CrossCore® Embedded Studio Release Notes

2012 March 29

Table of Contents

Preface

Thank you for purchasing CrossCore® Embedded Studio for Analog Devices Processors.

The first release of our latest development environment, CrossCore Embedded Studio 1.0.0 is designed to make developing software applications for Analog Devices processors even easier.

CrossCore Embedded Studio represents a major step forward in embedded system development. It combines an industry-leading integrated development environment (IDE) with Analog Devices' advanced optimizing compiler technology. It also supports standards such as ISO/IEC C and C++, MISRA-C, Embedded C and MCAPI™, plus additional features to get your products to market even sooner.

Purpose of This Document

This document briefly introduces features of CrossCore Embedded Studio 1.0.0, which supports the ADSP-BF60x family of Analog Devices Blackfin processors.

Details of this product and other products in the CrossCore family are found in the "Related Documents" section of these release notes or in the online Help that is accessible from within CrossCore Embedded Studio.

Intended Audience

This publication is primarily intended for programmers looking for a short overview of CrossCore Embedded Studio. For additional introductory information, you can view the video tutorials that are available through the "Welcome Page" that appears when you first start using CrossCore Embedded Studio, or at:

http://videos.analog.com/category/products/processors-dsp/ .

Technical or Customer Support

There are several options for contacting support:

- Submit your questions online at: http://www.analog.com/support
- E-mail your processor and DSP software and development tools questions from within CrossCore Embedded Studio. To do this: Go to "Help->E-mail Support…" This will create a new e-mail addressed to processor.tools.support@analog.com, and will automatically attach your CrossCore Embedded Studio version information (ProductInfo.html).
- E-mail your processor and DSP applications and processor questions to: processor.support@analog.com or processor.china@analog.com (Greater China support)

- Post your questions in the Processors and DSP online technical support community in Engineer Zone: http://ez.analog.com/community/dsp

Platform and Processor Support

This release of CrossCore Embedded Studio supports the ADSP-BF60x family, which are dual-core Blackfin processors. The following processors are supported:

ADSP-BF606
ADSP-BF607
ADSP-BF608
ADSP-BF609

Support for SHARC processors, and other processors in the Blackfin family will be available in a future update to CrossCore Embedded Studio.

Related Products and Documentation

CrossCore Embedded Studio can work in concert with other software add-ins and hardware as part of a comprehensive software development solution. All of these products install easily into the CrossCore Embedded Studio environment and are available through the Analog Devices web site for CrossCore Embedded Studio, www.analog.com/cces. Associated documentation can also be accessed there. This section identifies these products.

Software Add-Ins:

1. Real Time Kernel
2. USB Stack (Device)
3. File System
4. lwIP Lightweight TCP/IP Stack

Development Hardware:

1. ADSP-BF609 Evaluation Hardware for the ADSP-BF60x Blackfin Family of Processors
2. WVGA/LCD EI3 Extender Board
3. Video Decoder & Video Encoder EI3 Extender Boards
4. Audio EI3 Extender Board
5. Camera EI3 Extender Board
6. HPUSB and USB Emulators
7. ICE-100B Emulator

For target processor information, refer to your processor's hardware reference manual, programming reference, or data sheet. All documentation is available online and also directly from within the IDE without needing an external URL.

Online Technical Documentation

Full documentation for CCES and any optionally installed add-ins is available from your Windows start menu by clicking Analog Devices -> CrossCore Embedded Studio 1.0.0 -> CrossCore Embedded Studio Help or by clicking the Help -> Help Contents menu directly from within the IDE. This documentation includes the following manuals:

- Graphical Development Environment
- System Runtime Documentation
- Licensing Guide
- Assembler and Preprocessor Manual
- C/C++ Compiler and Library Manual for Blackfin® Processors
- Linker and Utilities Manual
- Loader and Utilities Manual
- Development Hardware Documentation
- ADSP-BF6xx Blackfin Processor Hardware Reference
- ADSP-BF5xx/BF60x Blackfin Processor Programming Reference

Additional documentation for Analog Devices processor hardware can be downloaded directly from the IDE by clicking Help -> Install New Software… and selecting "CrossCore Embedded Studio Software and Documentation" in the "work with" field of the Install dialog.

