CCES 2.0.0 Release Notes

CCES 2.0.0 Release Notes
June 2015

http://www.analog.com
processor.tools.support@analog.com

Contents

1 Introduction

This document describes the changes for CrossCore Embedded Studio (CCES) 2.0.0. This release adds support for the ADSP-SC58x and ADSP-2158x processor families.

1.1 Supported Operating Systems

This release of CCES is supported on the following operating systems:

- Windows Vista Business, Enterprise, or Ultimate SP2 (32-bit only)
- Windows 7 Professional, Enterprise, or Ultimate (32 and 64-bit)
- Windows 8.1 Pro or Enterprise (32 and 64-bit)

> **Note**
>
> Windows Vista, Windows 7, and Windows 8.1 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.

1.2 System Requirements

Verify that your PC has these minimum requirements for the CCES installation:

- 2 GHz single core processor; 3.3GHz dual core or better recommended
- 1 GB RAM; 4GB or more recommended
- 2 GB available disk space
- One open USB port

> **Notes**
>
> - A faster disk drive decreases the build time, especially for a large amount of source files. 4GB of RAM or more will substantially increase the performance of the IDE.
> - For proper viewing of documentation under Windows Internet Explorer 9 or greater is recommended.

2 New and Noteworthy

2.1 ADSP-SC58x/ADSP-2158x Processor Support Added

The ADSP-SC58x/ADSP-2158x processors are supported in CCES 2.0.0. These processors have the following key features:

- Two SHARC+ cores (except ADSP-SC582 which only has one SHARC+ core); a proprietary toolchain is included to support these cores.
- ARM Cortex-A5 core (ADSP-SC58x processors); a GNU toolchain is included to support this core, including GDB with QEMU (Simulator) and OpenOCD (Emulator).
    - Functionality specific to Analog Devices processors is detailed in the **Analog Devices ARM Toolchain Manual** which can be found from the **Help > Help Contents** menu of the IDE.
    - Open source documentation relevant to the tools is also provided with the online help.
    - The source files for the GNU toolchain and related open source software are available from http://analog.com/opensource.
- High-Performance Floating-Point FFT Accelerator (FFTA), accessible from the ARM and SHARC+ cores. Details on how to use the accelerator in your application can be found in **Using the SHARC+ FFTA Accelerator** within the on-line help.

For more details, see **ADSP-SC58x and ADSP-2158x Processor Support**, later in this document.

2.2 ADSP-BF70x Silicon Revision 1.0 Revised

ADSP-BF70x silicon revision 1.0 has been revised in CCES 2.0.0 to include workarounds for silicon anomalies that have been characterized since CCES 1.2.0. For details, refer to **ADSP-BF70x Processor Support,** later in this document.

2.3 Upgraded Development Environment

The IDE has been upgraded from Eclipse 3.7.2 found in CCES 1.2.0 to Eclipse 4.4.0 in CCES 2.0.0. As the culmination of more than 4 years of development this update brings along a LOT of new functionality and features. For more details, see **IDE Changes**, later in this document.

3 ADSP-SC58x and ADSP-2158x Processor Support

3.1 Developing for ADSP-SC58x/ADSP-2158x Processors

The SHARC+ cores in these processors represent a significant advancement upon the earlier SHARC cores in earlier SHARC processors. There are a number of significant changes of which you should be aware:

- The ARM core is the primary core, in ADSP-SC58x processors, and is the core that boots. The boot process follows the advanced, flexible boot architecture found on Blackfin processors.
- There is a new memory map, supporting the ARM core and the SHARC+ cores. Custom LDFs will need to be replaced.
- The SHARC+ cores introduce a number of performance improvements to the architecture:
    - Hardware support for double-precision floating-point operations.
    - A new cache architecture, with instruction and data caches.
    - A Branch Target Buffer.
- The SHARC+ cores have a longer pipeline:
    - A given sequence of instructions may have different stalls on SHARC+, compared to SHARC. The compiler will schedule instructions to avoid stalls, where possible; this can lead to minor variances in arithmetic results relative to SHARC, when commutative operations are re-ordered.
    - Some DSP library functions included with CrossCore Embedded Studio may require more cycles on SHARC+, compared to SHARC. Please contact Analog Devices for assistance if you encounter difficulties.
- The SHARC+ architecture accepts SHARC assembly language, but is not binary-compatible; SHARC assembly code, C and C++ must be rebuilt for correct execution on SHARC+.

- The SHARC+ core supports word-addressed memory spaces, but is primarily a byte-addressed architecture:
  - Peripherals deal with byte-addressed spaces. SSL/DD device drivers are available to assist in development.
  - RTOS and other middleware products operate in byte-addressed space.
  - The compiler supports byte- and word-addressed spaces, and has support for interoperability between them. Refer to **Using Byte Addressing** in the **Compiler Manual for SHARC Processors**.
  - SHARC assembly code may need changes to operate correctly when dealing with byte addresses.

Supported silicon revisions

CCES 2.0.0 supports silicon revisions 0.0, 0.1 and 1.0 of ADSP-SC58x and ADSP-2158x processors. Please note:

- Support for silicon revision 0.0 is deprecated; it will not be supported in CCES 2.1.0 and later releases.
- Silicon revision 1.0 is currently equivalent silicon revision 0.1. Later releases of CCES 2.1.0 will refine the support for silicon revision 1.0, to disable workarounds for silicon anomalies that no longer apply.

3.2 Drivers and Services for the ADSP-SC58x Processors

The SSL 2.0 driver model provides support for almost all of the the ADSP-SC58x peripherals. For a complete list of supported peripheral please see the ADSP-SC58x API Reference for both the SHARC+ and Cortex-A cores in the online help.

The main features of this driver model are:

- Small footprint and minimal cycle counts
- Easy to use/modify
- Interrupts and DMA have been abstracted
- Switching between Interrupt and DMA Mode supported via API
- Works with or without a RTOS (uC/OS-II, uC/OS-III and non-RTOS are supported)

Three Programming Models are supported

- Non-Blocking Mode

- Blocking Mode
- Callback Mode

For more information on these programming models, see the **Low-Level Driver API Reference** within the **Device Drivers User Guide** in the online help.

