
AFF3CT Documentation

Release v3.0.2-77-g8fc3f45d9

AFF3CT team

Mar 31, 2024

USER MANUAL

1	Introduction	1
2	Installation Guide	3
2.1	Get the Source Code	3
2.1.1	Git Installation	3
	Windows/macOS	3
	Linux	6
2.1.2	Clone AFF3CT from GitHub	6
2.2	Compilation	7
2.2.1	CMake Installation	7
	Windows/macOS	7
	Linux	8
2.2.2	C++ GNU Compiler Installation	9
	Windows	9
	macOS	9
	Linux	9
2.2.3	Compilation with a Makefile Project	9
	Windows	9
	macOS	10
	Linux	10
2.2.4	Compilation with a Visual Studio 2017 Solution	10
2.2.5	Compilation with a Visual Studio 2019 Solution	13
2.2.6	CMake Options	13
2.2.7	Compiler Options	14
	Build Type	14
	Specific Options	14
2.3	Installation	15
2.3.1	From Source	16
	Makefile Project	16
	Visual Studio Solution	16
2.3.2	Precompiled Versions	16
	From AFF3CT website	16
	On Debian / Ubuntu	16
2.3.3	Contents	17
3	Simulation	19
3.1	Overview	19
3.1.1	Basic Arguments	19
3.1.2	Output	20
3.1.3	Philosophy	21

3.2	Parameters	21
3.2.1	Simulation parameters	21
	--sim-type	22
	--sim-cde-type, -C	22
	--sim-prec, -p	23
	--sim-noise-type, -E	23
	--sim-noise-min, -m	24
	--sim-noise-max, -M	24
	--sim-noise-step, -s	24
	--sim-noise-range, -R	24
	--sim-pdf-path	25
	--sim-meta	25
	--sim-coded	25
	--sim-coset, -c	26
	--sim-dbg	26
	--sim-dbg-hex	27
	--sim-dbg-limit, -d	27
	--sim-dbg-fra	28
	--sim-dbg-prec	29
	--sim-seed, -S	29
	--sim-stats	29
	--sim-threads, -t	31
	--sim-inter-fra, -F	31
	--sim-crc-earlyt	32
	--sim-ite, -I	32
	--sim-sequence-path	32
	--sim-max-fra, -n	32
	--sim-stop-time	33
	--sim-crit-nostop	33
	--sim-err-trk	33
	--sim-err-trk-rev	33
	--sim-err-trk-path	34
	--sim-err-trk-thold	34
	References	34
3.2.2	Source parameters	34
	--src-info-bits, -K	34
	--src-type	35
	--src-implement	35
	--src-path	36
	--src-no-reset	36
	--src-fifo	36
	--src-start-idx	36
	References	36
3.2.3	CRC parameters	36
	--crc-type, --crc-poly	37
	--crc-size	38
	--crc-implement	38
	References	39
3.2.4	Codec parameters	39
	Codec Common	39
	Common Encoder parameters	39
	--enc-type	39
	--enc-path	40
	--enc-start-idx	40

Common Decoder parameters	40
--dec-type, -D	40
--dec-imlem	41
--dec-flips	41
--dec-hamming	41
--dec-seed	41
References	42
Codec BCH (Bose, Ray-Chaudhuri and Hocquenghem)	42
BCH Encoder parameters	42
--enc-cw-size, -N	42
--enc-info-bits, -K	42
--enc-type	42
--enc-simd	43
BCH Decoder parameters	43
--dec-type, -D	43
--dec-imlem	43
--dec-corr-pow, -T	44
References	44
Codec LDPC (Low-Density Parity-Check)	44
LDPC Encoder parameters	44
--enc-cw-size, -N	44
--enc-info-bits, -K	45
--enc-type	45
--enc-g-path	46
--enc-g-method	46
--enc-g-save-path	46
LDPC Decoder parameters	46
--dec-h-path	46
--dec-type, -D	48
--dec-imlem	48
--dec-simd	49
--dec-h-reorder	50
--dec-ite, -i	50
--dec-min	51
--dec-norm	51
--dec-off	52
--dec-mwbf-factor	52
--dec-synd-depth	52
--dec-ppbf-proba	52
--dec-no-synd	52
References	53
LDPC Puncturer parameters	53
--pct-fra-size, -N	53
--pct-type	53
--pct-pattern	53
Codec Polar	53
Polar Encoder parameters	53
--enc-type	53
--enc-no-sys	54
--enc-fb-gen-method	54
--enc-fb-awgn-path	55
--enc-fb-dump-path	55
--enc-fb-noise	55
References	56

