References
Codec RSC DB (Double Binary)

RSC DB Encoder parameters
-enc-info-bits, -K
--enc-type
--enc-no-buff
--enc-std

RSC DB Decoder parameters
--dec-type, -D
--dec-implem
--dec-max

References
Codec Turbo

Turbo Encoder parameters
-enc-info-bits, -K
--enc-type
--enc-sub-type
--enc-json-path
--enc-sub-no-buff
--enc-sub-poly
--enc-sub-std

Turbo Decoder parameters
--dec-type, -D
--dec-implem
--dec-sub-type, -D
--dec-sub-implem
--dec-sub-simd
--dec-crc-start
--dec-fnc
--dec-fnc-ite-m
--dec-fnc-ite-M
--dec-fnc-ite-s
--dec-fnc-q
--dec-ite, -i
--dec-sc
--dec-sf-type
--dec-sub-max

References
Codec Turbo DB

Turbo DB Encoder parameters
-enc-info-bits, -K
--enc-type
--enc-sub-type
--enc-sub-std

Turbo DB Decoder parameters
--dec-type, -D
--dec-implem
--dec-sub-type, -D
--dec-sub-implem
--dec-crc-start
--dec-fnc
Turbo DB Puncturer parameters

Codec TPC (Turbo Product Code)

TPC Encoder parameters

TPC Decoder parameters

Codec Uncoded

Uncoded Encoder parameters

Uncoded Decoder parameters

Interleaver parameters

Modem parameters

References
4 Library Examples

4.1 Bootstrap

4.2 PyBER

References

3.7 Channel parameters

3.8 Quantizer parameters

3.9 Monitor parameters

3.10 Terminal parameters

3.11 Other parameters

3.3 PyBER

3.3.1 Install Python3

3.3.2 Run PyBER
AFF3CT (A Fast Forward Error Correction Toolbox!) is a toolbox dedicated to the Forward Error Correction (FEC or channel coding). It is written in C++ and it supports a large range of codes: from the well-spread Turbo codes to the new Polar codes including the Low-Density Parity-Check (LDPC) codes. AFF3CT includes many tools but the most important ones are:

- a standalone simulator to simulate communication chains (c.f. Fig. 1.1) based on a Monte Carlo method,
- a toolbox or a library that can be used through a well-documented API (Application Programming Interface).

The simulator targets high speed simulations and extensively uses parallel techniques like SIMD (Single Instruction Multiple Data), multi-threading and multi-nodes programming models. Below, a list of the features that motivated the creation of the simulator:

1. reproduce state-of-the-art decoding performances,
2. explore various channel code configurations, find new trade-offs,
3. prototype hardware implementation (fixed-point receivers, hardware in the loop tools),
4. reuse tried and tested modules and add yours,
5. alternative to MATLAB, if you seek to reduce simulations time.

AFF3CT was first intended to be a simulator but as it developed, the need to reuse sub-parts of the code intensified: the library was born. Below is a list of possible applications for the library:

1. build custom communication chains that are not possible with the simulator,

2. facilitate hardware prototyping,

3. enable various modules to be used in SDR (Software-Defined Radio) contexts.
2.1 Get the Source Code

**Important:** If you do not plan to modify the AFF3CT source code and you want to use the simulator/library as is, you can **download one of the latest builds** from the download page of the AFF3CT website and skip this section.

This project uses Git as the version-control system to manage the source code. The AFF3CT repository is hosted on GitHub. To get the source code, first install the Git software and secondly *clone* the AFF3CT repository locally.

### 2.1.1 Git Installation

**Windows/macOS**

Download Git from the official web page and launch the install. Just press the *Next* button until the installation is over.

**Warning:** On Windows, Git comes with the *Git Bash* terminal which is, to our mind, better suitable that the traditional *Windows Console*. We encourage you to use *Git Bash* instead of the *Windows Command Prompt* for the following steps.
Warning: It is recommended to add Git to your system PATH during the installation.
Note: On Windows, during the installation you may want to check the Linux symbolic links support.
Linux

Install Git from the package manager:

```bash
sudo apt install git
```

**Note:** On CentOS-like systems you have to replace `apt` by `yum`.

### 2.1.2 Clone AFF3CT from GitHub

Get the source code from GitHub:

```bash
git clone --recursive https://github.com/aff3ct/aff3ct.git
cd aff3ct
```

The AFF3CT repository contains some dependencies to other repositories. Technically those dependencies are managed by the [Git submodule feature](https://gitforwindows.org/). By default the submodules are not downloaded during the `git clone` process this is why the `--recursive` option has been added.

**Danger:** On the AFF3CT repository you may want to directly download the source code without making a `git clone`. This will get you an archive without the AFF3CT dependencies and the build process will fail. **Do not directly download AFF3CT from GitHub and please make a clone!**
If you want to manually get or update the AFF3CT submodules, you can use the following command:

```
git submodule update --init --recursive
```

**Warning:** When `git pull` is used to get the last commits from the repository, the submodules are not automatically updated and it is required to call the previous `git submodule` command.

## 2.2 Compilation

**Important:** If you do not plan to modify the AFF3CT source code and you want to use the simulator/library as is, you can **download one of the latest builds** from the download page of the AFF3CT website and skip this section.

This project uses **CMake** in order to generate any type of projects (Makefile, Visual Studio, Eclipse, CLion, XCode, etc.).

AFF3CT is portable and can be compiled on Windows, macOS and Linux. Of course it works on traditional x86 architectures like Intel and AMD CPUs but it also works on embedded architectures like ARM CPUs.

AFF3CT supports many C++11 compliant compilers, until now the following compilers have been tested: GNU (`g++`), Clang (`clang++`), Intel (`icpc`) and Microsoft (`MSVC`). In this section, a focus is given to compile AFF3CT with:

1. the GNU compiler on Windows and Linux (Makefile project),
2. the Microsoft compiler on Windows (Visual Studio 2017 solution),
3. the Clang compiler on macOS (Makefile project).

### 2.2.1 CMake Installation

**Windows/macOS**

Download CMake from the official web page and launch the installer. Just press the **Next** button until the installation is over.

**Important:** On Windows, if you plan to build AFF3CT from the Visual Studio IDE you can skip the CMake installation and directly go to the **Compilation with Visual Studio** section.

**Note:** On Windows, it is recommended to download a version of CMake with an installer: it looks like `cmake-x.x-x-win64-x64.msi`.

**Warning:** It is recommended to add CMake to your system `PATH` during the installation.

**Danger:** The CMake minimal version requirement is **3.0.2**.
Install Make and CMake from the package manager:

```
sudo apt install make cmake
```

**Note:** On CentOS-like systems you have to replace `apt` by `yum`.

### 2.2.2 C++ GNU Compiler Installation

**Windows**

Download the latest MinGW build from the [official web page](#) (tested with MinGW x86_64-6.2.0). Unzip the archive. Copy the extracted `mingw64` folder in the `C:\Programs\Git\` folder (overwrite all the duplicated files).

**Note:** We suppose that you have installed Git for Windows has explained in the [Git Installation on Windows](#) section and consequently you have Git Bash installed.
macOS

The instructions to install the C++ GNU compiler are not given for macOS because the native Clang compiler will be used instead in the next steps. Directly go to the *Compilation with a Makefile project on macOS* section.

Linux

Install the C++ GNU compiler from the package manager:

```
sudo apt install g++
```

**Note:** On CentOS-like systems you have to replace `apt` by `yum`.

2.2.3 Compilation with a Makefile Project

Go into the directory where you cloned AFF3CT, this directory will be refereed as `$AFF3CT_ROOT`.

Windows

Generate the Makefile from CMake:

```
mkdir build
cd build
cmake .. -G"MinGW Makefiles"
```

This last command line should fail but you can ignore it, continue with:

```
cmake .. -DCMAKE_CXX_COMPILER="g++.exe" -DCMAKE_BUILD_TYPE="Release" -DCMAKE_CXX_FLAGS="-funroll-loops -march=native"
```

Build AFF3CT with the Makefile:

```
mingw32-make -j4
```

Once finished, the AFF3CT executable should be located in the `$AFF3CT_ROOT/build/bin` folder.

**Danger:** Run the previous commands on *Git Bash* (Start Menu > Git > Git Bash) and not on the Windows Command Prompt. If you try to run the previous commands on the Windows Command Prompt, CMake will not find the GNU compiler (`g++.exe` and `gcc.exe` commands) because it has not been added to the system PATH, same for the `mingw32-make` command.

macOS

Generate the Makefile from CMake:

```
mkdir build
cd build
cmake .. -G"Unix Makefiles" -DCMAKE_CXX_COMPILER="clang++" -DCMAKE_BUILD_TYPE="Release"
-DCMAKE_CXX_FLAGS="-funroll-loops -march=native"
```

2.2. Compilation
Build AFF3CT with the Makefile:

```
make -j4
```

Once finished, the AFF3CT executable should be located in the $AFF3CT_ROOT/build/bin folder.

**Linux**

Generate the Makefile from CMake:

```
mkdir build
cd build
cmake .. -G"Unix Makefiles" -DCMAKE_CXX_COMPILER="g++" -DCMAKE_BUILD_TYPE="Release" -
-DCMAKE_CXX_FLAGS="-funroll-loops -march=native"
```

Build AFF3CT with the Makefile:

```
make -j4
```

Once finished, the AFF3CT executable should be located in the $AFF3CT_ROOT/build/bin folder.

### 2.2.4 Compilation with a Visual Studio 2017 Solution

Since Microsoft Visual Studio 2017, Visual natively supports CMake. To generate the AFF3CT solution, open the $AFF3CT_ROOT folder from the IDE.

Select the *Release* target and press the green play button `aff3ct.exe` to start the compilation.
Once AFF3CT is compiled you can browse the build by right clicking on `CMakeList.txt` > `Cache` > `Open Cache Folder`.
Note: Visual Studio should not be mixed up with Visual Studio Code. Visual Studio is the Windows native IDE and Visual Studio Code is a portable code editor.

Note: Visual Studio 2017 Community is free for Open-source contributors, students and freelance developers.
**Warning:** The Visual Studio default compiler (MSVC) is known to generate significantly slower AFF3CT executable than the GNU compiler. **If you target an high speed executable it is recommended to use the GNU or Clang compilers.**

**Danger:** When compiling AFF3CT in debug mode, the `src\Factory\Module\Decoder\Polar\Decoder_polar.cpp` file generates the following error: fatal error C1128. To fix this, you need to compile with the `/bigobj` parameter.

The compilation can also be started from the command line after calling the `%VS_PATH%\VC\Auxiliary\Build\vcvars64.bat` batch script (where `%VS_PATH%` is the location of Visual Studio on your system):

```
deenv /build Release aff3ct.sln
```

### 2.2.5 Compilation with a Visual Studio 2019 Solution

The compilation process on Visual Studio 2019 is almost the same than on Visual Studio 2017. Note that many improvements have been made on the MSVC (Microsoft Visual C++) compiler on Visual Studio 2019 and now the produced binaries are competitive with other standard compilers like GNU and Clang.

### 2.2.6 CMake Options

CMake allows to define project specific options. AFF3CT takes advantage of this feature and provides the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFF3CT_COMPILE_EXE</td>
<td>BOOLEAN</td>
<td>ON</td>
<td>Compile the executable.</td>
</tr>
<tr>
<td>AFF3CT_COMPILE_STATIC_LIB</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Compile the static library.</td>
</tr>
<tr>
<td>AFF3CT_COMPILE_SHARED_LIB</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Compile the shared library.</td>
</tr>
<tr>
<td>AFF3CT_LINK_GSL</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Link with the GSL library (used in the channels).</td>
</tr>
<tr>
<td>AFF3CT_LINK_MKL</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Link with the MKL library (used in the channels).</td>
</tr>
<tr>
<td>AFF3CT_SYSTEMC_SIMU</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Enable the SystemC simulation (incompatible with the library compilation).</td>
</tr>
<tr>
<td>AFF3CT_SYSTEMC_MODULE</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Enable the SystemC support (only for the modules).</td>
</tr>
<tr>
<td>AFF3CT_MPI</td>
<td>BOOLEAN</td>
<td>OFF</td>
<td>Enable the MPI support.</td>
</tr>
<tr>
<td>AFF3CT_Polar.Bit_Packing</td>
<td>BOOLEAN</td>
<td>ON</td>
<td>Enable the bit packing technique for Polar code SC decoding.</td>
</tr>
<tr>
<td>AFF3CT_Accounters</td>
<td>BOOLEAN</td>
<td>ON</td>
<td>Enable the colors in the terminal.</td>
</tr>
<tr>
<td>AFF3CT_Backtrace</td>
<td>BOOLEAN</td>
<td>ON</td>
<td>Enable the backtrace display when and exception is raised. On Windows and macOS this option is not available and automatically set to OFF.</td>
</tr>
<tr>
<td>AFF3CT_EXT_LINES</td>
<td>BOOLEAN</td>
<td>ON</td>
<td>Enable external strings for the help documentation. If ON the help doc will be parsed from the <code>strings.rst</code> external file. If OFF the strings will be self-contained in the binary. On MSVC this option is not available and automatically set to ON.</td>
</tr>
<tr>
<td>AFF3CT_PREC</td>
<td>STRING</td>
<td>MULTI</td>
<td>Select the precision in bits (can be ‘8’, ‘16’, ‘32’, ‘64’ or ‘MULTI’).</td>
</tr>
</tbody>
</table>

Considering an option `AFF3CT_OPTION` we want to set to `ON`, here is the syntax to follow:

```
cmake .. -DAFF3CT_OPTION="ON"
```
2.2.7 Compiler Options

Build Type

CMake allows to select the type of build through the CMAKE_BUILD_TYPE built-in variable. Release and Debug are the common values that the variable can get. For instance, to compile in release mode:

```
cmake .. -DCMAKE_BUILD_TYPE="Release"
```

Note: In CMake it is recommended to not explicitly set the compiler optimization level flags (–O0, –O1, –O2, –O3, etc.). Those compiler options will be set automatically by the CMAKE_BUILD_TYPE built-in variable. For instance, with the GNU compiler, if CMAKE_BUILD_TYPE is set to Release, the code will be compiled with the –O3 flag.

Note: If you need to develop in AFF3CT it is recommended to compile in the Debug mode (or eventually RelWithDebInfo mode) during the development process to add the debug symbols in the binary files. It will certainly ease the debug process but be careful, the execution speed will be seriously affected in this mode, be sure to switch to the Release mode when the code is stable.

Note: In Visual Studio solutions, the CMAKE_BUILD_TYPE built-in variable has no effect and the build type is directly managed by Visual.

Specific Options

CMake has a built-in variable you can set to specify the compiler options: CMAKE_CXX_FLAGS. For instance, it can be used like this:

```
cmake .. -DCMAKE_CXX_FLAGS="-funroll-loops -march=native"
```

Many parts of the AFF3CT code use the SIMD parallelism and this type of instructions often requires additional compiler options to be enabled:
### Option Description

- **-msse2**: Enable the SSE2 (Streaming SIMD Extensions 2) set of instructions on x86 CPUs (Central Process Units) (128-bit vector size, required for 32-bit and 64-bit data).

- **-msse3**: Enable the SSSE3 (Supplemental Streaming SIMD Extensions 3) set of instructions on x86 CPUs (128-bit vector size, specifically required for 32-bit data and the SC (Successive Cancellation) FAST decoder, see the `--dec-type`, `-D` and `--dec-implem` parameters).

- **-msse4**: Enable the SSE4.1 (Streaming SIMD Extensions 4.1) set of instructions on x86 CPUs (128-bit vector size, required for 8-bit and 16-bit data).

- **-mavx**: Enable the AVX (Advanced Vector Extensions) set of instructions on x86 CPUs (256-bit vector size, required for 32-bit and 64-bit data).

- **-mavx2**: Enable the AVX2 (Advanced Vector Extensions 2) set of instructions on x86 CPUs (256-bit vector size, required for 8-bit and 16-bit data).

- **-mavx512f**: Enable the AVX-512F (Advanced Vector Extensions 512-bit Foundation) set of instructions on x86 CPUs (512-bit vector size, required for 32-bit and 64-bit data).

- **-mavx512bw**: Enable the AVX-512BW (Advanced Vector Extensions 512-bit Bytes-Words) set of instructions on x86 CPUs (512-bit vector size, required for 8-bit and 16-bit data).

- **-mfpu=neon**: Enable the NEON (ARM SIMD instructions) set of instructions on ARM V7 (Advanced RISC Machine Vesion 7) and ARM V8 (Advanced RISC Machine Vesion 8) CPUs (128-bit vector size, required for 8-bit, 16-bit data and 32-bit data).

- **-march=native**: Let the compiler choose the best set of instructions available on the current architecture (it does not work for ARM V7 architectures since the NEON instruction set is not IEEE (Institute of Electrical and Electronics Engineers) 754 compliant).

---

**Warning:** Previous options are only valid for the GNU (GNU’s Not Unix!) and the Clang compilers but it exists similar options for the other compilers like the Microsoft compiler (MSVC) or the Intel compiler (icpc).

**Danger:** Some AFF3CT routines require the floating-point operations to be IEEE-compliant: numerical instabilities has been reported when compiling with the `--ffast-math` flag. Be aware that the `--ofast` option is the combination of `-O3` and `--ffast-math`. **We recommend to avoid the `--ffast-math` option unless you know what you are doing.**

---

### 2.3 Installation

In order to be installed on a system, AFF3CT can either be compiled locally and installed (see *From Source*), or remotely precompiled versions can be downloaded and installed (see *Precompiled Versions*).

#### 2.3.1 From Source

Once AFF3CT has been compiled, it is possible (not mandatory) to install it on your system. On Unix-like systems, traditionally, the fresh build is installed in the `/usr/local` directory (this is the CMake default installation path). This location can be changed by setting the `CMAKE_INSTALL_PREFIX` built-in variable with an other path. For instance, to install AFF3CT in the current build:

```
cmake .. -DCMAKE_INSTALL_PREFIX="install"
```

This command do not install AFF3CT. It only prepares the project to be installed in the selected location.
Makefile Project

To install AFF3CT, call the `install` target on the current Makefile:

```
make install
```

**Note:** Depending on the chosen `CMAKE_INSTALL_PREFIX` location, the administrator privileges (`sudo`) can be required.

Visual Studio Solution

In case of a Visual Studio Solution, an `INSTALL` project is defined and ensures the installation when triggered. This can be done from the Visual Studio IDE or from the command line after calling the `%VS_PATH%\VC\Auxiliary\Build\vcvars64.bat` batch script (where `%VS_PATH%` is the location of Visual Studio on your system):

```
devenv /build Release aff3ct.sln /project INSTALL
```

2.3.2 Precompiled Versions

From AFF3CT website

If you do not plan to modify the AFF3CT source code and you want to use the simulator/library as is, you can download one of the latest builds from the download page of the AFF3CT website. Precompiled binaries are available for the most common operating systems: Windows, macOS and Linux.

On Debian / Ubuntu

Each new version of AFF3CT is deployed on PPA (Personal Package Archive) repositories for the aptitude package manager. Two different repositories are available. The first one, `stable`, holds versions that are released after a lot of testing to ensure performance and stability. The second one, `dev`, holds the latest development versions of AFF3CT.

Select the channel to use (`stable` or `dev`, not both!):

```
# stable
sudo add-apt-repository ppa:aff3ct/aff3ct-stable

# dev
sudo add-apt-repository ppa:aff3ct/aff3ct-dev
```

Update package list and install:

```
sudo apt-get update
sudo apt-get install aff3ct aff3ct-dev aff3ct-doc
```

The package `aff3ct` contains the `bin/`, `conf/` and `refs/` folders. The package `aff3ct-dev` contains the `include/` and `lib/` folders. The package `aff3ct-doc` contains the `doc/` folder.
2.3.3 Contents

The installed package is organized as follow:

- **bin/**
  - `aff3ct-M.m.p` the AFF3CT executable binary.

- **include/**
  - `aff3ct-M.m.p/` contains all the includes required by AFF3CT.

- **lib/**
  - `libaff3ct-M.m.p.a` the AFF3CT static library.
  - `libaff3ct-M.m.p.so` the AFF3CT shared library.
  - **cmake/**
    - `aff3ct-M.m.p/` contains the CMake configuration files required to link with AFF3CT.

- **share/**
  - `aff3ct-M.m.p`
    - `conf/` contains some input files to configure the AFF3CT simulator.
    - `refs/` many results from AFF3CT simulations.
    - `doc/` contains the AFF3CT documentation.

*M* stands for the major number of the version, *m* the minor number and *p* the id of the last patch.
In this section, first an overview of the simulator capabilities is given, secondly the parameters of the simulator are describe and finally PyBER, a GUI (Graphical User Interface) tool dedicated to the display of BER/FER (Bit and Frame Error Rate) curves, is presented.

3.1 Overview

The AFF3CT toolbox comes with a simulator dedicated to the communication chains. The simulator focuses on the channel coding level. It can be used to reproduce/validate state-of-the-art BER/FER performances as well as an exploration tool to bench various configurations.

The AFF3CT simulator is based on a Monte Carlo method: the transmitter emits frames that are randomly noised by the channel and then the receiver try to decode the noised frames. The transmitter continues to emit frames until a fixed number of frame errors in achieved (typically 100 frame errors). A frame error occurs when the original frame from the transmitter differs from the the receiver decoded frame. As a consequence, when the SNR (Signal Noise Ratio) decreases, the number of frames to simulate increases as well as the simulation time.

3.1.1 Basic Arguments

The AFF3CT simulator is a command line program which can take many different arguments. The command line interface make possible to write scripts that run a battery of simulations for instance. Here is a minimalist command line using AFF3CT:

```
aff3ct -C "POLAR" -K 1723 -N 2048 -m 1.0 -M 4.0 -s 1.0
```

-\(C\) is a required parameter that defines the type of channel code that will be used in the communication chain (see the \texttt{--sim-cde-type}, \texttt{-C} section). \(-K\) is the number of information bits and \(-N\) is the frame size (bits transmitted over the channel). The Fig. 3.1 illustrates those parameters in a simplified communication chain.

The simulator computes the BER (Bit Error Rate) and the FER (Frame Error Rate) for a SNR range (by default the SNR is \(E_b/N_0\) in dB). \(-m\) is the first SNR value to simulate with and \(-M\) is the last one (see the \texttt{--sim-noise-}
C = \{\text{Polar, LDPC, Turbo, ...}\}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{communication_chain.png}
\caption{Code-related parameters in the communication chain.}
\end{figure}

...min, -m and \texttt{--sim-noise-max, -M sections for more information). \texttt{-s} is the step between each SNR (c.f. section \texttt{--sim-noise-step, -s}). The Fig. 3.2 shows the output BER for each simulated SNR values.

\begin{figure}[h]
\centering
\begin{tabular}{c|c|c}
\hline
\text{Eb/N0 dB} & \text{BER} \\
\hline
9.23x10^{-2} & 1.0 \\
7.05x10^{-2} & 2.0 \\
2.43x10^{-2} & 3.0 \\
2.93x10^{-4} & 4.0 \\
\hline
\end{tabular}
\caption{SNR-related parameters in the communication chain.}
\end{figure}

\subsection{3.1.2 Output}

The output of following command will look like:

```
aff3ct -C "POLAR" -K 1723 -N 2048 -m 1.0 -M 4.0 -s 1.0
```

(continues on next page)
All the line beginning by the # character are intended present the simulation but there are not computational results. On the top, there is the list of the simulation parameters (above the # Parameters : line). After that, the simulation results are shown, each line corresponds to the decoding performance considering a given SNR. Each line is composed by the following columns:

- **Es/N0**: the SNR expressed as $E_s/N_0$ in dB (c.f. the `--sim-noise-type, -E` section),
- **Eb/N0**: the SNR expressed as $E_b/N_0$ in dB (c.f. the `--sim-noise-type, -E` section),
- **FRA**: the number of simulated frames,
- **BE**: the number of bit errors,
- **FE**: the number of frame errors (see the `--mnt-max-fe, -e` section if you want to modify it),
- **BER**: the bit error rate ($BER = \frac{BE}{FRA \times K}$),
- **FER**: the frame error rate ($FER = \frac{FE}{FRA}$),
- **SIM_THR**: the simulation throughput ($SIM_{THR} = \frac{K \times FRA}{T}$ where $T$ is the simulation time),
- **ET/RT**: during the computation of the point, this column displays an estimation of the remaining time (RT), once the computations are done this is the total elapsed time (ET).

