SHARC+ Dual Core DSP with ARM Cortex-A5

Silicon Anomaly List

ADSP-SC570/571/572/573/ADSP-21571/573

ABOUT ADSP-SC570/571/572/573/ADSP-21571/573 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC+® ADSP-SC570/571/572/573/ADSP-21571/573 product(s) and the functionality specified in the ADSP-SC570/571/572/573/ADSP-21571/573 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. The REVID bits <31:28> of the TAPC0_IDCODE register can be used to differentiate the revisions as shown below.

Silicon REVISION	TAPC0_IDCODE.REVID
0.2	b#0010
0.0	b#0000

APPLICABILITY

Peripheral-specific anomalies may not apply to all processors. See the table below for details. An "x" indicates that anomalies related to this peripheral apply only to the model(s) indicated, and the list of specific anomalies for that peripheral appear in the rightmost column

Non-Automotive:

Peripheral/Core	SC570W	SC571W	SC572W	SC573W	21571W	21573W	Anomalies
ARM Cortex-A5	x	x	x	x			20000082,20000085

Automotive:

Peripheral/Core	SC570W	SC571W	SC572W	SC573W	21571W	21573W	Anomalies
MLB 6-pin mode			x	x		x	20000079
ARM Cortex-A5	x	x	x	x			20000082,20000085

ANOMALY LIST REVISION HISTORY

The following revision history lists the anomaly list revisions and major changes for each anomaly list revision.

Date	Anomaly List Revision	Data Sheet Revision	Additions and Changes
04/14/2023	G	B	Added Anomaly 20000123
07/27/2020	F	B	Added Anomaly 20000108 Revised Anomalies 20000096, 20000031 for Clarity Fixed Anomaly 20000067 Workaround Description
05/08/2019	E	B	Added Anomaly 20000101
10/23/2018	D	B	Added Anomaly 20000096
08/24/2018	C	B	Added Silicon Revision 0.2 Added Anomalies 20000090, 20000091, 20000093, 20000094

SHARC+ and SHARC are registered trademarks of Analog Devices, Inc.

NR004534G **Document Feedback**

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

One Analog Way, Wilmington, MA 01887 U.S.A.
©2023 Analog Devices, Inc. All rights reserved.
Technical Support **www.analog.com**

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-SC570/571/572/573/ADSP-21571/573 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	Rev 0.0	Rev 0.2
1	20000002	Data Forwarding from Rn/Sn to DAG Register May Fail in Presence of Stalls	x	x
2	20000003	Transactions on SPU and SMPU MMR Regions May Cause Errors	x	x
3	20000031	GP Timer Generates First Interrupt/Trigger One Edge Late in EXTCLK Mode	x	x
4	20000037	DMC Read State Machine May Not Be in a Correct State after DMC Initialization	x	x
5	20000043	Key Unwrapping on the SHARC+ Core Fails when Using ROM API	x	x
6	20000051	Secure SPI Master Boot Only Supported from Memory-Mapped SPI Devices on SPI2	x	x
7	20000062	Writes to the SPI_SLVSEL Register Do Not Take Effect	x	x
8	20000067	DMC Clock Signal May Violate JEDEC Timing Specification in Self-Refresh Mode	x	x
9	20000069	PCSTK and MODE1STK Loads Do Not Occur If Next Instruction Is L2 or L3 Access	x	x
10	20000071	Burst Mode in IIR Accelerator May Not Work Properly	x	x
11	20000072	Floating-Point Computes Targeting F0 Register Can Cause Pipeline Stalls	x	x
12	20000073	DDR Frequency Is Limited to 300 MHz When Using OTP for DMC Programming	x	x
13	20000074	Peripheral Interrupt Request for Link Port DMA Is Not Supported	x	x
14	20000076	SPI Slave Transmit DMA Peripheral Interrupt Is Generated Prematurely	x	x
15	20000077	Bit Clear Instructions Affecting IRPTL Register Can Cause Core Hang when Single-Stepped	x	x
16	20000078	Bit-Reversed Addressing Mode May Fail for Non-L1 Addresses	x	x
17	20000079	MLB Operation at 3072x Fs and 4096x Fs Is Not Functional	x	x
18	20000080	Quad-SPI Master Boot Modes Are Not Functional	x	x
19	20000081	SEC Interrupts Do Not Latch when Aligned with an Explicit Core Write to IRPTL Register	x	x
20	20000082	Unaligned Half-Word Reads of Non-Cacheable Memory Locations Return Incomplete Data	x	x
21	20000083	Speculatively Executed Pre-Modify DM Reads Can Cause Processor Malfunction	x	x
22	20000084	Simultaneous OTP Accesses by Multiple Cores Can Cause Core Hang	x	x
23	20000085	SPU Protection for Peripherals Doesn't Work Against Accesses from Cortex-A5 Core	x	x
24	20000088	High Leakage Current During Reset when HADC Is Used	x	x
25	20000090	Single-Ended Clock/DQS Measurements May Violate JESD79-3E/-2E Vix and VSWING Specs	x	x
26	20000091	Accesses to DMC_CPHY_CTL Register Do Not Function As Expected	x	x
27	20000093	Power-Up Sequencing May Cause Pins to Be Unexpectedly Driven	x	.
28	20000094	SPDIF Receiver Output Clock Is Unreliable	x	x
29	20000096	Type 18a USTAT Instructions Fail When Following Specific Code Sequence	x	x
30	20000101	SMPU Hang when Exclusive Read Arrives While Non-Exclusive Write Is Pending	x	x
31	20000108	Factory Serial Number Cannot Be Read When Device Is Locked	x	x
32	20000123	Boot failure with Ignore Block in Page Mode	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-SC570/571/572/573/ADSP-21571/573 including a description, workaround, and identification of applicable silicon revisions.