Polar Decoder parameters	56
--dec-type, -D	56
--dec-implement	56
--dec-simd	57
--dec-ite, -i	58
--dec-flips	58
--dec-lists, -L	58
--dec-partial-adaptive	58
--dec-polar-nodes	58
References	59
Polar Puncturer parameters	59
--pct-fra-size, -N	59
--pct-info-bits, -K	59
--pct-type	59
References	60
Codec Polar MK (Multi-Kernel)	60
Polar MK Encoder parameters	60
--enc-type	60
--enc-sys	60
--enc-pc-kernel	60
--enc-pc-path	61
--enc-fb-gen-method	61
--enc-fb-awgn-path	62
--enc-fb-dump-path	62
--enc-fb-noise	63
References	63
Polar MK Decoder parameters	63
--dec-type, -D	63
--dec-implement	64
--dec-lists, -L	64
--dec-node-type	64
References	65
Codec RA (Repeat and Accumulate)	65
RA Encoder parameters	65
--enc-cw-size, -N	65
--enc-info-bits, -K	65
--enc-type	65
RA Decoder parameters	65
--dec-type, -D	65
--dec-implement	66
--dec-ite, -i	66
Codec Repetition	66
Repetition Encoder parameters	66
--enc-cw-size, -N	66
--enc-info-bits, -K	67
--enc-type	67
--enc-no-buff	67
Repetition Decoder parameters	67
--dec-type, -D	67
--dec-implement	68
Codec RS (Reed-Solomon)	68
RS Encoder parameters	68
--enc-cw-size, -N	68
--enc-info-bits, -K	68

--enc-type	69
RS Decoder parameters	69
--dec-type, -D	69
--dec-implement	69
--dec-corr-pow, -T	70
References	70
Codec RSC (Recursive Systematic Convolutional)	70
RSC Encoder parameters	70
--enc-info-bits, -K	70
--enc-type	71
--enc-no-buff	71
--enc-poly	71
--enc-std	72
RSC Decoder parameters	72
--dec-type, -D	72
--dec-implement	72
--dec-simd	73
--dec-max	73
References	74
Codec RSC DB (Double Binary)	74
RSC DB Encoder parameters	74
--enc-info-bits, -K	74
--enc-type	74
--enc-no-buff	75
--enc-std	75
RSC DB Decoder parameters	75
--dec-type, -D	75
--dec-implement	76
--dec-max	76
References	76
Codec Turbo	76
Turbo Encoder parameters	76
--enc-info-bits, -K	76
--enc-type	77
--enc-sub-type	77
--enc-json-path	77
--enc-sub-no-buff	78
--enc-sub-poly	78
--enc-sub-std	78
Turbo Decoder parameters	78
--dec-type, -D	78
--dec-implement	79
--dec-sub-type, -D	79
--dec-sub-implement	79
--dec-sub-simd	79
--dec-crc-start	79
--dec-fnc	80
--dec-fnc-ite-m	80
--dec-fnc-ite-M	80
--dec-fnc-ite-s	80
--dec-fnc-q	81
--dec-ite, -i	81
--dec-sc	81
--dec-sf-type	81

--dec-sub-max	82
References	82
Turbo Puncturer parameters	82
--pct-type	82
--pct-pattern	83
Codec Turbo DB	83
Turbo DB Encoder parameters	83
--enc-info-bits, -K	83
--enc-type	83
--enc-sub-type	84
--enc-sub-std	84
Turbo DB Decoder parameters	84
--dec-type, -D	84
--dec-implement	84
--dec-sub-type, -D	85
--dec-sub-implement	85
--dec-crc-start	85
--dec-fnc	85
--dec-fnc-ite-m	85
--dec-fnc-ite-M	86
--dec-fnc-ite-s	86
--dec-fnc-q	86
--dec-ite, -i	86
--dec-sf-type	87
--dec-sub-max	87
References	88
Turbo DB Puncturer parameters	88
--pct-type	88
--pct-fra-size, -N	88
Codec TPC (Turbo Product Code)	88
TPC Encoder parameters	88
--enc-sub-cw-size, -N	88
--enc-sub-info-bits, -K	89
--enc-type	89
--enc-sub-type	89
--enc-ext	89
TPC Decoder parameters	89
--dec-sub-cw-size, -N	90
--dec-sub-info-bits, -K	90
--dec-type, -D	90
--dec-implement	91
--dec-ite, -i	91
--dec-alpha	92
--dec-beta	92
--dec-c	92
--dec-p	92
--dec-t	93
--dec-cp-coef	93
--dec-sub-type, -D	93
--dec-sub-corr-pow, -T	93
--dec-sub-implement	93
References	93
Codec Uncoded	93
Uncoded Encoder parameters	93

	Uncoded Decoder parameters	94
	--dec-type, -D	94
	--dec-imlem	94
3.2.5	Interleaver parameters	94
	--itl-type	94
	--itl-cols	95
	--itl-path	95
	--itl-read-order	96
	--itl-seed	97
	--itl-uni	97
	References	97
3.2.6	Modem parameters	97
	--mdm-type	98
	--mdm-imlem	98
	--mdm-bps	98
	--mdm-const-path	99
	--mdm-max	100
	--mdm-no-sig2	100
	--mdm-cpm-k	100
	--mdm-cpm-p	100
	--mdm-cpm-L	101
	--mdm-cpm-upf	101
	--mdm-cpm-map	101
	--mdm-cpm-ws	101
	--mdm-cpm-std	102
	--mdm-ite	102
	--mdm-psi	102
	--mdm-cb-path	103
	--mdm-rop-est	104
	References	104
3.2.7	Channel parameters	104
	--chn-type	104
	--chn-imlem	106
	--chn-blk-fad	108
	--chn-gain-occur	108
	--chn-path	108
	References	109
3.2.8	Quantizer parameters	109
	--qnt-type	109
	--qnt-imlem	109
	--qnt-range	110
	--qnt-bits	110
	--qnt-dec	110
3.2.9	Monitor parameters	111
	--mnt-max-fe, -e	111
	--mnt-err-hist	111
	--mnt-err-hist-path	112
	--mnt-mutinfo	112
	--mnt-red-lazy	112
	--mnt-red-lazy-freq	112
	--mnt-mpi-comm-freq	113
3.2.10	Terminal parameters	113
	--ter-type	113
	--ter-freq	114

