

## ABOUT ADSP-21261 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the SHARC ADSP-21261 product(s) and the functionality specified in the ADSP-21261 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]
0.1	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
09/16/2009	J	F	Added anomalies: <a href="#">06000028</a> , Added common note on Tools action for the anomalies <a href="#">06000012</a>
03/14/2008	I	0	Removed the information related to 0.0 revision and added information for 0.1 revision
12/28/2006	H	0	Removed anomalies: 06000027 - This information is added to the HRM
11/24/2006	G	0	Added anomalies: 06000027
10/17/2006	F	0	Document Format Update
08/17/2006	E	0	Modified anomalies: <a href="#">06000012</a>
06/21/2006	D	0	Modified anomalies: <a href="#">06000022</a>
03/14/2006	C	0	Added anomalies: <a href="#">06000022</a>

#### NR002914J

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.

## SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-21261 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.1
1	<a href="#">06000006</a>	FLG0-3 pins are unusable when CLK source for SPI Module is disabled	x
2	<a href="#">06000009</a>	Type 1 Instruction (Dual-access) failing if reference is to a memory mapped register	x
3	<a href="#">06000012</a>	Incorrect number of loop iterations for counter based loops with PMDA access	x
4	<a href="#">06000016</a>	Conditional RTI in SIMD not evaluated properly	x
5	<a href="#">06000018</a>	Pin Sensitivity (RD_B, DAI_P[20:1])	x
6	<a href="#">06000019</a>	AD15-0 Pins driven low during Reset	x
7	<a href="#">06000020</a>	Indirect jumps or calls followed by Long Word accesses using PM bus can vector to an unknown location	x
8	<a href="#">06000021</a>	Parallel Port Bus Contention	x
9	<a href="#">06000022</a>	SPI can generate spurious clock for an additional word when used in receive DMA mode at maximum SPICLK frequency	x
10	<a href="#">06000028</a>	Incorrect Popping of stacks possible when exiting IRQx/Timer Interrupts with DB modifiers	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-21261 including a description, workaround, and identification of applicable silicon revisions.

### 1. 06000006 - FLG0-3 pins are unusable when CLK source for SPI Module is disabled:

#### DESCRIPTION:

The ADSP-2126x DSP gives the user the ability to disable the on chip core clock to various peripheral modules in order to conserve power. This is configured in the PMCTL register. When the clock signal for the SPI module is disabled, the pins\* for FLG0-3 are unusable until the clock is re-enabled. These pins are used by the SPI module as device selects in SPI Master mode.

\*Note that the FLG0-3 pins share functionality with other signals, configurable in the SYSCTL Register. Pin function is as follows: FLG0 or IRQ0, FLG1 or IRQ1, FLG2 or IRQ2, FLG3 or TIMEXP. For more details on this functionality please consult the System Design Chapter of the ADSP-2126x Hardware Reference Manual.

#### WORKAROUND:

None

#### APPLIES TO REVISION(S):

0.1

### 2. 06000009 - Type 1 Instruction (Dual-access) failing if reference is to a memory mapped register:

#### DESCRIPTION:

When using a type1 instruction, if one of the instructions is accessing memory-mapped IOP registers, the access will fail. For example:

```
DM(I4,M5)=R9, PM(I12,M13)=R11 //Where I12 points to a memory-mapped register
```

The list of effected memporry-mapped registers is given below.

IOAS, IOAE, PSA1S, PSA1E, PSA2S, PSA2E, PSA3S, PSA3E, PSA4S, PSA4E, OSPID, EEMUIN, EEMUSTAT, EEMUOUT, SYSCTL, BRKCTL, REVPIID, DMA1S, DMA1E, DMA2S, DMA2E, PMA1S, PMA1E and EEMUSTAT.

#### WORKAROUND:

1. Do not use a dual-access instruction to access memory-mapped registers.

2. For Type 1 instructions where both accesses are writes, the anomaly can be avoided by using the DM bus to access the memory-mapped register. Multi-function instructions with writes only have only failed when the PM bus is used to access the memory-mapped registers. Reads will continue to fail regardless of bus used.