1. 20000002 - Data Forwarding from Rn/Sn to DAG Register May Fail in Presence of Stalls:

DESCRIPTION:
An instruction involving a DAG operation such as address generation or modify following a type5a instruction may fail under the following conditions:

1. The type5a instruction updates the source register of the subsequent DAG operation.
2. The type5a instruction uses the same source register to both load to the DAG register and store the result of the compute operation.
3. The DAG operation follows within six instructions of the type5a instruction.
4. The pipeline is stalled due to a data/control dependency or an L1 memory bank conflict.

When these conditions are met, the type5a instruction produces the expected result and updates the DAG register correctly; however, the data forwarded to the DAG is incorrect, and the DAG register used as the destination in the subsequent DAG operation is incorrectly updated.

Consider the following type5a instruction sequence:

```
1: r2 = r2 - r13, i4 = r2; // r2 is destination of compute AND source of DAG load
2: if eq jump target1;     // Dependency on previous instruction stalls the pipe
3: nop;
4: nop;
5: nop;
6: nop;
7: i5 = b2w (i4);          // Uses source register (i4) stored to by type5a instruction
```

In the above case, **i5** (line 7) is updated with an incorrect value, even though **i4** (line 1) contains the correct value. The same would be true if the instruction on line 7 appeared anywhere in lines 3 through 6.

WORKAROUND:
There are two potential workarounds for this issue:

1. Split the type5a instruction which conforms to the use case into two separate instructions.
2. Avoid using the relevant DAG register in a DAG operation within six instructions of the type5a instruction.

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 Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
0.0, 0.2

2. 20000003 - Transactions on SPU and SMPU MMR Regions May Cause Errors:

DESCRIPTION:
Non-secure reads or writes to the upper half of each SPU and SMPU instance's MMR space will be erroneously blocked and cause a bus error when configured as a non-secure slave.

For each instance of the SPU and SMPU, the affected MMR address range can be calculated as follows:

- Lower bound = Instance Address Offset + **0x800**
- Upper bound = Instance Address Offset + **0xFFF**

WORKAROUND:
Do not access the documented system MMR ranges from a non-secure slave.

APPLIES TO REVISION(S):
0.0, 0.2

3. 20000031 - GP Timer Generates First Interrupt/Trigger One Edge Late in EXTCLK Mode:

DESCRIPTION:
When any GP Timer is configured in External Clock mode, the first interrupt/trigger should occur when the corresponding **TIMER_DATA_ILAT** bit sets after the **TIMER_TMRn_CNT** register reaches the value programmed in the **TIMER_TMRn_PER** register. Instead, the interrupt/trigger and the setting of the **TIMER_DATA_ILAT** bit occur one signal edge later. At this point, the **TIMER_TMRn_CNT** register will have rolled over to 1. Subsequent interrupts/triggers occur after the correct number of edges.

For example, if **TIMER_TMRn_PER**=7, the first interrupt/trigger will occur after the timer pin samples eight edges. From that point forward, interrupts/triggers will correctly occur every seven signal edges.

WORKAROUND:
For interrupts/triggers to occur every **n** edges detected on the timer pin, the **TIMER_TMRn_PER** register must be configured to **n-1** for the initial event and then reprogrammed to **n** for subsequent events, as shown in the following pseudocode:

```
TIMER_TMRn_PER = n-1;   // Configure PERIOD register with n-1
TIMER_RUN_SET = 1;      // Enable the timer
TIMER_TMRn_PER = n;     // Configure PERIOD register with n
```

The second write to the **TIMER_TMRn_PER** register does not take effect until the 2nd period; therefore, this sequence can be performed when the timer is first enabled.

APPLIES TO REVISION(S):
0.0, 0.2

4. 20000037 - DMC Read State Machine May Not Be in a Correct State after DMC Initialization:

DESCRIPTION:
DMC read accesses require that the DMC read state machine be in the correct state. Due to this anomaly, the DMC read state machine may not be in a correct state after initialization, which may result in DMC read failures.

WORKAROUND:
After DMC initialization, the data capture logic must be reset to place the DMC read state machine into a valid state. For example, the following C code can be used for DMC0:

```
#include <sys/platform.h>  /* defines REG and BITM macros used below */

uint32_t uiDMC_Data;

uiDMC_Data = (void)(*((volatile uint32_t *)0x80000000uL)); /* Dummy DMC memory read */
*pREG_DMC0_PHY_CTL0 |= BITM_DMC_PHY_CTL0_RESETDAT;         /* Set DMCx_PHY_CTL0.RESETDAT */
*pREG_DMC0_PHY_CTL0 &= ~BITM_DMC_PHY_CTL0_RESETDAT;        /* Clear DMCx_PHY_CTL0.RESETDAT */
```

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 Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
0.0, 0.2

5. 20000043 - Key Unwrapping on the SHARC+ Core Fails when Using ROM API:

DESCRIPTION:
If the stack is mapped to L1 memory, the SHARC+ ROM API call for BLw (key unwrapping) secure boot fails. Key unwrap operation uses the PKTE DMA engine, which is configured to work only with L2 space in the boot ROM.