Note: You may notice slightly different values in BER and FER columns if you run the command line on your computer. This is because the simulation is multi-threaded by default: the order of threads execution is not predictable. If you want to have reproducible results you can launch AFF3CT in mono-threaded mode (see the `--sim-threads, -t` section).
3.1.3 Philosophy

To understand the organization of the parameters in the simulator, it is important to be aware of the simulator structure. As illustrated in the Fig. 3.3, a simulation contains a set of modules (Source, Codec, Modem, Channel and Monitor in the example). A module can contain one or more tasks. For instance, the Source module contains only one task: generate(). In contrast, the Modem module contains two tasks: modulate() and demodulate(). A task can be assimilated to a process which is executed at runtime.

Each module or task has its own set of arguments. Still, some of the arguments are common to several modules and tasks:

- `--xxx-type` is often used to define the type of each module: the type of modulation, channel or channel decoder.
- `--xxx-implem` specifies the type of implementation used. The keywords NAIVE or STD are often used to denote a readable but unoptimized source code, whereas FAST stands for a source code that is optimized for a high throughput and/or low latency.

3.2 Parameters

The AFF3CT simulator is highly customizable and contains many arguments classified in three categories:

- **Required** identified by the image, they are needed to launch a simulation.
- **Optional** set up the simulation in many ways instead of the default configuration.
- **Advanced** identified by the image, they lead to a more extensive use of the simulator.

3.2.1 Simulation parameters

The simulation parameters allow to customize the communication chain from an high level point of view. Various communication chain skeletons are available and can be selected as well as the channel code family to simulate, it is also possible to enable debug and benchmarking tools.

```
--sim-type
```

Type text
Allowed values BFEBFERIEXIT
Default BFER
Examples --sim-type BFEBFERI

Select the type of simulation (or communication chain skeleton).

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFEBFER</td>
<td>The standard BER/FER chain (Fig. 3.4).</td>
</tr>
<tr>
<td>BFEBFEBFERI</td>
<td>The iterative BER/FER chain (Fig. 3.5).</td>
</tr>
<tr>
<td>EXIT</td>
<td>The EXIT chart simulation chain (not documented at this time).</td>
</tr>
</tbody>
</table>

Fig. 3.4: The standard BER/FER chain.

--sim-cde-type, -C [REQUIRED]

Type text

Allowed values BCH LDPC POLAR RA REP RS RSC RSC_DB TURBO TURBO_DB TURBO_PROD UNCODED

Examples -C BCH

Select the channel code family to simulate.

Description of the allowed values:
Fig. 3.5: The iterative BER/FER chain.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCH</td>
<td>The Bose–Chaudhuri–Hocquenghem codes [BRC60].</td>
</tr>
<tr>
<td>LDPC</td>
<td>The Low-Density Parity-Check codes [Gal63][MN95].</td>
</tr>
<tr>
<td>POLAR</td>
<td>The Polar codes [Ari09].</td>
</tr>
<tr>
<td>RA</td>
<td>The Repeat Accumulate codes [DHJM98].</td>
</tr>
<tr>
<td>REP</td>
<td>The Repetition codes [RL09].</td>
</tr>
<tr>
<td>RS</td>
<td>The Reed-Solomon codes [RS60].</td>
</tr>
<tr>
<td>RSC</td>
<td>The Recursive Systematic Convolutional codes [RL09].</td>
</tr>
<tr>
<td>RSC_DB</td>
<td>The Recursive Systematic Convolutional codes with double binary symbols [RL09].</td>
</tr>
<tr>
<td>TURBO</td>
<td>The Turbo codes [BGT93].</td>
</tr>
<tr>
<td>TURBO_DB</td>
<td>The Turbo codes with double binary symbols [BGT93].</td>
</tr>
<tr>
<td>TURBO_PROD</td>
<td>The Turbo Product codes [RL09].</td>
</tr>
<tr>
<td>UNCODED</td>
<td>An uncoded simulation.</td>
</tr>
</tbody>
</table>

Note: Only POLAR, RSC, RSC_DB, LDPC and UNCODED codes are available in BFERI simulation type.

---sim-prec, -p

**Type**  integer

**Default**  32

**Allowed values**  8 16 32 64

**Examples**  --sim-prec 8

Specify the representation of the real numbers in the receiver part of the chain.

64-bit and 32-bit precisions imply a floating-point representation of the real numbers. 16-bit and 8-bit imply a fixed-point representation of the real numbers (see the Quantizer parameters to configure the quantization).
Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8-bit precision.</td>
</tr>
<tr>
<td>16</td>
<td>16-bit precision.</td>
</tr>
<tr>
<td>32</td>
<td>32-bit precision.</td>
</tr>
<tr>
<td>64</td>
<td>64-bit precision.</td>
</tr>
</tbody>
</table>

`--sim-noise-type, -E`

**Type** text

**Allowed values** EBN0 ESN0 EP ROP

**Default** EBN0

**Examples** -E EBN0

Select the type of noise used to simulate.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBN0</td>
<td>SNR per information bit</td>
</tr>
<tr>
<td>ESN0</td>
<td>SNR per transmitted symbol</td>
</tr>
<tr>
<td>EP</td>
<td>Event Probability</td>
</tr>
<tr>
<td>ROP</td>
<td>Received Optical Power</td>
</tr>
</tbody>
</table>

ESN0 is automatically calculated from EBN0 and vice-versa with the following equation:

\[
\frac{E_s}{N_0} = \frac{E_b}{N_0} + 10 \log(R.bps),
\]

where \(R\) is the bit rate and \(bps\) the number of bits per symbol.

**Note:** When selecting EP the simulator runs in reverse order, i.e., from the greatest event probability to the smallest one.

**Hint:** When selecting a BEC or BSC channel the EP noise type is automatically set except if you give another one. This is the same for the OPTICAL channel with the ROP noise type. The channel type is set with the `--chn-type` argument.

`--sim-noise-min, -m`

**Type** real number

**Examples** -m 0.0

Set the minimal noise energy value to simulate.

**Attention:** This argument is another way to set the noise range to simulate. It is ignored or not required if the `--sim-noise-range, -R` argument is given, so you may find other piece of information in its description.
--sim-noise-max, -M

Type  real number

Examples  -M 5.0

Set the maximal noise energy value to simulate.

**Attention:** This argument is another way to set the noise range to simulate. It is ignored or not required if the --sim-noise-range, -R argument is given, so you may find other piece of information in its description.

--sim-noise-step, -s

Type  real number

Default  0.1

Examples  -s 1.0

Set the noise energy step between each simulation iteration.

**Attention:** This argument is another way to set the noise range to simulate. It is ignored or not required if the --sim-noise-range, -R argument is given, so you may find other piece of information in its description.

--sim-noise-range, -R

Type  MATLAB style vector

Default  step of 0.1

Examples  -R "0.5:1,1:0.05:1.2,1.21"

Set the noise energy range to run in a MATLAB style vector.

The above example will run the following noise points:

0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.05, 1.1, 1.15, 1.2, 1.21

**Attention:** The numerical range for a noise point is \([-214.748; 213.952]\) with a precision of \(10^{-7}\).

**Note:** If given, the --sim-noise-min, -m, --sim-noise-max, -M, and --sim-noise-step, -s parameters are ignored. But it is not required anymore if --sim-noise-min, -m and --sim-noise-max, -M are set.

--sim-pdf-path

Type  file

Rights  read only
Examples --sim-pdf-path example/path/to/the/right/file

Give a file that contains PDF (Probability Density Function) for different ROP (Received Optical Power).

To use with the **OPTICAL channel**. It sets the noise range from the given ones in the file. However, it is overwritten by --sim-noise-range, -R or limited by --sim-noise-min, -m and --sim-noise-max, -M with a minimum step of --sim-noise-step, -s between two values.

**--sim-meta**

**Type** text

**Examples** --sim-meta "TITLE"

Add meta-data at the beginning of the AFF3CT standard output (INI format is used). The value of the parameter will be affected to the *title* meta-data and the *command line* will be added.

**Note:** *PyBER*, our GUI tool, can take advantage of those meta-data to enhance the display of the simulated curves.

**--sim-coded**

Enable the coded monitoring.

By default, in the simulation, the information bits are extracted from the decoded codewords and then they are compared to the initially generated information bits. When this parameter is enabled, the decoded codewords are directly compared with the initially encoded codewords.

**Note:** This parameter can have a negative impact on the BER performance.

**Note:** In some rare cases, to enable this parameter can reduce the simulation time.

**--sim-coset, -c**

Enable the *coset* approach.

The *coset* approach is a “trick” to simulate BER/FER performance **without an encoder**. It is based on the AZCW (All Zero Code Word) technique (see the --src-type parameter). In the specific case of modulation with memory effect, AZCWs (All Zero Code Words) can lead to erroneous BER/FER performance. The *coset* approach solves this problem by randomly generating N bits, those bits represent a frame but there are certainly not a codeword. Then, those random bits (or symbols) can be modulated avoiding AZCW unexpected effects. On the receiver side, the idea is to force the decoder to work on an AZCW (because the N received LLRs (Log Likelihood Ratios) are not a valid codeword). Before the decoding process, knowing the initial bits sequence, when a bit is 1 then the corresponding input LLR (Log Likelihood Ratio) is replaced by its opposite. This way, the decoder “believe” it is decoding an AZCW which is a valid codeword. After the decoding process, knowing the initial bits sequence, when a bit is 1, then the corresponding output bit is flipped.

**--sim-db**

Enable the debug mode. This print the input and the output frames after each task execution.

### 3.2. Parameters
Note: By default, the debug mode runs the simulation on one thread. It is strongly advise to remove the \texttt{-sim-threads, -r} parameter from your command line if you use it.

**Hint:** To limit the size of the debug trace, use the \texttt{-mnt-max-fe, -e} or \texttt{-sim-max-fra, -n} parameters to reduce the number of simulated frames. You may also think about using \texttt{-sim-dbg-limit, -d} when the frame size is too long to be display on a screen line.

\textbf{--sim-dbg-hex}

Enable the debug mode and print values in the hexadecimal format. This mode is useful for having a fully accurate representation of floating numbers.
---sim-dbg-limit, -d

**Type** integer

**Default** 0

**Examples** --sim-dbg-limit 1

Enable the debug mode and set the max number of elements to display per frame. 0 value means there is no dump limit.

```
aff3ct -C "REP" -K 4 -N 8 -m 1.0 -M 1.0 --sim-dbg-limit 3
# [...]  
```

---sim-dbg-fra

**Type** integer

**Default** 0

**Examples** --sim-dbg-fra 10

Enable the debug mode and set the max number of frames to display. 0 value means there is no frame limit. By default, a task works on one frame at a time. This behavior can be overridden with the --src-fra, -F parameter and a task can be executed on many frames. In that case, you may want to reduce the number of displayed frames on screen:

```
aff3ct -C "REP" -K 4 -N 8 -m 1.0 -M 1.0 -F 8 --sim-dbg-fra 4
# [...]  
```
# (continued from previous page)

```
# (OUT) X_N2 = [f1(-1.00, -1.00, 1.00, -1.00, -1.00, -1.00, 1.00, -1.00),
# f2(1.00, -1.00, -1.00, 1.00, 1.00, -1.00, -1.00, 1.00),
# f3(-1.00, 1.00, -1.00, -1.00, -1.00, 1.00, -1.00, -1.00),
# f4(-1.00, 1.00, 1.00, 1.00, -1.00, 1.00, 1.00, 1.00),
# f5->f8:(...)]
# Returned status: 0
#
# Channel_AWGN_LLR::add_noise(const float32 X_N[8x8], float32 Y_N[8x8])
# (IN) X_N = [f1(-1.00, -1.00, 1.00, -1.00, -1.00, -1.00, 1.00, -1.00),
# f2(1.00, -1.00, -1.00, 1.00, 1.00, -1.00, -1.00, 1.00),
# f3(-1.00, 1.00, -1.00, -1.00, -1.00, 1.00, -1.00, -1.00),
# f4(-1.00, 1.00, 1.00, 1.00, -1.00, 1.00, 1.00, 1.00),
# f5->f8:(...)]
# (OUT) Y_N = [f1(-0.29, -0.24, 1.55, -0.58, -0.33, -0.51, 0.80, -2.88),
# f2(0.15, -0.71, -1.85, 1.69, -0.02, -0.50, 0.07, 0.79),
# f3(-1.03, 1.39, -1.03, -2.03, -0.67, 0.91, -0.45, -0.88),
# f4(-0.37, -1.07, 1.49, 0.94, -0.21, 1.35, 1.06, 0.97),
# f5->f8:(...)]
# Returned status: 0
# [...]```

```
--sim-dbg-prec

Type integer
Default 2
Examples --sim-dbg-prec 1

Enable the debug mode and set the decimal precision (number of digits for the decimal part) of the floating-point elements.
```

```bash
aff3ct -C "REP" -K 4 -N 8 -m 1.0 -M 1.0 --sim-dbg-prec 4
# [...]```

```
--sim-seed, -S

Type integer
Default 0
Examples --sim-seed 42
```
Set the PRNG (Pseudo Random Number Generator) seed used in the Monte Carlo simulation.

**Note:** AFF3CT uses PRNG to simulate the random. As a consequence the simulator behavior is reproducible from a run to another. It can be helpful to debug source code. However, when simulating in multi-threaded mode, the threads running order is not deterministic and so results will most likely be different from one execution to another.

```bash
aff3ct -C "POLAR" -K 1723 -N 2048 -m 4.2 -M 4.2 -t 1 --sim-stats
```

Each line corresponds to a task. The tasks are sorted by execution time in the simulation (descending order). The first column group identifies the task:

- **MODULE**: the module type,
- **TASK**: the task name,
- **TIMER**: the name of the current task timer (it is possible to put timers inside a task to measure sub-parts of this task), * indicates that the whole task execution time is considered.

```
# | MODULE | TASK | TIMER | CALLS | TIME | PERC | AVERAGE | MINIMUM | MAXIMUM | AVERAGE | MINIMUM | MAXIMUM |
# | | | | | (s) | (%) | (Mb/s) | (Mb/s) | (us) | (us) | (us)
---|---------|--------|--------|-------|-------|------|--------|--------|------|-------|--------|-------|
# Channel | add_noise | * | 14909 | 0.72 | 37.59 | 42.
# Source | generate | * | 14909 | 0.60 | 31.13 | 42.
# Encoder | encode | * | 14909 | 0.37 | 19.06 | 83.
# Decoder | decode_siho | * | 14909 | 0.22 | 11.32 | 117.
# Monitor | check_errors | * | 14909 | 0.01 | 0.42 | 3186.
# Modem | demodulate | * | 14909 | 0.00 | 0.25 | 6350.
# Modem | modulate | * | 14909 | 0.00 | 0.23 | 6962.
---|-------|-------|-------|-------|-------|------|--------|--------|------|-------|--------|-------|
# TOTAL | | | | | 14909 | 1.93 | 100.00 | 13.
# | 3.38 | 14.10 | 129.19 | 122.21 | 509.42
# [...]```

3.2. Parameters
The second column group gives basic statistics:

- **CALLS**: the number of times this task has been executed,
- **TIME**: the cumulative time of all the task executions,
- **PERC**: the percentage of time taken by the task in the simulation.

The third column group shows the **average, the minimum and the maximum throughputs**. Those throughputs are calculated considering the size of the output frames. If the task does not have outputs (c.f the `check_errors` routine from the monitor) then the number of input bits is used instead. For instance, the `encode` task takes $K$ input bits and produces $N$ output bits, so $N$ bits will be considered in the throughput computations.

The last column group shows the **average, the minimum and the maximum latencies**.

The **TOTAL** line corresponds to the full communication chain. For the throughput computations, the last socket of the last task determines the number of considered bits. In a standard BER/FER simulation the last task is the `check_errors` routine from the monitor and consequently the number of information bits ($K$) is considered. However, if the `-sim-coded` parameter is enabled, it becomes the codeword size ($N$).

**Note:** Enabling the statistics can increase the simulation time due to the measures overhead. This is especially true when short frames are simulated.

**Warning:** In multi-threaded mode the reported time is the cumulative time of all the threads. This time is bigger than the real simulation time because it does not consider that many tasks have been executed in parallel.

**Warning:** The task throughputs will not increase with the number of threads: the statistics consider the performance on one thread.

`--sim-threads, -t`

Type: integer

Default: 0

Examples: `--sim-threads 1`

Specify the number of threads used in the simulation. The 0 default value will automatically set the number of threads to the hardware number of threads available on the machine.

**Note:** Monte Carlo methods are known to be embarrassingly parallel, which is why, in most cases, the simulation throughput will increase linearly with the number of threads (unless this number exceeds the number of cores available). However, in some cases, especially for large frames, when the number of threads is high, the memory footprint can exceed the size of the CPU caches and it becomes less interesting to use a large number of threads.

`--sim-crc-start`

Type: integer

Default: 2

Examples: `--sim-crc-start 1`
Set the number of simulation iterations to proceed before starting the CRC (Cyclic Redundancy Check) checking in the turbo demodulation process. It reduces the number of false positive CRC detections.

**Note:** Available only for BFERI simulation type (c.f. the `--sim-type` parameter).

`--sim-ite, -I`

*Type* integer
*Default* 15
*Examples* `--sim-ite 10`

Set the number of global iterations between the demodulator and the decoder.

**Note:** Available only for BFERI simulation type (c.f. the `--sim-type` parameter).

`--sim-max-fra, -n` [ADVANCED]

*Type* integer
*Default* 0
*Examples* `--sim-max-fra 1`

Set the maximum number of frames to simulate per noise point. When a noise point reaches the maximum frame limit, the simulation is stopped. 0 value means no limit.

**Note:** The default behavior is to stop the simulator when the limit is reached but it is possible to override it with the `--sim-crit-nostop` parameter. In this case, the remaining noise points will also be simulated and the limit will be applied for each of them.

`--sim-stop-time` [ADVANCED]

*Type* integer
*Default* 0
*Examples* `--sim-stop-time 1`

Set the maximum time (in seconds) to simulate per noise point. When a noise point reaches the maximum time limit, the simulation is stopped. 0 value means no limit.

**Note:** The default behavior is to stop the simulator when the limit is reached but it is possible to override it with the `--sim-crit-nostop` parameter. In this case, the remaining noise points will also be simulated and the limit will be applied for each of them.
**--sim-crit-nostop**  
Stop only the current noise point instead of the whole simulation.
To combine with the `--sim-max-fra`, `-n` and/or the `--sim-stop-time` parameters.

**--sim-err-trk**  
Track the erroneous frames. When an error is found, the information bits from the source, the codeword from the encoder and the applied noise from the channel are dumped in several files.

**Tip:** When working on the design of a new decoder or when improving an existing one, it can be very interesting to have a database of erroneous frames to work on (especially if those errors occur at low BER/FER when the simulation time is important). This way it is possible to focus only on those erroneous frames and quickly see if the decoder improvements have an impact on them. It is also interesting to be able to easily extract the erroneous frames from the simulator to characterize the type of errors and better understand why the decoder fails.

**Note:** See the `--sim-err-trk-rev` argument to replay the erroneous dumped frames.

**--sim-err-trk-rev**  
Replay dumped frames. By default this option reverts the `--sim-err-trk` parameter by replaying the erroneous frames that have been dumped.

**Tip:** To play back the erroneous frames, just add `--rev` to the `--sim-err-trk` argument and change nothing else to your command line.

**--sim-err-trk-path**  

Type: file  
Rights: read/write  
Default: error_tracker  
Examples: `--sim-err-trk-path errors/err`  

Specify the base path for the `--sim-err-trk` and `--sim-err-trk-rev` parameters.
For the above example, the dumped or read files will be:

- `errors/err_0.64.src`
- `errors/err_0.64.enc`
- `errors/err_0.64.chn`
Note: For SNR noise type, the value used to define the filename is the noise variance $\sigma$.

Danger: Be careful, if you give a wrong path you will not have a warning message at the beginning of the simulation. It can be frustrating when running a very long simulation...

--sim-err-trk-thold

Type integer
Default 0

Examples --sim-err-trk-thold 1

Specify a threshold value in number of erroneous bits before which a frame is dumped.

References

3.2.2 Source parameters

The source generates $K$ information bits: it is the simulation starting point.

--src-info-bits, -K

Type integer

Examples --src-info-bits 64 -K 128

Select the number of information bits $K$.

Warning: This argument is required only with the UNCODED simulation code type (cf. the --sim-cde-type, -C parameter).

--src-type

Type text

Allowed values AZCW RAND USER
Default RAND

Examples --src-type AZCW

Method used to generate the $K$ information bits.

Description of the allowed values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZCW</td>
<td>Set all the information bits to 0.</td>
</tr>
<tr>
<td>RAND</td>
<td>Generate randomly the information bits based on the MT 19937 (Mersenne Twister 19937) PRNG [MN98].</td>
</tr>
<tr>
<td>USER</td>
<td>Read the information bits from a given file, the path can be set with the --src-path parameter.</td>
</tr>
</tbody>
</table>

Note: For the USER type, when the number of simulated frames exceeds the number of frames contained in the files, the frames are replayed from the beginning of the file and this is repeated until the end of the simulation.

**--src-implem**

Type text

Allowed values FAST STD

Default STD

Examples --src-implem FAST

Select the implementation of the algorithm to generate the information bits.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard implementation working for any source type.</td>
</tr>
<tr>
<td>FAST</td>
<td>Fast implementation, only available for the RAND source type.</td>
</tr>
</tbody>
</table>

**--src-fra, -F**

Type integer

Default 1

Examples --src-fra 8

Set the number of frames to process for each task execution.

The default behavior is to generate one frame at a time. This parameter enables to process more than one frame when the generate task (from the source module) is called.

The number of frames consumed and produced when a task is executed is called the inter frame level or IFL (Inter Frame Level). Setting the IFL in the source module will automatically affect the IFL level in all the other simulation modules (c.f. Fig. 3.6).

The IFL also allows multi-user configurations to be simulated (see Fig. 3.7). This configurations is used when using SCMA (Sparse Code Multiple Access) modulation (see the --mdm-type SCMA parameter).

Note: For short frames, increase the IFL can increase the simulation throughput, it can hide task call overheads.

Note: For large frames, increase the IFL can decrease the simulation throughput due the CPU cache size limitation.
Fig. 3.6: 3-way inter frame level in the communication chain.

Fig. 3.7: 3-way inter frame level with multi-user channel in the communication chain.
**--src-path**

Type: file  
Rights: read only  
Examples: `--src-path conf/src/GSM-LDPC_2112.src`

Set the path to a file containing one or more frames (information bits), to use with the USER source type. An ASCII (American Standard Code for Information Interchange) file is expected:

```
# 'F' has to be replaced by the number of contained frames.
F

# 'K' has to be replaced by the number of information bits.
K

# a sequence of 'F * K' bits (separated by spaces)
B_0 B_1 B_2 B_3 B_4 B_5 […] B_{(F*K)-1}
```

**--src-start-idx**

Type: integer  
Default: 0  
Examples: `--src-start-idx 42`

Give the start index to use in the USER source type. It is the index of the first frame to read from the given file.

**References**

### 3.2.3 CRC parameters

The following parameters concern the Cyclic Redundancy Check (CRC) module. CRC bits can be concatenated to the information bits in order to help the decoding process to know if the decoded bit sequence is valid or not.

**Note:** The CRC is only available for some specific decoders that have been designed to take advantage of the CRC like in [LST12][TLLeGal+16].

**Warning:** Using a CRC does not guarantee to know if the decoded frame is the good one, it can be a false positive. It is important to adapt the size of the CRC with the frame size and the targeted FER.

**--crc-type, --crc-poly**

Type: text  
Default: NO  
Examples: `--crc-type "32-GZIP"`  
`--crc-poly "0x04C11DB7" --crc-size 32`
Select the CRC type you want to use among the predefined (or not) polynomials.

Table 3.1 shows a list of the predefined polynomials. If you want a specific polynomial that it is not available in the table you can directly put the polynomial in hexadecimal. In this case you have to specify explicitly the size of the polynomial with the \texttt{-crc-size} parameter. The type \texttt{NO} deactivates the CRC.

Table 3.1: List of the predefined CRC polynomials.

<table>
<thead>
<tr>
<th>Type</th>
<th>Polynomial</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-GZIP</td>
<td>0x04C11DB7</td>
<td>32</td>
</tr>
<tr>
<td>32-CASTAGNOLI</td>
<td>0x1EDC6F41</td>
<td>32</td>
</tr>
<tr>
<td>32-AIXM</td>
<td>0x814141AB</td>
<td>32</td>
</tr>
<tr>
<td>32-KOOPMAN</td>
<td>0x32583499</td>
<td>32</td>
</tr>
<tr>
<td>30-CDMA</td>
<td>0x2030B9C7</td>
<td>30</td>
</tr>
<tr>
<td>24-LTEA</td>
<td>0x864CFB</td>
<td>24</td>
</tr>
<tr>
<td>24-RADIX-64</td>
<td>0x864CFB</td>
<td>24</td>
</tr>
<tr>
<td>24-FLEXRAY</td>
<td>0x5D6DCB</td>
<td>24</td>
</tr>
<tr>
<td>24-5GA</td>
<td>0x864CFB</td>
<td>24</td>
</tr>
<tr>
<td>24-5GB</td>
<td>0x800063</td>
<td>24</td>
</tr>
<tr>
<td>24-5GC</td>
<td>0xB2B117</td>
<td>24</td>
</tr>
<tr>
<td>21-CAN</td>
<td>0x102899</td>
<td>21</td>
</tr>
<tr>
<td>17-CAN</td>
<td>0x1685B</td>
<td>17</td>
</tr>
<tr>
<td>16-IBM</td>
<td>0x8005</td>
<td>16</td>
</tr>
<tr>
<td>16-CCITT</td>
<td>0x1021</td>
<td>16</td>
</tr>
<tr>
<td>16-PROFIBUS</td>
<td>0x1DCF</td>
<td>16</td>
</tr>
<tr>
<td>16-OPENSAFETY-B</td>
<td>0x755B</td>
<td>16</td>
</tr>
<tr>
<td>16-OPENSAFETY-A</td>
<td>0x5935</td>
<td>16</td>
</tr>
<tr>
<td>16-DNP</td>
<td>0x3D65</td>
<td>16</td>
</tr>
<tr>
<td>16-T10-DIF</td>
<td>0x8BB7</td>
<td>16</td>
</tr>
<tr>
<td>16-DECT</td>
<td>0x0589</td>
<td>16</td>
</tr>
<tr>
<td>16-CDMA2000</td>
<td>0xC867</td>
<td>16</td>
</tr>
<tr>
<td>16-ARINC</td>
<td>0xA02B</td>
<td>16</td>
</tr>
<tr>
<td>16-CHAKRAVARTY</td>
<td>0x2F15</td>
<td>16</td>
</tr>
<tr>
<td>16-5G</td>
<td>0x1023</td>
<td>16</td>
</tr>
<tr>
<td>15-MPT1327</td>
<td>0x6815</td>
<td>15</td>
</tr>
<tr>
<td>15-CAN</td>
<td>0x4599</td>
<td>15</td>
</tr>
<tr>
<td>14-DARC</td>
<td>0x0805</td>
<td>14</td>
</tr>
<tr>
<td>13-BBC</td>
<td>0x1CF5</td>
<td>13</td>
</tr>
<tr>
<td>12-CDMA2000</td>
<td>0xF13</td>
<td>12</td>
</tr>
<tr>
<td>12-TELECOM</td>
<td>0x80F</td>
<td>12</td>
</tr>
<tr>
<td>11-FLEXRAY</td>
<td>0x385</td>
<td>11</td>
</tr>
<tr>
<td>11-5G</td>
<td>0x621</td>
<td>11</td>
</tr>
<tr>
<td>10-CDMA2000</td>
<td>0x3D9</td>
<td>10</td>
</tr>
<tr>
<td>10-ATM</td>
<td>0x233</td>
<td>10</td>
</tr>
<tr>
<td>8-WCDMA</td>
<td>0x9B</td>
<td>8</td>
</tr>
<tr>
<td>8-SAE-J1850</td>
<td>0x1D</td>
<td>8</td>
</tr>
<tr>
<td>8-DARC</td>
<td>0x39</td>
<td>8</td>
</tr>
<tr>
<td>8-DALLAS</td>
<td>0x31</td>
<td>8</td>
</tr>
<tr>
<td>8-CCTT</td>
<td>0x07</td>
<td>8</td>
</tr>
<tr>
<td>8-AUTOSAR</td>
<td>0x2F</td>
<td>8</td>
</tr>
<tr>
<td>8-DVB-S2</td>
<td>0xD5</td>
<td>8</td>
</tr>
<tr>
<td>7-MVB</td>
<td>0x6D</td>
<td>7</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.1 – continued from previous page

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-MMC</td>
<td>0x09</td>
<td>7</td>
</tr>
<tr>
<td>6-CDMA2000-A</td>
<td>0x27</td>
<td>6</td>
</tr>
<tr>
<td>6-CDMA2000-B</td>
<td>0x07</td>
<td>6</td>
</tr>
<tr>
<td>6-DARC</td>
<td>0x19</td>
<td>6</td>
</tr>
<tr>
<td>6-ITU</td>
<td>0x03</td>
<td>6</td>
</tr>
<tr>
<td>5-ITU</td>
<td>0x15</td>
<td>5</td>
</tr>
<tr>
<td>5-EPC</td>
<td>0x09</td>
<td>5</td>
</tr>
<tr>
<td>5-USB</td>
<td>0x05</td>
<td>5</td>
</tr>
<tr>
<td>4-ITU</td>
<td>0x03</td>
<td>4</td>
</tr>
<tr>
<td>1-PAR</td>
<td>0x1</td>
<td>1</td>
</tr>
</tbody>
</table>

--crc-size

**Type** integer

**Range** $[0 \rightarrow \infty[$

**Examples** --crc-size 8

Size the CRC (divisor size in bits minus one), required if you selected an unknown CRC.

--crc-implem

**Type** text

**Allowed values** STD FAST INTER

**Default** FAST

**Examples** --crc-implem FAST

Select the CRC implementation you want to use.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>The standard implementation is generic and support any size of CRCs (Cyclic Redundancy Checks). On the other hand the throughput is limited.</td>
</tr>
<tr>
<td>FAST</td>
<td>This implementation is much faster than the standard one. This speedup is achieved thanks to the bit packing technique: up to 32 bits can be computed in parallel. This implementation does not support polynomials higher than 32 bits.</td>
</tr>
<tr>
<td>INTER</td>
<td>The inter-frame implementation should not be used in general cases. It allow to compute the CRC on many frames in parallel that have been reordered.</td>
</tr>
</tbody>
</table>

References

3.2.4 Codec parameters

Codec Common

Common Encoder parameters

This section describes the parameters common to all encoders.
**--enc-type**

**Type** text  
**Allowed values** NO AZCW COSET USER  
**Examples** --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZCW</td>
<td>Select the AZCW encoder which is optimized to encode ( K ) information bits all set to 0.</td>
</tr>
<tr>
<td>COSET</td>
<td>Select the coset encoder (see the (--sim-coset, -c) parameter), this encoder add random bits from ( X_K ) to ( X_N ).</td>
</tr>
<tr>
<td>USER</td>
<td>Read the codewords from a given file, the path can be set with the (--enc-path) parameter.</td>
</tr>
</tbody>
</table>

**Tip:** The AZCW encoder allows to have a working communication chain without implementing an encoder. This technique can also reduce the simulation time especially when the *encode* task is time consuming.

**Danger:** Be careful, the AZCW technique can lead to unexpected behaviors with broken decoders.

**Note:** Only use the COSET encoder if know what you are doing. This encoder is set by default when the simulation is run with the \(--sim-coset, -c\) parameter.

**Note:** For the USER type, when the number of simulated frames exceeds the number of codewords contained in the files, the codewords are replayed from the beginning of the file and this is repeated until the end of the simulation.

**--enc-path**

**Type** file  
**Rights** read only  
**Examples** --enc-path example/path/to/the/right/file

Set the path to a file containing one or more codewords, to use with the USER encoder.

An ASCII file is expected:

```plaintext
# 'F' has to be replaced by the number of contained frames.
F

# 'N' has to be replaced by the codeword size.
N

# a sequence of 'F * N' bits (separated by spaces)
B_0 B_1 B_2 B_3 B_4 B_5 [...] B_(F*N)-1
```

---

3.2. Parameters
--enc-start-idx

Type integer

Examples --enc-start-idx 1

Give the start index to use in the USER encoder. It is the index of the first codeword to read from the given file.

Common Decoder parameters

This section describes the parameters common to all decoders.

--dec-type, -D

Type text

Allowed values CHASE ML

Examples --dec-type ML

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHASE</td>
<td>Select the Chase decoder from [Cha72].</td>
</tr>
<tr>
<td>ML</td>
<td>Select the perfect ML (Maximum Likelihood) decoder.</td>
</tr>
</tbody>
</table>

Note: The Chase and the ML decoders have a very high computational complexity and cannot be used for large frames.

--dec-implem

Type text

Allowed values NAIVE STD

Examples --dec-implem STD

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAIVE</td>
<td>Select the naive implementation (very slow and only available for the ML decoder).</td>
</tr>
<tr>
<td>STD</td>
<td>Select the standard implementation.</td>
</tr>
</tbody>
</table>

--dec-flips

Type integer

Examples --dec-flips 1
Set the maximum number of bit flips in the decoding algorithm.

**Note:** Used in the Chase decoding algorithm.

---

**--dec-hamming**

Compute the Hamming distance instead of the Euclidean distance in the ML and Chase decoders.

**Note:** Using the Hamming distance will heavily degrade the BER/FER performances. The BER/FER performances will be the same as an hard input decoder.

---

**--dec-seed**

**Type** integer

**Examples** --dec-seed 1

Specify the decoder PRNG seed (if the decoder uses one).

---

**References**

**Codec BCH (Bose, Ray-Chaudhuri and Hocquenghem)**

**BCH Encoder parameters**

---

**--enc-cw-size, -N**

**Type** integer

**Examples** --enc-cw-size 127

Set the codeword size $N$.

$N = 2^m - 1$, where $m$ is an integer from 3.

---

**--enc-info-bits, -K**

**Type** integer

**Examples** --enc-info-bits 92

Set the number of information bits $K$.

This argument is not required if --dec-corr-pow, -T is given, as it is calculated automatically.
---enc-type

Type text

Allowed values BCH AZCW COSET USER

Default BCH

Examples --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCH</td>
<td>Select the standard BCH encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common –enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common –enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common –enc-type parameter.</td>
</tr>
</tbody>
</table>

BCH Decoder parameters

--dec-type, -D

Type text

Allowed values ALGEBRAIC CHASE ML

Default ALGEBRAIC

Examples --dec-type ALGEBRAIC

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGEBRAIC</td>
<td>Select the Berlekamp-Massey algorithm [Ber15][Mas69] followed by a Chien search [Chi64].</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common –dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common –dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text

Allowed values FAST GENIUS STD

Default STD

Examples --dec-implem FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>A standard implementation of the BCH.</td>
</tr>
<tr>
<td>FAST</td>
<td>Select the fast implementation optimized for SIMD architectures.</td>
</tr>
<tr>
<td>GENIUS</td>
<td>A really fast implementation that compare the input to the original codeword and correct it only when the number of errors is less or equal to the BCH correction power.</td>
</tr>
</tbody>
</table>

**Note:** In the STD implementation, the Chien search finds roots of the error location polynomial. If the number of found roots does not match the number of found errors by the BM (Berlekamp-Massey) algorithm, then the frame is not modified.

However, in the FAST implementation the correction of the bits is done at the same time as the execution of the Chien search. Then when the latter fails, the frame can be modified.

**Note:** When a frame is very corrupted and when the above STD and FAST implementations can be wrong in the correction by converging to another codeword, the GENIUS implementation cannot fail. Results may then differ from a real word implementation.

```
--dec-corr-pow, -T
```

**Type** integer

**Default** 5

**Examples** `--dec-corr-pow 18`

Set the correction power of the BCH decoder. This value corresponds to the number of errors that the decoder is able to correct.

**References**

**Codec LDPC (Low-Density Parity-Check)**

**LDPC Encoder parameters**

```
--enc-cw-size, -N
```

**Type** integer

**Examples** `--enc-cw-size 1024`

Set the codeword size $N$.

**Note:** This parameter value is automatically deduced if the $H$ parity matrix is given with the `--dec-h-path` parameter or if the $G$ generator matrix is given with the `--enc-g-path` parameter.
---enc-info-bits, -K

**Type** integer

**Examples** --enc-info-bits 512

Set the number of information bits $K$.

**Note:** This parameter value is automatically deduced if the $G$ generator matrix is given with the --enc-g-path parameter.

**Note:** In some cases, this parameter value can be automatically deduced if the $H$ parity matrix is given with the --dec-h-path parameter. For regular matrices, $K = N - M$ where $N$ and $M$ are the $H$ parity matrix dimensions. For non-regular matrices, $K$ has to be given.

---enc-type

**Type** text

**Allowed values** LDPC LDPC_H LDPC_DVBS2 LDPC_IRA LDPC_QC AZCW COSET USER

**Default** AZCW

**Examples** --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPC</td>
<td>Select the generic encoder that encode from a given $G$ generator matrix (to use with the --enc-g-path parameter).</td>
</tr>
<tr>
<td>LDPC_H</td>
<td>Build the $G$ generator matrix from the given $H$ parity matrix and then encode with the LDPC method (to use with the --dec-h-path parameter).</td>
</tr>
<tr>
<td>LDPC_DVBS2</td>
<td>Select the optimized encoding process for the DVB-S2 (Digital Video Broadcasting - Satellite 2) $H$ matrices (to use with the --enc-cw-size, -N and --enc-info-bits, -K parameters).</td>
</tr>
<tr>
<td>LDPC_IRA</td>
<td>Select the optimized encoding process for the IRA (Irregular Repeat Accumulate) $H$ parity matrices (to use with the --dec-h-path parameter).</td>
</tr>
<tr>
<td>LDPC_QC</td>
<td>Select the optimized encoding process for the QC (Quasi-Cyclic) $H$ parity matrices (to use with the --dec-h-path parameter).</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

**Note:** The LDPC_DVBS2 encoder type allow the simulation of the DVB-S2 standard but without the BCH code. All matrices described by the standard (Tables 5a/5b page 22-23) are available. You just need to give to the arguments --enc-info-bits, -K and --enc-cw-size, -N the real $K$ and $N$ LDPC dimensions, respectively.
--enc-g-path

Type file
Rights read only
Examples --enc-g-path example/path/to/the/G_matrix.alist

Give the path to the $G$ generator matrix in an AList or QC formatted file.

--enc-g-method

Type text
Allowed values IDENTITY LU_DEC
Default IDENTITY
Examples --enc-g-method IDENTITY

Specify the method used to build the $G$ generator matrix from the $H$ parity matrix when using the LDPC_H encoder.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTITY</td>
<td>Generate an identity on $H$ to get the parity part.</td>
</tr>
<tr>
<td>LU_DEC</td>
<td>Generate a hollow $G$ thanks to the LU decomposition with a guarantee to have the systematic identity. Do not work with irregular matrices.</td>
</tr>
</tbody>
</table>

LU_DEC method is faster than IDENTITY.

--enc-g-save-path

Type file
Rights write only
Examples --enc-g-save-path example/path/to/the/generated/G_matrix.alist

Set the file path where the $G$ generator matrix will be saved (AList file format). To use with the LDPC_H encoder.

Hint: When running the LDPC_H encoder, the generation of the $G$ matrix can take a non-negligible part of the simulation time. With this option the $G$ matrix can be saved once for all and used in the standard LDPC decoder after.

3.2. Parameters
Examples --dec-h-path conf/dec/LDPC/AR4JA_4096_8192.qc
--dec-h-path conf/dec/LDPC/MACKAY_504_1008.alist

Give the path to the $H$ parity matrix. Support the AList and the QC formats.

This argument is not required if the encoder type --enc-type is LDPC_DVBS2.

For the AList format, an ASCII file composed by positive integers is expected:

```
# -- Part 1 --
# 'nVN' is the total number of variable nodes and 'nCN' is the total number of check nodes
nVN nCN
# 'dmax_VN' is the higher variable node degree and 'dmax_CN' is the higher check node degree
dmax_VN dmax_CN
# list of the degrees for each variable nodes
d_VN_{1} d_VN_{2} \[\ldots\] d_VN_{nVN}
# list of the degrees for each check nodes
d_CN_{1} d_CN_{2} \[\ldots\] d_CN_{nCN}
#
# -- Part 2 --
# each following line describes the check nodes connected to a variable node, the first
# check node index is '1' (and not '0')
# variable node '1'
VN_{1}_CN_{idx_1} \[\ldots\] VN_{1}_CN_{idx_d_VN_{1}}
# variable node '2'
VN_{2}_CN_{idx_1} \[\ldots\] VN_{2}_CN_{idx_d_VN_{2}}
[\ldots]
# variable node 'nVN'
VN_{nVN}_CN_{idx_1} \[\ldots\] VN_{nVN}_CN_{idx_d_VN_{nVN}}
#
# -- Part 3 --
# each following line describes the variables nodes connected to a check node, the first
# variable node index is '1' (and not '0')
# check node '1'
CN_{1}_VN_{idx_1} \[\ldots\] CN_{1}_VN_{idx_d_CN_{1}}
# check node '2'
CN_{2}_VN_{idx_1} \[\ldots\] CN_{2}_VN_{idx_d_CN_{2}}
[\ldots]
# check node 'nCN'
CN_{nCN}_VN_{idx_1} \[\ldots\] CN_{nCN}_VN_{idx_d_CN_{nCN}}
```

In the part 2 and 3, it is possible to pad, at the end of the indexes list, with zeros when the current node degree is smaller than the maximum node degree. AFF3CT will be able to read the file even if it is padded with zeros.

For the QC format, an ASCII file composed by integers is expected:

```
# 'C' is the number of columns (there is 'C * Z' variable nodes)
# 'R' is the number of rows (there is 'R * Z' check nodes)
# 'Z' is the expansion factor
C R Z