	--ter-no	114
	--ter-sigma	114
3.2.11	Other parameters	114
	--help, -h	114
	--Help, -H	115
	--version, -v	116
	--keys, -k	116
	--except-no-bt	117
	--no-legend	117
	--full-legend	117
	--no-colors	117
3.3	PyBER	117
3.3.1	Install Python3	118
3.3.2	Run PyBER	118
4	Library Examples	119
4.1	Bootstrap	119
4.2	Tasks	124
4.3	Factory	127
4.4	OpenMP	130
5	License	137
	Bibliography	139

INTRODUCTION

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).

Fig. 1.1: The communication chain.

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.

INSTALLATION GUIDE

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.

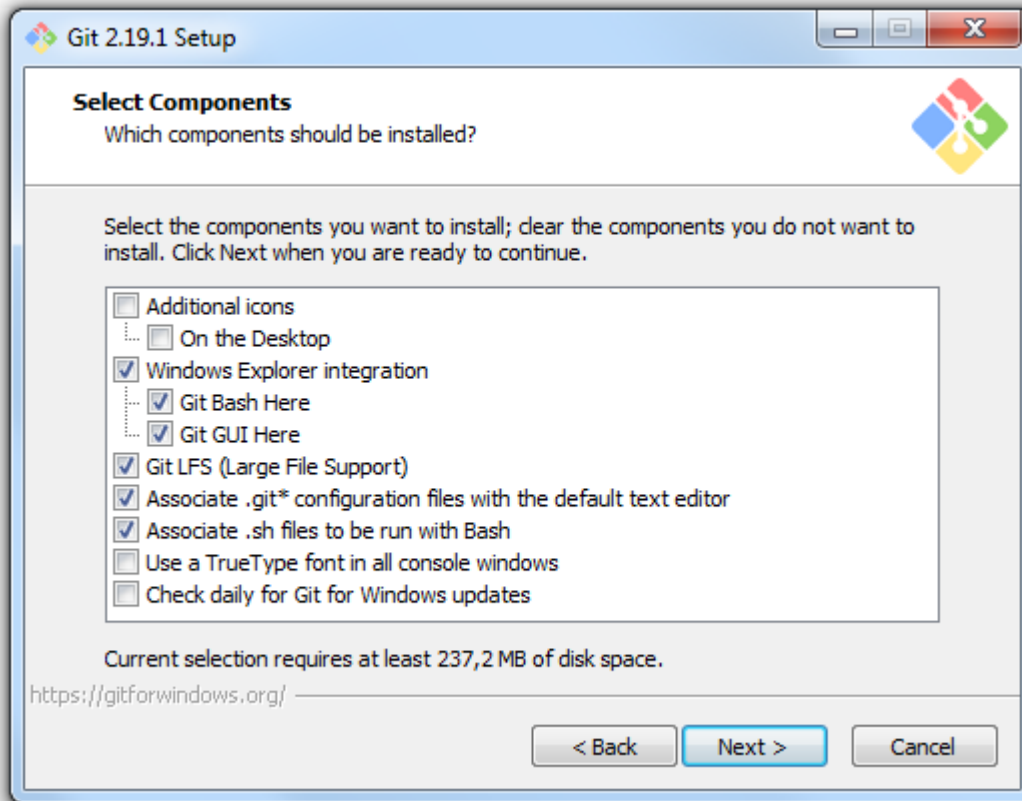
This project uses [Git](#) as the version-control system to manage the source code. The AFF3CT repository is hosted on [GitHub](#). To get the source code, first install the Git software and secondly *clone* the [AFF3CT repository](#) locally.

