

Silicon Anomaly List
ADSP-BF522/BF524/BF526(C)
ABOUT ADSP-BF522/BF524/BF526(C) SILICON ANOMALIES

These anomalies represent the currently known differences between revisions of the Blackfin ADSP-BF522/BF524/BF526(C) product(s) and the functionality specified in the ADSP-BF522/BF524/BF526(C) data sheet(s) and the Hardware Reference book(s).

SILICON REVISIONS

A silicon revision number with the form "-x.x" is branded on all parts. The implementation field bits <15:0> of the DSPID core MMR register can be used to differentiate the revisions as shown below.

Silicon REVISION	DSPID<15:0>
0.0	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-BF526(C)	ADSP-BF524(C)	ADSP-BF522(C)	Anomalies
USB	x	x		05000346 , 05000415 , 05000420 , 05000456
Ethernet MAC	x			05000423

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
03/13/2009	C	PrG	Added Silicon Revision 0.1 Added Anomalies: 05000254 , 05000346 , 05000422 , 05000431 , 05000435 , 05000439 , 05000440 , 05000443 , 05000452 , 05000456 , 05000457 Revised Anomalies: 05000265
08/12/2008	B	PrD	Added Anomalies: 05000353 , 05000414 , 05000415 , 05000416 , 05000418 , 05000420 , 05000421 , 05000423 , 05000425 , 05000426 , 05000429 , 05000432
05/20/2008	A	PrD	Initial Revision

NR003768C

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use, nor for any infringements of patents or other rights of third parties that may result from its use. Specifications subject to change without notice. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices. Trademarks and registered trademarks are the property of their respective owners.

SUMMARY OF SILICON ANOMALIES

The following table provides a summary of ADSP-BF522/BF524/BF526(C) 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	05000122	Rx.H Cannot Be Used to Access 16-bit System MMR Registers	x	x
3	05000245	False Hardware Error from an Access in the Shadow of a Conditional Branch	x	x
4	05000254	Incorrect Timer Pulse Width in Single-Shot PWM_OUT Mode with External Clock	x	x
5	05000265	Sensitivity To Noise with Slow Input Edge Rates on External SPORT TX and RX Clocks	x	x
6	05000310	False Hardware Errors Caused by Fetches at the Boundary of Reserved Memory	x	x
7	05000313	PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes	x	.
8	05000346	USB Calibration Value Is Not Initialized	x	.
9	05000353	bfrom_SysControl() Firmware Function Performs Improper System Reset	x	.
10	05000366	PPI Underflow Error Goes Undetected in ITU-R 656 Mode	x	x
11	05000382	8-Bit NAND Flash Boot Mode Not Functional	x	.
12	05000388	CRC32 Checksum Support Not Functional	x	.
13	05000401	PPI Data Signals D0 and D8 do not Tristate After Disabling PPI	x	.
14	05000403	Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall	x	.
15	05000404	Lockbox SESR Disallows Certain User Interrupts	x	.
16	05000405	Lockbox SESR Firmware Does Not Save/Restore Full Context	x	x
17	05000407	Lockbox SESR Firmware Arguments Are Not Retained After First Initialization	x	.
18	05000408	Lockbox Firmware Memory Cleanup Routine Does not Clear Registers	x	x
19	05000409	Lockbox firmware leaves MDMA0 channel enabled	x	.
20	05000411	bfrom_SysControl() Firmware Function Cannot be Used to Enter Power Saving Modes	x	.
21	05000414	OTP_CHECK_FOR_PREV_WRITE Bit is Not Functional in bfrom_OtpWrite() API	x	.
22	05000415	DEB2_URGENT Bit Not Functional	x	.
23	05000416	Speculative Fetches Can Cause Undesired External FIFO Operations	x	x
24	05000418	PPI Timing Requirements tSFSPE and tHFSPE Do Not Meet Data Sheet Specifications	x	.
25	05000420	USB PLL_STABLE Bit May Not Accurately Reflect the USB PLL's Status	x	.
26	05000421	TWI Fall Time (Tof) May Violate the Minimum I2C Specification	x	x
27	05000422	TWI Input Capacitance (Ci) May Violate the Maximum I2C Specification	.	x
28	05000423	Certain Ethernet Frames With Errors are Misclassified in RMII Mode	x	.
29	05000425	Multichannel SPORT Channel Misalignment Under Specific Configuration	x	.
30	05000426	Speculative Fetches of Indirect-Pointer Instructions Can Cause False Hardware Errors	x	x
31	05000429	WB_EDGE Bit in NFC_IRQSTAT Incorrectly Reflects Buffer Status Instead of IRQ Status	x	.
32	05000431	Incorrect Use of Stack in Lockbox Firmware During Authentication	x	x
33	05000432	bfrom_SysControl() Does Not Clear SIC_IWR1 Before Executing PLL Programming Sequence	x	.
34	05000435	Certain SIC Registers are not Reset After Soft or Core Double Fault Reset	x	.
35	05000439	Preboot Cannot be Used to Alter the PLL_DIV Register	x	.
36	05000440	bfrom_SysControl() Cannot be Used to Write the PLL_DIV Register	x	.
37	05000443	IFLUSH Instruction at End of Hardware Loop Causes Infinite Stall	x	x
38	05000452	Incorrect Default Hysteresis Setting for $\overline{\text{RESET}}$, $\overline{\text{NMI}}$, and BMODE Signals	x	.
39	05000456	USB Receive Interrupt Is Not Generated in DMA Mode 1	x	x
40	05000457	Host DMA Port Responds to Certain Bus Activity Without $\overline{\text{HOST_CE}}$ Assertion	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-BF522/BF524/BF526(C) including a description, workaround, and identification of applicable silicon revisions.

