ANALOG DEVICES

SHARC Processor

Silicon Anomaly List — ADSP-21364

ABOUT ADSP-21364 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC® ADSP-21364 product(s) and the functionality specified in the ADSP-21364 data sheet(s) and the Hardware Reference book(s).

SILICON REVISIONS

A silicon revision number with the form "-x.x" is branded on all parts (see the data sheet for information on reading part branding). The silicon revision can also be electronically read by reading the REVPID register either via JTAG or DSP code.

The following DSP code can be used to read the register:
<UREG> = REVPID;

Silicon REVISION	REVPID[7:4]
0.5	0101
0.3	0011
0.1	0001

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
10/14/2010	O	F	Added anomalies: 07000022 &07000023
08/13/2009	N	D	Modified anomalies: 07000009
06/02/2009	M	D	Modified anomalies: 07000009, Added common note on Tools action for all the core related anomalies
02/25/2009	L	D	Added anomalies: 07000021
07/19/2007	K	0	Modified anomalies: 07000010 - Removed some special characters
12/28/2006	J	0	Removed anomalies: 07000020-This information is added to the HRM
12/05/2006	I	0	Modified anomalies: 07000019-Changed the Title
11/24/2006	H	0	Added anomalies: 07000020
11/09/2006	G	0	Modified anomalies: 07000019-Added the Workaround
09/28/2006	F	0	Added Anomalies:07000019
06/21/2006	E	0	Modified Anomalies:07000011

NR002951O

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-21364 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.1	0.3	0.5
1	07000005	Parallel Port Bus Contention	x	.	.
2	07000006	Delayed indirect jumps and calls can fail under specific conditions	x	.	.
3	07000007	SPDIF Receiver output data order change	x	x	.
4	07000008	Instruction Cache needs to be initialized	x	x	.
5	07000009	Some Core Stalls not executed properly	x	x	x
6	07000010	Memory write operations can fail under certain conditions while DMA to internal memory is in progress	x	x	x
7	07000011	SPI can generate spurious clock for an additional word when used in receive DMA mode at maximum SPICLK frequency	x	x	x
8	07000019	Input Shift Register Anomaly in ASRCs (Asynchronous Sample Rate Converters) impacts Daisy-Chained TDM mode	x	x	x
9	07000021	Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers	x	x	x
10	07000022	Conditional FLAG instructions involving DAG index registers must not be followed immediately by an instruction that uses the same index register	x	x	x
11	07000023	Hardware /IRQx interrupts may cause unpredictable behavior under some conditions	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-21364 including a description, workaround, and identification of applicable silicon revisions.

1. 07000005 - Parallel Port Bus Contention:

DESCRIPTION:

A bus contention issue has been identified on the ADSP-2136x processor's Parallel Port. When crossing an external memory address page while using DMA-READ transfers, an ALE cycle is inserted to update the upper address bits held by the external latch. When the Parallel Port transitions from a Read-cycle to an ALE cycle, insufficient time was provided to allow the external SRAM or FLASH to tri-state its bus. The bus tri-state from /OE de-assertion timing is typically longer for Flash than for SRAM devices, this necessitates a separate examination for both the SRAM and FLASH cases.

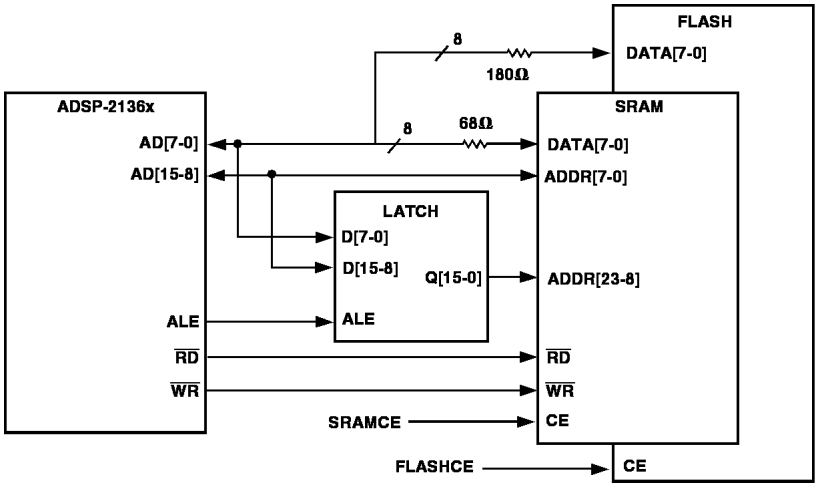
WORKAROUND:

a) SRAM CONTENTION :

