Blackfin
Embedded Processor

Silicon Anomaly List — ADSP-BF522/BF524/BF526(C)

# ABOUT ADSP-BF522/BF524/BF526(C) SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF522/BF524/BF526(C) product(s) and the functionality specified in the ADSP-BF522/BF524/BF526(C) 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 implementation field bits <15:0> of the DSPID core MMR register can be used to differentiate the revisions as shown below.

| Silicon REVISION | DSPID<15:0> |
|---|---|
| 0.2 | 0x0002 |

## APPLICABILITY

Each anomaly applies to specific silicon revisions. See Summary or Detailed List for affected revisions. Additionally, not all processors described by this anomaly list have the same feature set. Therefore, peripheral-specific anomalies may not apply to all processors. See the below table for details. An "x" indicates that anomalies related to this peripheral apply only to the model indicated, and the list of specific anomalies for that peripheral appear in the rightmost column.

| Peripheral | ADSP-BF526(C) | ADSP-BF524(C) | ADSP-BF522(C) | Anomalies |
|---|---|---|---|---|
| USB | x | x | | 05000483, 05000510 |
| Ethernet MAC | x | | | None |
| CODEC | x | x | x | 05000487 |

## 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 |
|---|---|---|---|
| 01/25/2016 | H | D | Added Anomalies: 05000510, 05000511<br>Revised Anomaly: 05000265<br>Removed Silicon Revision 0.1 |
| 03/14/2014 | G | D | Added Anomalies: 05000503, 05000506<br>Removed Silicon Revision 0.0 |
| 05/23/2011 | F | B | Added Anomalies: 05000487, 05000490, 05000491, 05000494, 05000498, 05000501 |
| 03/15/2010 | E | A | Added Silicon Revision 0.2<br>Added Anomalies: 05000119, 05000434, 05000473, 05000475, 05000477, 05000481, 05000483 |
| 08/14/2009 | D | 0 | Added Anomaly: 05000461 |

Document Feedback

**One Technology Way, P.O.Box 9106, Norwood, MA 02062-9106 U.S.A.**
**Tel: 781.329.4700 ©2016 Analog Devices, Inc. All rights reserved.**
**Technical Support www.analog.com**

## SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF522/BF524/BF526(C) anomalies and the applicable silicon revision(s) for each anomaly.

| No. | ID | Description | Rev 0.2 |
|---|---|---|---|
| 1 | 05000074 | Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported | x |
| 2 | 05000119 | DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops | x |
| 3 | 05000122 | Rx.H Cannot Be Used to Access 16-bit System MMR Registers | x |
| 4 | 05000245 | False Hardware Error from an Access in the Shadow of a Conditional Branch | x |
| 5 | 05000254 | Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock | x |
| 6 | 05000265 | Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks | x |
| 7 | 05000310 | False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory | x |
| 8 | 05000366 | PPI Underflow Error Goes Undetected in ITU-R 656 Mode | x |
| 9 | 05000405 | Lockbox SESR Firmware Does Not Save/Restore Full Context | x |
| 10 | 05000408 | Lockbox Firmware Memory Cleanup Routine Does not Clear Registers | x |
| 11 | 05000416 | Speculative Fetches Can Cause Undesired External FIFO Operations | x |
| 12 | 05000421 | TWI Fall Time (Tof) May Violate the Minimum $I^2C$ Specification | x |
| 13 | 05000422 | TWI Input Capacitance (Ci) May Violate the Maximum $I^2C$ Specification | x |
| 14 | 05000426 | Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors | x |
| 15 | 05000431 | Incorrect Use of Stack in Lockbox Firmware During Authentication | x |
| 16 | 05000434 | SW Breakpoints Ignored Upon Return From Lockbox Authentication | x |
| 17 | 05000443 | IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall | x |
| 18 | 05000461 | False Hardware Error when RETI Points to Invalid Memory | x |
| 19 | 05000473 | Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15 | x |
| 20 | 05000475 | Possible Lockup Condition when Modifying PLL from External Memory | x |
| 21 | 05000477 | TESTSET Instruction Cannot Be Interrupted | x |
| 22 | 05000481 | Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption | x |
| 23 | 05000483 | Possible USB Data Corruption When Multiple Endpoints Are Accessed by the Core | x |
| 24 | 05000487 | The CODEC Zero-Cross Detect Feature is not Functional | x |
| 25 | 05000490 | SPI Master Boot Can Fail Under Certain Conditions | x |
| 26 | 05000491 | Instruction Memory Stalls Can Cause IFLUSH to Fail | x |
| 27 | 05000494 | EXCPT Instruction May Be Lost If NMI Happens Simultaneously | x |
| 28 | 05000498 | CNT_COMMAND Functionality Depends on CNT_IMASK Configuration | x |
| 29 | 05000501 | RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes | x |
| 30 | 05000503 | SPORT Sign-Extension May Not Work | x |
| 31 | 05000506 | Hardware Loop Can Underflow Under Specific Conditions | x |
| 32 | 05000510 | USB Wakeup from Hibernate State Requires Re-Enumeration | x |
| 33 | 05000511 | Lower 16 Bits of CNT_COUNTER Register Do Not Update Properly | 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-BF522/BF524/BF526(C) including a description, workaround, and identification of applicable silicon revisions.

### 1. 05000074 - Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported:

**DESCRIPTION:**
A multi-issue instruction with dsp32shiftimm in slot 1 and a P register store in slot 2 is not supported. It will cause an exception.

The following type of instruction is not supported because the P3 register is being stored in slot 2 with a dsp32shiftimm in slot 1:

```
R0 = R0 << 0x1 || [ P0 ] = P3 || NOP;      // Not Supported - Exception
```

This also applies to rotate instructions:

```
R0 = ROT R0 by 0x1 || [ P0 ] = P3 || NOP;  // Not Supported - Exception
```

Examples of supported instructions:

```
R0 = R0 << 0x1 || [ P0 ] = R1 || NOP;
R0 = R0 << 0x1 || R1 = [ P0 ] || NOP;
R0 = R0 << 0x1 || P3 = [ P0 ] || NOP;
R0 = ROT R0 by R0.L || [ P0 ] = P3 || NOP;
```

**WORKAROUND:**
In assembly programs, separate the multi-issue instruction into 2 separate instructions. 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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):**
0.2

### 2. 05000119 - DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops:

**DESCRIPTION:**
After completion of a Peripheral Receive DMA, the DMAx_IRQ_STATUS:DMA_RUN bit will be in an undefined state.