Introduction

This chapter describes CrossCore Embedded Studio, the requirements for running version 1.0.0 and some of the benefits provided by this release.

Product Release Description

CrossCore Embedded Studio integrates an Eclipse-based IDE with the latest versions of our mature code generation and debugging tools, enabling programmers to move easily between editing, debugging, and deployment of final products.

Release 1.0.0 includes the code generation tool chain comprised of the processor-specific software necessary for completing a project: assembler, C/C++ compiler and libraries, linker, loader, splitter, and utilities.

Release 1.0.0 System Requirements

To install and run Release 1.0.0, your computer must provide the following software, configuration, and system resources:

- 2 GHz single core Intel Pentium 32-bit processor (or x86 compatible); A 3.3 GHz or faster dual core machine is recommended.
- Windows XP Professional SP3 (32-bit only), Windows Vista Business Enterprise/Ultimate SP2 (32-bit only), Windows 7 Professional/Enterprise/Ultimate (32 and 64-bit)
- At least 1 GB of internal memory; 4 GB or more is recommended
- At least 2 GB of available disk space

Windows Vista and Windows 7 users may experience User Access Control (UAC) related errors if the software is installed into a protected location such as “Program Files” or “Program Files (x86)”. We recommend installing the software in a non-UAC-protected location. The default installation location is “C:/Analog Devices/CrossCore Embedded Studio 1.0.0”.

Getting Started with CrossCore Embedded Studio

The first time you start CrossCore Embedded Studio you will see the Welcome Screen which has links to learning resources to help you use this product effectively. You can return to the Welcome Screen anytime by selecting Welcome from the Help menu. The Welcome Screen has links to the following areas:

Getting Started Video

The Getting Started video will introduce you to the basics of this product. It’s recommended that new users begin by watching this informative presentation.

Working with Projects

The *Working with Projects* section gives you three easy ways to begin working with CrossCore Embedded Studio projects.

- Selecting “Create a new project” will open a dialog that guides users through the new project creation process, along with options for creating a customized Linker Description File and startup code.

- Selecting “Import an existing CCES project” will import an existing CrossCore Embedded Studio project into your workspace.

- Selecting the Example Browser will enable you to search all of the examples that have been installed on your system. You can search by any number of criteria including processor, processor family, platform, language, and keyword. Many of the related products in the CrossCore family also come with examples. When

these products are installed, their examples will also be available through the Example Browser.

Viewing Documentation

Under the *Viewing Documentation* heading you will find a link to open the online help facility and a link to find and download additional processor documentation.

Getting Help

Finally, the Welcome Page includes links to get additional help on CrossCore Embedded Studio.

Processors and DSP Community on EngineerZone: http://ez.analog.com/community/dsp. Here you can search the FAQs, blogs, and forums for more information. You can also use the forum to ask questions to the community.

Private Technical Support: http://www.analog.com/support. Submit a private support request to our technical support team.

Online Training Modules

One of the best ways to become familiar with CrossCore Embedded Studio is to visit the online training site at http://videos.analog.com/category/products/processors-dsp/. Here are some of the available training modules:

- CrossCore Embedded Studio Introduction and Overview
- Navigating Through the Eclipse Based IDE
- Creating, Configuring, and Building Projects
- Debugging on a Hardware Target
- Creating and Debugging a Boot Image
- System Services and Device Drivers in CrossCore Embedded Studio
- Transitioning from VDK to a new RTOS
- An introduction to the ADSP-BF609 Blackfin Processor

License Activation

The New License Wizard starts automatically the first time you run CrossCore Embedded Studio or thereafter if no valid licenses are detected. When the wizard appears, click Yes to start the New License Wizard. If you don't use the wizard when you first begin CrossCore Embedded Studio you can start the wizard later by choosing Help->Manage Licenses->New and the New License Wizard will appear.

On the first page of the wizard you will be asked whether you would like a free 90 day evaluation of the product or if you have a serial number that you would like to register. If you choose to enter a serial number then you will be asked to enter it at this time. Enter the number exactly as it appears, including dashes.

The next page will ask you to select a license activation method. If you have internet access the “one-step” activation is recommended. The one-step activation will install, register, and validate your license and you will be ready to begin using CrossCore Embedded Studio.

