



Technical notes on using Analog Devices DSPs, processors and development tools  
Visit our Web resources <http://www.analog.com/ee-notes> and <http://www.analog.com/processors> or  
e-mail [processor.support@analog.com](mailto:processor.support@analog.com) or [processor.tools.support@analog.com](mailto:processor.tools.support@analog.com) for technical support.

## Designing and Debugging Systems with SHARC® Processors

Contributed by Aseem Vasudev Prabhugaonkar and Alberto Comaschi

Rev 1 – November 13, 2006

### Introduction

This EE-Note provides vital information related to designing systems with SHARC® processors. These guidelines are intended to assist hardware engineers as well as firmware or software engineers in reducing design cycle times. Some of the recommendations below are also described in the *Hardware Reference* manuals of the respective SHARC processors. This EE-Note is divided into three sections: hardware board design guidelines, software procedures and tricks, and debugging tips. The tips apply to all SHARC processors unless specified otherwise.

### Hardware and Board Design Check Points

This section provides tips for board designers.

#### SPI Interface

*SPI Boot Interface on ADSP-2126x and ADSP-21362/3/4/5/6 vs. ADSP-21367/8/9 and ADSP-2137x SHARC Processors*

The SHARC processors listed above support boot-loading from SPI memory device. When configured for SPI flash boot, they can boot-load the application image from an on-board SPI memory.

Most of the commonly used SPI flash devices require a falling edge on chip select ( $/CS$ ) prior to the start of an instruction after power up. For ADSP-2126x and ADSP-21362/3/4/5/6 processors, pull up the SPI flash chip select using a 4.7k $\Omega$  resistor. These SHARC processors do not actively provide logic-high to logic-low transition due to the absence of an internal pull-up on this signal; therefore, the SPI flash select may remain at a low logic level (or be undefined) before the first instruction is driven by the processor. The pull-up resistor is not required for designs with ADSP-21367/8/9 and ADSP-2137x processors because they have an internal pull-up resistor to ensure the falling edge requirement during an SPI flash boot-loading operation. One example of an SPI flash memory device that has this requirement is the ST Microelectronics MP25P80 serial flash device.

#### MOSI and MISO

The SPI interface requires that all MOSI pins are tied together and all MISO pins are tied together. To prevent contention and possible damage to the pins, double-check that these pins have not been interchanged. Connect MISO to MISO, and connect MOSI to MOSI. If the peripheral pin names are DIN or DOUT, connect them according to their master or slave function. Proper schematic signal names will reduce confusion.

## JTAG Design and Booting Issues

Most systems are initially designed for JTAG connectivity, so that the prototypes and pre-production units may be tested and debugged with a JTAG ICE (in-circuit emulator). In this case, the JTAG  $\overline{\text{TRST}}$  signal (TAP reset) is driven by the ICE. However, if the system runs in standalone mode by booting operation or whenever the ICE is not used, connect the  $\overline{\text{TRST}}$  signal to board ground. Failing to ground the  $\overline{\text{TRST}}$  signal can lead to boot failures and memory access failures during runtime. Furthermore, pull-down resistors on  $\overline{\text{TRST}}$  are not recommended because SHARC processors already have an on-chip pull-up resistor on this signal. Refer to *Analog Devices JTAG Emulation Technical Reference (EE-68)*<sup>[1]</sup> for JTAG system design guidelines.

## Two-Wire Interface (TWI)

The on-chip two-wire interface is an I<sup>2</sup>C-compatible peripheral. Because SCL and SDA are open drain, both TWI signals require pull-up resistors as stated in the I<sup>2</sup>C standard. (Refer to the I<sup>2</sup>C standard and I<sup>2</sup>C-compatible device data sheets to select the pull-up resistor's value.)

## Driving the $\overline{\text{RESET}}$ Input

Avoid the use of RC (resistor/capacitor) circuitry to drive the SHARC processor's  $\overline{\text{RESET}}$  input signal. Analog Devices recommends power monitoring ICs to provide power-on and manual  $\overline{\text{RESET}}$  to the processor. An RC network in combination with Schmitt trigger level gates may also be used to drive the  $\overline{\text{RESET}}$  input.

