CCES 1.1.0 Release Notes

Introduction

This document describes the changes for CrossCore Embedded Studio (CCES) 1.1.0. This release adds support for the ADSP-BF70x processor family revision 0.1, the new ICE-1000 and ICE-2000 emulator hardware and general maintenance updates.

Supported Operating Systems

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

- Windows XP Professional SP3 (32-bit only)
- 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 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.

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 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.

Hardware Requirements

ICE-100B

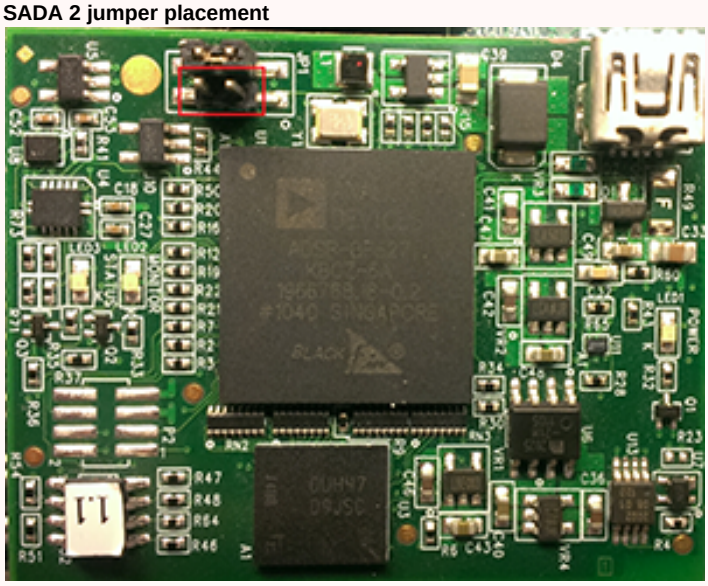
Starting with CCES 1.1.0 users are required to install a jumper on JP2 of the ICE-100B in order to use the emulator. When going back to prior CCES releases or any VisualDSP++ release, users must have this jumper removed. When installing or removing the jumper, USB should be disconnected or unplugged and then plugged in again to the emulator after the jumper has been changed. Please refer to the following diagram for the location and proper placement of JP2 highlighted in red.

ICE-100B jumper placement

Standalone Debug Agent 2 (SADA 2)

Starting with CCES 1.1.0 users are required to install a jumper on JP1(pins 1+2) of the SADA 2 in order to use the direct USB connection through the debug agent. When going back to prior CCES releases or any VisualDSP++ release, users must have this jumper removed. When installing or removing the jumper, USB should be disconnected or unplugged and then plugged in again to the debug agent after the jumper has been changed. Please refer to the following diagram for the location and proper placement of JP1 highlighted in red.

SADA 2 jumper placement



New Functionality

Support for ADSP-BF70x Family Processors

This release of CCES adds support for the following new Blackfin+ parts as well as support for the new ADSP-BF707 EZ-Board evaluation

hardware.

- ADSP-BF700
- ADSP-BF701
- ADSP-BF702
- ADSP-BF703
- ADSP-BF704
- ADSP-BF705
- ADSP-BF706
- ADSP-BF707

> **Note**
> Please note that you will need an ICE-1000 emulator or an ICE-2000 emulator to connect to an ADSP-BF70x processor. Previous ICE emulators are not compatible with the ADSP-BF70x processor family.

Default Configurations

When a new project is created for a processor in the ADSP-BF70x family, the out-of-the-box configuration is set by the Startup Code/LDF Add-in, and is as follows:

- Instruction cache is enabled.
- Data cache is disabled and Data CPLBs for memory protection is enabled.
- Parity-checking is enabled, for L1 Instruction SRAM and L1 Data SRAM.
- L2 ECC is enabled.
- External memory support is disabled.
- Stack and heap are allocated using L1 data memory.