The alternative method of activation will install a temporary license so you can continue to use CrossCore Embedded Studio. You will need to register and activate your permanent license through the Analog Devices website.

Full information on CrossCore Embedded Studio licensing can be found by choosing Help->Help Contents->CrossCore Embedded Studio 1.0.0->Licensing Guide.

Notable Features

IDE

The CrossCore Embedded Studio IDE is based on the industry standard Eclipse environment. Eclipse features a best-in-class smart editing environment and a language-aware editor. The editor provides code completion of language constructs and source browsing of names.

The project environment features different perspectives, making it easy for users to switch back and forth between views pertinent to specific development tasks. One perspective can be used for editing and building your project while another can provide the visibility necessary for debugging your program.

In addition to the Eclipse features, Analog Devices has enhanced the environment to provide project customization. The build options can be easily changed to take best advantage of the code generation tools including the Optimizing C/C++ Compiler.

User Interfaces for Source Generation

The Startup Code/LDF add-in makes it easy to create and configure a DSP Project. It helps you create a new project with startup code that sets up peripherals like cache, DMA, and I/O, etc.

In additional to startup code, a linker description file (LDF) can be generated and added to the project to link in only necessary libraries, and specify SDRAM partitioning, etc. This takes away much of the complexity of configuring your application’s memory layout.

The Startup Code/LDF add-in automatically adds the generated files to your project, and updates them to reflect changes as you configure your project’s settings, while preserving your custom modifications to the generated files.

Analysis and Diagnostic Support

CrossCore Embedded Studio provides tools to more quickly identify common code errors at runtime and provide you with the information necessary to efficiently optimize your program.

Reports for Instrumented Profiling, Heap Debugging, and Code Coverage

CrossCore Embedded Studio provides support for generating various profiling and tracing reports in HTML that can be launched from within the IDE.

The compiler has several profiling and tracing facilities. These include:

- Heap debugging (.hpl files): To identify memory leaks, corruption and other problems relating to malloc and free.
- Instrumented profiling (.prf files): To identify the functions that consume the most cycles.
- Code coverage (.pgo files): To identify which parts of your application have been exercised by your testing.

Each of these facilities generates data files during application execution, containing profile or trace data. These files can be converted into HTML-formatted reports through the IDE.

To generate an HTML report, open the File menu, and select the New...>Code Analysis Report option. There is a choice for each of the report types.

For further details go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->C/C++ Compiler and Library Manual for Blackfin Processors, Chapter 2, “Analyzing Your Application” section.

Stack Overflow Detection

The Blackfin compiler supports stack overflow detection in CrossCore Embedded Studio 1.0.0. This facility is enabled through the -rtcheck-stack compiler switch, and instruments the generated code to check the stack pointer against the end of the stack. If an overflow is detected, the application jumps to adi_stack_overflowed, where a breakpoint is placed automatically by the IDE. For more information go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->C/C++ Compiler and Library Manual for Blackfin Processors, Chapter 2, “Stack Overflow Detection” section.

Fatal Error Diagnostic Information

In CrossCore Embedded Studio 1.0.0, the run-time libraries use a common, extensible API for reporting fatal application errors. When the API is invoked, the application transfers control to the adi_fatal_error function. The IDE automatically places a breakpoint on this label when your application is loaded, so the application halts. When this occurs, the IDE will retrieve diagnostic information passed to the API by the run-time library, and will display it in the console window.

Inter-core Communication

CrossCore Embedded Studio 1.0.0 includes support for the Multicore Communications API (MCAPI™), version 2.0, defined by The Multicore Association.

MCAPI allows communication between cores using

- unconnected messages
- connected packet streams
- connected scalar streams.

Communication within a core is also supported.

MCAPI is available on all Blackfin processors, but is most useful with dual-core processors such as ADSP-BF609. MCAPI is enabled by default when creating new projects for ADSP-BF609. MCAPI configuration can be added to your project via the System Configuration utility.

To see the MCAPI 2.0 Specification document, open CrossCore Embedded Studio Help: Help -> Help Contents->CrossCore Embedded Studio 1.0.0->System Runtime Documentation->Multicore Communication API (MCAPI) Specification.

The MCAPI examples can be found here: Help -> Browse Examples and search on “MCAPI”.