2.1.1 Git Installation

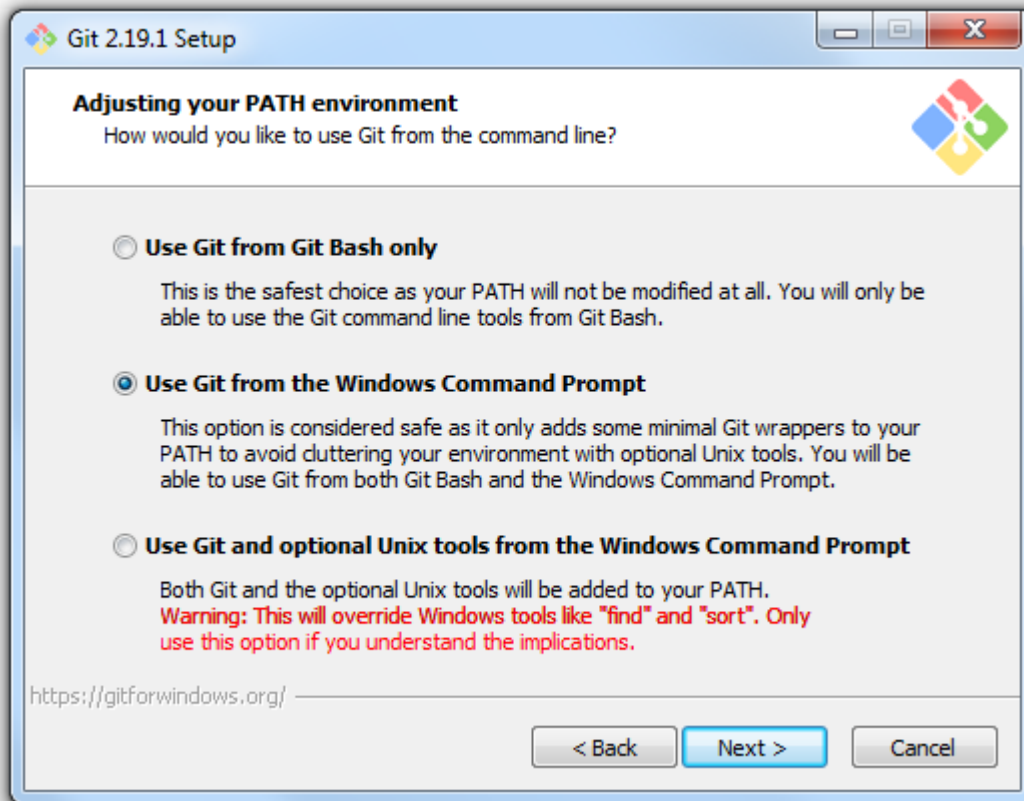
Windows/macOS

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

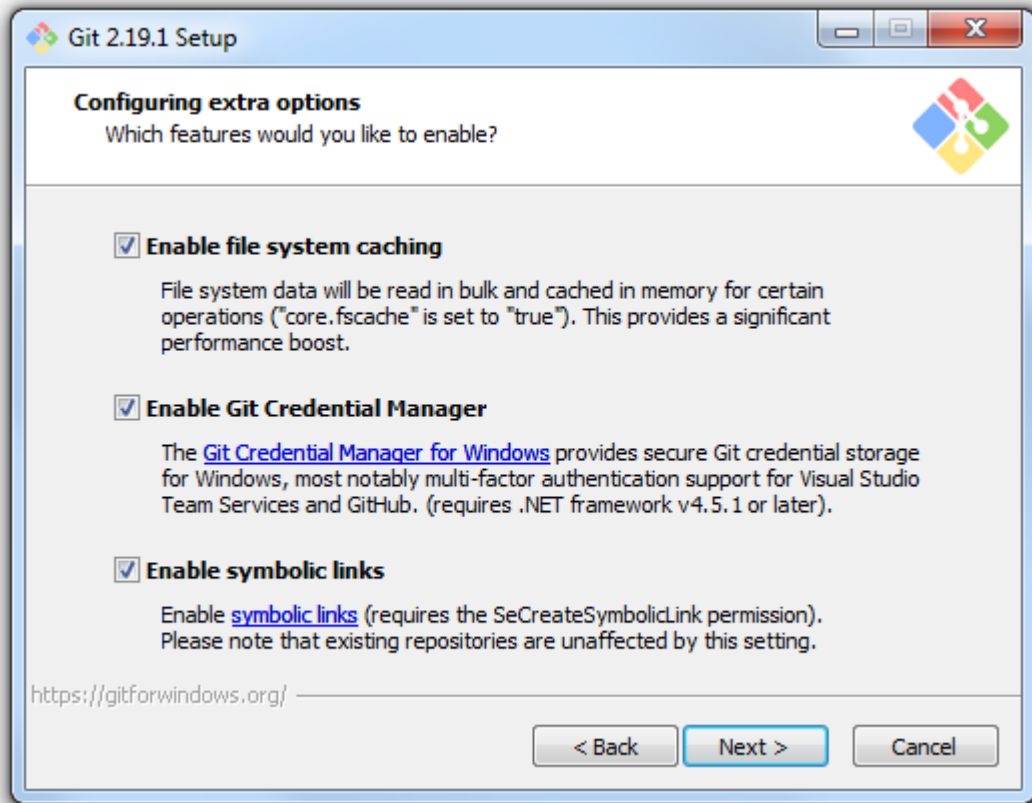
Warning: On Windows, Git comes with the **Git Bash** terminal which is, to our mind, better suitable than the traditional **Windows Console**. We encourage you to use **Git Bash** instead of the **Windows Command Prompt** for the following steps.