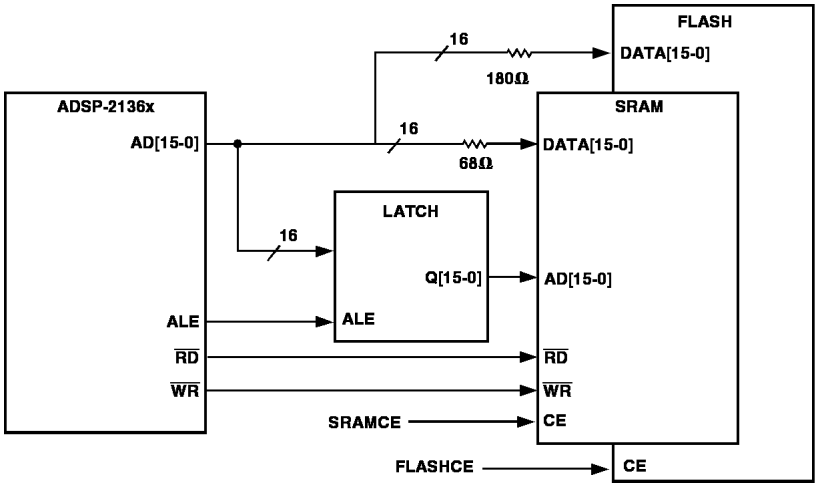
Hardware based work-around:

A typical 8ns SRAM will require 5-7ns to tri-state from /OE de-assertion. The ADSP-2136x only provides for approximately 4.0 ns; therefore, a 3.0ns contention possibility exists. At this time the recommended work-around is to place series resistors before the SRAM data lines, as shown in the diagram below. This resistor will limit the maximum pin current to about 22mA, while still providing sufficiently fast rise-fall rates to meet timing requirements. Our recommendation is 68-Ohms. This option will operate at full speed and require no software changes.

Memory Read—Parallel Port 8-Bit:



Memory Read—Parallel Port 16-Bit:



b) SRAM CONTENTION:

Software based work-around (1):
A Parallel Port hold cycle can be enabled, which will provide an additional 6.5ns for the external device to disable (A bus hold cycle is selected by setting the PPBHC bit in the PPCTL Register). This option will incur an additional 5ns per SRAM transfer, which will lower the overall external bandwidth; otherwise this option will be transparent to software.

Software based work-around (2):
Structure software to always ensure DMA transfers in a 256 byte (or smaller) external burst (in 8-bit mode), this will avoid an ALE cycle after a read. This option will require software to be written such that external pages will never be crossed by DMA access; however, it will allow full speed operation within a page.

c) FLASH CONTENTION:
Hardware based work-around:
A typical FLASH memory has a bus tri-state from OE disable time of 30ns. Bench testing of several FLASH devices revealed that this disable time is typically less than 10ns. However, to ensure safe operation under worst case specifications, the maximum specified FLASH disable must be accommodated. Should the FLASH truly require 30ns to tri-state, it would still be driving the bus during the falling edge of ALE, which strobes the upper addresses into the latch. So unlike the SRAM case above, for FLASH it is also necessary to ensure that valid logic levels are presented to the latch by the AD[15:0] lines during contention. To accomplish this, larger resistors need to be used; our recommendation is 180 Ohms. This resistor value is larger than typically encountered for the purpose of reducing reflections along digital lines; however, since the FLASH is a rather slow device, the rise/fall time delay induced will be commensurately small. To ensure proper operation, the delay incurred by this resistor should be accounted for by adding an additional wait state; simulations with a single FLASH indicate delays of less than 5ns. If these resistor values cannot be accommodated, it is possible to place a tri-state buffer in the data path of the FLASH, as its pass-though delay is relatively insignificant; unfortunately, the turn-off time of such buffers will probably exceed the required 2.5ns, and thus necessitate smaller series resistors between the ADSP-2136x device and the buffer to avoid excessive pad current.

