ANALOG DEVICES

Blackfin® Embedded Processor

Silicon Anomaly List — ADSP-BF512/BF514/BF516/BF518(F)

ABOUT ADSP-BF512/BF514/BF516/BF518(F) SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin ADSP-BF512/BF514/BF516/BF518(F) product(s) and the functionality specified in the ADSP-BF512/BF514/BF516/BF518(F) 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>
0.0	0x0000
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-BF518(F)	ADSP-BF516(F)	ADSP-BF514(F)	ADSP-BF512(F)	Anomalies
RSI	x	x	x		None
Ethernet MAC	x	x			None
IEEE-1588	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
01/26/2010	E	0	Added Anomaly: 05000482 Deleted Anomaly: 05000469
12/16/2009	D	PrH	Added Anomalies: 05000119, 05000434, 05000469, 05000472, 05000473, 05000477, 05000481
06/12/2009	C	PrE	Added Silicon Revision 0.1 Added Anomalies: 05000461, 05000462 Revised Anomalies: 05000435, 05000444
02/03/2009	B	PrD	Added Anomalies: 05000254, 05000452, 05000453, 05000455 Revised Anomalies: 05000435, 05000439, 05000444
10/21/2008	A	PrC	Initial Revision

NR003827E

One Technology Way, P.O.Box 9106, Norwood, MA 02062-9106 U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.461.3113

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF512/BF514/BF516/BF518(F) anomalies and the applicable silicon revision(s) for each anomaly.

No.	ID	Description	0.0	0.1
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	05000405	Lockbox SESR Firmware Does Not Save/Restore Full Context	x	x
10	05000408	Lockbox Firmware Memory Cleanup Routine Does not Clear Registers	x	x
11	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x
12	05000421	TWI Fall Time (Tof) May Violate the Minimum I2C Specification	x	x
13	05000422	TWI Input Capacitance (Ci) May Violate the Maximum I2C Specification	x	x
14	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
15	05000430	Software System Reset Corrupts PLL_LOCKCNT Register	x	.
16	05000431	Incorrect Use of Stack in Lockbox Firmware During Authentication	x	x
17	05000434	SW Breakpoints Ignored Upon Return From Lockbox Authentication	x	x
18	05000435	Certain SIC Registers are not Reset After Soft or Core Double Fault Reset	x	.
19	05000438	PORTx_DRIVE and PORTx_HYSTERESIS Registers Read Back Incorrect Values	x	.
20	05000439	Preboot Cannot be Used to Alter the PLL_DIV Register	x	.
21	05000440	bfrom_SysControl() Cannot be Used to Write the PLL_DIV Register	x	.
22	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x
23	05000444	Incorrect L1 Instruction Bank B Memory Map Location	x	.
24	05000452	Incorrect Default Hysteresis Setting for $\overline{\text{RESET}}$, $\overline{\text{NMI}}$, and BMODE Signals	x	.
25	05000453	PWM_TRIPB Signal Not Available on PG10	x	.
26	05000455	PPI_FS3 is Driven One Half Cycle Later Than PPI Data	x	.
27	05000461	False Hardware Error when RETI Points to Invalid Memory	x	x
28	05000462	Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign	x	x
29	05000472	Incorrect Default MSEL Value in PLL_CTL	x	x
30	05000473	Interrupted SPORT Receive Data Register Read Results In Underflow when SLEN > 15	x	x
31	05000477	TESTSET Instruction Cannot Be Interrupted	x	x
32	05000481	Reads of ITEST_COMMAND and ITEST_DATA Registers Cause Cache Corruption	x	x
33	05000482	PLL Latches Incorrect Settings During Reset	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-BF512/BF514/BF516/BF518(F) 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, 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):
0.0, 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):
0.0, 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, 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):

0.0, 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, 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):

0.0, 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):
0.0, 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. Unexpected high frequency transitions on the RSCLK/TSCLK can cause the SPORT to recognize an extra noise-induced glitch clock pulse.

The high frequency transitions on the RSCLK/TSCLK are most likely to be caused by noise on the rising or falling edge of external serial clocks. This noise, coupled with a slowly transitioning serial clock signal, 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.