For example, both writes below will pass:

```
DM(I4,M5)=R9, PM(I12,M13)=R11 //Where I4 points to a memory-mapped register
//and I12 does not.
```

#### APPLIES TO REVISION(S):

0.1

**3. 0600012 - Incorrect number of loop iterations for counter based loops with PMDA access:**

---

**DESCRIPTION:**

There are a few scenarios where loop instructions do not execute as expected:

**Case 1:**

In the rare situation of a one instruction loop with only one iteration, with a PM data memory access, the loop will iterate twice instead of once.

**Case 2:**

In a one instruction loop with two iterations, with a PM data memory access, the loop will iterate three times instead of twice if the Cache Disable bit is set (CADIS bit in Mode2 register).

**Case 3:**

In a two instruction loop with one iteration, if the second instruction is a pm data memory access, the loop will iterate twice instead of once.

**Case 4:**

In a one instruction loop with three iterations, with a PM data memory access, the loop will iterate twice instead of three times if any interrupt related branching happens during the loop's termination period.

**Case 5:**

In a one instruction loop with four or more iterations, with a PM data memory access, the loop will iterate one less (LCNTR-1) time instead of LCNTR times if any interrupt related branching happens during the loop's termination period and if the Cache Disable bit is set (CADIS bit in the Mode2 register).

**WORKAROUND:****Case 1:**

Use the DM bus instead of the PM bus.

**Case 2:**

1. Use the DM bus instead of the PM bus.
2. Clear the CADIS bit in Mode2 (cleared by default).

**Case 3:**

1. Use the dm bus instead of the pm bus.
2. Move the PM access to the first instruction of the loop.

**Case 4:**

1. Use the DM bus instead of the PM bus.
2. Instead of using a single instruction loop with three iterations, explicitly write the instruction three times.

**Case 5:**

1. Clear the CADIS bit in the Mode2 register (cleared by default).
2. Use the DM bus instead of the PM bus.
3. Unroll the loop, for example one instruction 32 iteration loop can be changed to two instruction 16 iteration loop. There is no problem if the number of instructions is more than one in the loop.

**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

**4. 06000016 - Conditional RTI in SIMD not evaluated properly:**

---

**DESCRIPTION:**

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

```
IF COND RTI (DB); //Note that the DB modifier is not needed to cause the
                  //anomalous behavior.
```

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

**The DSP's behavior in executing the instruction is as follows:**

- 1) Branches to the address stored at the top of the PC stack.
- 2) Pops the status stack if the ASTATx/y and MODE1 status registers registers have been pushed - if the interrupt was IRQ2-0 or the timer interrupt.
- 3) Clears the appropriate bit in the interrupt mask pointer (IMASKP) register.

**The behavior seen with respect to the conditions in PEx and PEy are as follows:**

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

**WORKAROUND:**

None

**APPLIES TO REVISION(S):**

0.1

**5. 0600018 - Pin Sensitivity (RD\_B, DAI\_P[20:1]):****DESCRIPTION:**

There is some sensitivity to noise on the RD\_B, SPORTx\_CLK, and SPORTx\_FS signals of the ADSP-2126x processors. This has the potential to cause errors during parallel port read operations including booting, and sport issues when the sport is driving the sport clock & frame sync and an external device is driving the data. (The SPORT signals are routed through the signal routing unit (SRU) to the DAI\_P[20:1] pins.)