d) FLASH CONTENTION:
Software based work-around:
The Parallel Port Bus Hold Cycle is too short to be effective for FLASH. The method of performing DMAs on a 256 byte (or smaller) external bursts (in 8-bit mode) is recommended when making FLASH accesses. Be advised that during parallel port booting, the processor uses 8-bit mode with consecutive DMA reads, and so contention will be present.

APPLIES TO REVISION(S):
0.1

2. 07000006 - Delayed indirect jumps and calls can fail under specific conditions:

DESCRIPTION:
A delayed indirect jump (or indirect call) to block-n doesn't work if it is followed by a dm access to block-n, and a DMA access also happens to block-n and coincides with the dm access.

For example:

```
back: jump (m14,i12) (db);    // M14 + I12 is in Block 0
r0=dm(i6,m7);   //I6 + M7 is to Block 0 & DMA access to Block 0 comes here
nop;
nop;
jump back;
nop;
nop;
```

In the example above, the instruction fetch from location M14 + I12 doesn't happen.

Note: These instructions should be in the same memory block as the DM access as well as the DMA access (DMA Data or TCBs). i12, i6, m14, m7 values should be initialized such that the access remains in the same memory block.

WORKAROUND:
1. Adding a nop after the delayed indirect branch will solve the problem.

2. Move the DMA data (and TCBs if chaining is used) to a different memory block.

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

APPLIES TO REVISION(S):
0.1

3. 07000007 - SPDIF Receiver output data order change:

DESCRIPTION:
The SPDIF Receiver (DIR) sends the Right Channel information before the Left Channel information for all the samples in SPDIF Receiver output. However the output is still compliant with the I2S standard.

```
Ln represents the left channel from frame n
Rn represents the right channel from frame n

t=0 -------------------------> t=3*Fs
```

What the SPDIF RX receives, and should output in I2S format.
L0 -> R0 -> L1 -> R1 -> L2 -> R2 -> L3 -> R3 -> ...

Anomalous output:
R0 -> L0 -> R1 -> L1 -> R2 -> L2 -> R3 -> ...

WORKAROUND:
If the output of the SPDIF receiver is routed via a Serial Port to the processor's internal memory, the data can be manipulated in software to correct the temporal change in order by writing the Left Channel data to memory before the Right Channel data. In applications using the SPDIF receiver to receive encoded data streams such as AC-3 etc, the temporal order of the samples is very important. In order for decoding and stream detect algorithms to function, it is imperative to either implement this workaround, or adjust the algorithm to expect the input data to be in a modified order.

For some applications where the SPDIF stream consists of two channels of PCM audio data, no workaround is necessary. The data is retained in the correct I2S channel, and there is no swapping of the Left and Right channels.

APPLIES TO REVISION(S):
0.1, 0.3

4. 07000008 - Instruction Cache needs to be initialized:

DESCRIPTION:
The instruction cache in the ADSP-2136x processor is not properly initialized upon power-up. As a result of this the first data read (cache hit) from the cache may be incorrect, possibly causing the processor to execute an invalid instruction. This sort of scenario can apply in code that includes PM data accesses within a loop. (In the second iteration of the loop, the PM data access instruction will be read from cache by the program sequencer, and due to this issue the instruction from cache may be invalid.