WORKAROUND:
The ROM API application must resolve the stack to either L2 or L3 memory.

APPLIES TO REVISION(S):
0.0, 0.2

6. 20000051 - Secure SPI Master Boot Only Supported from Memory-Mapped SPI Devices on SPI2:

DESCRIPTION:
Secure SPI master boot can only be done in memory-mapped mode from SPI2. SPI0 and SPI1 do not support memory-mapped mode; therefore, they cannot support secure SPI master boot. The same restriction applies when calling the ROM API to boot.

WORKAROUND:
If Secure SPI boot is needed, configure the **dBootCommand** to use SPI2 in Memory-Mapped mode. When calling the ROM API, ensure that the lowest nibble (boot source device) of the boot command parameter is 0x7. The memory-mapped address from which the boot needs to be started from must be passed as the start address parameter.

The following is an example where the ROM API is called to boot from the SPI flash mapped to 0x60000000 in Memory-Mapped mode using SPI2:

```
adi_rom_Boot(0x60000000,0,0,0,0x207);
```

APPLIES TO REVISION(S):
0.0, 0.2

7. 20000062 - Writes to the SPI_SLVSEL Register Do Not Take Effect:

DESCRIPTION:
A single write to the **SPI_SLVSEL** register should change the state of the register and cause the modified software-controlled SPI slave selects to assert or de-assert. Instead, a single write to **SPI_SLVSEL** has no effect.

WORKAROUND:
Any write to **SPI_SLVSEL** should be done twice (back-to-back) with the same value in order for the change to take effect.

APPLIES TO REVISION(S):
0.0, 0.2

8. 20000067 - DMC Clock Signal May Violate JEDEC Timing Specification in Self-Refresh Mode:

DESCRIPTION:
The differential DMC clock signals (**DMC_CK** and **$\overline{\text{DMC\_CK}}$**) may violate the JEDEC timing specification when the device enters Self-Refresh mode. This anomaly applies to all three DDR modes (DDR3/DDR2/LPDDR).

WORKAROUND:
Bit 18 of the **DMC_PHY_CTL1** register must be set if the DDR device (DDR3/DDR2/LPDDR) must be placed into Self-Refresh mode, per the following code for DMC0:

```
*pREG_DMC0_PHY_CTL1|=0x40000;
```

This bit need not be set each time the device is put in Self-Refresh mode.

APPLIES TO REVISION(S):
0.0, 0.2

9. 20000069 - PCSTK and MODE1STK Loads Do Not Occur If Next Instruction Is L2 or L3 Access:

DESCRIPTION:
Writes to the **PCSTK** and **MODE1STK** registers may not happen correctly if the next instruction is an access to a non-L1 memory location, as in the following code sequence:

```
1: MODE1STK = r0;
2: PCSTK    = dm(0,i6);  // i6 points to L2 or L3 memory space
3: px2      = dm(0,i6);
```

Because **i6** points to non-L1 memory in this sequence, the **MODE1STK** write on line 1 fails due to the use of **i6** on line 2, and the write to **PCSTK** on line 2 also fails because of the same use of **i6** on line 3.

WORKAROUND:
Insert a **NOP;** instruction between the write to the **PCSTK**/**MODE1STK** register and the next memory access instruction.

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 Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):
0.0, 0.2

10. 20000071 - Burst Mode in IIR Accelerator May Not Work Properly:

DESCRIPTION:
The burst mode in the IIR accelerator (enabled by the **IIR_CTL1.BURSTEN** bit) can be used to enhance the performance of the IIR accelerator by enabling burst accesses while loading the TCB and coefficients. This mode may not provide the expected results.

WORKAROUND:
If the expected coefficient modifier (**IIR_COEFMOD**) for all channels is 1, the following code sequence can be used prior to enabling the IIR accelerator:

1. Set ***pREG_IIR0_INMOD** = 1;
2. Start the IIR accelerator by programming the **IIR0_CTL1** register.

For other cases, do not use burst mode (set **IIR_CTL1.BURSTEN** = 0).

APPLIES TO REVISION(S):
0.0, 0.2

11. 20000072 - Floating-Point Computes Targeting F0 Register Can Cause Pipeline Stalls:

DESCRIPTION:
Any floating-point compute instruction with **F0** as the destination register will cause pipeline stalls when followed immediately by a no-operand or single-operand compute instruction with **Rx** as the unused source register, as in the following code sequence:

```
F0 = PASS F4;
R10 = PASS R11; // Y operand is not used. Flushed to 0 in opcode by assembler.
```

WORKAROUND:
There are two possible workarounds:
1. Do not use the F0 register as the destination in the above code sequence.
2. Ensure that the instruction that immediately follows the compute operation is not of the form described in the code example above.