# each 'B_r_{y}_c_{x}' is a sub-matrix bloc of size 'Z * Z'
# 'B_r_{y}_c_{x} = -1' means a zero matrix
# 'B_r_{y}_c_{x} = 0' means an identity matrix
# 'B_r_{y}_c_{x} = s' with 's' between '1' and 'Z-1' means an identity matrix shifted 's' times
```

(continues on next page)
to the right
B_r(1)c(1) B_r(1)c(2) [... ] B_r(1)c(C)
B_r(2)c(1) B_r(2)c(2) [... ] B_r(2)c(C)
[... ]
B_r(R)c(1) B_r(R)c(2) [... ] B_r(R)c(C)

# puncturing pattern (optional)
# 'T_c(x)' can be '0' or '1'
# - if 'T_c(x) = 0', does not transmit the 'Z' consecutive bits
# - if 'T_c(x) = 1', transmits the 'Z' consecutive bits
T_c(1) T_c(2) [... ] T_c(C)

---

--dec-type, -D

Type text
Allowed values BIT_FLIPPING BP_PEELING BP_FLOODING
BP_HORIZONTAL_LAYERED BP_VERTICAL_LAYERED CHASE ML

Default BP_FLOODING

Examples --dec-type BP_HORIZONTAL_LAYERED

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT_FLIPPING</td>
<td>Select the BF (Bit Flipping) category of algorithms.</td>
</tr>
<tr>
<td>BP_PEELING</td>
<td>Select the BP-P (Belief Propagation Peeling) algorithm from [Spi01].</td>
</tr>
<tr>
<td>BP_FLOODING</td>
<td>Select the BP-F (Belief Propagation with Flooding scheduling) algorithm from [MN95].</td>
</tr>
<tr>
<td>BP_HORIZONTAL_LAYERED</td>
<td>Select the BP-HL (Belief Propagation with Horizontal Layered scheduling) algorithm from [YPNA01].</td>
</tr>
<tr>
<td>BP_VERTICAL_LAYERED</td>
<td>Select the BP-VL (Belief Propagation with Vertical Layered scheduling) algorithm from [ZF02].</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common –dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common –dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

---

--dec-implem

Type text
Allowed values STD GALA GALB GALE WBF MWBF PPBF SPA LSPA AMS MS NMS OMS

Default SPA

Examples --dec-implem AMS

Select the implementation of the decoder algorithm.

Description of the allowed values:

3.2. Parameters
## Table 3.2: LDPC decoder types and available implementations.

<table>
<thead>
<tr>
<th>Decoder</th>
<th>STD</th>
<th>GALA</th>
<th>GALB</th>
<th>GALE</th>
<th>PPBF</th>
<th>WBF</th>
<th>MWBF</th>
<th>SPA</th>
<th>LSPA</th>
<th>AMS</th>
<th>MS</th>
<th>NMS</th>
<th>OMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP-P</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP-F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP-HL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BP-VL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*+*: compatible with the **--dec-simd** INTER parameter.

**+: require the C++ compiler to support the dynamic memory allocation for over-aligned data, see the P0035R4 paper. This feature is a part of the C++17 standard (working on the C++ GNU compiler version 8.1.0). When compiling with the GNU compiler in C++11 mode, the **-faligned-new** option enables specifically the required feature.

**/**: compatible with the **--dec-simd** INTRA parameter.

---

**--dec-simd**

Type: text

Allowed values: INTER

Examples: **--dec-simd INTER**

Select the SIMD strategy.

Table 3.2 shows the decoders and implementations that support SIMD.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER</td>
<td>Select the intra-frame strategy.</td>
</tr>
<tr>
<td>INTRA</td>
<td>Select the inter-frame strategy.</td>
</tr>
</tbody>
</table>
In the intra-frame strategy, SIMD units process several LLRs in parallel within a single frame decoding. In the inter-frame strategy, SIMD units decodes several independent frames in parallel in order to saturate the SIMD unit. This approach improves the throughput of the decoder but requires to load several frames before starting to decode, increasing both the decoding latency and the decoder memory footprint.

When the inter-frame SIMD strategy is set, the simulator will run with the right number of frames depending on the SIMD length. This number of frames can be manually set with the --src-fra, -F parameter. Be aware that running the simulator with the --src-fra, -F parameter set to 1 and the --dec-simd parameter set to INTER will completely be counterproductive and will lead to no throughput improvements.

**--dec-h-reorder**

Type: text

Allowed values: ASC DSC NONE

Default: NONE

Examples: --dec-h-reorder ASC

Specify the order of execution of the CNs (Check Nodes) in the decoding process depending on their degree.

The degree of a CN (Check Node) is the number of VNs (Variable Nodes) that are connected to it.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC</td>
<td>Reorder from the smallest CNs degree to the biggest CNs degree.</td>
</tr>
<tr>
<td>DSC</td>
<td>Reorder from the biggest CNs degree to the smallest CNs degree.</td>
</tr>
<tr>
<td>NONE</td>
<td>Do not change the order.</td>
</tr>
</tbody>
</table>

**--dec-ite, -i**

Type: integer

Default: 10

Examples: --dec-ite 30

Set the maximal number of iterations in the LDPC decoder.

By default, in order to speedup the decoding time, the decoder can stop the decoding process if all the parity check equations are verified (also called the syndrome detection). In that case the decoder can perform less decoding iterations than the given number. To force the decoder to make all the iterations, use the --dec-no-synd parameter.

**--dec-min**

Type: text

Allowed values: MIN MINL MINS
**Default** MINL

**Examples** --dec-min MIN

Define the min\(^*\) operator approximation used in the AMS update rule.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINS</td>
<td>( \min^*(a, b) = \min(a, b) + \log(1 + \exp(-(a + b))) - \log(1 + \exp(-</td>
</tr>
<tr>
<td>MINL</td>
<td>( \min^*(a, b) \approx \min(a, b) + \text{corr}(a + b) - \text{corr}(</td>
</tr>
<tr>
<td>MIN</td>
<td>( \min^*(a, b) \approx \min(a, b) ).</td>
</tr>
</tbody>
</table>

**MIN** for Min Star is the exact min\(^*\) operator. MINL for Min Linear is a linear approximation of the min\(^*\) function. MIN for Min is the simplest approximation with only a min function.

**--dec-norm**

**Type** real number

**Default** 1.0

**Examples** --dec-norm 0.75

Set the normalization factor used in the NMS update rule.

**--dec-off**

**Type** real number

**Default** 0.0

**Examples** --dec-off 0.25

Set the offset used in the OMS update rule.
--dec-mwbf-factor

Type real number
Default 0.0
Examples --dec-mwbf-factor 1.0

Give the weighting factor used in the MWBF algorithm.

--dec-synd-depth

Type integer
Default 1
Examples --dec-synd-depth 2

Set the number of iterations to process before enabling the syndrome detection. In some cases, it can help to avoid false positive detections.

--dec-ppbf-proba

Type list of real numbers
Examples --dec-ppbf-proba "0,0.001,0.1,0.3,1,1,1"

Give the probabilities of the Bernouilli distribution of the PPBF. The number of given values must be equal to the biggest variable node degree plus two.

Thus, with a parity matrix that has its largest variable node at 5, you must give 7 values. Each value corresponds to an energy level as described in [LGK+19].

--dec-no-synd

Disable the syndrome detection, all the LDPC decoding iterations will be performed.

References

LDPC Puncturer parameters

--pct-fra-size, -N

Type integer
Examples --pct-fra-size 912

Set the frame size $N$. This is not necessarily the codeword size if a puncturing pattern is used.
---pct-type

**Type**  text  
**Allowed values**  LDPC NO  
**Default**  LDPC  
**Examples**  --pct-type LDPC  

Select the puncturer type.  

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Disable the puncturer.</td>
</tr>
<tr>
<td>LDPC</td>
<td>Puncture the LDPC codeword.</td>
</tr>
</tbody>
</table>

---pct-pattern

**Type**  binary vector  
**Examples**  --pct-pattern "1,1,1,0"  

Give the puncturing pattern following the LDPC code.  

The number $P$ of values given in this pattern must be as $N_{cw} = P \times Z$ where $Z$ is the number of bits represented by a single value in the pattern.  

This LDPC puncturer behavior is such as, for the above example, the first three quarter bits are kept and the last quarter is removed from the frame.

**Codec Polar**

**Polar Encoder parameters**

---enc-type

**Type**  text  
**Allowed values**  POLAR AZCW COSET USER  
**Default**  POLAR  
**Examples**  --enc-type AZCW  

Select the encoder type.  

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLAR</td>
<td>Select the standard Polar encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>
--enc-no-sys

Enable non-systematic encoding. By default the encoding process is systematic.

--enc-fb-gen-method

   Type  text
   Allowed values  FILE GA TV BEC 5G
   Examples  --enc-fb-gen-method FILE

Select the frozen bits generation method.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GA</td>
<td>Select the GA (Gaussian Approximation) method from [Tri12].</td>
</tr>
<tr>
<td>TV</td>
<td>Select the TV (Tal &amp; Vardy) method which is based on Density Evolution (DE (Density Evolution)) approach from [TV13], to use with the --enc-fb-awgn-path parameter.</td>
</tr>
<tr>
<td>FILE</td>
<td>Read the best channels from an external file, to use with the --enc-fb-awgn-path parameter.</td>
</tr>
<tr>
<td>BEC</td>
<td>Generate frozen bits for the BEC (Binary Erasure Channel) channel from [Ari09].</td>
</tr>
<tr>
<td>5G</td>
<td>Generate the frozen bits as described in the 5G standard [3GPP].</td>
</tr>
</tbody>
</table>

Note: By default, when using the GA or the TV method, the frozen bits are optimized for each SNR point. To override this behavior you can use the --enc-fb-noise parameter.

Note: When using the FILE method, the frozen bits are always the same regardless of the SNR value.

Note: When using the BEC method, the frozen bits are optimized for each erasure probability.

Note: When using the 5G method, the codeword size must be inferior to 1024.

--enc-fb-awgn-path

   Type  path
   Rights  read only
   Examples  --enc-fb-awgn-path example/path/to/the/right/place/

Set the path to a file or a directory containing the best channels to select the frozen bits.

An ASCII file is expected, for instance, the following file describes the most reliable channels optimized for a codeword of size \( N = 8 \) and for an AWGN (Additive White Gaussian Noise) channel where the noise variance is \( \sigma = 0.435999 \):

```
% 3.2. Parameters  55
```
Given the previous file, if we suppose a Polar code of size $N = 8$ with $K = 4$ information bits, the frozen bits are at the 0, 1, 2, 4 positions in the codeword. The strategy is to freeze the less reliable channels.

**Warning:** The FILE frozen bits generator expects a file and not a directory.

**Warning:** The TV frozen bits generator expects a directory and not a file. AFF3CT comes with input configuration files, a part of those configuration files are a set of best channels pre-generated with the TV method (see conf/cde/awgn_polar_codes/TV/).

---

### --enc-fb-dump-path

**Type** folder  
**Rights** write only  
**Examples** --enc-fb-dump-path example/path/to/the/right/place/

Set the path to store the best channels.

**Note:** Works only for the GA and BEC frozen bits generation methods.

---

### --enc-fb-noise

**Type** real number  
**Examples** --enc-fb-noise 1.0

Select the noise for which the frozen bits will be optimized.

Can be a gaussian noise variance $\sigma$ for GA and TV generation methods, or an event probability for the BEC generation method. All the noise points in the simulation will use the same frozen bits configuration.

---

### References

**Polar Decoder parameters**

---

### --dec-type, -D

**Type** text  
**Allowed values** SC SCAN SCF SCL SCL_MEM ASCL ASCL_MEM CHASE ML  
**Default** SC  
**Examples** --dec-type ASCL
Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>Select the original SC algorithm from [Ari09].</td>
</tr>
<tr>
<td>SCAN</td>
<td>Select the SCAN (Soft CANCELion) algorithm from [FB14].</td>
</tr>
<tr>
<td>SCF</td>
<td>Select the SCF (Successive Cancellation Flip) algorithm from [ABSB14].</td>
</tr>
<tr>
<td>SCL</td>
<td>Select the SCL (Successive Cancellation List) algorithm from [TV11], also support the improved CA (CRC Aided)-SCL algorithm.</td>
</tr>
<tr>
<td>SCL_MEM</td>
<td>Select the SCL algorithm, same as the previous one but with an implementation optimized to reduce the memory footprint.</td>
</tr>
<tr>
<td>ASCL</td>
<td>Select the A-SCL (Adaptive Successive Cancellation List) algorithm from [LST12], PA-SCL (Partially Adaptive Successive Cancellation List) and FA-SCL (Fully Adaptive Successive Cancellation List) variants from [LeonardonCL+17] are available (see the --dec-partial-adaptive parameter).</td>
</tr>
<tr>
<td>ASCL_MEM</td>
<td>Select the A-SCL algorithm, same as the previous one but with an implementation optimized to reduce the memory footprint.</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text

Allowed values NAIVE FAST

Default FAST

Examples --dec-implem FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAIVE</td>
<td>Select the naive implementation which is typically slow (not supported by the A-SCL decoders).</td>
</tr>
<tr>
<td>FAST</td>
<td>Select the fast implementation, available only for the SC, SCL, SCL-MEM, A-SCL and A-SCL-MEM decoders.</td>
</tr>
</tbody>
</table>

Warning: FAST implementations only support systematic encoding of Polar codes.

Note: The SC FAST implementation has been presented in [LeGalLJego15][CLeGalL+15][CAL+16].

Note: The SCL, CA-SCL and A-SCL FAST implementations have been presented in [LeonardonCL+17].

--dec-simd

Type text
Allowed values

**INTER INTRA**

**Examples**  
--dec-simd INTER

Select the SIMD strategy.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER</td>
<td>Select the inter-frame strategy, only available for the SC FAST decoder (see [LeGalLego15][CLeGalL+15][CAL+16]).</td>
</tr>
<tr>
<td>INTRA</td>
<td>Select the intra-frame strategy, only available for the SC (see [CLeGalL+15][CAL+16]), SCL and A-SCL decoders (see in [LeonardonCL+17]).</td>
</tr>
</tbody>
</table>

**Note:** In the intra-frame strategy. SIMD units process several LLRs in parallel within a single frame decoding. This approach is efficient in the upper layers of the tree and in the specialized nodes, but more limited in the lowest layers where the computation becomes more sequential. In the inter-frame strategy, SIMD units decodes several independent frames in parallel in order to saturate the SIMD unit. This approach improves the throughput of the decoder but requires to load several frames before starting to decode, increasing both the decoding latency and the decoder memory footprint.

**Note:** When the inter-frame SIMD strategy is set, the simulator will run with the right number of frames depending on the SIMD length. This number of frames can be manually set with the --src-fra, -F parameter. Be aware that running the simulator with the --src-fra, -F parameter set to 1 and the --dec-simd parameter set to INTER will completely be counterproductive and will lead to no throughput improvements.

**--dec-ite, -i**

**Type** integer

**Examples** --dec-ite 1

Set the number of decoding iterations in the SCAN decoder.

**--dec-flips**

**Type** integer

**Examples** --dec-flips 1

Set the maximum number of bit flips in the decoding algorithm.
Corresponds to the $T$ parameter of the SCF decoding algorithm [ABSB14].

**--dec-lists, -L**

**Type** integer

**Examples** --dec-lists 1

Set the number of lists to maintain in the SCL and A-SCL decoders.
Select the partial adaptive (PA-SCL) variant of the A-SCL decoder (by default the FA-SCL is selected).

**--dec-polar-nodes**

Type text

Default "\{R0,R1,R0L,REP,REPL,SPC\}"

Examples --dec-polar-nodes "\{R0,R1\}"

Set the rules to enable in the tree simplifications process. This parameter is compatible with the SC FAST, the SCL FAST, SCL-MEM FAST, the A-SCL FAST and the the A-SCL-MEM FAST decoders.

Here are the available node types (or rules):

- **R0**: Rate 0, all the bits are frozen,
- **R0L**: Rate 0 left, the next left node in the tree is a R0,
- **R1**: Rate 1, all the bits are information bits,
- **REP**: Repetition code,
- **REPL**: Repetition left, the next left node in the tree is REP,
- **SPC**: SPC (Single Parity Check) code.

Those node types are well explained in [SGV+14][CLeGalL+15]. It is also possible to specify the level in the tree where the node type will be recognized. For instance, the following value "\{R0,R1,R0L,REP_2-8,REPL,SPC_4+\}" matches:

- **R0**: all the Rate 0 nodes,
- **R0L**: all the Rate 0 left nodes,
- **R1**: all the Rate 1 nodes,
- **REP_2-8**: the repetition nodes with a size between 2 and 8 (including 2 and 8),
- **REPL**: all the repetition left nodes (will be automatically limited by the REP_2-8 rule),
- **SPC_4+**: the SPC nodes with a size equal or higher than 4.

To disable the tree cuts you can use the following value: "\{R0_1,R1_1\}".

References

Polar Puncturer parameters

**--pct-fra-size, -N**

Type integer

Examples --pct-fra-size 1

Set the frame size $N$. This is not necessarily the codeword size if a puncturing pattern is used.
--pct-info-bits, -K \textbf{REQUIRED}

Type integer

Examples --pct-info-bits 1

Set the number of information bits $K$.

--pct-type

Type text

Allowed values NO SHORTLAST

Default NO

Examples --pct-type NO

Select the puncturer type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Disable the puncturer.</td>
</tr>
<tr>
<td>SHORTLAST</td>
<td>Select the short last puncturing strategy from [NCL13][WL14][Mil15].</td>
</tr>
</tbody>
</table>

References

Codec RA (Repeat and Accumulate)

RA Encoder parameters

--enc-cw-size, -N \textbf{REQUIRED}

Type integer

Examples --enc-cw-size 1

Set the codeword size $N$.

--enc-info-bits, -K \textbf{REQUIRED}

Type integer

Examples --enc-info-bits 1

Set the number of information bits $K$.
--enc-type

Type text
Allowed values RA AZCW COSET USER
Default RA
Examples --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>Select the standard RA encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

RA Decoder parameters

--dec-type, -D

Type text
Allowed values RA CHASE ML
Default RA
Examples --dec-type CHASE

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA</td>
<td>Select the RA decoder based on the MS update rule in the CNS.</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text
Allowed values STD
Default STD
Examples --dec-implem STD

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select the STD implementation.</td>
</tr>
</tbody>
</table>
--dec-ite, -i

Type integer

Examples --dec-ite 1

Set the number of iterations to perform in the decoder.

Codec Repetition

Repetition Encoder parameters

--enc-cw-size, -N

Type integer

Examples --enc-cw-size 1

Set the codeword size $N$.

$N$ has to be divisible by $K$.

--enc-info-bits, -K

Type integer

Examples --enc-info-bits 1

Set the number of information bits $K$.

--enc-type

Type text

Allowed values REP AZCW COSET USER

Examples --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REP</td>
<td>Select the standart repetition decoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>
--enc-no-buff

Disable the buffered encoding.

Without the buffered encoding, considering $K$ information bits $U_0, U_1, ..., U_{K-1}$, the corresponding sequence of bits in the codeword is organized as follow:

$$X_0^0, X_0^1, [...], X_0^{rep-1}, X_1^0, X_1^1, [...], X_1^{rep-1}, [...], X_{K-1}^0, X_{K-1}^1, [...], X_{K-1}^{rep-1},$$

with $rep = N/K$.

With the buffered encoding, considering $K$ information bits $U_0, U_1, [...], U_{K-1}$, the corresponding sequence of bits in the codeword is organized as follow:

$$X_0^0, X_1^0, [...], X_0^1, X_1^1, [...], X_1^{rep-1}, X_{K-1}^0, X_{K-1}^1, [...], X_{K-1}^{rep-1},$$

with $rep = N/K$.

Repetition Decoder parameters

--dec-type, -D

Type text

Allowed values REPETITION CHASE ML

Default REPETITION

Examples --dec-type CHASE

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPETITION</td>
<td>Select the repetition decoder.</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text

Allowed values STD FAST

Default STD

Examples --dec-implem FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select the STD implementation.</td>
</tr>
<tr>
<td>FAST</td>
<td>Select the fast implementation, much more faster without the --enc-no-buff parameter.</td>
</tr>
</tbody>
</table>

Codec RS (Reed-Solomon)

RS Encoder parameters
--enc-cw-size, -N

Type integer

Examples --enc-cw-size 127

Set the codeword size $N$.

$N = 2^{m-1}$, where $m$ is an integer from 3 that represents also the number of bits per symbol. Thus, the binary codeword size is $N \times m$.

--enc-info-bits, -K

Type integer

Examples --enc-info-bits 1

Set the number of information bits $K$.

This argument is not required if the correction power $T$ is given with --dec-corr-pow, -T, as it is calculated automatically with the formula $K = N - 2.T$.

--enc-type

Type text

Allowed values RS AZCW COSET USER

Default RS

Examples --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>Select the standard RS encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

RS Decoder parameters

The RS decoder was described by Reed and Solomon in 1960 [ISR60].

--dec-type, -D

Type text

Allowed values ALGEBRAIC CHASE ML

Default ALGEBRAIC

Examples --dec-type ALGEBRAIC
Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGEBRAIC</td>
<td>Select the Berlekamp-Massey algorithm [Ber15][Mas69] followed by a Chien search [Chi64].</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

**--dec-implem**

*Type* text

*Allowed values* STD GENIUS

*Default* STD

*Examples* --dec-implem GENIUS

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>A standard implementation of the RS.</td>
</tr>
<tr>
<td>GENIUS</td>
<td>A really fast implementation that compare the input to the original codeword and correct it only when the number of symbols errors is less or equal to the RS correction power.</td>
</tr>
</tbody>
</table>

**Note:** In the STD implementation, the Chien search finds roots of the error location polynomial. If the number of found roots does not match the number of found errors by the Berlekamp–Massey algorithm, then the frame is not modified.

When a frame is very corrupted and when the above algorithms can be wrong in the correction by converging to another codeword, the GENIUS implementation cannot fail. Results may then differ from a real word implementation.

**--dec-corr-pow, -T**

*Type* integer

*Default* 5

*Examples* --T 18

Set the correction power of the RS decoder. This value corresponds to the number of symbols errors that the decoder is able to correct.

It is automatically calculated from the input and codeword sizes. See also the argument --enc-info-bits, -K.

**References**

**Codec RSC (Recursive Systematic Convolutional)**
RSC Encoder parameters

--enc-info-bits, -K [REQUIRED]

Type integer

Examples --enc-info-bits 1

Set the number of information bits $K$.

The codeword size $N$ is automatically deduced: $N = 2 \times (K + \log_2(ts))$ where $ts$ is the trellis size.

--enc-type

Type text

Allowed values RSC AZCW COSET USER

Default RSC

Examples --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSC</td>
<td>Select the standard RSC encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

--enc-no-buff

Disable the buffered encoding.

Without the buffered encoding, considering the following sequence of $K$ information bits: $U_0, U_1, [...], U_{K-1}$, the encoded bits will be organized as follow:

$X_s^0, X_s^1, X_s^2, [...], X_s^K, X_p^0, X_p^1, [...], X_p^K, X_p^s, X_p^t, X_p^{log_2(ts)-1}, X_p^{log_2(ts)-1}$, where $s$ and $p$ are respectively systematic and parity bits, $t$ the tail bits and $ts$ the trellis size.

With the buffered encoding, considering the following sequence of $K$ information bits: $U_0, U_1, [...], U_{K-1}$, the encoded bits will be organized as follow:

$X_s^0, X_s^1, [...], X_s^K, X_s^0, X_s^1, [...], X_s^{log_2(ts)-1}, X_p^0, X_p^1, [...], X_p^K, X_p^0, X_p^1, [...], X_p^{log_2(ts)-1}$, where $s$ and $p$ are respectively systematic and parity bits, $t$ the tail bits and $ts$ the trellis size.

--enc-poly

Type text

Default "(013,015)"

Examples --enc-poly "(023, 033)"
Set the polynomials that define the RSC code (or the trellis structure). The expected form is \( \{A, B\} \) where \( A \) and \( B \) are given in octal.

**--enc-std**

**Type** text

**Allowed values** CCSDS LTE

**Examples** --enc-std CCSDS

Select a standard: set automatically some parameters (can be overwritten by user given arguments).

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSDS</td>
<td>Set the (--enc-poly) parameter to ({023,033}) according to the CCSDS (Consultative Committee for Space Data Systems) standard (16-stage trellis).</td>
</tr>
<tr>
<td>LTE</td>
<td>Set the (--enc-poly) parameter to ({013,015}) according to the LTE (Long Term Evolution) standard (8-stage trellis).</td>
</tr>
</tbody>
</table>

**RSC Decoder parameters**

**--dec-type, -D**

**Type** text

**Allowed values** BCJR CHASE ML

**Examples** --dec-type BCJR

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCJR</td>
<td>Select the BCJR (Bahl, Cocke, Jelinek and Raviv algorithm or Maximum A Posteriori (MAP)) algorithm from [BCJR74].</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common (--dec-type, -D) parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common (--dec-type, -D) parameter.</td>
</tr>
</tbody>
</table>

**--dec-implem**

**Type** text

**Allowed values** GENERIC STD FAST VERY_FAST

**Default** STD

**Examples** --dec-implem FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:
## AFF3CT Documentation, Release v2.3.3-2-g91414a0d

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERIC</td>
<td>Select the generic BCJR implementation that can decode any trellis (slow compared to the other implementations).</td>
</tr>
<tr>
<td>STD</td>
<td>Select the STD BCJR implementation, specialized for the ({013,015}) polynomials (c.f. the (-enc-poly) parameter).</td>
</tr>
<tr>
<td>FAST</td>
<td>Select the fast BCJR implementation, specialized for the ({013,015}) polynomials (c.f. the (-enc-poly) parameter).</td>
</tr>
<tr>
<td>VERY_FAST</td>
<td>Select the very fast BCJR implementation, specialized for the ({013,015}) polynomials (c.f. the (-enc-poly) parameter).</td>
</tr>
</tbody>
</table>

### --dec-simd

**Type** text

**Allowed values** INTER INTRA

**Examples** `--dec-simd INTER`

Select the SIMD strategy.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER</td>
<td>Select the inter-frame strategy, only available for the BCJR <strong>STD</strong>, <strong>FAST</strong> and <strong>VERY_FAST</strong> implementation (see [CTL+16]).</td>
</tr>
<tr>
<td>INTRA</td>
<td>Select the intra-frame strategy, only available for the BCJR <strong>STD</strong> and <strong>FAST</strong> implementations (see [WWY+13]).</td>
</tr>
</tbody>
</table>

**Note:** In the intra-frame strategy, SIMD units process several LLRs in parallel within a single frame decoding. In the inter-frame strategy, SIMD units decodes several independent frames in parallel in order to saturate the SIMD unit. This approach improves the throughput of the decoder but requires to load several frames before starting to decode, increasing both the decoding latency and the decoder memory footprint.

**Note:** When the inter-frame SIMD strategy is set, the simulator will run with the right number of frames depending on the SIMD length. This number of frames can be manually set with the `--src-fra, -F` parameter. Be aware that running the simulator with the `--src-fra, -F` parameter set to 1 and the `--dec-simd` parameter set to INTER will completely be counterproductive and will lead to no throughput improvements.

### --dec-max

**Type** text

**Allowed values** MAXS MAXL MAX

**Examples** `--dec-max MAX`

Select the approximation of the \(\text{max}^*\) operator used in the trellis decoding.

Description of the allowed values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXS</td>
<td>$\max^*(a, b) = \max(a, b) + \log(1 + \exp(-</td>
</tr>
<tr>
<td>MAXL</td>
<td>$\max^*(a, b) \approx \max(a, b) + \max(0, 0.301 - (0.5</td>
</tr>
<tr>
<td>MAX</td>
<td>$\max^*(a, b) \approx \max(a, b)$.</td>
</tr>
</tbody>
</table>

MAXS for Max Star is the exact $\max^*$ operator. MAXL for Max Linear is a linear approximation of the $\max^*$ function. MAX for Max is the simplest $\max^*$ approximation with only a max function.

**Note:** The BCJR with the max approximation is also called the max-log-MAP (Maximum A Posteriori) algorithm.

### References

**Codec RSC DB (Double Binary)**

**RSC DB Encoder parameters**

`--enc-info-bits, -K` **REQUIRED**

**Type** integer

**Examples** `--enc-info-bits 1`

Set the number of information bits $K$.

The codeword size $N$ is automatically deduced: $N = 2 \times K$.

`--enc-type`

**Type** text

**Allowed values** RSC_DB AZCW COSET USER

**Default** RSC_DB

**Examples** `--enc-type AZCW`

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSC_DB</td>
<td>Select the standard RSC DB encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common <code>--enc-type</code> parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common <code>--enc-type</code> parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common <code>--enc-type</code> parameter.</td>
</tr>
</tbody>
</table>

`--enc-no-buff`

Disable the buffered encoding.
--enc-std

Type text
Allowed values DVB-RCS1 DVB-RCS2
Default DVB-RCS1
Examples --enc-std DVB-RCS2

Select a standard.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVB-RCS1</td>
<td>Select the configuration of the DVB-RCS1 (Digital Video Broadcasting - Return Channel via Satellite 1) standard.</td>
</tr>
<tr>
<td>DVB-RCS2</td>
<td>Select the configuration of the DVB-RCS2 (Digital Video Broadcasting - Return Channel via Satellite 2) standard.</td>
</tr>
</tbody>
</table>

RSC DB Decoder parameters

--dec-type, -D

Type text
Allowed values BCJR CHASE ML
Default BCJR
Examples --dec-type BCJR

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCJR</td>
<td>Select the BCJR DB decoder [BCJR74].</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text
Allowed values GENERIC DVB-RCS1 DVB-RCS2
Default DVB-RCS1
Examples --dec-implem DVB-RCS1

Select the implementation of the decoder algorithm.

Description of the allowed values:
Value | Description
---|---
**GENERIC** | Select a generic implementation that works on any trellis.
**DVB-RCS1** | Select an implementation dedicated to the DVB-RCS1 trellis (faster than the GENERIC implementation).
**DVB-RCS2** | Select an implementation dedicated to the DVB-RCS2 trellis (faster than the GENERIC implementation).

---

**--dec-max**

**Type** text

**Allowed values** MAXS MAXL MAX

**Examples** --dec-max MAX

Select the approximation of the \( \max^* \) operator used in the trellis decoding.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXS</td>
<td>( \max^*(a, b) = \max(a, b) + \log(1 + \exp(-</td>
</tr>
<tr>
<td>MAXL</td>
<td>( \max^*(a, b) \approx \max(a, b) + \max(0, 0.301 - (0.5</td>
</tr>
<tr>
<td>MAX</td>
<td>( \max^*(a, b) \approx \max(a, b) ).</td>
</tr>
</tbody>
</table>

MAXS for Max Star is the exact \( \max^* \) operator. MAXL for Max Linear is a linear approximation of the \( \max^* \) function. MAX for Max is the simplest \( \max^* \) approximation with only a \( \max \) function.

**Note:** The BCJR with the max approximation is also called the max-log-MAP algorithm.

---

**References**

**Codec Turbo**

**Turbo Encoder parameters**

**--enc-info-bits, -K** [REQUIRED]

**Type** integer

**Examples** --enc-info-bits 1

Set the number of information bits \( K \).

The codeword size \( N \) is automatically deduced: \( N = 3 \times K + 4 \times \log_2(ts) \) where \( ts \) is the trellis size.

---

**--enc-type**

**Type** text

**Allowed values** TURBO A2CW COSET USER
**Default** TURBO

**Examples** --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TURBO</td>
<td>Select the standard Turbo encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

**--enc-sub-type**

Please refer to the RSC --enc-type parameter.

**--enc-json-path**

**Type** file  
**Rights** write only

**Examples** --enc-json-path example/path/to/the/right/file

Select the file path to dump the encoder and decoder internal values (in JSON (JavaScript Object Notation) format). Those values can be observed with the dedicated Turbo Code Reader available on the AFF3CT website: http://aff3ct.github.io/turbo_reader.html.

**Note:** Using this parameter will slow down considerably the encoder and decoder throughputs.

**--enc-sub-no-buff**

Disable the buffered encoding.

**Without the buffered encoding**, considering the following sequence of $K$ information bits: $U_0, U_1, ..., U_{K-1}$, the encoded bits will be organized as follow:

$$X_{sn}^0, X_{pn}^0, X_{pi}^0, ..., X_{sn}^{K-1}, X_{pn}^{K-1}, X_{pi}^{K-1}, X_{sn}^{*}, X_{pn}^{*}, X_{pi}^{*}, ..., X_{sn}^{t}, X_{pn}^{t}, X_{pi}^{t}, X_{sn}^{*}, X_{pn}^{*}, X_{pi}^{*}, ..., X_{sn}^{t}, X_{pn}^{t}, X_{pi}^{t}, X_{sn}^{t}, X_{pn}^{t}, X_{pi}^{t},$$

where $sn$ and $pn$ are respectively systematic and parity bits in the natural domain, $si$ and $pi$ are respectively systematic and parity bits in the interleaved domain, $t$ the tail bits and and $ts$ the trellis size.

**With the buffered encoding**, considering the following sequence of $K$ information bits: $U_0, U_1, ..., U_{K-1}$, the encoded bits will be organized as follow:

$$X_{sn}^0, ..., X_{sn}^{K-1}, X_{sn}^{*}, ..., X_{sn}^{t}, X_{pn}^0, ..., X_{pn}^{K-1}, X_{pn}^{*}, ..., X_{pn}^{t}, X_{pi}^0, ..., X_{pi}^{K-1}, X_{pi}^{*}, ..., X_{pi}^{t}, X_{sn}^{*}, ..., X_{sn}^{t}, X_{pn}^{*}, ..., X_{pn}^{t}, X_{pi}^{*}, ..., X_{pi}^{t},$$

where $sn$ and $pn$ are respectively systematic and parity bits in the natural domain, $si$ and $pi$ are respectively systematic and parity bits in the interleaved domain, $t$ the tail bits and and $ts$ the trellis size.

**--enc-sub-poly**

Please refer to the RSC --enc-poly parameter.
--enc-sub-std

Type text

Allowed values CCDS LTE

Examples --enc-sub-std CCDS

Select a standard: set automatically some parameters (can be overwritten by user given arguments).

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCDS</td>
<td>Set the --enc-sub-poly parameter to {023, 033} according to the CCSDS standard (16-stage trellis) and select the CCSDS interleaver (see the --itl-type parameter).</td>
</tr>
<tr>
<td>LTE</td>
<td>Set the --enc-sub-poly parameter to {013, 015} according to the LTE standard (8-stage trellis) and select the LTE interleaver (see the --itl-type parameter).</td>
</tr>
</tbody>
</table>

Turbo Decoder parameters

--dec-type, -D

Type text

Allowed values TURBO CHASE ML

Default TURBO

Examples --dec-type CHASE

Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TURBO</td>
<td>Select the Turbo decoder, the two sub-decoders are from the RSC code family.</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text

Allowed values STD FAST

Default FAST

Examples --dec-implem FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select the STD implementation.</td>
</tr>
<tr>
<td>FAST</td>
<td>Select the fast implementation from [CTL+16].</td>
</tr>
</tbody>
</table>
--dec-sub-type, -D

Please refer to the RSC --dec-type, -D parameter.

--dec-sub-implem

Please refer to the RSC --dec-implem parameter.

--dec-sub-simd

Please refer to the RSC --dec-simd parameter.

--dec-crc-start

Type integer
Default 2
Examples --dec-fnc-crc-ite 1
Set the first iteration to start the CRC checking.

Note: This parameter requires the Turbo code to be concatenated with a CRC to work, see the CRC parameters.

--dec-fnc

Enable the FNC (Flip aNd Check) post processing technique.

Note: The FNC post processing technique is detailed in [TLLeGal+16].

Note: This parameter requires the Turbo code to be concatenated with a CRC to work, see the CRC parameters.

--dec-fnc-ite-m

Type integer
Default 3
Examples --dec-fnc-ite-m 2
Set the first iteration at which the FNC is used.
See the --dec-fnc parameter.
--dec-fnc-ite-M

Type  integer
Default  10
Examples  --dec-fnc-ite-M  6

Set the last iteration at which the FNC is used.
See the --dec-fnc parameter.

--dec-fnc-ite-s

Type  integer
Default  1
Examples  --dec-fnc-ite-s  2

Set the iteration step for the FNC technique.
See the --dec-fnc parameter.

--dec-fnc-q

Type  integer
Default  10
Examples  --dec-fnc-q  6

Set the search space for the FNC technique.
See the --dec-fnc parameter.

--dec-ite, -i

Type  integer
Default  6
Examples  --dec-ite  8

Set the maximal number of iterations in the Turbo decoder.
If the Turbo code is concatenated with a CRC and if the CRC is checked, the decoder can stop before making all the iterations.

--dec-sc

Enable the Self-Corrected (SC) decoder.

Note: The SC decoder is detailed in [Ton17] (in French).
**Note:** This parameter requires the Turbo code to be concatenated with a CRC to work, see the *CRC parameters*.

**--dec-sf-type**

Type: text

**Allowed values** ADAPTIVE ARRAY CST LTE LTE_VEC

**Examples**