When a new application is built without the Startup Code/LDF Add-in, the configuration is provided by the run-time library and default LDF, as follows:

- Instruction cache is enabled.
- Data cache and data CPLBs are enabled.
- Parity-checking is enabled, for L1 Instruction SRAM and L1 Data SRAM.
- L2 ECC is enabled.
- External memory support is disabled.
- Stack and heap are allocated using L1 data memory.

Utility ROM

The ADSP-BF70x processors include extensive functionality in L2 ROM space, which includes:

- Commonly-accessed run-time library support, including DSP library functions and maths emulation.
- Device drivers.
- Two configurations of the Micrium uC/OS-III Real-Time Operating System. The Micrium uC/OS-III Add-in must be installed and the appropriate add-ins must be added to the application to utilitize either of these configurations.
- FFT twiddle tables.

Support for using this functionality is enabled by default when building applications for ADSP-BF70x parts using CCES 1.1.0. A list of the functions and data exported by the L2 Utility ROM is included later in this document.

Adding a driver to an application as source via the add-in mechanism only disables the ROM version of that specific driver. Any API calls to other drivers that are including in the ROM still uses the ROM.

Debug Support for ADSP-BF707 EZ-Board Evaluation Hardware

This release of CCES adds native debug support for the ADSP-BF707 EZ-Board.

Functional Simulation of ADSP-BF70x Processors

In this release, functional simulation of the ADSP-BF70x parts is supported for the following components:

- Instruction set and core machine state
- Core event controller
- Internal and external memory spaces
- Utility ROM images in L2 ROM space
- Core timer

Many additional system components and peripherals such as below are being developed for future releases of the simulator but are not yet present:

- DDE, SEC, TRU, SPU, SMPU, CGU, DPM, RCU
- GPIO, PINT
- GPT, RTC, WDT
- SPI, SPT
- Other peripherals, accelerators, and infrastructure.

If your program references a currently unsupported simulation module you will receive ***unimplemented MMR*** error messages when attempting to program the module's MMR registers.

Simulation of the ADSP-BF70x parts can be done in a CrossCore Embedded Studio simulator session or using the command line simulator

Interactive Debug using CCES

To establish a simulator session for the ADSP-BF707 part make the following selections on the Debug Configurations dialog:

Target	ChipFactory Instruction Set Simulator
Platform	ADSP-BF707 Functional-Sim
Processor	ADSP-BF707

Command Line Simulation using CHIPFACTORY.EXE

ADSP-BF70x parts may also be simulated using the "chipfactory.exe" command line simulator.

Simulator command-line switches

The simulator supports the following command-line switches:

`-bound-exe-time` ***`num`***

Bound the execution to *num* cycles. 0 means infinity. (Can also be achieved with environment variable ADI_CCES_SIM_BOUND_EXE_TIME)

Default value : 0x0

`-help`

Print help information

`-ignore-fail-label`

The simulator will halt if it encounters a “fail” label. This is a label that begins with one or more underscores, followed by the four characters “fail”. For example, both `_fail` and `____fail_on_return` are fail labels, and will cause the simulation to halt. Use the `–ignore-fail-label` switch to cause the simulator to ignore fail labels.

Default value : false

`-syntax`

Print syntax information

`-proc` ***`proc-name`***

Simulate processor identified by proc-name. For example:

```
-proc ADSP-BF707.
```

Default value: none - this is a required switch.

`-sim-type` ***`sim-type`***

Platform name. Use the following value, for ADSP-BF70x processors:

```
-sim-type Functional-Sim
```

Default value: Functional-Sim

`-quiet`

Be as quiet as possible

Default value: false

`-xml-path` ***`path`***

Sets the path to the folder containing the 'System/Chipfactory' directory in order to load the processor description and instantiate the described simulator.

The ***sim-type*** and ***proc-name*** are also used to select the desired processor description.

Default value : The location of the CrossCore Embedded Studio installation directory.

`-no-exit-value`

Ignore the value passed to exit() by the application being simulated.