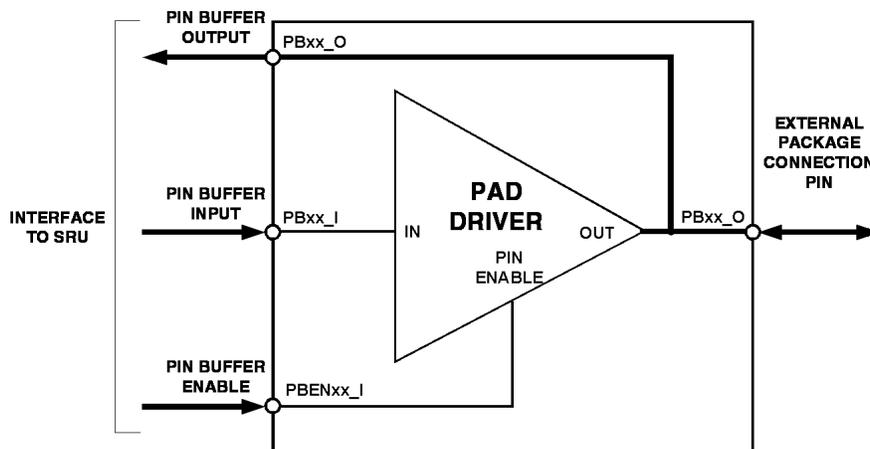
**WORKAROUND:****RD\_B:**

1. Failures were not seen in memory reads where the RD\_B trace length was kept to approximately 7 cm. However, this pin is still susceptible and external capacitance may be needed for memory reads to function properly with other board designs.

**SPORTx\_CLK & SPORTx\_FS:**

1. In certain cases, by programming the SRU CLK and FS registers, the reflection sensitivity issue in these signals can be avoided. Consider this Sport Clock example: (The Frame Sync configuration is analogous) In its default routing, the SRU maps the signal from the DAI pin (PBxx\_O) back to the Sport Clock Input (SPORTxCLK\_I), as well as routing the sport clk output (SPORTxCLK\_O) to the Pin Buffer Input (PBxx\_I). This connection of PBxx\_O to SPORTxCLK\_I opens a vulnerability to a glitch coming in even though the Sport is driving the clock as an output.

By programming the Signal Routing Unit (SRU) to remove this input path, we can avoid this vulnerability.



In the figure, by default SPORTxCLK\_I would be connected to PBxx\_O and SPORTxCLK\_O would be routed to PBxx\_I. The workaround is to leave the routing of SPORTCLKx\_I to PBxx\_I as before, providing the sport clock off-chip, but also routing it directly back to SPORTCLKx\_I to give the state machine it's signal. This effectively closes off the external access to SPORTCLK\_I. For more information about the SRU, consult the DAI Chapter of the ADSP-2126x Hardware Reference Manual.

2. Use the Precision Clock Generator or an external device to generate the Sport Clock and Frame Sync.

**APPLIES TO REVISION(S):**

0.1

**6. 06000019 - AD15-0 Pins driven low during Reset:**

---

**DESCRIPTION:**

The AD15-0 pins on the ADSP-2126x processor are driven to zero during and after reset. This can have negative implications for the system when first powered on as these pins can be configured to be used as GPIO flags. Using these pins as selects for other devices within the system could cause problems at system power-up.

**WORKAROUND:**

None

**APPLIES TO REVISION(S):**

0.1

## 7. 06000020 - Indirect jumps or calls followed by Long Word accesses using PM bus can vector to an unknown location:

### DESCRIPTION:

Indirect jumps or calls followed immediately by instructions with a Long Word access using the PM bus (or invalid instruction) can jump to an unexpected location.

For example, in the following sequences of instructions will cause the processor to vector to an unexpected location:  
(where  $Ic = I15-I8$ ,  $Md = M15-8$ )

```

| jump | (Md,Ic);
| call |

PM ( |Md,Ic      | ) = ureg(LW);
     |Ic,Md      |
     |<addr32>   |
     |<data32>,Ic|

| jump | (Md,Ic);
| call |

ureg    = PM ( |Md,Ic      | )(LW);
          |Ic,Md      |
          |<addr32>   |
          |<data32>,Ic|

| jump | (Md,Ic);
| call |

invalid instruction;
```

Note that anomaly will occur if the Call or Jump is conditional as well.

