ANALOG DEVICES

SHARC® Embedded Processor

Silicon Anomaly List

ADSP-21161N

ABOUT ADSP-21161N SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC ADSP-21161N product(s) and the functionality specified in the ADSP-21161N 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]
1.3	0110010
1.2	0110010
1.1	0101010
1.0	0101010
0.3	0100010

Notes:
The revision 0.3 silicon is Engineering Grade and has only been tested with an internal VDD greater than 1.95 to 2.00 volts for 100 MHz operation at 25 degrees C.

Revision 1.3 silicon improves manufacturability and reliability of the ADSP-21161N. There is NO change in the performance, functionality, or product specification between revision 1.2 and 1.3.

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	P	A	Added anomalies: 04000068,Added common note on Tools action for the anomalies 04000046 and 04000063.
05/10/2007	O	A	Document Format Update.
12/28/2006	N	A	Removed anomalies: 04000066. This information is added to the HRM.
11/27/2006	M	A	Added anomalies: 04000066
04/19/2006	L	A	Added anomalies: 04000065
02/16/2006	K	A	Added anomalies: 04000064,Added information for Rev 1.3 Silicon
08/12/2005	J	A	Added anomalies: 04000062 and 04000063,Modified anomalies: 04000052
11/25/2004	I	A	Added anomalies: 04000057,04000058,04000059,04000060 and 04000061

NR002446P

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-21161N anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.3	1.0	1.1	1.2	1.3
1	04000001	The PLL does not reliably lock on power-up	x	x	x	.	.
2	04000002	CCLK-to-CLKIN ratios 3:1, 4:1($\overline{\text{CLKDBL}}$=1), 6:1 and 8:1 do not function	x	.	.	.	.
3	04000003	Minimum High Level Input Voltage (VIH) must be greater than 2.3 Volts	x	.	.	.	.
4	04000004	TTL Input Buffer for all inputs can fail to switch from high to low transitions for core power supply VDDINT < 1.9 Volts	x	.	.	.	.
5	04000005	FS0, FS1, FS2, FS3, $\overline{\text{RD}}$, and SDCLK0 ESD protection feedback paths are missing impedance-matching resisters	x	.	.	.	.
6	04000006	SDRAM reads fail for SDBUF = 1, SDCKR = 0, and tRP = tRAS = 1	x	x	x	x	x
7	04000007	Simultaneous EPBx DMAs can corrupt EP DMA data	x	.	.	.	.
8	04000008	Handshake Mode DMA and External Handshake Mode DMA are not functional	x	.	.	.	.
9	04000009	External Port DMA handshake modes limited in throughput	x	x	x	x	x
10	04000010	Simultaneous DMA/CORE access of EPBx	x	.	.	.	.
11	04000011	$\overline{\text{SBTS}}$ deassertion can cause a resumed external memory access to abort	x	.	.	.	.
12	04000012	Host asynchronous access overlapping DSP to DSP transfer fail	x	x	x	.	.
13	04000013	Host writes fail for async host write cycles not aligned to rising edge of CLKIN	x	x	.	.	.
14	04000014	Link port data corruption may occur when a particular alignment of LxCLK and CCLK occurs at 1:1 LCLK/CCLK ratio	x	.	.	.	.
15	04000015	SPORT transfers fail at frequencies above 40 MHz	x	.	.	.	.
16	04000016	SPORT companding fails for first active transmit channel in multichannel frame at core voltages (VDDINT) > 2.0 volts	x	x	x	x	x
17	04000017	Broadcast mode (BDCSTx=1 in MODE1 register) works improperly when source buffer is located in external memory	x	.	.	.	.
18	04000018	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	x	.	.	.	.
19	04000019	BMAX/BCNT counter expire corrupts synchronous burst transfers in progress during a BTC (bus transition cycle)	x	x	x	.	.
20	04000020	SPI transfers do not operate reliably at higher SPICLK frequencies greater than 12.5 MHz	x	x	x	x	x
21	04000021	Shadow Write FIFO may erroneously return invalid memory read data when a internal memory write is followed by a read of the same address during simultaneous core/DMA access to the same internal memory block	x	.	.	.	.
22	04000022	SIMD read from internal memory with Shadow Write FIFO hit does not function correctly	x	x	x	x	x
23	04000023	SPORTS exhibit random transmit bit failures at VDDINT = 1.95 volts	x	.	.	.	.
24	04000024	MMS accesses fail if destination is the DSP's own IOP register	x	x	x	x	x
25	04000025	Link port buffers and link buffer status bits do not flush properly upon power-up and subsequent resets affecting initial link port data transmission	x	x	x	.	.
26	04000026	IMASKP register bits are left-shifted by 1 bit when modifying bits 14 to 30	x	x	x	x	x
27	04000027	Non-SDRAM writes to other banks fail before an SDRAM power-up sequence when using the SDRAM buffering option [SDBUF = 1]	.	x	x	x	x
28	04000028	32-bit host accesses fail when IPACK[1:0] is set to 0x1 for 48-bit external memory instruction execution	x	x	x	.	.
29	04000029	Inactive EPBx DMA channel parameter registers (EIEPx, EMEPx, ECEPx) may be read incorrectly	x	x	x	x	x
30	04000030	Direct writes to IMASKP or LIRPTL may cause highest priority interrupt to be serviced twice	x	x	x	x	x
31	04000031	Larger than expected minimum SDRAM data input hold time tHDSDK specification	.	.	x	x	x

No.	ID	Description	0.3	1.0	1.1	1.2	1.3
32	04000032	Asserting $\overline{SBTS}$ during a deadlock situation may not release the host bus in the expected next cycle	x	x	x	x	x
33	04000033	External port synchronous accesses aborted with $\overline{SBTS}$ access was may resume as asynchronous accesses	x	x	x	x	x
34	04000034	New receive RXSR shift register SPI data in master mode operation can overwrite previously received data or be ignored during a full SPIRX buffer status	x	x	x	.	.
35	04000035	Core writes to the SPITX register do not update the buffer status bits in SPISTAT properly	x	x	x	.	.
36	04000036	$\overline{MSx}$ lines not connected to SDRAM devices can be driven low for 1 CCLK cycle by new bus master [with ID2:0 greater than 1] accessing SDRAM after BTC for the 1st access after $\overline{RESET}$	x	x	x	.	.
37	04000037	BMSTR and TIMEXP pins cannot be disabled through boundary scan register	x	x	x	.	.
38	04000038	Asynchronous Host Reads can fail if tSADRDL is less than one external clock cycle	x	x	x	.	.
39	04000039	32 bit Byte wide link port DMA transfers running at a 1:1 link clock to core clock ratio will transfer at less than full speed	x	x	x	x	x
40	04000040	Conditional type 10 instruction may fail in SIMD	x	x	x	x	x
41	04000041	Broadcast load fails in type 10 instruction	x	x	x	x	x
42	04000042	A non-SDRAM write that follows an SDRAM read can be corrupted	x	x	x	x	x
43	04000043	Illegal DAG stalls can occur under certain circumstances	x	x	x	x	x
44	04000044	Rn=MANT Fx results will be rounded if RND32 is enabled	x	x	x	x	x
45	04000045	Conditional instructions using DAG1 fail if executed from external SDRAM	x	x	x	x	x
46	04000046	Execution of instructions that modify interrupt latch registers may cause incoming interrupts to be ignored	x	x	x	x	x
47	04000047	SPI slave transfer fails when preceded by a master transfer	x	x	x	x	x
48	04000048	A breakpoint following a JUMP, CALL, RTI or RTS can trigger an early/improper emulator break	x	x	x	x	x
49	04000049	DAG register write followed directly by a DAG register read fails during context switch	x	x	x	x	x
50	04000050	System registers written with a PM access do not have a 1 cycle effect latency	x	x	x	x	x
51	04000051	Conditional RTI fails in SIMD mode	x	x	x	x	x
52	04000052	Single instruction loops can terminate early	x	x	x	x	x
53	04000053	Bit reversal fails with indirect jump	x	x	x	x	x
54	04000054	VIPD bit gets cleared when branching to ISR	x	x	x	x	x
55	04000055	Link Port DMA failures occur when SPI is enabled in core mode	x	x	x	x	x
56	04000056	SPORTs used with an external Serial Clk may lockup on start-up in I^2S mode of operation	x	x	x	x	x
57	04000057	DSP will not respond to host with $\overline{HBG}$ when SDRAM is configured for self refresh mode	x	x	x	x	x
58	04000058	MU (Multiplier underflow) flag gets set anomalously for some non-underflow cases	x	x	x	x	x
59	04000059	Data shift problem on disabling and re-initializing SPI slave in CPHASE=0 mode	x	x	x	.	.
60	04000060	Top of the loop address stack is filled with zero when PUSH LOOP instruction is executed	x	x	x	x	x
61	04000061	Data corruption on the MISO signal during the SPI slave mode if the OPD bit is set	x	x	x	x	x
62	04000062	Serial port can miss a word in transmit mode if that word is written in the TX buffer	x	x	x	x	x
63	04000063	DAG stall causes external port to malfunction	x	x	x	x	x
64	04000064	Read of slave DMA's EPBx by host or master DSP in a multiprocessing may return wrong data	x	x	x	x	x
65	04000065	Simultaneous DMA and CORE writes to EPBx of slave DSP can corrupt the core written data in a multiprocessing	x	x	x	x	x

No.	ID	Description	0.3	1.0	1.1	1.2	1.3
66	04000068	Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers	x	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-21161N including a description, workaround, and identification of applicable silicon revisions.