## Bypass Capacitors

Appropriate bypass capacitors on the internal power supply become critical at higher operating speeds. Unwanted parasitic inductance in capacitors and traces reduces the effectiveness at high frequency. Two things are needed when processors operate above 100 MHz. First, capacitors should be physically small and their leads should be short to reduce inductance. Surface mounted capacitors of size 0402 will yield better results than larger sizes. Second, lower values of capacitance will raise the resonant frequency of the LC circuit. Although several 0.1 uF capacitors work well below 50 MHz, a mix of 0.1, 0.01, 0.001 uF and even 100 pF is preferred in the 500-MHz range for VDD\_INT.

## Filter Circuitry for AVDD Supply

This section applies to ADSP-2116x, ADSP-2126x, and ADSP-21362/3/4/5/6 SHARC processors. The data sheets for these devices recommend filter circuitry for the AVDD supply used by the on-chip PLL. Older revisions of the data sheets suggested using a 10 $\Omega$  series resistor; for better noise immunity and PLL stability, this rating has now been updated to a high-impedance (600- to 1000 $\Omega$  @100 MHz) ferrite bead instead. These types of ferrite beads have a DC resistance of less than 1 $\Omega$ .

## Unused Input Signals

Never leave unused processor inputs floating. Based on the active polarity of the inputs signal, use a pull-up resistor or a pull-down resistor. The recommended value of the resistors is 10k $\Omega$  for pull-up resistors and 100 $\Omega$  for pull-down resistors. Only those inputs that have an internal pull-up/pull-down resistor may be left floating. Check the device data sheet to identify the inputs that have internal pull-up/down resistors

by default. Leaving signals such as un-used bus requests (`/BRx`) and host bust request (`/HBR`) floating may cause boot failures and other issues during application runtime.

### EZ-KIT Lite® Schematics

The EZ-KIT Lite® evaluation system schematics are a good starting reference. Because the EZ-KIT Lite board is for evaluation and development, extra circuitry is provided in some cases. Read the EZ-KIT Lite board schematic carefully, because sometimes a component is not populated and sometimes it has been added to make it easier to access, etc. The design database for the SHARC processor EZ-KIT Lite boards is available online and contains all of the electronic information required for design, layout, fabrication, and assembly: [ftp://ftp.analog.com/pub/tools/Hardware/Reference\\_Designs](ftp://ftp.analog.com/pub/tools/Hardware/Reference_Designs)

### Test Points and Signal Access

The debug process can be aided on the prototype boards by adding test points on signals such as `CLKOUT/RSTOUT`, SDRAM clock (`SDCLK`), `/MSx` memory bank selects, `/BMS`, and `/RESET`. If selection pins such as boot mode (`BOOTCFG`) or core clock rate (`CLKCFG`) are connected directly to power or ground, they will be inaccessible under a BGA-package chip. For debugging, it is helpful to use pull-up and pull-down resistors instead of tying to power or ground directly.

### Signal Integrity Tips

Rapid signal rise times and fall times are a primary cause of signal integrity problems. These edge rates on SHARC processors differ from pin to pin. Likewise, some pins have greater sensitivity to noise and reflections than other pins. Use simple signal integrity methods to prevent transmission-line reflections that may cause extraneous clock and sync signals. Short trace lengths and series termination are critical for the following peripherals and signals:

- SPORT interface signals (`TCLK`, `RCLK`, `RFS`, and `TFS`): Noise or glitches on these signals can cause improper SPORT functionality. Symptoms such as SPORT lock-up condition, channel swap, channel shift, and data corruption may appear due to glitches on these signals. For this reason, in cases of long traces or simulation results that indicate reflections, use termination resistors on these lines.
- `CLKIN` source: When using a crystal as the source of `CLKIN`, use capacitors for the crystal, as recommended by the crystal manufacturer. Use the fundamental mode crystal as often as possible. When using an oscillator as the source of `CLKIN`, leave the `XTAL` pin of the processor N/C (not connected). Refer to the data sheet for `XTAL/CLKIN` circuitry design and for recommended component values. Avoid routing high-speed signals (for example SDRAM clock, external port data bus, or address bus) close to (or beneath) the `XTAL/CLKIN` signal and circuitry. Crosstalk may induce noise, affecting PLL performance. When using external oscillators to drive `CLKIN`, use spread spectrum oscillators to reduce EMI emissions due to the clock source. Use low-skew clock buffers/drivers when designing systems with multiple SHARC processors, taking `CLKIN` from a single clock oscillator.
- SDRAM clocks, control, address, and data also benefit from short traces and series termination to prevent reflections and reduce unwanted EMI.
- As often as possible, avoid using sockets for ICs such as the memory chips. Sockets degrade the signal integrity performance by adding unnecessary parasitic. In cases where signals have multiple sources, it