1. 05000074 - Multi-Issue Instruction with dsp32shiftime in slot1 and P-reg Store in slot2 Not Supported:

DESCRIPTION:

A multi-issue instruction with dsp32shiftime 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 dsp32shiftime in slot 1:

```
R0 = R0 << 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;
```

WORKAROUND:

In assembly programs, separate the multi-issue instruction into 2 separate instructions. The VisualDSP++ runtime libraries do not use the unsupported instructions. Additionally, the VisualDSP++ Blackfin compiler does not generate the unsupported instructions when targeting the parts and silicon revisions affected by this anomaly

APPLIES TO REVISION(S):

0.0, 0.1

2. 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 (where P0 points to a 16-bit system MMR), this access would fail:

```
w[P0] = R5.H;
```

WORKAROUND:

Use other forms of 16-bit transfers when accessing 16-bit system MMR registers. For example (where p0 points to a 16-bit system MMR):

```
w[p0] = r5.l;
r4.l = w[p0];
r3 = w[p0](z);
w[p0] = r3;
```

The VisualDSP++ Blackfin compiler will not normally emit a problem instruction when generating code. It will insert a pack instruction to swap register halves in the cases where the MMR load occurs with a constant address, e.g. *MMR_Reg = value; It cannot, however, identify pointers unknown at compile time (such as parameters to functions) as pointers to MMRs. The VisualDSP++ runtime libraries also avoid this anomaly.

APPLIES TO REVISION(S):

0.0, 0.1

3. 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.

The VisualDSP++ Blackfin compiler includes a workaround for this hardware anomaly. The compiler will automatically enable the workaround for the appropriate silicon revisions and part numbers, or you can enable the workaround manually by specifying the compiler flag '-workaround speculative-loads'.

With the workaround enabled, the compiler will insert NOPs to avoid the anomaly condition.

The macro `__WORKAROUND_SPECULATIVE_LOADS` will be defined at compile, assemble and link build phases when the workaround is enabled.

There are various checks in the compiler which avoid over-applying this workaround. For example, before the workaround is applied, it ensures that the load is:

- not through SP or FP (in which case from the stack and not illegal)
- not to a volatile qualified type address (in which case from a known legal address)
- from an address unknown to the compiler
- not duplicated in the branch targets (in which case it must be okay to execute speculatively)
- not executed previous to the jump (in which case it must be okay to execute speculatively)

The VisualDSP++ run-time libraries also avoid this anomaly for appropriate silicon revisions and part numbers.

APPLIES TO REVISION(S):

0.0, 0.1

4. 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

5. 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 multichannel 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

6. 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:

- 1) Leave at least 76 bytes free before any boundary with a reserved memory space. This will prevent false hardware errors from occurring.
- 2) Have the hardware error handler confirm whether the hardware error was valid or not before taking action. This can be done by verifying if the CODE_FAULT_ADDR register contains an address that is within a valid bank. In that case, no action is performed.