APPLIES TO REVISION(S):
0.0, 0.2

12. 20000073 - DDR Frequency Is Limited to 300 MHz When Using OTP for DMC Programming:

DESCRIPTION:
For DDR2 and DDR3 modes of operation, the DMC parameters programmed to the DMC space in OTP memory must not result in a DCLK frequency above 300 MHz.

WORKAROUND:
For Non-Secure boot, use initialization code if the DMC needs to operate at a DCLK frequency greater than 300 MHz. This workaround is not valid for Secure boot, as initialization code is not supported.

For Secure boot, ensure the DMC space in OTP memory is configured to not exceed a frequency of 300 MHz. After booting, DCLK can be changed to the desired frequency.

APPLIES TO REVISION(S):
0.0, 0.2

13. 20000074 - Peripheral Interrupt Request for Link Port DMA Is Not Supported:

DESCRIPTION:
Setting **DMA_CFG.INT**=0x3 enables the DMA interrupt to come from the peripheral after it performs the last grant to the peripheral. This setting is not functional for Link Port DMA.

WORKAROUND:
For Link Port DMA, do not use the **DMA_CFG.INT**=0x3 setting. Instead, set **DMA_CFG.INT** such that the DMA channel asserts the interrupt when either the X count (**DMA_CFG.INT**=0x1) or Y count (**DMA_CFG.INT**=0x2) expires.

APPLIES TO REVISION(S):
0.0, 0.2

14. 20000076 - SPI Slave Transmit DMA Peripheral Interrupt Is Generated Prematurely:

DESCRIPTION:
When the SPI port is configured as a slave, the peripheral interrupt (PIRQ) associated with the **DMA_CFG.INT**=0x3 setting for a transmit DMA is asserted by the SPI when the last data word is loaded into the SPI shift register, not after the data has fully shifted out. As such, any interrupt code associated with the event may be executed before the final word has actually reached its destination, which may cause application timing issues such as premature disabling of the SPI port, etc.

WORKAROUND:
If interrupt handling code must execute after the data has fully transferred to the master device, insert a polling loop awaiting the **SPI_STAT.SPIF** bit to set at the beginning of the SPI DMA handler code, per the following pseudo-code:

```
while(!(*pREG_SPI_STAT & SPIF)); // Wait for SPIF = 1
...                              // Now data is fully shifted out to the master
```

APPLIES TO REVISION(S):
0.0, 0.2

15. 20000077 - Bit Clear Instructions Affecting IRPTL Register Can Cause Core Hang when Single-Stepped:

DESCRIPTION:
Any **`BIT CLR IRPTL <data32>;`** instruction, regardless of the value of the argument, will clear any pending emulation interrupt in the IRPTL register when executed in an uninterruptible region of code, such as in the delay slots of a delayed branch. In such scenarios, the emulation interrupt for single-stepping will be pending in IRPTL to be serviced and should be cleared by a subsequent **`BIT CLR IRPTL <data32>;`** instruction. Hence, single-stepping over this instruction leads to the emulator losing control over the core, thus causing the core to hang. For example, consider the sequence:

```
jump (m14,i12) (db);
bit clr irptl 0x000000;
nop;
nop;
nop;
```

Single-stepping the **`BIT CLR IRPTL <data32>;`** instruction in other places in the code works as expected, as the emulator interrupt gets serviced before it gets cleared as a result of this anomaly.

WORKAROUND:
When debugging, do not single-step through **`BIT CLR IRPTL <data32>;`** instructions that occur in non-interruptible regions of code.

APPLIES TO REVISION(S):
0.0, 0.2

16. 20000078 - Bit-Reversed Addressing Mode May Fail for Non-L1 Addresses:

DESCRIPTION:
Bit-reversed addressing mode for non-L1 addresses can fail if:
1. normal word aliases of the byte word addresses are used, OR
2. the byte word address is a multiple of 0 or 8.

WORKAROUND:
Use only L1 locations for bit-reversed addressing.

If using non-L1 locations, do not use normal word aliasing, and take the following precautions:
1. If in SISD mode, pack the data such that it starts at an address which is not a multiple of 0 or 8 (assumes the accesses are not byte accesses).
2. If in SIMD mode and making 32-bit accesses, add an offset of 1 to the address **only** when the data can still be packed starting at 0.

APPLIES TO REVISION(S):
0.0, 0.2

17. 20000079 - MLB Operation at 3072x Fs and 4096x Fs Is Not Functional:

DESCRIPTION:
The MLB supports up to 4096x Fs in 6-pin mode; however, the 3072x Fs and 4096x Fs modes are not functional.

WORKAROUND:
Do not use the 3072x Fs or 4096x Fs modes.

APPLIES TO REVISION(S):
0.0, 0.2

18. 20000080 - Quad-SPI Master Boot Modes Are Not Functional:

DESCRIPTION:

When the processor is configured to boot as a SPI Master, the boot ROM fails to configure the flash devices for quad-SPI operation for the modes associated with the SPI Master **BCODE** settings of 0xA (QOR READ using Quad Mode Method 1) and 0xB (QIOR READ using Quad Mode Method 1). Due to this anomaly, the boot fails for these cases.

WORKAROUND:

Use any of the single-bit or dual-bit modes (**BCODE**=0x1-0x9) associated with SPI Master booting.