may be difficult to keep the traces short, and simulation may be appropriate. IBIS models that can assist with signal simulation are available from the Analog Devices Web site.

### General Guidelines and Pointers

- Powering up the SHARC processors: Power up both power supplies (VDDINT [core power supply] and VDDEXT [IO power supply]) simultaneously. If they can not be brought up simultaneously, the time difference between stable power supplies should not exceed the specification in the data sheet ( $t_{VDDDEVDD}$  [VDDINT before VDDEXT]).
- Multiplexed signals: Beware of signals that are multiplexed and have I/O functionality. These signals have a default functionality after power on /RESET; software programming is necessary to change the functionality from the default functionality to a desired functionality. From a system design standpoint; this may cause contention of signals. Some examples are described below.
  - On ADSP-21367/8/9 SHARC processors, /MS2 and /MS3 strobes are multiplexed with flag and interrupt pins. After power on, these signals are configured as inputs. Therefore, a pull-up resistor should be used for these signals if these signals are used as memory select signals to avoid external port contention. For example, /MS1 (bank 1) is used to perform boot-load from external flash memory. Memory devices interfaced to /MS2 or /MS3 can misinterpret the logic level on their chip select pin and start driving the bus, causing bus contention.
  - Another example is RSTOUT/CLKOUT signal multiplexed with running reset functionality on ADSP-2137x SHARC processors (see [Figure 1](#)). At and after power on /RESET, this signal behaves as an RSTOUT signal. Execute software to configure this signal as an input and running reset signal. RSTOUT/CLKOUT signal on ADSP-2137x processors should be driven by the OD (open drain) output of the host when used to achieve running reset functionality. During and after power up, this signal is configured to be an output until the software configures this as an input and running reset signal. Connecting this to an active drain output of the host may cause contention and may damage the drivers.
- Boot memory chip select: ADSP-2106x and ADSP-2116x SHARC processors have a dedicated signal (/BMS [byte memory select]) to drive the parallel boot memory device. For ADSP-2126x and ADSP-21362/3/4/5/6 processors, there is no dedicated boot memory select signal; therefore, memory selects must be derived from the parallel port address driven by the processor. For ADSP-21367/8/9 and ADSP-2137x processors, /MS1 (memory bank 1 select) must be used as boot memory select signal. Booting occurs through bank 1. There is no dedicated signal like /BMS for ADSP-21367/8/9 and ADSP-2137x SHARC processors.