Default value : false

`-version`

Print version number

Chipfactory command line simulator example

```
chipfactory -proc ADSP-BF707 hello.dxe
Unimplemented MMR Write to 0x20000020
Unimplemented MMR Read from 0x20000060
Unimplemented MMR Write to 0x20000064
Hello World
Halt label "___lib_prog_term" hit at PC = 0x11a03096
 - Simulation result : 0 (ok)
 - Simulated application result : 0

        (69304 simulated chip cycles) : (end-start wall clock=30)       (2310.13
KHz)   (2.31 MHz)
```

In this simulation, support for ROM image used in interactive debugging will also be supported when using the command line simulator. However, whether a specific portion of a ROM image can be simulated is a function of whether support for all parts of the processor required are being simulated.

ICE-1000 Emulator

Support for a new low cost emulator has been added to this release of CCES which can debug all legacy SHARC and Blackfin processors. The ICE-1000 is one of two emulators that will allow users to debug the ADSP-BF70x family processors.

ICE-2000 Emulator

Support for a new emulator has been added to this release of CCES which can debug all legacy SHARC and Blackfin processors. The ICE-2000 is one of two emulators that will allow users to debug the ADSP-BF70x family processors. The ICE-2000 operating frequency can be 1MHz, 5MHz, 9MHz, 15MHz, 23MHz, or 46MHz but defaults to 9MHz. In order to successfully run at a certain emulator operating frequency, it is recommended that the core clock of the processor being debugged is running at 2X the emulator operating frequency. So if the emulator operating frequency is 9MHz, it is recommended that the core clock be at least 18MHz. Other variations on the target board can also affect the emulator operating frequency. Additionally the **ADSP-BF70x processors** should have a **core clock and system clock** which is 2X the emulator operating frequency.

Changed Functionality

Blackfin Compiler

Compiler error cc0137

The Blackfin compiler 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;
  }
}
```

Run Time Libraries

Data cache invalidation

The library functions dcache_invalidate() and cache_invalidate() now always use the appropriate configuration bits to invalidate the data caches for all Blackfin parts. This means that the caches can no longer be invalidated individually, with the exception of BF70x parts where data cache B can be invalidated without also invalidating data cache A. Any other options will invalidate both cache banks, the equivalent of calling dcache_invalidate_both(). The behavior of instruction cache invalidation remains unchanged.

Scratchpad

The L1 SRAM space known as scratchpad in Blackfin architectures has been replaced by L1 Data C, in the Blackfin+ architecture, and the output sections in the .ldf files reflect this change. The .ldf files for ADSP-BF70x continue to accept L1_scratchpad input sections, which are mapped to L1 Data C.

Updated Language Standards Support

The Blackfin compiler accepts 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 compiler conforms to the ANSI/ISO 14882:2003 Standard.

The -g++ switch may be used with the Blackfin compiler. It directs the compiler 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.

These language conformance enhancements are only available in the Blackfin compiler. They are not available in the SHARC compiler.

Updated USB Controller Driver for ADSP-BF52x, ADSP-BF54x, ADSP-BF60x and new driver for ADSP-BF70x

The separate USB Device and Host mode controller drivers have been replaced with a single integrated driver for both host and device modes of operation. This integrated driver has also been ported to the ADSP-BF70x family of processors. It is also enhanced to provide Isochronous transfers and to improve stability.

As a result, the device driver API has been changed. As such, all C/USB Device Stack 1.0.0 and 1.0.1 products are incompatible with CCES 1.1.0, even though they may be displayed within the Add-in Manager dialog. Please upgrade your C/USB Device Stack products to release 1.1.0. Please refer to the Release Notes of the C/USB Device 1.1.0 products for further information.

Known Problems and Limitations

No System Reset

Currently only a core reset is supported on the BF70x which has shown limitations when peripherals are running at the time of a core reset. There may be cases where users will run an example and then reload to run the example a second time and they will get exceptions. This could be due to the peripheral interrupt being serviced at the wrong time causing an exception. In order to fix this, users need to identify the peripheral that is causing the issue and create a custom board support package xml file that adds the appropriate register to reset the peripheral. 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 standard support files for the upcoming release. An example custom board support file is shown below and more information on custom board support files can be found in the CCES help.

Custom Board Support(Resetting Registers)