**WORKAROUND:**
The DMA interrupt and/or the DMAx_IRQ_STATUS:DMA_DONE bits should be used to determine when the channel has completed running.

**APPLIES TO REVISION(S):**
0.2

## 3. 05000122 - Rx.H Cannot Be Used to Access 16-bit System MMR Registers:

**DESCRIPTION:**
When accessing 16-bit system MMR registers, the high half of the data registers may not be used.  If a high half register is used, incorrect data will be written to the system MMR register, but no exception will be generated. For example, this access would fail:

```
    W[P0] = R5.H;   // P0 points to a 16-bit System MMR
```

**WORKAROUND:**
Use other forms of 16-bit transfers when accessing 16-bit system MMR registers. For example:

```
    W[P0] = R5.L;   // P0 points to a 16-bit System MMR
    R4.L = W[P0];
    R3 = W[P0](Z);
    W[P0] = R3;
```

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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):**
0.2

## 4. 05000245 - False Hardware Error from an Access in the Shadow of a Conditional Branch:

### DESCRIPTION:

If a load accesses reserved or illegal memory on the opposite control flow of a conditional jump to the taken path, a false hardware error will occur.

The following sequences demonstrate how this can happen:

**Sequence #1:**
For the "predicted not taken" branch, the pipeline will load the instructions that sequentially follow the branch instruction that was predicted not taken. By the pipeline design, these instructions can be speculatively executed before they are aborted due to the branch misprediction. The anomaly occurs if any of the three instruction slots following the branch contain loads which might cause a hardware error:

```
BRCC X [predicted not taken]
R0 = [P0];     // If any of these three loads accesses non-existent
R1 = [P1];     // memory, such as external SDRAM when the SDRAM
R2 = [P2];     // controller is off, then a hardware error will result.
```

**Sequence #2:**
For the "predicted taken" branch, the one instruction slot at the destination of the branch cannot contain an access which might cause a hardware error:

```
BRCC X (BP)
Y: ...
   ...
X: R0 = [P0];  // If this instruction accesses non-existent memory,
               // such as external SDRAM when the SDRAM controller
               // is off, then a hardware error will result.
```

### WORKAROUND:

If you are programming in assembly, it is necessary to avoid the conditions described above.

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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):

0.2

## 5. 05000254 - Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock:

**DESCRIPTION:**
If a Timer is in PWM_OUT mode AND is clocked by an external clock as opposed to the system clock (i.e., clocked by a signal applied to either PPI_CLK or a flag pin) AND is in single-pulse mode (PERIOD_CNT = 0), then the generated pulse width may be off by +1 or -1 count. All other modes are not affected by this anomaly.

**WORKAROUND:**
The suggested workaround is to use continuous mode instead of the single-pulse mode. You may enable the timer and immediately disable it again. The timer will generate a single pulse and count to the end of the period before effectively disabling itself. The generated waveform will be of the desired length.

If PULSEWIDTH is the desired width, the following sequence will produce a single pulse:

```
TIMERx_CONFIG = PWM_OUT|CLK_SEL|PERIOD_CNT|IRQ_ENA;  // Optional: PULSE_HI|TIN_SEL|EMU_RUN
TIMERx_PERIOD = PULSEWIDTH + 2;                      // Slightly bigger than the width
TIMERx_WIDTH  = PULSEWIDTH;
TIMER_ENABLE  = TIMENx;
TIMER_DISABLE = TIMDISx;
<wait for interrupt (at end of period)>
```

**APPLIES TO REVISION(S):**
0.2

## 6. 05000265 - Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks:

**DESCRIPTION:**

A noisy board environment combined with slow input edge rates on external SPORT receive (RSCLK) and transmit clocks (TSCLK) may cause a variety of observable problems. When excessive noise occurs during high-frequency transitions on a slowly ramping RSCLK/TSCLK signal, it can cause an additional bit-clock with a short period due to high sensitivity of the clock input. A slow slew rate input allows any noise on the clock input around the switching point to cause the clock input to cross and re-cross the switching point. This oscillation can cause a glitch clock pulse in the internal logic of the serial port, which can result in numerous operational failures.

Problems which may be observed due to this glitch clock pulse include:

- In stereo serial modes, a frame sync may be missed, causing left/right data swapping.
- In multi-channel mode, multi-channel frame delay (MFD) counts may appear inaccurate or frames may be skipped.
- In normal (early) frame sync mode, received data words could be shifted right one bit. The MSB may be incorrectly captured in sign extension mode.
- In any mode, received or transmitted data words may appear to be partially right shifted if noise occurs on any input clocks between the start of frame sync and the last bit to be received or transmitted.

In Stereo Serial mode (bit 9 set in SPORTx_RCR2), unexpected high frequency transitions on RSCLK/TSCLK can cause the SPORT to miss rising or falling edges of the word clock. This causes left or right words of Stereo Serial data to be lost. This may be observed as a Left/Right channel swap when listening to stereo audio signals. The additional noise-induced bit-clock pulse on the SPORT's internal logic results in the FS edge-detection logic generating a pulse with a smaller width and, at the same time, prevents the SPORT from detecting the external FS signal during the next "normal" bit-clock period. The FS pulse with smaller width, which is the output of the edge-detection logic, is ignored by the SPORT's sequential logic. Due to the fact that the edge detection part of the frame sync logic was already triggered, the next "normal" RSCLK will not detect the change in RFS. In $I^2S$/EIAJ mode, this results in one stereo sample being detected/transferred as two left/right channels, and all subsequent channels will be word-swapped in memory.

In multi-channel mode, the MFD logic receives the extra sync pulse and begins counting early or double-counting (if the count has already begun). A MFD of zero can roll over to 15, as the count begins one cycle early.