The Cross Core Embedded Studio environment supports and simplifies the use of the SSL 2.0 model in the following ways:

- Adding Device Drivers and Services sources through the *System Configuration Manager* (system.svc) Add-In manager.
- Pin multiplexing code generation via a CCES GUI
- SRU (system routing unit) Code Generation via a CCES GUI
- Example Manager
- Code Sketches also available via the Example Manager

Examples for supported peripherals are found in the ADSP-SC58x Board Support Package (BSP), available separately from www.analog.com.

- Power On Self Test (POST)
- Working examples for on and off-chip peripherals

USB Driver PMU usage

The USB enumeration process, for both host and device mode, may require a delay of precisely 1 msec. On the Cortex-A the USB driver uses the PMU (Performance Management Unit) to precisely calculate the delay. The USB driver will reset the PMU and then use its cycle counting registers. The PMU is a shared resource and any other part of the application that requires the PMU must be aware of the USB enumeration process and its dependency on the PMU.

3.3 Creating Projects for ADSP-SC58x

To create a project within the CCES IDE for ADSP-SC58x you should choose the SHARC family, and then select your target processor, e.g. ADSP-SC589. The IDE will then let you choose to create projects for each (or all) of the cores. Whether you are just targeting the ARM core, one of the SHARC+ cores, or all three cores, you can choose which projects you would like to create.

3.4 Using Pre-load Files

External memory needs to be configured appropriately before you can load your application into it. When your application boots, this is done through initcodes; when you load your application into your target using the debugger, this can be done by the IDE automatically for simple processors.

For heterogeneous processors such as the ADSP-SC58x processors, more flexibility is required. This release of CCES 2.0.0 introduces the concept of *pre-load files*, which are equivalent to initcodes, but used during the debugging phase of development.

These pre-load files are only used for the ADSP-SC58x and ADSP-2158x processors currently. You can find the pre-built binary files in **SHARC\ldr**. The projects used to create the pre-load binary files are located in **SHARC\ldr\init_code\SC589_Init**. The main purpose of these pre-load files is to set up clocks and DMC settings so that the debugger is able to load your application to external memory. If you wish to change this setup, you can update the source files, rebuild the pre-load executable, and place the binary in the **SHARC\ldr** folder in place of the existing file. The master core is generally the only core that will need a pre-load so debug configurations will automatically fill the pre-load file in for you as one of the applications that will be loaded. For an ADSP-SC58x processor, the pre-load will be part of the ARM Core 0 project. For an ADSP-2158x processor the pre-load will be part of the first SHARC+ core project.

In most cases, you will want your project executables to have the options set as described in **Core Option Settings for Debug Configurations**, as described under **Debugging Multiple Cores**, below.

See also **Access to Uninitialized External Memory on ADSP-SC58x/ADSP-2158x Processors** in the **Known Problems and Limitations** section for additional information on pre-load files.

Note: Do not use preload files when building bootable LDR files. See also the the **Init Code** section in the **Loader and Utilities Manual** for additional information.

3.5 Supported Emulators

Only the ICE-1000 and ICE-2000 emulators can be used to debug ADSP-SC58x and ADSP-2158x processors.

3.6 Simulation of ADSP-21584 and ADSP-SC589 Processor cores

CCES 2.0.0 has the following options for simulating ADSP-2158x/ADSP-SC58x processor cores:

- SHARC+ cores:
    - Functional simulation.
    - Cycle-accurate simulation.
- ARM Cortex-A5 core:
    - Functional simulation, using the open source QEMU simulator.

The SHARC+ Cycle Accurate Simulator is accurate to within a tolerance of +/- 3%, compared to silicon.

The following cases are known to differ from the silicon:

- cjump/rframe to use of I6/I7 register.
- Some inaccuracies in stalls during read/write to core MMR registers.
- Write to CCNTR to LCE based non-branch instruction.
- Floating point compute or any multiplier operation followed by move of the result to any register outside the relevant execution unit.

For more information on these new simulators please see the **Simulator User's Guide** in the online help.

3.7 Debugging Multiple Cores

There is heterogeneous (ARM Cortex-A5 and 2 SHARC+ cores) debugging support when debugging applications with the CrossCore Debugger and an emulator.

When the processor starts execution, the SHARC+ cores are held in IDLE until enabled by the application running on the ARM core. This means:

- If the application running on Core 0 does not explicitly enable the other cores, the SHARC+ cores will not run their applications.
- When you load your application into the processor using the debugger and run to the start of main() on Core 0, the other cores will still be in IDLE.

The run-time libraries include the adi_core_enable() function to release other cores from IDLE. When you create new projects for ADSP-SC58x or ADSP-2158x processors and elect to generate template code for your projects, the IDE populates the main() function of the booting core with sample code that enables the other cores.

3.7.1 Core Option Settings in Debug Configurations

In most cases, you will want your project executables to have the following options set:

Application	**"Reset" Option**	**"Run after Load" Option**	**"Reset" Option**	**"Run after Load" Option**
	Emulator		**Simulator**	
Pre-load on booting core	Set	Set	N/A	N/A
Application on booting core	Unset	Set	Set	Unset

Application	"Reset" Option	"Run after Load" Option	"Reset" Option	"Run after Load" Option
Application on other cores	Set	Unset	Unset	Unset

These options are accessed in the ‘Modify configuration and continue’ window during session startup. Select the appropriate core/dxe and click ‘Edit…’

With this configuration, when you launch the emulator debug session, the pre-load file will run to completion, the main application for the booting core will be loaded, and it will run to the start of main(). The other cores will be loaded and halted. At this point, you can use Run->MP Resume to run the applications on your processor; when the booting core invokes adi_core_enable(), the respective core will start executing its application.

