid in the external clock cycle previous to the one where the $\overline{RD}$ is asserted will resolve this problem. tSADRDL should be a minimum of one external clock cycle.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 48. 02000048 - 32-bit wide link port transfers limited in throughput with 1:1 LCLK-to-CCLK ratio:

**DESCRIPTION:**
There is a latency in the internal update to the link port transmit buffer status delaying a DMA request. This occurs when a link port running at a 1:1 core to link clock ratio is transmitting byte wide link port data with DMA. This latency results in a stall of 2 link clock cycles for every other word transmitted out of the link port. This anomaly does not create data loss or corruption, only a reduction in overall transfer speed.

The following link port transfers work properly and are not affected by this anomaly:
1. 48 bit wide transfers.
2. Nibble wide link port transfers.
3. Transfers where the link ports are running at 1:2, 1:3 and 1:4 link clock to core ratios.

**WORKAROUND:**
Maximum throughput can be achieved by sending a 32-bit data buffer as 48-bit data. Because of the way memory is organized in the DSP, data in 32-bit (2 column) memory can be accessed as 48-bit (3 column) memory. (Memory organization is discussed further in the hardware reference manual.) Data buffers in 2 column memory can be transferred as 48-bit data using the following 3 steps:

1. Program the DMA index register to point to the 3 column address properly corresponding to the beginning of the 2 column data buffer. In order for the first address in a two column data buffer to be translated properly into a three column address the buffer must be located at an address whose value is a multiple of 3 in 32-bit addressing. In other words, the buffer would need to start at address 0x50000, 0x50003, 0x50006, 0x50009, 0x5000C etc. in block 1. The 48-bit address translation is obtained by multiplying the decimal address by 2/3. For example:
location 0x5000C in 32-bit space would translate to 0x50008 in 48-bit space,
0x5000C - 0x50000 = 0xC = 12 decimal,
12 *2 / 3 = 8 = 0x8.
2. Program the DMA count register so that it will send the number of 48 bit words that corresponds to the length of the data buffer. Each 48-bit word that is transferred will consist of 1.5 32 bit words.
3. Configure the link port to send as 48-bit data with the LxEXT bit in the LCTL register.

The receiving DSP will also have to be configured to receive 48 bit-words and will place the words in memory such that they will be accessible as 32-bit data from a buffer declared in 2 column memory.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 49. 02000050 - Conditional type 10 instruction may fail in SIMD:

**DESCRIPTION:**
Given that the type 10 conditional instruction is as follows (see page 4-14 of the ADSP-21160 SHARC DSP Instruction Set Reference for more details on the instruction type):

```
IF COND Jump |(Md, Ic)        |, Else compute, |DM(Ia, Mb)=dreg;|
             |(PC, <reladdr6>)|                |dreg=DM(Ia, Mb);|
```

In SIMD mode, if the condition is TRUE for both PEx and PEy the jump occurs and if any condition is FALSE, then the else block of this instruction is executed. If both conditions in PEx and PEy are FALSE the I register post modifies as intended. The I register erroniously fails to post modifiy if only one of the conditions in PEx and PEy are FALSE. This instruction does not work as intended only if one, but not both, of the conditions in PEx and PEy are FALSE.

**WORKAROUND:**
When in SIMD mode, substitute a type 8 instruction and a type 4 to replace the type 10 instruction. For example:

```
IF  av JUMP (PC , 0x 0b)  , ELSE R3 = R1 + R2 , DM(I1, M7) = R5 ;
```

could be separated into:

```
IF  av JUMP (PC , 0x 0c)
R3 = R1 + R2, DM(I1, M7) = R5 ;
```

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 50. 02000051 - Broadcast load fails in type 10 instruction:

**DESCRIPTION:**
Broadcast loads as described on page 4-17 of the ADSP-21160 SHARC DSP Instruction Set Reference fail. In the example SIMD instruction, IF TF JUMP(M8, I8), else R6=dm(I1,M1);, both R6 and S6 should be loaded with the value in the address pointed to by I1. On the DSP, when the instruction is executed R6 is loaded properly, but S6 is erroneously loaded with the value in the address pointed to by I1+1.