In early frame sync mode, if the noise occurs on the driving edge of the clock the same cycle that FS becomes active, the FS logic receives the extra runt pulse and begins counting the word length one cycle early. The first bit will be sampled twice and the last bit will be skipped.

In all modes, if the noise occurs in any cycle after the FS becomes active, the bit counting logic receives the extra runt pulse and advances too rapidly. If this occurs once during a work unit, it will finish counting the word length one cycle early. The bit where the noise occurs will be sampled twice, and the last bit will be skipped.

While the above audio failures are possible signatures associated with this anomaly, numerous other failures are possible due to the internal logic being subjected to what amounts to an out-of-spec clock signal. Even though the external signal is within specification, the noise causes multiple transitions to be sensed where only one transition actually occurred, resulting in an out-of-spec clock being presented to the internal logic. This can lead to illegal logic states and/or incorrect advancement of state machines, which adversely affects the SPORT itself and synchronization with other logic units like the DMA engine. A number of failure scenarios may result from this, including misreported SPORT/DMA errors and unexpected DMA halts.

**WORKAROUND:**

1) Decrease the sensitivity to noise by increasing the slew rate of the bit clock or make the rise and fall times of serial bit clocks short, such that any noise around the transition produces a short duration noise-induced bit-clock pulse. This small high-frequency pulse will not have any impact on the SPORT or on the detection of the frame-sync. Sharpen edges as much as possible, if this is suitable and within EMI requirements.
2) If possible, use internally generated bit-clocks and frame-syncs.
3) Follow good PCB design practices. Shield RSCLK with respect to TSCLK lines to minimize coupling between the serial clocks.
4) Separate RSCLK, TSCLK, and Frame Sync traces on the board to minimize coupling which occurs at the driving edge when FS switches.

A specific workaround for problems observed in Stereo Serial mode is to delay the frame-sync signal such that noise-induced bit-clock pulses do not start processing the frame-sync. This can be achieved if there is a larger serial resistor in the frame-sync trace than the one in the bit-clock trace. Frame-sync transitions should not cross the 50% point until the bit-clock crosses the 10% of VDD threshold (for a falling edge bit-clock) or the 90% threshold (for a rising edge bit-clock).

To improve immunity to noise, optional hysteresis can be enabled for input pins by setting the appropriate bits in the PORTx_HYSTERESIS register.

**APPLIES TO REVISION(S):**

0.2

## 7. 05000310 - False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory:

**DESCRIPTION:**
Due to fetches near boundaries of reserved memory, a false Hardware Error (External Memory Addressing Error) is generated under the following conditions:

1) A single valid CPLB spans the boundary of the reserved space. For example, a CPLB with a start address at the beginning of L1 instruction memory and a size of 4MB will include the boundary to reserved memory.

2) Two separate valid CPLBs are defined, one that covers up to the byte before the boundary and a second that starts at the boundary itself. For example, one CPLB is defined to cover the upper 1kB of L1 instruction memory before the boundary to reserved memory, and a second CPLB is defined to cover the reserved space itself.

As long as both sides of the boundary to reserved memory are covered by valid CPLBs, the false error is generated. Note that this anomaly also affects the boundary of the L1_code_cache region if instruction cache is enabled. In other words, the boundary to reserved memory, as described above, moves to the start of the cacheable region when instruction cache is turned on.

**WORKAROUND:**
Leave at least 76 bytes free before any boundary with a reserved memory space. This will prevent false hardware errors from occurring.

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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):**
0.2

## 8. 05000366 - PPI Underflow Error Goes Undetected in ITU-R 656 Mode:

**DESCRIPTION:**
If the PPI port is configured in ITU-R 656 Output Mode, the FIFO Underrun bit (UNDR in PPI_STATUS) does not get set when a PPI FIFO underrun occurs. An underrun can happen due to limited bandwidth or the PPI DMA failing to gain access to the bus due to arbitration latencies.

**WORKAROUND:**
None.

**APPLIES TO REVISION(S):**
0.2

## 9. 05000405 - Lockbox SESR Firmware Does Not Save/Restore Full Context:

**DESCRIPTION:**
Embedding `asm("raise 2;");` to call authentication in C code is not recommended. Registers R0-R3 and P0-P2 are not saved in the SESR. The compiler will assume that any C code after `asm("raise 2;");` will still be able to use these registers safely, although they are overwritten inside the SESR.

**WORKAROUND:**
The following C code instruction, which informs the compiler that it is not safe to use the registers R0-R3 and P0-P2, may be used to begin authentication:

```
asm("raise 2;":::"R0", "R1", "R2", "R3", "P0", "P1", "P2");
```

When using assembly code, the user must save the registers R0-R3 and P0-P2, issue the `raise 2;` instruction, then restore the registers R0-R3 and P0-P2.

**APPLIES TO REVISION(S):**
0.2

## 10. 05000408 - Lockbox Firmware Memory Cleanup Routine Does not Clear Registers:

**DESCRIPTION:**
The security firmware memory clear routine does not clear processor registers. It only clears on-chip SRAM memory.

Sensitive information may remain in processor registers upon exiting from Secure Mode, so users must not rely on the firmware memory clear routine to clear registers.

**WORKAROUND:**
Users should clear out processor registers prior to exiting Secure Mode. The security firmware memory clear routine may be called to clear on-chip SRAM or users may substitute their own memory clear routine instead.

**APPLIES TO REVISION(S):**
0.2

## 11. 05000416 - Speculative Fetches Can Cause Undesired External FIFO Operations:

**DESCRIPTION:**

When an external FIFO device is connected to an asynchronous memory bank, memory accesses can be performed by the processor speculatively, causing improper operations because the FIFO will provide data to the Blackfin, and the data will be dropped whenever the fetch is made speculatively or if the speculative access is canceled. "Speculative" fetches are reads that are started and killed in the pipeline prior to completion. They are caused by either a change of flow (including an interrupt or exception) or when performing an access in the shadow of a branch. This behavior is described in the Blackfin Programmer's Reference.