1. 04000001 - The PLL does not reliably lock on power-up:

DESCRIPTION:

The PLL is susceptible to fail to lock to the CLKIN frequency during a power-up reset due to the following two start-up conditions:
1. External/Internal clock oscillator startup settling time During system power up while the external oscillator (or external crystal used in conjunction with the internal oscillator) begins to oscillate, the PLL can fail to lock to the desired CLKIN frequency if the core frequency goes below 33 MHz for more than 20 microseconds (CLKIN frequency will vary based on PLL ratio and CLKDBL configuration).
2. VDDINT ramp rate (from 0 to 1.8 volts) exceeding 2 milliseconds. The power-supply must ramp up from 0 to 1.8 volts within this 2 millisecond time period if the CLKIN frequency is present before or during the ramping of the VDDINT and VDDEXT supplies.

How to recognize this problem exists:
The following is a list of some (but not necessarily all) known symptoms of this anomaly:
1. One possible symptom is that CLKOUT is not running.
2. For revision 0.3, another possible symptom is that CLKOUT runs but the DSP fails to function properly. It has been observed that booting does not complete successfully (no HBG after HBR, failure to attempt to read from EPROM by driving BMS and RD, and link port or SPI port fails to accept data).
3. For revision 1.0 and 1.1, another symptom is that CLKOUT and CLKIN are running, but these signals are 270 degrees out of phase with SDCLK0 (3/4 of a clock cycle). SDCLK0 is functional after reset and it operates at 1/2 CCLK frequency by default. If the PLL is reset reliably you would normally observe SDCLK0 180 degrees out of phase with CLKIN/CLKOUT.

WORKAROUND:

For silicon revision 0.3:
Perform all of the following steps to reliably reset on power-up:
1. With RESET asserted, apply power to the DSP.
2. If possible, ensure that VDDINT ramps up to 1.8 volts within 2 milliseconds.
3. Allow CLKIN to run for a minimum of 10 us.
4. Stop CLKIN (CLKIN can be held high or low but should not float) for a minimum of 10 us and no longer than 500 ms.
5. Restart CLKIN and allow to run for a minimum of 2000 cycles before de-asserting RESET.

For silicon revisions 1.0 and 1.1:
Perform all of the following steps to reliably reset on power-up:
1. Please follow the power-up sequencing specification provided in the datasheet.
2. If the 2 msec VDDINT ramp-up rate described in the datasheet specification cannot be met in your power-supply sub-system, then CLKIN must not be applied to the DSP until the VDDINT supply is fully ramped. Once VDDINT 1.8 volt supply level is achieved at the slower ramp-rate, the CLKIN source [which conforms to the minimum 33 MHz core frequency requirement described in issue (1) above] can then be applied to the DSP. Another reliable method for resetting the PLL is powering up VDDEXT first to allow the external clock oscillator to stabilize within 25 msec, then ramping VDDINT (within 2 msec).
3. Allow CLKIN to run for a minimum of 2000 cycles before de-asserting RESET.

Note: These workarounds will reliably reset the PLL on power-up under all conditions. (delaying, stopping and restarting the CLKIN signal can be manually controlled by the host processor or an FPGA if the CLKIN source is provided by these devices, or an external CLKIN delay or gate circuit if the clock source is derived from an external clock oscillator). Analog Devices has no specific circuit recommendations to work around this anomaly.

APPLIES TO REVISION(S):

0.3, 1.0, 1.1

2. 04000002 - CCLK-to-CLKIN ratios 3:1, 4:1($\overline{\text{CLKDBL}}$=1), 6:1 and 8:1 do not function:

DESCRIPTION:
The CCLK-to-CLKIN ratios of 3:1, 4:1($\overline{\text{CLKDBL}}$=1), 6:1 and 8:1 do not function properly.

WORKAROUND:
For revision 0.3, use only 2:1 ($\overline{\text{CLKDBL}}$=1) and 4:1 ($\overline{\text{CLKDBL}}$=0) CCLK-to-CLKIN ratios by configuring the $\overline{\text{CLKDBL}}$ and CLK_CFG[1:0] pins as follows:
2:1 = $\overline{\text{CLKDBL}}$=1, CLK_CFG[1:0]=00
4:1 = $\overline{\text{CLKDBL}}$=0, CLK_CFG[1:0]=00

APPLIES TO REVISION(S):
0.3

3. 04000003 - Minimum High Level Input Voltage (VIH) must be greater than 2.3 Volts:

DESCRIPTION:
The ADSP-21161N Data Sheet on Page 18 under "ADSP-21161N Specifications / Table 4: Recommended Operating Conditions" states that the VIH (High Level Input Voltage) is specified to have a minimum voltage as follows:
VIH1 = 2.0 Volts Minimum
VIH2 = 2.2 Volts Minimum
To guarantee correct operation of inputs for revision 0.x silicon, the minimum VIH must be greater than 2.3 Volts. This restriction does not include CLKIN.

WORKAROUND:
Ensure all input voltages from all peripherals and memory interfaced to the ADSP-21161 are greater than 2.3 volts.

APPLIES TO REVISION(S):
0.3

4. 04000004 - TTL Input Buffer for all inputs can fail to switch from high to low transitions for core power supply VDDINT < 1.9 Volts:

DESCRIPTION:
The trip point for high to low transitions in the TTL input buffer is sensitive at low temperatures and nominal core power supplied voltages for all input pins of the DSP (except CLKIN) and under these conditions the TTL input buffers can fail to transition unless the core voltage VDDINT is greater than 1.9 Volts.

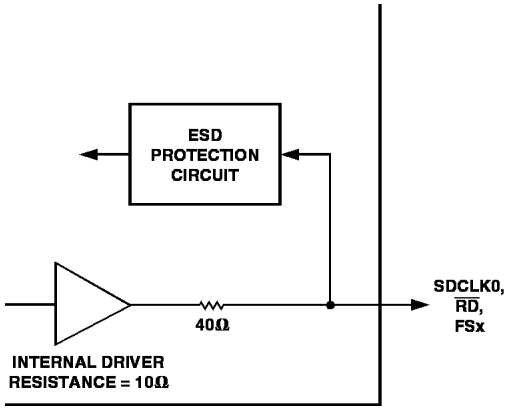
WORKAROUND:
To guarantee this condition does not exist for operating conditions at room temperature (25 degrees C), apply an internal core power supply voltage of VDDINT > or = 1.9 Volts to all VDDINT input pins.

APPLIES TO REVISION(S):
0.3

5. 04000005 - FS0, FS1, FS2, FS3, $\overline{RD}$, and SDCLK0 ESD protection feedback paths are missing impedance-matching resisters:

DESCRIPTION:

The FS0, FS1, FS2, FS3, $\overline{RD}$, and SDCLK0 pins include an internal ESD protection circuit feedback path which does not contain a serial terminating resister. Depending on the trace-length to other devices, serial transfers and memory accesses may fail because of transmission line reflections in the traces. The following diagram illustrates the internal driver implementation of these pins with the unterminated ESD protection feedback path:



Because of the unterminated feedback path, there is a potential for a failure in serial transfers to transmit incorrect data or latch received data correctly in the serial ports.

SDRAM reads may fail with SDCLK greater than 70 MHz depending on the proximity of the SDRAM devices to the DSP. If the SDRAM controller attempts to execute read commands to an external SDRAM at the full core clock rate, the read data returned by the SDRAM may be latched incorrectly or not latched at all for SDCLK frequencies greater than 70 MHz SDCLK operation. The controller's READ command functions correctly but the returned data is not captured by the controller when SDCLK > 70 MHz.

WORKAROUND:

General Work Around:
Ensure that traces are kept as short as possible, and if necessary, some load termination techniques may need to be implemented.

SDRAM Work Around 1:
If the SDCLK0 pin is not terminated via parallel termination techniques, and if your SDRAM is used for infrequent core accesses or background DMAs, set the SDRAM clock rate to 1/2 CCLK (core clock) by clearing the SDCKR bit (bit 21) in the SDCTL control register.

SDRAM Work Around 2:
If the SDCLK0 pin is not terminated via parallel termination techniques, and if your SDRAM is used for packed instruction code execution or requires high priority master-mode DMA accesses to SDRAM, the application program may execute more efficiently with CCLK < 70MHz with the SDRAM clock rate equal to the core clock rate (SDCLK=CCLK) by setting the SDCKR bit in SDCLK.

APPLIES TO REVISION(S):

0.3

6. 04000006 - SDRAM reads fail for SDBUF = 1, SDCKR = 0, and tRP = tRAS = 1:

DESCRIPTION:
Under expected SDRAM operation, if the refresh counter expires (SDRDIV=0) during a read, the read is first stopped by the SDRAM controller and the pre-refresh command sequence is normally applied by the controller.

However, when the SDRAM controller is configured with the SDCTL register under the following mode, a data conflict can occur for reads after the refresh counter expires.

1. buffering option enabled (SDBUF=1).
2. SDCLK-to-CCLK ratio is zero (SDCKR=0).
3. Precharge delay & CAS latency are both set to 1 SDCLK cycle (tRP = tRAS = 1).

This configuration sets the Bank Cycle Time tRC = tRAS + tRP = 2 SDCLK cycles.
With these configurations programmed in SDCTL, any valid data that is driven out of the SDRAM is still latched, but since the bank cycle time (tRC) is 2 cycles, the refresh is completed and the read from the next location continues. This continuing read conflicts with the previous data being read.