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

7. 05000313 - PPI Is Level-Sensitive on First Transfer In Single Frame Sync Modes:

DESCRIPTION:

When the PPI is configured to trigger on a single external frame sync, all of the transfers require an edge on the frame sync except for the first transfer. For the first transfer only, the frame sync input is level-sensitive. This will make the PPI begin a transfer if the frame sync is at the active state, which can cause the PPI to start prematurely.

This anomaly does not apply when the PPI uses 2 or 3 frame syncs.

WORKAROUND:

When using a single external frame sync with the PPI, ensure that the frame sync is in the inactive state when the PPI is enabled.

APPLIES TO REVISION(S):

0.0

8. 05000346 - USB Calibration Value Is Not Initialized:

DESCRIPTION:

After execution of the preboot routine, the USB_APHY_CALIB register may not be properly initialized. The USB peripheral may not function properly without proper initialization.

WORKAROUND:

It is recommended that users program the USB_APHY_CALIB register with a nominal value of 0x6510 in their application code if they wish to utilize the USB peripheral.

```
#define SYSMMR_BASE          0xFFC00000

...

P1.H = HI(SYSMMR_BASE); // P1 Points to the beginning of SYSTEM MMR Space
P1.L = LO(SYSMMR_BASE);

...

r0 = 0x6510(z); // recommended value to be set
w[p1 + lo(USB_APHY_CALIB)] = r0;
ssync;
```

APPLIES TO REVISION(S):

0.0

9. 05000353 - bfrom_SysControl() Firmware Function Performs Improper System Reset:

DESCRIPTION:

The *bfrom_SysControl()* firmware function does not properly execute a software reset and cannot be used to perform system reset of the processor. Some side effects that may be observed due to this are that not all System MMRs are reset correctly.

WORKAROUND:

Use the following code to perform system reset instead of *bfrom_SysControl(SYSCTRL_SYSRESET)*. This code must execute out of non-cache L1 memory:

```

/* Issue system soft reset */
P0.L = LO(SWRST) ;
P0.H = HI(SWRST) ;
R0.L = 0x0007 ;
W[P0] = R0 ;
SSYNC ;

/* Wait for System reset to complete (needs to be 5 SCLKs). */
/* Assuming a worst case CCLK:SCLK ratio (15:1), use 5*15 = 75 */
/* as the loop count. */
P1 = 75;
LSETUP(start, end) LCO = P1 ;
    start:
    end:
        NOP ;

/* Clear system soft reset */
R0.L = 0x0000 ;
W[P0] = R0 ;
SSYNC ;

/* Core reset - forces reboot */
RAISE 1 ;

```

APPLIES TO REVISION(S):

0.0

10. 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

11. 05000382 - 8-Bit NAND Flash Boot Mode Not Functional:

DESCRIPTION:

The 8-bit NAND flash boot mode (BMODE[3:0] = b#1100 and b#1101) is not functional.

WORKAROUND:

Use one of the other boot modes.

APPLIES TO REVISION(S):

0.0

12. 05000388 - CRC32 Checksum Support Not Functional:

DESCRIPTION:

The CRC32 wrapper routine (*bfrom_Crc32Callback()*) is not functional.

WORKAROUND:

None

APPLIES TO REVISION(S):

0.0

13. 05000401 - PPI Data Signals D0 and D8 do not Tristate After Disabling PPI:

DESCRIPTION:

The PPI data signals D0 and D8 do not get tristated when PPI is disabled after a transmit operation. The rest of the PPI data signals are unaffected.

WORKAROUND:

Disable PPI as normal. Then reconfigure the PPI settings for receive mode. Re-enable the PPI port and then disable it again. D0 and D8 will be tristated.

APPLIES TO REVISION(S):

0.0

14. 05000403 - Level-Sensitive External GPIO Wakeups May Cause Indefinite Stall:

DESCRIPTION:

When level-sensitive GPIO events are used to wake the processor from the low-power sleep mode of operation, the processor may stall indefinitely if the width of the wakeup pulse is too short. When this occurs, the PLL begins transitioning from the sleep mode due to the level sensed on the GPIO pin, but then reverts back to the sleep mode if the trigger level is removed before the core has had sufficient time to break the idle state to resume execution.