**WORKAROUND:**
When in SIMD mode, substitute a type 8 instruction and a type 4 to replace the type 10 instruction. For example:

```
IF  av JUMP (PC , 0x 0b)  , ELSE R3 = R1 + R2 , DM(I1, M7) = R5 ;
```

could be separated into:

```
IF  av JUMP (PC , 0x 0c)
R3 = R1 + R2, DM(I1, M7) = R5 ;
```

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 51. 02000052 - Rn=MANT Fx results will be rounded if RND32 is enabled:

**DESCRIPTION:**
The instruction Rn=MANT Fx was designed to work independently of the rounding mode, but it does not. For example, consider the following set of instructions:

```
R2=0x45678901;
F1=float R2;
R0=mant F1;
```

If rounding is enabled (RND32) the result in R0 would be R0=8ACF120000, but if rounding is not enabled the result would be R0=8ACF120200.

**WORKAROUND:**
If the desired result of the MANT instruction is unrounded, but rounding is enabled in the code, the user must disable rounding manually before executing the MANT instruction and then re-enable the instruction after the MANT instruction has been executed. Keep in mind that writes to MODE1 have a 2 cycle effect latency. The workaround implemented for the example presented above would be as follows:

```
Bit CLR MODE1 RND32;
R2=0x45678901;
F1=float R2;
R0=mant F1;
Bit SET MODE1 RND32;
```

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 52. 02000053 - In Serial Port Multichannel mode, an external RFS one cycle early causes data corruption:

**DESCRIPTION:**
In Multichannel mode the DSP should ignore an external frame sync that occurs when the SPORT is still in the process of receiving a frame of data. The DSP erroneously samples the RFS one cycle before the end of a frame of data so an externally generated RFS sent one cycle early would not be properly ignored. This will cause corruption of the current frame of data. Once the failure mode has occurred the DSP will no longer recognize valid RFS signals. This failure will only occur if the RFS signal comes one cycle early, so externally generated RFS signals received at any time before the last cycle of the frame will be ignored as intended. This failure occurs independently of the mulitchannel frame delay configuration as well.

**WORKAROUND:**
To avoid problems with multichannel mode in this configuration, ensure that the device transmitting the RFS signal does not send it one cycle before the end of an existing frame of data. Once this failure has occurred, the SPORT must be disabled, reconfigured and re-enabled to resume normal operation.

**APPLIES TO REVISION(S):**
1.1, 1.2

## 53. 02000054 - Illegal DAG stalls can occur under certain circumstances:

**DESCRIPTION:**
When a DAG register load is followed by a read of the same register the DSP automatically inserts a one cycle stall between those instructions. The DSP erroneously identifies certain instruction bit patterns to be DAG register reads when they are not. This results in an unintended additional stall. There are no functional failures beyond the stall.

**WORKAROUND:**
Typically DAG register loads are part of initialization code so the possibility of an additional stall is not a problem. In such cases no workaround is necessary.

In cases where cycle accuracy of code using the affected DAG register loads is critical (ex. M or I registers directly written to in a critical loop) a nop instruction must be inserted after the DAG register load to ensure cycle count predictability. In addition to a NOP any instruction that doesn't involve data addressing, modify/bit-reverse instructions or indirect jumps can be used.
If there is a series of loads to the DAG registers, then there need not be NOPs between each of the loads, only the last load needs to be followed by a NOP. For example:

```
B0 = 0x50000;
M0 = 0x1;
L0 = 0xFF;
B1 = 0x55000;
M1 = 0x1;
L1 = 0xFF;
NOP;  // This needs to be added for predicting the number of cycles for
      // execution accurately.
```

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 54. 02000055 - Execution of instructions that modify interrupt latch registers may cause incoming interrupts to be ignored:

**DESCRIPTION:**
When the execute phase of bit manipulation instruction that modifies an interrupt latch register is extended due to the core being held off, some of the interrupts that are latched during this period in the interrupt latch register can be lost. The core can be held off when fetching the next instruction from external memory, accessing data from external memory, reading from empty buffer, writing to full buffer or IOP register reads that take more than one core clock cycle.
The specific Group IV system register bit manipulation instructions that are affected are as follows:

```
BIT SET IRPTL <data32>; BIT SET LIRPTL <data32>;BIT SET IMASKP <data32>;
BIT CLR IRPTL <data32>; BIT CLR LIRPTL <data32>;BIT CLR IMASKP <data32>;
BIT TGL IRPTL <data32>; BIT TGL LIRPTL <data32>;BIT TGL IMASKP <data32>;
```

The interrupts that can be missed are IRQx, EMUI, TMZHI, TMZLI, VIRPT, LPxI, EPxI. The LpxI interrupts are affected only for DMA driven transfer mode.

This failure will occur under any of the following conditions:
1. When the bit manipulation instruction that modifies an interrupt latch register is executed from external memory.
2. When the bit manipulation instruction is executed from internal memory in a delayed branch to a JUMP or CALL to external memory. For example:

```
a. JUMP/CALL ext_mem_location (db);
   BIT CLR IRPTL <data32>;
   NOP;
b. JUMP/CALL ext_mem_location (db);
   NOP;
   BIT CLR IRPTL <data32>;
```

3.When the bit manipulation instruction is executed from internal memory and it is immediately followed by an external memory data access. For example:

```
a. BIT CLR IRPTL <data32>;
   dm(ext_mem) = r0;
b. BIT CLR IRPTL <data32>;
   pm(ext_mem) = r0;
c. BIT CLR IRPTL <data32>;
   r0 = dm(ext_mem);
d. BIT CLR IRPTL <data32>;
   r0 = pm(ext_mem);
```

4. When the bit manipulation instruction is executed from the emulator (running or stepping).
5. When the bit manipulation instruction is executed from internal memory and it is immediately followed by an access from a core breakpoint register. The core breakpoint registers are proprietary and are only used by our emulator. These will not cause an error in a user's application.

**WORKAROUND:**
1. The workaround for condition 1 is to place the bit manipulation operation in internal memory.
2. The workaround for condition 2 is to avoid the bit manipulation instruction from being within the delayed branch of a JUMP or CALL to external memory by placing it before the JUMP or CALL to external memory.
3. The workaround for conditions 3 and 5 is to place a NOP; instruction directly following the bit manipulation instruction.
4. Compiler fixes will be implented in a service pack scheduled for June 2003.

These fixes will cause the compiler to avoid the failure modes via the workarounds described above.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 55. 02000056 - A breakpoint following a JUMP, CALL, RTI or RTS can trigger an early/improper emulator break:

**DESCRIPTION:**
Because of the hardware's failure to detect when an EMUIDLE instruction has aborted, placing a breakpoint on a memory location directly after a JUMP, CALL, RTI or RTS can cause the emulator to break falsely with the error: "Emulator halted at software breakpoint, but no breakpoint found"

**WORKAROUND:**
Do not set a software breakpoint directly after a jump, call, rti or rts.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 56. 02000057 - DAG register write followed directly by a DAG register read fails during context switch:

**DESCRIPTION:**
Switching the context of the DAG registers from primary to secondary sets has a 1 cycle affect latency, enabling the following code structure:

```
BIT SET MODE1 SRD1L;// enable context switch in DAG1 from primary to secondary
I2 = 0x52;          // write to DAG1 primary register
R0 = I2;            // read of DAG1 secondary register
```

If the secondary I2 register is previously set to 0xAA, then in the instruction example above R0 should be 0xAA, however the DSP erroneously forwards the primary load to the secondary register read so that in the example above R0 will actually equal 0x52. This anomaly only affects a DAG write followed by a read that takes advantage of the latency of the context switch.

