Release Notes for CrossCore Embedded Studio 2.2.0

Release Notes for CrossCore Embedded Studio 2.2.0
February 2016

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

Contents

1 Introduction

This document describes the changes for CrossCore Embedded Studio (CCES) 2.2.0. You can find the release notes for older releases in the `docs` sub-directory of your CCES installation.

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**
>
> 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

> **Note**
>
> - A faster disk drive or SSD 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 and ADSP-2158x parts revision 1.0 support

> The default silicon revision used as the target of new projects and by command-line tools when the revision isn't specified has changed to be revision 1.0 for the ADSP-SC58x and ADSP-2158x parts.

Full support for revision 1.0 of the ADSP-SC58x and ADSP-2158x parts has been added in CCES 2.2.0.

- A new set of libraries for the SHARC+ cores has been added in folder `SHARC/lib/sc589_rev_1.0`
- Tools silicon anomaly workarounds have been configured correctly for revision 1.0 builds
- SHARC+ caches are no longer disabled, when building for revision 1.0

2.2 Revised Dynamically-Loadable Modules (DLM) support

> Support for dynamically-loadable modules (DLMs) using `libdyn.h` has been deprecated, and will be removed in a future release.

Dynamically-loadable modules (DLMs for short) are loader streams containing executable code and /or data that can be loaded and used by a running application. DLMs are analogous to overlays, but where an overlay is created as part of the linking process of the main application, DLMs are created separately from the main application, possibly at a much later date. CCES 2.2.0 provides a new `adi_libldr.h` include file and `libldr.dlb` libraries that support DLMs, enabling you to load code and data from a loader stream at runtime.

Supported parts:

SHARC Processors	
ADSP-2158x	ADSP-21584, ADSP-21587

SHARC Processors	
ADSP-SC58x	ADSP-SC582, ADSP-SC584, ADSP-SC587 ADSP-SC589
Blackfin Processors	
ADSP-BF50x	ADSP-BF504, ADSP-BF504F, ADSP-BF506F
ADSP-BF51x	ADSP-BF512, ADSP-BF514, ADSP-BF516, ADSP-BF518
ADSP-BF52x	ADSP-BF522, ADSP-BF524, ADSP-BF526, ADSP-BF523, ADSP-BF525, ADSP-BF527
ADSP-BF54x	ADSP-BF542, ADSP-BF544, ADSP-BF547 ADSP-BF548, ADSP-BF549 ADSP-BF542M ADSP-BF544M, ADSP-BF547M, ADSP-BF548M, ADSP-BF549M
ADSP-BF59x	ADSP-BF592-A
ADSP-BF60x	ADSP-BF606, ADSP-BF607, ADSP-BF608, ADSP-BF609
ADSP-BF70x	ADSP-BF700, ADSP-BF701, ADSP-BF702, ADSP-BF703, ADSP-BF704, ADSP-BF705, ADSP-BF706, ADSP-BF707

Please see *Dynamically-Loadable Modules* in the *System Run-Time Documentation* for information on how to use the APIs in `adi_libldr.h` to load a DLM from a loader stream.

2.3 New and modified FFTA accelerator support

2.3.1 Existing code needs to be re-built

Due to changes in the FFTA accelerator support, any existing code must be rebuilt from source when being relinked against the new SSL/DD or `libfftacc` libraries.

2.3.2 New method for programming DMA for FFTA accelerator

In order to improve the flexibility and performance of the FFTA support, a supplementary model is provided for programming the DMA transfers.

For small FFTs, the model allows you to define and use your own DMA descriptor chain, giving you control over synchronization points and buffer usage. It also allows you to set up a repeating DMA sequence in advance without the necessity to initiate each DMA transfer explicitly. This model, while more complex than using the `adi_fft_*` functions, can thus give access to improved performance.

For large FFTs, functions are provided which create a repeating DMA descriptor chain to transfer the input and output buffers to the FFTA. It is then possible to repeatedly perform large FFTs in non-blocking fashion, initiating FFT transformations and later returning to query whether they have completed.

Full details are provided in the online help, in the section "Using the SHARC+ FFTA accelerator".