### WORKAROUND:

#### In the assembly applications:

1. Place one nop just after the jump or call instruction.
2. Use the DM bus instead of PM for the data access.
3. Use either a PC-relative Jump/Call, or use a direct address.

i.e. `jump (pc,12); -or- call (pc,12);`  
`jump (<addr32>); -or- call (<addr32>);`

4. If possible, do not use the optional (LW) modifier in the data access.

#### In C applications:

1. One situation that can cause this anomaly in many C/mixed applications is at the end of a program code. It is likely that the last instruction in an application will be an indirect jump that is a return from the last function linked. If the memory that follows such a jump is not initialised or is data this could trigger the anomaly. This can be avoided by inserting a NOP at the end of your application code.

This workaround can be implemented in the VisualDSP++ LDF by placing a FILL command and a ". = . + 2;" command after your code sections INPUT\_SECTIONS commands. e.g.:

```

seg_pmco
{
    INPUT_SECTIONS( (seg_pmco) (seg_pmco)
    FILL(0x0) // workaround 06000020 at the end of instruction memory
```

```
. = . +2;  
>seg_pmco
```

**Note:**

There must be a space on both sides of the "." .

**APPLIES TO REVISION(S):**

0.1

## 8. 0600021 - Parallel Port Bus Contention:

### DESCRIPTION:

A bus contention issue has been identified on the ADSP-2126x 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. (Timing information assumes a 200 MHz core clock rate)

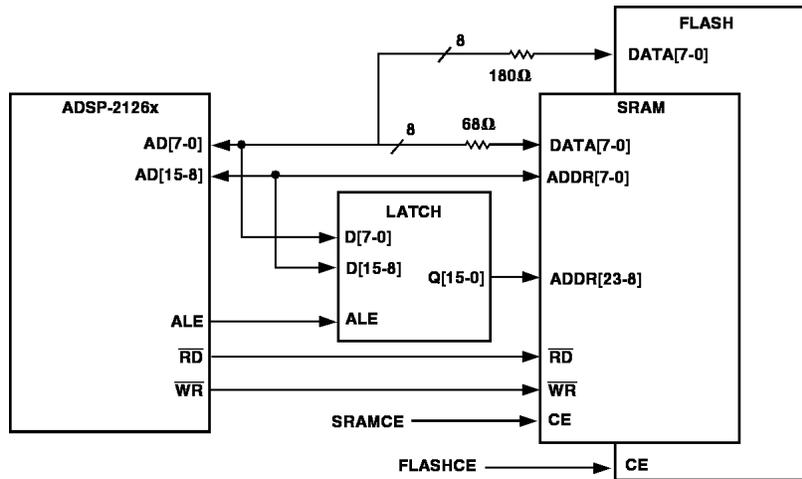
### WORKAROUND:

#### a) SRAM CONTENTION :

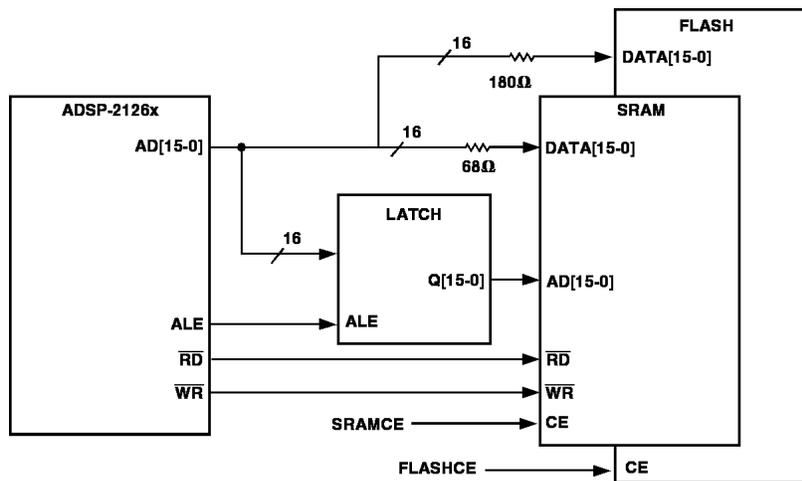
##### Hardware based work-around:

A typical 8ns SRAM will require 5-7ns to tri-state from /OE de-assertion. The ADSP-2126x 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-through 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-2126x 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

### 9. 0600022 - 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

### 10. 0600028 - 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