**WORKAROUND:**
1. Placing a nop or an instruction that does not write a DAG register between the read and the write in the example will yield the correctly expected results as follows:

```
BIT SET MODE1 SRD1L;// enable context switch in DAG1 from primary to secondary
I2 = 0x52;          // write to DAG1 primary register
Nop;
R0 = I2;            // read of DAG1 secondary register
```

2. Do not take advantage of the fact that the affect latency of a context switch should allow you 1 instruction cycle where your instructions will still pertain to the initial context. In the example below, the code is rearranged so that the primary load occurs before the context switch is initiated. Now the secondary registers will be correctly accessed after the requisite 1 cycle effect latency.

```
I2 = 0x52;          // write to DAG1 primary register
BIT SET MODE1 SRD1L;// enable context switch in DAG1 from primary to secondary
Nop;
R0 = I2;            // read of DAG1 secondary register
```

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 57. 02000058 - System registers written with a PM access do not have a 1 cycle effect latency:

**DESCRIPTION:**
The 1 cycle effect latency when accessing system registers does not occur when the accesses use the PM bus. For example, in the following instruction, the MODE1 setting will come into affect in the cycle just after the MODE1 write:

```
MODE1 = PM(I9,M9);
NOP; //MODE1 setting is in effect for this instruction
NOP; //without the anomaly, the setting would be in effect starting at this location
```

The following system registers are affected:
MODE1, MODE2, IRPTL, IMASK, IMASKP, MMASK, FLAGS, LIRPTL, ASTATX, ASTATY, STKYX, STKYY, USTAT1, USTAT2, USTAT3, USTAT4
DM accesses and bit manipulation instructions do correctly insert a 1 cycle effect latency when writing system registers.

**WORKAROUND:**
1. Use a DM access such as MODE1=DM(I4, M4) instead of PM accesses when writing to system registers.
2. Use bit manipulations instructions such as BIT SET MODE1 0x5; instead of PM accesses when writing to system registers.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 58. 02000059 - Conditional RTI fails in SIMD mode:

**DESCRIPTION:**
A conditional RTI in SIMD mode will not execute as expected. Consider the example:

```
IF COND RTI (DB);
```

The above instruction in SIMD mode should be executed when COND in both PEx and PEy processing elements are true.

The DSP's behavior in executing the instruction is as follows:
1. Branches to the address stored at the top of the PC stack.
2. Pops the status stack if the ASTATx/y and MODE1 status registers have been pushed - if the interrupt was $\overline{\text{IRQ2-0}}$ or the timer interrupt.
3. Clears the appropriate bit in the interrupt mask pointer (IMASKP) register.

The behavior seen with respect to the conditions in PEx and PEy are as follows:
a. When condition in PEx and PEy both are false then this instruction is not executed. This is expected.
b. When PEx is false and PEy is true, then action 1 does not take place but actions 2 and 3 take place. This is anomalous. In this case none of the above actions should be taken.
c. When PEx is true, and PEy is false then same as case b.
d. When PEx is true, and PEy is true then all three actions take place. This is expected.

**WORKAROUND:**
Do not include a conditional in an RTI statement if SIMD will be enabled. Seperating the conditional and the RTI into separate instructions. The following is an acceptable alternative:

```
If NOT cond jump (pc,2);
RTI;
```

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 59. 02000060 - Single instruction loops can terminate early:

**DESCRIPTION:**
In a single instruction non-counter based loop, the sequencer tests the termination condition every cycle. After the cycle when the condition becomes true, the sequencer completes three more iterations of the loop before exiting. But if the single instruction used in the loop is a PM instruction, then the loop is executed only two more times.