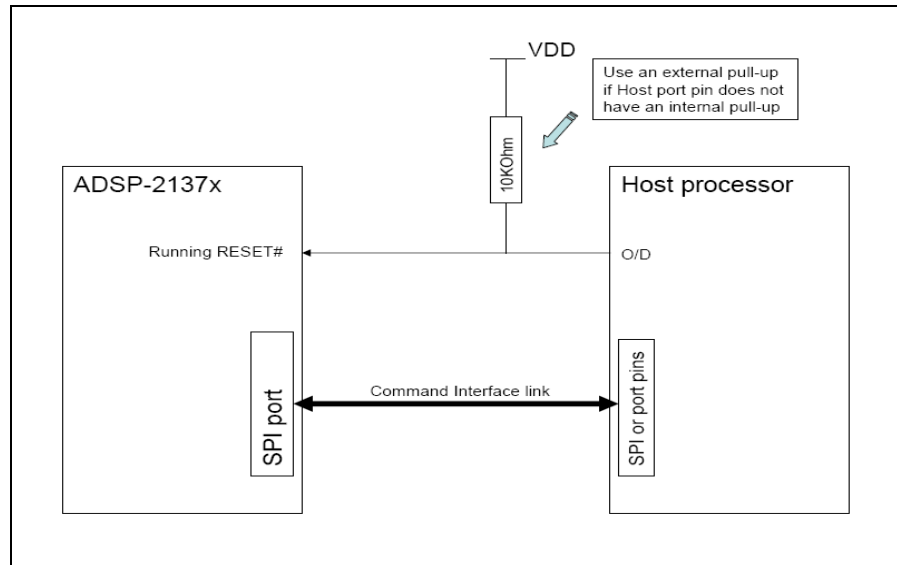


Figure 1. Typical system block diagram showing running reset scheme

- SDRAM address pin mapping and  $DQM$  signals: ADSP-21367/8/9 and ADSP-2137x SHARC processors have different address pin mappings compared to ADSP-21161 processors. When used in 32-bit mode, connect ADDR1 on ADSP-21367/8/9 and ADSP-2137x processors to ADDR0 of the SDRAM. When used in 16-bit mode, connect ADDR0 to ADDR0 of the SDRAM. Unlike ADSP-21161 processors, there is no  $DQM$  support on ADSP-21367/8/9 and ADSP-2137x processors. Certain SDRAM devices have a specific  $DQM$  requirement during an SDRAM power-up sequence. For such SDRAM memories, the  $DQM$  signal must be driven using a processor's flag pin during the SDRAM power-up sequence. For details about the SDRAM interface with ADSP-21367/8/9 processors, refer to *Interfacing 133 MHz SDRAM Memory to ADSP-21367 SHARC Processors (EE-286)*<sup>[2]</sup>.

## Software Procedures and Tips

This section highlights recommended procedures for certain peripherals and units of SHARC processors.

### PCG (Peripheral Clock Generator) Programming

This section deals with considerations to be taken in to account while using PCGs (Precision Clock Generators) to generate signals for SHARC SPORTs.

- When using PCGs to generate signals such as  $FS$  (frame sync as  $LRCLK$ ) and  $SCLK$  (serial bit clock) for a SPORT configured in  $I^2S$  mode, it is necessary to program the delay registers of PCG to ensure that the signal timings confirm with the  $I^2S$  protocol timings. The  $LRCLK$  signal must be driven at the falling edge of serial clock.
- PCG when used to generate signals for SPORTs configured in TDM mode requires that the frame sync remains active only for one serial bit clock cycle. Note that the frame syncs in TDM mode are level sensitive, not edge sensitive. The PCG pulse width register ( $PCG\_PW$ ) should be loaded with a value of "1" so the frame sync is active for a single serial clock cycle. In this configuration, it does not matter when the SPORT is enabled or re-enabled because the frame sync is active for a single serial clock

cycle. If the SPORT is enabled while the frame sync is inactive, it will wait for the next active frame sync. In a system where PCG configuration demands a 50% duty cycle, use an internal DAI interrupt to enable the SPORT.

Consider a scenario where PCG frame sync and clock have started before the SPORT is enabled or re-enabled. As serial ports can be enabled at any point in time after the frame sync and clock have started, there could be situations where the serial port starts latching the data (in the case of a receiver) or starts driving the data (in the case of a transmitter) whenever a valid frame sync is sampled. Note that the frame sync is level sensitive and not edge sensitive in TDM mode. This situation will repeat whenever a serial port is disabled and re-enabled, causing loss of synchronization. This may cause symptoms such as channel shift and data loss. The correct procedure before starting the serial port would be to wait for the inactive edge of frame sync and then enable the serial ports while the frame sync is still inactive. This ensures that the serial port waits for an active frame sync level and samples the data.

This procedure should also be repeated when serial ports are disabled and re-enabled to ensure that the serial port does not sample data in the middle of active frame. This is implemented using the following procedure. PCG frame sync is mapped to a DAI interrupt. An inactive edge of the frame sync asserts the DAI interrupt. This is an indication that the frame sync is inactive for some duration. The serial port should be enabled within this duration while the frame sync is inactive. Perform the same procedure when a serial port is disabled and re-enabled.

### **Enabling I2S Using IDPs (Input Data Ports)**

When IDPs are used to receive data from external devices, there is a sequence to be followed to enable the IDP ports when configured to receive data in I2S mode. Failing to follow this sequence can give rise to channel shift or swap.