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:

```
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:

```
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](#). 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

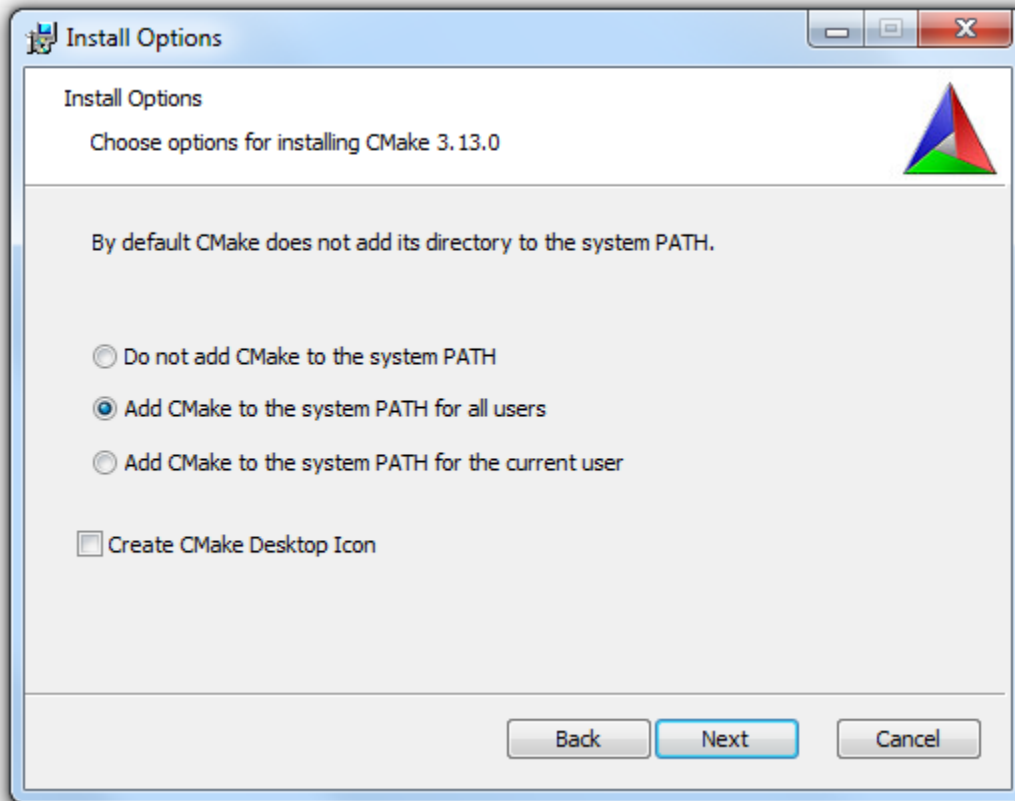
Download [CMake](#) from the official web page and launch the installer. Just press the *Next* button until the installation is over.

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**.



Linux

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"
```

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"
```

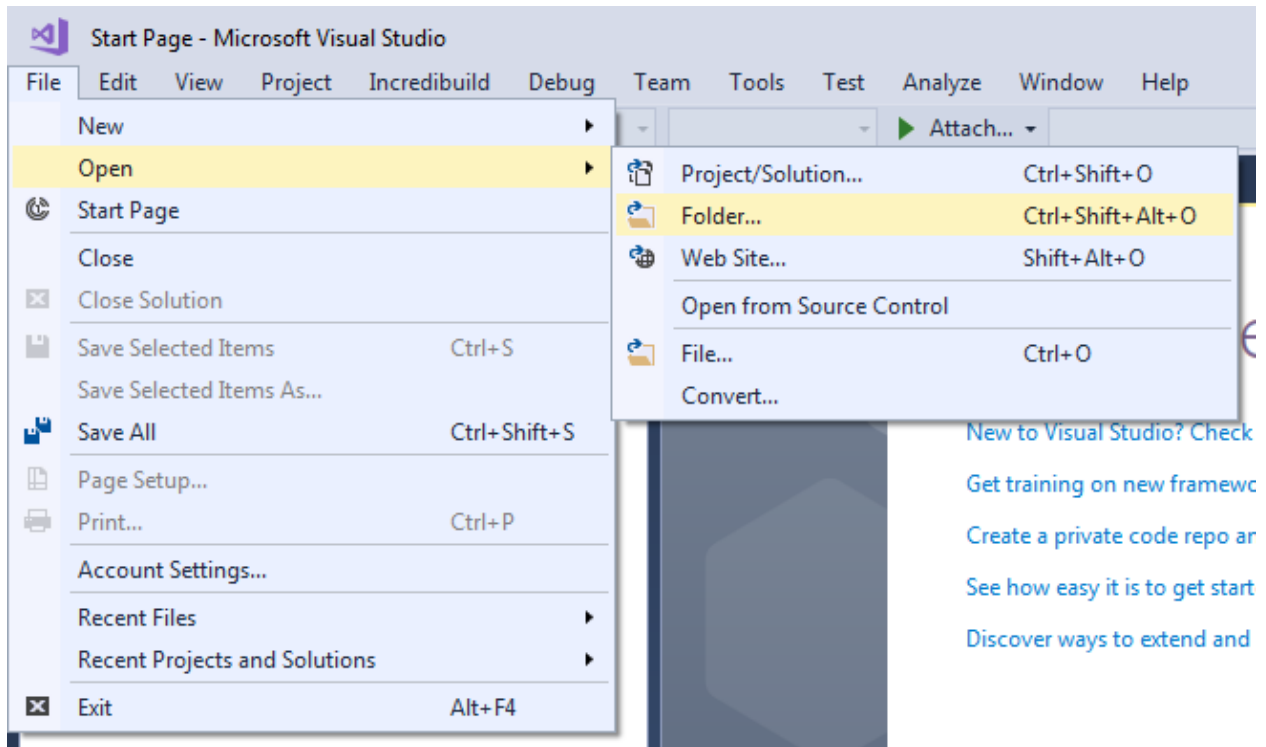
Build AFF3CT with the Makefile:

```
make -j4
```