2.3.3 Twiddle tables for large FFTs

Twiddle tables passed to `accel_*` functions for use in the large FFT regime must now be in the format required by the accelerator. You can continue to use the twiddle tables provided in the `libfftacc` library, which are named `accel_twiddles_N` for N = {4096, 8192, 16384, 32768, 65536 }. However twiddles generated by the `twidfft` function are no longer supported, and their use in `accel_*` functions will cause an error to be raised. Instead, twiddles should be generated for the FFTA accelerator using the `accel_twidfft` function. This function generates a twiddle table of size `N` (the number of points of the FFT), rather than `N/2` table generated by the `twidfft` function. The buffer passed in to the function must be aligned to at least a 32-byte boundary.

2.3.4 Small Single-Shot FFTA accelerator functions removed

When the FFTA accelerator is used for FFTs with a small number of points using the Single-Shot functions, the performance achieved can be worse than performing the FFT on the core itself. As a result, functions that give poor performance have been removed from the `libfftacc` library. Specifically, the following functions have been removed:

```
accel_cfftN, for 64 <= N <= 512
accel_ifftN, for 64 <= N <= 512
accel_rfftN, for 64 <= N <= 512
```

Moreover, it is not recommended to use any of the Single-Shot FFTA accelerator functions for FFTs of 512 points or fewer. For such FFTs, best performance is achieved using the Continuous Pipelined mode of operation.

2.3.5 New magnitude-squared FFTs

New functions are now provided to perform small FFTs with windowing producing magnitude-squared output. Functions are also provided to perform large FFTs producing magnitude-squared output, both with and without windowing. Further details are available in the online help.

2.4 Updates to Preloads and Init Codes

2.4.1 Preloads and init codes updated for ADSP-SC58x for silicon anomaly 20000018

The preload and init code executables provided in CCES for the ADSP-SC584 and ADSP-SC589 EZ-Kit support have changes to disable speculative accesses to unused DMC and SMC memory using the SMPU as required to work around silicon anomaly `20000018`. The source for the preloads and init code also has a workaround for SPI flash memory, but it is disabled in the executables provided in CCES. If you do not use SPI flash memory at all, either in your application or in the boot process, you can rebuild the preload or init code with the UNUSED_SPI_FLASH #define set to 1. Alternatively, if SPI flash is used only in certain parts of your application, use the workaround during execution of parts of your program where SPI memory is not set up and therefore speculative accesses must not be performed. The workaround consists of executing the following three lines of code (be sure to include the `<sys/platform.h>` header to provide the necessary macros):

```
*pREG_SPI2_TXCTL = (BITM_SPI_TXCTL_TEN | BITM_SPI_TXCTL_TTI);
*pREG_SPI2_RXCTL = (BITM_SPI_RXCTL_REN | BITM_SPI_RXCTL_RTI);
*pREG_SPI2_CTL = (BITM_SPI_CTL_EN|BITM_SPI_CTL_MSTR|
BITM_SPI_CTL_MMSE);
```

This workaround to avoid speculative accesses to SPI flash is not required on silicon revisions 1.0 and later.

2.4.2 Initcode updates for BF70x processors

- BF706 initcodes are now provided (CCES-12193).

```
<cces2.2.0 install>/Blackfin/ldr/BF706_init_icache_disable_v10.dxe
<cces2.2.0 install>/Blackfin/ldr/BF706_init_only_icache_disable_v10.dx

<cces2.2.0 install>/Blackfin/ldr/BF706_init_v10.dxe
```

- BF707 revision 1.0 initcodes are now provided (CCES-12192).

```
<cces2.2.0 install>/Blackfin/ldr/BF707_init_icache_disable_v10.dxe
<cces2.2.0 install>/Blackfin/ldr/BF707_init_only_icache_disable_v10.dx

<cces2.2.0 install>/Blackfin/ldr/BF707_init_v10.dxe
```

- Workaround for anomaly 19000051 " DLL in the DMC PHY May Not Lock to the New DCLK Frequency" included (CCES-11042).
- Changes to `REG_DMC0_PHY_CTL3` to match updated first time DMC initialization steps (CCES-13806).
- Initcodes are now built from a single BF70x_init project rather than duplicate projects for each BF706 and BF707 and silicon revision. It is therefore necessary to modify the processor settings as required whenever building initcodes using this project.