As a result, the processor does not wake up properly, at which point only a hardware reset can exit the resulting stall condition.

WORKAROUND:

There are two ways to avoid this anomaly:

- 1) Use edge-sensitivity for the pin(s) being used to generate the wakeup event.
- 2) Ensure that the edge on the wakeup signal is clean and held at the trigger level for at least 3 system clock (SCLK) cycles.

APPLIES TO REVISION(S):

0.0

15. 05000404 - Lockbox SESR Disallows Certain User Interrupts:

DESCRIPTION:

When the processor enters Secure Entry Mode, interrupts are shut off (i.e. IMASK is cleared except for bits [4:0] because they are hardwired). The user can select interrupts to be unmasked via an argument to the SESR. The IVTMR (core timer) and IVHW (HW error) interrupts cannot be unmasked inside the SESR. Therefore, IVTMR and IVHW cannot be serviced during the digital signature authentication process (Secure Entry Mode).

WORKAROUND:

None

APPLIES TO REVISION(S):

0.0

16. 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

17. 05000407 - Lockbox SESR Firmware Arguments Are Not Retained After First Initialization:

DESCRIPTION:

Lockbox SESR firmware arguments must be reset in subsequent digital signature authentication attempts as the arguments are not retained after first initialization.

WORKAROUND:

Lockbox SESR arguments must be programmed upon every authentication attempt.

If digital signature authentication is to be attempted following a failure or abortion of the authentication process, users must reset their SESR arguments for subsequent authentication attempts.

APPLIES TO REVISION(S):

0.0

18. 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

19. 05000409 - Lockbox firmware leaves MDMA0 channel enabled:

DESCRIPTION:

During the digital signature authentication process, the security firmware uses the MDMA0 channel, but does not disable the channel after use. If the customer application made use of that same MDMA0 channel, the customer may encounter problems configuring it.

WORKAROUND:

Customer applications should first disable the MDMA0 channel used by the security firmware before attempting to reconfigure it. This is only necessary following a successful authentication process.

APPLIES TO REVISION(S):

0.0

20. 05000411 - *bfrom_SysControl()* Firmware Function Cannot be Used to Enter Power Saving Modes:

DESCRIPTION:

The *bfrom_SysControl()* firmware function does not manage SIC_IWRx wakeup bits properly and cannot be used to enable the processor to enter sleep or deep sleep mode.

WORKAROUND:

Write to the PLL_CTL register directly to enter sleep or deep sleep mode.

APPLIES TO REVISION(S):

0.0

21. 05000414 - OTP_CHECK_FOR_PREV_WRITE Bit is Not Functional in bfrom_OtpWrite() API:

DESCRIPTION:

`OTP_CHECK_FOR_PREV_WRITE` is a flag which is provided to enable the check for unprogrammed OTP bits in the firmware OTP write operation.

If the `OTP_CHECK_FOR_PREV_WRITE` bit is set in the `dFlags` argument in the `bfrom_OtpWrite()` API, the OTP data bits are first checked for zero (unprogrammed) values before the write is allowed to proceed. This error detection for a write to a previously programmed page causes dedicated error messages and the write does not occur. If the `OTP_CHECK_FOR_PREV_WRITE` bit is cleared (default), the write proceeds without first performing this check.

The `OTP_CHECK_FOR_PREV_WRITE` bit is not functional in firmware and therefore this feature cannot be configured by the user. The firmware OTP write operation always performs the check for unprogrammed OTP bits to the data page target of the write operation. If the page is detected as being previously programmed (i.e., page contains a bit whose value is '1'), the write will not occur.

WORKAROUND:

None.

APPLIES TO REVISION(S):

0.0

22. 05000415 - DEB2_URGENT Bit Not Functional:

DESCRIPTION:

The `DEB2_URGENT` bit[12] in the `USB_PLLOSC_CTRL` register is not functional. Therefore, USB accesses to the EBIU are always lower priority than core EBIU accesses. For a complete description of the `DEB2_URGENT` bit please refer to the BF52x Hardware Reference manual.