```
--dec-sf-type ADAPTIVE
--dec-sf-type CST 0.5
```

Select a scaling factor (SF (Scaling Factor)) to be applied to the extrinsic values after each half iteration. This is especially useful with the max-log-MAP sub-decoders (BCJR with the max approximation): the SF helps to recover a part of the decoding performance loss compare to the MAP algorithm (BCJR with the max* operator).

**Note:** The SF technique is detailed in [VF00].

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPTIVE</td>
<td>Select the adaptive SF, for the first and second iterations a SF of 0.5 is applied, for the other iterations the SF is 0.85.</td>
</tr>
<tr>
<td>ARRAY</td>
<td>Select an hard-coded array of SFs (Scaling Factors) (c.f. Table 3.3).</td>
</tr>
<tr>
<td>CST</td>
<td>Set the same SF to be applied for each iterations.</td>
</tr>
<tr>
<td>LTE</td>
<td>Select a 0.75 SF.</td>
</tr>
<tr>
<td>LTE_VEC</td>
<td>Select a 0.75 vectorized SF (faster than LTE), used in [CTL+16].</td>
</tr>
</tbody>
</table>

Table 3.3: Hard-coded array of SFs.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.30</td>
</tr>
<tr>
<td>4</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>0.70</td>
</tr>
<tr>
<td>6</td>
<td>0.80</td>
</tr>
<tr>
<td>7</td>
<td>0.90</td>
</tr>
<tr>
<td>8</td>
<td>0.95</td>
</tr>
</tbody>
</table>

**--dec-sub-max**

Please refer to the RSC *--dec-max* parameter.

**References**
Turbo Puncturer parameters

--pct-type

Type text
Allowed values NO TURBO
Default NO
Examples --pct-type NO

Select the puncturer type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Disable the puncturer.</td>
</tr>
<tr>
<td>TURBO</td>
<td>Enable the puncturing patterns.</td>
</tr>
</tbody>
</table>

Note: The frame size will be automatically set from the given puncturing pattern (c.f. the --pct-pattern parameter).

--pct-pattern

Type list of (list of (boolean:including set={0|1}):limited length [1:inf]):limited length [3;3], elements of same length
Examples --pct-pattern "11,10,01"

Define the puncturing pattern.

Considering the "11,10,01" puncturing pattern, the first sub-pattern 11 defines the emitted systematic bits, the second sub-pattern 10 defines the emitted parity bits in the natural domain and the third sub-pattern 01 defines the emitted parity bits in the interleaved domain. 1 means that the bit has to be transmitted and 0 means that the bit transmission has to be erased.

Given the following frame: $X_0^{sn}, X_1^{pn}, X_2^{pi}, X_3^{sn}, X_4^{pn}, X_5^{pi}, X_6^{sn}, X_7^{pn}, X_8^{pi}$, with the "11,10,01" puncturing pattern, the underlined bits will not be emitted. In the previous example, tail bits are not taken into account but in reality they are always emitted.

Codec Turbo DB

Turbo DB Encoder parameters

--enc-info-bits, -K

Type integer
Examples --enc-info-bits 40

Set the number of information bits $K$.
The codeword size $N$ is automatically deduced: $N = 3 \times K$. 
**--enc-type**

Type text

**Allowed values** TURBO_DB AZCW COSET USER

**Default** TURBO_DB

**Examples** --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TURBO_DB</td>
<td>Select the standard Turbo DB encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

**--enc-sub-type**

Please refer to the RSC DB --enc-type parameter.

**--enc-sub-std**

Type text

**Allowed values** DVB-RCS1 DVB-RCS2

**Default** DVB-RCS1

**Examples** --enc-sub-std DVB-RCS2

Select a standard.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVB-RCS1</td>
<td>Set the DVB-RCS1 trellis and select the DVB-RCS1 interleaver (see the --itl-type parameter).</td>
</tr>
<tr>
<td>DVB-RCS2</td>
<td>Set the DVB-RCS2 trellis and select the DVB-RCS2 interleaver (see the --itl-type parameter).</td>
</tr>
</tbody>
</table>

**Turbo DB Decoder parameters**

**--dec-type, -D**

Type text

**Allowed values** TURBO_DB CHASE ML

**Default** TURBO_DB

**Examples** --dec-type CHASE
Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TURBO_DB</td>
<td>Select the standard Turbo decoder.</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

--dec-implem

Type text

Allowed values STD

Default STD

Examples --dec-implem STD

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select the STD implementation.</td>
</tr>
</tbody>
</table>

--dec-sub-type, -D

Please refer to the RSC DB --dec-type, -D parameter.

--dec-sub-implem

Please refer to the RSC DB --dec-implem parameter.

--dec-crc-start

Type integer

Default 2

Examples --dec-fnc-crc-ite 1

Set the first iteration to start the CRC checking.

Note: This parameter requires the Turbo code to be concatenated with a CRC to work, see the CRC parameters.

--dec-fnc

Enable the FNC post processing technique.
Note: The FNC post processing technique is detailed in [TLLeGal+16].

Note: This parameter requires the Turbo code to be concatenated with a CRC to work, see the CRC parameters.

--dec-fnc-ite-m

Type  integer
Default 3
Examples  --dec-fnc-ite-m 2
Set the first iteration at which the FNC is used.
See the --dec-fnc parameter.

--dec-fnc-ite-M

Type  integer
Default 10
Examples  --dec-fnc-ite-M 6
Set the last iteration at which the FNC is used.
See the --dec-fnc parameter.

--dec-fnc-ite-s

Type  integer
Default 1
Examples  --dec-fnc-ite-s 2
Set the iteration step for the FNC technique.
See the --dec-fnc parameter.

--dec-fnc-q

Type  integer
Default 10
Examples  --dec-fnc-q 6
Set the search space for the FNC technique.
See the --dec-fnc parameter.
--dec-ite, -i

Type  integer
Default  6
Examples  --dec-ite 8

Set the maximal number of iterations in the Turbo decoder.
If the Turbo code is concatenated with a CRC and if the CRC is checked, the decoder can stop before making all the iterations.

--dec-sf-type

Type  text
Allowed values  ADAPTIVE ARRAY CST LTE LTE_VEC
Examples
--dec-sf-type ADAPTIVE
--dec-sf-type CST 0.5

Select a scaling factor (SF) to be applied to the extrinsic values after each half iteration.
This is especially useful with the max-log-MAP sub-decoders (BCJR with the max approximation): the SF helps to recover a part of the decoding performance loss compare to the MAP algorithm (BCJR with the max* operator).

Note:  The SF technique is detailed in [VF00].

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAPTIVE</td>
<td>Select the adaptive SF, for the first and second iterations a SF of 0.5 is applied, for the other iterations the SF is 0.85.</td>
</tr>
<tr>
<td>ARRAY</td>
<td>Select an hard-coded array of SFs (c.f. Table 3.3).</td>
</tr>
<tr>
<td>CST</td>
<td>Set the same SF to be applied for each iterations.</td>
</tr>
<tr>
<td>LTE</td>
<td>Select a 0.75 SF.</td>
</tr>
<tr>
<td>LTE_VEC</td>
<td>Select a 0.75 vectorized SF (faster than LTE).</td>
</tr>
</tbody>
</table>

Table 3.4: Hard-coded array of SFs.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>0.30</td>
</tr>
<tr>
<td>4</td>
<td>0.40</td>
</tr>
<tr>
<td>5</td>
<td>0.70</td>
</tr>
<tr>
<td>6</td>
<td>0.80</td>
</tr>
<tr>
<td>7</td>
<td>0.90</td>
</tr>
<tr>
<td>8</td>
<td>0.95</td>
</tr>
</tbody>
</table>
--dec-sub-max

Please refer to the RSC –dec-max parameter.

References

Turbo DB Puncturer parameters

--pct-type

Type  text
Allowed values  NO TURBO_DB
Default  NO
Examples  --pct-type NO

Select the puncturer type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Disable the puncturer.</td>
</tr>
<tr>
<td>TURBO_DB</td>
<td>Enable the puncturer.</td>
</tr>
</tbody>
</table>

--pct-fra-size, -N

Type  integer
Examples  --pct-fra-size 1

Set the frame size $N$. This is not necessarily the codeword size if a puncturing pattern is used.

The puncturer supports $R = 2/5$, $R = 1/2$, $R = 2/3$ and $R = 4/5$ with $R = K/N$.

Codec TPC (Turbo Product Code)

The TPC is an alliance of two crossed BCH codes. The same BCH code is used for columns and rows.

TPC Encoder parameters

--enc-sub-cw-size, -N  **REQUIRED**

Type  integer
Examples  --enc-sub-cw-size 127

Set the codeword size $N$.

Give the sub-encoder code codeword size. You can extend this codeword with a parity bit with the –enc-ext option. Then the codeword size of the TPC is the square of this value.
**--enc-sub-info-bits, --K** "REQUIRED"

*Type* integer

*Examples* --enc-sub-info-bits 120

Set the number of information bits \( K \).

Give the sub-encoder code input size (number of information bits). Then the number of information bits of the TPC is the square of this value.

**--enc-type**

*Type* text

*Allowed values* TPC AZCW COSET USER

*Default* TPC

*Examples* --enc-type AZCW

Select the encoder type.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC</td>
<td>The TPC encoder.</td>
</tr>
<tr>
<td>AZCW</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>COSET</td>
<td>See the common --enc-type parameter.</td>
</tr>
<tr>
<td>USER</td>
<td>See the common --enc-type parameter.</td>
</tr>
</tbody>
</table>

**--enc-sub-type**

Please refer to the BCH --enc-type parameter.

**--enc-ext**

Extend the sub-encoder codeword with a parity bit in order to increase the distance of the code.

**TPC Decoder parameters**

The TPC decoder first decodes columns once with the Chase-Pyndiah algorithm, then rows, and columns again then rows again and so on.

Let’s say \( C \) is the \( N \times N \) a priori matrix from the demodulator.

Let’s say \( R_{c_i+1} \) is the \( N \times N \) a posteriori matrix computed by this decoder after the \( i^{th} \) iteration on the columns. Initially, \( R_{c_0} = C \).

Let’s say \( R_{r_i+1} \) is the \( N \times N \) a posteriori matrix computed by this decoder after the \( i^{th} \) iteration on the rows, with \( R_{r_i} = R_{r_{i+1}} \).

The process of the columns for the \( i^{th} \) iteration gives:

\[
R_{c_{i+1}} = \alpha_{2i+0} \cdot W_{c_i} + C
\]
with $W_i^c$ the extrinsic from the Chase-Pyndiah decoder computed on $R_i^c$.

The process of the rows for the $i^{th}$ iteration gives:

$$R_{i+1}^c = \alpha_{2i+1} W_i^c + C$$

with $W_i^c$ the extrinsic from the Chase-Pyndiah decoder computed on $R_i^c$.

Parameter $\alpha$ is set with the argument --dec-alpha.

```
--dec-sub-cw-size, -N [REQUIRED]
```

Type integer

Examples --dec-sub-cw-size 1

Set the codeword size $N$.

```
--dec-sub-info-bits, -K [REQUIRED]
```

Type integer

Examples --dec-sub-info-bits 1

Set the number of information bits $K$.