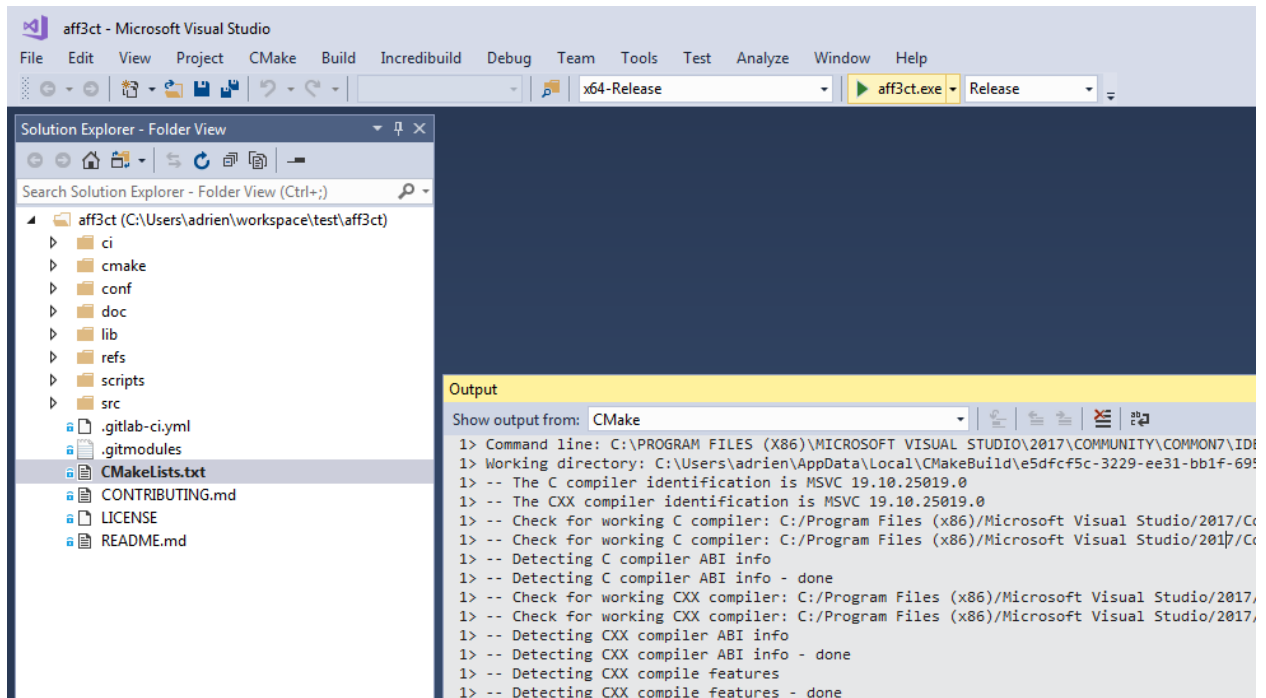
Once finished, the AFF3CT executable should be located in the \$AFF3CT_ROOT/build/bin folder.