**WORKAROUND:**
1. Use a dm data move inside the single instruction loop.
2. Increase the length of the loop, by unrolling the loop or in the worst case by adding a "nop;" instruction.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 60. 02000061 - Bit reversal fails with indirect jump:

**DESCRIPTION:**
If bit reverse mode is set and the program tries to do an indirect jump, bit-reverse setting is not considered.
For example if the code given below is executed, DSP tries to jump to the location pointed to by I8 instead of jumping to the bit-reversed value of I8 i.e. 0x250000.

```
I8 = 0x0000A400;
M8 = 0x0;
BIT SET MODE1 BR8;
NOP;
JUMP(M8,I8);
```

**WORKAROUND:**
Do not use bit reverse mode with indirect jump.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 61. 02000062 - VIPD bit gets cleared when branching to ISR:

**DESCRIPTION:**
The DSP was supposed to clear VIPD bit on return from the VIRPT interrupt service routine. But the VIPD bit gets cleared right when branching to the VIRPT interrupt service routine.

**WORKAROUND:**
1. Use one of the FLAGS to indicate that the vector interrupt is processed. The host should now poll the FLAG instead of the VIPD bit. Also, it should toggle the flag before writing a new address into the VIRPT register.
2. Use any of the IOP registers (like MSGRx, unused peripheral registers etc) to indicate the vector interrupt status. The host should now read this IOP register before writing a new address into the VIRPT register.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 62. 02000063 - MU (Multiplier underflow) flag gets set anomalously for some non-underflow cases:

**DESCRIPTION:**
Multiplier underflow flag gets set anomalously for some non-underflow cases. This behavior can be narrowed down to the below given cases of input vectors:
1. This occurs only in floating point multiplication.
2. The input vectors are such that the resultant exponent is -126 (This is the smallest normal exponent that can be represented in IEEE format).
3. TRUNC bit of MODE1 is set to 0.
4. RND32 bit of MODE1 is cleared.

Please note that this anomalous behavior occurs for only some combinations of mantissa inputs. Two example pair of inputs (floating point numbers) are:

```
X = 0x008A7DBEF6
Y = 0x3F7FF9FFF9

X = 0x26FFF70002
Y = 0x193FFFFFFE
```

**WORKAROUND:**
None

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 63. 02000064 - Top of the loop address stack is filled with zero when PUSH LOOP instruction is executed:

**DESCRIPTION:**
When a PUSH LOOP instruction is executed, top of the loop address stack is filled with zero instead of getting filled with the contents of LADDR.

**WORKAROUND:**
After executing the PUSH LOOP instruction, LADDR must be written with the content which is to be pushed on the loop address stack. A write to the LADDR will update the top of the loop address stack.

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 64. 02000065 - DAG stall causes external port to malfunction:

**DESCRIPTION:**
Whenever the instruction sequence results in a DAG stall (see code example below & DAG register transfer restrictions section in the ADSP-21160 Hardware Reference Manual), for certain alignment of the stall cycle with external clock phase, the processor incorrectly responds as if it is performing an external memory access.
DAG stall code example:

```
i12 = dm(m7,i6); // inst1
r0  = pm(i12,0); // inst2
```

Instruction inst1 loads i12 register, and it is used in the next instruction, hence the sequencer will stall the next instruction for a cycle. These two instructions function properly. However, during the stall cycle, the processor assumes an external memory access and wrong switching can happen on $\overline{BRx}$, ADDR lines. Similarly the corresponding $\overline{MSx}$ line may de-assert unexpectedly in the middle. Note that the dm and the pm access in the example can be any combination of internal/external memory accesses. Note that the $\overline{RD}$ and $\overline{WR}$ lines do NOT switch. However, the incorrect $\overline{BRx}$ and ADDR assertions can result in:
- delayed $\overline{HBG}$ assertion if there is a $\overline{HBR}$ assertion typically in MP system.
- DMA read/write from/to wrong location, if external port DMA coincides with this cycle.