WORKAROUND:

None

APPLIES TO REVISION(S):

0.0

23. 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

24. 05000418 - PPI Timing Requirements tSFSPe and tHFSPe Do Not Meet Data Sheet Specifications:

DESCRIPTION:

The data sheet PPI timing specifications tSFSPe and tHFSPe are not being met. Instead, use the following data:

Specification	Min	Unit
tSFSPe	2.1	ns
tHFSPe	7.0	ns

WORKAROUND:

None

APPLIES TO REVISION(S):

0.0

25. 05000420 - USB PLL_STABLE Bit May Not Accurately Reflect the USB PLL's Status:

DESCRIPTION:

The PLL_STABLE bit in the USB_PLLOSC_CTRL register does not automatically update when there is a change in the PLL's status.

WORKAROUND:

Read the value of USB_PLLOSC_CTRL, and write it back to the register. Immediately read USB_PLLOSC_CTRL again and the correct status of PLL_STABLE will be reflected.

APPLIES TO REVISION(S):

0.0

26. 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

27. 05000422 - TWI Input Capacitance (Ci) May Violate the Maximum I2C Specification:

DESCRIPTION:

TWI input capacitance (Ci) 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):

0.1

28. 05000423 - Certain Ethernet Frames With Errors are Misclassified in RMII Mode:

DESCRIPTION:

In RMII-mode at 100Base-T, some RX frames with FCS errors are incorrectly classified as having alignment errors in the EMAC_RX_STAT register.

WORKAROUND:

Ensure that all error frames are properly discarded.

APPLIES TO REVISION(S):

0.0

29. 05000425 - Multichannel SPORT Channel Misalignment Under Specific Configuration:

DESCRIPTION:

When using the Serial Port in Multi-Channel Mode, the transmit and receive channels can get misaligned if a very specific configuration for the SPORT is met, as follows:

- 1) Window Offset (WOF) = 0.
- 2) Frame Delay (MFD) > 0.
- 3) Window Size is an odd multiple of 8 (i.e., WSIZE is an even number).
- 4) The time between RFS pulses is exactly equal to the window duration.

When this exact configuration is used, the multi-channel mode channel enable registers are mismatched after the first window concludes, which results in the TDV signal being driven according to incorrect channel assignments and receive data being sampled on the wrong channels. So, the first window will send and receive properly, but all windows after the first will be misaligned and data sent and received will be corrupted.

This error occurs for external and internal clocks and RFS.

WORKAROUND:

There are several workarounds possible:

- 1) Use a window offset other than 0.
- 2) Use a frame delay of 0.
- 3) Use a window size that is an even multiple of 8.
- 4) For internal RFS, make sure that SPORTx_RFSDIV is at least equal to the window size (# of enabled channels * SLEN).

APPLIES TO REVISION(S):

0.0

30. 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:
RETS;
```

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: RETS; // 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: RETS;
```

APPLIES TO REVISION(S):

0.0, 0.1

31. 05000429 - WB_EDGE Bit in NFC_IRQSTAT Incorrectly Reflects Buffer Status Instead of IRQ Status:

DESCRIPTION:

The NAND Flash Controller (NFC) "write buffer edge detect" (WB_EDGE) interrupt is generated when the 4-word-deep write buffer of the NFC becomes empty. The functional generation of the interrupt is correct. The functional description of the WB_EDGE bit is described as "The write buffer edge detect (WB_EDGE) is set when the write buffer transitions from 'not empty' to 'empty.'" This correctly describes the behavior through which the interrupt is generated. However, the status bit for this interrupt does not align with the functional behavior of the interrupt.

The WB_EDGE bit of NFC_IRQSTAT register currently behaves as a "write buffer empty" status bit instead of as an actual IRQ status bit. The WB_EDGE bit is set when the NFC write buffer is empty and cleared when the buffer is not empty. The IRQ is cleared correctly after the W1C operation, but the bit will remain set. This does not reflect the IRQ status, and the behavior of this bit is identical to the WB_EMPTY bit in the NFC_STAT register.

The NFC_IRQSTAT default reset value of 0x0004 indicates that an IRQ is already active. The intended default reset value is 0x0000.

WORKAROUND:

Do not depend upon the WB_EDGE bit of NFC_IRQSTAT register to determine status of the write buffer.

The status of the write buffer should be determined by checking the WB_EMPTY bit or WB_FULL bit in the NFC_STATUS register.