If quad-SPI booting is desired:

1. For non-secure booting, use an initialization code block booted in using one of the single/dual-bit modes above, where the code contained in the block manually changes the SPI configuration to quad-SPI mode. Once that initialization code executes, the rest of the boot stream will be in QOR/QIOR mode.
2. For secure booting, initialization blocks cannot be used. The SPI Master boot mode **dbootcommand** parameter in OTP memory can be programmed with the correct SPI I/O protocol to boot in QOR/QIOR mode, as well as the number of dummy bytes, address bytes, etc. This requires that the NOAUTO bit in **dbootcommand** is set so that the boot ROM uses the settings provided in the rest of its fields.

APPLIES TO REVISION(S):

0.0, 0.2

19. 20000081 - SEC Interrupts Do Not Latch when Aligned with an Explicit Core Write to IRPTL Register:

DESCRIPTION:

The **BIT SET IRPTL <data32>;** and **BIT CLR IRPTL <data32>;** instructions block internal interrupt signals during the instruction's execution. If an internal interrupt signal is pulsed (asserted for one cycle rather than asserted and held) during this time, then it will not be latched in **IRPTL**, and the interrupt will be missed.

The sources with pulsed interrupt request signals that are sensitive to this issue are the illegal opcode, core timer, emulation, and illegal address detection core interrupts, as well as the SEC system interrupt (SECI).

WORKAROUND:

The core timer interrupt is predictable, thus the core write to the **IRPTL** register can be sufficiently padded by **NOP;** instructions in the application code to prevent the precise timing alignment required for the anomaly to manifest. However, the SEC interrupt sources are either unpredictable or asynchronous in nature, where this workaround is not applicable. For those interrupt sources, the workarounds are:

1. Do not use the **BIT SET IRPTL <data32>;** and **BIT CLR IRPTL <data32>;** instructions to generate or ignore interrupts.
2. If core software interrupts are required in the application, instead use one of the eight System Software interrupts (SOFTx_INT), as follows:
   a. If the desired software interrupt behavior is asynchronous (i.e., the ISR associated with the raised software interrupt does not have to execute before the application code that immediately follows where the software interrupt is raised), raise the interrupt by writing the source ID for the chosen SOFTx_INT system interrupt to the **SEC0_RAISE** register.
   b. If the desired software interrupt behavior is synchronous (i.e., the ISR associated with the raised software interrupt must execute before the application code continues beyond where the software interrupt is raised), a software semaphore must also be used. For example, the ISR associated with the SOFTx_INT interrupt sets a dedicated flag in memory. This flag must be cleared in the application code immediately before the software interrupt is raised (by writing the desired SOFTx_INT interrupt source ID to the **SEC0_RAISE** register). Once the software interrupt is raised, the application must then poll for the flag to be set again by the ISR code before proceeding to the instructions that must follow the ISR.

APPLIES TO REVISION(S):

0.0, 0.2

20. 20000082 - Unaligned Half-Word Reads of Non-Cacheable Memory Locations Return Incomplete Data:

DESCRIPTION:

When the MMU is enabled without alignment fault checking (**SCTLR[1:0]** = b#01), unaligned half-word read accesses may return only half the data if the memory region being accessed is defined as Normal Non-Cacheable memory.

This issue can occur when the virtual address (VA) of the location being accessed by the following instructions is unaligned, specifically:

1. (in all ARM/Thumb addressing modes) when an **LDRH**, **LDRHT**, **LDRSH**, or **LDRSHT** load register instruction reads a location whose **VA[1:0]** = b#01,
2. (for ARM/Thumb with NEON SIMD enabled) when a **VLD1.16** single n-element structure to one or all lanes vector load instruction with the {align} field omitted reads a location whose **VA[1:0]** = b#01, OR
3. (for ARM/Thumb with NEON SIMD enabled) when a **VLD2.16**, **VLD3.16**, or **VLD4.16** single n-element structure to one or all lanes vector load instruction with the {align} field omitted reads a location whose **VA[0]** = 1.

WORKAROUND:

This issue only occurs when accessing non-cacheable memory. If all data memory that can be accessed by the ARM core is made cacheable, the issue is avoided.

If any portion of data memory that is accessible by the ARM core cannot be cached, then the half-word access instructions defined above must not exist in the application unless unaligned accesses are not possible (e.g., if the GNU compiler -mno-unaligned-access switch is used to build the application).

APPLIES TO REVISION(S):

0.0, 0.2

21. 20000083 - Speculatively Executed Pre-Modify DM Reads Can Cause Processor Malfunction:

DESCRIPTION:

When performing a DAG operation with modify, it is expected behavior for the processor to potentially malfunction (e.g., cause a core hang, etc.) if the DAG index register points to different memory regions before and after the modify value is applied; however, speculatively-executed pre-modify **DM** read accesses may violate this policy when a specific code sequence gets flushed from the pipeline as a result of a change in program flow (e.g., a branch, loop, or interrupt), specifically:

1. a UREG register is updated via a compute or load operation,
2. the same UREG register is used to load a DAG register, and
3. a pre-modify **DM** read instruction uses this DAG register (or a related DAG register; e.g., **I0** and **B0** are related).