The following table lists the observation on the $\overline{MSx}$ line and the $\overline{BRx}$ line under various conditions:

| S.No | Condition | Example | $\overline{MSx}$ | $\overline{BRx}$ |
|---|---|---|---|---|
| 1. | DAG Register initialized to internal memory initially. | i12 = 0x40000; // instx<br>i12 = dm(m7,i6); // inst1<br>r0 = pm(i12,0); // inst2 | Y | Y |
| 2. | DAG Register initialized to external memory initially. | i12 = 0x4000000; // instx<br>i12 = dm(m7,i6); // inst1<br>r0 = pm(i12,0); // inst2 | Y | Y |
| 3. | Wait register is initialized with zero wait states. | r0 = 0x1800000;<br>dm(WAIT) = r0;<br>i12 = dm(m7,i6); // inst1<br>r0 = pm(i12,0); // inst2 | Y | Y |
| 4. | Wait register is initialized with non-zero wait states. | r0 = 0x01CE739C;<br>dm(WAIT) = r0;<br>i12 = dm(m7,i6); // inst1<br>r0 = pm(i12,0); // inst2 | Y | Y |
| 5. | In the DAG Stall sequence, the DAG register is loaded using dm indirect load from memory and accessed using pm access. | i12 = dm(i7,m6); // inst1<br>r0 = pm(i12,0); // inst2 | Y | Y |
| 6. | When the code(the DAG Stall sequence with conditional instruction) is executed from external memory. | i8 = 0x4000000; // instx<br>m8=1; // inst1<br>if ne pm(i8,m8)=r3; // inst2<br>// condition should become false | Y | Not Applicable. |

**Note:**
Y - Problem seen (For $\overline{MSx}$ case the $\overline{MSx}$ pulse will be deasserted in the middle of the DMA transfer and for the $\overline{BRx}$ case wrong assertion of $\overline{BRx}$ will occur).
N - Problem not seen (No wrong assertions of $\overline{MSx}$ or $\overline{BRx}$ seen).

**WORKAROUND:**
Insert an instruction that does not use DAGs (for example, nop;) between the two instructions, as follows:

```
i12 = dm(m7,i6);// inst1
nop;            // workaround
r0  = pm(i12,0);// inst2
```

**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, 1.1, 1.2

## 65. 02000066 - Read of slave DSP's EPBx by host or master DSP in a multiprocessing may return wrong data:

**DESCRIPTION:**
This failure occurs under the following conditions:
1. At least one master mode DMA and one slave mode DMA or core access to the EPBx are enabled or active at the same time.
2. The master mode DMA can be in any direction (transmit or receive).
3. The slave mode DMA must be in transmit direction, where the host or master SHARC is reading data from the slave SHARC's EPBx.
4. The core access must be a core write to the EPBx, where the host or master SHARC is reading data from the slave SHARC's EPBx.
5. The packing mode of the slave DSP's EPBx is programmed such that the internal side is 32/64bits (packing modes 001 and 100).

Under the above scenario, the Master mode DMA works correctly. However, the Slave mode DMA or core write results in the wrong data being read by the host or master SHARC. For example, if the packing mode is 100 (32/64 internal to 32 external), the failure may look like:

```
Expected data: 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, ...
Actual data:   0x10, 0x30, 0x30, 0x50, 0x50, ...
```

**WORKAROUND:**
There are 2 possible workarounds for this issue:
1. Program 64-bit internal to 64-bit external packing (no pack) mode on the internal side of the slave transmit EPBx.
2. Make sure that master mode DMA is not enabled when host or master SHARC reads the slave SHARC's EPBx. This can be accomplished by implementing the following routine before starting a master mode DMA:
a. The DSP asserts the bus lock (bit set mode2 BUSLK;).
b. The DSP waits for mastership of the bus (waitbm: nop; IF NOT BM JUMP waitbm;).
c. The DSP starts master DMA. Note that at this time the bus is locked and owned,thus no other device gets the ownership of the bus.
d. After the master DMA is completed, disable the DMA and release the bus lock (for example this can be done in master DMA's interrupt service routine).

**APPLIES TO REVISION(S):**
0.0, 0.1, 1.1, 1.2

## 66. 02000069 - 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, 1.1, 1.2