1. Connect the frame sync internally using the SRU (Signal Routing Unit) to the DAI interrupt.
2. Configure the DAI interrupt for the inactive edge of the frame sync.
3. Wait for the DAI interrupt, and enable the IDP port inside the DAI interrupt service routine.
4. Clear the DAI interrupt by reading the DAI interrupt latch register.

This procedure will ensure that the IDP ports are enabled at the correct time, avoiding issues like channel shift or swap in the received data.

### **DAI Interrupts**

Unlike other interrupts, the DAI interrupt is not cleared automatically inside the DAI interrupt service routine. Once latched, the interrupt remains latched until the latch status is cleared explicitly by reading the DAI interrupt latch register. DAI interrupts are cleared by reading the DAI interrupt latch register. If this does not occur, the program will keep sequencing to the DAI interrupt service routine every time after coming out of it.

### **Disabling Peripheral Inside Its DMA Interrupt Service Routine**

SHARC processor peripherals support DMA mode for data transfer. The DMA interrupt is generated when the DMA transfer count expires. When a peripheral is configured to receive data, the DMA count



expires when the entire data is received and moved in to the internal memory. When a peripheral is configured to transmit data out, data remains in the DMA FIFO of the peripheral even though the DMA count expires and generates an interrupt. If the software intends to disable the DMA and the peripheral inside the ISR, it should poll the DMA FIFO status first. When the FIFO shows empty status, the peripheral can be safely disabled; otherwise, data loss may occur.

### Double SPORT/SPI Interrupts

This section applies to ADSP-21367/8/9 and ADSP-2137x SHARC processors only. Consider a scenario where SPI is used in core mode to transmit data. When the core encounters a transmit buffer empty condition, a transmit interrupt is generated. Inside the transmit interrupt service routine, the software writes a new value to the transmit buffer and returns from interrupt. In such a situation, after returning from an ISR, the processor again detects a transmit buffer empty condition and sequences to SPI transmit ISR. This happens due to highly pipelined IOP writes. After writing in to transmit buffer it takes 10 core clock cycles for the actual value to be written and update the transmit buffer status to "not empty". Therefore, under such scenarios, delay the return from interrupt routine for at least 10 core clock cycles.

### SHARC PLL Programming

For ADSP-2116x SHARC processors, the external `CLKCFG` (clock configuration) signal configures the PLL (and the core clock and external port clock frequency). For ADSP-2126x, ADSP-2136x, and ADSP-2137x SHARC processors, in addition to the external `CLKCFG` signals, the PLL can be configured in software. Thus, various programmable ratios are possible in software by using the PLL multiplier and divisor counts, which are programmed in the power management control register (`PMCTL`). This also provides flexibility to the user to change the core frequency in software. For proper PLL configuration, follow the recommended sequence, as described in *Managing the Core PLL on ADSP-2136x SHARC Processors (EE-290)*<sup>[3]</sup>. Ensure that the `DIVEN` bit is cleared before switching to bypass mode and before switching out of by-pass mode. EE-290 also provides code examples and a step-by-step procedure to integrate the PLL configuration "C-callable" function in to the library using the `elfar` utility.

### Interfacing SPORTs to Gated Clock Devices

Certain system designs require interfacing SHARC SPORTs to gated clock devices such as data converters or the SPI of host processors. Guidelines to ensure proper operation and data transfer with gated clock devices with examples are discussed in *Interfacing Gated Clocks to ADSP-21065L SHARC Processors (EE-244)*<sup>[4]</sup>, *Interfacing AD7676 ADCs to ADSP-21065L SHARC Processors (EE-247)*<sup>[5]</sup>, and *Interfacing AD7676 ADCs to ADSP-21365 SHARC Processors (EE-248)*<sup>[6]</sup>.

## Debugging Tips

It is often said that prevention is better than cure! System design engineers and software engineers should always consult the anomaly documentation of the SHARC processor they plan to use. This ensures awareness of known silicon issues and proposed workarounds, and shortens design cycle time. For issues that must be dealt with in hardware (such as board design), it helps you to avoid expensive and time-consuming board re-spins.

This section considers commonly encountered issues and provides tips on how to debug them. These tips supplement your other debugging efforts. The most commonly reported scenarios are discussed, as follows.