WORKAROUND:
If the SDBUF=1 register buffering option is a requirement to reduce capacitive loading, then consider increasing tRP or tRAS cycles in the SDCTL register to configure tRC to be 3 SDCLK cycles or greater. Note that by increasing tRP we only add a one cycle latency to the initial opening of a new row address. Otherwise, throughput is not affected for sequential reads to the same page after the first initial read.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

7. 04000007 - Simultaneous EPBx DMAs can corrupt EP DMA data:

DESCRIPTION:

Simultaneous external port DMA transfers using DMA channels 10, 11, 12 and 13, can cause EPBx DMA data to become corrupt in that it may have a DMA transfer cycle shift to the destination. This applies to both fixed or rotating priority EP DMA transfers.

The following scenario is used to demonstrate one possible symptom of the data corruption. Under this application, two EPBx DMAs are active and enabled simultaneously, i.e., both EPBx channels arbitrate to transmit 32-bit data from a source buffer in internal memory to a destination buffer residing in an external x32-bit SDRAM (PMODE=100, TRAN=1, DTYPE=0 in DMACx register). The data alignment expected for a 4 word transfer with no DMA packing should normally be:

Source1[4]	Source2[4]	Dest1[4]	Dest2[4]
0x11111111	0x55555555	0x11111111	0x55555555
0x22222222	0x66666666	0x22222222	0x66666666
0x33333333	0x77777777	0x33333333	0x77777777
0x44444444	0x88888888	0x44444444	0x88888888

However, simultaneous external DMA requests conflict, causing destination data to be offset. For example, a resulted transfer could look like:

Source1[4]	Source2[4]	Dest1[4]	Dest2[4]
0x11111111	0x55555555	0x44444444	0x77777777
0x22222222	0x66666666	0x11111111	0x88888888
0x33333333	0x77777777	0x22222222	0x55555555
0x44444444	0x88888888	0x33333333	0x66666666

Note that different DMA parameters and modes will show alternative destination offsets, and this is dependant on the fixed/rotating priority option, the packing mode used, and the duration (# of words) transferred. The probability for misalignment is greatly reduced for smaller packing modes (ex. 8-to-48 or 8-to-32 DMA data packing) and smaller DMA sequences, as the packing logic allows enough time for the other active DMA channel to arbitrate and be granted the DMA request from the external port to EPBx. However, the use of smaller width packing to increase the successful completion of simultaneous packed DMAs is not guaranteed under all conditions. The application program should therefore test the DMA channel activity to ensure that only 1 DMA sequence is active at a time.

Note that any single EPBx DMA (and EPBx DMA chaining) is still functional for all 4 EP channels if the DMAs are set up sequentially and DMA operation is not concurrent. This includes all DMA packing modes from internal-to-external or external-to-internal memory for master and slave mode DMA. Note that all other DMA channels (SPORTs, SPI, Link Ports) can all still execute DMA transfers simultaneously with at least 1 EPBx DMA channel and do not affect EPBx DMA operation.

WORKAROUND:

Poll the DMA10ST (bit 10), DMA11ST(bit 11), DMA12ST(bit 12), or DMA13ST(bit 13) DMA EPBx DMA status bits in the DMASTAT register to ensure previous external port DMA sequence is complete, or wait until a EPBx DMA interrupt is latched and serviced before attempting to start a new external port single DMA or a new chained DMA sequence on another external port channel.

Since DMA chaining cannot be done across DMA channels, the DMAxCHST chaining status bits must also be polled along with the DMAxST channel status bits to ensure a DMA sequence for a given EPBx channel is complete, if the new DMA is not programmed using the interrupt service routine DMA setup method.

APPLIES TO REVISION(S):

0.3

8. 04000008 - Handshake Mode DMA and External Handshake Mode DMA are not functional:

DESCRIPTION:
DMA handshake modes (external handshake and handshake) using the $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ pins are not functional. Incorrect data is driven or latched during these accesses.

WORKAROUND:
Do not use DMA handshaking modes using the $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ signals.

APPLIES TO REVISION(S):
0.3

9. 04000009 - External Port DMA handshake modes limited in throughput:

DESCRIPTION:
DMA transfers should be supported at the full system CLKIN/CLKOUT rates of 50 MHz. However, full bandwidth at 2:1 core clock (CCLK) to CLKIN/CLKOUT ratio is not possible.

Updated simulation 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.

tWDR = tCCLK + 4.5 nsec min
tDMARH = tCCLK + 4.5 nsec min

The sampling rate of $\overline{\text{DMARx}}$ signal by the internal circuitry of the ADSP-21161 prohibits maximum throughput at a CCLK-to-CLKOUT ratio of 2:1. Maximum asynchronous throughput using $\overline{\text{DMARx}}$ and $\overline{\text{DMAGx}}$ handshaking equals:
tWDR + tDMARH = (tCCLK + 4.5) + (tCCLK + 4.5) = 14.5 ns + 14.5 ns = 29 ns, or 34.5 MHz.

CCLK to CLKIN ($\overline{\text{CLKDBL}}$=1) ratios of 3:1, 4:1 will support full speed throughput at the CLKIN frequency. CCLK to CLKIN ($\overline{\text{CLKDBL}}$=0) ratios of 4:1, 6:1, 8:1 will support full speed throughput at the CLKIN frequency.

WORKAROUND:
If the maximum $\overline{\text{DMARx}}$/$\overline{\text{DMAGx}}$ throughput limit of 50 MHz (CLKIN/CLKOUT maximum frequency) is required, synchronize the assertions/deassertions of $\overline{\text{DMARx}}$ with respect to CLKOUT. See ADSP-21161N data sheet for timing details.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

10. 04000010 - 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-21161 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-21161.

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):
0.3

11. 04000011 - $\overline{\text{SBTS}}$ deassertion can cause a resumed external memory access to abort:

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

WORKAROUND:
Do not assert $\overline{\text{SBTS}}$ 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:
1. The host can use a programmable i/o pin to assert an $\overline{\text{IRQx}}$ pin which would cause the DSP to vector to internal memory to the $\overline{\text{IRQx}}$ interrupt vector address, which then causes the DSP to halt external accesses, allowing the host to gain control of the system.
2. 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.
3. 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):
0.3

12. 04000012 - Host asynchronous access overlapping DSP to DSP transfer fail:

DESCRIPTION:
Assertion of $\overline{\text{CS}}$ 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 $\overline{\text{CS}}$ 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 $\overline{\text{HBR}}$ and $\overline{\text{CS}}$ (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 $\overline{\text{CS}}$ with the assertion of $\overline{\text{HBG}}$. In other words, do not assert $\overline{\text{CS}}$ until after $\overline{\text{HBG}}$ is asserted.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

13. 04000013 - Host writes fail for async host write cycles not aligned to rising edge of CLKIN:

DESCRIPTION:

Host asynchronous writes fail (either by the DSP not responding with a returned $\overline{HBG}$ in response to $\overline{HBR}$, or the write data fails to be latched properly) if the rising edge of the $\overline{WR}$ and $\overline{HBR}$ strobes are relatively coincident with the rising edge of the internal core clock. The write data may be written to the wrong address. The write data transfer can also fail to complete as $\overline{HBG}$ might not be driven low.

WORKAROUND:

Only deassert the $\overline{WR}$ and $\overline{HBR}$ signal in the following window with respect to rising edge of CLKIN:

CCLK-to-CLKIN ratio	Allowed range for $\overline{WR}$_ rising edge after any 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]
6:1	[(n(tCK/6)+tCK/12)- Delta1 to (n(tCK/6)+tCK/12) + Delta2]
8:1	[(n(tCK/8)+tCK/16)- Delta1 to (n(tCK/8)+tCK/16) + 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 not yet been fully verified over frequency, temperature and voltage.

APPLIES TO REVISION(S):

0.3, 1.0

14. 04000014 - Link port data corruption may occur when a particular alignment of LxCLK and CCLK occurs at 1:1 LCLK/CCLK ratio:

DESCRIPTION:
Data corruption on the link port receiver occurs under particular alignment of Link Clock(LxCLK) to Core Clock (CCLK) 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-21161N 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.

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.
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 CCLK: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):
0.3

15. 04000015 - SPORT transfers fail at frequencies above 40 MHz:

DESCRIPTION:
SPORT0, SPORT1, SPORT2 and SPORT3 fail to transfer and receive (latch) data correctly for SCLKx frequencies above 40 MHz.

WORKAROUND:
Do not exceed 40 MHz SCLK frequencies to ensure proper SPORT operation.

APPLIES TO REVISION(S):
0.3

16. 04000016 - SPORT companding fails for first active transmit channel in multichannel frame at core voltages (VDDINT) > 2.0 volts:

DESCRIPTION:
(Note that this anomaly is not applicable in revision 0.x silicon operating at recommended operating core voltages of 1.9 to 1.95 volts) 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, regardless of if the enabled channel is channel 0 or a higher channel N. If the multichannel compand select registers are set to enable a higher channel with prior slots in the TDM frame disabled, then channel N will not transmit companded data.

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 1st transmitted frame. If, however, slots 0-4 are disabled and slots 5-31 are enabled, then slot 5 data will not be compessed after the 1st transmitted frame.