Another case that can occur is when the access is performed as part of a hardware loop, where a change of flow occurs from an exception. Since exceptions can't be disabled, the following example shows how an exception can cause a speculative fetch, even with interrupts disabled:

```
CLI R3;                            /* Disable Interrupts */
LSETUP( loop_s, loop_e) LC0 = P2;
   loop_s: R0 = W[P0];             /* Read from a FIFO Device  */
   loop_e: W[P1++] = R0;           /* Write that Generates a Data CPLB Page Miss */
STI R3;                            /* Enable Interrupts */
RTS;
```

In this example, the read inside the hardware loop is made to a FIFO with interrupts disabled. When the write inside the loop generates a data CPLB exception, the read inside the loop will be done speculatively.

**WORKAROUND:**

First, if the access is being performed with a core read, turn off interrupts prior to doing the core read. The read phase of the pipeline must then be protected from seeing the read instruction before interrupts are turned off:

```
CLI R0;
NOP; NOP; NOP;  /* Can Be Any 3 Instructions */
R1 = [P0];
STI R0;
```

To protect against an exception causing the same undesired behavior, the read must be separated from the change of flow:

```
CLI R3;                           /* Disable Interrupts */
LSETUP( loop_s, loop_e) LC0 = P2;
   loop_s: NOP;                   /* 2 NOPs to Pad Read */
           NOP;
           R0 = W[P0];
   loop_e: W[P1++] = R0;
STI R3;                           /* Enable Interrupts */
RTS;
```

The loop could also be constructed to place the NOP padding at the end:

```
LSETUP( .Lword_loop_s, .Lword_loop_e) LC0 = P2;
   .Lword_loop_s: R0 = W[P0];
                  W[P1++] = R0;
                  NOP;          /* 2 NOPs to Pad Read */
   .Lword_loop_e: NOP;
```

Both of these sequences prevent the change of flow from allowing the read to execute speculatively. The 2 inserted NOPs provide enough separation in the pipeline to prevent a speculative access. These NOPs can be any two instructions.

Reads performed using a DMA transfer do not need to be protected from speculative accesses.

**APPLIES TO REVISION(S):**

0.2

## 12. 05000421 - TWI Fall Time (Tof) May Violate the Minimum I$^2$C Specification:

**DESCRIPTION:**
TWI Fall Time (Tof) may be less than the minimum I$^2$C specification. This is not a functional issue, but may have an impact on signal integrity.

**WORKAROUND:**
None

**APPLIES TO REVISION(S):**
0.2

## 13. 05000422 - TWI Input Capacitance (Ci) May Violate the Maximum I$^2$C Specification:

**DESCRIPTION:**
TWI input capacitance (Ci) may be may be more than the maximum I$^2$C specification. This is not a functional issue, but may have an impact on signal integrity.

**WORKAROUND:**
None

**APPLIES TO REVISION(S):**
0.2

## 14. 05000426 - Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors:

### DESCRIPTION:

A false hardware error is generated if there is an indirect jump or call through a pointer which may point to reserved or illegal memory on the opposite control flow of a conditional jump to the taken path. This commonly occurs when using function pointers, which can be invalid (e.g., set to -1). For example:

```
    CC = P2 == -0x1;
    IF CC JUMP skip;
    CALL (P2);
  skip:
    RTS;
```

Before the IF CC JUMP instruction can be committed, the pipeline speculatively issues the instruction fetch for the address at -1 (0xffffffff) and causes the false hardware error. It is a false hardware error because the offending instruction is never actually executed. This can occur if the pointer use occurs within two instructions of the conditional branch (predicted not taken), as follows:

```
BRCC X [predicted not taken]
Y: JUMP (P-reg);  // If either of these two p-regs describe non-existent
   CALL (P-reg);  // memory, such as external SDRAM when the SDRAM
X: RTS;           // controller is off, then a hardware error will result.
```

### WORKAROUND:

If instruction cache is on or the ICPLBs are enabled, this anomaly does not apply.

If instruction cache is off and ICPLBs are disabled, the indirect pointer instructions must be 2 instructions away from the branch instruction, which can be implemented using NOPs:

```
BRCC X [predicted not taken]
Y: NOP;          // These two NOPs will properly pad the indirect pointer
   NOP;          // used in the next line.
   JUMP (P-reg);
   CALL (P-reg);
X: RTS;
```

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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):

0.2

## 15. 05000431 - Incorrect Use of Stack in Lockbox Firmware During Authentication:

**DESCRIPTION:**
Some of the cryptographic routines utilized by the security firmware use the stack in ways that do not conform to the run-time execution model by using stack locations that reside both inside and outside the currently allocated stack frame. This causes problems when an interrupt occurs during authentication and the interrupt handler pushes data onto the stack thus possibly overwriting live data. When live data is overwritten by the interrupt handler, the behavior of the security firmware is undefined upon return from interrupt.

This anomaly can manifest itself as a processor hang. The corruption of the stack when an interrupt is serviced during authentication can result in the Program Counter not being properly returned from the Lockbox SESR firmware. There is no evidence of this anomaly resulting in a security compromise. Issuing reset may be required to recover from this anomalous behavior.

**WORKAROUND:**
There are several workarounds possible:

1) Interrupt handlers that occur during authentication should wrap themselves with the following two instructions:

```
/* Start of original interrupt handler */
SP += -60;
<body of interrupt handler>
SP += 60;
RTI;
/* End of original interrupt handler */
```

2) When using the Device Driver/System Services Libraries (DD/SS) distributed with VisualDSP++, the DD/SS libraries will have to be rebuilt from source with the fix presented above implemented. For example, to implement the fix in the version of the DD/SS distributed with VisualDSP++ 5.0 update 3, the following steps must be taken:

   a) In /Blackfin/lib/src/services/int/adi_int_module.h, add the *SP += -60;* \ instruction immediately after the __STARTFUNC(Name) entry point for both ADI_INT_ISR_FUNCTION and ADI_INT_EXC_FUNCTION. For example:

```
#define ADI_INT_ISR_FUNCTION(Name,IVG,Nested) \
   __STARTFUNC(Name) \
      SP += -60; \                         // <-- ADD THIS INSTRUCTION
      [--SP] = R0; \
      [--SP] = P1; \
```

   In the same file, increase the length of the adi_int_ISR_Entry array from 16 to 18.

   b) In /Blackfin/lib/src/services/int/adi_int_asm.asm, the complementary stack modification must be made to the end of the handlers. The *SP += 60;* \ instruction must be inserted before the RTI in ADI_INT_ISR_EPILOG and before the RTX in ADI_INT_EXC_EPILOG. For example:

```
#define ADI_INT_EXC_EPILOG(Nested) \
   unlink;\
   SP +=  12;\
      .
      .
      .
   SP += 60; \                             // <-- ADD THIS INSTRUCTION
   RTX;\
   NOP;\
   NOP;\
   NOP;
```

   c) Rebuild the DD/SS library with these changes and use them instead of the prebuilt libraries distributed with VisualDSP++.

