ANALOG DEVICES

SHARC® Embedded Processor

Silicon Anomaly List

ADSP-21160N

ABOUT ADSP-21160N SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC ADSP-21160N product(s) and the functionality specified in the ADSP-21160N 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]
0.1	0101100
0.0	0100100

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

NR002531L

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

No.	ID	Description	0.0	0.1
1	02000014	RFRAME instruction does not function correctly	x	x
2	02000044	IMASKP bits are left shifted by 1 bit for writes to bits 14 to 30	x	x
3	02000045	Inactive EPBx DMA channel parameter registers (EIx, EMx, ECx) may be read incorrectly	x	x
4	02000046	Direct writes to IMASKP or LIRPTL may cause highest priority interrupt to be serviced twice	x	x
5	02000047	Asynchronous Host Reads can fail if tSADRDL is less than one external clock cycle	x	.
6	02000048	32-bit wide link port transfers limited in throughput with 1:1 LCLK-to-CCLK ratio	x	x
7	02000049	Internal memory operation will not be possible while $\overline{\text{TRST}}$ is active (low)	x	.
8	02000050	Conditional type 10 instruction may fail in SIMD	x	x
9	02000051	Broadcast load fails in type 10 instruction	x	x
10	02000052	Rn=MANT Fx results will be rounded if RND32 is enabled	x	x
11	02000053	In Serial Port Multichannel mode, an external RFS one cycle early causes data corruption	x	.
12	02000054	Illegal DAG stalls can occur under certain circumstances	x	x
13	02000055	Execution of instructions that modify interrupt latch registers may cause incoming interrupts to be ignored	x	x
14	02000056	A breakpoint following a JUMP, CALL, RTI or RTS can trigger an early/improper emulator break	x	x
15	02000057	DAG register write followed directly by a DAG register read fails during context switch	x	x
16	02000058	System registers written with a PM access do not have a 1 cycle effect latency	x	x
17	02000059	Conditional RTI fails in SIMD mode	x	x
18	02000060	Single instruction loops can terminate early	x	x
19	02000061	Bit reversal fails with indirect jump	x	x
20	02000062	VIPD bit gets cleared when branching to ISR	x	x
21	02000063	MU (Multiplier underflow) flag gets set anomalously for some non-underflow cases	x	x
22	02000064	Top of the loop address stack is filled with zero when PUSH LOOP instruction is executed	x	x
23	02000065	DAG stall causes external port to malfunction	x	x
24	02000066	Read of slave DSP's EPBx by host or master DSP in a multiprocessing may return wrong data	x	x
25	02000069	Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers	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-21160N including a description, workaround, and identification of applicable silicon revisions.

1. 02000014 - RFRAME instruction does not function correctly:

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

2. 02000044 - IMASKP bits are left shifted by 1 bit for writes to bits 14 to 30:

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

The following example illustrates anomalous behavior:

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

3. 02000045 - Inactive EPBx DMA channel parameter registers (EIx, EMx, ECx) may be read incorrectly:

DESCRIPTION:

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

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

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

WORKAROUND:

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

APPLIES TO REVISION(S):

0.0, 0.1

4. 02000046 - Direct writes to IMASKP or LIRPTL may cause highest priority interrupt to be serviced twice:

DESCRIPTION:

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

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

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

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

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

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

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

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

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

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

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

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

WORKAROUND:

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

APPLIES TO REVISION(S):

0.0, 0.1

5. 02000047 - Asynchronous Host Reads can fail if tSADRDL is less than one external clock cycle:

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

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

APPLIES TO REVISION(S):
0.0

6. 02000048 - 32-bit wide link port transfers limited in throughput with 1:1 LCLK-to-CCLK ratio:

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

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

7. 02000049 - Internal memory operation will not be possible while TRST is active (low):

DESCRIPTION:
This prevents the DSP from booting properly as well as hangs functionality if applied to a DSP that has been successfully booted. Normally, the TRST signal on the JTAG port is recommended to be tied low to hold the Test Access Port (TAP) in reset, when the emulator is not connected. On boards designed with the ADI DSP JTAG header, this is accomplished by placing a jumper across TRST to BTRST, under the assumption that BTRST is tied low on the target pc board.

