



# CellBE Execution Framework User Guide

Document: HSS0007-UG-01  
Customer: Open Source  
Author: Gavin Nottage  
Status: Release  
Issue: 1  
Date: 09 January 2009

Harmonic Software Systems Ltd

Unit 3 Basepoint Business & Innovation Centre, Metcalf Way, Crawley, RH11 7XX

01293 817635

[www.harmonicss.co.uk](http://www.harmonicss.co.uk)

[sales@harmonicss.co.uk](mailto:sales@harmonicss.co.uk)

© Copyright Harmonic Software Systems Ltd, all rights reserved.

Copyright subsists in all Harmonic Software System Ltd deliverables including magnetic, optical and/or any other soft copy of these deliverables. This document may not be reproduced, in full or in part, without written permission.

## Issue History

ISSUE	DATE	SAVED BY	COMMENT
1	9 January 2009	HSS	Initial Release

## Contents

<a href="#">1 Introduction.....</a>	<a href="#">3</a>
<a href="#">2 Installation &amp; Initial Configuration.....</a>	<a href="#">4</a>
<a href="#">3 CEF Design.....</a>	<a href="#">6</a>
<a href="#">4 Developing An Algorithm.....</a>	<a href="#">8</a>
<a href="#">5 Building.....</a>	<a href="#">10</a>
<a href="#">6 Usage.....</a>	<a href="#">11</a>
<a href="#">7 Testing.....</a>	<a href="#">12</a>
<a href="#">8 Limitations of CEF.....</a>	<a href="#">15</a>

# 1 Introduction

This users guide provides usage instructions for the CellBE Execution Framework. A general understanding of the CellBE processor is assumed.

## 1.1 *Intended Audience*

This users guide is intended for anyone who wants to use CEF.

## 1.2 *CEF Licence*

CEF is released under the BSD licence. This users guide is released with CEF, and should be supplied with any version of CEF (whether modified or not) without removal of the Harmonic Software Systems Logo and branding.

## 1.3 *References*

1. *Harmonic Software Systems Website:* [www.harmonicss.co.uk](http://www.harmonicss.co.uk)
2. *SDK Installation Guide, IBM*

## 1.4 *Glossary*

CEF                    Cell Execution Framework

## 2 Installation & Initial Configuration

### 2.1 Requirements

The following describe the platforms used to create, develop and test CEF at Harmonic Software Systems. Other configurations may also work.

#### 2.1.1 Compilation

- CellBE SDK 3.0 from IBM, running on Fedora Linux 7
- CellBE SDK 3.1 from IBM, running on Fedora Linux 9

#### 2.1.2 Execution

- SDK 3.0 Simulator
- PS3 running Yellow Dog Linux 6

### 2.2 Environment

The following environment values should be setup:

- CELL\_TOP
- CEF\_HOME

They can be set by adding the following to the `.bashrc` file in the home directory:

```
CELL_TOP='/opt/cell/sdk'  
export CELL_TOP  
CEF_HOME='path_of_cef'  
export CEF_HOME
```

It's possible to check that they has been set with the following commands:

```
env | grep CELL_TOP  
env | grep CEF_HOME
```

### 2.3 Installation

The CEF code is released as a tar.bz2 file, and thus requires the following to extract the source files:

```
tar -xjf cef.tar.bz2
```

The CEF can be extracted anywhere as long as the environment is setup.

### 2.4 Directory Structure

The release contains the following directory structure:

```
/doc  
/cef  
/tools
```

The `doc` directory contains CEF documentation, including this document.

The `cef` directory contains the Cell BE Execution Framework code, for the PPE and SPE, as well as the user algorithm:

```
/cef/core/ppe
/cef/core/spe
/cef/user/FFT
/cef/user/rotation
/cef/user/template
/cef/user/testthru
```

Four user algorithms are provided. One is a template, which can be copied and renamed to create a new user algorithm. The `testthru` algorithm will copy incoming data into the output stream.