3) When using the Device Driver/System Services Libraries (DD/SS) distributed with VisualDSP++, the DD/SS libraries will have to be rebuilt from source with the fix presented above implemented. For example, to implement the fix in the version of the DD/SS distributed with VisualDSP++ 5.0 update 8, the following steps must be taken:

   a) In /Blackfin/lib/src/services/int/adi_int_module.h, add the *SP += -60;* \ instruction immediately after the __STARTFUNC(Name) entry point for both ADI_INT_ISR_FUNCTION and ADI_INT_EXC_FUNCTION. For example:

```
#define ADI_INT_ISR_FUNCTION(Name,IVG,Nested) \
   __STARTFUNC(Name) \
      SP += -60; \                         // <-- ADD THIS INSTRUCTION
      [--SP] = R0; \
      [--SP] = P1; \
```

b) In the same file, the complementary stack modification must be made to the end of the handlers. The *SP += 60; * instruction must be inserted before the RTI in ADI_INT_ISR_EPILOG and before the RTX in ADI_INT_EXC_EPILOG. For example:

```
#define ADI_INT_EXC_EPILOG(Nested) \
   unlink;\
   SP +=  12;\
      .
      .
      .
   SP += 60; \                             // <-- ADD THIS INSTRUCTION
   RTX;\
   NOP;\
   NOP;\
   NOP;
```

c) Rebuild the DD/SS library with these changes and use them instead of the prebuilt libraries distributed with VisualDSP++.

4) Do not enable interrupts to be serviced during authentication.

**APPLIES TO REVISION(S):**
0.2

## 16. 05000434 - SW Breakpoints Ignored Upon Return From Lockbox Authentication:

**DESCRIPTION:**
Upon returning from a failed Lockbox authentication attempt, software breakpoints are not able to halt the debugger until after the fourth breakpoint is executed.

This anomaly occurs when Condition #1 AND Condition #2 OR Condition #3 are met:

1) The code initiating the authentication by executing instruction "RAISE 2;" is executing from L1 Code Cache memory configured as SRAM.

and

2) The failure from authentication is due to a message-digital signature-message size mismatch.

or

3) The failure from authentication is due to fact that the public key is not programmed and the firmware reads back all 0's.

Note that if the combination of Condition 1) and either Condition 2) or Condition 3) are not met, the anomaly will not be encountered.

**WORKAROUND:**
There are several workarounds possible:

1) Use a hardware breakpoint instead of a software breakpoint to break right after returning from authentication.
2) Do not initiate authentication from L1 Cache Code area of memory
3) Place multiple (at least four) NOPs after the return point of authentication and place a regular software breakpoint at each NOP. The first four will not execute as expected but the fifth (5th) breakpoint will trigger the debugger to halt. NOPs may be replaced with non-critical code if desired.
4) Do not link the calling routine that executes the instruction "RAISE 2;" that initiates authentication into L1 Cache Code space.

**APPLIES TO REVISION(S):**
0.2

## 17. 05000443 - IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall:

**DESCRIPTION:**

If the IFLUSH instruction is placed on a loop end, the processor will stall indefinitely. For example, the following two code examples will never exit the loop:

```
P1 = 2;
LSETUP (LOOP1_S, LOOP1_E) LC1 = P1;
LOOP1_S: NOP;
LOOP1_E: IFLUSH[P0++];

LSETUP (LOOP2_S, LOOP2_E) LC1 = P1;
LOOP2_S: NOP; NOP; NOP; NOP;       // Any number of instructions...
LOOP2_E: IFLUSH[P0++];
```

**WORKAROUND:**

Do not place the IFLUSH instruction at the bottom of a hardware loop. If the IFLUSH is padded with any instruction at the bottom of the loop, the problem is avoided:

```
LSETUP (LOOP_S, LOOP_E) LC1 = P1;
LOOP_S: IFLUSH[P0++];
LOOP_E: NOP;                     // Pad the loop end
```

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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):**

0.2

## 18. 05000461 - False Hardware Error when RETI Points to Invalid Memory:

**DESCRIPTION:**

When using CALL/JUMP instructions targeting memory that does not exist, a hardware error condition will be triggered. If interrupts are enabled, the Hardware Interrupt (IRQ5) will fire. Since the RETI register will have an invalid location in it, it must be changed before executing the RTI instruction, even if servicing a different interrupt. Consider the following sequence:

```
P2.L = LO (0xFFAFFFFC);  // Load Address in Illegal Memory to P2
P2.H = HI (0xFFAFFFFC);
CALL(P2);                // Call to Bad Address Generates Hardware Error IRQ5
....

IRQ5_code:               // Hardware Error Interrupt Routine
RAISE 14;                // (1)
RTI;                     // (2)

IRQ14_code:
[--SP] = ( R7:0, P5:0 ); // (3)
[--SP] = RETI;           // (4)
....
```

When the hardware error occurs, the program counter points to the invalid location 0xFFAFFFFC, which is loaded into the RETI register during the service of the IRQ5 hardware error event. When the RTI instruction (2) is executed, a fetch of the instruction pointed to by the RETI register, which is an illegal address, is requested before hardware sees the level 14 interrupt pending. This fetch causes another hardware error to be latched, even though this instruction is not executed. Execution will go to IRQ14 (3). As soon as interrupts are re-enabled (4), the pending hardware error will fire.

**WORKAROUND:**

1) Ensure that code doesn't jump to or call bad pointers.
2) Always set the RETI register when returning from a hardware error to something that will not cause a hardware error on the memory fetch.