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

WORKAROUND:
Work Around #1:
Do not use SPORT hardware companding (compression) on SPORT2 or SPORT3 in multichannel mode for the first transmit channel when multichannel operation is used. This anomaly does not apply for SPORT0 and SPORT1, which are fixed to expand compressed rx data in multichannel mode.

Work Around #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). A PDF version of the chapter is available for download at the following URL: http://www.analog.com/publications/documentation/Using_ADSP-2100_Vol1/Chapter_11.pdf.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

17. 04000017 - Broadcast mode (BDCSTx=1 in MODE1 register) 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 32-bit normal word external memory space (DATA[47:16]), 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 32-bit wide (normal word) external memory.

APPLIES TO REVISION(S):
0.3

18. 04000018 - 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:

DESCRIPTION:
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]
```

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-21161:
Example of correct operation:

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

Unlike the ADSP-21160, the use of the LW option for 64-bit accesses to external memory are not supported since the ADSP-21161 external data bus is 32-bits wide.

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 memory are even address aligned for consistent and correct functionality.

APPLIES TO REVISION(S):
0.3

19. 04000019 - 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:
Work around 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.

Work around 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.

Work around 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):
0.3, 1.0, 1.1

20. 04000020 - SPI transfers do not operate reliably at higher SPICLK frequencies greater than 12.5 MHz:

DESCRIPTION:
There are 2 issues which prevent the SPI interface from running at it's maximum 25 MHz frequency:
1. In the 21161 SPI timings, at faster SPI baud rates, an ADSP-21161 SPI slave's setup and hold timings may not meet (an ADSP-21161 or other device's) SPI master data setup requirements. SPI master/slave configurations will not function reliably for BAUDR frequencies = 0 (25 MHz SPICLK @100 MHz CCLK - the SPICLK frequency is configurable in the SPICTL register by programming the BAUDR bits). SPI characterization results reveal a potential 10 nsec synchronization delay in the ADSP-21161 Slave's SPI data driving data out of MISO, which violates the ADSP-21161 master SPI's data input setup time on MISO with respect to SPICLK edge. [Refer to the timing specifications tSSPIDM (master), tDDSPIDS (slave) and tHDSPIDS (slave)].

2. At BAUDR = 0, SPI slave accesses can fail when the SPICLK aligns to the internal CCLK frequency. If the SPICLK edge aligns with the CCLK edge the SPI will drive incorrect data on MISO and read incorrect data on MOSI and the SPICLK duty cycle is not exactly a 50:50 duty cycle.

WORKAROUND:
Do not use BAUDR=0 in SPICTL to generate a 25 MHz SPICLK as a master device. To guarantee proper operation, the SPICTL register BAUDR bits must be programmed to a 1 or greater to generate SPICLK frequencies less than 12.5 MHz. If the DSP is a SPI slave, then check the master SPI device's data input setup timing requirements to verify that the SPI host can tolerate the ADSP-21161 slave SPI port's data setup and hold timings at faster baud rates. For long traces to a host SPI device, do not exceed an operating SPICLK frequency of 12.5 MHz in order to ensure reliable SPI operation.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

21. 04000021 - Shadow Write FIFO may erroneously return invalid memory read data when a internal memory write is followed by a read of the same address during simultaneous core/DMA access to the same internal memory block:

DESCRIPTION:
This anomaly has been identified in the shadow write FIFOs that exist between the internal memory array of the ADSP-21161 (revision <=0.3) and core & IOP busses that access the memory. (See pg 5-23 of ADSP-21161 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.

For information on tools that can assist in finding this anomaly in your code for revision 0.3 silicon, please examine the software development tools patch and documentation which is posted on our FTP site at the following location: ftp://ftp.analog.com/pub/dsp/2116x/anomalies/Shadow_Write_FIFO_Anomaly/

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
*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.
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.

APPLIES TO REVISION(S):

0.3

22. 04000022 - 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-21161 and core & IOP busses that access the memory.(See pg 5-23 of ADSP-21161 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 in SIMD mode will cause the 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 if the data is in the shadow write FIFO. For example, a SIMD mode explicit access to Normal-Word address 0x40001 will result in an implicit access to Normal-Word address 0x40000 if the reading of the data from 0x40001 is attempted while the data is still in the shadow write FIFO (from a previous memory write instruction).

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 operation when the data is not in the Shadow Write FIFO.
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

To better demonstrate when the failing access occurs, the following shows the failing cases for a SIMD shadow aligned/non-aligned access when a SIMD read immediately follows a SIMD write:

Address of Write Data in Sh Wr FIFO	Immediate Read after WRITE	Result	Resultant register addr contents
1. 0x50001	r0=dm(0x50000) r0=dm(0x50001) r0=dm(0x50002)	Fail Pass Fail	r0=(0x50002) , s0=(0x50001) r0=(0x50001) , s0=(0x50002) r0=(0x50002)*, s0=(0x50003)
2. 0x50002	r0=dm(0x50001) r0=dm(0x50002) r0=dm(0x50003)	Fail Pass Fail	r0=(0x50001) , s0=(0x50002)* r0=(0x50002) , s0=(0x50003) r0=(0x50003) , s0=(0x50002)
3. 0xA0002	r0=dm(0xA0000) r0=dm(0xA0001) r0=dm(0xA0002) r0=dm(0xA0003) r0=dm(0xA0004)	Fail Pass Pass Pass Fail	r0=(0xA0004) , s0=(0xA0002) r0=(0xA0001) , s0=(0xA0003) r0=(0xA0002) , s0=(0xA0004) r0=(0xA0003) , s0=(0xA0005) r0=(0xA0004)*, s0=(0xA0006)

Address of Write Data in Sh Wr FIFO	Immediate Read after WRITE	Result	Resultant register addr contents
4. 0xA0003	r0=dm(0xA0001)	Fail	r0=(0xA0005) , s0=(0xA0003)
	r0=dm(0xA0002)	Pass	r0=(0xA0002) , s0=(0xA0004)
	r0=dm(0xA0003)	Pass	r0=(0xA0003) , s0=(0xA0005)
	r0=dm(0xA0004)	Pass	r0=(0xA0004) , s0=(0xA0006)
	r0=dm(0xA0005)	Fail	r0=(0xA0005)*, s0=(0xA0007)
5. 0xA0004	r0=dm(0xA0002)	Fail	r0=(0xA0002) , s0=(0xA0004)*
	r0=dm(0xA0003)	Pass	r0=(0xA0003) , s0=(0xA0005)
	r0=dm(0xA0004)	Pass	r0=(0xA0004) , s0=(0xA0006)
	r0=dm(0xA0005)	Pass	r0=(0xA0005) , s0=(0xA0007)
	r0=dm(0xA0006)	Fail	r0=(0xA0006) , s0=(0xA0004)
6. 0xA0005	r0=dm(0xA0003)	Fail	r0=(0xA0003) , s0=(0xA0005)*
	r0=dm(0xA0004)	Pass	r0=(0xA0004) , s0=(0xA0006)
	r0=dm(0xA0005)	Pass	r0=(0xA0005) , s0=(0xA0007)
	r0=dm(0xA0006)	Pass	r0=(0xA0006) , s0=(0xA0008)
	r0=dm(0xA0007)	Fail	r0=(0xA0007) , s0=(0xA0005)
7. 0x28001	r0=dm(0x50000)	Pass	r0=(0x50000) , s0=(0x50001)
	r0=dm(0x50001)	Fail	r0=(0x50001) , s0=(0x50002)*
	r0=dm(0x50002)	Pass	r0=(0x50002) , s0=(0x50003)
	r0=dm(0x50003)	Fail	r0=(0x50003) , s0=(0x50002)
	r0=dm(0x50004)	Pass	r0=(0x50004) , s0=(0x50005)
8. 0x28001	r0=dm(0xA0001)	Pass	r0=(0xA0001) , s0=(0xA0003)
	r0=dm(0xA0002)	Fail	r0=(0xA0002) , s0=(0xA0004)*
	r0=dm(0xA0003)	Fail	r0=(0xA0003) , s0=(0xA0005)*
	r0=dm(0xA0004)	Pass	r0=(0xA0004) , s0=(0xA0006)
	r0=dm(0xA0005)	Pass	r0=(0xA0005) , s0=(0xA0007)
	r0=dm(0xA0006)	Fail	r0=(0xA0006) , s0=(0xA0004)
	r0=dm(0xA0007)	Fail	r0=(0xA0007) , s0=(0xA0005)
	r0=dm(0xA0008)	Pass	r0=(0xA0008) , s0=(0xA000A)
9. 0x50002	r0=dm(0x28000)	Pass	r0=(0x50000) , r1=(0x50001)
	r0=dm(0x28001)	Pass	r0=(0x50002) , r1=(0x50003)
	r0=dm(0x28002)	Pass	r0=(0x50004) , r1=(0x50005)
10. 0x50003	r0=dm(0x28000)	Pass	r0=(0x50000) , r1=(0x50001)
	r0=dm(0x28001)	Fail	r0=(0x50004) , r1=(0x50003)
	r0=dm(0x28002)	Fail	r0=(0x50004)*, r1=(0x50005)
	r0=dm(0x28003)	Pass	r0=(0x50006) , r1=(0x50007)
11. 0x50002	r0=dm(0xA0001)	Pass	r0=(0xA0001) , s0=(0xA0003)
	r0=dm(0xA0002)	Fail	r0=(0xA0002) , s0=(0xA0004)*
	r0=dm(0xA0003)	Fail	r0=(0xA0003) , s0=(0xA0005)*
	r0=dm(0xA0004)	Pass	r0=(0xA0004) , s0=(0xA0006)
	r0=dm(0xA0005)	Pass	r0=(0xA0005) , s0=(0xA0007)
	r0=dm(0xA0006)	Fail	r0=(0xA0006) , s0=(0xA0004)
	r0=dm(0xA0007)	Fail	r0=(0xA0007) , s0=(0xA0005)
	r0=dm(0xA0008)	Pass	r0=(0xA0008) , s0=(0xA000A)
12. 0xA0004	r0=dm(0x28000)	Pass	r0=(0x50000) , r1=(0x50001)
	r0=dm(0x28001)	Pass	r0=(0x50002) , r1=(0x50003)
	r0=dm(0x28002)	Pass	r0=(0x50004) , r1=(0x50005)
13. 0xA0006	r0=dm(0x28000)	Pass	r0=(0x50000) , r1=(0x50001)
	r0=dm(0x28001)	Fail	r0=(0x50002)@, r1=(0x50003)
	r0=dm(0x28002)	Fail	r0=(0x50004)*, r1=(0x50005)
	r0=dm(0x28003)	Pass	r0=(0x50006) , r1=(0x50007)
14. 0xA0004	r0=dm(0x50000)	Pass	r0=(0x50000) , s0=(0x50001)
	r0=dm(0x50001)	Fail	r0=(0x50001) , s0=(0x50002)*
	r0=dm(0x50002)	Pass	r0=(0x50002) , s0=(0x50003)
	r0=dm(0x50003)	Fail	r0=(0x50003) , s0=(0x50002)@
	r0=dm(0x50004)	Pass	r0=(0x50004) , s0=(0x50005)

Notes:
a '*' asterisk indicates old data from memory is accessed instead of new data in Shadow Write FIFO
Items 1 and 2 is for normal word accesses.
Items 3,4,5 and 6 is for short word access.
Items 7 to 15 are when mixing short, normal, and long word accesses for "SIMD" write following "SIMD" read.
'@' means, PEx and PEy data is partly from shadow and partly from memory.

Thus, for normal word accesses, observe that an access to an even address followed by an access to the adjacent (higher or lower) odd address will result in an access failure. Similarly an access to an odd address followed by an access to the adjacent (higher or lower) even address will result in an access failure.

For information on tools that can assist in finding this anomaly in your code, please examine the software development tools patch and documentation which is posted on our FTP site at the following location:
ftp://ftp.analog.com/pub/dsp/2116x/anomalies/Shadow_Write_FIFO_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. However, note that for program generated addresses (i.e. indirect addressing using the index/pointer DAG registers) which are odd cannot be handled with this ".ALIGN" workaround.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

23. 04000023 - SPORTS exhibit random transmit bit failures at VDDINT = 1.95 volts:

DESCRIPTION:
For revision 0.3 silicon, if the VDDINT core voltage is supplied 1.95 volts, the SPORTs can fail to transmit data properly. Note that this failure occurs only at VDDINT = 1.95 volts at room temperature. For example, if the SPORTs were operating in multichannel mode, it is possible for the data alignment to be off by 1 or 2 bits, or in some cases a complete timeslot.

WORKAROUND:
To prevent these random bit or timeslot alignment failures (if operating in multichannel mode), ensure that VDDINT = 2.0 volts. This problem does not exist in revision 1.x silicon.

APPLIES TO REVISION(S):
0.3

24. 04000024 - MMS accesses fail if destination is the DSP's own IOP register:

DESCRIPTION:
If the DSP attempts to access one of it's own IOP registers using it's corresponding multiprocessor memory space equivalent address, the data access will fail to write/read the data to/from the IOP register. Refer to the following documentation in the ADSP-21161 HW reference manual: "Multiprocessor Memory" on pages 5-19 to 5-21. This anomalous behavior does not operate according to paragraph 2 on page 5-21:

"(…) Instead of using it's own IOP register address range, a DSP can access it's IOP space through the corresponding address range in multiprocessor memory space. In this case, the DSP reads or writes to it's own IOP registers and does not make an access on the external system bus. Note that such self-accesses through multiprocessor memory space may only be accomplished with processor-core-generated addresses, not I/O processor-generated addresses."

For example, if a DSP who's multiprocessing ID pins ID2-0 = 001 attempts to execute the following instruction to message register MSGR0 (IOP address 0x8) using the MMS-equivalent address:

```
R0 = 0x12345678;
DM(0x100008) = R0;/*DSP ID1 writing to it's own MSGR0 register via MMS space */
```

The data specified in R0 is not written to MSGR0 using the ID1 equivalent address. Note, however, the following instructions will correctly write the data in R0 to MSGR0…

```
R0 = 0x12345678;
DM(0x8) = R0; /* DSP ID1 writing to it's own MSGR0 register via MMS space */
```

WORKAROUND:
Ensure that all read and write to a DSP's own IOP registers use the internal IOP address space address, which has the address range from 0x0000 0000 to 0x0000 01FF. Do not use the equivalent MMS address of the processor's own IOP registers, which is determined by the configuration of the multiprocessor ID2-0 pins.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

25. 04000025 - Link port buffers and link buffer status bits do not flush properly upon power-up and subsequent resets affecting initial link port data transmission:

DESCRIPTION:
The link port buffers (LBUFx) will not be flushed and the link buffer status bits may not be cleared properly upon a power-on RESET or subsequent RESET (either a "software" or "hardware" reset) if the link port buffer status is "not empty" at the time a RESET is applied to the DSP. "Not empty" status indicates that there is one or more words in the LBUFx. After the completion of the RESET the DSP would then contain a non-empty buffer status and the link port would recognized the non-empty status. Under these conditions transmission of unintended data values present in LBUFx prior to the RESET command 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 LCTL register after a software reset to check the buffer status is not reliable because these status bits also fail during this particular condition and show a clear status.
This "non-emtpy" buffer status can cause potentially cause problems after reset the transmit link port is again enabled without properly flushing the FIFOs. (For example, in a case of a link port boot, where a 256-word bootstrap occurs from a transmitting link port the receiving link port would receive 2 incorrect words, preventing reliable execution of the 256-word loader kernel). 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 which are configured as transmitters only, whereas unintended data could be transmitted due to a non-empty status and hence does not affect link ports configured as receivers.

WORKAROUND:
Note: These workarounds will have no adverse impact if status was already clear.

To prevent this from affecting your system, you can set up a small dummy (* internal loopback) transfer to clear the LBUFx FIFO status. Two methods of flushing the link buffer data are described below.

1. The execution of a simple core driven link port loopback transfer between the two link port buffers (LBUF0 and LBUF1) 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 the two link port buffers, the following set of steps can be implemented using internal loopback mode:
a. Enable one LBUFx as a transmitter and one LBUFx as a receiver without any DMA or Interrupt service routines (e.g., assign LBUF0 as tx and LBUF1 as rx).
b. Establish loopback mode transfer between them (i.e., assign one transmit buffer and one receive buffer to the same link port, either Link Port 0 or Link Port 1).
c. Write at least 3 dummy words to the transmit link buffer.
d. Give enough time for data transfer to complete. (**15 link clock cycles will be enough).
e. Read two words from the receiver link buffer.
f. After the internal loopback mode transfer completes, disable both link ports.
g. Reconfigure the selected link port as a transmitter in the mode required in your application code.

* Link port core or DMA loopback example code is available in the ADSP-21161 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 the 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.

2. Prior to enabling the link port transmitter, 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 (IILBx) 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):
0.3, 1.0, 1.1

26. 04000026 - IMASKP register bits are left-shifted by 1 bit when modifying bits 14 to 30:

DESCRIPTION:
When using the core to access the IMASKP register interrupts, writes to bits 14 to 30 will result in a left shift by 1 bit position to the left such that the next significant higher bit next to the intended bits will be set. For example, if attempting to write to IMASKP register bit 15, then IMASKP register bit 16 will be set instead.