2.4.3 Initcode updates for BF60x processors

- Workaround for anomaly `16000046` "`DLL in the DMC PHY May Not Lock to the New DCLK Frequency`" included (CCES-11042).
- Initcodes are now built from a single BF60x_init project rather than duplicate projects for each BF609 silicon revision. It is therefore necessary to modify the processor settings as required whenever building initcodes using this project.

2.5 cdef types for read-only MMRs are now const

The cdef include files for the ADSP-SC58x and ADSP-2158x parts now use const qualified types in the definitions of p-macros for read-only MMRs. Types for any variables assigned using these macros will need to be modified to also include the const qualifier to avoid compilation errors.

2.6 Assembly editor

Source code folding is now supported in the assembly editor. Supported source code blocks include:

- Multi-line comments (`/* */`)
- Conditional preprocessor directives (`#if, #ifdef, #ifndef, #elif, #else`)
- Assembly labels (`foo:, bar:`)

```
/*
 * dotprod.asm
 * created on: Feb 22, 2016
 */

.SECTION/CODE seg_pmco;

#if DOTPROD

_dotprod:

  DM(I7, M7) = I8;  // M7 == -1
  I4 = R4;
  I8 = R8;

  R0 = DM(1, I6);

  LCNTR = R12, DO (PC, loop_end) UNTIL LCE;
    R1 = DM(I4, M6), R2 = PM(I8, M14);
    F1 = F1 * F2;
loop_end:
    F0 = F0 + F1;

  I8 = DM(1, I7);

  I12 = DM(M7, I6);      // M7 == -1
  JUMP (M14, I12) (DB);  // M14 == 1
    NOP;
    RFRAME;  // equivalent to: I7 = I6, I6 = DM(0, I6)

_dotprod.end:

#endif
```

2.7 Updates to Hardware Breakpoint Support

2.7.1 Removed Hardware Breakpoints dialog box for ADSP-SC58x Cortex-A5 Cores

The Hardware Breakpoints dialog box for ADSP-SC58x Cortex-A5 Cores has been removed. Instructions for setting hardware breakpoints are located in the CCES Online Help in the page `Integrated Development Environment > Debugging Targets > Hardware Breakpoints/Watchpoints for SHARC Processors`.

2.7.2 Persistent SHARC+ Hardware Breakpoints

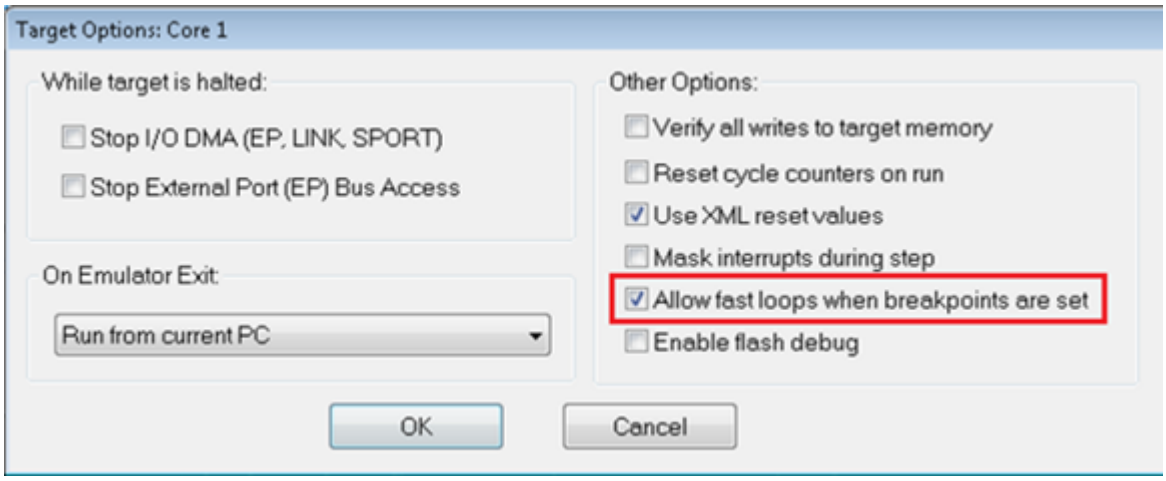
Hardware breakpoints on SHARC+ cores are no longer cleared after a `Run Menu->Restart` or `Reset` command.