```
<?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
1.1.0\System\ArchDef\ADSP-custom-board.xsd"
 processor-family="Blackfin"
    file="ADSP-BF706_mini_custom.xml">

<custom-register-reset-definitions>
    <!-- Reset EPPI peripheral -->
 <register name="EPPI0_CTL" reset-value="0x00000000" core="Common" />

</custom-register-reset-definitions>
</custom-cces-proc-xml>
```

Unlocking a Secure Part

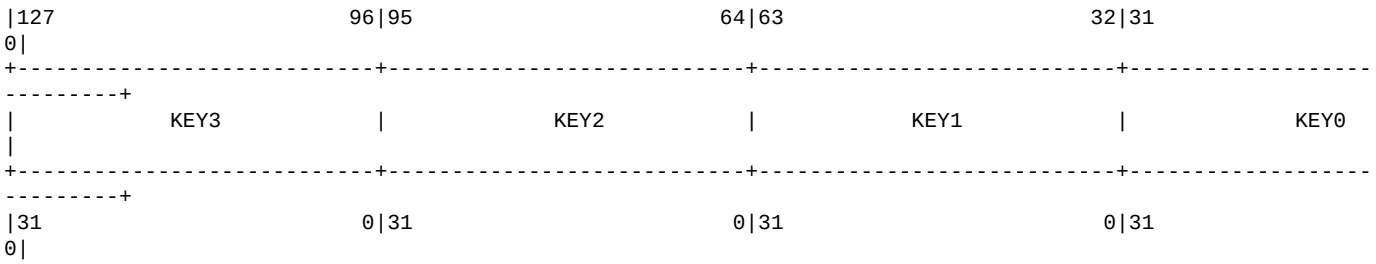
The ADSP-BF70x processors can be locked in order to protect customer IP. Once a processor is locked, it cannot be accessed externally by an emulator unless a special key is provided, which must also have been programmed into the processor's OTP memory *before* the part is locked.

A custom board support file is recommended in this situation with the contents similar to the following, in order to make the access key available to the emulator:

Custom Board Support Example

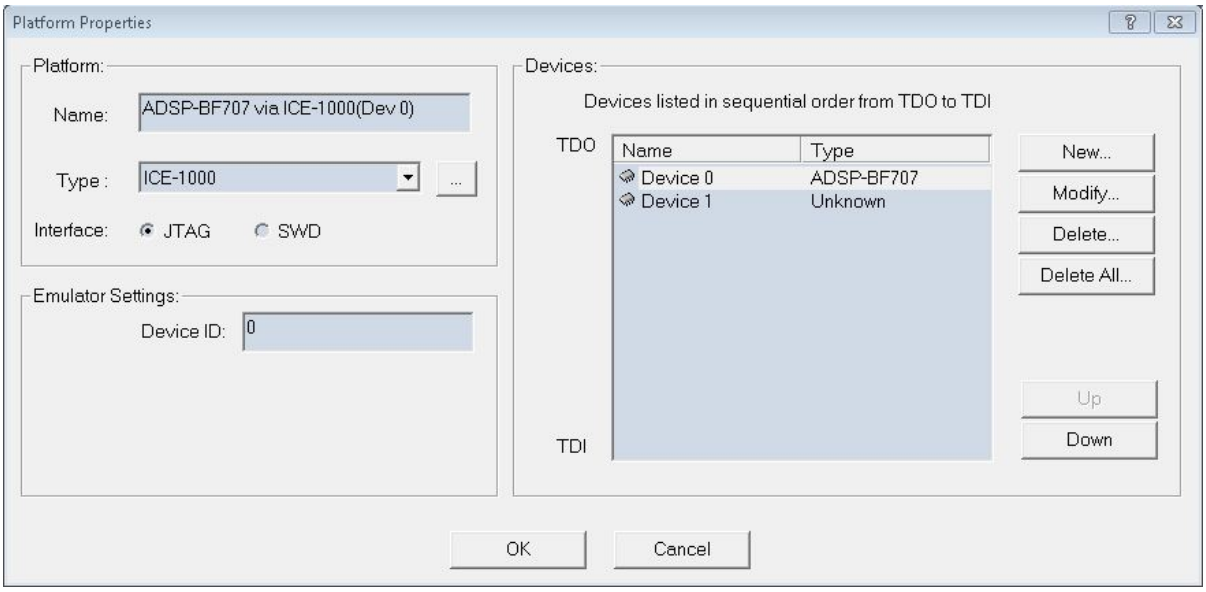
```
<?xml version="1.0" standalone="yes"?>
<custom-cces-proc-xml
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="\Analog Devices\CrossCore Embedded Studio
1.1.0\System\ArchDef\ADSP-custom-board.xsd"
 processor-family="Blackfin"
    file="example_custom_board_support.xml">