When this occurs, a stale value in the UREG register gets forwarded to the speculatively-executed pre-modify read operation, which may cause the index plus modifier operation to violate the memory boundary policy. Consider the following sequences with r4 = 0x37070000:

1. A compute instruction stores to a UREG register that is then moved to the DAG register used in a pre-modify read instruction:

```
1: r4 = r1 + r2;                    // Computation updating UREG register r4
2: m0 = r4;                         // r4 transferred to DAG register m0
3: r4 = r4 + 1, f0 = dm(m0,i0);     // DM pre-modify read uses m0
```

   If an interrupt occurs just before this sequence, instruction 1 does not execute due to the pipeline flush; however, instructions 2 and 3 are already staged in the pipeline, and instruction 2 updates **m0** with the stale **r4** value (0x37070000) rather than the computation result from instruction 1. As 0x37070000 is a very large modify value, the DAG policy is violated and a malfunction occurs even though instruction 3 is never actually executed. The same would be true if instruction 2 stored to either **i0** or **b0**.

2. The same as above, except in the context of a branch instruction:

```
1: jump Here (db);
2: r4 = r1 + r2;                     // Computation updating UREG register r4
3: m0 = r4;                          // r4 transferred to DAG register m0
...                                  // Any number of instructions
4: Here: r4 = r4 + 1, f0 = dm(m0,i0); // DM pre-modify read uses m0
```

   When the BTB predicts the instruction 1 branch as taken, instructions 2, 3, and 4 are placed in the pipeline sequentially. If an interrupt occurs just before this sequence, the same speculative read behavior as described in case #1 above occurs (i.e., instruction 4 uses an **m0** value that violates the DAG policy). This behavior holds true if the **jump Here (db);** instruction is placed between or after instructions 2 and 3, and the anomaly would also manifest if instruction 3 stored to either **i0** or **b0**.

WORKAROUND:

The anomaly does not occur with data reads via the PM bus, so instead use PM reads in such sequences. If DM reads are required, ensure separation of at least four unrelated instructions between the transfer from the UREG register to the DAG register and either the preceding or subsequent instructions in the sequence. Using the example code sequences above to show both implementations:

1. Ensure that four instructions that do not use the affected registers are between the DAG register load and the DM read instruction:

```
r4 = r1 + r2;                       // Computation updating UREG register r4
m0 = r4;                            // DAG register load
nop; nop; nop; nop;                 // ANY 4 instructions that do not use m0
r4 = r4 + 1, f0 = dm(m0,i0);        // DM read instruction
```

2. Ensure that four instructions that do not use the affected registers are between the UREG register update and the DAG register load:

```
r4 = r1 + r2;                       // UREG register update
nop; nop;                           // ANY 2 instructions that do not use r4
jump Here (db);
nop;                                // Instruction that does not use r4
m0 = r4;                            // DAG register load
...                                 // Any number of instructions
Here: r4 = r4 + 1, f0 = dm(m0,i0);  // DM pre-modify read uses m0
```

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 Analog Devices, please consult the "Silicon Anomaly Tools Support" help page in the applicable documentation and release notes for details.

For all other tool chains and operating systems, see the appropriate supporting documentation for details.

APPLIES TO REVISION(S):

0.0, 0.2

22. 20000084 - Simultaneous OTP Accesses by Multiple Cores Can Cause Core Hang:

DESCRIPTION:
When multiple cores simultaneously access OTP memory, one of the cores may hang.

WORKAROUND:
There are two possible workarounds:

1. Dedicate a single core to make all OTP accesses.
2. If multiple cores require access to OTP memory, place such accesses in a critical region of code using a software solution such as Peterson's algorithm to guarantee mutual exclusion without deadlock. For example, if two cores require OTP access, declare a two-element Boolean array to track individual core requests to enter a critical region of code and an arbitration indicator between the two:

```
bool flag[2] = {false, false};  // Requests for two cores having OTP privileges
int turn;                       // Core priority indicator (0 = 1st core, 1 = 2nd core)
```

The first core with OTP privileges must then execute the following code to perform an OTP access:

```
flag[0] = true;                 // Set flag to request OTP access on 1st core
turn = 1;                       // Grant priority to 2nd core

while (flag[1] && turn == 1)    // If 2nd core has priority and is accessing OTP
{
   // Wait for 2nd core to complete OTP access
}

// Now, the 1st core is in the critical section
 <Perform OTP access here>      // Read or write
// End of critical section

flag[0] = false;                // Clear request for OTP access for 1st core
```

A second core requiring OTP access must then run complementary code to monitor and modify the shared variables governing the exclusivity of the OTP access:

```
flag[1] = true;                 // Set flag to request OTP access for 2nd core
turn = 0;                       // Grant priority to 1st core

while (flag[0] && turn == 0)    // If 1st core has priority and is accessing OTP
{
   // Wait for 1st core to complete OTP access
}

// Now, the 2nd core is in the critical section
 <Perform OTP access here>      // Read or write
// End of critical section

flag[1] = false;                // Clear request for OTP access for 2nd core
```

APPLIES TO REVISION(S):
0.0, 0.2

23. 20000085 - SPU Protection for Peripherals Doesn't Work Against Accesses from Cortex-A5 Core:

DESCRIPTION:
The SPU fails to protect against write accesses from the Cortex-A5 core master to the peripheral MMR regions. Any writes initiated by the Cortex-A5 core will reach the MMR region even if the SPU is properly configured to block them.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0, 0.2

24. 20000088 - High Leakage Current During Reset when HADC Is Used:

DESCRIPTION:
The HADC can draw current beyond specification when the device is held in reset ($\overline{\text{SYS\_HWRST}}$ is low) and the HADC input pins (**HADC_VINx**) are at different potential voltages.