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

• In stereo serial modes, this will show up as missed frame syncs, causing left/right data swaps.
• In multichannel mode, this will show up as MFD counts appearing inaccurate or skipped frames.
• In Normal (Early) Frame sync mode, data words received will 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 FS-logic was already 'triggered', the next 'normal' RSCLK will not detect the change in RFS anymore. 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 multichannel mode, the mutlichannel frame delay (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.

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):
0.0, 0.1

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

DESCRIPTION:
Fetches at the boundary of either reserved memory or L1 Instruction cache memory (if instruction cache enabled) which is covered by a valid CPLB cause a false Hardware Error (External Memory Addressing Error).

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

Note that this anomaly also happens on the boundary of L1_code_cache if instruction cache is enabled.

APPLIES TO REVISION(S):
0.0, 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):
0.0, 0.1

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):
0.0, 0.1

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):
0.0, 0.1

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):

0.0, 0.1

12. 05000421 - TWI Fall Time (Tof) May Violate the Minimum I2C Specification:

DESCRIPTION:
TWI Fall Time (Tof) may be less than the minimum I2C specification. This is not a functional issue, but may have an impact on signal integrity.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0, 0.1

13. 05000422 - TWI Input Capacitance (Ci) May Violate the Maximum I2C Specification:

DESCRIPTION:
TWI input capacitance (Ci) may be may be more than the maximum I2C specification. This is not a functional issue, but may have an impact on signal integrity.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0, 0.1

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;
```

APPLIES TO REVISION(S):

0.0, 0.1

15. 05000430 - Software System Reset Corrupts PLL_LOCKCNT Register:

DESCRIPTION:
Software reset (`raise 1;`, core double fault, watchdog reset, *bfrom_SysControl()* call with *SWRST* flag) will return with PLL_LOCKCNT register value set to 0x0007 (instead of intended reset value of 0x0200). This may result in failure to subsequently re-program PLL and Voltage Regulator control registers.

Hardware reset will result in proper reset value loaded into PLL_LOCKCNT and no action is required.

WORKAROUND:
Following any software reset (`raise 1;`, core double fault, watchdog reset, *bfrom_SysControl()* call with *SWRST* flag) users cannot rely upon the default reset value within the PLL_LOCKCNT register and must explicitly write a value of 0x0200 or larger to PLL_LOCKCNT register before making subsequent changes to PLL or Voltage Regulator control registers.

Do not use the Preboot mechanism for setting PLL or Voltage Regulator control registers if power settings will be changed before entering software reset. It is recommended to use application code or init code to explicitly write to PLL_LOCKCNT prior to changing the PLL or Voltage Regulator control registers as stated above.

APPLIES TO REVISION(S):
0.0

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

APPLIES TO REVISION(S):
0.0, 0.1

17. 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):

0.0, 0.1

18. 05000435 - Certain SIC Registers are not Reset After Soft or Core Double Fault Reset:

DESCRIPTION:
SIC_IMASK1, SIC_IAR4, SIC_IAR5, SIC_IAR6, and SIC_IWR1 do not get reset to their default values after a soft or double fault reset. This may cause the boot kernel to hang, depending on the boot mode, if the following bits in SIC_IWR1 are been modified from their default value of b#1:

Interrupt Name	Bit Number	Boot Modes Affected
MDMA0	10	All
MDMA1	11	All

Watchdog and hardware resets are not affected.

WORKAROUND:
1. If possible, do not change SIC_IWR1 bits affecting the boot mode(s) in use from their default settings.

2. If it is necessary to change the bits in SIC_IWR1 that affect the boot mode(s) in use, the following workaround may be used before altering SIC_IWR1:

a. Set the BCODE field of the SYSCR register to BCODE_NOBOOT so that the processor will begin executing out of L1 Instruction Memory after a software or double fault reset:

```
*pSYSCR |= BCODE_NOBOOT;
```

b. Place the code below at the beginning of L1 instruction memory (0xFFA00000)