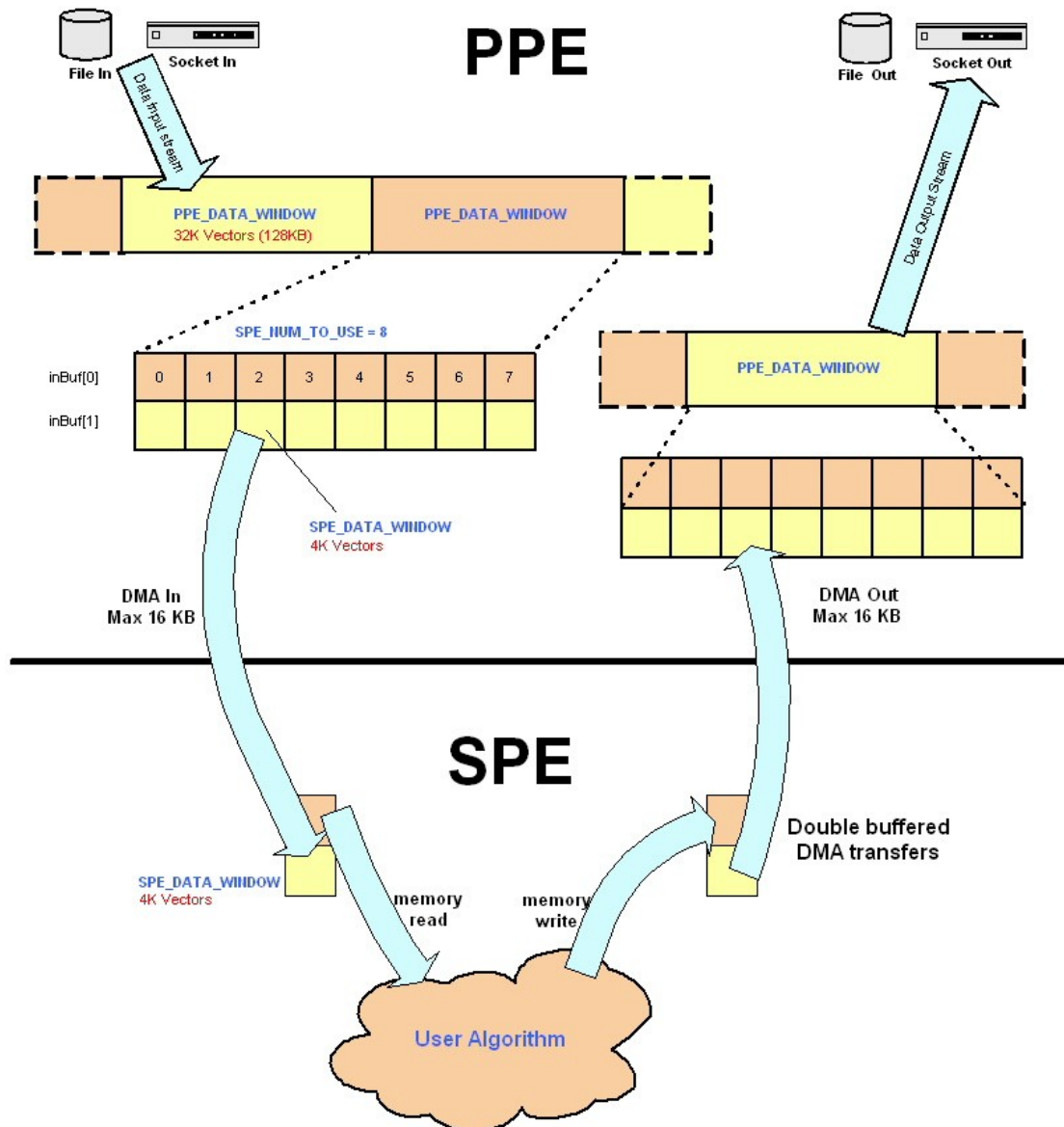
### 3 CEF Design

This section explains important aspects of the design of the framework, so that algorithms and data structures can be designed to suit.

#### 3.1 Overview

CEF can be thought of as a way to get your algorithm processing data faster. Therefore from the outside it has been designed to be simple to use. Input data and output data can be read from or written to files, or use a socket.

The following diagram shows the inner workings of CEF:



The input data is read in chunks (`PPE_DATA_WINDOW`) that fulfil the algorithm for all SPEs used. It's double buffered so that while the SPE is DMAing and processing, the next set of data can be read in by the PPE. Further details can be found in the comments in the `cef.c` and `cef_spe.c` files in the `core` directory.

## 3.2 *Directory Structure*

Within the `cef` directory, there are two subdirectories, one each for the `core` files that don't need any user changes, and the `user` files that hold the algorithms.

