ANALOG DEVICES

Blackfin Embedded Processor

Silicon Anomaly List — ADSP-BF504/BF504F/BF506F

ABOUT ADSP-BF504/BF504F/BF506F SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin® ADSP-BF504/BF504F/BF506F product(s) and the functionality specified in the ADSP-BF504/BF504F/BF506F 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>
0.1	0x0001

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-BF506F	ADSP-BF504F	ADSP-BF504	Anomalies
Internal ADC	x			None
Internal Flash	x	x		None

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	Added Anomalies: 05000506, 05000508, 05000511 Revised Anomaly: 05000265 Removed Silicon Revision 0.0
05/03/2012	C	A	Added Anomaly: 05000503
01/28/2012	B	A	Added Revision 0.1 Silicon
02/18/2011	A	0	Initial Revision

NR004111D Document Feedback

One Technology Way, P.O.Box 9106, Norwood, MA 02062-9106 U.S.A.
Tel: 781.329.4700
Technical Support **www.analog.com**

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF504/BF504F/BF506F anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	Rev 0.1
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	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x
10	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x
11	05000447	UART IrDA Receiver Fails on Extended Bit Pulses	x
12	05000461	False Hardware Error when RETI Points to Invalid Memory	x
13	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x
14	05000476	SPORT0 Data Transmit Error in Multi-Channel Mode with Internal Clock	x
15	05000477	TESTSET Instruction Cannot Be Interrupted	x
16	05000478	Disabling ACM During an Ongoing Transfer Can Lead to Undefined ACM Behavior	x
17	05000481	Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption	x
18	05000486	TWI Vbus Minimum Specification Can Be Violated under Certain Conditions	x
19	05000491	Instruction Memory Stalls Can Cause IFLUSH to Fail	x
20	05000494	EXCPT Instruction May Be Lost If NMI Happens Simultaneously	x
21	05000498	CNT_COMMAND Functionality Depends on CNT_IMASK Configuration	x
22	05000501	RXS Bit in SPI_STAT May Become Stuck In RX DMA Modes	x
23	05000503	SPORT Sign-Extension May Not Work	x
24	05000506	Hardware Loop Can Underflow Under Specific Conditions	x
25	05000508	UART Receive DMA Hangs under Certain Conditions	x
26	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-BF504/BF504F/BF506F 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

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

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

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

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

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):

0.1

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

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

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

10. 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):
0.1

11. 05000447 - UART IrDA Receiver Fails on Extended Bit Pulses:

DESCRIPTION:
The UART fails reception when the width of the receive pulse is wider than 3/16th. As defined by the standard, all IrDA transmitters assert bit pulses for exactly 3/16th of a bit period. Wired connections of an IrDA transmitter to the Blackfin IrDA receiver work properly. If the connection is not hard wired but is implemented instead via an infrared link, it has been observed that infrared tranceiver devices extend the output pulse beyond the 3/16th duration. This can cause the Blackfin UART IrDA receiver to fail at higher bit rates. The baud rate where the Blackfin UART IRDA function fails depends on the characteristics of the particular IRDA transceiver module that is used. When an infrared tranceiver that employs extended bit pulses is used, the Blackfin receiver still properly detects the start bit. However, the Blackfin may not properly latch in data. The receive interrupt count may also not match the number of transmitted bytes.

WORKAROUND:
There are several workarounds possible:

1) Add external logic ensuring IRDA RXD pulse width is always 3/16th parts of the UART bit rate.
2) Use external IRDA Encoder/Decoder (for example: HSDL-7000, MCP2122, TIR1000).
3) Use IRDA transceiver modules where tPW max = 3/16 parts of the UART Baud Rate (1.6us @ 115200 bps). ADI has not identified any devices meeting this criteria.

APPLIES TO REVISION(S):
0.1

12. 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

13. 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

14. 05000476 - SPORT0 Data Transmit Error in Multi-Channel Mode with Internal Clock:

DESCRIPTION:
When SPORT0 is configured in multi-channel mode (MCMEN=1 in SPORT0_MCMC2 register) with internal clock generation (IRCLK=1 in SPORT0_RCR1 register), the data transmitted may be incorrect.

WORKAROUND:
There are several workarounds:

1) Do not use multi-channel mode on SPORT0.
2) If using multi-channel mode on SPORT0, use an external clock.
3) If using multi-channel mode with an internal clock on SPORT0, use it for receive operation only.
3) If multi-channel mode transmit operation with an internal clock is required, use SPORT1.