The extra current is drawn from the sources driving the **HADC_VINx** input pins. The total current drawn from all the HADC inputs will depend on how many pins are connected to ground and how many are connected to higher potential. The worst case cumulative current through all the channels could be up to ~400mA; however, no individual driver will have to source the sum of all the leakage currents. The maximum current that any individual driver will have to source is ~121mA.

WORKAROUND:
When the processor is not in the reset state, the extra current is not drawn.

To avoid the extra current consumption while the processor is in the reset state, the following workarounds are possible:

1. Connect all the **HADC_VINx** inputs to ground, or leave them all floating. This is applicable only if the HADC is not used in the system, as it is likely not possible to ground the pins only while the processor is being reset.
2. Add a current-limiting resistor of up to 750 ohms to each of the HADC input channels to limit the leakage current during reset. The total resistance of 750 ohms is the sum of the series-limiting resistor and the output impedance of the circuit driving the respective **HADC_VINx** pin. This additional circuitry will not degrade the performance of the HADC.

APPLIES TO REVISION(S):
0.0, 0.2

25. 20000090 - Single-Ended Clock/DQS Measurements May Violate JESD79-3E/-2E Vix and VSWING Specs:

DESCRIPTION:
The Dynamic Memory Controller may fail to meet the JESD79-3E standard Vix(ac) specification for clock and DQS measurements in DDR3 mode. In DDR2 mode, it may fail to meet the JESD79-2E standard Vix(ac) specification for clock measurements and both the Vix(ac) and VSWING(Max) specifications for DQS measurements.

The Vix(ac) and VSWING(Max) specifications define single-ended requirements for differential signals. Since DDR2/DDR3 SDRAMs have true differential receivers and the processor meets all the differential requirements, failure to meet these single-ended specifications is negligible.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0, 0.2

26. 20000091 - Accesses to DMC_CPHY_CTL Register Do Not Function As Expected:

DESCRIPTION:
Configuring the **DMC_CPHY_CTL** register is required when initializing the DMC interface; however, no accesses to this register occur as expected:

1. Reads do not return the correct value and generate false data read errors. When initiated by a SHARC+ core, the respective core's data read interrupt (C1_IRQ0 or C2_IRQ0) is raised. When initiated by the ARM core, the synchronous data abort exception occurs.
2. Writes work as expected; however, a false data write error occurs. When initiated by a SHARC+ core, the respective core's data write interrupt (C1_IRQ1 or C2_IRQ1) is raised. When initiated by the ARM core, the synchronous data abort exception occurs.

WORKAROUND:
There is no workaround for read accesses. Do not read the **DMC_CPHY_CTL** register from any core.

When writing the **DMC_CPHY_CTL** register from a SHARC+ core, the application must await the subsequent false data write error interrupt and clear it. For example, the following code shows how to clear the false interrupt after writing to the **DMC_CPHY_CTL** register from SHARC+ core 1:

```
*pREG_DMC_CPHY_CTL = 0x1234;     // Write to register

// Wait for false data write error interrupt to latch
while((*pREG_SEC0_SSTAT33 & BITM_SEC_SSTAT_PND) != BITM_SEC_SSTAT_PND);

// Write SEC_SSTAT.PND bit to clear the false interrupt
*pREG_SEC0_SSTAT33 = (uint32_t)BITM_SEC_SSTAT_PND;

// Wait for the false interrupt to clear
while(*pREG_SEC0_SSTAT33 & BITM_SEC_SSTAT_PND);
```

When writing the **DMC_CPHY_CTL** register from the ARM core, do not enable the asynchronous abort exception. If the asynchronous abort exception must be enabled for other purposes, write the **DMC_CPHY_CTL** register from one of the SHARC+ cores instead and employ the described workaround to then clear the false error.

APPLIES TO REVISION(S):
0.0, 0.2

27. 20000093 - Power-Up Sequencing May Cause Pins to Be Unexpectedly Driven:

DESCRIPTION:
If VDD_EXT ramps before VDD_INT during power-on reset sequencing, some processor pins may be unexpectedly driven until the VDD_INT supply is within specification. The affected processor pins are:

1. $\overline{\text{SYS\_HWRST}}$
2. All JTAG pins (including $\overline{\text{JTG\_TRST}}$)
3. All GPIO pins
4. All DAI, MLB, TWI, and USB pins

If VDD_EXT is ramped before VDD_INT, dedicated input pins (e.g., $\overline{\text{SYS\_HWRST}}$ and $\overline{\text{JTG\_TRST}}$) can be unexpectedly driven as outputs, and I/O pins may be driven low or high rather than being tri-stated (as expected). When driven high, the voltage level seen on these pins will be that of the VDD_EXT supply. Regardless of whether the pin is being driven high or low, any affected pin will be tri-stated (as documented) once VDD_INT has ramped to within specification.