Peripheral Controller Driver Model

The driver model deployed in CrossCore Embedded Studio was designed with the following guidelines in mind:

- Ease of use
- Minimal footprint
- Minimal latency

For further information go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->System Runtime->System Services and device drivers

For examples and/or "Code Sketches" using the driver model go to Help->Browse Examples and search on the appropriate keywords (e.g. “audio”, “video”, “twi”, etc.) (Code Sketches are interactive code examples where input values can be modified to show their effect on the code in real time.)

Standardized Interrupt Management

To make the programming of Analog Devices processors consistent, the same APIs and methodology will be available for Blackfin and SHARC processors and supported operating systems in future updates. These interrupt management APIs begin with the adi_int prefix.

The types of interrupts that can be managed with these APIs are:

SHARC processors:
- All core interrupts

Blackfin processors:
- All core interrupt levels
- All system interrupts
- Exceptions
- Non-maskable interrupt (NMI)

The same interrupt APIs and methodology are also available for the following operating environments:
- Bare-metal (no operating system). Support for this configuration is shipped with CrossCore Embedded Studio
- Real time operating system. Support for this configuration is shipped in the real time kernel add-in available for CrossCore Embedded Studio.

This interrupt management mechanism is not only available for use within applications, but it is also the mechanism used by CrossCore Embedded Studio's system services and device drivers.

For further information go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->System Runtime Documentation->Interrupt Support.

Compiler Language Standards Support

Embedded C Support

The Blackfin compiler in CrossCore Embedded Studio 1.0.0 provides support for the native fixed-point types *fract* and *accum*, defined in Chapter 4 of the "Extensions to support embedded processors" ISO/IEC draft technical report TR 18037. These native fixed-point types allow you to write your applications in a more natural manner, without sacrificing performance.

For instance, the following function is an example of a dot product implemented using *fract* and *accum*, with natural fractional, saturating multiplication, addition, and assignment operators instead of built-in functions.

```
#include <stdfix.h>

accum dot_product(fract *a, fract *b, int n)
{
     accum sum = 0.0k;
     int i;
     for (i = 0; i < n; i++)
```

```
            sum += a[i] * b[i];
        return sum;
    }
```

For more information go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->C/C++ Compiler and Library Manual for Blackfin Processors, Chapter 1 “Using Native Fixed-Point Types”.

C99 and C++2003 Support

The compiler conforms to the ISO/IEC language standards:

- C99 is the default C language accepted by the compiler. This is a freestanding implementation of the ISO/IEC 9899:1999 C language standard.
- C89 mode is available through the -c89 compiler switch. In this mode, the compiler supports a hosted implementation of the ISO/IEC 9899:1990 C language standard.
- C++2003 is the default C++ language accepted by the compiler. This is a hosted implementation of the ISO/IEC 14882:2003 C++ language standard.

In all these modes, the compiler accepts Analog Devices’ language extensions. Some language features are standard features in some modes, and extensions in others; for further information go to Help-> Help Contents->CrossCore Embedded Studio 1.0.0->C/C++ Compiler and Library Manual for Blackfin Processors, and refer to the feature descriptions for details on standards conformance.

Full C++ Standard Library

The Blackfin compiler supports both the abridged and full C++ standard libraries. For more details go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->C/C++ Compiler and Library Manual for Blackfin Processors for documentation on the -full-cpplib switch.

MISRA-C:2004 Support

The compiler provides comprehensive support for MISRA-C: 2004, a set of guidelines published by the Motor Industry Software Reliability Association (MISRA). The compiler detects violations at compile-time, at link-time and at run-time.

For details go to Help->Help Contents->CrossCore Embedded Studio 1.0.0->C/C++ Compiler and Library Manual for Blackfin Processors and find the section “MISRA-C Compiler “in Chapter 1 “Compiler”.

Device Programmer

The Device Programmer is a utility for programming device memory, such as parallel or serial flash devices, on a target board. The Device Programmer is invoked in a command window as cldp, which is installed in the root folder of CrossCore Embedded Studio. The device programmer can also be added as a post build step in CrossCore

Embedded Studio to automatically program device memory after creating a Loader File artifact.