**APPLIES TO REVISION(S):**

0.2

## 19. 05000473 - Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15:

**DESCRIPTION:**
A SPORT receive underflow error can be erroneously triggered when the SPORT serial length is greater than 16 bits and an interrupt occurs as the access is initiated to the 32-bit SPORTx_RX register. Internally, two accesses are required to obtain the 32-bit data over the internal 16-bit Peripheral Access Bus, and the anomaly manifests when the first half of the access is initiated but the second is held off due to the interrupt. Application code vectors to service the interrupt and then issues the read of the SPORTx_RX register again when it subsequently resumes execution after the interrupt has been serviced. The previous read that was interrupted is still pending awaiting the second half of the 32-bit access, but the SPORT erroneously sends out two requests again. The first access completes the previous transaction, and the second access generates the underflow error, as it is now attempting to make a read when there is no new data present.

**WORKAROUND:**
The anomaly does not apply when using valid serial lengths up to 16 bits, so setting SLEN < 16 is one workaround.

When the length of the serial word is 17-32 bits (16 <= SLEN < 32), accesses to the SPORTx_RX register must not be interrupted, so interrupts must be disabled around the read. In C:

```
int temp_IMASK;

temp_IMASK = cli();
RX_Data = *pSPORT0_RX;
sti(temp_IMASK);
```

In assembly:

```
P0.H = HI(SPORT0_RX);
P0.L = LO(SPORT0_RX);

CLI R0;
R1 = [P0];
STI R0;
```

**APPLIES TO REVISION(S):**
0.2

## 20. 05000475 - Possible Lockup Condition when Modifying PLL from External Memory:

**DESCRIPTION:**
Synchronization logic in the EBIU can get corrupted if PLL alterations are made by code that resides in external memory. When this occurs, an infinite stall will occur, and the part will need to be reset. The lockup is dependent on what the original ratio was, what the new ratio is, and other factors, thus making it impossible to specify any cases where this is safe.

**WORKAROUND:**
The CCLK::SCLK ratio should not be changed via the external interface, whether it's from asynchronous memory or SDRAM. Only make modifications to the PLL_CTL and PLL_DIV registers from code executing in on-chip memory.

**APPLIES TO REVISION(S):**
0.2

## 21. 05000477 - TESTSET Instruction Cannot Be Interrupted:

**DESCRIPTION:**
When the TESTSET instruction gets interrupted, the write portion of the TESTSET may be stalled until after the interrupt is serviced. After the ISR completes, application code continues by reissuing the previously interrupted TESTSET instruction, but the pending write operation is completed prior to the new read of the TESTSET target data, which can lead to deadlock conditions.

For example, in a multi-threaded system that utilizes semaphores, thread A checks the availability of a semaphore using TESTSET. If this original TESTSET operation tested data with a low byte of zero (signifying that the semaphore is available), then the write portion of TESTSET sets the MSB of the low byte to 1 to lock the semaphore. When this anomaly occurs, the write doesn't happen until TESTSET is re-issued after the interrupt is serviced. Therefore, thread A writes the byte back out with the lock bit set and then immediately reads that value back, now erroneously indicating that the semaphore is locked. Provided the semaphore was actually still free when TESTSET was reissued, this means that the semaphore is now permanently locked because thread A thinks it was locked already, and any other threads that subsequently pend on the same semaphore are being locked out by thread A, which will now never release it.

**WORKAROUND:**
The TESTSET instruction must be made uninterruptible to avoid this condition:

```
CLI R0;
TESTSET(P0);
STI R0;
```

There is no workaround other than this, so events that cannot be made uninterruptible, such as an NMI or an Emulation event, will always be sensitive to this issue. Additionally, due to the need to disable interrupts, User Mode code cannot implement this workaround.

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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):**
0.2

## 22. 05000481 - Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption:

**DESCRIPTION:**
Reading the ITEST_COMMAND or ITEST_DATA registers will erroneously trigger a write to these registers in addition to reading the current contents of the register. The erroneous write does not update the read state of the register, however, the data written to the register is acquired from the most recent MMR write request, whether the most recent MMR write request was committed or speculatively executed. The bogus write can set either register to perform unwanted operations that could result in:

1) Corrupted instruction L1 memory and/or instruction TAG memory.
and/or
2) Garbled instruction fetch stream (stale data used in place of new fetch data).

**WORKAROUND:**
Never read ITEST_COMMAND or ITEST_DATA. The only exception to this strict workaround is in the case of performing the read atomically and immediately after a write to the same register. In this case, the erroneous write will still occur, but it will be with the exact same data as the intentional write that preceded it.

**APPLIES TO REVISION(S):**
0.2

## 23. 05000483 - Possible USB Data Corruption When Multiple Endpoints Are Accessed by the Core:

**DESCRIPTION:**

When using core transfers to fetch data from endpoint FIFOs with multiple endpoints enabled, data corruption can occur when the core is reading data from one endpoint FIFO and a change in code flow occurs immediately prior to this read committing in the pipeline. If this code accesses a different endpoint FIFO, the core will read the data from the different endpoint FIFO; however, when the application resumes with the read access to the previous endpoint FIFO that did not commit, the read is corrupted.

Note that this change in code flow could be due to a different endpoint interrupt, or the read could be speculatively executed in the shadow of a conditional branch. Exceptions can also cause this problem, but only in the unusual case where an access to an endpoint FIFO takes place in the exception handler. The most likely scenario for this corruption to occur is when the core is reading data from one endpoint and gets interrupted to service a different endpoint.

**WORKAROUND:**

1) Use the USB DMA to read from the Endpoint FIFOs.
2) When multiple Endpoint FIFOs are enabled, disable interrupts around reads from an endpoint FIFO.
3) Ensure that Endpoint FIFO reads do not occur in the shadow of a conditional branch by placing three NOPs between the branch instruction and the read.

**APPLIES TO REVISION(S):**

0.2

## 24. 05000487 - The CODEC Zero-Cross Detect Feature is not Functional:

**DESCRIPTION:**

The DAC zero-cross detect feature is not functional. Therefore, bit 7 of registers 2 and 3 should always be set to b#0.