```
*pSIC_IWR1=0xFFFFFFFF;
asm ("raise 1;");
```

The processor will then boot normally.

APPLIES TO REVISION(S):
0.0

19. 05000438 - PORTx_DRIVE and PORTx_HYSTERESIS Registers Read Back Incorrect Values:

DESCRIPTION:
The drive strength and hysteresis control registers (PORTx_DRIVE and PORTx_HYSTERESIS) do not reflect their actual values when read back.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

20. 05000439 - Preboot Cannot be Used to Alter the PLL_DIV Register:

DESCRIPTION:
The preboot routine must not be used to alter the value of the PLL_DIV register. Doing so may result in the processor ceasing to execute instructions.

WORKAROUND:
There are two possible workarounds for this issue.

1. The initcode feature described in the Hardware Reference Manual's *System Reset and Booting* chapter can be used to change PLL_DIV during boot time. The initcode must be located in L1 memory and must access PLL_DIV directly rather than using the *bfrom_SysControl()* routine.

2. The default SCLK and CCLK frequencies can still be changed with the preboot routine by modifying the default MSEL value in the PLL_CTL register. Program OTP_SET_PLL to '1' and OTP_PLL_CTL to the desired value in the PBS00L page. However, the field OTP_PLL_DIV must be programmed with the default PLL_DIV value of 0x0004. See the *System Reset and Booting* chapter in the Hardware Reference Manual for details on how to customize power management using the preboot routine.

APPLIES TO REVISION(S):
0.0

21. 05000440 - bfrom_SysControl() Cannot be Used to Write the PLL_DIV Register:

DESCRIPTION:
The *bfrom_SysControl()* firmware function cannot be used to program the PLL_DIV register. Doing so may result in the processor ceasing to execute instructions.

WORKAROUND:
PLL_DIV must be changed by modifying the register directly rather than using *bfrom_SysControl()*.

APPLIES TO REVISION(S):
0.0

22. 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
```

APPLIES TO REVISION(S):
0.0, 0.1

23. 05000444 - Incorrect L1 Instruction Bank B Memory Map Location:

DESCRIPTION:
L1 Instruction Bank B is incorrectly located in the processor's memory map, as follows:

Starting Address	Ending Address	Description
0xFFA10000	0xFFA13FFF	Instruction Bank C SRAM/Cache
0xFFA0C000	0xFFA0FFFF	RESERVED
0xFFA08000	0xFFA0BFFF	Instruction Bank B SRAM
0xFFA04000	0xFFA07FFF	RESERVED
0xFFA00000	0xFFA03FFF	Instruction Bank A SRAM

The correct memory map is shown here:

Starting Address	Ending Address	Description
0xFFA10000	0xFFA13FFF	Instruction Bank C SRAM/Cache
0xFFA0C000	0xFFA0FFFF	RESERVED
0xFFA08000	0xFFA0BFFF	RESERVED
0xFFA04000	0xFFA07FFF	Instruction Bank B SRAM
0xFFA00000	0xFFA03FFF	Instruction Bank A SRAM

Consequently:
1. DMA errors are not generated when accessing the reserved range 0xFFA04000-0xFFA07FFF.
2. Lockbox does not protect L1 Instruction Bank B.
3. In order to access L1 Instruction Bank B using the DTEST_COMMAND register, bit 14 must be b#0 and bit 23 must be b#1.

WORKAROUND:
None. Once silicon with the correct memory map is provided, rebuild applications with an updated linker description file (LDF).

APPLIES TO REVISION(S):
0.0

24. 05000452 - Incorrect Default Hysteresis Setting for $\overline{\text{RESET}}$, $\overline{\text{NMI}}$, and BMODE Signals:

DESCRIPTION:
The setting b#00 for the NMI_RST_BMODE_SE field in the NONGPIO_HYSTERESIS register should enable hysteresis such that it is on by default for the $\overline{\text{RESET}}$, $\overline{\text{NMI}}$, and BMODE signals. Instead, the setting b#00 incorrectly disables hysteresis for those signals.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

25. 05000453 - PWM_TRIPB Signal Not Available on PG10:

DESCRIPTION:
The signal PWM_TRIPB is not available in its alternate location, PG10.

WORKAROUND:
Use PWM_TRIPB in its default location, PG14.

APPLIES TO REVISION(S):
0.0

26. 05000455 - PPI_FS3 is Driven One Half Cycle Later Than PPI Data:

DESCRIPTION:
The PPI_FS3 frame sync signal is driven half a PPI_CLK cycle later than PPI_DATA in PPI Tx Mode with internal frame sync generation.