```
--dec-type, -D
```

Type text

Allowed values CHASE CP ML

Default CP

Examples --dec-type CP

Select the decoder algorithm.

This algorithm will decode each column and row of the TPC.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td>Decode with the Chase-Pyndiah algorithm of the TPC</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common --dec-type, -D parameter.</td>
</tr>
</tbody>
</table>

The CP algorithm is the implementation of [Pyndiah1998] but in a more generic way in order to let the user chose its configuration:

- **Chase step**: find the more reliable codeword $D$:
  - Take hard decision $H$ on input $R$.
  - Select the $p$ (set with --dec-p) least reliable positions from $R$ to get a metric set $P$ of $p$ elements.
  - Create $t$ (set with --dec-t) test vectors from test patterns.
- Hard decode with the sub-decoder to get the competitors with good syndrome set $C$.
- Remove competitors from $C$ to keep $c$ of them (set with --dec-c).
- Compute the metrics $C_m$ (euclidean distance) of each competitor compared to $H$.
- Select the competitors with the smallest metric to get the decided word $D$ with a metric $D_m$ and where:

$$D_j = \begin{cases} +1 & \text{when } H_j = 0 \\ -1 & \text{when } H_j = 1 \end{cases}$$

- **Pyndiah step: compute reliabilities of each bit of $D$**
  - $a, b, c, d$ and $e$ are simulation constants changeable by the user with --dec-cp-coef
  - Compute the reliability $F$ of $D$ for each bit $D_j$ of the word:
    * Find $C^*$ the competitor with the smallest metric $C_m$ that have $C^*_j \neq D_j$.
      * when $C^*$ exists:
        $$F_j = b.D_j.(C_m - D_m)$$
      * when $C^*$ does not exist and if --dec-beta is given:
        $$F_j = D_j.beta$$
    * else:
      $$F_j = D_j.\left[\sum_{i=0}^{e} P_i - c.D_m + d.|R_j|\right]$$

- Compute extrinsic $W = F - a.R$

--dec-implem

**Type** text

**Allowed values** STD

**Default** STD

**Examples** --dec-implem STD

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>A standard implementation</td>
</tr>
</tbody>
</table>

--dec-ite, -i

**Type** integer

**Default** 4

**Examples** --dec-ite 8

Set the number of iterations in the turbo decoding process.
--dec-alpha

**Type** list of real numbers

**Default** all at 0.5

**Examples** --dec-alpha "0.1,0.1,0.2,0.25,0.3,0.35,.5,.5,1.2"

Give the *weighting factor* alpha, one by half iteration (so twice more than the number of iterations).
The first one is for the first columns process, the second for the first rows process, the third for the second columns process, the fourth for the second rows process, and so on.
If there are not enough values, then the last one given is automatically extended to the rest of the half-iterations. Conversely, if there are too many, the surplus is truncated.

--dec-beta

**Type** list of real numbers

**Examples** --dec-beta "0.1,0.1,0.2,0.25,0.3,0.35,.5,.5,1.2"

Give the *reliability factor* beta, one by half iteration (so twice more than the number of iterations).
The first one is for the first columns process, the second for the first rows process, the third for the second columns process, the fourth for the second rows process, and so on.
If there are not enough values, then the last one given is automatically extended to the rest of the half-iterations. Conversely, if there are too many, the surplus is truncated.
If not given, then beta is dynamically computed as described in --dec-type, -D.

--dec-c

**Type** integer

**Default** 0

**Examples** --dec-c 3

Set the *number of competitors*. A value of 0 means that the latter is set to the number of test vectors, 1 means only the decided word.

--dec-p

**Type** integer

**Default** 2

**Examples** --dec-p 1

Set the number of *least reliable positions*.

--dec-t

**Type** integer

**Default** 0
Examples --dec-t 1

Set the number of test vectors. A value of 0 means equal to $2^p$ where $p$ is the number of least reliable positions.

--dec-cp-coef

Type list of real numbers

Default "1,1,1,1,0"

Examples --dec-cp-coef "0,0.25,0,0,3"

Give the 5 CP constant coefficients $a, b, c, d, e$.

See the --dec-type, -D parameter.

--dec-sub-type, -D

Please refer to the BCH --dec-type, -D parameter.

--dec-sub-corr-pow, -T

Please refer to the BCH --dec-corr-pow, -T parameter.

--dec-sub-implem

Please refer to the BCH --dec-implem parameter.

References

Codec Uncoded

Uncoded Encoder parameters

There is no encoder when running an uncoded simulation.

Uncoded Decoder parameters

--dec-type, -D

Type text

Allowed values NONE CHASE ML

Default NONE

Examples --dec-type CHASE

3.2. Parameters
Select the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>Select the NONE decoder.</td>
</tr>
<tr>
<td>CHASE</td>
<td>See the common (-dec-type, -D) parameter.</td>
</tr>
<tr>
<td>ML</td>
<td>See the common (-dec-type, -D) parameter.</td>
</tr>
</tbody>
</table>

\(--dec-implem\)

Type text

Allowed values HARD_DECISION

Default HARD_DECISION

Examples --dec-implem HARD_DECISION

Select the implementation of the decoder algorithm.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARD_DECISION</td>
<td>Take the hard decision on the input LLRs.</td>
</tr>
</tbody>
</table>

3.2.5 Interleaver parameters

The interleaving process is frequent in coding schemes. It can be found directly in the code definition (for instance in Turbo or Turbo Product codes) or in larger schemes like for the turbo demodulation in the receiver (see the iterative BER/FER chain in Fig. 3.5).

\(--itl-type\)

Type text

Allowed values CCSDS COL_ROW DVB-RCS1 DVB-RCS2 GOLDEN LTE NO RANDOM RAND_COL ROW_COL_USER

Default RANDOM

Examples --itl-type RANDOM

Select the interleaver type.

Description of the allowed values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Disable the interleaving process: the output is the input (Fig. 3.8).</td>
</tr>
<tr>
<td>COL_ROW</td>
<td>Fill the interleaver by column, read it by row (can be customized with the \textit{--itl-read-order} parameter) (Fig. 3.9).</td>
</tr>
<tr>
<td>ROW_COL</td>
<td>Fill the interleaver by row, read it by column (can be customized with the \textit{--itl-read-order} parameter) (Fig. 3.10).</td>
</tr>
<tr>
<td>RANDOM</td>
<td>Generate a random sequence for the entire frame (based on the MT 19937 PRNG [MN98]) (Fig. 3.11).</td>
</tr>
<tr>
<td>RAND_COL</td>
<td>Generate multiple random sequences decomposed in independent columns (based on the MT 19937 PRNG [MN98]) (Fig. 3.12).</td>
</tr>
<tr>
<td>GOLDEN</td>
<td>Select the interleaver described in [CLGH99].</td>
</tr>
<tr>
<td>CCSDS</td>
<td>Select the interleaver defined in the CCSDS standard.</td>
</tr>
<tr>
<td>LTE</td>
<td>Select the interleaver defined in the LTE standard.</td>
</tr>
<tr>
<td>DVB–RCS1</td>
<td>Select the interleaver defined in the DVB–RCS1 standard.</td>
</tr>
<tr>
<td>DVB–RCS2</td>
<td>Select the interleaver defined in the DVB–RCS2 standard.</td>
</tr>
<tr>
<td>USER</td>
<td>Select the interleaver sequence (LUT (Look Up Table)) from an external file (to use with the \textit{--itl-path} parameter) (Fig. 3.13).</td>
</tr>
</tbody>
</table>

**Fig. 3.8:** Interleaver NO.

**Fig. 3.9:** Interleaver COL_ROW.
Fig. 3.10: Interleaver ROW_COL.

Fig. 3.11: Interleaver RANDOM.

Fig. 3.12: Interleaver RAND_COL.
--itl-cols

Type integer
Default 4
Examples --itl-cols 1
Specify the number of columns used for the RAND_COL, ROW_COL or COL_ROW interleavers.

--itl-path

Type file
Rights read only
Examples --itl-path conf/itl/GSM-LDPC_4224.itl
Set the file path to the interleaver LUT (to use with the USER interleaver).
An ASCII file is expected:

```
# the number of LUTs contained in the file (only one LUT here)
1

# the frame size 'N'
16

# the LUT definition (here the frame is reversed, 0 becomes 15, 1 becomes 14, etc.)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

If there is more than one interleaved sequence then for each new frame a new LUT is used in the natural order given by the file. Here is an example with two LUTs (Look Up Tables):

```
# the number of LUTs contained in this file
2

# the frame size 'N'
16

# first and second LUTs definition
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8
```

3.2. Parameters
Note: When the number of simulated frames exceeds the number of LUT contained in the files, the LUTs from the beginning of the file are reused and this is repeated until the end of the simulation.

--itl-read-order

Type: text

Allowed values: BOTTOM_LEFT BOTTOM_RIGHT TOP_LEFT TOP_RIGHT

Examples: --itl-read-order BOTTOM_LEFT

Change the read order of the COL_ROW and ROW_COL interleavers.

The read starts from the given corner of the array to the diagonally opposite one. The read is made row by row for the COL_ROW interleaver and column by column for the ROW_COL one.

Description of the allowed values (see also the figures just below):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP_LEFT</td>
<td>Read is down from the top left corner to the bottom right corner.</td>
</tr>
<tr>
<td>TOP_RIGHT</td>
<td>Read is down from the top right corner to the bottom left corner.</td>
</tr>
<tr>
<td>BOTTOM_LEFT</td>
<td>Read is down from the bottom left corner to the top right corner.</td>
</tr>
<tr>
<td>BOTTOM_RIGHT</td>
<td>Read is down from the bottom right corner to the top left corner.</td>
</tr>
</tbody>
</table>

Fig. 3.14 depicts the read order options on the COL_ROW interleaver.

Fig. 3.15 depicts the read order options on the ROW_COL interleaver.
Fig. 3.15: Interleaver \texttt{ROW\_COL} read orders.
**--itl-seed**

*Type* integer  
*Default* 0  
*Examples* --itl-seed 48

Select the seed used to initialize the PRNG.  
All the threads/nodes have the same seed (except if a uniform interleaver is used, see the –itl-uni parameter).

**Note:** This parameter has no effect if the selected interleaver is not randomly generated.

**--itl-uni**

Enable to generate a new LUT *for each new frame* (i.e. uniform interleaver).  
By default, if this parameter is not used, the random interleavers generate the LUT only once for the whole simulation.

**Note:** This parameter has no effect if the selected interleaver is not randomly generated.

**References**

**3.2.6 Modem parameters**

AFF3CT comes with a set of predefined MODEMS (modulators/demodulators). A MODEM (modulator/demodulator) transforms a sequence of bits into a suitable form for the transmission on a physical medium. In the AFF3CT “philosophy”, the MODEM is a *module* containing three *tasks*: modulate, filter and demodulate (read the Philosophy section for more information about modules and tasks).

**--mdm-type**

*Type* text  
*Allowed values* BPSK CPM OOK PAM PSK QAM SCMA USER  
*Default* BPSK  
*Examples* --mdm-type SCMA

Select the modulation type.  
Description of the allowed values:
## Parameters

### --mdm-implem

**Type** text

**Allowed values** FAST STD

**Default** STD

**Examples** --mdm-implem FAST

Select the MODEM implementation.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select a standard implementation working for any MODEM.</td>
</tr>
<tr>
<td>FAST</td>
<td>Select a fast implementation, only available for the BPSK MODEM at this time.</td>
</tr>
</tbody>
</table>

### --mdm-bps

**Type** integer

**Default** 1

**Examples** --mdm-bps 1

Set the number of bits used to generate a symbol (BPS (Bit Per Symbol)).

This parameter has no effect on the BPSK and OOK MODEMS where the BPS is forced to 1. This is the same for the SCMA MODEM where the BPS is forced to 3.

**Note:** For the QAM MODEM, only even BPS values are supported.

### --mdm-const-path

**Type** file

**Rights** read/write

**Examples** --mdm-const-path conf/mod/16QAM_ANTI_GRAY.mod

Give the path to the ordered modulation symbols (constellation), to use with the USER MODEM.

An ASCII file is expected, for instance here is the definition of a 16-QAM with an anti-Gray mapping (the lines starting with a # are ignored):
The number of bits per symbol is automatically computed from the number of given symbols. The latter has to be a power of 2.

---

**--mdm-max**

Type: text

Allowed values: MAX S MAXS MAXL MAX

Examples: --mdm-max MAX

Select the approximation of the max* operator used in the PAM, QAM, PSK, CPM and user demodulators.

Description of the allowed values:
### Value Description

| MAXS       | \( \max^*(a, b) = \max(a, b) + \log(1 + \exp(-|a - b|)) \) |
| MAXSS      | \( \max^*(a, b) \approx \max(a, b) + \left\{ \begin{array}{ll} 0 & \text{if } d \geq 37 \\ \exp(-|a - b|) & \text{if } 9 \leq d < 37 \\ \log(1 + \exp(-|a - b|)) & \text{else} \end{array} \right. \) |
| MAXL       | \( \max^*(a, b) \approx \max(a, b) + \max(0, 0.301 - (0.5|a - b|)) \) |
| MAX        | \( \max^*(a, b) \approx \max(a, b) \) |

**MAXS** for *Max Star* is the exact \( \max^* \) operator. **MAXSS** for *Max Star Safe* allows to avoid numeric instabilities due to the exponential operation and the limited precision of the floating-point representation. **MAXL** for *Max Linear* is a linear approximation of the \( \max^* \) function. **MAX** for *Max* is the simplest \( \max^* \) approximation with only a \( \max \) function.

**--mdm-no-sig2**

Turn off the division by \( \sigma^2 \) in the demodulator where \( \sigma \) is the Gaussian noise variance.

**--mdm-cpm-k**

- **Type** integer
- **Default** 1
- **Examples** --mdm-cpm-k 1

Set the CPM index numerator.

**--mdm-cpm-p**

- **Type** integer
- **Default** 2
- **Examples** --mdm-cpm-p 1

Set the CPM index denominator.

**--mdm-cpm-L**

- **Type** integer
- **Default** 2
- **Examples** --mdm-cpm-L 1

Set the CPM pulse width (also called memory depth).

**--mdm-cpm-upf**

- **Type** integer
- **Default** 1
- **Examples** --mdm-cpm-upf 1

Select the symbol upsampling factor in the CPM.
--mdm-cpm-map

Type text
Allowed values GRAY NATURAL
Default NATURAL
Examples --mdm-cpm-map GRAY

Select the CPM symbols mapping layout.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAY</td>
<td>Gray code switching only one bit at a time from a symbol to the following.</td>
</tr>
<tr>
<td>NATURAL</td>
<td>The natural binary code incrementing the value from a symbol to the next one.</td>
</tr>
</tbody>
</table>

--mdm-cpm-ws

Type text
Allowed values GMSK RCOS REC
Default GMSK
Examples --mdm-cpm-ws GMSK

Select the CPM wave shape.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMSK</td>
<td>Gaussian Minimum Shift Keying</td>
</tr>
<tr>
<td>RCOS</td>
<td>Raised COSinus</td>
</tr>
<tr>
<td>REC</td>
<td>RECtangular</td>
</tr>
</tbody>
</table>

--mdm-cpm-std

Type text
Allowed values GSM
Examples --mdm-cpm-std GSM

Set the CPM parameters according to a standard.

Description of the allowed values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSM</td>
<td>-mdm-bps</td>
<td>1</td>
<td>Bit per symbol.</td>
</tr>
<tr>
<td></td>
<td>-mdm-cpm-upf</td>
<td>5</td>
<td>Upsampling factor.</td>
</tr>
<tr>
<td></td>
<td>-mdm-cpm-k</td>
<td>1</td>
<td>Modulation index numerator.</td>
</tr>
<tr>
<td></td>
<td>-mdm-cpm-p</td>
<td>2</td>
<td>Modulation index denominator.</td>
</tr>
<tr>
<td></td>
<td>-mdm-cpm-L</td>
<td>3</td>
<td>Memory depth.</td>
</tr>
<tr>
<td></td>
<td>-mdm-cpm-map</td>
<td>NATURAL</td>
<td>Mapping layout.</td>
</tr>
<tr>
<td></td>
<td>-mdm-cpm-ws</td>
<td>GMSK</td>
<td>Wave shape.</td>
</tr>
</tbody>
</table>

**Note:** When this parameter is used, if you set any of the other MODEM parameters, it will override the configuration from the standard.

```bash
--mdm-ite
```

**Type** integer  
**Default** 1  
**Examples** `--mdm-ite 5`

Set the number of iterations in the SCMA demodulator.

```bash
--mdm-psi
```

**Type** text  
**Allowed values** PSI0 PSI1 PSI2 PSI3  
**Examples** `--mdm-psi PSI0`

Select the $\psi$ function used in the SCMA demodulator.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSI0</td>
<td>$\psi_0 = \exp\left(\frac{-</td>
</tr>
<tr>
<td>PSI1</td>
<td>$\psi_1 \approx \psi_0 \approx \frac{1}{</td>
</tr>
<tr>
<td>PSI2</td>
<td>$\psi_2 \approx \psi_0 \approx \frac{8(</td>
</tr>
<tr>
<td>PSI3</td>
<td>$\psi_3 \approx \psi_0 \approx \frac{8(</td>
</tr>
</tbody>
</table>

Where $n_0 = \begin{cases} 1 & \text{if } \sigma^2 \text{ is disabled} \\ 4\sigma^2 & \text{else} \end{cases}$.

See the `--mdm-no-sig2` parameter to disable the division by $\sigma^2$.

### 3.2. Parameters
Give the path to the codebook, to use with the SCMA MODEM.

**Note:** Only 3 BPS codebook symbols are supported at this time.

### Codebook format

A codebook is designed for a **number_of_users** $V$, a **number_of_orthogonal_resources** $K$ and **codebook_size** $M$.

The codebook file then looks as a table of $V \times K$ rows and $2 \cdot M$ columns (real and imaginary parts):

<table>
<thead>
<tr>
<th>V</th>
<th>K</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re(User 1, Resource 1, Code 1)</td>
<td>Im(User 1, Resource 1, Code 1)</td>
<td>...</td>
</tr>
<tr>
<td>Re(User 1, Resource K, Code 1)</td>
<td>Im(User 1, Resource K, Code 1)</td>
<td>...</td>
</tr>
<tr>
<td>Re(User 2, Resource 1, Code 1)</td>
<td>Im(User 2, Resource 1, Code 1)</td>
<td>...</td>
</tr>
<tr>
<td>Re(User 2, Resource K, Code 1)</td>
<td>Im(User 2, Resource K, Code 1)</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Descriptions of the codebooks of the configuration files

Codebooks are normalized, so the average power of signal will be equal to 1.

<table>
<thead>
<tr>
<th>Codebook Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS1</td>
<td>From [Pro]</td>
</tr>
<tr>
<td>CS2</td>
<td>LDS based on QPSK constellation</td>
</tr>
<tr>
<td>CS3</td>
<td>Based on [CWC15] and own optimization for AWGN channel</td>
</tr>
<tr>
<td>CS4</td>
<td>From [ZXX+16]</td>
</tr>
<tr>
<td>CS5</td>
<td>From [SWC17]</td>
</tr>
<tr>
<td>CS6</td>
<td>From [KS17]</td>
</tr>
<tr>
<td>CS7</td>
<td>From [KS17]</td>
</tr>
<tr>
<td>CS8</td>
<td>From [WZC15]</td>
</tr>
</tbody>
</table>
The simulation results for CS1-CS7 (AWGN and Rayleigh fading channels) can be found in [KS17]. The simulation results for CS8 can be found in [KS16] (defined as CS2 in the paper).

```bash
--mdm-rop-est
```

*Type* integer

*Default* 0

*Examples* --mdm-rop-est 256

Set the number of known bits for the ROP estimation in the OOK demodulator on an optical channel.

The estimation is done from a known set of bits that is the output of the modulation. If left to 0, the demodulation is done with the exact applied ROP in the channel.

**References**

**3.2.7 Channel parameters**

The channel represents the physical support such as optical fiber, space, water, air, etc. It is during the passage in the channel that the frames are altered/noised and errors can occur. The channel coding theory has been invented to correct errors induced by the channel (or at least reduce the number of errors to an acceptable rate).

```bash
--chn-type
```

*Type* text

*Allowed values* NO BEC BSC AWGN RAYLEIGH RAYLEIGH_USER OPTICAL USER

*Default* AWGN

*Examples* --chn-type AWGN

Select the channel type.

Description of the allowed values:
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>Disable the channel noise, ( Y = X ).</td>
</tr>
</tbody>
</table>
| BEC   | Select the Binary Erasure Channel (BEC):  
\[
Y_i = \begin{cases} 
\text{erased} & \text{if } e = 1 \\
X_i & \text{else}
\end{cases}, \text{ with } P(e = 1) = p_e \text{ and } P(e = 0) = 1 - p_e.
\] |
| BSC   | Select the Binary Symmetric Channel (BSC):  
\[
Y_i = \begin{cases} 
\neg X_i & \text{if } e = 1 \\
X_i & \text{else}
\end{cases}, \text{ with } P(e = 1) = p_e \text{ and } P(e = 0) = 1 - p_e.
\] |
| AWGN  | Select the Additive White Gaussian Noise (AWGN) channel:  
\[
\sigma^2 X_i + \mathcal{N}(0, \sigma^2).
\] |
Where:

- $\sigma$ is the Gaussian noise variance, $p_e$ is the event probability and ROP is the Received optical power of the simulated noise points. They are given by the user through the `-sim-noise-range, -R` argument.

- $X$ is the original modulated frame and $Y$ the noisy output.

- $\mathcal{N}(\mu, \sigma^2)$ is the Normal or Gaussian distribution.

- $\mathcal{U}(a, b)$ is the Uniform distribution.

For the OPTICAL channel, the CDF (Cumulative Distribution Function) are computed from the given PDF with the `--sim-pdf-path` argument. This file describes the latter for the different ROP. There must be a PDF for a bit transmitted at 0 and another for a bit transmitted at 1.

**Note:** The NO, AWGN and RAYLEIGH channels handle complex modulations.

---

**Warning:** The BEC, BSC and OPTICAL channels work only with the OOK modulation (see the `--mdm-type` parameter).

---

**--chn-implem**

Type  text

Allowed values  STD FAST GSL MKL

Default  STD

Examples  `--chn-implem FAST`

Select the implementation of the algorithm to generate the noise.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select the standard implementation based on the C++ standard library.</td>
</tr>
<tr>
<td>FAST</td>
<td>Select the fast implementation (handwritten and optimized for SIMD architectures).</td>
</tr>
<tr>
<td>GSL</td>
<td>Select an implementation based of the GSL (GNU Scientific Library).</td>
</tr>
<tr>
<td>MKL</td>
<td>Select an implementation based of the MKL (Intel Math Kernel Library) (only available for x86 architectures).</td>
</tr>
</tbody>
</table>

**Note:** All the proposed implementations are based on the MT 19937 PRNG algorithm [MN98]. The Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$ is implemented with the Box-Muller method [BM+58] except when using the GSL where the Ziggurat method [MT00] is used instead.

---

**Attention:** To enable the GSL or the MKL implementations, you need to have those libraries installed on your system and to turn on specific **CMake Options**.

---

The Table 3.5, Table 3.6 and Table 3.7 present the throughputs of the different channel implementations depending on the frame size. The testbed for the experiments is an *Intel(R) Xeon(R) CPU E3-1270 v5 @ 3.60GHz* 8 threads CPU (Central Process Unit).
Table 3.5: Comparison of the AWGN channel implementations
(throughputs are in Mb/s).

<table>
<thead>
<tr>
<th>Frame size</th>
<th>STD</th>
<th>FAST</th>
<th>MKL</th>
<th>GSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>31.80</td>
<td>99.12</td>
<td>29.80</td>
<td>41.39</td>
</tr>
<tr>
<td>32</td>
<td>30.05</td>
<td>134.94</td>
<td>53.72</td>
<td>46.76</td>
</tr>
<tr>
<td>64</td>
<td>30.78</td>
<td>165.92</td>
<td>93.80</td>
<td>52.79</td>
</tr>
<tr>
<td>128</td>
<td>31.24</td>
<td>188.06</td>
<td>148.31</td>
<td>54.77</td>
</tr>
<tr>
<td>256</td>
<td>31.41</td>
<td>199.14</td>
<td>204.53</td>
<td>55.38</td>
</tr>
<tr>
<td>512</td>
<td>31.52</td>
<td>199.43</td>
<td>267.74</td>
<td>55.49</td>
</tr>
<tr>
<td>1024</td>
<td>31.79</td>
<td>199.71</td>
<td>331.71</td>
<td>56.15</td>
</tr>
<tr>
<td>2048</td>
<td>31.61</td>
<td>200.16</td>
<td>342.06</td>
<td>56.32</td>
</tr>
<tr>
<td>4096</td>
<td>31.88</td>
<td>198.88</td>
<td>343.40</td>
<td>57.42</td>
</tr>
<tr>
<td>8192</td>
<td>30.43</td>
<td>195.78</td>
<td>342.59</td>
<td>56.92</td>
</tr>
</tbody>
</table>

Table 3.6: Comparison of the BEC/BSC channel implementations
(throughputs are in Mb/s).

<table>
<thead>
<tr>
<th>Frame size</th>
<th>STD</th>
<th>FAST</th>
<th>MKL</th>
<th>GSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>36.18</td>
<td>114.87</td>
<td>104.1</td>
<td>58.92</td>
</tr>
<tr>
<td>32</td>
<td>40.28</td>
<td>170.64</td>
<td>184.99</td>
<td>69.14</td>
</tr>
<tr>
<td>64</td>
<td>42.84</td>
<td>223.78</td>
<td>319.65</td>
<td>77.26</td>
</tr>
<tr>
<td>128</td>
<td>43.28</td>
<td>252.41</td>
<td>474.18</td>
<td>87.78</td>
</tr>
<tr>
<td>256</td>
<td>43.42</td>
<td>272.82</td>
<td>624.92</td>
<td>93.71</td>
</tr>
<tr>
<td>512</td>
<td>43.51</td>
<td>273.22</td>
<td>738.72</td>
<td>95.36</td>
</tr>
<tr>
<td>1024</td>
<td>43.64</td>
<td>275.41</td>
<td>865.25</td>
<td>97.84</td>
</tr>
<tr>
<td>2048</td>
<td>43.63</td>
<td>272.78</td>
<td>996.88</td>
<td>97.25</td>
</tr>
<tr>
<td>4096</td>
<td>42.78</td>
<td>274.71</td>
<td>1109.13</td>
<td>97.65</td>
</tr>
<tr>
<td>8192</td>
<td>43.67</td>
<td>272.71</td>
<td>1116.41</td>
<td>98.47</td>
</tr>
</tbody>
</table>

Table 3.7: Comparison of the optical channel implementations (throughputs are in Mb/s).

<table>
<thead>
<tr>
<th>Frame size</th>
<th>STD</th>
<th>FAST</th>
<th>MKL</th>
<th>GSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>6.69</td>
<td>7.56</td>
<td>6.71</td>
<td>6.53</td>
</tr>
<tr>
<td>32</td>
<td>9.89</td>
<td>10.98</td>
<td>10.28</td>
<td>9.56</td>
</tr>
<tr>
<td>64</td>
<td>12.67</td>
<td>14.30</td>
<td>14.05</td>
<td>12.15</td>
</tr>
<tr>
<td>128</td>
<td>14.40</td>
<td>16.33</td>
<td>16.33</td>
<td>15.88</td>
</tr>
<tr>
<td>256</td>
<td>15.82</td>
<td>17.74</td>
<td>18.22</td>
<td>15.07</td>
</tr>
<tr>
<td>512</td>
<td>16.52</td>
<td>18.46</td>
<td>18.29</td>
<td>15.79</td>
</tr>
<tr>
<td>1024</td>
<td>17.18</td>
<td>19.14</td>
<td>19.31</td>
<td>16.19</td>
</tr>
<tr>
<td>2048</td>
<td>16.96</td>
<td>18.76</td>
<td>20.30</td>
<td>16.42</td>
</tr>
<tr>
<td>4096</td>
<td>17.19</td>
<td>18.65</td>
<td>20.29</td>
<td>16.47</td>
</tr>
<tr>
<td>8192</td>
<td>17.26</td>
<td>18.98</td>
<td>20.58</td>
<td>16.63</td>
</tr>
</tbody>
</table>

**Note:** The reported values are the *average* throughputs given by the simulator integrated statistics tool (see the `--sim-stats` parameter).
**--chn-blk-fad**

*Type* text  
*Allowed values* NOFRAME ONETAP  
*Default* 1  
*Examples* --chn-gain-occur 10  

Set the block fading policy for the Rayleigh channel.