  <custom-register-reset-definitions>
    <register name="userkey0" reset-value="0x00001111" core="Common" />
 <register name="userkey1" reset-value="0x22223333" core="Common" />
 <register name="userkey2" reset-value="0x44445555" core="Common" />
 <register name="userkey3" reset-value="0x66667777" core="Common" />
  </custom-register-reset-definitions>
</custom-cces-proc-xml>
```

The 128 bit user key format is:



Debugging Multiprocessor BF70x Target

Multiprocessor support for more than 1 BF70x processor in a JTAG scan chain is limited. If there are more than 1 BF70x processor in a scan chain, users can only connect to 1 processor at a time. A custom platform would need to be created in this case using the Target Configurator so that the processors that are not being debugged are set to unknown as shown in the picture below. Even though Device 1 is also a BF70x processor, it is set to unknown when the user wants to debug Device 0. When the user wants to debug Device 1, then they will have to make Device 0 unknown and Device 1 would be set to ADSP-BF707.



Build Issues with Windows 8.0

There is a known bug in the Eclipse framework used by the CCES IDE which may cause builds to crash with the error: "starter.exe has quit unexpectedly". This crash has not been observed under either Windows 7 or Windows 8.1 and seems specific to just the first release of Windows 8. We are working to fix this issue in an upcoming release of CCES.

For more information please see the following bug report:

- Bug 429838 - starter.exe has stopped working [pop up when building with eclipsec.exe]

Interaction with Verdasys DGAgent

The CCES build tools (compiler, assembler and linker) can interact with the *Digital Guardian* product from Verdasys (https://www.verdasys.com/) such that the build is prevented from completing. This generally manifests as a "compiler hang". We are working to identify the reason for this issue. If the compiler appears to be hanging, you can determine whether you're running into this problem, using Microsoft's freely-available ProcessExplorer utility:

- Invoke ProcessExplorer.
- Click on the CPU column, to sort by CPU utilization.
- Double-click on the process that is consuming the most CPU cycles.
- Click on the *Threads* tab.
- If the *Start Address* column lists dgapi.dll, then you're hitting this problem.

To obtain ProcessExplorer, please visit http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx.

ADSP-BF707 Secure Boot Requires LDR File With No Fill Blocks

ADSP-BF70x processors support Secure Booting, where the loader stream is digitally signed and/or encrypted using the new signtool utility, and the processor's Boot Kernel decrypts the stream and authenticates the signature before permitting the stream to boot. Silicon anomaly 19000022 describes a problem processing "fill blocks" when booting in secure mode, which causes the boot to fail. To avoid this problem, build the loader stream using the elfloader's -NoFillBlock switch.