2.2.4 Compilation with a Visual Studio 2017 Solution

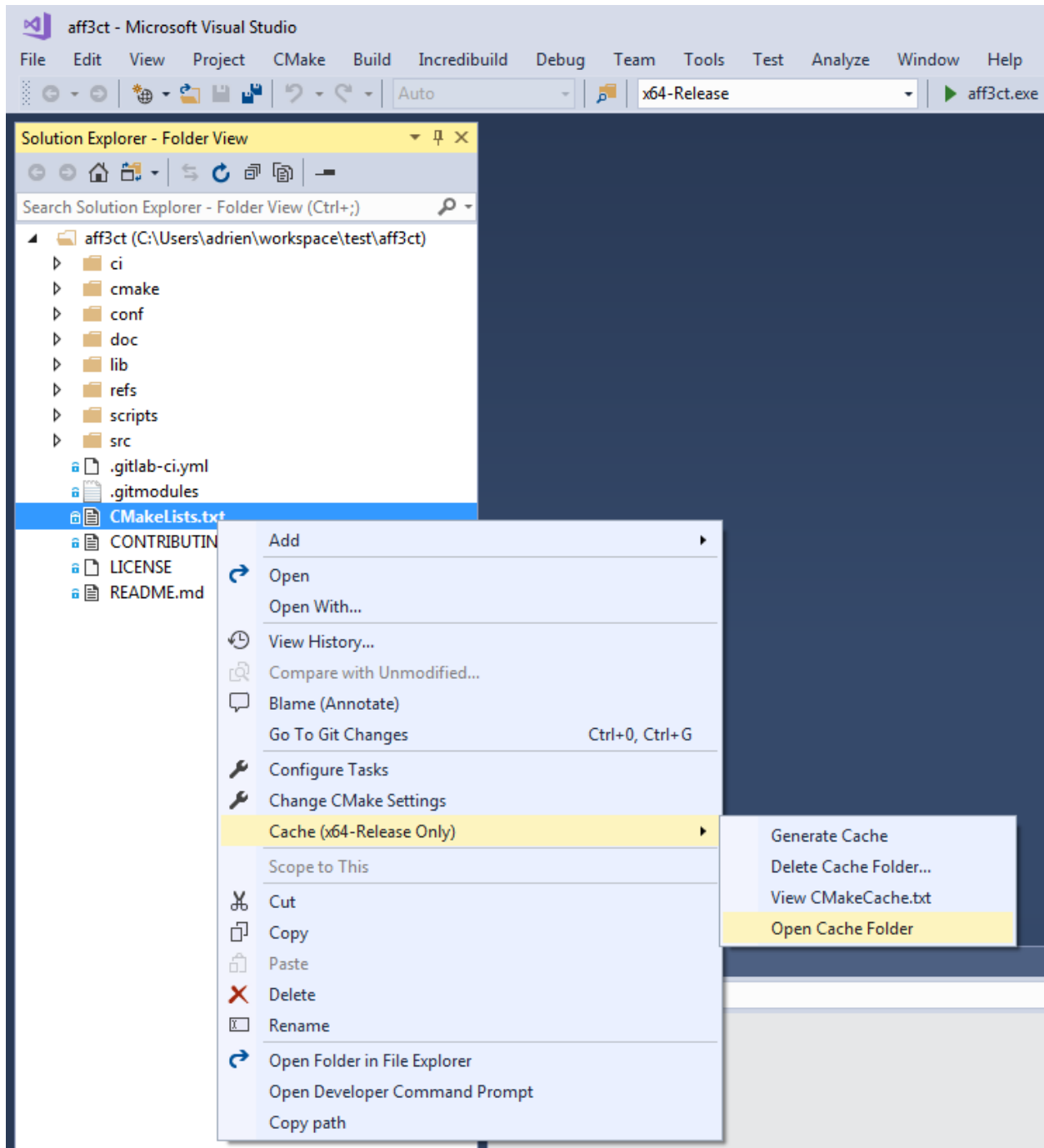
Since Microsoft Visual Studio 2017, Visual natively supports CMake. To generate the AFF3CT solution, open the \$AFF3CT_ROOT folder from the IDE.



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):

```
devenv /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:

Option	Type	Default	Description
AFF3CT_COMPILE_EXECUTABLE	BOOLEAN	ON	Compile the executable.
AFF3CT_COMPILE_STATIC_LIB	BOOLEAN	OFF	Compile the static library.
AFF3CT_COMPILE_SHARED_LIB	BOOLEAN	OFF	Compile the shared library.
AFF3CT_LINK_GSL	BOOLEAN	OFF	Link with the GSL library (used in the channels).
AFF3CT_LINK_MKL	BOOLEAN	OFF	Link with the MKL library (used in the channels).
AFF3CT_MPI	BOOLEAN	OFF	Enable the MPI support.
AFF3CT_POLAR_BIT_PACKING	BOOLEAN	ON	Enable the bit packing technique for Polar code SC decoding.
AFF3CT_COLORS	BOOLEAN	ON	Enable the colors in the terminal.
AFF3CT_BACKTRACE	BOOLEAN	ON	Enable the backtrace display when an exception is raised. On Windows and macOS this option is not available and automatically set to OFF.
AFF3CT_EXTERNAL_STRINGS	BOOLEAN	ON	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.
AFF3CT_PRECISION	STRING	MULTI	Select the precision in bits (can be '8', '16', '32', '64' or 'MULTI').

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
<code>-msse2</code>	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).
<code>-msse3</code>	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 <code>--dec-type</code> , <code>-D</code> and <code>--dec-implement</code> parameters).
<code>-msse4.1</code>	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).
<code>-mavx</code>	Enable the AVX (Advanced Vector Extensions) set of instructions on x86 CPUs (256-bit vector size, required for 32-bit and 64-bit data).
<code>-mavx2</code>	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).
<code>-mavx512f</code>	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).
<code>-mavx512bw</code>	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).
<code>-mfpu=neon</code>	Enable the NEON (ARM SIMD instructions) set of instructions on ARMv7 (Advanced RISC (Reduced Instruction Set Computer) Machine Version 7) and ARMv8 (Advanced RISC (Reduced Instruction Set Computer) Machine Version 8) CPUs (128-bit vector size, required for 8-bit, 16-bit data and 32-bit data).
<code>-march=native</code>	Let the compiler choose the best set of instructions available on the current architecture (it does not work for ARMv7 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-bin aff3ct-doc libaff3ct libaff3ct-dev
```

- The package `aff3ct-bin` contains the `bin/`, `conf/` and `refs/` folders.
- The package `aff3ct-doc` contains the `doc/` folder.
- The package `libaff3ct` contains the `lib/` folder.
- The package `libaff3ct-dev` contains the `include/` folder and depends on the `libaff3ct` package.

Note: Only LTS releases of Ubuntu are officially supported, starting from 16.04. For other releases, you may try to use [the pinning method](#).

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.

SIMULATION

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) increases, 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 *--sim-cde-type*, -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.