WORKAROUND:
The workaround for this issue is to place the code snippet below into the processor's initialization routine. It will force the cache to be initialized properly, and after executing this code snippet the instruction cache will function reliably. It is important to execute this code before the instruction cache is to be relied upon.

```
BIT SET MODE2 CADIS; // Disable the cache
NOP;                 // One nop for effect latency
READ CACHE 0;        // Read cache instruction
BIT CLR MODE2 CADIS; // Re-enabling the cache.
```

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

APPLIES TO REVISION(S):
0.1, 0.3

5. 07000009 - Some Core Stalls not executed properly:

DESCRIPTION:
Under certain specific conditions outlined below, 3 types of stalls that are normally incurred do not get executed. In certain cases this can cause unexpected code operation.

1.Multiplier stalls:
In normal operation, if both operands in a multiplier instruction are produced by either the multiplier or the ALU in the previous instruction, the pipeline is stalled for 1 cycle.

```
[1] F0=F0+F4, F1=F0-F4, R14 = DM(I0,M0);
[2] F4=F0*F1; <---- this should stall
```

Anomalous behavior:
If: The memory access in instruction 1 is to a memory-mapped IOP register.
-Or-
DMA is simultaneously accessing the same bank as the memory access in instruction 1.

Then: Stall does not occur, and this results in the product in F4 being incorrect.

2.Register writes mode1 / mode2:
In normal operation, a stall cycle is added after any write to MODE1 or MODE2 registers.

-A stall is also added when the content of MODE1 and MODE2 registers are modified through the bit manipulation instruction.

-The value of MODE1 also changes when PUSH STS or POP STS instructions are executed, or when the sequencer branches to or returns from an ISR involving push/pop of status stack (such as any of the hardware interrupts like IRQ and core timer interrupts). In such cases the pipeline is stalled for a cycle.

```
Example1:
[1] MODE1= DM(I0,M0); /*Enable bit reverse addressing for I8 */
[2] PM(I8,M8)=R14; /* stalls for a cycle but unaffected by mode setting */
[3] PM(I8,M8)=R14; /* performs bit reversed mode of addressing */
[4] …

Example2:
[1] MODE1= DM(I0,M0); /*Enable bit reverse addressing for I8 */
[2] r0 = r1 + r2; /* stalls for a cycle */
[3] PM(I8,M8)=R14; /* performs bit reversed mode of addressing */
[4] …
```

Anomalous behavior:
If: The memory access in instruction 1 is to a memory-mapped IOP register.
-Or-
DMA is simultaneously accessing the same bank as the memory access in instruction 1.

Then: Stall does not occur, and this results in instruction 3 not being affected by the mode change.

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

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

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

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

DMA is simultaneously accessing the same bank as the memory access in instruction 1.

Then: Stall does not occur, and this results in the RTS/RTI vectoring to some unknown location instead of the value from PCSTK.

Note: Multiplier stalls case is fixed for the 0.4 and 0.5 revision of the silicon.

WORKAROUND:
For cases 1 and 3, add a nop between instructions 1 and 2.
For case 2 add a nop between instructions 2 and 3.

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

APPLIES TO REVISION(S):
0.1, 0.3, 0.5

6. 07000010 - Memory write operations can fail under certain conditions while DMA to internal memory is in progress:

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

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

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

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

Example:

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

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

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

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

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

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

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

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

Example:

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

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

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

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

APPLIES TO REVISION(S):
0.1, 0.3, 0.5

7. 07000011 - SPI can generate spurious clock for an additional word when used in receive DMA mode at maximum SPICLK frequency:

DESCRIPTION:
SPI can generate spurious clock for an additional word when the following three conditions are met:-

1. The SPI is configured for 32-bit, receive master DMA mode and performs back to back DMA transfers.
2. The SPI DMA is disabled and re-enabled between back to back DMA transfers without disabling the SPI itself.
3. The SPI is operated at the maximum SPICLK frequency (SPIBAUD = 2).

When the above three conditions are met, SPI can generate clock for an additional word even after the FIFO and the receive buffer is full. This clock for another word can cause the receive buffer overflow error which will result in loss of the additional received data word.