APPLIES TO REVISION(S):
0.1

15. 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):
0.1

16. 05000478 - Disabling ACM During an Ongoing Transfer Can Lead to Undefined ACM Behavior:

DESCRIPTION:
If the ACM is disabled exactly one cycle before the assertion of the chip select $\overline{CS}$ signal (i.e., when an ACM event is transitioning from the setup stage to the active stage), then the ACM will not shutdown gracefully. When this occurs, subsequent enabling of the ACM will result in undefined behavior.

WORKAROUND:
There are two workarounds possible:

1) Disable the ACM after all enabled events have completed. If the ECOM bit in the ACM_STAT register is set, it is safe to disable the ACM.

2) If waiting for the completion of all events is not desirable, the ACM must be disabled twice prior to further use. In other words, after the ACM is disabled, it must be enabled and disabled again to clear the anomalous condition before using it again.

APPLIES TO REVISION(S):
0.1

17. 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):
0.1

18. 05000486 - TWI Vbus Minimum Specification Can Be Violated under Certain Conditions:

DESCRIPTION:
The TWI (I^2C) Vbustwi Minimum specification is violated when the TWI_DT field in the Drive Strength Control register (NONGPIO_DRIVE) is programmed to 000, which is the default configuration. For this case, the actual Vbustwi minimum specification is 3.07 V. For all other combinations of Vddext and Vbustwi, the Vbustwi minimum specifications in the data sheet are not violated.

WORKAROUND:
None. When the Blackfin processor is configured for a nominal Vddext of 3.3 V and a nominal Vbustwi of 3.3 V (i.e., in the NONGPIO_DRIVE register, TWI_DT = b#000), ensure that Vbustwi does not drop below 3.07 V.

APPLIES TO REVISION(S):
0.1

19. 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):
0.1

20. 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

21. 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):
0.1

22. 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

23. 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

24. 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

25. 05000508 - UART Receive DMA Hangs under Certain Conditions:

DESCRIPTION:
When the UART is in Receive DMA mode and the receive FIFO is empty, a permanent lock-up can occur if new UART RX data is received into the FIFO in the same SCLK cycle as the arrival of the internal DMA grant signal from the previous DMA read to get the last data from the UART_RBR register. When this precise timing is met, the UART does not clear the old data from the UART_RBR register, but it does properly clear the UART_LSR.DR status bit and the internal DMA grant signal. When this occurs, new data entering the UART RX FIFO will not be forwarded to the UART_RBR register, which prevents the UART from raising DMA data read requests and causes the UART RX DMA to hang.

WORKAROUND:
The cause for the internal DMA grant to the UART being delayed long enough for this condition to occur is the fact that the UART channel is the lowest priority in the DMA arbitration scheme and can be held off by other DMA channels enabled in the system. If the UART channel is given higher DMA priority, this condition can be avoided.

If prioritizing the UART DMA over other DMA activity is not an option, then the UART hang condition can be detected and corrected in software by checking the UART_MSR.RFCS and UART_LSR.DR status bits for an illegal combination that results from this anomaly. When the lock-up occurs, the UART_LSR.DR bit is properly cleared by the UART but the data from the FIFO isn't loaded to the UART_RBR register, thus forcing the UART RX FIFO to fill as new data comes in, eventually causing the UART_MSR.RFCS bit to set. Since the UART_MSR.RFCS bit indicates the status of the RX FIFO behind the UART_RBR register, it cannot be set if UART_LSR.DR = 0 (indicating that the UART_RBR register is empty). Once this invalid status is detected, a dummy core read of the UART_RBR register will clear the old data in the UART_RBR register, and the FIFO data will be forwarded properly. Hence, pending DMA read requests will be issued again, and the UART resumes operation:

```
short temp_data;

// Two reads with the following status is the hang condition
if((*pUART_MSR & RFCS) && (!(*pUART_LSR & DR)))   // RFCS = 1, DR = 0
{
  // Read status a 2nd time, perform corrective action if still true
  if((*pUART_MSR & RFCS) && (!(*pUART_LSR & DR)))
    temp_data = *pUART_RBR;  // dummy core read access to UART_RBR register
}
```

APPLIES TO REVISION(S):
0.1

26. 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):
0.1