The Device Programmer interfaces with the target board using a device programmer interface application (dpia) that is installed separately from CrossCore Embedded Studio as part of the EZ-Board/EZ-Kit board support installation. If not interfacing to an Analog Devices, Inc. EZ-Board/EZ-Kit, then users must create their own dpia.

For more information on how to use the Device Programmer or creating a custom dpia please go Help->Help Contents->CrossCore Embedded Studio 1.0.0->Graphical Development Environment >Device Programmer.

Known Limitations

Supported Processors

This release of CrossCore Embedded Studio supports the ADSP-BF60x family of Blackfin processors. Support for SHARC processors, and other processors in the Blackfin family will be available in a future update to CrossCore Embedded Studio. Note that the documentation for CrossCore Embedded Studio already reflects our plans to include these other processors.

Note that many components of the toolchain for other processors are included in this release; however, they should not be considered production-quality.

Simulator

There is no simulator for the ADSP-BF60x family of processors. Consequently, to run programs for this family you will need the ADSP-BF609 EZ Kit Evaluation Hardware and its associated board support package or your own board with a BF60x processor and board support software.

Image Viewer

Support for the Image Viewer will be available in a future update.

Pipeline Viewer

There is no support for the Pipeline Viewer in this release.

ADSP-BF60x Loader

The CrossCore Embedded Studio loader does not provide a switch for creating forward blocks for the ADSP-BF609 where an entire boot stream can be contained in the payload of a block, and that entire stream can be forwarded to a peripheral.

Documentation

The following documentation is not up-to-date in this release. Instead the included documentation represents the toolchain and libraries at the time of the CrossCore Embedded Studio Beta Release.

- C/C++ Compiler for SHARC Processors.
- C/C++ Library Manual for SHARC Processors.

Anomalies

The following table is a list of known anomalies in CrossCore Embedded Studio 1.0.0.

Tar	Summary	Release Note
TAR-48052	part function names in call stack for C++ functions	When debugging an application, the displayed name for C++ functions in the call stack will not give all available information about the functions. Information about namespace scope, templates, etc. will be missing. For example, the function "my_test::array<short, long>::array(int)" will be displayed as "array(int)". This issue may make it difficult to navigate sources that use these standard C++ features
TAR-47733	Symbol manager should reset breakpoints when memory initialization is complete	If code that has breakpoints set in it is initialized using the runtime initialization support, the instruction at the breakpoints will be corrupted and the program will fail to execute properly. To avoid the problem, do one of the following: 1) Disable run-time initialization support when you need to rely on the use of breakpoints (for example, when debugging) 2) Ensure that code containing breakpoints is not initialized at runtime. This can be done in one of two ways: - by adding the following line to the the appropriate source file: #pragma file_attr("requiredForROMBoot") - By adding the function name (with a prefixed underscore) to the list of functions defined in the LDF as "OBJS_LIBS_WITH_BREAKPOINTS"
TAR-48245	cldp crashes with commands file that does not have a new line at the end	When using a command file(-@ filename), the command line device programmer will not work if the file does not have a new line at the end.
TAR-48048	Duplicate entries for HPUSB emulator seen during Found New Hardware Wizard	The USB device driver for the Analog Devices, Inc. line of emulators inadvertently did not receive an update of it's version. Therefore, there may be times when installing the driver where the user may be asked to pick between device drivers with the same exact version. It is safe to choose the oem*.inf file where * is a higher number.

TAR-48373	Memory Browser loses memory tabs on relaunch	If you create a new memory tab in the Memory Browser view and then re-launch your debug configuration using the "Relaunch" menu, it loses the memory tab that you just created. To reproduce: 1. Launch any debug configuration 2. Open the Memory Browser view 3. Type 0 for the address and click Go 4. In the Debug view, right-click on the top level node and choose "Relaunch" Workaround: Terminate your debug configuration first and then re-launch it.
TAR-48160	Warnings and errors when the project name contain "-"	Projects with a dash (-) in their name may fail to build To reproduce: 1. Install CCES, and create a project named TAR-48041, other things set as default 2. Build the new created projects, there are warnings 3. Open the project Properties setting page, Enable MISRA-C 4. Rebuild the project, there is a error indicate the name should not include "-" Workaround: Do not use a dash in the name of your project.
TAR-47906	Project paths that contain an ampersand (&) will not build	If a project path has the ampersand character in it then it fails to build. To reproduce: * Create a new project where the path to the project has a ampersand in it. * Build the project to see the error. Workaround: Do not use ampersands in the path or project name.
TAR-48189	String index out of range in Debug As, Debug Configuration menu	When there is more than one project open and you select a dxe to run or debug, you may see a pop-up about 'String index is out of range' and cannot load the dxe. Workaround: To load the dxe, select to run or debug from the project level instead of selecting