### Boot Failure

These are usually reported as cases where a user is able to load and execute an application using a JTAG ICE, but is unable to execute the applications when boot-loaded. Consider the following points while debugging:

- Check the `BOOTCFG` pins of processor and ensure that they are properly connected to digital voltage level high or level low. When resistors are used to provide flexibility to change boot mode, ensure that the right ones are populated. The best way is to check the voltage levels with an oscilloscope at the processor pins.
- Ensure that the `/TRST` signal of the JTAG ICE is connected to board ground. Do not leave this signal floating. Letting this signal float may cause boot failures or other memory access failures.
- Ensure that the correct boot kernel is used before generating the loader (`.LDR`) file. If you are using a modified boot kernel, try using the default boot kernel supplied with VisualDSP++® together with an example application (like flag toggle) to confirm basic boot-loading.
- Ensure that you have selected the correct parameters while generating the `.LDR` file. Selecting an inappropriate parameter may cause the boot to fail.
- Check whether the code boot-loaded using an ICE (in-circuit emulator). Connect the ICE to the target and open a simulator session in VisualDSP++. Turn on the target to start boot-loading. Change the session from “simulator” to “emulator” and look at the Disassembly window. This shows whether the application boot-loaded from the external source. If you do not see the expected code, there is a boot failure. You can also use this method to check whether the processor downloaded the initial 256 instructions.
- Ensure correct power on reset timings as specified in the data sheet.
- Check the `CLKCFG` signals and ensure that the PLL is not overdriven. Ensure that the ratio selected in combination with the `CLKIN` frequency does not exceed the core clock to a value greater than specified.
- Probe boot-related signal like `/BMS` (`/MS1` for ADSP-21367/8/9 and ADSP-2137x processors), `/RD`, `ADDR`, and `DATA` on the bus. Check for the integrity of tracks on the board as there may be open circuits or short circuits on the board. You can also monitor `RSTOUT` to confirm the PLL lock.



- Check whether there is bus contention on the external bus during a boot-load operation. For example, for ADSP-21367/8/9 and ADSP-2137x processors, `/MS2` and `/MS3` (if used) should have external pull-up resistors. Otherwise, memory devices interfaced to `/MS2` and `/MS3` can drive bus causing bus contention at boot up.

### Application Crash

Most of the time, this problem is reported as a software crash while running the application. There are cases where the processor simply hangs after sequencing to an unknown location or restarts the whole application. Several things may cause this to happen, including:

- Processor performing indirect accesses which can cause corruption of heap or stack memory locations defined in the `.LDF` file. This may cause corruption of vital data needed during run time and therefore cause a software crash. Ensure that the application does not perform such memory accesses, which can give rise to corruption. At runtime, this can be debugged using the hardware breakpoint feature of VisualDSP++ using an ICE. If the processor performs illegal access in those regions, these accesses can be detected.
- Check whether the application performs any harmful/unintentional accesses to system registers or IOP registers that may cause the application to hang or restart. For example, check if the application is setting soft-restart/re-boot by mistake anywhere in the code. Check for indirect jumps or calls.
- Board designs with insufficient decoupling/bypass capacitors can lead to unpredictable behavior, often erroneously correlated with particular code execution. Ensure that the board has proper decoupling capacitors and bulk capacitors (capacitors type/value and placement is critical) for the processor and other devices so as to provide sufficient switching current/power required during system runtime. Heavy core or I/O switching demands excessive switching currents; if the current is not provided, possible symptoms include system shutdown, application crash, or incorrect execution of specific code.
- It is a good practice to have check-sum validation for boot-loaded applications to ensure that the entire data and code is downloaded correctly without any bit errors. Bit errors may occur in a noisy environment. Due to bit errors, a processor might execute the wrong (or illegal) instructions or process bad data, giving rise to unpredicted behavior.
- Ensure that the length registers (`Lx`) of DAGs (data address generators) are explicitly initialized to zero when not used as circular buffers. When an ICE is used to download code and execute, symptoms related to this may not be noticed because the ICE would have initialized the length registers to zero and hence the symptoms would be noticed only when the code is boot-loaded and executed as in a stand-alone system.
- Check for any known VisualDSP++ compiler bugs related to a specific tool version (or update). The compiler may generate bad code, causing unpredicted behavior. It is recommended that you always use the latest version of VisualDSP++.