If "Run after load" is "Set" for the non-booting cores, they will attempt to execute their applications before the booting core executes adi_core_enable(); this will be indicated in the debugger by showing those cores as *Running*, when the booting core halts at main(). In this situation, you can halt those cores, and then use Run->MP Resume, to run all cores. This behavior is due to a limitation in the processor's Reset handling; see **No System Reset** under **Known Problems and Limitations**, for details.

For debugging multiple cores with the simulator, refer to the **Simulator User's Guide**, section Features -> Simulator Sessions -> Handling Multiple DXE Files.

3.8 Debugging Only a SHARC+ Core

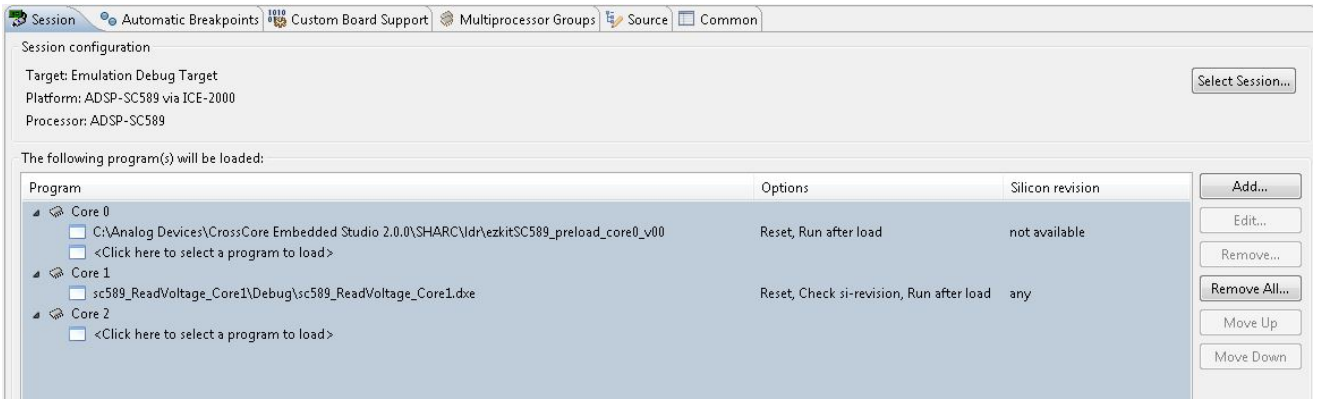
For convenience, you may only want to debug a SHARC+ core with the emulator and not run an ARM application each time to enable your SHARC+ core. This can be achieved by following these steps:

- Create an XML file that has the following contents:

Release Sharc Cores

```
<?xml version="1.0" standalone="yes"?>
<custom-cces-proc-xml
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="\Analog Devices\CrossCore
Embedded Studio 2.0.0\System\ArchDef\ADSP-custom-board.xsd"
    processor-family="sharc"
    file="example_custom_board_support.xml">
  <custom-register-reset-definitions>
    <register name="RCU0_MSG_SET" reset-value="0x00180000" core=
"Common" />
  </custom-register-reset-definitions>
</custom-cces-proc-xml>
```

- In the debug configuration, click on the Custom Board Support tab and check ***Enable customizations.***
- Then point to the XML file that you have just created.
- With this XML configuration in place, you do not need to unset the ***Run after load*** option on the SHARC+ core.



- Now upon connecting to just the SHARC+ core you should be halted at main().
- Since you will only be debugging one core, there is no need to use any of the MP run control options such as MP Resume in this case.

3.9 Creating Bootable LDR Files for ADSP-SC58x / ADSP-2158x

Multi-core bootable LDR files for ARM and SHARC+ cores can be created for the ADSP-SC58x / ADSP-2158x processors using the CCES elfloader tool described in the online help. Refer to **Loader for ADSP-SC58x / ADSP-2158x Multicore Processors** in the **Loader and Utilities Manual** for details.

The ADSP-SC58x / ADSP-2158x processors have on-chip L2 boot ROMs for ARM and SHARC+ cores. With the boot kernel code already resident in the on-chip boot ROMs, there is no need to load a boot kernel from the LDR files as was required when booting applications for prior SHARC processor families. ADSP-SC58x / ADSP-2158x LDR files contain a series of bootable blocks in the format expected by the modern on-chip boot ROM developed by Analog Devices. This LDR file format is different than the ones used for existing SHARC processors.

Refer to **Boot ROM and Booting the Processor** in **ADSP-SC58x SHARC+ Processor Hardware Reference** for details.

4 ADSP-BF70x Processor Support

4.1 ADSP-BF70x silicon revision 1.0

CCES 2.0.0 supports silicon revisions 0.0 and 1.0 of ADSP-BF70x processors, with silicon revision 1.0 being the default.

Silicon revision 1.0 includes a number of significant silicon fixes including:

- Improved booting functionality.
- Support for misaligned data accesses.
- Corrections to LUT, MSI and Pinmuxing.
- Corrections to the Branch Target Buffer.
- An updated Utility ROM

4.1.1 Utility ROM Improvements for ADSP-BF70x revision 1.0

Silicon revision 1.0 of ADSP-BF70x processors includes the following updates to the Utility ROM:

- The `libdsp`/`libcc`/`libc` components have been further optimized for the Blackfin+ core.
- uC/OS-III has been updated to use version 3.04.04 of Micrium's product.
- `LIBDRV` was updated to add new drivers for RSI and HADC peripherals.

A new set of libraries, LDF symbol maps, include files and DXE files for simulation debugging have been included in CCES 2.0.0 to support using the ADSP-BF70x parts silicon revision 1.0 utility ROM.

The IDE-based simulator debug sessions and the chipfactory.exe command-line simulation automatically load the correct ROM DXE files based on the silicon revision that the application being simulated was built for.

> **Warning**
>
> Before running an executable on an ADSP-BF70x processor, ensure that the executable has been built for the correct silicon revision:
>
> - Applications that use the utility ROM and are built for 0.0 silicon will not function correctly on 1.0 silicon.