If software requires an interrupt handler that services WB_EDGE interrupts, the following procedure is advised:

- 1) Only unmask the WB_EDGE interrupt prior to performing NFC operations that are issued through the write buffer. Otherwise mask off the interrupt.
- 2) From within the interrupt handler, service all interrupts except for the WB_EDGE interrupt and clear them by performing the appropriate W1C operation to the NFC_IRQSTAT register.
- 3) Service the WB_EDGE interrupt only when the SIC_ISRx register indicates an NFC interrupt is still latched and the only bit set in the NFC_IRQSTAT register is the WB_EDGE bit.
- 4) Mask off the WB_EDGE interrupt within the WB_EDGE interrupt handler if the handler does not issue any NFC write buffer transactions. If the WB_EDGE IRQ handler does issue NFC write buffer transactions, ensure the current WB_EDGE interrupt is cleared before performing the transactions.

APPLIES TO REVISION(S):

0.0

32. 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; \
        [--SP] = R0; \
        [--SP] = P1; \
        // <-- ADD THIS INSTRUCTION
```

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; \
    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

33. 05000432 - bfrom_SysControl() Does Not Clear SIC_IWR1 Before Executing PLL Programming Sequence:**DESCRIPTION:**

The *bfrom_SysControl()* firmware function does not clear the SIC_IWR1 register before executing the PLL programming sequence. Therefore, any interrupt enabled in the SIC_IWR1 register prior to the call to *bfrom_SysControl()* can prematurely terminate the idle sequence required for the PLL to relock properly. SIC_IWR0 is properly handled.

WORKAROUND:

Before using *bfrom_SysControl()* to change PLL settings, save the value of SIC_IWR1 to a temporary location and set SIC_IWR1 to 0. Then make the call to *bfrom_SysControl()* and subsequently restore the saved value of SIC_IWR1 after *bfrom_SysControl()* executes. The *bfrom.h* header file in VisualDSP++ 5.0 Update 3 or later must be included in order to use *bfrom_SysControl()*.

The following example sets a CCLK of 300 MHz and an SCLK of 75 MHz, assuming that CLKIN is 25 MHz and PLL_DIV is at its default value of 0x0004:

```

u32 SIC_IWR1_reg;           // 32-bit data element to store SIC_IWR1 register

ADI_SYSCTRL_VALUES frequency; // Create SYSCTRL_VALUES structure called frequency

// Set MSEL = 0-63 --> VCO = CLKIN*MSEL
frequency.uwPllCtl = SET_MSEL(12);

//Set PllLockCnt to default value
frequency.uwPllLockCnt = 0x0200;

SIC_IWR1_reg = *pSIC_IWR1; // Save value of SIC_IWR1
*pSIC_IWR1=0;              // Disable wakeups from SIC_IWR1

// Call bfrom_SysControl to set PLL
bfrom_SysControl(SYSCTRL_WRITE | SYSCTRL_EXTVOLTAGE | SYSCTRL_PLLCTL |
                 SYSCTRL_LOCKCNT , &frequency, NULL);

*pSIC_IWR1 = SIC_IWR1_reg; // Restore original SIC_IWR1 register value

```

APPLIES TO REVISION(S):

0.0

34. 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
NFC status	16	NAND Flash
HOSTDP status	17	Host DMA
Host read done	18	Host DMA

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_IWR=0xFFFFFFFF;
asm ("raise 1;");
```

The processor will then boot normally.

APPLIES TO REVISION(S):

0.0

35. 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

36. 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

37. 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 (1f, 2f) LC1 = P1;
1: NOP;
2: IFLUSH[P0++];