Each algorithm must have its own directory under the `user` directory. There are several examples and a template included to guide a user. See [Developing An Algorithm](#).

### 3.2.1 Configuration

Configuration is done for each algorithm within the respective `alg_config.h` file, where the number of SPEs to be used, and how much data each SPE should process at a time can be set.

Other configuration parameters are considered run-time and are command line parameters. These are the input and output type (file or socket), and verbosity. See [6.1](#) below.

## 4 Developing An Algorithm

### 4.1 *Where*

Each user algorithm resides in its own directory under the user directory. A template directory is included to enable a developer to start coding or porting their algorithm straight away. Copy the template directory and rename to create a new directory with a unique name.

The Makefile should be copied, but not require any changes.

### 4.2 *Example code*

There are examples of algorithms with associated data structures provided under the user directory.

#### 4.2.1 FFT

As released, this uses a 1k Fast Fourier Transform, as detailed in the SDK Example Library API.

#### 4.2.2 Rotation

For 3D modelling data, it's useful to be able to rotate each point. This example simply rotates the input data in 3D according to the degrees defined in the algorithm.

#### 4.2.3 Testthru

This is a simple test for CEF, where the input data is returned directly as output data.

### 4.3 *Data*

The algorithm developed needs an input data stream to be processed, creating the output data stream. Within CEF the PPE can read an input stream from a file or socket, and write the output stream to a separate file or the same socket.

To achieve the best performance, the data should be structured to take advantage of the register width and DMA abilities of the CellBE processor. It is advisable to structure the data this way before writing the algorithm that will process it.

Since the SPE has limited space to store the data that its algorithm processes, the PPE breaks it down into chunks. There is a limit of 16kB (4096 registers). It is algorithm dependant as to how this data chunk affects things. For the FFT, it determines the size, but for other algorithms it makes no discernable difference, as if it were a single algorithm natively processing the large data set.

At present there is a limitation of a single input buffer, however, the algorithm can accept parameters through the use of command line arguments. These parameters are passed to the SPE as doubles, but can then be cast into an appropriate data type.

Note that the input data should be a multiple of the PPE\_DATA\_WINDOW. If this is not the case, as warning will be displayed when the CEF is run. One way to get around this is to pad the input data.

### 4.3.1 Explaining Data types and `alg_config.c`

Each algorithm requires a configuration to tell CEF how to handle the data that it transfers. The file includes comments to inform the user of each parameters usage. It is worth explaining that there are distinctions in data types that can be used within CEF.

**Register** 128-bit wide register. Multiple values will be inserted into each register as configured.

**Byte** The standard 8-bit data representation. DMA transfers use this data type to define an exact size of data. There are 16 bytes in a register.

## 4.4 Supporting Code

In the `tools` directory are some examples of simple data generation tools: The `cos` and `sin` tools generate waveforms for FFT usage, and `testthru` generates a simple incrementing number. In the `SCFFT` directory is a more advanced application that generates a sine wave and uses a socket to communicate with CEF.

This supporting code is provided as examples, so that a developer can easily add the functionality in their own situation. All example code is expected to natively run on the Linux host, and is compiled with GCC.

## 5 Building

### 5.1 Command Line

Simply traverse into the directory containing the algorithm to be used (e.g. `cef/user/FFT`), and type:

```
make
```

This will include the core files and the selected user algorithm. The resulting CellBE executable file can be found in the `core/pep` directory, and is called `cef`.

For an FFT algorithm, the Makefile will define `WRITE_FFT_STYLE` which will make sure that the data is read from and written to the file or socket with consideration for both the real and imaginary components.

## 6 Usage

Once the framework including a specific algorithm is built, it can be run from the command line.

### 6.1 Command Line Options

The following arguments allow input and output data buffers to be configured:

Mnemonic	Parameters	Description
-i	filename	Input File
-o	filename	Output File
-s	port	Socket (both input and output)
-v		Verbose mode for PPE
-w		Verbose mode for SPE
-b		Benchmark
-1 .. 8	value	Generic Parameters
-h		Help

## 7 Testing

This section details the methods used to test the release of CEF. It not only describes by example the usage of CEF, but also supporting code that is included to enable testing.