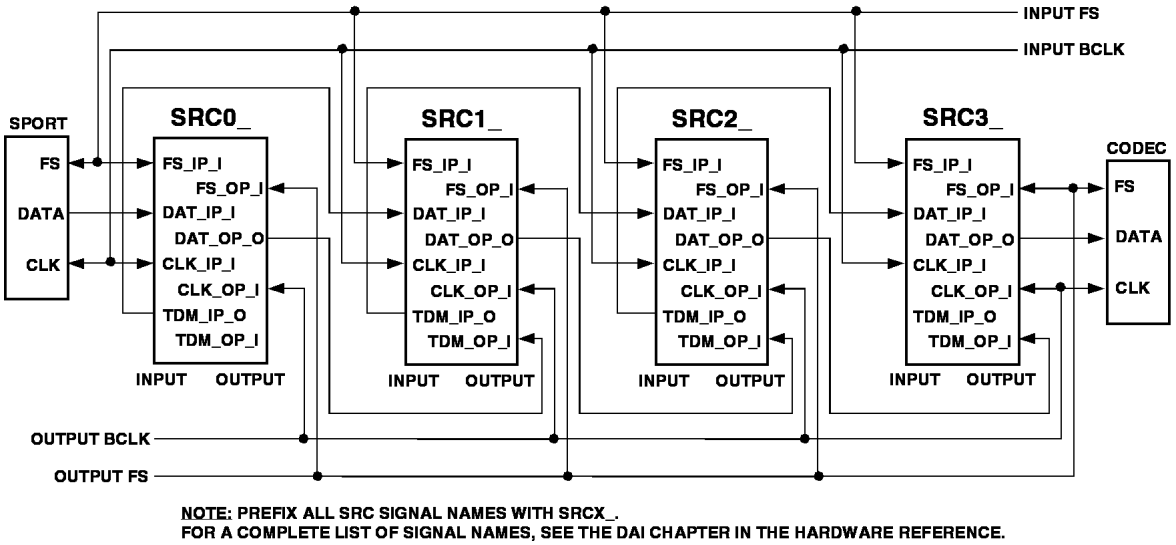
WORKAROUND:
1. Do not disable the SPI DMA between back to back DMA transfers.

2. If the application demands disabling SPI DMA between back to back DMA transfers then the sequence given below must be followed:
   - Disable the SPI DMA
   - Disable the SPI
   - Clear the FIFO and the receive buffer
   - Configure the DMA descriptors and Enable the SPI
   - Enable the SPI DMA

APPLIES TO REVISION(S):
0.1, 0.3, 0.5

8. 07000019 - Input Shift Register Anomaly in ASRCs (Asynchronous Sample Rate Converters) impacts Daisy-Chained TDM mode:

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



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

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

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

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

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

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

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

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

The output data will be as follows,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

APPLIES TO REVISION(S):
0.1, 0.3, 0.5

9. 07000021 - 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):
0.1, 0.3, 0.5

10. 07000022 - Conditional FLAG instructions involving DAG index registers must not be followed immediately by an instruction that uses the same index register:

DESCRIPTION:
In the following instruction sequence shown below:

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

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

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

WORKAROUND:
Separate the second and third instructions in the above sequence by at least two NOP's.

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

APPLIES TO REVISION(S):
0.1, 0.3, 0.5

11. 07000023 - Hardware /IRQx interrupts may cause unpredictable behavior under some conditions:

DESCRIPTION:
The /IRQx interrupts may lead to an unpredictable behavior of the processor under some conditions. The problem happens under a specific timing window between the /IRQx signal and the internal clock references. At this point, a couple of malfunctions (out of many possible ones) that may occur are - the fetch address corresponding to the /IRQx interrupt from the IVT is corrupted, this can lead to corrupted instruction and/or the sequencer branching to unknown/invalid location. This problem is applicable for both edge-sensitive and level-sensitive configurations of the /IRQx interrupts and is seen only when the processor is operated with the core-clock frequency above 285MHz.

WORKAROUND:
1) DAI interrupts should be used for external interrupts, instead of /IRQx interrupts.
2) If possible, the core clock frequency can be reduced to 285MHz or lower.

APPLIES TO REVISION(S):
0.1, 0.3, 0.5