- Applications that use the utility ROM and are built for 1.0 silicon will not function correctly on 0.0 silicon.
- Applications that do not use the utility ROM, or are built for revisions *any* or *none* are unaffected by this update to the utility ROM.

4.2 Updated Silicon Anomaly Support

A number of silicon anomalies have been addressed in silicon revision 1.0, and are no longer enabled by default. Refer to the Analog Devices Silicon Anomaly List for your respective processor, for details.

4.2.1 Assembler detection for STI-CLI silicon anomaly 19000010

The CCES 2.0.0 Blackfin assembler will issue new anomaly detection warning, ea5526, for adjacent `STI` and `CLI` instructions. The warning is to indicate where a workaround is required for silicon anomaly 19000010 “STI Directly Before CLI Does Not Enable Interrupts”.

Compiler and runtime library workarounds for 19000010 were previously added in CCES 1.1.0.

4.2.2 Watchdog service workaround for silicon anomaly 19000011

The watchdog system-service provided in the CCES 2.0.0 has been updated to incorporate a workaround for silicon anomaly 19000011 " The WRDO Bit in WDOG_CTL is Erroneously Cleared Under Certain Conditions ".

4.2.3 Branch Predictor cleared during application startup for silicon anomaly 19000047

The branch predictor can operate incorrectly if the predictor learns from a control-flow instruction within an initcode, then the subsequent application happens to have a different control-flow instruction mapped to the same location. To work around this situation, the startup code in CCES 2.0.0 flushes the branch predictor's learned information at the beginning of the application.

4.2.4 Branch Predictor disabled during ISRs for silicon anomaly 19000054

Silicon anomaly 19000054 describes a situation where the branch predictor's operation can lead to a self-nested interrupt returning to User Mode instead of to the preceding interrupt level. To avoid this situation, the interrupt dispatchers in CCES 2.0.0 disable the branch predictor during interrupt service routines.

5 IDE Changes

The IDE has been upgraded from Eclipse 3.7.2 found in CCES 1.2.0 to Eclipse 4.4.0 in CCES 2.0.0. As the culmination of more than 4 years of development this update brings along a LOT of new functionality and features. Please refer to the individual release notes for the major Eclipse components for specifics as to these new features:

5.1 Platform Changes Since 3.7

- New and Noteworthy in Eclipse 4.0 (Helios)
- New and Noteworthy in Eclipse 4.1 (Indigo)
- New and Noteworthy in Eclipse 4.2 (Juno)
- New and Noteworthy in Eclipse 4.3 (Kepler)
- New and Noteworthy in Eclipse 4.4 (Luna)

5.2 C/C++ Development Tools (CDT) Changes Since 8.0

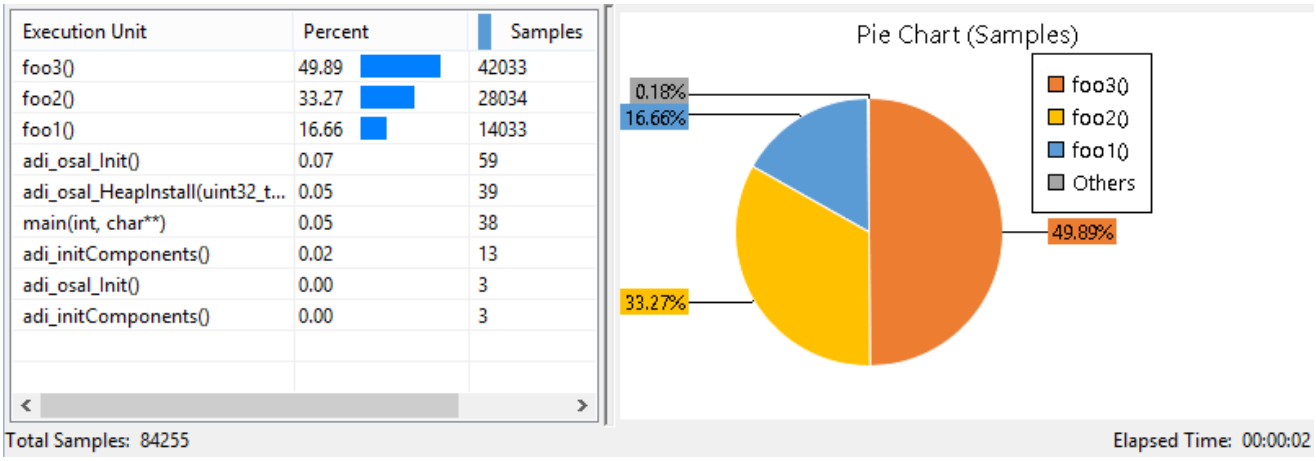
- New and Noteworthy in CDT 8.1
- New and Noteworthy in CDT 8.2
- New and Noteworthy in CDT 8.3
- New and Noteworthy in CDT 8.4

5.3 New IDE Features Added Since CCES 1.2.0

- Native GIT support via EGit
- Support for remote development via the Remote System Explorer
- Pipeline Viewer

Cycle	Decode	Address	DataFetch1	DataFetch2	Execute1	Execute2	Writeback
6545	M (R7:5) =	M (R7:5) =	M (R7:5) =	P5 = [SP ++	L2 = 0 (X)	L1 = 0 (X)	L0 = 0 (X)
6546	RTS ;	M (R7:5) =	M (R7:5) =	M (R7:5) =	P5 = [SP ++	L2 = 0 (X)	L1 = 0 (X)
6547	K	RTS ;	M (R7:5) =	M (R7:5) =	M (R7:5) =	P5 = [SP ++	L2 = 0 (X)
6548	K	A	RTS ;	M (R7:5) =	M (R7:5) =	M (R7:5) =	P5 = [SP ++
6549	K	K	A	RTS ;	M (R7:5) =	M (R7:5) =	M (R7:5) =
6550	K	K	K	A	RTS ;	M (R7:5) =	M (R7:5) =
6551	R0 = 0 ;	K	K	K	A	RTS ;	M (R7:5) =
6552	UNLINK ;	R0 = 0 ;	K	K	K	A	RTS ;
6553	M UNLINK ;	M UNLINK ;	R0 = 0 ;	K	K	K	A
6554	RTS ;	M UNLINK ;	M UNLINK ;	R0 = 0 ;	K	K	K
6555	S RTS ;	B	M UNLINK ;	M UNLINK ;	R0 = 0 ;	K	K
6556	S RTS ;	B	B	M UNLINK ;	M UNLINK ;	R0 = 0 ;	K