There is no problem processing "fill blocks" in non-secure mode.

Some CAN registers are displayed incorrectly in register windows

Several sets of registers for the CAN0 and CAN1 peripherals of the ADSP-BF70x processor family are not displayed correctly when viewed in register windows in the IDE. They have fields which are defined as being in bit positions 0-7 within the register, but which should be defined to be in positions 8-15. This issue is CCES-7713, and does not affect the correct functioning of applications.

Functions in L2 Utility ROM

C and DSP Run-Time Library Functions

The following table contains a list of functions from the C and DSP run-time libraries that are also available in the L2 Utility ROM. The ROM also includes compiler support functions, which are hidden routines that the compiler knows about and relies on to provide run-time services for the code that it generates (such as emulation of floating-point arithmetic).

Function	Header File
a_compress	filter.h
a_expand	filter.h
abs	stdlib.h
abs_fr16	stdlib.h
acos_fr16	math.h
acos_fr32	math.h
acosd	math.h
acosf	math.h
alog10d	math.h
alog10f	math.h
alogd	math.h
alogf	math.h
arg_fr16	complex.h
arg_fr32	complex.h
argd	complex.h
argf	complex.h
asin_fr16	math.h
asin_fr32	math.h
asind	math.h
asinf	math.h
atan_fr16	math.h
atan_fr32	math.h
atan2_fr16	math.h
atan2_fr32	math.h
atan2d	math.h
atan2f	math.h
atand	math.h
atanf	math.h
autocoh_fr16	stats.h
autocoh_fr32	stats.h
autocohd	stats.h
autocohf	stats.h
autocorr_fr16	stats.h
autocorr_fr32	stats.h

autocorrd	stats.h
autocorrf	stats.h
cabs_fr16	complex.h
cabs_fr32	complex.h
cabsd	complex.h
cabsf	complex.h
caddd	complex.h
cartesian_fr16	complex.h
cartesian_fr32	complex.h
cartesiand	complex.h
cartesianf	complex.h
cdiv_fr16	complex.h
cdiv_fr32	complex.h
cdivf	complex.h
ceild	math.h
ceilf	math.h
cexpd	complex.h
cexpf	complex.h
cfft_fr16	filter.h
cfft_fr32	filter.h
cfft2d_fr16	filter.h
cfft2d_fr32	filter.h
cfftf_fr16	filter.h
cfftf_fr32	filter.h
cfir_fr16	filter.h
cfir_fr32	filter.h
clip	math.h
clip_fr16	math.h
cmatmmlt_fr16	matrix.h
cmatmmlt_fr32	matrix.h
cmatmmltd	matrix.h
cmatmmltf	matrix.h
cmlt_fr32	complex.h
cmltd	complex.h
coeff_iirdf1_fr16	filter.h
coeff_iirdf1_fr32	filter.h
conjd	complex.h
conv2d_fr16	filter.h

conv2d_fr32	filter.h
conv2d3x3_fr16	filter.h
conv2d3x3_fr32	filter.h
convolve_fr16	filter.h
convolve_fr32	filter.h
copysign_fr16	math.h
copysign_fr32	math.h
copysignd	math.h
copysignf	math.h
cos_fr16	math.h
cos_fr32	math.h
cosd	math.h
cosf	math.h
coshd	math.h
coshf	math.h
cotd	math.h
cotf	math.h
countones	math.h