The latest VisualDSP++ release notes are located at the Analog Devices Web site:

[www.analog.com/processors/sharc/evaluationDevelopment/crosscore/toolsUpgrades/index.html](http://www.analog.com/processors/sharc/evaluationDevelopment/crosscore/toolsUpgrades/index.html)

Also, release notes for older VisualDSP++ versions are located at:

[www.analog.com/processors/sharc/evaluationDevelopment/crosscore/toolsUpgrades/archives.html](http://www.analog.com/processors/sharc/evaluationDevelopment/crosscore/toolsUpgrades/archives.html)

### **Data Loss**

This is reported as the loss of the first few words of data while receiving over SPORTs, SPI, or other serial interfaces. In such cases, ensure that the slave is always enabled before enabling the master. The master generates signals such as serial clock and frame sync, and the slave takes these signals from a master device.

### **Data Corruption**

This is reported as corruption in data when received over serial links like SPORTs, SPI, link ports, and so on. In such cases, ensure signal integrity and consider issues due to overshoots/undershoots and noise spikes due to crosstalk or other sources that induce noise. Try performing internal loopback wherever possible to rule out functional issues with the processor. Look for SI issues using an oscilloscope in infinite persistence mode. Under such cases, try using terminations (preferably series termination) with serial clock, frame sync, and so on. Sometimes these issues show up as shift/swap in received channels even when other functional recommendations have been implemented in design.

### **VisualDSP++ Portability Issues**

These are usually reported as issues seen when the version (or update) of VisualDSP++ is changed. In such cases, check for differences in the compiler, known tools issues, etc. Try turning off code optimization to further pin-point compiler optimization-related issues. Use the latest version/update of VisualDSP++ available at the beginning of project as often as possible.

## Appendix

Board Design Checklist	
<input type="checkbox"/>	Check the size and placement of decoupling capacitors
<input type="checkbox"/>	Check the boot mode selection pins (BOOTCFG)
<input type="checkbox"/>	Check the core clock rate selection pins (CLKCFG)
<input type="checkbox"/>	Check JTAG pins configurations. See <i>JTAG Emulation Technical Reference (EE-68)</i> <sup>[1]</sup>
<input type="checkbox"/>	Check test points and probe point accessibility. Unused PF pins are very useful for hardware and software debug purposes (for example driving LEDs or probe points). If possible, pin out all DSP signals through vias.
<input type="checkbox"/>	Check the reset circuitry. Reset should be generated by a reset supervisory IC, not by RC circuitry. RC network use is possible in combination with a Schmitt trigger gate to drive RESET/ of the processor.
<input type="checkbox"/>	Check power supply ratings and thermal requirements. It is preferable to be able to adjust the core voltage by simply changing a resistor value.
<input type="checkbox"/>	Check component placement, and avoid long traces
<input type="checkbox"/>	Check the processor's data sheet and errata for up-to-date information

## References

- [1] *Analog Devices JTAG Emulation Technical Reference (EE-68)*, Rev 9, October 2004, Analog Devices, Inc.
- [2] *Interfacing 133 MHz SDRAM Memory to ADSP-21367 SHARC Processors (EE-286)*, Rev 2, March 2006, Analog Devices, Inc.
- [3] *Managing the Core PLL on ADSP-2136x SHARC Processors (EE-290)*, Rev 1, June 2006, Analog Devices, Inc.
- [4] *Interfacing Gated Clocks to ADSP-21065L SHARC Processors (EE-244)*, Rev 1, September 2004, Analog Devices, Inc.
- [5] *Interfacing AD7676 ADCs to ADSP-21065L SHARC Processors (EE-247)*, Rev 1, October 2004, Analog Devices, Inc.
- [6] *Interfacing AD7676 ADCs to ADSP-21365 SHARC Processors (EE-248)*, Rev 1, October 2004, Analog Devices, Inc.
- [7] *Hardware Design Checklist for the Blackfin Processors (EE-281)*, Rev 1, October 2005, Analog Devices, Inc.

## Document History

Revision	Description
<i>Rev 1 – November 13, 2006 by Alberto Comaschi and Aseem Vasudev Prabhugaonkar</i>	Initial Release