- Linear and Statistical Profiler

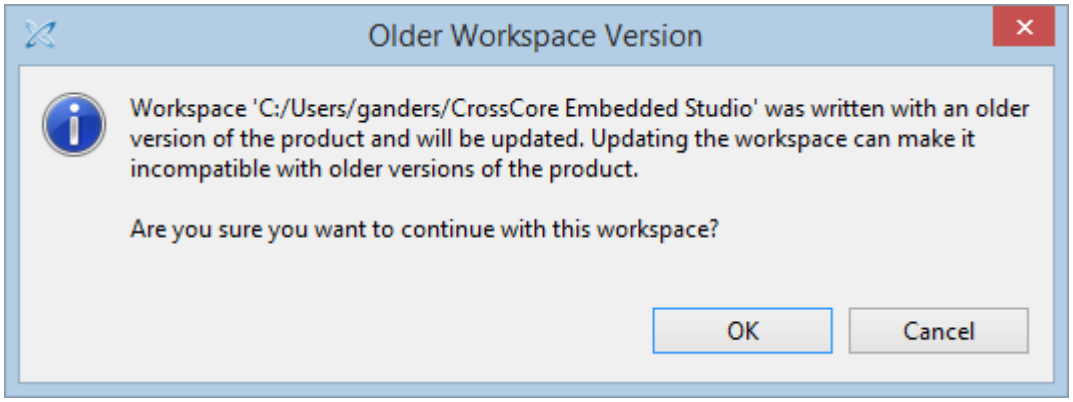


5.4 Eclipse Features Removed Since CCES 1.2.0

- Native CVS support via Platform-CVS

5.5 Workspace Compatibility with Previous Versions of CCES

Due to the move from Eclipse 3.7 to 4.4 it may not be possible to use the same workspace with CCES 2.0 and earlier versions of CCES. When you point CCES 2.0 to a workspace created by CCES 1.x you will see the following message. We recommend keeping separate workspaces for multiple versions of CCES.



5.6 IDE Workspace Default Changed for CCES 2.0.0

- The default workspace path is more Linux friendly `${home}/cces/2.0.0`. For example, on Windows the default workspace path is `C:\Users\${username}\cces\2.0.0` and on Linux the default workspace path is `/home/${username}/cces/2.0.0`.

6 Toolchain Updates

6.1 SHARC Compiler

The CCES 2.0.0 SHARC compiler has been updated to provide improved language standards compliance. These changes bring the SHARC compiler in line with the Blackfin compiler, which was updated for CCES 1.1.0.

6.1.1 Updated Language Standards Support

The compilers accept many features of the ANSI/ISO 14882:2011 Standard (C++11), when the -c++11 switch is used. Note that the underlying run-time library conforms to ANSI/ISO 14882:2003. When the -c++ switch is used, the compilers conform to the ANSI/ISO 14882:2003 Standard.

The -g++ switch may be used with the compilers. It directs the compilers to support many of the GNU G++ extensions to the C++ language. The -g++ switch may be used in conjunction with either the -c++ or -c++11 switches.

6.1.2 Universal-character-names in narrow string literals

The way the SHARC compiler in CCES 2.0.0 (and the Blackfin compiler as of CCES 1.1.0) handles universal-character-names in narrow string literals has changed. Previously the Unicode value of a universal-character-name appearing in a narrow string literal was truncated (with a warning) to the least-significant byte and represented as a single character in the value. Now the value is the UTF-8 variable width encoding representation of the Unicode character. For example, the string "\u20AC" (the Euro symbol) was previously equivalent to "\0xAC"; it is now equivalent to "\0xE2\0x82\0xAC".

6.1.3 Compiler error cc0137

The SHARC compiler in CCES 2.0.0 (and the Blackfin compiler in CCES 1.1.0) raises error cc0137 for uses of decrement or increment operators for the result of a cast in a single expression. Previous versions of the Blackfin compiler and the CCES 1.1.0 SHARC compiler issue a warning for this problem. For example the following source will cause new error cc0137.

cc0137 example

```
void func(void *buffer, unsigned short us, int len) {
  for (int i=0; i<len; i++)
    *((unsigned short *)buffer)++ = us;
```

```
}
```

Correct the error by performing the cast in a separate expression from the decrement or increment. For the example above the correction is shown below.

example cc0137 correction

```
void func(void *buffer, unsigned short us, int len) {
  unsigned short *usPtr = (unsigned short *)buffer;
  for (int i=0; i<len; i++) {
    *usPtr++ = us;
  }
}
```

6.1.4 Inlining support in C99

The support for inline functions that do not specify a storage class has changed in the CCES 2.0.0 SHARC compiler (and CCES 1.1.0 Blackfin compiler). Previously such inline functions were implicitly treated as if they had static storage class. Such functions are now treated in a more standard conforming way. The compiler support for inlining in C99 is described below. (Be aware that C++ also has "extern inline"; this section specifically refers to behavior in C99 mode.)

inline (with static storage class)

```
static inline void func() { } int main() { func(); }
```

- A static instance of "func" is generated when a call is not inlined or the address of the function is taken
  - the call and the address refer to the static instance.
- No instance of "func" is generated if all calls are inlined and the address of the function is not taken.
- The behavior is the same as C89 mode.
- The behavior is the same as gcc with C99 enabled.

inline (with no storage class specifier)

```
inline void func() { } int main() { func(); }
```

- If a call is not inlined, then function "func" is a static instance to which the call refers.
- If the address of func is taken, the address and all calls to func within the translation unit instead refer to an externally defined function.
- If all calls are inlined, then no static instance is generated.