crosscoh_fr16	stats.h
crosscoh_fr32	stats.h
crosscohd	stats.h
crosscohf	stats.h
crosscorr_fr16	stats.h
crosscorr_fr32	stats.h
crosscorrd	stats.h
crosscorrf	stats.h
csubd	complex.h
cvecdot_fr16	vector.h
cvecdot_fr32	vector.h
cvecdotd	vector.h
cvecdotf	vector.h
cvecsadd_fr16	vector.h
cvecsadd_fr32	vector.h
cvecsaddd	vector.h
cvecsaddf	vector.h
cvecsmlt_fr16	vector.h
cvecsmlt_fr32	vector.h
cvecsmltd	vector.h

cvecsmltf	vector.h
cvecssub_fr16	vector.h
cvecssub_fr32	vector.h
cvecssubd	vector.h
cvecssubf	vector.h
cvecvadd_fr16	vector.h
cvecvadd_fr32	vector.h
cvecvaddd	vector.h
cvecvaddf	vector.h
cvecvmlt_fr16	vector.h
cvecvmlt_fr32	vector.h
cvecvmltd	vector.h
cvecvmltf	vector.h
cvecvsub_fr16	vector.h
cvecvsub_fr32	vector.h
cvecvsubd	vector.h
cvecvsubf	vector.h
expd	math.h
expf	math.h
fabs	math.h
fabsd	math.h
fabsf	math.h
fclipd	math.h
fclipf	math.h
fft_magnitude_fr16	filter.h
fft_magnitude_fr32	filter.h
fir_decima_fr16	filter.h
fir_decima_fr32	filter.h
fir_fr16	filter.h
fir_fr32	filter.h
fir_interp_fr16	filter.h
fir_interp_fr32	filter.h
floord	math.h
floorf	math.h
fmaxd	math.h
fmind	math.h
fminf	math.h
fmodf	math.h

frexpd	math.h
frexpf	math.h
gen_bartlett_fr16	window.h
gen_bartlett_fr32	window.h
gen_blackman_fr16	window.h
gen_blackman_fr32	window.h
gen_gaussian_fr16	window.h
gen_gaussian_fr32	window.h
gen_hamming_fr16	window.h
gen_hamming_fr32	window.h
gen_hanning_fr16	window.h
gen_hanning_fr32	window.h
gen_harris_fr16	window.h
gen_harris_fr32	window.h
gen_kaiser_fr16	window.h
gen_kaiser_fr32	window.h
gen_rectangular_fr16	window.h
gen_rectangular_fr32	window.h
gen_triangle_fr16	window.h
gen_triangle_fr32	window.h
gen_vonhann_fr16	window.h
histogram_fr16	stats.h
histogram_fr32	stats.h
histogramd	stats.h
histogramf	stats.h
ifft_fr16	filter.h
ifft_fr32	filter.h
ifft2d_fr16	filter.h
ifft2d_fr32	filter.h
ifftf_fr16	filter.h
ifftf_fr32	filter.h
iir_fr16	filter.h
iir_fr32	filter.h
iirdf1_fr16	filter.h
iirdf1_fr32	filter.h
isalnum	ctype.h
isalpha	ctype.h
iscntrl	ctype.h

isdigit	ctype.h
isgraph	ctype.h
isinf	math.h
isinfd	math.h
islower	ctype.h
isnan	math.h
isnand	math.h
isprint	ctype.h
ispunct	ctype.h
isspace	ctype.h
isupper	ctype.h
isxdigit	ctype.h
labs	stdlib.h
lcountones	math.h
ldexpd	math.h
llabs	stdlib.h
llclip	math.h
lldiv	stdlib.h
llmax	math.h
llmin	math.h
log10d	math.h
log10f	math.h
logd	math.h
logf	math.h
matmmlt_fr16	matrix.h
matmmlt_fr32	matrix.h
matmmltd	matrix.h
matmmltf	matrix.h
max	math.h
max_fr16	math.h
maxf	math.h
mean_fr16	stats.h
mean_fr32	stats.h
meand	stats.h
meanf	stats.h
meansf	stats.h
memchr	string.h
memcmp	string.h

memcpy	string.h
memmove	string.h
memset	string.h
min	math.h
min_fr16	math.h
modfd	math.h
modff	math.h
mu_compress	filter.h
mu_expand	filter.h
normd	complex.h
normf	complex.h
polar_fr16	complex.h
polar_fr32	complex.h