*Note:* At this time the FRAME and ONETAP block fading are not implemented.

**--chn-gain-occur**

*Type* integer  
*Default* 1  
*Examples* --chn-gain-occur 10  

Give the number of times a gain is used on consecutive symbols. It is used in the RAYLEIGH_USER channel while applying gains read from the given file.

**--chn-path**

*Type* file  
*Rights* read  
*Examples* --chn-path example/path/to/the/right/file  

Give the path to a file containing the noise. The expected type of noise vary depending of the channel type (see the --chn-type parameter for more details):

- **USER**: the file must contain frame with the noise applied on it,
- **USER_ADD**: the file must contain only the noise $Z$,
- **RAYLEIGH_USER**: the file must contain the gain values $H$.

The expected file format is either ASCII or binary (the format is automatically detected). Here is the file structure expected in ASCII:

```
# 'F' has to be replaced by the number of contained frames.
F

# 'N' has to be replaced by the frame size.
N

# a sequence of 'F * N' floating-point values (separated by spaces)
Y_0 Y_1 Y_2 Y_3 Y_4 [...] Y_{F*N-1}
```

In binary mode, $F$ and $N$ have to be 32-bit unsigned integers. The next $F \times N$ floating-point values can be either in 32-bit or in 64-bit.
3.2.8 Quantizer parameters

The quantizer is a module that ensures the real numbers transformation from a floating-point representation to a fixed-point representation. This module is enabled only if the receiver part of the communication chain works on a fixed-point representation (cf. the \texttt{--sim-prec}, \texttt{-p} parameter).

\begin{quote}
\textbf{Warning:} Some decoders are not \textit{fixed-point ready}, please refer to the decoders documentation for more details.
\end{quote}

\begin{itemize}
\item \texttt{--qnt-type} \\
\hspace{1em} \textbf{Type}: text \\
\hspace{1em} \textbf{Allowed values}: CUSTOM POW2 \\
\hspace{1em} \textbf{Default}: POW2 \\
\hspace{1em} \textbf{Examples}: \texttt{--qnt-type CUSTOM}
\end{itemize}

Select the quantizer type.

Description of the allowed values ($Y_i$ stands for a floating-point representation and $Q_i$ for the fixed-point representation of a real number):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| CUSTOM | $Q_i = \left\{ \begin{array}{ll}
+v_{sat} & \text{when } v_i > +v_{sat} \\
-v_{sat} & \text{when } v_i < -v_{sat}, \text{ with } v_i = \left\lfloor \frac{Y_i}{\Delta} \right\rfloor \text{ and } v_{sat} = 2^{p_b-1} - 1 \text{ and } \Delta = \frac{p_r}{v_{sat}} \\
v_i & \text{else}
\end{array} \right.$ |
| POW2 | $Q_i = \left\{ \begin{array}{ll}
+v_{sat} & \text{when } v_i > +v_{sat} \\
-v_{sat} & \text{when } v_i < -v_{sat}, \text{ with } v_i = \left\lfloor Y_i * F \right\rfloor \text{ and } v_{sat} = 2^{p_b-1} - 1 \text{ and } F = 2^{p_d}
\end{array} \right.$ |

Where $p_r$, $p_b$ and $p_d$ are respectively given through the \texttt{--qnt-range}, \texttt{--qnt-bits} and \texttt{--qnt-dec} parameters.

\begin{itemize}
\item \texttt{--qnt-implem} \\
\hspace{1em} \textbf{Type}: text \\
\hspace{1em} \textbf{Allowed values}: STD FAST \\
\hspace{1em} \textbf{Default}: STD \\
\hspace{1em} \textbf{Examples}: \texttt{--qnt-implem FAST}
\end{itemize}

Select the implementation of the quantizer.

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select a standard implementation.</td>
</tr>
<tr>
<td>FAST</td>
<td>Select a fast implementation, only available for the POW2 quantizer.</td>
</tr>
</tbody>
</table>
--qnt-range

Type  real number

Examples  --qnt-range 1.0

Select the min/max bounds for the CUSTOM quantizer.

--qnt-bits

Type  integer

Default  8 else see Table 3.8

Examples  --qnt-bits 1

Set the number of bits used in the fixed-point representation.

Note:  If the amplitude of the current number exceeds the maximum amplitude that can be represented with the current quantization, then a saturation is applied (c.f. the --qnt-type parameter).

Table 3.8: Default values of the total number of bits for the different codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPC</td>
<td>6</td>
</tr>
<tr>
<td>POLAR</td>
<td>6</td>
</tr>
<tr>
<td>REP</td>
<td>6</td>
</tr>
<tr>
<td>RSC</td>
<td>6</td>
</tr>
<tr>
<td>RSC_DB</td>
<td>6</td>
</tr>
<tr>
<td>TPC</td>
<td>6 on 8-bit and 8 on 16-bit</td>
</tr>
<tr>
<td>TURBO</td>
<td>6</td>
</tr>
<tr>
<td>TURBO_DB</td>
<td>6</td>
</tr>
</tbody>
</table>

--qnt-dec

Type  integer

Default  3 else see Table 3.9

Examples  --qnt-dec 1

Set the position of the decimal point in the quantified representation.
Table 3.9: Default values of the decimal point position for the different codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDPC</td>
<td>2</td>
</tr>
<tr>
<td>POLAR</td>
<td>1</td>
</tr>
<tr>
<td>REP</td>
<td>2</td>
</tr>
<tr>
<td>RSC</td>
<td>1 on 8-bit and 3 on 16-bit</td>
</tr>
<tr>
<td>RSC_DB</td>
<td>1 on 8-bit and 3 on 16-bit</td>
</tr>
<tr>
<td>TPC</td>
<td>2 on 8-bit and 3 on 16-bit</td>
</tr>
<tr>
<td>TURBO</td>
<td>2 on 8-bit and 3 on 16-bit</td>
</tr>
<tr>
<td>TURBO_DB</td>
<td>2 on 8-bit and 3 on 16-bit</td>
</tr>
</tbody>
</table>

3.2.9 Monitor parameters

The monitor is the last module in the chain: it compares the decoded information bits with the initially generated ones from the source. Furthermore, it can also compute the mutual information (MI (Mutual Information)) from the demodulator output.

`--mnt-max-fe, -e`

Type integer

Default 100

Examples `--mnt-max-fe 25`

Set the maximum number of frame errors to simulated for each noise point.

`--mnt-err-hist`

Type integer

Examples `--mnt-err-hist 0`

Enable the construction of the errors per frame histogram. Set also the maximum number of bit errors per frame included in the histogram (0 means no limit).

The histogram is saved in CSV (Comma-Separated Values) format:

```
"Number of error bits per wrong frame"; "Histogram (noise: 5.000000dB, on 10004 frames)"
0; 0
1; 7255
2; 2199
3; 454
4; 84
5; 11
6; 12
```

`--mnt-err-hist-path`

Type file
Path to the output histogram. When the files are dumped, the current noise value is added to this name with the .txt extension.

An output filename example is `hist_2.000000.txt` for a noise value of 2 dB. For Gnuplot users you can then simply display the histogram with the following command:

```
gnuplot -e "set key autotitle columnhead; plot 'hist_2.000000.txt' with lines; pause -1"
```

`--mnt-mutinfo`

Enable the computation of the mutual information (MI).

Note: Only available on BFER simulation types (see the `--sim-type` parameter for more details).

`--mnt-red-lazy`

Enable the lazy synchronization between the various monitor threads.

Using this parameter can significantly reduce the simulation time, especially for short frame sizes when the monitor synchronizations happen very often.

Note: This parameter is not available if the code has been compiled with MPI (Message Passing Interface).

Note: By default, if the `--mnt-red-lazy-freq` parameter is not specified, the interval/frequency is set to the same value than the `--ter-freq` parameter.

Warning: Be careful, this parameter is known to alter the behavior of the `--sim-max-fra, -n` parameter.

`--mnt-red-lazy-freq`

Type integer

Default 1000

Examples `--mnt-red-lazy-freq 200`

Set the time interval (in milliseconds) between the synchronizations of the monitor threads.

Note: This parameter automatically enables the `--mnt-red-lazy` parameter.
--mnt-mpi-comm-freq

**Type**  
integer

**Default**  
1000

**Examples**  
--mnt-mpi-comm-freq 1

Set the time interval (in milliseconds) between the MPI communications. Increase this interval will reduce the MPI communications overhead.

**Note:** Available only when compiling with the MPI support *CMake Options*.

**Note:** When this parameter is specified, the *--ter-freq* parameter is automatically set to the same value except if the *--ter-freq* is explicitly defined.

### 3.2.10 Terminal parameters

The terminal is an observer module that reads and display the monitor informations in real time. The terminal displays two types of results: **intermediate results** and **final results**. The intermediate results are printed on the *error output* during the simulation of a noise point and refreshed at a defined frequency (see the *--ter-freq* parameter). On the other hand, the final results are printed on the *standard output* once the simulation of the noise point is over.

**--ter-type**

**Type**  
text

**Allowed values**  
STD

**Default**  
STD

**Examples**  
--ter-type STD

Select the terminal type (the format to display the results).

Description of the allowed values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Select the standard format.</td>
</tr>
</tbody>
</table>

**Note:** For more details on the standard output format see the *Output section*.)
**--ter-freq**

*Type* integer  
*Default* 500  
*Examples*  

```
--ter-freq 1
```

Set the display frequency (refresh time) of the intermediate results in milliseconds. Setting 0 disables the display of the intermediate results.

**Note:** When MPI is enabled, this value is by default set to the same value than the `--mnt-mpi-comm-freq` parameter.

**--ter-no**

Disable completely the terminal report.

**--ter-sigma**

Show the standard deviation ($\sigma$) of the Gaussian/Normal distribution in the terminal.

**Note:** Work only if the `--sim-noise-type, -E` parameter is set to EBN0 or ESNO.

### 3.2.11 Other parameters

**--help, -h**

Print the help with all the required (denoted as `{R}`) and optional arguments. The latter change depending on the selected simulation type and code.

```
aff3ct -h
```

Usage: `./bin/aff3ct -C <text> [optional args...]`

Simulation parameter(s):

{R}  

--sim-cde-type, -C <text:including set={BCH|LDPC|POLAR|RA|REP|RS|RSC|RSC_DB|TPC|TURBO|TURBO_DB|UNCODED}>

Select the channel code family to simulate.

--sim-prec, -p <integer:including set={8|16|32|64}>

Specify the representation of the real numbers in the receiver part of the chain.

--sim-type <text:including set={BFER|BFERI|EXIT}>

Select the type of simulation (or communication chain skeleton).

Other parameter(s):

--Help, -H

Print the help like with the `--help, -h` parameter plus advanced arguments (denoted as '{A}').

--help, -h

Print the help with all the required (denoted as '{R}') and optional arguments. The latter change depending on the selected simulation type and code.
code.
--no-colors
  Disable the colors in the shell.
--version, -v
  Print informations about the version of the source code and compilation options.

--Help, -H

Print the help like with the --help, -h parameter plus advanced arguments (denoted as {A}).

aff3ct -H

Usage: ./bin/aff3ct -C <text> [optional args...]

Simulation parameter(s):
(R) --sim-cde-type, -C <text:including set={BCH|LDPC|POLAR|RA|REP|RS|RSC|RSC_→DB|TPC|TURBO|TURBO_DB|UNCODED}>
  Select the channel code family to simulate.
--sim-prec, -p <integer:including set={8|16|32|64}>
  Specify the representation of the real numbers in the receiver part of the chain.
--sim-type <text:including set={BFER|BFERI|EXIT}>
  Select the type of simulation (or communication chain skeleton).

Other parameter(s):
--Help, -H
  Print the help like with the '--help, -h' parameter plus advanced arguments (denoted as '{A}').
(A) --except-a2l
  Enhance the backtrace when displaying exception. This change the program addresses into filenames and lines. It may take some seconds to do this work.
(A) --except-no-bt
  Disable the backtrace display when running an exception.
(A) --full-legend
  Display the legend with all modules details when launching the simulation.
  --help, -h
  Print the help with all the required (denoted as '{R}') and optional arguments. The latter change depending on the selected simulation type and code.
(A) --keys, -k
  Display the parameter keys in the help.
  --no-colors
  Disable the colors in the shell.
(A) --no-legend
  Disable the legend display (remove all the lines beginning by the '#' character).
  --version, -v
  Print informations about the version of the source code and compilation options.
--version, -v

Print informations about the version of the source code and compilation options.

aff3ct -v

aff3ct (Linux 64-bit, g++-5.4, AVX2) v2.1.1-48-g1c72c3d
Compilation options:
  * Precision: 8/16/32/64-bit
  * Polar bit packing: on
  * Terminal colors: on
  * Backtrace: on
  * External strings: on
  * MPI: off
  * GSL: off
  * MKL: off
  * SystemC: off
Copyright (c) 2016-2018 - MIT license.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

--keys, -k

Display the parameter keys in the help.

aff3ct -h -k

Usage: ./bin/aff3ct -C <text> [optional args...]

Simulation parameter(s):
{R} --sim-cde-type, -C <text:including set={BCH|LDPC|POLAR|RA|REP|RS|RSC|RSC_‐
  →DB|TPC|TURBO|TURBO_DB|UNCODED}>
  [factory::Launcher::parameters::p+cde-type,C]
  Select the channel code family to simulate.
  --sim-prec, -p <integer:including set={8|16|32|64}>
  [factory::Launcher::parameters::p+prec,p]
  Specify the representation of the real numbers in the receiver part of the
  chain.
  --sim-type <text:including set={BFER|BFERI|EXIT}>
  [factory::Launcher::parameters::p+type]
  Select the type of simulation (or communication chain skeleton).

Other parameter(s):
  --Help, -H
  [factory::Launcher::parameters::Help,H]
  Print the help like with the '--help, -h' parameter plus advanced
  arguments (denoted as '{A}').
  --help, -h
  [factory::Launcher::parameters::help,h]
  Print the help with all the required (denoted as '{R}') and optional
  arguments. The latter change depending on the selected simulation type and
  code.
  --no-colors
  [factory::Launcher::parameters::no-colors]
  Disable the colors in the shell.

(continues on next page)
--version, -v

[factory::Launcher::parameters::version,v]
Print informations about the version of the source code and compilation options.

--except-a2l  ADVANCED

Enhance the backtrace when displaying exception. This change the program addresses into filenames and lines. It may take some seconds to do this work.

Note: This option works only on Unix based OS (Operating System) and if AFF3CT has been compiled with debug symbols (-g compile flag) and without NDEBUG macro (-DNDEBUG flag).

--except-no-bt  ADVANCED

Disable the backtrace display when running an exception.

--no-legend  ADVANCED

Disable the legend display (remove all the lines beginning by the # character).

Tip: Use this option when you want to complete an already existing simulation result file with new noise points. Pay attention to use >> instead of > to redirect the standard output in order to add results at the end of the file and not overwriting it.

--full-legend  ADVANCED

Display the legend with all modules details when launching the simulation.
This additional information can help to understand a problem in the simulation. Data can of course be redundant from one module to another.

--no-colors

Disable the colors in the shell.

3.3 PyBER

PyBER is our Python GUI to display the AFF3CT outputs.
It can be downloaded here: https://github.com/aff3ct/PyBER.
3.3.1 Install Python3

Download and install Anaconda3: https://www.anaconda.com/download/ (tested on Anaconda3-4.2.0-Windows-x86_64).

Next next next... Install! Pay attention to add Python to the PATH when installing it.

3.3.2 Run PyBER

From your terminal, go to the folder where PyBER is located and run it:

```
python PyBER.py &
```
All the following examples simulate a basic communication chain with a BPSK modem and a repetition code over an AWGN channel. The BER/FER results are calculated from 0.0 dB to 10.0 dB with a 1.0 dB step.

Note: All the following examples of code are available in a dedicated GitHub repository: https://github.com/aff3ct/my_project_with_aff3ct. Sometime the full source codes in the repository may slightly differ from the ones on this page, but the philosophy remains the same.
4.1 Bootstrap

The bootstrap example is the easiest way to start using the AFF3CT library. It is based on C++ classes and methods that operate on buffers. Keep in mind that this is the simplest way to use AFF3CT, but not the most powerful way. More advanced features such as benchmarking, debugging, command line interfacing, and more are illustrated in the Tasks and Factory examples.

Fig. 4.2: Simulated communication chain: classes and methods.

Fig. 4.2 illustrates the source code: green boxes correspond to classes, blue boxes correspond to methods, and arrows represent buffers. Some of the AFF3CT classes inherit from the Module abstract class. Generally speaking, any class defining methods for a communication chain is a module (green boxes in Fig. 4.2).

Listing 4.1: Bootstrap: main function

```cpp
#include <aff3ct.hpp>
using namespace aff3ct;

int main(int argc, char** argv)
{
    params1 p; init_params1(p); // create and initialize the parameters defined by the user
    modules1 m; init_modules1(p, m); // create and initialize modules
    buffers1 b; init_buffers1(p, b); // create and initialize the buffers required by the modules
    util1 u; init_util1(m, u); // create and initialize utils

    // display the legend in the terminal
    u.terminal->legend();

    // loop over SNRs range
    for (auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
    {
        // compute the current sigma for the channel noise
        const auto esn0 = tools::ebn0_to_esn0(ebn0, p.R);
        const auto sigma = tools::esn0_to_sigma(esn0);
        u.noise->set_noise(sigma, ebn0, esn0);

        // update the sigma of the modem and the channel
        m.modem->set_noise(*u.noise);
        m.channel->set_noise(*u.noise);
    }

    (continues on next page)
```
Listing 4.1 gives an overview of what can be achieved with the AFF3CT library. The first lines 6–9 are dedicated to the objects instantiations and buffers allocation through dedicated structures. \( p \) contains the simulation parameters, \( b \) contains the buffers required by the modules, \( m \) contains the modules of the communication chain and \( u \) is a set of convenient helper objects.

Line 15 loops over the desired SNRs (Signal Noise Ratios) range. Lines 31–40, the while loop iterates until 100 frame errors have been detected by the monitor. The AFF3CT communication chain methods are called inside this loop. Each AFF3CT method works on input(s) and/or output(s) buffer(s) that have been declared at line 8. Those buffers can be \texttt{std::vector}, or pointers to user-allocated memory areas. The sizes and the types of those buffers have to be set in accordance with the corresponding sizes and types of the AFF3CT modules declared at line 7. If there is a size and/or type mismatch, the AFF3CT library throws an exception. The AFF3CT modules are classes that use the C++ meta-programming technique (e.g. C++ templates). By default those templates are instantiated to \texttt{int32_t} or \texttt{float}.

Listing 4.2: Bootstrap: parameters

```c
struct params1 {
  int K = 32; // number of information bits
  int N = 128; // codeword size
  int fe = 100; // number of frame errors
  int seed = 0; // PRNG seed for the AWGN channel
  float ebn0_min = 0.00f; // minimum SNR value
  float ebn0_max = 10.01f; // maximum SNR value
  float ebn0_step = 1.00f; // SNR step
  float R; // code rate (R=K/N)
};

void init_params1(params1 &p)
```

4.1. Bootstrap
{  
}

Listing 4.2 describes the params1 simulation structure and the init_params1 function used at line 6 in Listing 4.1.

Listing 4.3: Bootstrap: modules

```cpp
struct modules1
{
    std::unique_ptr<module::Source_random<> > source;
    std::unique_ptr<module::Encoder_repetition_sys<> > encoder;
    std::unique_ptr<module::Modem_BPSK<> > modem;
    std::unique_ptr<module::Channel_AWGN LLR<> > channel;
    std::unique_ptr<module::Decoder_repetition_std<> > decoder;
    std::unique_ptr<module::Monitor_BFER<> > monitor;
};

void init_modules1(const params1 &p, modules1 &m)
{
    m.source = std::unique_ptr<module::Source_random <> >(new module::Source_random <> (p.K));
    m.encoder = std::unique_ptr<module::Encoder_repetition_sys<> >(new module::Encoder_repetition_sys<>
        (p.K, p.N));
    m.modem = std::unique_ptr<module::Modem_BPSK <>(new module::Modem_BPSK <>(p.N));
    m.channel = std::unique_ptr<module::Channel_AWGN LLR<> >(new module::Channel_AWGN LLR<>
        (p.N, p.seed));
    m.decoder = std::unique_ptr<module::Decoder_repetition_std<> >(new module::Decoder_repetition_std<>
        (p.K, p.N));
    m.monitor = std::unique_ptr<module::Monitor_BFER<> >(new module::Monitor_BFER<>
        (p.K, p.fe));
}
```

Listing 4.1 describes the modules1 structure and the init_modules1 function used at line 7 in Listing 4.1. The init_modules1 function allocates the modules of the communication chain. Those modules are allocated on the heap and managed by smart pointers (std::unique_ptr). Note that the init_modules1 function takes a params1 structure from Listing 4.2 in parameter. These parameters are used to build the modules.

Listing 4.4: Bootstrap: buffers

```cpp
struct buffers1
{
    std::vector<int > ref_bits;
    std::vector<int > enc_bits;
    std::vector<float > symbols;
    std::vector<float > noisy_symbols;
    std::vector<float > LLRs;
    std::vector<int > dec_bits;
};

void init_buffers1(const params1 &p, buffers1 &b)
{
    b.ref_bits = std::vector<int >(p.K);
    b.enc_bits = std::vector<int >(p.N);
}
```

(continues on next page)
Listing 4.4 describes the buffers1 structure and the init_buffers1 function used at line 8 in Listing 4.1. The init_buffers1 function allocates the buffers of the communication chain. Here, we chose to allocate buffers as instances of the std::vector C++ standard class. As for the modules in Listing 4.3, the size of the buffers is obtained from the params1 input structure (cf. Listing 4.2).

Listing 4.5: Bootstrap: utils

```cpp
struct utils1 {
    std::unique_ptr<tools::Sigma<> > noise;     // a sigma noise type
    std::vector<std::unique_ptr<tools::Reporter>> reporters; // list of reporters
    // displayed in the terminal
    std::unique_ptr<tools::Terminal_std> terminal;    // manage the output
    // text in the terminal
};

void init_utils1(const modules1 &m, utils1 &u) {
    // create a sigma noise type
    u.noise = std::unique_ptr<tools::Sigma<> >(new tools::Sigma<>());
    // report the noise values (E_s/N_0 and E_b/N_0)
    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_ ˓→noise<>(*u.noise)));
    // report the bit/frame error rates
    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_ ˓→BFER<>(*m.monitor)));
    // report the simulation throughputs
    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_ ˓→throughput<>(*m.monitor)));
    // create a terminal that will display the collected data from the reporters
    u.terminal = std::unique_ptr<tools::Terminal_std>(new tools::Terminal_std(u. ˓→reporters));
}
```

Listing 4.5 describes the utils1 structure and the init_utils1 function used at line 9 in Listing 4.1. The init_utils1 function allocates 1) the noise object that contains the type of noise we want to simulate (e.g. sigma), 2) a terminal object that takes care of printing the BER/FER to the console. Three reporters are created, one to print SNR, second one to print BER/FER, and the last one to report the simulation throughput in the terminal.