2.7.3 Consistent debugging of hardware counter based loops for SHARC+ core (CCES-12421)

Software breakpoints inside F1-active mode loops cause incorrect execution.

By default, the emulator sets the `SLOWLOOP` bit in the `MODE2` Status register to allow for a better debugging experience when user software breakpoints are set. Setting a user software breakpoint anywhere in the code (except where there is an existing automatic breakpoint) will cause the `SLOWLOOP` bit to be set for that core. This mode causes all hardware counter based loops to execute in E2-active mode. This causes a slight increase in cycles in exchange for debugging capability within the loops.

If you do not set breakpoints within hardware counter based loops, then you can continue to execute loops in F1-active (ie. fast) mode, by selecting the '`Target`' menu, '`Settings`', '`Target Options`…', '`Allow fast loops when breakpoints are set`' option, as shown below.



This menu is only available once you have connected to the emulator, loaded your application and selected one of the SHARC+ cores. Note that the Target Options cannot be accessed while the core is running. The ‘`Allow fast loops when breakpoints are set`’ option will affect both SHARC+ cores. The target option is only valid for the debug configuration in which it was set.

The `MODE2 SLOWLOOP` bit is not changed immediately after setting or removing software breakpoints or changing the ‘`Allow fast loops when breakpoints are set`’ setting. The setting is changed when the user resumes debugging the application.

Setting the `SLOWLOOP` bit manually through the Register Browser, or executing instructions that change it, may interfere with the debugger’s settings and result in unexpected behavior.

For more information on F1 and E2-active loops, see sections `Loop Categories and Restrictions on Ending Loops` in CCES online help.

> **Note**
>
> Note that the restrictions on ending loops are still applicable. Even with `SLOWLOOP` bit set, a breakpoint on the last instruction of a loop may still cause incorrect execution.

2.8 ADSP-SC58x/ADSP-2158x processors and JTAG

2.8.1 Support for connecting multiple ADSP-SC58x/ADSP-2158x processors in the same JTAG scan chain

Support has been added to allow for connecting multiple ADSP-SC58x/ADSP-2158x processors in the same JTAG scan chain using the CrossCore Debugger. See section `Configuring a Scan Chain` in the CCES online help for more information on JTAG scan chains.

2.8.2 Able to connect to target boards with multiple SHARC processors (CCES-13791)

It is again possible to connect to target boards with more than one SHARC processor in the same JTAG scan chain.

2.8.3 Unlocking a secure ADSP-SC58x/ADSP-2158x processor

The ADSP-SC58x/ADSP-2158x processors can be locked in order to protect customer IP. Once a processor is locked, it cannot be accessed externally by an emulator unless a special key is provided, which must also have been programmed into the processor's OTP memory *before* the part is locked.

A custom board support file is used in this situation in order to make the access key available to the emulator. For more details, see the CCES Online Help under `Integrated Development Environment > Custom Board Support > Unlocking a Secure Processor`.

2.9 ADSP-SC58X debugging improvements

Performance has significantly improved for the ADSP-SC589 Cycle Accurate Simulator.

Performance has significantly improved for ARM core debugging using the CrossCore Debugger.

The Emulator and Instruction Set Simulator now allow each SHARC core to access the L1 memory of the other SHARC core in the multiprocessor space.

2.10 ChipFactory Simulator Target name change

The ChipFactory Simulator encompasses both Cycle Accurate and Instruction Set simulators. To correctly reflect this, the ChipFactory Simulator Debug Configuration Target name has changed from `Chipfactory Instruction Set Simulator` to simply `Chipfactory Simulator`. Any existing Debug Configurations using `Chipfactory Instruction Set Simulator` will no longer load and will need to be recreated.

3 Known Issues

3.1 Watchdog timer not halted when exiting user space (ADSP-BF70x, ADSP-SC58x, ADSP-2158x)

When users are debugging and the watchdog timer is enabled, the watchdog timer will not halt when halting the application in the debugger. This behaviour will occur when stepping, halting at a breakpoint, or halting using the debug menu.