**WORKAROUND:**

None

**APPLIES TO REVISION(S):**

0.2

## 25. 05000490 - SPI Master Boot Can Fail Under Certain Conditions:

### DESCRIPTION:

Master Mode SPI Booting can fail under certain combinations of SPI_BAUD, CCLK::SCLK ratio, boot block size, and SDRAM Refresh Rate. The root cause for this problem is described in anomaly 05-00-0501. This is the manifestation of that anomaly due to the boot ROM sequence, which does not incorporate the software workaround to the underlying hardware problem of a stuck RXS bit in the SPI_STAT register.

When the RXS bit gets stuck as a result of anomaly 05-00-0501, a subsequent re-enabling of the SPI port results in DMA requests to a FIFO that has not yet been populated. This causes bogus data retrieved at the end of the previous block to be interpreted by the boot ROM as an invalid block header for the next block, which causes the boot to abort. There are two places in the boot ROM where the device is susceptible to this, manifesting in one of three ways:

1) When a bootable image block size exceeds 64K, it is broken into multiple DMA work units. In the DMA handler invoked between the work units, the anomaly can be encountered.

2) When SPI_BAUD = 2, the maximum SPI baud rate of SCLK/4 aligns exactly with the boot ROM execution frequency, which allows for the SPI disable to align exactly with a word being received (as a result of the SPI's behavior to continue issuing clocks even after the RX DMA is completed). At this particular baud rate, the SPI issues exactly 40 clocks between when the DMA completes and when the SPI is disabled in the ROM. This equates to exactly 5 additional received bytes, which completely fills the 4-deep SPI RX FIFO and the shift register, which asserts the RXS bit as the SPI is shut down.

3) When system timing parameters allow for any single word to get transferred from the shift register to the SPI FIFO exactly as the SPI port is being shut down, the anomaly can theoretically be encountered, though it has not been observed on silicon or in simulations. All of the system timing parameters mentioned above must combine to cause the timing that triggers the anomaly.

### WORKAROUND:

For case 1), do not allow block sizes over 64K.

For case 2), using any SPI_BAUD setting other than 2 avoids the timing required to encounter the anomaly.

For case 3), if the problem were to occur, the behavior would be consistent and repeatable. Changing any of the SPI_BAUD, CCLK::SCLK ratio, block sizes, and/or SDRAM refresh rate will alter the timing (as aligned to boot ROM execution) such that the problem can be avoided.

A 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 (VisualDSP++, VDK, CrossCore Embedded Studio, the GNU Tool Chain, and the Linux kernel), 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):

0.2

## 26. 05000491 - Instruction Memory Stalls Can Cause IFLUSH to Fail:

**DESCRIPTION:**
When an instruction memory stall occurs when executing an IFLUSH instruction, the instruction may fail to invalidate a cache line. This could be a problem when replacing instructions in memory and could cause stale, incorrect instructions in cache to be executed rather than initiating a cache line fill.

**WORKAROUND:**
Instruction memory stalls must be avoided when executing an IFLUSH instruction. By placing the IFLUSH instruction in L1 memory, the prefetcher will not cause instruction cache misses that could cause memory stalls. In addition, padding the IFLUSH instruction with NOPs will ensure that subsequent IFLUSH instructions do not interfere with one another, and wrapping SSYNCs around it ensures that any fill/victim buffers are not busy. The recommended routine to perform an IFLUSH is:

```
SSYNC;          // Ensure all fill/victim buffers are not busy
LSETUP (LS, LE)
LS:  IFLUSH;
     NOP;
     NOP;
LE:  NOP;
SSYNC;          // Ensure all fill/victim buffers are not busy
```

Since this loop is four instructions long, the entire loop fits within one loop buffer, thereby turning off the prefetcher for the duration of the loop and guaranteeing that successive IFLUSH instructions do not interfere with each other.

**APPLIES TO REVISION(S):**
0.2

## 27. 05000494 - EXCPT Instruction May Be Lost If NMI Happens Simultaneously:

**DESCRIPTION:**

A software exception raised by issuing the EXCPT instruction may be lost if an NMI event occurs simultaneous to execution of the EXCPT instruction. When this precise timing is met, the program sequencer believes it is going to service the EXCPT instruction and prepares to write the address of the next sequential instruction after the EXCPT instruction to the RETX register. However, the NMI event takes priority over the Exception event, and this address erroneously goes to the RETN register. As such, when the NMI event is serviced, program execution incorrectly resumes at the instruction after the EXCPT instruction rather than at the EXCPT instruction itself, so the software exception is lost and is not recoverable.

**WORKAROUND:**

Either do not use NMI or protect against this lost exception by forcing the exception to be continuously re-raised and verified in the exception handler itself. For example:

```
EXCPT 0;
JUMP -2;  // add this jump -2 after every EXCPT instruction
```

Then, in the exception handler code, read the EXCAUSE field of the SEQSTAT register to determine the cause of the exception.
If EXCAUSE < 16, the handler was invoked by execution of the EXCPT instruction, so the RETX register must then be modified to skip over the JUMP -2 that was inserted in the workaround code:

```
R2 = SEQSTAT;
R2 <<= 0x1A;
R2 >>= 0x1A;   // Mask Everything Except SEQSTAT[5:0] (EXCAUSE)
R1 = 0xF (Z);
CC = R2 <= R1; // Check for EXCAUSE < 16
IF !CC JUMP CONTINUE_EX_HANDLER;
R2 = RETX;
R2 += 2;       // Modify RETX to Point to Instruction After Inserted JUMP -2;
RETX = R2;
JUMP END_EX_HANDLER;

CONTINUE_EX_HANDLER: // Rest of Exception Handler Code Goes Here
    .
    .
    .
END_EX_HANDLER: RTX;
```

In this fashion, the JUMP -2 guarantees that the soft exception is re-raised when this anomaly occurs. When the NMI does not occur, the above exception handler will redirect the application code to resume after the JUMP -2 workaround code that re-raises the exception.

