SHARC+ Single Core High Performance DSP (up to 1 GHz)

Silicon Anomaly List

ADSP-21562/21563/21565/21566/21567/21569

ABOUT ADSP-21562/21563/21565/21566/21567/21569 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC+® ADSP-21562/21563/21565/21566/21567/21569 product(s) and the functionality specified in the ADSP-21562/21563/21565/21566/21567/21569 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- and core-specific anomalies may not apply to all processors. See the table below for details. An "x" indicates that anomalies related to this peripheral/core apply only to the model indicated, and the list of specific anomalies for that peripheral/core appear in the rightmost column.

Feature	21562	21563	21565	21566	21567	21569	Anomalies
OSPI0				x	x	x	20000099
CGU1				x	x	x	20000106

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
04/02/2024	H	RevD	Added Anomalies 20000128, 20000129
04/07/2023	G	RevC	Added Anomaly 20000105, 20000123
02/22/2021	F	RevB	Added Datasheet Revision RevB
01/13/2021	E	RevA	Added Anomaly 20000100
08/05/2020	D	Rev0/PrG	Added Datasheet Revision PrG
04/07/2020	C	Rev0/PrF	Added Anomaly 20000106 Revised Anomalies for Clarity/Completeness
08/23/2019	B	PrE	Added Anomalies 20000103, 20000104 Revised Anomalies for Clarity/Completeness
10/23/2018	A	PrB	Initial Version

One Analog Way, Wilmington, MA 01887 U.S.A.

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-21562/21563/21565/21566/21567/21569 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	20000062	Writes to the SPI_SLVSEL Register Do Not Take Effect	x	x
5	20000069	PCSTK and MODE1STK Loads Do Not Occur If Next Instruction Is L2 or L3 Access	x	x
6	20000072	Floating-Point Computes Targeting F0 Register Can Cause Pipeline Stalls	x	x
7	20000096	Type 18a USTAT Instructions Fail When Following Specific Code Sequence	x	x
8	20000097	SMPU11 Does Not Block Write Accesses From Core or DMA Master	x	x
9	20000098	Boot ROM Does Not Properly Raise INTR_SOFT3 Fault	x	.
10	20000099	Dual Transfer Rate (DTR) OSPI0 Boot Failure	x	.
11	20000100	Invalid HADC_CHAN_MSK Register Reset State Causes Unexpected HADC Operation	.	x
12	20000103	Unreliable SPDIF Receiver Clock Output Pulse Width at Sample Rates Above 96KHz	x	x
13	20000104	Stale Reads of Prefetch Buffer When Cache Is Disabled	x	x
14	20000105	Housekeeping ADC (HADC) and Thermal Monitor Unit (TMU) Blocks Are Not Functional	x	.
15	20000106	DDR Clock (DCLK) Signal Generated Through CGU1 May Not Work As Expected	x	x
16	20000123	Boot failure with Ignore Block in Page Mode	x	x
17	20000128	Explicit core write to IRPTL register can cause pending FIRx/IIRx accelerator channel completion interrupt to be cleared	x	x
18	20000129	OTP Program API fails to program OTP memory correctly in presence of a leaky bit	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-21562/21563/21565/21566/21567/21569 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. 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

5. 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

6. 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

7. 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

8. 20000097 - SMPU11 Does Not Block Write Accesses From Core or DMA Master:

DESCRIPTION:
The system memory protection unit covering the SPI flash address memory space (SMPU11) is not fully functional for write accesses. It properly detects a write protection violation when a core or DMA master performs a write access to that space, but it does not block the write access itself as expected.

WORKAROUND:
Use write protection in the OSPI0 controller to block the writes to the flash memory address space, as described in this pseudo-code:

```
OSPI_WRPROT_UP  = Upper Flash Memory Address;  // Set upper memory boundary
OSPI_WRPROT_LWR = Lower Flash Memory Address;  // Set lower memory boundary
OSPI_WRPROT_CTL = 0x2;                         // Enable write protection
```

After write protection for the memory region is enabled, write protect the OSPI0 registers using the SPU to ensure this configuration is not unintentionally changed during run-time.

APPLIES TO REVISION(S):
0.0, 0.2

9. 20000098 - Boot ROM Does Not Properly Raise INTR_SOFT3 Fault:

DESCRIPTION:
The boot ROM's default error handler function fails to raise the INTR_SOFT3 fault signaling to the system that a boot error occurred prior to the boot ROM execution entering an endless loop.

WORKAROUND:
Utilize an initialization block as part of the loader stream to overwrite the default error handler function pointer in the boot ROM structure (**`ADI_ROM_BOOT_CONFIG::pErrorFunction`**) with a pointer to a custom error handler that raises the INTR_SOFT3 fault.

APPLIES TO REVISION(S):
0.0

10. 20000099 - Dual Transfer Rate (DTR) OSPI0 Boot Failure:

DESCRIPTION:
Power-on reset booting via the octal SPI controller (OSPI0) in single, dual, or quad data modes using Dual Transfer Rate (DTR) does not work properly. Due to a read setup timing violation, the OSPI0 controller fails to correctly sample the data read from the flash memory.

WORKAROUND:
1. Utilize an initialization block in the loader stream to update the delay field in the Read Delay Capture Register (RDCR) based on the tCLQV (Clock Low to Output Valid) specification in the flash memory datasheet. The initialization block can boot over OSPI0 in single, dual, or quad data modes using Single Transfer Rate (STR).
2. Program the **`ospi_read_data_capture`** field in OTP memory with the correct read capture delay based on the tCLQV specification in the flash memory datasheet.