- The behavior is the same as C89 mode, apart from in c89 taking the address refers to the local instance.
- The behavior differs from gcc with C99 enabled. In C99 mode there is no local instance of the function generated. All references are to a function defined elsewhere.

inline (with extern storage class specifier)

```
extern inline void func() { } int main() { func(); }
```

- An external definition of the function is always created, this could lead to multiply defined symbols if it is declared in a header file.
- If all calls are not inlined, then the call refers to the external definition in this module.
- If the address is taken, then the call refers to the external definition in this module.
- This behavior differs from C89 mode. In this mode there is no definition of the function created. Taking the address or not inlining a call will lead to an external reference being created.
- The behavior is compatible with gcc with C99 enabled.

Treatment of double-precision floating-point denormal values

The hardware support for double-precision floating-point arithmetic on the ADSP-215xx and ADSP-SC58x family of processors flushes denormal input values to zero before each operation. In contrast, the run-time library support used with earlier SHARCs without double-precision hardware support does not flush denormals to zero. Therefore when denormals are present, the output given by programs built for ADSP-215xx and ADSP-SC58x processors, and those built for earlier SHARC parts, may not match exactly.

6.1.5 cc21k SHARC compiler driver switch changes

`-aligned-stack` switch is no longer supported and will be ignored if used. The stack uses of the compiler will by default attempt to retain double-word alignment assuming the stack is aligned at the start of a function unless the `-no-aligned-stack` switch is used.

6.2 New Compiler Warnings

6.2.1 cc1486: invalid section qualifier

The SHARC and Blackfin compilers will now issue warning `cc1486` for uses of `#pragma sections` and `#pragma default_section` with invalid or unknown section qualifiers. The invalid qualifier will be ignored as before and compilation will continue. Fix this problem by selecting a correct section qualifier or by deleting the invalid one if it is not required.

SHARC valid section qualifiers are `NO_INIT`, `RUNTIME_INIT`, `ZERO_INIT`, `DOUBLE32`, `DOUBLE64`, `DOUBLEANY`, `SW` (for VISA parts code only), `NW` (for VISA parts code only), `DMAONLY`, `PM` and `DM` (for data only).

Blackfin valid section qualifiers are `NO_INIT`, `RUNTIME_INIT`, `ZERO_INIT`, `DOUBLE32`, `DOUBLE64` and `DOUBLEANY`.

6.3 Run-Time Library Changes

6.3.1 Library performance optimizations

A number of functions in the SHARC library have received performance improvements, as they are common to SHARC and SHARC+, and have been optimized to avoid SHARC+ stalls. In some cases, these performance improvements may result in minor changes in computed values.

Several SHARC functions have increased in code size, to overcome stalls.

6.3.2 exit

The behaviour of exit() has been rationalized between architectures to provide a consistent interface. This also removes any differing behaviour with threaded/non-threaded applications.

The new behaviour is now as follows:

_Exit(int val) saves val to _exit_value and terminates the application (jumping to __lib_prog_term), without running any atexit handlers.

exit(int val) invokes any atexit() handlers before calling _Exit(val) to save val to _exit_value and terminate the application.

6.3.3 rand (SHARC only)

The rand implementation on SHARC processors has been replaced with an implementation which uses a 64-bit seed to give a period of > 2^32.
This will result in a different series of values returned from rand() compared to previous releases, but reduces the possibility of repeating patterns.

6.3.4 Data cache invalidation (Blackfin only)

The library functions dcache_invalidate() and cache_invalidate() now always invalidate the data caches for all Blackfin parts by modifying configuration bits. This approach is significantly faster than explicitly flushing portions of the cache as was done on some Blackfin parts in previous

releases. This performance improvement means that the caches can no longer be invalidated individually, with the exception of BF70x parts where cache B can be invalidated without also invalidating bank A. Any other options will invalidate both cache banks, the equivalent of calling dcache_invalidate_both(). The behavior of instruction cache invalidation remains unchanged.

6.3.5 The %a conversion specifier

In this release, the support for the %a conversion specifier conforms to the description in the C99 standard (ISO/IEC 9899). Compared to previous releases, you may notice the following differences in behavior when printing values using this specifier:

– If an application is built with the switch -double-size-64, then the %a conversion specifier will output arguments as IEEE double precision values, regardless of whether the length modifier L has been specified. Previously, if neither a precision nor the L length modifier was specified, the %a conversion code would format arguments as a float and would display no more than 6 digits after the decimal point.

For example:

```
printf ("%a\n",0.1L);
```

According to the C99 standard, this statement would display the text *0x1.999999999999ap-4*, but previous releases would display *0x1.999999p-4* instead.

– A formatted value would not be rounded if the precision was specified as zero.

For example:

```
printf ("%.0a\n",1.5);
```

According to the C99 standard, this statement would display the text *0x1p+1*, but previous releases would display *0x1p+0*.

> **Note**
>
> There shall be no different behavior if the %a conversion code is used to read a string that represents a floating-point value and to convert it to binary.

6.3.6 INTR_TAPC0_KEYFAIL renamed

The ADSP-BF70x parts def-header include files have been changed to rename macro `INTR_TAPC0_KEYFAIL` to `INTR_TAPC_KEYFAIL`. A definition of `INTR_TAPC0_KEYFAIL` will only be available if macro `_INCLUDE_LEGACY_SYSREG_NAMES` is defined before including the def-header.

6.3.7 Core-management functions for multi-core processors

The run-time libraries for Blackfin and SHARC processors include functions to identify the current core, and to enable other cores. These functions are:

- adi_core_enable()
- adi_core_id()

For details, refer to the library manual for the respective processor:

- **C/C++ Library Manual for SHARC Processors**
- **C/C++ Compiler and Library Manual for Blackfin Processors**

6.4 LDF and Linking Related Changes

6.4.1 Implicit support for External memory sections in SHARC+ cores

In the LDFs for previous SHARC parts, implicit support was provided for some external memory input sections (seg_ext_data, seg_sdram, seg_ext_code) when SDRAM was not enabled.
In SHARC+ LDF's these input memory sections are only available when SDRAM is enabled via the "Use external memory" checkbox in the LDF configuration, or the USE_SDRAM macro in the default non-generated LDFs.