A 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 (VisualDSP++, VDK, the GNU Tool Chain, and the Linux kernel), 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):**

0.2

## 28. 05000498 - CNT_COMMAND Functionality Depends on CNT_IMASK Configuration:

**DESCRIPTION:**
The counter's "Zero Once" mode is only functional if at least one of the CZMZIE (Counter Zeroed by Zero Marker Interrupt Enable), CZMEIE (Zero Marker Error Interrupt), and/or CZMIE (CZM Pin Interrupt Enable/Push-Button Interrupt) bits in the CNT_IMASK register is set.

**WORKAROUND:**
If the counter is to be reset only on the first active level on the CZM pin, do all of the following:

1) Clear the ZMZC bit in the CNT_CONFIG register.
2) Set the W1ZMONCE bit in the CNT_COMMAND register.
3) Set the CZMZIE, CZMEIE, and/or CZMIE bits in the CNT_IMASK register.

As long as the SIC_IMASK register doesn't enable the counter interrupt, no action will be taken by the processor as a result of enabling one of these counter interrupts. If an alternate interrupt from the same counter is desired, software must ignore the extra interrupts resulting from the enable bit being set. To this regard, the CZMZIE interrupt is the most convenient to choose for this workaround.

Note that the Zero-marker-zeros-counter (ZMZC) mode is not affected by this anomaly.

**APPLIES TO REVISION(S):**
0.2

## 29. 05000501 - RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes:

**DESCRIPTION:**
When in SPI receive DMA modes, the RXS bit in SPI_STAT can get set and erroneously get stuck high if the SPI port is disabled as hardware is updating the status of the RXS bit. When in RX DMA mode, RXS will set as a word is transferred from the shift register to the internal FIFO, but it is then automatically cleared immediately by the hardware as DMA drains the FIFO. However, there is an internal 2 system clock (SCLK) latency for the status register to properly reflect this. If software disables the SPI port in exactly this window of time before RXS is cleared, the RXS bit doesn't get cleared and will remain set, even after the SPI is disabled. If the SPI port is subsequently re-enabled, the set RXS bit will cause one of two problems to occur:

1) If enabled in core RX mode, the SPI RX interrupt request will be raised immediately even though there is no new data in the SPI_RDBR register.
2) If enabled in RX DMA mode, DMA requests will be issued, which will cause the processor to DMA data from the SPI FIFO even though there is actually no new data present.

In master mode, the SPI will continue issuing clocks after RX DMA is completed until the SPI port is disabled. If any SPI word is received exactly as software disables the SPI port, the problem will occur.

In slave mode, the host would have to continue providing clocks and the chip-select for this possibility to occur.

**WORKAROUND:**
Reading the SPI_RDBR register while the SPI is disabled will clear the stuck RXS condition and not trigger any other activity. If using RX DMA mode, be sure to include this dummy read after the SPI port disable.

**APPLIES TO REVISION(S):**
0.2

## 30. 05000503 - SPORT Sign-Extension May Not Work:

**DESCRIPTION:**
In multichannel receive mode, the SPORT sign-extension feature (RDTPYE=b#01 in SPORTx_RCR1) is not reliable for channel 0 data when configured for MSB-first data reception. This is regardless of any channel offset and/or multichannel frame delay.

**WORKAROUND:**
1) If possible, use receive bit order of LSB-first.
2) Do not use channel 0.
3) Ignore channel 0 data.
4) Use software to manually apply sign extension to the channel 0 data before processing.

**APPLIES TO REVISION(S):**
0.2

## 31. 05000506 - Hardware Loop Can Underflow Under Specific Conditions:

**DESCRIPTION:**
When two consecutive hardware loops are separated by a single instruction, and the two hardware loops use the same loop registers, and the first loop contains a conditional jump to its loop bottom, the first hardware loop can underflow. For example:

```
P0 = 16;
LSETUP(loop_top1, loop_bottom1) LC0 = P0;
   loop_top1:    nop;
                 if CC JUMP loop_bottom1;
                 nop;
                 nop;
   loop_bottom1: nop;

nop;                         // Any single instruction

LSETUP(loop_top2, loop_bottom2) LC0 = P0;
   loop_top2:    nop;
   loop_bottom2: nop;
```

If a stall occurs on the instruction that is between the two loops, the top loop can decrement its loop count from 0 to 0xFFFFFFFF and continue looping with the incorrect loop count.

**WORKAROUND:**
There are several workarounds to this issue:

1) Do not use the same loop register set in consecutive hardware loops.
2) Ensure there is not exactly one instruction between consecutive hardware loops.
3) Ensure the first loop does not conditionally jump to its loop bottom.

**APPLIES TO REVISION(S):**
0.2

## 32. 05000510 - USB Wakeup from Hibernate State Requires Re-Enumeration:

**DESCRIPTION:**
Applications may choose to place the processor in hibernate state when the USB suspend is seen on the bus. Logic was added at the PHY level to maintain the state of the USB data lines so that the processor will appear to still be connected to the host during the USB suspend state. This would prevent re-enumeration when the USB bus was subsequently resumed. However, when in hibernate, the USB controller also loses its logic state because power is removed. As a result, re-enumeration is unavoidable when coming out of the hibernate state.

**WORKAROUND:**
None

**APPLIES TO REVISION(S):**
0.2

## 33. 05000511 - Lower 16 Bits of CNT_COUNTER Register Do Not Update Properly:

**DESCRIPTION:**
If the CNT_COUNTER register is read in the same SCLK cycle that the counter is being incremented by hardware, the MMR read may incorrectly return the previous value in the lower 16 bits. Under most circumstances, this is nearly negligible, as the value of the 32-bit read would be only one less than the correct value. However, in the case of a 16-bit overflow, the value read from the counter will be off by 0xffff from the expected value, as the upper 16 bits do get properly updated. For example, if the current counter value is 0x0000ffff, the next counter increment should result in CNT_COUNTER containing the value 0x00010000. When this anomaly manifests, the value read will incorrectly be 0x0001ffff.

**WORKAROUND:**
There is no workaround for this anomaly when it occurs any time other than when the counter wraps at 16 bits. For the case where the value in the lower 16 bits is 0xFFFF, a second read of the CNT_COUNTER register will ensure that the correct value is read.

**APPLIES TO REVISION(S):**
0.2

ANALOG DEVICES

www.analog.com