WORKAROUND:
None

APPLIES TO REVISION(S):
0.0

27. 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):
0.0, 0.1

28. 05000462 - Synchronization Problem at Startup May Cause SPORT Transmit Channels to Misalign:

DESCRIPTION:
When the SPORT is configured in multichannel mode with an external SPORT clock, a synchronization problem may occur when the SPORT is enabled. This synchronization issue manifests when the skew between the external SPORT clock and the Blackfin processor's internal System Clock (SCLK) causes the channel counters inside the SPORT to get out-of-sync. When this occurs, a "dead" channel is inserted at the beginning of the window, and the rest of the transmit channels are right-shifted one location throughout the active window. The last channel data will be sent as the first enabled transmit channel data in the second window after another "dead" channel is inserted. All data will be sent sequentially and in its entirety, but it is transmitted on the wrong channels with respect to the frame sync and will never recover.

WORKAROUND:
When this error occurs, the SPORT must be restarted and checked again for this error. The failure is extremely rare to begin with, so the probability of seeing consecutive restarts showing the failure is infinitesimally small.

A software solution is possible based on the timing of the SPORT interrupt. In the SPORT ISR, the CYCLES register can be set to zero the first time the interrupt occurs and then read back the second time the interrupt occurs. This will provide a time reference in core clocks for the frequency of the SPORT interrupt itself. If the value read the second time exceeds the duration of the multichannel window (in core clocks), then a "dead" channel was inserted into the stream, and the SPORT must be restarted.

Hardware workarounds are going to be heavily dependent on how the multichannel mode SPORT is configured. In multichannel mode, TFS functions as a Transmit Data Valid (TDV) signal and will always be driven to the active state (as governed by the LTFS bit in the SPORTx_TCR1 register) during transmit channels. Therefore, the TDV signal can be routed to one of the GPIO pins configured to generate an interrupt upon detection of the TDV pin changing states, based upon how the application configures the channels within the active frame, to detect the "dead" channel. If all the channels in the window are configured as transmit channels and there is no window offset and no multichannel frame delay, then TDV should go active as soon as the RFS pulse is received. If the period of the RFS pulse is exactly the window size (i.e., there are no extra clocks after the active window before the next RFS is detected), then TDV will remain active throughout operation. Therefore, if TDV goes inactive while the SPORT is on, the failure happened and the SPORT must be restarted and run again with this test in place until the failure is not detected.

For applications that have a window offset, a multichannel frame delay, extra clocks between the end of the active window and the next frame sync, and/or non-transmit channels inside the active window, the first TDV assertion would need to be tracked manually to detect the "dead" channel. One idea might be to do the following:

1. Connect TFS (TDV) to a GPIO interrupt and configure the interrupt to occur when TDV goes active.
2. Connect RFS to a GPIO interrupt and configure the interrupt to occur when RFS goes active.
3. Connect the SPORT receive clock to a TMRx pin configured in EXT_CLK mode.

When the GPIO interrupt for the active RFS pulse signifying the start of the window occurs, enable the Timer that is being used to track the SPORT receive clock. When the GPIO interrupt for the TDV signal transition occurs, check the TIMERx_COUNTER register to determine how many SPORT clocks have passed since the frame started. If it is one channel's worth over the expected value, the error occurred and the SPORT must be restarted and tested again. The GPIO interrupts should also be disabled if the startup condition is not detected.

APPLIES TO REVISION(S):
0.0, 0.1

29. 05000472 - Incorrect Default MSEL Value in PLL_CTL:

DESCRIPTION:
The default value of MSEL in the PLL_CTL register is incorrectly set to 0x5. This leads to a violation of the minimum VCO frequency (specified in the data sheet) at boot time.

WORKAROUND:
Use a CLKIN frequency of at least 14MHz (Tckin <= 71.4 ns).

APPLIES TO REVISION(S):
0.0, 0.1

30. 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):
0.0, 0.1

31. 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.

APPLIES TO REVISION(S):
0.0, 0.1

32. 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):
0.0, 0.1

33. 05000482 - PLL Latches Incorrect Settings During Reset:

DESCRIPTION:
The PLL can latch incorrect SSEL and CSEL values during reset when VDDINT is powered before VDDMEM. In this case, the PLL_DIV register will show the correct default value when read, but SSEL and CSEL may actually be set to different (possibly random) values. This results in a different CCLK and SCLK frequency after reset than what the default values would imply.

WORKAROUND:
There are two possible workarounds:
1) Issue an extra hardware reset of any duration.
2) Ensure that VDDMEM ramps up fully before turning on VDDINT.

APPLIES TO REVISION(S):
0.0, 0.1