LSETUP (1f, 2f) LC1 = P1;
1: NOP; NOP; NOP; NOP;          // Any number of instructions...
2: 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 (1f, 2f) LC1 = P1;
1: IFLUSH[P0++];
2: NOP;                          // Pad the loop end
```

APPLIES TO REVISION(S):

0.0, 0.1

38. 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

39. 05000456 - USB Receive Interrupt Is Not Generated in DMA Mode 1:

DESCRIPTION:

Whether the USB is used in host or device mode, the USB receive interrupt may not be generated when DMA Mode 1 is used.

For DMA Mode 1 host mode receive operations where the transfer size is an integer multiple of MaxPacketSize, extra "in" tokens are sent out by the USB controller at the end of the DMA transfer. This causes the slave device to send an additional data packet back to the Blackfin processor, where it is received in the USB FIFO, but no USB RX interrupt is generated. Taking the Mass Storage Class as a specific example, this causes the devices to send a status packet, which should generate a USB RX interrupt, however, this interrupt may be lost.

For DMA Mode 1 device mode receive operations where the transfer size is unknown, the Short Packet Interrupt must be relied upon to indicate the end of the transfer. However, this anomaly prevents the USB controller from issuing an RX interrupt for the corresponding endpoint when a short/null packet is received.

This anomaly does not apply to device mode when the size of the receive transfer is known in advance, as the DMA Completion Interrupt is generated at the end of the transfer and the endpoint receive interrupt is not used.

This anomaly also does not apply to transmit operations.

WORKAROUND:

In all affected cases described above, use DMA Mode 0 instead of DMA Mode 1.

APPLIES TO REVISION(S):

0.0, 0.1

40. 05000457 - Host DMA Port Responds to Certain Bus Activity Without $\overline{\text{HOST_CE}}$ Assertion:

DESCRIPTION:

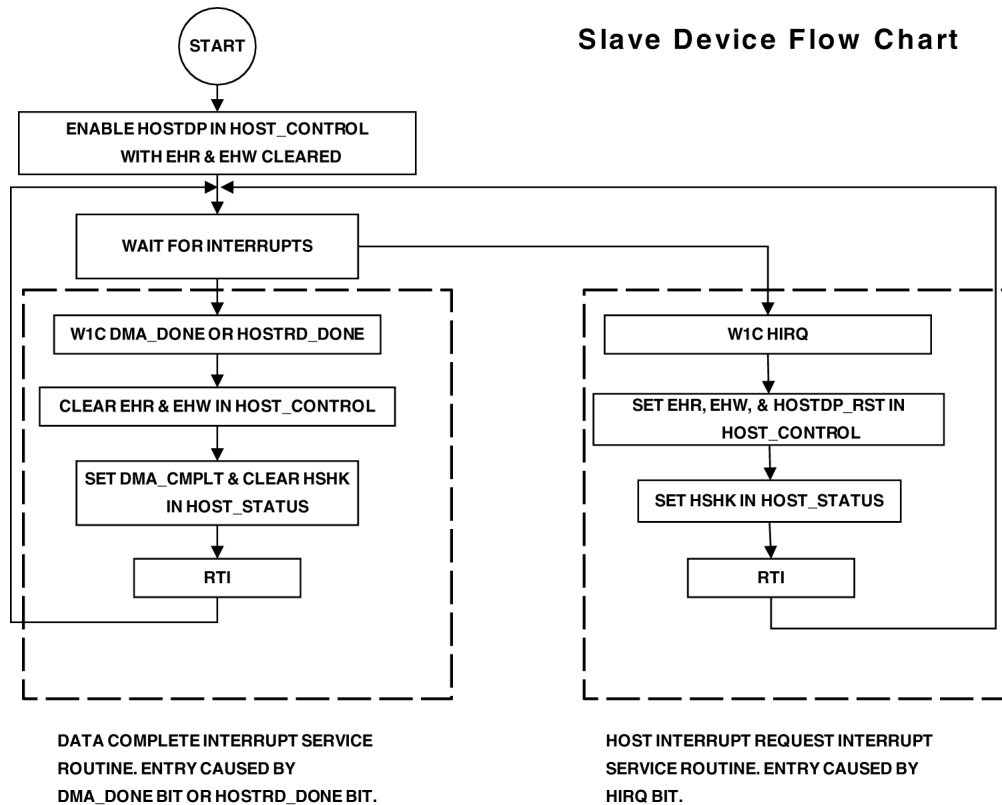