```
R0 = 0x00008000;
IMASKP = R0; /* IMASKP contains the value of 0x00010000 */
```

WORKAROUND:
For enabling the upper bits 15 to 30 in IMASKP correctly, bit shifting the value in IMASKP[30:15] one position toward the lsb, then OR'ing the data with the lower 14 bits before writing the 32-bit data to IMASKP will result in the appropriate interrupt bits being set. The LPISUM bit 14 is a read only bit and cannot be modified, however setting bit 14 will result in bit 15 being modified. IMASKP bit 31 is a reserved bit and cannot be modified. Setting bits 0 to 13 does not result in a left shift by 1 bit and are not affected by this anomaly.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

27. 04000027 - Non-SDRAM writes to other banks fail before an SDRAM power-up sequence when using the SDRAM buffering option [SDBUF = 1]:

DESCRIPTION:
External memory non-SDRAM writes can fail whenever the SDRAM controller is configured for SDBUF=1 (register buffering) option and the power up sequence has not yet been completed by the SDRAM controller. Setting the SDPSS bit in the SDCTL register initiates the SDRAM power-up sequence. This failure occurs as a result of the MRS command is applied by the SDRAM controller conflicting with the non-SDRAM write access started by either the core or DMA controller.

WORKAROUND:
When the SDRAM controller is programmed with register buffer option enabled, do not perform non-SDRAM write accesses to external memory until the power-up sequence is completed by the SDRAM controller.

APPLIES TO REVISION(S):
1.0, 1.1, 1.2, 1.3

28. 04000028 - 32-bit host accesses fail when IPACK[1:0] is set to 0x1 for 48-bit external memory instruction execution:

DESCRIPTION:
When IPACK is set for 48-bit instruction execution in SYSCON and the host width is set up for 32-bit Host Bus Width, all host read/write accesses fail, regardless of the external memory type (SDRAM, SBSRAM or asynchronous SRAM). However, 32-bit host accesses with IPACK set to 16, 8, or 32-bit execution work properly. Also, 16-bit and 8-bit host accesses work properly during the execution of non-packed 48-bit instructions in external memory.

WORKAROUND:
1. If full-instruction execution is critical in your system and you require simultaneous host access, consider modifying your 32-bit host interface to act as 16-bit host, i.e., transferring 16-bit packed data on DATA16-DATA31. This will double the time required to complete host accesses.

2. If fast host throughput is critical in your system and you require simultaneous host access during external instruction execution, consider modifying the instruction packing mode to x32 instruction execution. This will cut external memory execution throughput in half.

3. If your application requires x48 instruction execution to execute code as fast as possible, and you can tolerate long host lockout periods, then you can lock out the host during the execution of external code (setting the BUSLK bit in the MODE2 register) and clear the bit when exiting external space. When exiting external full instruction width space, IPACK must be change to another value other than 0x1 before clearing the BUSLK bit. The host processor must be designed to handle potentially long lockout periods. If the host needs to interrupt external execution, then it can use an $\overline{\text{IRQx}}$ pin to force the DSP currently executing external code to vector to internal memory (however, most systems will not tolerate long DSP execution idle times). The $\overline{\text{IRQx}}$ interrupt service routine should first immediately change IPACK from 0x1 to another value as this would guarantee the instruction packing mode is changed before $\overline{\text{HBG}}$ is given to the host processor. In order for this to work, interrupt nesting must be on while the DSP is not currently servicing a higher priority interrupt.

4. If your application requires x48 instruction execution, but you are able to tolerate longer DSP lockout periods when attempting to execute code externally while the host accesses the external bus, then set up your host accesses such that it monitors the program counter by inspecting the PC_SHDW register via 16-bit host accesses. If the DSP is found to be executing code in internal memory, then before the host can fully take control of the host port with 32-bit accesses it should disable the x48 instruction execution mode in SYSCON using 16-bit accesses.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

29. 04000029 - Inactive EPBx DMA channel parameter registers (EIEPx, EMEPx, ECEPx) 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 EIEPx, EMEPx, ECEPx) 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 EIEPx, EMEPx, ECEPx DMA channel parameter registers should be considered when viewing the DMA addressing window in the debugger interface.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

30. 04000030 - 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/IMSKP, 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):

0.3, 1.0, 1.1, 1.2, 1.3

31. 04000031 - Larger than expected minimum SDRAM data input hold time tHDSDK specification:

DESCRIPTION:
The minimum hold time requirement for the specification tHDSDK is larger that the targeted minimum of 1.5ns defined in our datasheet. This may violate the data hold time provided by some SDRAMs. For example, some SDRAMs specify a minimum 2.5ns data out hold time switching characteristic with a 30pF load capacitance. Check your SDRAM manufacturer's datasheet to verify that the SDRAM device provides an appropriate data out hold time including the load capacitance.

```
For revision 1.1 silicon tHDSDK = 3.0ns.
For revision 1.1 silicon tHCADSDK = 2.5ns
For revision 1.2 silicon tHDSDK = 2.3ns.
For revision 1.2 silicon tHCADSDK = 2.0ns
For revision 1.3 silicon tHDSDK = 2.3ns.
For revision 1.3 silicon tHCADSDK = 2.0ns
```

WORKAROUND:
Delay the SDCLKx output signal from the DSP SDRAM interface. Assuming a worst case 2.5ns SDRAM data out hold time with a 30pF of load capacitance the delay for revision 1.1 silicon should be between 1.0 to 1.5ns (i.e., 1.0 nsec < SDCLKx delay < 1.5 nsec) With the same assumptions used in for revision 1.1 silicon, revision 1.2 silicon should be between 0.3 to 1.0ns (i.e., 0.3 nsec < SDCLKx delay < 1.0 nsec).

Delaying the SDRAM's clock will delay the output data by the timescale, thus resulting in a valid hold time requirement for DSP SDRAM reads. We recommend adding a capacitor on the trace between the DSP and SDRAM clock pins to delay the SDCLKx signal. Since delay characteristics are system-dependent, we have no specific recommendations on capacitor values or placement between the interconnection of the clock pins. However, dipped mica caps (silver mica) offer a good tolerance numbers of +/- 0.5pF (for capacitors less than 100pf). These are available with a temperature coefficient of less than 100ppm/deg C. NPO surface mount capacitors have excellent tolerance numbers near 0pF with temperature coefficients around 30 ppm/deg C.

We have not observed problems at room temperature (25 degrees C). The choice of capacitor value should account for timing variations at higher temperatures to still be within the SDCLKx delay window.

Note that when using this SDCLKx scheme, the SDRAM controller's register buffering mode of operation (enabled by setting SDBUF=1 in SDCTL register) cannot be used, because we cannot guarantee that tHCADSDK will be greater than 2.5 nsec when register buffering is enabled.

Also consider load capacitance factored in with the SDRAM data out hold time specification. Determine the load capacitance of the data bus to see if the capacitance adds additional delay in the timing to sufficiently meet the DSP's input hold time requirement (i.e., the addition of the SDRAM's output data hold time with the additional load capacitance delay.)

APPLIES TO REVISION(S):
1.1, 1.2, 1.3

32. 04000032 - Asserting $\overline{\text{SBTS}}$ during a deadlock situation may not release the host bus in the expected next cycle:

DESCRIPTION:
During a host deadlock situation where the core was trying to do an external access and the host asserts $\overline{\text{HBR}}$ in the same cycle, $\overline{\text{SBTS}}$ assertion may not break the properly. When a host processor uses the $\overline{\text{SBTS}}$ for deadlock resolution (Host is trying to access the DSP bus and DSP is trying to access host bus at the same time), the DSP bus master aborts the ongoing access, but $\overline{\text{HBG}}$ waits for ACK. The slave which was forced to abort the access may not give a high ACK as in some cases ACK is given only on the completion of access, so there can be a situation in which $\overline{\text{SBTS}}$ has aborted the access and since access was not complete so ACK is low. $\overline{\text{HBG}}$ depends on ACK so $\overline{\text{HBG}}$ will not be asserted and this may halt all the remaining activities preventing the host from accessing the bus for an indefinite period of time.

WORKAROUND:
1. To clear up a deadlock you can assert a false ACK to the DSP after $\overline{\text{SBTS}}$ is asserted. The ADSP-21161 will think that the external port DSP transaction actually completed when it sees the ACK driven and will allow the host to get control of the bus. If $\overline{\text{SBTS}}$ is asserted in cycle 0 ($\overline{\text{HBR}}$ already asserted), a fake ACK asserted prior to cycle 1 will be interpreted as completion of the access. A fake ACK can also be asserted in cycle 1 or later. $\overline{\text{HBG}}$ will be granted in cycle n+2 if a fake ACK is asserted in cycle n. ACK can be asserted in this situation without any detrimental effect to the DSP. Be cautious that when the DSP resumes the aborted word will not be transmitted.

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

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

33. 04000033 - External port synchronous accesses aborted with $\overline{\text{SBTS}}$ access was may resume as asynchronous accesses:

DESCRIPTION:
When $\overline{\text{SBTS}}$ (and $\overline{\text{HBR}}$) aborts any access, after host access completion the aborted access should start like a refresh access (wait count should be reloaded, $\overline{\text{MS}}$x line and $\overline{\text{RD}}$/$\overline{\text{WR}}$ strobe should start from the beginning.) But in some of the cases, when aborted access restarts, the $\overline{\text{MS}}$x line behaves incorrectly.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

34. 04000034 - New receive RXSR shift register SPI data in master mode operation can overwrite previously received data or be ignored during a full SPIRX buffer status:

DESCRIPTION:
Under normal SPI data transfer operation during an active SPI data transfer, if the SPIRX receive buffer of the SPI becomes full, and the receive shift register (RXSR) also becomes full, the SPI will stall the serial peripheral interface clock (SPICLK) until a SPI data word has been taken out of the SPIRX receive buffer. During this time when the SPICLK signal is stalled, we expect the RXSR shift register to hold the received data until a data word has been moved out of the SPIRX receive buffer. Once the data has been moved out of the SPIRX receive buffer, the contents of the RXSR receive shift register can be written into the buffer and the SPICK can be sent to receive the next word.

However, what happens is that once the RXSR shift register becomes full, it will automatically send the word to the SPIRX buffer instead of remaining in the RXSR shift register until an empty SPIRX buffer location is available. Depending on the GM bit, this new SPI word either overwrites the receive buffer or is ignored. The shift register then gets flushed out for the next SPI transfer (so if valid data was still present and needed to be queued it would be lost at this point).

Thus, if the GM bit in SPICTL = 0, the word in the shift register is missed. If GM in SPICTL = 1, then the previous word queued in the receive buffer is overwritten. The error status RBSY is set during this time.This issue is seen for the following SPI configuration:

1. The SPI is the master device.
2. The data transfer is interrupt or DMA driven.

This behavior is not seen when the SPI is in the slave mode.

WORKAROUND:
For DMA transfers this will not be a problem since the I/O processor will ensure that the SPIRX FIFO is emptied well before the new RXSR data fills up in the next transfer. Higher SPORT priority DMA activity should still enough free IOP bus cycles to allow the SPI DMA read to internal memory to occur before the RXSR fills up with new data.

For core interrupt driven transfers, ensure that the SPIRX buffer status indicated in SPISTAT is partially full versus reading a full buffer status when processing new SPI data.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

35. 04000035 - Core writes to the SPITX register do not update the buffer status bits in SPISTAT properly:

DESCRIPTION:
Core writes to the SPITX can cause failures in the transmit FIFO status bits (TXS) defined in the SPISTAT register. This error condition can make core driven transfers difficult because FIFO status bits will not be read reliably. The problem will also cause enabled interrupts to behave incorrectly because interrupt generation is based on the status of the transmit FIFO.

Status bits behave properly for DMA-driven transfers.

WORKAROUND:
1. Since the TXS status bits are accurate for DMA transfers, set up single word DMA driven transfers instead to replace core interrupt drive driven transfers. This can be done by setting the SPI DMA parameter registers as follows: IISTX = single fixed internal memory address, IMSTX = 0, ICSTX = 1. Between each single word DMA transfer the CSTX register must be reset to one and the TDMAEN bit in SPICTL must be cleared and reset. For an example DMA sequence, refer to the example SPI DMA source in chapter 6 (I/O processor) under "SPI Port DMA section" in the ADSP-21161 SHARC DSP Hardware Reference Manual.
2. If the timer is not used in your system, you can use the timer ISR to write data out of SPITX, setting TCOUNT to the number of CCLK cycles which is greater in duration than the number of SPICLK cycles required to transmit an 8-, 16- or 32-bit SPI word.
3. Use an $\overline{\text{IRQx}}$ pin connected to the SPI device select (FLAGx) to give an indication when the SPI port has finished transmitting data. This will only work under Non-Seamless Operation (NSML = 1) and DCPH0 = 1 in SPICTL (to make the FLAGx device select pin toggle between transfers). When FLAGx is deactivated between transfers, an edge sensitive interrupt (through an inverter) will indicate when it is safe to write the next word to SPITX.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

36. 04000036 - $\overline{\text{MSx}}$ lines not connected to SDRAM devices can be driven low for 1 CCLK cycle by new bus master [with ID2:0 greater than 1] accessing SDRAM after BTC for the 1st access after $\overline{\text{RESET}}$:

DESCRIPTION:
In a shared SDRAM multiprocessing system, for the first Core Clock (CCLK) cycle after a DSP [with an IDx greater than 001] becomes bus master and attempts to access SDRAM for the first time after $\overline{\text{RESET}}$, then the new bus master will incorrectly drive any of the other non-SDRAM connected $\overline{\text{MSx}}$ lines low for the duration the CCLK cycle. This does not occur on subsequent bus masterships.

WORKAROUND:
No workaround required. This should normally not cause problems in your system as the erroneously driven $\overline{\text{MSx}}$ signal will not cause a false read or write since $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are not activated.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

37. 04000037 - BMSTR and TIMEXP pins cannot be disabled through boundary scan register:

DESCRIPTION:
For board level JTAG scan testing, the BMSTR and TIMEXP [output-only pins] cannot be disabled/tristated via the JTAG scan path path. Normally all pins, even output-only pins, can be tristated.

WORKAROUND:
For revisions prior to 1.2, board-level JTAG testing should account for the inability to disable these 2 pins.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

38. 04000038 - 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 $\overline{\text{RD}}$ is asserted will resolve this problem. tSADRDL should me a minimum of one external clock cycle.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

39. 04000039 - 32 bit Byte wide link port DMA transfers running at a 1:1 link clock to core clock ratio will transfer at less than full speed:

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):

0.3, 1.0, 1.1, 1.2, 1.3

40. 04000040 - 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 0b) ;
R3 = R1 + R2, DM(I1, M7) = R5 ;
```