APPLIES TO REVISION(S):
0.0

11. 20000100 - Invalid HADC_CHAN_MSK Register Reset State Causes Unexpected HADC Operation:

DESCRIPTION:
The reset state of the HADC Channel Mask (**`HADC_CHAN_MSK`**) register incorrectly enables unsupported HADC channels, which leads to erroneous HADC activity. For models supporting four HADC channels, unsupported channels 4-7 are erroneously enabled. For models supporting two HADC channels, unsupported channels 2-7 are erroneously enabled. The erroneously enabled channels generate false HADC interrupts/triggers and are included in the auto-scan process, thus also wasting HADC bandwidth.

WORKAROUND:
Prior to enabling the HADC block, ensure that the application sets the bits in the **`HADC_CHAN_MSK`** register that are associated with the unimplemented channels to explicitly disable them, specifically:

1. For 4-channel models, initially write 0x0000FFF0 to the **`HADC_CHAN_MSK`** register to enable only channels 0-3.
2. For 2-channel models, initially write 0x0000FFFC to the **`HADC_CHAN_MSK`** register to enable only channels 0-1.

While the application can also explicitly disable supported HADC channels by also setting the associated lower-order bits in the same write to properly initialize the full HADC block, subsequent dynamic manipulation of the **`HADC_CHAN_MSK`** register must utilize a read-modify-write operation to preserve the logic high state of the bits associated with the unsupported channels, as detailed above.

APPLIES TO REVISION(S):
0.2

12. 20000103 - Unreliable SPDIF Receiver Clock Output Pulse Width at Sample Rates Above 96KHz:

DESCRIPTION:
When the sampling rate of the SPDIF receiver input stream (FS_Rate) is above 96 KHz, the positive pulse width of the SPDIF receiver TDM output clock (SPDIF_RX_TDMCLK_O) can be as low as the period of the SPDIF receiver module clock, and the negative pulse width can be as high as one SPDIF receiver module clock period less than the ideal clock period. As a result, audio peripherals such as the ASRC, SPORT, and DAI pins may not function properly when SPDIF_RX_TDMCLK_O is used as the clock source.

WORKAROUND:
Do not use the SPDIF_RX_TDMCLK_O output clock as the source for external peripherals when the FS rate is above 96 KHz.

The Precision Clock Generator (PCG) can be used to divide the clock down such that audio peripherals like the ASRC, SPORT, and DAI pins may function internally; however, the clock and frame sync outputs from the PCG will still exhibit the duty cycle problem and must not be used to interface with external components.

APPLIES TO REVISION(S):
0.0, 0.2

13. 20000104 - Stale Reads of Prefetch Buffer When Cache Is Disabled:

DESCRIPTION:
When cache memory is disabled, core accesses to L2/L3 memory space may return stale data from the enabled DPORT prefetch buffer associated with L2/L3 address space if there is a pending write to the same L2/L3 address space.

WORKAROUND:
Do not enable the prefetch buffer unless cache is also enabled. When cache is enabled, core accesses to non-cached L2/L3 address space bypass the prefetch buffer, thus avoiding the issue.

APPLIES TO REVISION(S):
0.0, 0.2

14. 20000105 - Housekeeping ADC (HADC) and Thermal Monitor Unit (TMU) Blocks Are Not Functional:

DESCRIPTION:
The Housekeeping Analog-to-Digital Converter (HADC) and Thermal Monitor Unit (TMU) blocks are not functional.

WORKAROUND:
Do not use the HADC or TMU blocks.

APPLIES TO REVISION(S):
0.0

15. 20000106 - DDR Clock (DCLK) Signal Generated Through CGU1 May Not Work As Expected:

DESCRIPTION:
The Clock Distribution Unit (CDU0) interacts with the two Clock Generation Units (CGU0 and CGU1) to route various clocks in the device, including the DDR clock (DCLK). Due to this anomaly, the DCLK signal is not guaranteed across all operating conditions when derived from CGU1.

WORKAROUND:
Do not use CGU1 to generate DCLK. Generate the DCLK frequency from CGU0.

APPLIES TO REVISION(S):
0.0, 0.2

16. 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

17. 20000128 - Explicit core write to IRPTL register can cause pending FIRx/IIRx accelerator channel completion interrupt to be cleared:

DESCRIPTION:
Any BIT SET IRTPL <data32>; and BIT CLR IRTPL <data32>; instruction, regardless of the value of the argument, will clear any pending FIRx/IIRx accelerator channel completion interrupt latched in the IRPTL register when executed. This can result in FIRx/IIRx channel completion interrupt miss.

WORKAROUND:
1. Do not use BIT SET IRPTL <data32>; and BIT CLR IRPTL <data32>; instruction to generate or ignore interrupts, if FIRx/IIRx accelerators are used.
2. Use System Event Controller (SEC) to manage FIRx/IIRx accelerator channel completion interrupts if IRPTL must be explicitly written to manage other interrupts.

APPLIES TO REVISION(S):
0.0, 0.2

18. 20000129 - OTP Program API fails to program OTP memory correctly in presence of a leaky bit:

DESCRIPTION:
When using the OTP Write API i.e. adi_rom_otp_pgm() or adi_rom_otp_pgm_data(), the OTP memory controller fails to program a 16-bit short word offset in OTP memory correctly in presence of a leaky bit.
As the result of this, the OTP Read API i.e. adi_rom_otp_get() or adi_rom_otp_get_data() reads the same 16-bit short word address as 0x0 and does not match with the expected programmed value.

WORKAROUND:
1. Avoid using the OTP Write API provided in the Boot ROM. Contact Analog Devices for updated OTP Service library to be used with the development tool chain for programing the OTP memory space.

APPLIES TO REVISION(S):
0.0, 0.2