polard	complex.h
polarf	complex.h
powd	math.h
powf	math.h
rfft_fr16	filter.h
rfft_fr32	filter.h
rfft2d_fr16	filter.h
rfft2d_fr32	filter.h
rfftf_fr16	filter.h
rfftf_fr32	filter.h
rms_fr16	stats.h
rms_fr32	stats.h
rmsd	stats.h
rmsf	stats.h
rsqrtd	math.h
rsqrtf	math.h
sin_fr16	math.h
sin_fr32	math.h
sind	math.h
sinf	math.h
sinhd	math.h
sinhf	math.h
sqrt_fr16	math.h
sqrt_fr32	math.h
sqrtd	math.h

sqrtf	math.h
strcat	string.h
strchr	string.h
strcmp	string.h
strcoll	string.h
strcpy	string.h
strcspn	string.h
strftime	string.h
strlen	string.h
strncat	string.h
strncmp	string.h
strncpy	string.h
strpbrk	string.h
strrchr	string.h
strspn	string.h
strstr	string.h
strxfrm	string.h
tan_fr16	math.h
tan_fr32	math.h
tand	math.h
tanf	math.h
tanhd	math.h
tanhf	math.h
tolower	ctype.h
toupper	ctype.h
transpm_fr16	matrix.h
transpm128	matrix.h
transpm16	matrix.h
transpm32	matrix.h
transpm64	matrix.h
transpmd	matrix.h
transpmf	matrix.h
twidfft2d_fr16	filter.h
twidfftf_fr16	filter.h
twidfftf_fr32	filter.h
twidfftrad2_fr16	filter.h
twidfftrad2_fr32	filter.h
var_fr16	stats.h

var_fr32	stats.h
vard	stats.h
varf	stats.h
vecdot_fr16	vector.h
vecdot_fr32	vector.h
vecdotd	vector.h
vecdotf	vector.h
vecmax_fr16	vector.h
vecmax_fr32	vector.h
vecmaxd	vector.h
vecmaxf	vector.h
vecmaxloc_fr16	vector.h
vecmaxloc_fr32	vector.h
vecmaxlocd	vector.h
vecmaxlocf	vector.h
vecmin_fr16	vector.h
vecmin_fr32	vector.h
vecmind	vector.h
vecminf	vector.h
vecminloc_fr16	vector.h
vecminloc_fr32	vector.h
vecminlocd	vector.h
vecminlocf	vector.h
vecsadd_fr16	vector.h
vecsadd_fr32	vector.h
vecsaddd	vector.h
vecsaddf	vector.h
vecsmlt_fr16	vector.h
vecsmlt_fr32	vector.h
vecsmltd	vector.h
vecsmltf	vector.h
vecssub_fr16	vector.h
vecssub_fr32	vector.h
vecssubd	vector.h
vecssubf	vector.h
vecvadd_fr16	vector.h
vecvadd_fr32	vector.h
vecvaddd	vector.h

vecvaddf	vector.h
vecvmlt_fr16	vector.h
vecvmlt_fr32	vector.h
vecvmltd	vector.h
vecvmltf	vector.h
vecvsub_fr16	vector.h
vecvsub_fr32	vector.h
vecvsubd	vector.h
vecvsubf	vector.h
zero_cross_fr16	stats.h
zero_cross_fr32	stats.h
zero_crossd	stats.h
zero_crossf	stats.h

The L2 ROM also contains these pre-computed twiddle tables, which can used when generating FFTs:

- `const complex_fract16 twidfftf_fr16_8k_table[]`
- `const complex_fract32 twidfftf_fr32_4k_table[]`
- `const complex_fract16 twidfftrad2_fr16_8k_table[]`
- `const complex_fract32 twidfftrad2_fr32_4k_table[]`

Device Driver functions

All APIs related to the following drivers:

- Rotary counter
- EPPI
- SPI
    - Interrupt mode is not functional for the ROM version of the SPI driver. To use interrupt mode you will need to include the SPI sources into your project using the Add-In manager.
- SPORT
- TWI

RTOS functions

All APIs in

- C-LIB version v1.37.00
- C-CPU version 1.29.01
- COS-III version 3.03.01