WORKAROUND:

1. Utilize weak pull-down resistors (<22 kOhm) on both the $\overline{\text{SYS\_HWRST}}$ and $\overline{\text{JTG\_TRST}}$ pins. Since the $\overline{\text{SYS\_HWRST}}$ signal needs to go high to take the processor out of reset, a push-pull reset supervisory IC is needed to drive the $\overline{\text{SYS\_HWRST}}$ reset pin.
2. Ensure that the VDD_INT supply is within specification prior to ramping up the VDD_EXT supply.

APPLIES TO REVISION(S):
0.0

28. 20000094 - SPDIF Receiver Output Clock Is Unreliable:

DESCRIPTION:
When operating properly, the SPDIF receiver output clock (SPDIF_RX_TDMCLK_O) frequency is 256 times the sampling rate. The SPDIF receiver, however, fails to maintain this relationship; thus, the SPDIF_RX_TDMCLK_O output clock is unreliable.

WORKAROUND:
Do not use the SPDIF_RX_TDMCLK_O output clock.

APPLIES TO REVISION(S):
0.0, 0.2

29. 20000096 - Type 18a USTAT Instructions Fail When Following Specific Code Sequence:

DESCRIPTION:
Type 18a ISA/VISA register bit manipulation instructions (**BIT SET**, **BIT CLR**, **BIT TGL**, **BIT TST**, and **BIT XOR**) using either USTAT register can fail when immediately following an external memory (EXT_MEM) or system MMR (SMMR) read-write sequence and a read of a core memory-mapped register (CMMR) involving the same USTAT register. Consider the following pseudo-code sequence:

```
1: USTAT# = dm(EXT_MEM|SMMR);  // EXT_MEM or SMMR read to USTAT1 or USTAT2
2: dm(EXT_MEM|SMMR) = USTAT#;  // EXT_MEM or SMMR write from the same USTAT register
3: USTAT# = dm(CMMR);          // CMMR read to the same USTAT register
4: bit <op> USTAT# <data32>;   // <op> = SET|CLR|TGL|TST|XOR, using the same USTAT register
```

In this code sequence, the type 18a instruction in line 4 erroneously performs the bit operation on the value loaded to the USTAT register in instruction 1 rather than performing the operation on the expected value loaded in instruction 3.

WORKAROUND:
Insert a **NOP;** instruction before the type 18a instruction in the above code sequence to avoid the issue.

APPLIES TO REVISION(S):
0.0, 0.2

30. 20000101 - SMPU Hang when Exclusive Read Arrives While Non-Exclusive Write Is Pending:

DESCRIPTION:
The System Memory Protection Unit (SMPU) holds exclusive reads of the associated memory space until all pending writes to that memory space complete. When an individual SMPU instance handles an exclusive read request from one core concurrently with a non-exclusive write request from a different master (core or DMA), that SMPU instance erroneously clears the pending write request while also correctly waiting for that write to complete.

Due to this anomaly, the write request being erroneously cleared leads to the non-exclusive write not taking place. In addition to the write not occurring to the target memory, the SMPU never receives the write complete required to initiate the pending exclusive read access. Because the exclusive read access to the memory is never launched, the SMPU deadlocks and causes a system hang due to the reading core indefinitely awaiting a read response for a read request that is still pending in the SMPU.

WORKAROUND:
This anomaly does not apply when:
1. the exclusive read access and non-exclusive write access are originated by an individual core, OR
2. when the exclusive access is a write access.

When two unique cores initiate the accesses that lead to the anomaly, mutual exclusion must be managed in software.

APPLIES TO REVISION(S):
0.0, 0.2

31. 20000108 - Factory Serial Number Cannot Be Read When Device Is Locked:

DESCRIPTION:
The Factory Serial Number (FSN) in the One-Time Programmable (OTP) memory space cannot be read when the device is locked.

WORKAROUND:
Prior to locking the device, application software must first read the FSN from OTP space and store it to OTP memory that can be accessed on a locked device, such as the General Purpose region described in the following table:

Name	Byte Address	Size (Bits)
General Purpose 1	0x300 + 0 - 0x33C	512

APPLIES TO REVISION(S):
0.0, 0.2

32. 20000123 - Boot failure with Ignore Block in Page Mode:

DESCRIPTION:
When page mode (ROM_BFLAG_PAGEMODE) is enabled in device boot, boot process will fail in the presence of ignore (BFLAG_IGNORE) block, when byte count from the target address of the payload, crosses the 1024 bytes page boundary due to loss of synchronization between source pointer and internal temp buffer. Also in non-secure host boot modes, when page mode is enabled, ignore block payload is processed from internal temp buffer instead of discarding it. This makes the boot rom fail to process ignore block as it is intended for. Therefore, the issue is applicable for the following cases:
1. Non Secure device and host boot modes with page mode enabled i.e. ROM_BFLAG_PAGEMODE is set in Global Flags in adi_rom_Boot().
2. Secure device boot.

WORKAROUND:
1. Do not use ignore blocks in the boot-stream for the above-mentioned cases.

APPLIES TO REVISION(S):
0.0, 0.2

©2023 Analog Devices, Inc. All rights reserved. Trademarks and registered trademarks are the property of their respective owners.