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

41. 04000041 - 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:
Use the same workaround for anomaly 04000040.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

42. 04000042 - A non-SDRAM write that follows an SDRAM read can be corrupted:

DESCRIPTION:
A non-SDRAM write that follows an SDRAM read will be corrupted if the following is true:
1. The DSP CLK_CFG pins are set up for a 2:1 core to clkin ratio.
2. SDCLK is ½ the core clock speed.
3. SDRAM buffering is enabled [SDBUF=1].
4. The non-SDRAM access is done with 0 wait states.

WORKAROUND:
If the non-SDRAM write is with the core, then placing 4 nop instructions between the SDRAM read and the non-SDRAM write will guarantee that the write will not be corrupted.

If DMA is used, the non-SDRAM writes must access memory with a minimum of 1 wait state. If 0 wait states are used, there is the risk that data will be corrupted.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

43. 04000043 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

44. 04000044 - 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;
NOP;
```

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

45. 04000045 - Conditional instructions using DAG1 fail if executed from external SDRAM:

DESCRIPTION:
When executing code from external SDRAM conditional instructions containing a DAG1 data access may not be performed correctly. The error condition can occur when the SDRAM is held off doing a refresh or instruction fetch when the conditional instruction containing the DAG1 access is about to be executed. The failure mode is a failure to honor the condition in the instruction, or data corruption in the DAG1 access. This issue affects all possible IPACK configurations. Code executed from internal memory or from non-SDRAM external memory is not affected.

WORKAROUND:
1. The conditional instruction can contain a DAG2 access instead of a DAG1 access.
For example the instruction:

```
if NE dm(i1,m1) =r0;
```

could be rewritten as:

```
if NE pm(i8,m8) =r0;
```

2. The conditional instruction can be broken up into multiple instructions.
For example the instruction:

```
if NE dm(i1,m1) =r0;
instA;
instB;
```

could be rewritten as:

```
if EQ jump no_write;
write: dm(i1,m1)=r0;
no_write: instA;
instB;
```

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

46. 04000046 - 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, SPIRI, SPITI, EPxI. The LPxI, SPIRI and SPITI 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 the 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.

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):
0.3, 1.0, 1.1, 1.2, 1.3

47. 04000047 - SPI slave transfer fails when preceded by a master transfer:

DESCRIPTION:
When the SPI port is in master mode there is an internal signal is driven high to enable the MOSI pin as an output. When the SPI port is reconfigured to be in slave mode this internal signal is not configured properly. As a result, the MOSI pin erroneously remains an output and continues to drive the last bit of data that was driven on MOSI when the SPI port was in master mode. This problem only occurs when the SPI port is first a master and then is reconfigured as a slave. No issue exists when the SPI port is configured as a slave and then changes to a master.

WORKAROUND:
1. Enable the DSP as an open drain output and transfer a dummy word to reconfigure the MOSI pin internally. Follow the steps below:
a. Enable the SPI as a master for your initial transfer using active drain mode (OPD = 0) and transfer the required data.
b. Disable the SPI port by clearing the SPICTL register.
c. Enable the SPI port as a master, this time with open drain enabled (OPD = 1).
d. Transfer 0xFF, 0xFFFF or 0xFFFFFFFF depending on the word size you have configured in the SPICTL register.
e. Disable the SPI port again by clearing the SPICTL register.

Once these steps have been completed, the SPI port can be successfully reconfigured in SPI slave mode.

2. If open drain mode is used for the main transfers, a dummy word can resolve the problem in a similar fashion as in workaround 1. Follow the steps below:
a. Enable the SPI as a master for your initial transfer using open drain mode (OPD = 1) and transfer the required data.
b. Reconfigure the SPI port to send out one more dummy word. Continue to use OPD =1 and the transfer the dummy word 0xFF, 0xFFFF or 0xFFFFFFFF depending on the word size you have configured in the SPICTL register.
c. Disable the SPI port by clearing the SPICTL register.

Once these steps have been completed, the SPI port can be successfully reconfigured in SPI slave mode.

3. Performing a software or hardware reset puts the SPI port in a state that will allow SPI slave transfers.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

48. 04000048 - 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:
1. Do not set a software breakpoint directly after a jump, call, rti or rts.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

49. 04000049 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

50. 04000050 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

51. 04000051 - 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:
1. 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):
0.3, 1.0, 1.1, 1.2, 1.3

52. 04000052 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

53. 04000053 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

54. 04000054 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

55. 04000055 - Link Port DMA failures occur when SPI is enabled in core mode:

DESCRIPTION:
When Link port DMA is used concurrently with the SPI peripheral on the ADSP-21161N, the Link Port data will be corrupted. Specifically, with Link Port DMA setup using DMA channels 8 & 9, and SPI enabled in core mode, if SPRINT is set, then Link Buffer 0 DMA data (DMA channel 8) will be corrupted. If SPTINT is set, then Link Buffer 1 DMA data (DMA channel 9) will be corrupted.

WORKAROUND:
Enable Link Ports in interrupt-driven (Core) mode, and use the SPI peripheral in DMA mode. Note that this will severely affect the DSP performance since the Link Ports can operate at speeds up to CCLK frequency with a data path width of up to 8 bits, causing very frequent interrupts.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

56. 04000056 - SPORTs used with an external Serial Clk may lockup on start-up in I^2S mode of operation:

DESCRIPTION:
In Serial Ports (SPORTs) used with an externally generated Serial Clock (SCLK) and enabled in I^2S mode (only), there is a possibility that the SPORT will lock up when it is enabled (as a transmitter or receiver). This can only occur if an edge of the external SCLK is generated coincident with the instruction cycle in which the SPORT is enabled. If this edge is aligned with the exact DSP core clock cycle in which the SPORT is enabled, the edge-detection circuitry within the SPORT will fail to recognize a rising or falling SCLK edge to initiate the shifting of data, resulting in a lock-up condition. Symptoms of this condition include the appearance of the SPORT data transmit pin stuck in a low state, while DMA activity halts after SPORT DMA chaining is activated.

WORKAROUND:
1. To ensure that the application does not provide this alignment of the externally generated SCLK with the SPORT Enable instruction, do not start the external SCLK until after the SPORT has been enabled.
2. If the above cannot be met in the system, then if and when the serial port lockup condition occurs, turn off the SPORTs and then turn them on again and ensure that there is no failure condition described above.
3. Use the SPORTs with an internally generated SCLK.

In summary, to guarentee correct operation of SPORTs using an external SCLK the following sequence is suggested while programming the SPORTS.
a. Ensure that externally generated SCLK is disabled.
b. Enable the SPORTs.
c. Enable DMA by writing to the DMA chain pointer registers. (Only if using the SPORT DMA).
d. After the SPORT peripheral and DMA channels are enabled (when DMA is used), activate the external serial clock source to the DSP.
e. Enable interrupts globally and SPORT interrupts to begin processing data.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

57. 04000057 - DSP will not respond to host with $\overline{\text{HBG}}$ when SDRAM is configured for self refresh mode:

DESCRIPTION:
DSP will not respond to host with $\overline{\text{HBG}}$ if SDRAM power up command and SDRAM self refresh commands are given at the same time. In other words, if the SDPSS bit and SDSRF bits are set in the SDRAM controller in the same cycle, the DSP will not respond to $\overline{\text{HBR}}$ by giving $\overline{\text{HBG}}$.

WORKAROUND:
For proper behavior, issue the power up command, wait for the power up sequence to complete, then issue other SDRAM commands. The power up sequence requires 100 SDRAM clock cycles to complete.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

58. 04000058 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

59. 04000059 - Data shift problem on disabling and re-initializing SPI slave in CPHASE=0 mode:

DESCRIPTION:
This problem occurs when the following sequence of operations are executed:

1. SPI slave, configured as a receiver, is setup for DMA transfer in CPHASE=0 mode.
2. Once the DMA transfer is complete, SPI slave is disabled by clearing the SPIEN bit of SPICTL control register.
3. DMA parameter registers are initialized for the next transfer.
4. SPI slave is enabled again by setting the SPIEN bit of SPICTL control register.
5. Steps 2 to 4 are repeated.

In the above situation, the first DMA transfer is always correct. Subsequent DMA transfers are corrupt. Data received would be shifted by (N-1) bits in Nth DMA transfer. Note that this problem happens in all modes where the device is SPI slave with CPHASE=0.

WORKAROUND:
Use CPHASE=1 mode of SPI slave if SPI has to be reconfigured.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1

60. 04000060 - 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):
0.3, 1.0, 1.1, 1.2, 1.3

61. 04000061 - Data corruption on the MISO signal during the SPI slave mode if the OPD bit is set:

DESCRIPTION:
The data driven on the MISO signal would be incorrect during the SPI slave mode if the OPD bit is set. This is due to the incorrect implementation of the OPD funcitonality on the MISO signal.

WORKAROUND:
Do not use SPI slave in the OPD mode.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

62. 04000062 - Serial port can miss a word in transmit mode if that word is written in the TX buffer:

DESCRIPTION:
When the code initially writes to the TX register and let the serial port transmit this word, wait for some time, now the tx-fifo status is empty. A write to the TX register during the last bits transmission causes the serial port transmitter to load the previous word again and miss this new word. When this write is at the SCLK's capture edge during the last bit transmission then only this problem comes. The root cause of the problem is half cycle difference between status change due to this write and then write to the transmit shifter from previous fifo stage. The data in previous fifo stage reaches half cycle later.

WORKAROUND:
This problem is seen when the serial port is used in core driven mode (core directly writes to the TX register using core's registers) and then wait for long time. If the serial port is used in interrupt driven mode or DMA mode then there will not be so much delay due to waiting and then this problem will not come. So preferably use interrupt driven mode or DMA mode of transfer to avoid this problem.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

63. 04000063 - 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-21161 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. This occurs only if the previous value in the associated index register was external-memory-address.

DAG stall code example:

```
i12 = external memory address; // instx
...
...
...
i12 = dm(m7,i6); // inst1
r0 = pm(i12,0); // inst2
```

Instruction instx has loaded the i12 register with external memory address. There can be any number of instructions between instx and inst1. Instruction inst1 loads internal memory address in i12 register, and it is used in the next instruction, so the sequencer stalls 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 external memory initially.	i12 = 0x4000000;//instx
		i12 = dm(m7,i6); // inst1
		r0 = pm(i12,0); // inst2
			Y	Y

2.	DAG Register initialized to internal memory initially.	i12 = 0x40000;//instx
		i12 = dm(m7,i6); // inst1
		r0 = pm(i12,0); // inst2
			N	N

3.	Wait register is initialized with zero wait states.	i12 = 0x4000000;//instx
		r0 = 0x1800000;
		dm(WAIT) = r0;
		i12 = dm(m7,i6); // inst1
		r0 = pm(i12,0); // inst2
			Y	Y

4.	Wait register is initialized with non-zero wait states.	i12 = 0x4000000;//instx
		r0 = 0x01CE739C;
		dm(WAIT) = r0;
		i12 = dm(m7,i6); // inst1
		r0 = pm(i12,0); // inst2
			N	Y

5	In the DAG Stall sequence, the DAG register is loaded and accessed under the given conditions:			

5.1.	dm indirect load from memory and dm access	i4 = 0x4000000;//instx
		i4 = dm(i7,m6);//inst1
		dm(i4,1) = r4;//inst2
			Y	N

5.2.	dm indirect load from memory and pm access	i9 = 0x4000000;//instx
		i9 = dm(i7,m6);//inst1
		pm(i9,1) = r4;//inst2
			Y	Y

S.No	Condition	Example	$\overline{\text{MSx}}$	$\overline{\text{BRx}}$

5.3.	pm indirect load from memory and dm access	i4 = 0x4000000;//instx
		i4 = pm(m9,i8);//inst1
		dm(i4,1) = r4;//inst2
			N	N

5.4.	pm indirect load from memory and pm access	i12 = 0x4000000;//instx
		i12 = pm(m9,i8);//inst1
		pm(i12,1) = r4;//inst2
			N	N

5.5.	dm direct load and dm access	i4 = 0x4000000;//instx
		i4 = 0x4322;//inst1
		dm(i4,1) = r4;//inst2
			Y	N

5.6.	pm direct load and pm access	i9 = 0x4000000;//instx
		i9 = 0x4322;//inst1
		pm(i91) = r4;//inst2
			Y	N

6.	When the code(the DAG Stall sequence with conditional instruction) is executed from external non-SDRAM memory.	i8 = 0x4000000;//instx
		m8=1;//inst1
		if ne pm(i8,m8)=r3; //inst2
		// condition should become false.
			Y	Not Applicable

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

0.3, 1.0, 1.1, 1.2, 1.3

64. 04000064 - Read of slave DMA'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 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 transmitting where the host or master SHARC is reading data from the slave SHARC's EPBx.
4. The packing mode of the slave DMA's EPBx, which is read by the host, is programmed such that the internal side is 32/64bits (packing modes 001, 100 and 110).

Under the above scenario, the Master mode DMA works correctly. However, the Slave mode DMA 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 or no-pack mode), 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 48 bits wide packing mode on the internal side of the slave transmit DMA's EPBx.
Any of the following packing modes can be used . . .

a. 16 external to 48 internal (PMODE=010).
b. 32 external to 48 internal (PMODE=011).
c. 8 external to 48 internal (PMODE=101).

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):
0.3, 1.0, 1.1, 1.2, 1.3

65. 04000065 - Simultaneous DMA and CORE writes to EPBx of slave DSP can corrupt the core written data in a multiprocessing:

DESCRIPTION:
This failure occurs under the following conditions:

1. On the slave DSP, the core write operation to the EPBx(which is read by host or master DSP) happen simultaneously with the external port DMA.
2. The DMA can be in any direction (transmit or receive), slave or master.
3. The packing mode can be of any type for both the DMA and the CORE write.

Under the above scenario, the DMA works correctly. However, the CORE write of EPBx results in the wrong data being read by the host or master SHARC. For example, if the core related EPBx's packing mode is 100 (32/64 internal to 32 external or no-pack mode) and the DMA is in master mode, the failure may look like:

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

For the 32/48 packing mode the fail patterns are different.

WORKAROUND:
There are two possible workarounds for this issue:
1. Use slave transmit DMA in place of the core EPBx write.
2. Make sure that the DMA is not enabled when the core is writing to the EPBx. This can be accomplished by implementing the following routine before starting the core access:
a. Check whether any of the DMA is enabled and active.
b. If the DMA is active, disable the DMA.
c. Write into the EPBx using core access.
d. Enable the DMA after the core write is complete.

APPLIES TO REVISION(S):
0.3, 1.0, 1.1, 1.2, 1.3

66. 04000068 - 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):
0.3, 1.0, 1.1, 1.2, 1.3