		the individual dxe.
TAR-47759	"Invalid format: STRING.Format" errors when debugging	When debugging an executable you may occasionally see a number of "Invalid format: STRING.Format" errors in the output console and no application output. Workaround: None. This error can be safely ignored.
TAR-48176	Not resolved errors in system/uCLIB/source lib files	When using the wizard to create a project that has uC/OS-III and then clicking on the system/uCLIB/source files lib_ascii, lib_mem.c, lib_math.c and lib_str.c files, you may see a number of errors in the Problems window. These are spurious errors detected by the editor and will not impact the ability to build or debug the project . To reproduce: * File, New, CrossCore Project. Select 609 and silicon rev any. Next. * Deselect MCAPI, Startup/LDF and pin muxing. Select RTOS. * Everything else default. * In Project Explorer, click on any of the files in system/uC-LIB/Source. Workaround: None
TAR-48254	Boot fails when Start Address (-p) with Initialization File (-init) due to incorrect NEXT PTR argument in initialization FIRST block	Do not use "Start address (-p <alternate-address>)" when building a ldr file with "Boot format Intel HEX (-f hex)" and "Initialization file (-init <init.dxe>)" at this release. Either: 1) Build the ldr file with "Boot format ASCII (-f ASCII)" with the default start address and specify the offset when programming the memory using the Device Programmer: cldp -offset <alternate-address> ... or 2) Build the ldr file with "Boot format HEX (-f hex)" and add "-kp <alternate-address>" in Additional Options instead of "Start address (-p <alternate-address>)".
TAR-44454	String comparison functions fail on signed values	String comparison functions (strcmp, strncmp and memcp) can return the wrong result if the parameter strings contain non-ASCII characters.

TAR-48374	BF60x def headers have incorrect casts for some registers in the EMAC module	In the "cdef" header files for BF60x processors (which contain C register and bitfield definitions), a number of macros for memory-mapped registers in the EMAC module are incorrect.The macros should use (volatile uint32_t *) instead of (void * volatile *). The list of incorrect macros is given below, followed by correct definitions: pREG_EMAC0_DMA_RXDSC_ADDR pREG_EMAC0_DMA_TXDSC_ADDR pREG_EMAC0_DMA_TXDSC_CUR pREG_EMAC0_DMA_RXDSC_CUR pREG_EMAC0_DMA_TXBUF_CUR pREG_EMAC0_DMA_RXBUF_CUR pREG_EMAC1_DMA_RXDSC_ADDR pREG_EMAC1_DMA_TXDSC_ADDR pREG_EMAC1_DMA_TXDSC_CUR pREG_EMAC1_DMA_RXDSC_CUR pREG_EMAC1_DMA_TXBUF_CUR pREG_EMAC1_DMA_RXBUF_CUR Correct definitions: #define pREG_EMAC0_DMA_RXDSC_ADDR ((volatile uint32_t *)REG_EMAC0_DMA_RXDSC_ADDR) /* EMAC0 RX Descriptor List Address */ #define pREG_EMAC0_DMA_TXDSC_ADDR ((volatile uint32_t *)REG_EMAC0_DMA_TXDSC_ADDR) /* EMAC0 TX Descriptor List Address */ #define pREG_EMAC0_DMA_TXDSC_CUR ((volatile uint32_t *)REG_EMAC0_DMA_TXDSC_CUR) /* EMAC0 TX current descriptor register */ #define pREG_EMAC0_DMA_RXDSC_CUR ((volatile uint32_t *)REG_EMAC0_DMA_RXDSC_CUR) /* EMAC0 RX current descriptor register */ #define pREG_EMAC0_DMA_TXBUF_CUR ((volatile uint32_t *)REG_EMAC0_DMA_TXBUF_CUR) /* EMAC0 TX current buffer pointer register */ #define pREG_EMAC0_DMA_RXBUF_CUR ((volatile uint32_t *)REG_EMAC0_DMA_RXBUF_CUR) /* EMAC0 RX current buffer pointer register */ #define pREG_EMAC1_DMA_RXDSC_ADDR ((volatile uint32_t *)REG_EMAC1_DMA_RXDSC_ADDR) /* EMAC1 RX Descriptor List Address */ #define pREG_EMAC1_DMA_TXDSC_ADDR ((volatile uint32_t