### 7.1 Algorithms

Each supplied algorithm is subject to at least one test. They are briefly described here to indicate where they fit into the testing regime. Remember that CEF is built from within an algorithm directory, and can support multiple algorithms by recompiling. Therefore the testing uses multiple algorithms, and details of the algorithm and its configuration are given.

#### 7.1.1 Testthru

This algorithm simply copies the input data to the output, allowing for a quick check that CEF is operating correctly.

#### 7.1.2 Rotation

This algorithm is able to rotate 3-dimensional points about an axis in all 3-dimensions. It's included to test bespoke additional parameters.

#### 7.1.3 FFT

This algorithm uses an FFT function supplied as a library with the SDK. The majority of testing uses this algorithm.

### 7.2 Tools

Included within CEF are several tools that provide either files or functionality use for the testing of the main framework. To build them, simply type make within the source directory.

#### 7.2.1 sin & cos

These are GNU/Linux executables that take arguments and generates output files of a simple sine or cosine wave with those characteristics.

Mnemonic	Parameters	Description
-o	filename	Output File
-p	value	Number of Points
-f	value	Frequency
-m	value	Magnitude
-h		Help

## 7.2.2 testthru

Another output file generating GNU/Linux executable. The file created is simply a list of incrementing numbers.

Mnemonic	Parameters	Description
-o	filename	Output File
-p	value	Number of Points
-h		Help

## 7.2.3 SCFFT

SCFFT (Socket Client for FFT) is a more advanced application that acts as a socket client to interact with CEF in its server role. It generates sine values and then sends them to CEF to process an FFT upon the data. The returned FFT data can be written to a file within SCFFT. For a thorough understanding of SCFFT it is advised to look at the comments in the source files.

Mnemonic	Parameters	Description
-i	text	IP Address
-p	value	Port of CEF server
-n	value	Number of values
-o	filename	Output File
-f	value	Frequency
-c	value	Change in Frequency
-h		Help

## 7.3 Test Cases

The tests can be run within the simulator or on an actual target running GNU/Linux. For the simulator no callthru steps are described, but would be required.

### 7.3.1 No Data

1. Run any compilation of CEF without specifying any input or output data sources

```
$ ./cef
```

### 7.3.2 Identical File Test

1. If not already compiled, compile the testthru tool

```
$ cd tools/testthru
```

- ```
$ make
```
2. Generate a test file with 24k data points
 

```
$ ./testthru -o test-data.in -p 24576
```
  3. Compile the testthru algorithm specifying 6 SPEs and a 2k data chunk within alg\_config.h.
 

```
$ cd user/testthru
$ make
```
  4. Run the testthru algorithm in CEF
 

```
$ ./cef -i test-data.in -o test-data.out
```
  5. Compare the files test-data.in and test-data.out, they should be identical.

### 7.3.3 Incomplete File Test

Follow the instructions for 7.3.2 above, except use 25k of data points. The output file should be truncated, with the last 1k data points missing. Also a warning should be seen regarding missing data.

### 7.3.4 Socket FFT Test

Following on from the mundane tests of basic functionality, the more complex FFT algorithm, coupled with data I/O using sockets proves that CEF works as intended.

This test case is to be executed between a dedicated target running CEF and the development workstation running SCFFT. It is not tested on the simulator.

1. If not already compiled, compile the SCFFT tool specifying a 1k PACKET\_SIZE and 12 BUFFERS within scfft.h
 

```
$ cd tools/SCFFT
$ make
```
2. Compile the FFT algorithm specifying 6 SPEs and a 4k data chunk within alg\_config.h
 

```
$ cd user/FFT
$ make
```
3. Run CEF on the target, supplying a port number
 

```
$ ./cef -s 2262
```
4. Run SCFFT on the workstation, supplying the IP address of the target, the same port number, an output filename, 144k points at 16Hz rising 1 Hz every 2k points.
 

```
$ ./scfft -i 192.168.1.104 -p 2262 -o scfft.out -n 147456
-f 16 -c 2048
```
5. View the output using gnuplot
 

```
gnuplot> plot "scfft.out" with dots
```

## 8 Limitations of CEF

The following limitations are present in this release of CEF:

- Maximum of 16kB data process chunk. Each SPE in use receives up to 16kB of data to process with the algorithm at any one time.
- Output Data size should be the same as Input Data Size. If output data is less, it can be padded in the algorithm.
- Only single Input and Output buffers are supported.
- The CEF is compile as a 32-bit app only.