ABOUT ADSP-BF592 SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF592 product(s) and the functionality specified in the ADSP-BF592 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
0.1	0x0001

APPLICABILITY

Each anomaly applies to specific silicon revisions. See Summary or Detailed List for affected revisions.

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
01/15/2016	D	B	Revised Anomaly: 05000265
03/14/2014	C	B	Added Anomalies: 05000503, 05000506
07/08/2011	B	0	Added Anomalies: 05000490, 05000494, 05000497, 05000499, 05000501 Removed Anomaly: 05000477 Removed Silicon Revision 0.0 Added Silicon Revision 0.2
07/19/2010	A	PrA	Initial Revision

Blackfin and the Blackfin logo are registered trademarks of Analog Devices, Inc.

NR004052D   **Document Feedback**

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF592 anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	Rev 0.1	Rev 0.2
1	05000074	Multi-Issue Instruction with dsp32shiftimm in slot1 and P-reg Store in slot2 Not Supported	x	x
2	05000119	DMA_RUN Bit Is Not Valid after a Peripheral Receive Channel DMA Stops	x	x
3	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x
4	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
5	05000254	Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock	x	x
6	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x
7	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
8	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x
9	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
10	05000461	False Hardware Error when RETI Points to Invalid Memory	x	x
11	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x	x
12	05000490	SPI Master Boot Can Fail Under Certain Conditions	x	x
13	05000494	EXCPT Instruction May Be Lost If NMI Happens Simultaneously	x	x
14	05000497	PG8 Cannot be Used as a GPIO Output	x	.
15	05000499	Maximum Idd-deepsleep Spec Can Be Exceeded under Certain Conditions	x	.
16	05000501	RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes	x	x
17	05000503	SPORT Sign-Extension May Not Work	x	x
18	05000506	Hardware Loop Can Underflow Under Specific Conditions	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-BF592 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):
0.1, 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):
0.1, 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):
0.1, 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):
0.1, 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):
0.1, 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 I²S/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):

0.1, 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):
0.1, 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):
0.1, 0.2

9. 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):

0.1, 0.2

10. **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):

0.1, 0.2

11. 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):
0.1, 0.2

12. **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):

0.1, 0.2

13. 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):
0.1, 0.2

14. 05000497 - PG8 Cannot be Used as a GPIO Output:

DESCRIPTION:
The port signal PG8 always outputs low regardless of PORTG register settings when configured as a GPIO output (PORTG_FER[8]=b#0 and PORTGIO_DIR[8]=b#1). PG8 works correctly when configured as a GPIO input or when used as a peripheral signal.

WORKAROUND:
PG8 will work as a GPIO output if SPI1 is enabled (but not used) by writing b#1 to the SPE bit of the SPI1_CTL register.

APPLIES TO REVISION(S):
0.1

15. 05000499 - Maximum Idd-deepsleep Spec Can Be Exceeded under Certain Conditions:

DESCRIPTION:
When the processor is started up at colder temperatures, there is a chance that the Idd-deepsleep maximum specifications across the voltage domain may be violated. The amount of excess current is dependent on process, Vddint voltage, and temperature, and it is amplified as temperature is reduced and as Vddint is increased. It will, however, never exceed the total maximum Idd-deepsleep specification for the 70 C data point at the same Vddint level.

WORKAROUND:
The issue is completely resolved if sufficiently high levels are present at start-up for both the junction temperature (Tj >= 25C) and the internal voltage (Vddint >= 1.45V). This recommendation is safe across the supported Tj and Vddint domains for all models.

APPLIES TO REVISION(S):
0.1

16. 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):
0.1, 0.2

17. 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):
0.1, 0.2

18. 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):
0.1, 0.2

w w w . a n a l o g . c o m