WORKAROUND:
1. If the board is not designed to be used with an emulator (no JTAG header), tie TRST to RESET. This will allow the JTAG circuitry to initialize properly and prevent TRST from going low after reset.
2. When the ADI DSP JTAG header is present on the board, BTRST should be tied to RESET instead of GND, under the assumption that there is no JTAG boundary scan controller on the board. In this case:
a. When the emulator is connected to the JTAG header, the VisualDSP emulator session will control the TRST signal. In order to debug booting, the user can hit run (F5) in the VisualDSP debugger window, reset the DSP and then halt the emulator to see the result of the boot. Be advised that the emulator pod has a pull down on TRST. If the emulator pod is connected while a VisualDSP emulator session is not active, the internal memory will not operate properly.
b.When the emulator is not connected to the JTAG header, jumper TRST to BTRST.

APPLIES TO REVISION(S):
0.0

8. 02000050 - Conditional type 10 instruction may fail in SIMD:

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

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

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

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

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

could be separated into:

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

APPLIES TO REVISION(S):
0.0, 0.1

9. 02000051 - Broadcast load fails in type 10 instruction:

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

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

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

could be separated into:

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

APPLIES TO REVISION(S):
0.0, 0.1

10. 02000052 - Rn=MANT Fx results will be rounded if RND32 is enabled:

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

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

11. 02000053 - In Serial Port Multichannel mode, an external RFS one cycle early causes data corruption:

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

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

APPLIES TO REVISION(S):
0.0

12. 02000054 - Illegal DAG stalls can occur under certain circumstances:

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

13. 02000055 - Execution of instructions that modify interrupt latch registers may cause incoming interrupts to be ignored:

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

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

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

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

14. 02000056 - A breakpoint following a JUMP, CALL, RTI or RTS can trigger an early/improper emulator break:

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

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

APPLIES TO REVISION(S):
0.0, 0.1

15. 02000057 - DAG register write followed directly by a DAG register read fails during context switch:

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

16. 02000058 - System registers written with a PM access do not have a 1 cycle effect latency:

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

17. 02000059 - Conditional RTI fails in SIMD mode:

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

18. 02000060 - Single instruction loops can terminate early:

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

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

APPLIES TO REVISION(S):
0.0, 0.1

19. 02000061 - Bit reversal fails with indirect jump:

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

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

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

APPLIES TO REVISION(S):
0.0, 0.1

20. 02000062 - VIPD bit gets cleared when branching to ISR:

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

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

APPLIES TO REVISION(S):
0.0, 0.1

21. 02000063 - MU (Multiplier underflow) flag gets set anomalously for some non-underflow cases:

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

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

```
X = 0x008A7DBEF6
Y = 0x3F7FF9FFF9

X = 0x26FFF70002
Y = 0x193FFFFFFE
```

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0, 0.1

22. 02000064 - Top of the loop address stack is filled with zero when PUSH LOOP instruction is executed:

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

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

APPLIES TO REVISION(S):
0.0, 0.1

23. 02000065 - DAG stall causes external port to malfunction:

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

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

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

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

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

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

2.	DAG Register initialized to external memory initially.	i12 = 0x4000000; // instx
		i12 = dm(m7,i6); // inst1
		r0 = pm(i12,0); // inst2
			Y	Y

3.	Wait register is initialized with zero wait states.	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.	r0 = 0x01CE739C;
		dm(WAIT) = r0;
		i12 = dm(m7,i6); // inst1
		r0 = pm(i12,0); // inst2
			Y	Y

5.	In the DAG Stall sequence, the DAG register is loaded using dm indirect load from memory and accessed using pm access.	i12 = dm(i7,m6); // inst1
		r0 = pm(i12,0); // inst2
			Y	Y

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

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

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

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

Note: This workaround may be built into the development tool chain and/or into the operating system source code. For tool chains and

Operating Systems supported by ADI, such as VisualDSP++ and VDK please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

APPLIES TO REVISION(S):

0.0, 0.1

24. 02000066 - Read of slave DSP's EPBx by host or master DSP in a multiprocessing may return wrong data:

DESCRIPTION:

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

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

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

WORKAROUND:

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

APPLIES TO REVISION(S):

0.0, 0.1

25. 02000069 - Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers:

DESCRIPTION:

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

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

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

WORKAROUND:

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

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

APPLIES TO REVISION(S):

0.0, 0.1