The Host DMA Port (HOSTDP) responds to certain bus activity even without the assertion of its chip enable, $\overline{\text{HOST_CE}}$. If the HOSTDP is the only slave on the bus (meaning that $\overline{\text{HOST_WR}}$ and $\overline{\text{HOST_RD}}$ are asserted by the host only for communicating with the HOSTDP), this anomaly will not be observed. There are two states in which the HOSTDP responds to bus activity (assertion of $\overline{\text{HOST_WR}}$ or $\overline{\text{HOST_RD}}$) without $\overline{\text{HOST_CE}}$ being asserted:

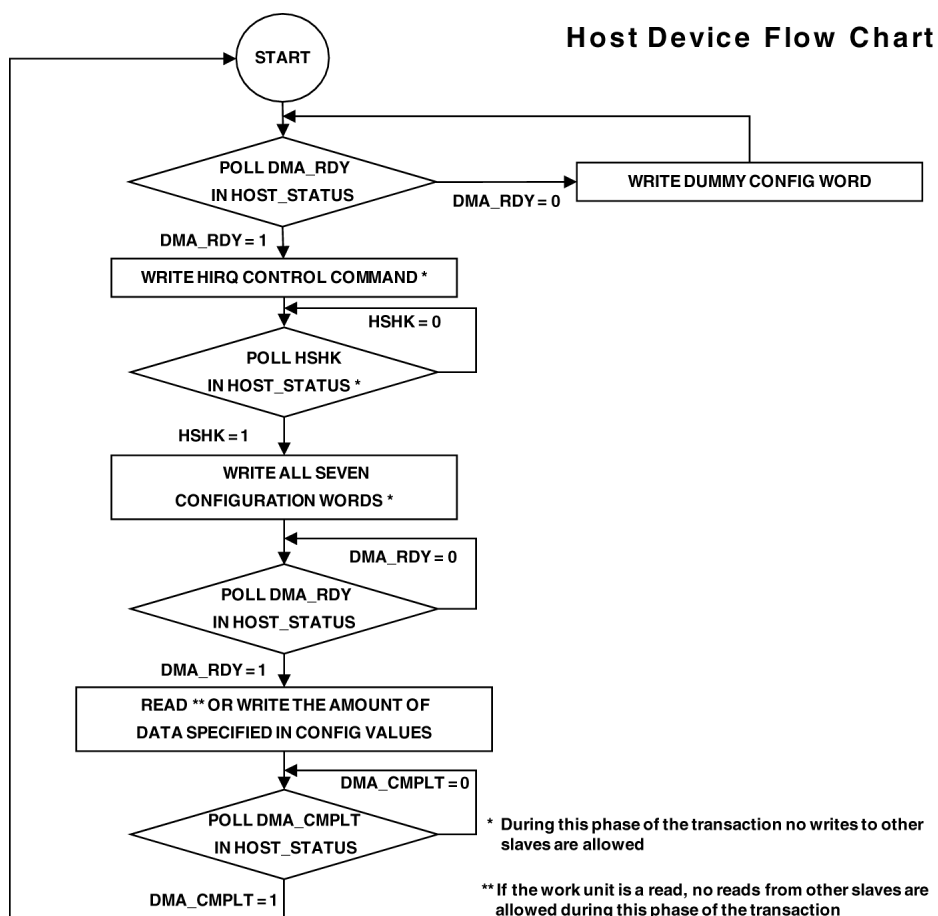
1. HOSTDP is Enabled and Waiting for the Host to Configure - While waiting for configuration in both acknowledge and interrupt modes, assertions of $\overline{\text{HOST_WR}}$ for other slaves can cause the HOSTDP to be erroneously configured.
2. HOSTDP is Configured for Data Reads - While waiting for data reads in both acknowledge and interrupt modes, assertions of $\overline{\text{HOST_RD}}$ for other slaves can cause the HOSTDP to subsequently return data out of order.

WORKAROUND:

The following workarounds can be used for state #1:

- a. Add additional logic - Connect to $\overline{\text{HOST_WR}}$ through a 2-input OR gate where one input of the OR gate is the host's write signal and the other is the host's chip select.
- b. Time bus accesses by the host processor - Do not write to other slaves on the bus between the time that the HOSTDP is enabled and the DMA_RDY bit in HOST_STATUS is set.
- c. Change the software on both the host and the slave as shown in the following flowcharts:





The following workarounds can be used for state #2:

- a. Add additional logic - Connect to $\overline{\text{HOST_RD}}$ through a 2-input OR gate where one input of the OR gate is the host's read signal and the other is the host's chip select.
- b. Time bus accesses by the host processor - Do not read from other slaves on the bus between the time that the final configuration word (YMODIFY) is written to the HOSTDP and the DMA_CPLT bit in HOST_STATUS is set.

APPLIES TO REVISION(S):

0.0, 0.1