If you run the bootstrap example, the expected output is shown in Listing 4.6.

Listing 4.6: Bootstrap: output

```bash
# ---------------------|------------------------------------------------------|------
# Signal Noise Ratio | Bit Error Rate (BER) and Frame Error Rate (FER) | ----- ˓→Global throughput | SNR |and elapsed time
# (SNR) | | |
```

(continues on next page)
# Tasks

Inside a Module class, there can be many public methods; however, only some of them are directly used in the communication chain. A method usable in a chain is named a Task. A Task is characterized by its behavior and its data: the input and output data are declared via a collection of Socket objects.

### Listing 4.7: Tasks: main function

```cpp
#include <aff3ct.hpp>
using namespace aff3ct;

int main(int argc, char** argv)
{
    params1 p; init_params1(p); // create and initialize the parameters defined by the user
    modules1 m; init_modules2(p, m); // create and initialize modules
```

---

Note: The full source code is available here: [https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/bootstrap/src/main.cpp](https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/bootstrap/src/main.cpp)
// the 'init_buffers1' function is not required anymore
utils1 u; init_utils1 (m, u); // create and initialize the utils

// display the legend in the terminal
u.terminal->legend();

// sockets binding (connect sockets of successive tasks in the chain: the output socket of a task fills the input socket of the next task in the chain)
using namespace module;
(*m.encoder)[enc::sck::encode ::U_K ].bind((*m.source
→[src::sck::generate ::U_K ]); (*m.modem )[mdm::sck::modulate ::X_N1].bind((*m.encoder)[enc::sck::encode
→::X_N ]); (*m.channel)[chn::sck::add_noise ::X_N ].bind((*m.modem 
→::X_N2)); (*m.modem )[mdm::sck::demodulate ::Y_N1].bind((*m.channel)[chn::sck::add_
noise ::Y_N ]); (*m.decoder)[dec::sck::decode_siho ::Y_N ].bind((*m.modem 
→::Y_N2));

// loop over the range of SNRs
for (auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
{
    // compute the current sigma for the channel noise
    const auto esn0 = tools::ebn0_to_esn0 (ebn0, p.R);
    const auto sigma = tools::esn0_to_sigma(esn0);
    u.noise->set_noise(sigma, ebn0, esn0);

    // update the sigma of the modem and the channel
    m.modem ->set_noise(*u.noise);
    m.channel->set_noise(*u.noise);

    // display the performance (BER and FER) in real time (in a separate thread)
    u.terminal->start_temp_report();

    // run the simulation chain
    while (!m.monitor->fe_limit_achieved())
    {
        (*m.source )[src::tsk::generate ].exec();
        (*m.encoder)[enc::tsk::encode ].exec();
        (*m.modem )[mdm::tsk::modulate ].exec();
        (*m.channel)[chn::tsk::add_noise ].exec();
        (*m.modem )[mdm::tsk::demodulate ].exec();
        (*m.decoder)[dec::tsk::decode_siho ].exec();
        (*m.monitor)[mnt::tsk::check_errors].exec();
    }

    // display the performance (BER and FER) in the terminal
    u.terminal->final_report();

    // reset the monitor and the terminal for the next SNR
Listing 4.7 shows how the Module, Task and Socket objects work together. Line 7, init_modules2 differs slightly from the previous init_modules1 function, Listing 4.8 details the changes.

Thanks to the use of Task and Socket objects, it is now possible to skip the buffer allocation part (see line 8), which is handled transparently by these objects. For that, the connections between the sockets of successive tasks in the chain have to be established explicitly: this is the binding process shown at lines 14–22, using the bind method. In return, to execute the tasks (lines 43–49), we now only need to call the exec method, without any parameters.

Using the bind and exec methods bring new useful features for debugging and benchmarking. In Listing 4.7, some statistics about tasks are collected and reported at lines 60–61 (see the –sim-stats section for more informations about the statistics output).

```
Listing 4.7: Tasks: module initialization

Listing 4.8: Tasks: modules
```
if (!tsk->is_debug() && !tsk->is_stats())
    tsk->set_fast(true);
}

The beginning of the init_modules2 function (Listing 4.8) is the same as the init_module1 function (Listing 4.3). At lines 10-24, each Module is parsed to get its tasks, each Task is configured to automatically allocate its outputs Socket memory (line 15) and collect statistics on the Task execution (line 19). It is also possible to print debug information by toggling boolean to true at line 17.

Note: The full source code is available here: https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/tasks/src/main.cpp.

4.3 SystemC/TLM

Alternatively, the AFF3CT modules support SystemC/TLM interfaces, Listing 4.9 highlights the modifications in the main function to use standard TLM interfaces.

Listing 4.9: SystemC/TLM: main function

```c++
#include <aff3ct.hpp>
using namespace aff3ct;

int sc_main(int argc, char** argv)
{
    // create and initialize the parameters defined by the user
    params1 p; init_params1 (p);
    modules1 m; init_modules2(p, m); // create and initialize modules
    utils1 u; init_utils1 (m, u); // create and initialize utils

    // display the legend in the terminal
    u.terminal->legend();

    // add a callback to the monitor to call the "sc_core::sc_stop()" function
    m.monitor->add_handler_check([&m, &u]() {
        if (m.monitor->fe_limit_achieved())
            sc_core::sc_stop();
    });

    // loop over the SNRs range
    for (auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
    {
        // compute the current sigma for the channel noise
        const auto esn0 = tools::ebn0_to_esn0 (ebn0, p.R);
        const auto sigma = tools::esn0_to_sigma(esn0);
        u.noise->set_noise(sigma, ebn0, esn0);

        // update the sigma of the modem and the channel
        m.modem->set_noise(*u.noise);
        m.channel->set_noise(*u.noise);
    }
```
// create "sc_core::sc_module" instances for each task
using namespace module;

m.source -> sc.create_module(+src::tsk::generate);
m.encoder -> sc.create_module(+enc::tsk::encode);
m.modem  -> sc.create_module(+mdm::tsk::modulate);
m.modem  -> sc.create_module(+mdm::tsk::demodulate);
m.channel-> sc.create_module(+chn::tsk::add_noise);
m.decoder-> sc.create_module(+dec::tsk::decode_siho);
m.monitor-> sc.create_module(+mnt::tsk::check_errors);

// declare a SystemC duplicator to duplicate the source 'generate'
tools::SC_Duplicator duplicator;

// bind the sockets between the modules
m.source -> sc[+src::tsk::generate].s_out[+src::sck::generate::U_K](duplicator.s_in);
duplicator.s_out1 -> (m.monitor-> sc[+mnt::tsk::check_errors].s_in[+mnt::sck::check_errors::U]);
duplicator.s_out2 -> (m.encoder-> sc[+enc::tsk::encode].s_in[+enc::sck::encode::U_K]);

m.encoder-> sc[+enc::tsk::encode].s_out[+enc::sck::encode::X_N](m.modem -> sc[+mdm::tsk::modulate].s_in[+mdm::sck::modulate::X_N1]);
m.modem  -> sc[+mdm::tsk::modulate].s_out[+mdm::sck::modulate::X_N2](m.channel-> sc[+chn::tsk::add_noise].s_in[+chn::sck::add_noise::X_N]);
m.channel-> sc[+chn::tsk::add_noise].s_out[+chn::sck::add_noise::Y_N](m.modem  -> sc[+mdm::tsk::modulate].s_in[+mdm::sck::modulate::Y_N1]);
m.modem  -> sc[+mdm::tsk::modulate].s_out[+mdm::sck::modulate::Y_N2](m.decoder-> sc[+dec::tsk::decode_siho].s_in[+dec::sck::decode_siho::Y_N]);
m.decoder-> sc[+dec::tsk::decode_siho].s_out[+dec::sck::decode_siho::Y_K](m.monitor-> sc[+mnt::tsk::check_errors].s_in[+mnt::sck::check_errors::Y]);

// display the performance (BER and FER) in real time (in a separate thread)
u.terminal-> start_temp_report();

// start the SystemC simulation
sc_core::sc_report_handler::set_actions(sc_core::SC_INFO, sc_core::SC_DONT_PRINT);
sccore::sc_start();

// display the performance (BER and FER) in the terminal
u.terminal-> final_report();

// reset the monitor and the terminal for the next SNR
m.monitor-> reset();
u.terminal-> reset();

// dirty way to create a new SystemC simulation context
sc_core::sc_curr_simcontext = new sc_core::sc_simcontext();
sc_core::sc_default_global_context = sc_core::sc_curr_simcontext;

// display the statistics of the tasks (if enabled)
tools::Stats::show({m.source.get(), m.encoder.get(), m.modem.get(), m.channel.get(), m.decoder.get(), m.monitor.get() }, true);
Note: The full source code is available here: https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/systemc/src/main.cpp.

4.4 Factory

In the previous Bootstrap, Tasks and SystemC/TLM examples, the AFF3CT Module classes were built statically in the source code. In the Factory example, factory classes are used instead, to build modules dynamically from command line arguments.

Listing 4.10: Factory: main function

```cpp
#include <aff3ct.hpp>
using namespace aff3ct;

int main(int argc, char** argv) {
  params3 p; init_params3 (argc, argv, p); // create and initialize the parameters from the command line with factories
  modules3 m; init_modules3 (p, m); // create and initialize modules
  utils1 u; init_utils3 (p, m, u); // create and initialize utils
  // [...]
  // display the statistics of the tasks (if enabled)
  tools::Stats::show({ m.source.get(), m.modem.get(), m.channel.get(), m.monitor.get(), m.encoder, m.decoder }, true);
  return 0;
}
```

The main function in Listing 4.10 is almost unchanged from the main function in Listing 4.7.

Listing 4.11: Factory: parameters

```cpp
struct params3 {
  float ebn0_min = 0.00f; // minimum SNR value
  float ebn0_max = 10.01f; // maximum SNR value
  float ebn0_step = 1.00f; // SNR step
  float R; // code rate (R=K/N)
  std::unique_ptr<factory::Source> source;
  std::unique_ptr<factory::Codec_repetition> codec;
  std::unique_ptr<factory::Modem> modem;
  std::unique_ptr<factory::Channel> channel;
  std::unique_ptr<factory::Monitor_BFER> monitor;
  std::unique_ptr<factory::Terminal> terminal;
};
```

(continues on next page)
The `init_params3` function takes two new input arguments from the command line: `argc` and `argv`. The function first allocates the factories (lines 18–23) and then those factories are supplied with parameters from the command line (line 29) thanks to the `factory::Command_parser` class. Lines 38–41, the parameters from the factories are printed to the terminal.

Note that in this example a repetition code is used, however it is very easy to select another code type, for instance by replacing repetition line 9 and line 19 by `polar` to work with polar code.

Listing 4.12: Factory: modules

```cpp
struct modules3
{
    std::unique_ptr<module::Source<> > source;
};
```
std::unique_ptr<module::Codec_SIHO<>> codec;
std::unique_ptr<module::Modem<>> modem;
std::unique_ptr<module::Channel<>> channel;
std::unique_ptr<module::Monitor_BFER<>> monitor;
module::Encoder<>* encoder;
module::Decoder_SIHO<>* decoder;

void init_modules3(const params3 &p, modules3 &m)
{
    m.source = std::unique_ptr<module::Source <>>(p.source ->build());
    m.codec = std::unique_ptr<module::Codec_SIHO <>>(p.codec ->build());
    m.modem = std::unique_ptr<module::Modem <>>(p.modem ->build());
    m.channel = std::unique_ptr<module::Channel <>>(p.channel->build());
    m.monitor = std::unique_ptr<module::Monitor_BFER<>>(p.monitor->build());
    m.encoder = m.codec->get_encoder().get();
    m.decoder = m.codec->get_decoder_siho().get();

    // configuration of the module tasks
    std::vector<\const module::Module*> modules_list = { m.source.get(), m.modem.
                                                      \get(), m.channel.get(), m.monitor.get(), m.encoder, m.decoder };  
    for (auto& mod : modules_list)  
        for (auto& tsk : mod->tasks)  
            tsk->set_autoalloc (true ); // enable the automatic
                                           // allocation of the data in the
                                           // tasks
            tsk->set_autoexec (false); // disable the auto execution
                                           // mode of the tasks
            tsk->set_debug (false); // disable the debug mode
            tsk->set_debug_limit(16 ); // display only the 16 first
                                           // bits if the debug mode is enabled
            tsk->set_stats (true ); // enable the statistics

    // enable the fast mode (= disable the useless verifs in the
    // tasks) if there is no debug and stats modes
    if (!tsk->is_debug() && !tsk->is_stats())
        tsk->set_fast(true);
}

In Listing 4.12 the modules3 structure changes a little bit because a Codec class is used to aggregate the Encoder
and the Decoder together. In the init_modules3 the factories allocated in Listing 4.11 are used to build the
modules (lines 14–18).

Listing 4.13: Factory: utils

void init_utils3(const params3 &p, const modules3 &m, utils1 &u)
{
    // create a sigma noise type
    u.noise = std::unique_ptr<tools::Sigma<>>(new tools::Sigma<>());
    // report noise values (Es/N0 and Eb/N0)
    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_
                                                             \noise<>(*u.noise)));  
    // report bit/frame error rates
    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_
                                                             \BFER<>(*m.monitor)));
9 // report simulation throughputs
10 u.reporters.push_back(std::unique_ptr<tools::Reporter>({new tools::Reporter_
11    ->throughput<(*m.monitor)}));
12 // create a terminal object that will display the collected data from the_
13    ->reporters
14 u.terminal = std::unique_ptr<tools::Terminal>(p.terminal->build(u.reporters));

In the Listing 4.13, the init_utils3 changes a little bit from the init_utils1 function (Listing 4.5) because at line 12 a factory is used to build the terminal.

To execute the binary it is now required to specify the number of information bits \(K\) and the frame size \(N\) as shown in Listing 4.14.

Listing 4.14: Factory: execute the binary

```
./bin/my_project -K 32 -N 128
```

Be aware that many other parameters can be set from the command line. The parameters list can be seen using \(-h\) as shown in Listing 4.15.

Listing 4.15: Factory: execute the binary

```
./bin/my_project -h
```

Those parameters are documented in the Parameters section.

**Note:** The full source code is available here: https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/factory/src/main.cpp.

### 4.5 OpenMP

In the previous examples the code is mono-threaded. To take advantage of the today multi-core CPUs some modifications have to be made. This example starts from the previous Factory example and adapts it to work on multi-threaded architectures using pragma directives of the well-known OpenMP language.

Listing 4.16: OpenMP: main function

```
int main(int argc, char** argv)
{
    params3 p; init_params3(argc, argv, p); // create and initialize the_
    -->parameters from the command line with factories
    utils4 u; // create an 'utils4' structure

    #pragma omp parallel
    {
        #pragma omp single
        {
            const size_t n_threads = (size_t)omp_get_num_threads();
            u.monitors.resize(n_threads);
            u.modules .resize(n_threads);
        }
    }
```
modules4 m; init_modules_and_utils4(p, m, u); // create and initialize the
modules and initialize a part of the utils

#pragma omp barrier
#pragma omp single
{
    init_utils4(p, u); // finalize the utils initialization
    u.terminal->legend();
}

// sockets binding (connect the sockets of the tasks = fill the input sockets
with the output sockets)

using namespace module;

(*m.encoder)[enc::sck::encode ::U_K ].bind((*m.source
->)[src::sck::generate ::U_K ]);  
(*m.modem )[mdm::sck::modulate ::X_N1].bind((*m.encoder)[enc::sck::encode
->::X_N ]);  
(*m.channel)[chn::sck::add_noise ::X_N ].bind((*m.modem
->)[mdm::sck::modulate ::X_N2]);

(*m.modem )[mdm::sck::demodulate ::Y_N1].bind((*m.channel)[chn::sck::add_
noise ::Y_N ]);  
(*m.decoder)[dec::sck::decode_siho ::Y_N ].bind((*m.modem
->)[mdm::sck::demodulate ::Y_N2]);

(*m.monitor)[mnt::sck::check_errors::U ].bind((*m.encoder)[enc::sck::encode
->::U_K ]);  
(*m.monitor)[mnt::sck::check_errors::V ].bind((*m.decoder)[dec::sck::decode_
siho::V_K ]);  

// loop over the SNRs range
for ( auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
{
    // compute the current sigma for the channel noise
    const auto esn0 = tools::ebn0_to_esn0(ebn0, p.R);
    const auto sigma = tools::esn0_to_sigma(esn0);

    #pragma omp single
    u.noise->set_noise(sigma, ebn0, esn0);

    // update the sigma of the modem and the channel
    m.modem ->set_noise(*u.noise);
    m.channel->set_noise(*u.noise);

    #pragma omp single
    u.terminal->start_temp_report();

    // run the simulation chain
    while (!u.monitor_red->is_done_all())
    {
        (*m.source )[src::tsk::generate ].exec();
        (*m.encoder)[enc::tsk::encode ].exec();
        (*m.modem )[mdm::tsk::modulate ].exec();
        (*m.channel)[chn::tsk::add_noise ].exec();
        (*m.modem )[mdm::tsk::demodulate ].exec();
    }
}
Listing 4.16 depicts how to use OpenMP pragmas to parallelize the whole communication chain. As a remainder:

- `#pragma omp parallel`: all the code after in the braces is executed by all the threads,
- `#pragma omp barrier`: all the threads wait all the others at this point,
- `#pragma omp single`: only one thread executes the code below (there is an implicit barrier at the end of the single zone).

In this example, a `params3` and an `utils4` structure are allocated in `p` and `u` respectively, before the parallel region (lines 3-4). As a consequence, `p` and `u` are shared among all the threads. On the contrary, a `modules4` structure is allocated in `m` inside the parallel region, thus each thread gets its own local `m`.

Listing 4.17: OpenMP: modules and utils

```cpp
struct modules4
{
    std::unique_ptr<module::Source<>*> source;
    std::unique_ptr<module::Codec_SIHO<>*> codec;
    std::unique_ptr<module::Modem<>*> modem;
    std::unique_ptr<module::Channel<>*> channel;
    module::Monitor_BFER<>* monitor;
    module::Encoder<>* encoder;
    module::Decoder_SIHO<>* decoder;
};

struct utils4
{
    
};
```
std::unique_ptr<tools::Sigma<>> noise; // a
→ sigma noise type
std::vector<std::unique_ptr<tools::Reporter>> reporters; // list
→ of reporters displayed in the terminal
std::unique_ptr<tools::Terminal> terminal; // manage
→ the output text in the terminal
std::vector<std::unique_ptr<module::Monitor_BFER<>>> monitors; // list
→ of the monitors from all the threads
std::unique_ptr<module::Monitor_BFER_reduction> monitor_red; // main
→ monitor object that reduce all the thread monitors
std::vector<std::vector<
const module::Module*>> modules; // list
→ of the allocated modules
std::vector<std::vector<
const module::Module*>> modules_stats; // list
→ of the allocated modules reorganized for the statistics

void init_modules_and_utils4(const params3 &p, modules4 &m, utils4 &u)
{
    // get the thread id from OpenMP
    const int tid = omp_get_thread_num();

    // set different seeds for different threads when the module use a PRNG
    p.source->seed += tid;
    p.channel->seed += tid;

    m.source = std::unique_ptr<module::Source <>>(p.source ->build());
    m.codec = std::unique_ptr<module::Codec_SIHO <>>(p.codec ->build());
    m.modem = std::unique_ptr<module::Modem <>>(p.modem ->build());
    m.channel = std::unique_ptr<module::Channel <>>(p.channel->build());
    u.monitors[tid] = std::unique_ptr<module::Monitor_BFER<>>(p.monitor->build());
    m.monitor = u.monitors[tid].get();
    m.encoder = m.codec->get_encoder().get();
    m.decoder = m.codec->get_decoder_siho().get();

    // configuration of the module tasks
    std::vector<
const module::Module*> modules_list = { m.source.get(), m.modem.
→get(), m.channel.get(), m.monitor, m.encoder, m.decoder };
    for (auto& mod : modules_list)
    {
        for (auto& tsk : mod->tasks)
        {
            tsk->set_autoalloc (true ); // enable the automatic
→allocation of the data in the tasks
            tsk->set_autoexec (false); // disable the auto execution
→mode of the tasks
            tsk->set_debug (false); // disable the debug mode
            tsk->set_debug_limit(16 ); // display only the 16 first
→bits if the debug mode is enabled
            tsk->set_stats (true ); // enable the statistics

            // enable the fast mode (= disable the useless verifs in the
→tasks) if there is no debug and stats modes
            if (!tsk->is_debug() && !tsk->is_stats())
                tsk->set_fast(true);
        }
    }
    u.modules[tid] = modules_list;
}

4.5. OpenMP
In Listing 4.17, there is a change in the modules4 structure compared to the modules3 structure (Listing 4.12): at line 7 the monitor is not allocated in this structure anymore, thus a standard pointer is used instead of a smart pointer. The monitor is now allocated in the utils4 structure at line 17, because all the monitors from all the threads have to be passed to build a common aggregated monitor for all of them: the monitor_red at line 18. monitor_red is able to perform the reduction of all the per-thread monitors. In the example, the monitor_red is the only member from u called by all the threads, to check whether the simulation has to continue or not (see line 54 in the main function, Listing 4.16).

In the init_modules_and_utils4 function, lines 25-30, a different seed is assigned to the modules using a PRNG. It is important to give a distinct seed to each thread. If the seed is the same for all threads, they all simulate the same frame contents and apply the same noise over it.

Lines 36-37, the monitors are allocated in u and the resulting pointer is assigned to m. At line 57 a list of the modules is stored in u.

In Listing 4.18, the init_utils4 function allocates and configure the monitor_red at lines 3-5. Note that the allocation of monitor_red is possible because the monitors have been allocated previously in the init_modules_and_utils4 function (Listing 4.17).

Lines 17-20, the u.modules list is reordered in the u.modules_stats to be used for the statistics of the tasks in the main function (Listing 4.16 line 84). In the u.modules list the first dimension is the number of threads and the second is the number of different modules while in u.modules_stats the two dimension are switched.

Note: The full source code is available here: https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/openmp/src/main.cpp.
CHAPTER 5

Classes

Work in progress...
Contributing Guidelines

We’re really glad you’re reading this, because we need volunteer developers to expand this project. Here are some important resources to communicate with us:

- The official website,

6.1 Submitting changes

Please send a GitHub Pull Request to AFF3CT with a clear list of what you’ve done (read more about pull requests). Please make your modifications on the development branch, any pull to the master branch will be refused (the master is dedicated to the releases).

Always write a clear log message for your commits. One-line messages are fine for small changes, but bigger changes should look like this:

```
git commit -m "A brief summary of the commit
> A paragraph describing what changed and its impact."
```

6.2 Regression Testing

We maintain a database of BER/FER reference simulations. Please give us some new references which solicit the code you added. We use those references in an automated regression test script. To propose new references please use our dedicated repository and send us a pull request on it.

6.3 Coding conventions

Start reading our code and you’ll get the hang of it. For the readability, we apply some coding conventions:
• we indent using tabulation (hard tabs),

• we ALWAYS put spaces after list items and method parameters ([1, 2, 3], not [1,2,3]), around operators (x += 1, not x+=1), and around hash arrows,

• we use the snake case (my_variable, not myVariable), classes start with an upper case (My_class, not my_class) and variables/methods/functions start with a lower case,

• the number of characters is limited to 120 per line of code.

This is open source software. Consider the people who will read your code, and make it look nice for them. It’s sort of like driving a car: Perhaps you love doing donuts when you’re alone, but with passengers the goal is to make the ride as smooth as possible.
Bibliography


[Pro] Altera University Program. The 1st 5g algorithm innovation competition-scma.