6.4.2 32-bit SHARC+ PM data changes

The LDFs for SHARC+ cores make use of a linker feature that allows sections of different types such as DM or PM to be mapped into a common byte-addressed memory segment, which means that memory does not need to be manually partitioned into segments for the various types. However, that feature has a limitation: it assumes that each section type corresponds to a particular memory width, for example 32 bits for DM and 48 bits for PM. This contrasts with mapping a section to a segment of the same type, where the section will assume the segment’s width. For example, a PM segment can have width 32 or 48.

As a consequence, 'seg_pmda' sections with type PM, which were mapped to PM segments with width 32 on previous SHARC parts, would wrongly end up with width 48 when mapped to a byte-addressed segment. Therefore, for SHARC+ cores such as those in the ADSP-SC589, the compiler and runtime libraries declare word-addressed 'seg_pmda' sections as type DM rather than PM to ensure that they end up with width 32.

6.4.3 SHARC+ Cache support (changes required for custom LDFs)

The CCES Run-Time Library now provides support for enabling instruction and data (DM/PM) caches on SHARC+ cores, using LDF symbols for configuration. Corresponding support is provided in the default generated and non-generated LDFs, by default each cache is enabled using 16KB of cache space

Custom LDF's will minimally need to define the following LDF symbols in order to link successfully. This will leave the caches disabled, so no memory layout alterations are required.

```
___ldf_icachesize = 0xffffffff;
___ldf_dmcachesize = 0xffffffff;
___ldf_pmcachesize = 0xffffffff;
```

6.4.4 USE_L1_ONLY macro no longer has an effect

It is no longer possible to disable the use of any memory beyond L1 by defining the LDF macro USE_L1_ONLY.

6.4.5 MEM_ASYNC sections renamed for ADSP-BF60x generated LDFs

The Startup/LDF addin for the ADSP-BF60x parts family will rename the four ASYNC memory sections when the LDF is regenerated using CCES 2.0.0. The changes are to have the numbers used match the published memory-map and the non-generated LDFs, they were in reverse order previously. Any user custom additions in generated LDFs using these sections (MEM_ASYNC0, MEM_ASYNC1, MEM_ASYNC2 and MEM_ASYNC3) may require corresponding changes to maintain the same memory usage seen prior to the change.

6.5 Blackfin Assembler and Branch Instruction Encoding

Some inconsistencies in the Blackfin assembler and linker support for JUMP and CALL instructions have been resolved. This may mean that some existing assembly code will receive different encodings.

- When building for ADSP-BF7xx processors, `JUMP` and `CALL` instructions (without the .X suffix) that target labels will be expanded to 64-bit encodings with a 32-bit target if necessary, instead of triggering an out-of-range linker error.
- The `-jcs2l` linker switch is no longer needed to enable expansion to 64-bit branches. (It is still needed to enable expansion to indirect branches via the `P1` register on ADSP-BF5xx and ADSP-BF6xx processors.)

- A `JUMP` (without suffix) to a numeric offset in the -0x1000000..0xFFFFFE `JUMP.L` range but outside the -0x1000..0xFFE JUMP.S range will be encoded as a 32-bit instruction, whereas with prior releases of CCES it would have been encoded as a 64-bit instruction.
- A `CALL.L` instruction has been added to allow users to explicitly select the 32-bit encoding with 24-bit target (cf. `JUMP.L`). An out-of-range error results if the target is outside the -0 x1000000..0xFFFFFE range.
- In disassembly output, relative branch instructions are now always printed with the appropriate suffix: .S for 16-bit encodings, .L for 32-bit encoding, and .XL for 64-bit encodings.

7 Known Problems and Limitations

7.1 No System Reset

Currently only a core reset is supported on the ADSP-SC58x/ADSP-2158x (Rev 0.1 silicon or older) and the ADSP-BF70x (Rev 0.0 silicon) processors, which has shown limitations when peripherals are running at the time of a core reset. There may be cases where you run an example and then reload to run the example a second time and you get exceptions. This could be due to the peripheral interrupt being serviced at the wrong time causing an exception. In order to fix this, identify the peripheral that is causing the issue and reset that peripheral in a pre-load file. See the ***Using Pre-load Files*** section of this document for more information on how to use them. The other option would be to use the hard reset on the target board in between running examples. Using Engineer Zone or sending a private support request is also a good option so that we are aware of the issue and can add it to the default pre-load files for the upcoming release.

The nature of the core-only resets on the ADSP-SC58x and ADSP-2158x processors means that a core has to be running in order for the reset operation to take effect, when that reset is triggered by another core - if a core is halted by the emulator during the reset operation, the reset operation has no effect on the halted core. For this reason, the emulator sets all cores running after an application is downloaded to the processor from the IDE.

Beginning with Rev 1.0 silicon, the ADSP-BF70x supports system reset so the issues associated with a core reset on this processor will no longer be present.

7.2 ICE-2000 JTAG Frequencies limited on ADSP-BF70x Rev 1.0 silicon

On silicon revision 1.0 of the ADSP-BF70x processor, the ICE-2000 will only work up to 23 MHz. The 46 MHz JTAG frequency will not work properly.

7.3 No SWD debug support for ADSP-SC58x/ADSP-2158x

Currently the Target Configurator allows a platform to be created to allow SWD debugging(instead of JTAG) for ADSP-SC58x/ADSP-2158x processors but SWD debugging will not work. JTAG is the only supported debug method for these processors.

7.4 GDB with OpenOCD with ICE-1000 or ADSP-SC584

When using GDB with OpenOCD and the ICE-1000 emulator, ensure the ICE-1000 is selected in the ‘Target’ tab when setting up a Debug Configuration. The configuration defaults to ICE-2000.