		)REG_EMAC1_DMA_TXDSC_ADDR) / EMAC1 TX Descriptor List Address */ #define pREG_EMAC1_DMA_TXDSC_CUR ((volatile uint32_t *)REG_EMAC1_DMA_TXDSC_CUR) /* EMAC1 TX current descriptor register */ #define pREG_EMAC1_DMA_RXDSC_CUR ((volatile uint32_t *)REG_EMAC1_DMA_RXDSC_CUR) /* EMAC1 RX current descriptor register */ #define pREG_EMAC1_DMA_TXBUF_CUR ((volatile uint32_t *)REG_EMAC1_DMA_TXBUF_CUR) /* EMAC1 TX current buffer pointer register */ #define pREG_EMAC1_DMA_RXBUF_CUR ((volatile uint32_t *)REG_EMAC1_DMA_RXBUF_CUR) /* EMAC1 RX current buffer pointer register */
TAR-48372	___cplb_ctrl is used as part of the instruction/parity workaround but may not be initialized before use	The ___cplb_ctrl variable is checked during the startup code sequence to see if instruction caching is enabled; if it is, instruction parity is disabled to avoid anomaly 16000005 ("Using L1 Instruction Cache with Parity Enabled is Unreliable"). If runtime initialization support is enabled (i.e. the "-mem" switch is used or "Runtime initialization" is selected in the linker project options), ___cplb_ctrl will be checked before it is initialized by the runtime initialization sequence, and will contain a random value. This can mean that parity and instruction caching get enabled together and the anomaly will be hit. To avoid this issue, do one of the following: - do not use runtime initialization. - disable instruction caching.. - disable parity support by defining __parity_ctrl to zero.
TAR-48443	USB Controller Driver header file MISRA errors	The USB controller driver is prebuilt and is included in the driver library file libdrv.dlb. If you are rebuilding this driver library from source and you are building with the -misra-strict compiler option you will encounter some MISRA-C warnings and errors.

TAR-48549	Watchdog services are not included in libssl	The ADSP-BF609 Watch Dog Timer (WDT) service sources were not built into the System Services library Blackfin\lib\bf609_rev_any\libssl.dlb Blackfin\lib\bf609_rev_none\libssl.dlb To use the WDT you will need to include its source file into your project. Blackfin\lib\src\services\source\wd\adi_wd.c The header file to include to include in your project is #include <services\wd\adi_wd.h> Please note that the documentation for the WDT is not available in help system. However, Blackfin\lib\src\services\source\wd\adi_wd.c contains the documenation for the WDT APIs. As of this release you will need to browse Blackfin\lib\src\services\source\wd\adi_wd.c for API documentation.
TAR-48092	The Power Service is unable to put the processor into deep sleep	The Power Service has an adi_pwr_SetPowerMode() API which is used to set the processor dynamic power management operating mode. This API is currently capable of setting the operating mode to Full On, Active, Active PLL Disabled or Sleep. The Deep Sleep and Hibernate modes are not currently supported by this API. The adi_pwr_SetPowerMode() API doesn't return an error code and the mode is not changed if attempting to set the mode to Deep Sleep or Hibernate.
TAR-48392	Disabling PPIRx broadcast in the Video Subsystem driver clobbers VSS connection register contents	If the Video Subsystem API adi_vss_EnablePPIRxBcast is called to disabled the EPPI receive broadcast, the VSS_CONN register contents will be cleared. This is due to a bug in the implementation of the API. By default the EPPI Rx broadcast is disabled. So it is not required to call the adi_vss_EnablePPIRxBcast to disable the broadcast. This will be fixed in the upcoming update of the CCES.