When using GDB with OpenOCD with the ADSP-SC584 EZ-Board, ensure ADSP-SC584 is selected in the ‘Target’ tab when setting up a Debug Configuration. The configuration defaults to ADSP-SC589 EZ-Board.

7.5 Access to Uninitialized External Memory on ADSP-SC58x/ADSP-2158x Processors

Access to a disabled memory - including speculative accesses - can result in a hang of the system and the only fix is to do a hard reset on the target board to recover from the hang.The following memories are affected:

Memory	Affected on 0.0, 0.1 silicon	Affected on 1.0 silicon	SMPU Available
DMC0	Yes	Yes	Yes
DMC1	Yes	Yes	Yes
SMC	Yes	Yes	Yes
PCIe	Yes	Yes	Yes
SHARC+ L1 via multi-processor space	Yes	No	No
Flash	Yes	No	No

There are two workarounds for this problem:

Enable the memory: If the memory is enabled, the memory device will respond to the access and the hang will not occur.

Configure the Secure Memory Protection Unit: If you do not intend to use the memory, you can configure the memory's corresponding SMPU (where available), to disable speculative reads, and to disallow all access to the memory space. Accesses will be rejected by the SMPU and not passed onto the disabled memory device, so no hang will occur.

Apply this configuration - whether enabling the memory, or configuring the SMPU - in the preload and/or initcode of your project. This will ensure that the workaround is applied before your application starts running, whether you are loading your application through the IDE, or booting it from a peripheral. See ***Using Pre-Load Files*** in this document for further details.

7.6 Changing silicon revision on a ADSP-BF707 that contains SSL drivers added as source from the UI may not work if the driver is in ROM

If an application for ADSP-BF707 contains the sources to the drivers that are in ROM (SPI, SPORT, UART, TWI, RSI, HADC) which were added via the SSL plugin, changing silicon revision in the project may cause errors because the UI might not define the appropriate macros to indicate that the driver used is not in the ROM. It is recommended that applications remove the driver sources, change silicon revision and then add the driver sources again.

7.7 OTP Programmer not visible through Examples Browser

The OTP Programmer example which is needed when using the Command Line Device Programmer (CLDP.exe) with the "-device otp" switch is not visible through the Examples Browser. A supplied dxe that can be used with CLDP.exe can be found in the CrossCore Embedded Studio installation directory under SHARC/Examples/Device_Programmer. To view or rebuild this project, "Import an existing CCES project" from the Welcome screen, then browse for the directory.

7.8 Ensure the volatile type qualifier is used where necessary

Compiler updates and improvements included in any CCES release can expose latent bugs in application software for the first time. For instance the MemCopyArrayMode example found in many Blackfin BSP products failed for the first time when built with CCES 1.1.0 in release configuration. The problem was tracked down to a pre-existing bug in the MemCopyArrayMode example. The example had omitted a necessary use of a volatile type qualifier on a variable declaration, necessary because the variable is updated in a callback ISR function.

Please review the CCES help topic "The volatile Type Qualifier" and ensure that your applications make use of volatile where necessary.

7.9 Passing data between SHARC+ and ARM cores

CrossCore Embedded Studio 2.0.0 provides MCAPI drivers to assist applications in passing data between the cores of the processor. When passing data between the cores, be aware that that some data types may vary in size between the ARM and SHARC+ cores. In particular, be aware of:

- *C enumerations.* For the SHARC+ processor, an enumeration is declared to be of type `int`. For bare metal ARM applications, the GNU GCC Compiler defaults to using the `-short-enums` switch. This uses the smallest possible data type to store the enumeration.
- *Bitfields*. The layout of the fields of a bitfield within enclosing integer datatypes is implementation-defined, and application developers may not rely on different toolchains using the same arrangement for a given bitfield declaration.

It is recommended that datatypes of specific sizes, such as those defined by `stdint.h`, are used to express data that must be accessed by heterogeneous cores.

7.10 No concurrent USB Host and Device stack capability for ADSP-SC58x Processors

The USB stack supports the connection to any of the USB0 (USB OTG) or USB1 (USB HS) ports provided on the ADSP-SC589 EZ-Board, but connection to both the ports at the same time is not supported. However, you can switch between USB0(USB OTG) and USB1(USB HS) ports while your application is running.

7.11 "Target not available"

When you launch a debug session on a multi-core processor, such as ADSP-SC58x, ADSP-2158x, ADSP-BF561 or ADSP-BF60x processors, the debugger connects to each core separately. At any given time, one of the cores has the focus, and this defaults to the booting core of the processor. On occasion, the debugger may grant the focus to another core as the connections are created, and this may lead to a "Target not available" message, and an inability to display assembly code in the disassembly window. Simply select the booting core in the Debug pane to grant the focus to the correct core.

7.12 Relaunching Debug Sessions with GDB with OpenOCD or QEMU

When using GDB with OpenOCD or QEMU, it is important to 'Terminate and Remove' the session before launching another session. Failure to do so could result in unstable behavior.

7.13 Stdio and Multithreaded Libraries in the GNU ARM Toolchain

The GNU ARM toolchain includes multithreaded libraries for use when building RTOS-based applications. The following issues are outstanding, relating to stdio:

- Semaphores for the standard streams `stdin`, `stdout` and `stderr` are allocated on a per-thread basis, rather than globally. This means that applications will require an additional three semaphores per thread.
- If the *first* use of one of these three standard streams is as a parameter to one of the printf() family of functions, a NULL-pointer-dereference error will occur.

7.14 QEMU Memory Limitations

In CCES 2.0.0, QEMU models the ADSP-SC58x memory only partially. It appears that this has a greater impact on ADSP-SC582 and ADSP-SC584 than on ADSP-SC587 and ADSP-SC589 as ARM programs built for these two processors are more likely to use memory space not supported in QEMU. As a consequence, a program may fail to load in QEMU or may fail to execute when code or data are being accessed from unsupported memory regions. This is addressed in future releases.

7.15 Other Known Issues

For the latest anomalies, please consult our Software and Tools Anomalies Search page.