Fig. 3.1: Code-related parameters in the 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 *--sim-noise-min*, -m and *--sim-noise-max*, -M sections for more information). -s is the step between each SNR (c.f. section *--sim-noise-step*, -s). The Fig. 3.2 shows the output BER for each simulated SNR values.

Fig. 3.2: SNR-related parameters in the communication chain.

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
# -----
# ---- A FAST FORWARD ERROR CORRECTION TOOLBOX >> ----
# -----
# Parameters :
# [...]
#
# The simulation is running...
# -----||-----||-----
↪-----
# Signal Noise Ratio || Bit Error Rate (BER) and Frame Error Rate (FER) || ↪
↪Global throughput
# (SNR) || || and↪
↪elapsed time
# -----||-----||-----
↪-----
# -----||-----||-----||-----||-----||-----||-----
↪--|-----
# Es/N0 | Eb/N0 || FRA | BE | FE | BER | FER || SIM_
↪THR | ET/RT
# (dB) | (dB) || | | | | || (Mb/
↪s) | (hhmmss)
# -----||-----||-----||-----||-----||-----||-----
↪--|-----
# 0.25 | 1.00 || 104 | 16425 | 104 | 9.17e-02 | 1.00e+00 || 4.
↪995 | 00h00'00
# 1.25 | 2.00 || 104 | 12285 | 104 | 6.86e-02 | 1.00e+00 || 13.
↪678 | 00h00'00
# 2.25 | 3.00 || 147 | 5600 | 102 | 2.21e-02 | 6.94e-01 || 14.
↪301 | 00h00'00
# 3.25 | 4.00 || 5055 | 2769 | 100 | 3.18e-04 | 1.98e-02 || 30.
↪382 | 00h00'00
# End of the simulation.

```

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:

- E_s/N_0 : the SNR expressed as E_s/N_0 in dB (c.f. the *--sim-noise-type*, *-E* section),
- E_b/N_0 : 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.

Fig. 3.3: Modules and tasks in the simulation.

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-implement` 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** BFER BFERI**Default** BFER**Examples** --sim-type BFERI

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

Description of the allowed values:

Value	Description
BFER	The standard BER/FER chain (Fig. 3.4).
BFERI	The iterative BER/FER chain (Fig. 3.5).

Fig. 3.4: The standard BER/FER chain.

Fig. 3.5: The iterative BER/FER chain.

--sim-cde-type, -C**Type** text**Allowed values** BCH LDPC POLAR POLAR_MK 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:

Value	Description
BCH	The Bose–Chaudhuri–Hocquenghem codes [Sim-BRC60].
LDPC	The Low-Density Parity-Check codes [Sim-Gal63, Sim-MN95].
POLAR	The Polar codes [Sim-Ari09].
POLAR_MK	The Polar mono- or multi-kernel codes [Sim-BBGL17].
RA	The Repeat Accumulate codes [Sim-DHJM98].
REP	The Repetition codes [Sim-RL09].
RS	The Reed-Solomon codes [Sim-RS60].
RSC	The Recursive Systematic Convolutional codes [Sim-RL09].
RSC_DB	The Recursive Systematic Convolutional codes with double binary symbols [Sim-RL09].
TURBO	The Turbo codes [Sim-BGT93].
TURBO_DB	The Turbo codes with double binary symbols [Sim-BGT93].
TURBO_PROD	The Turbo Product codes [Sim-RL09].
UNCODED	An uncoded simulation.

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:

Value	Description
8	8-bit precision.
16	16-bit precision.
32	32-bit precision.
64	64-bit precision.

--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:

Value	Description
EBN0	SNR per information bit
ESN0	SNR per transmitted symbol
EP	Event Probability
ROP	Received Optical Power

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

$$\frac{E_S}{N_0} = \frac{E_B}{N_0} + 10 \cdot \log(R \cdot bps),$$

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

Note: When selecting EP the simulator runs in reverse order, ie. 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-dbg`

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

```
aff3ct -C "REP" -K 4 -N 8 -m 1.0 -M 1.0 --sim-dbg
# [...]
# Source_random::generate(int32 U_K[4])
# {OUT} U_K = [ 1, 1, 0, 1]
# Returned status: 0
#
# Encoder_repetition_sys::encode(const int32 U_K[4], int32 X_N[8])
# {IN} U_K = [ 1, 1, 0, 1]
# {OUT} X_N = [ 1, 1, 0, 1, 1, 1, 0, 1]
# Returned status: 0
#
# Modem_BPSK::modulate(const int32 X_N1[8], float32 X_N2[8])
# {IN} X_N1 = [ 1, 1, 0, 1, 1, 1, 0, 1]
# {OUT} X_N2 = [-1.00, -1.00, 1.00, -1.00, -1.00, -1.00, 1.00, -1.00]
# Returned status: 0
#
# Channel_AWGN_LLR::add_noise(const float32 X_N[8], float32 Y_N[8])
# {IN} X_N = [-1.00, -1.00, 1.00, -1.00, -1.00, -1.00, 1.00, -1.00]
# {OUT} Y_N = [-0.29, -0.24, 1.55, -0.58, -0.33, -0.51, 0.80, -2.88]
# Returned status: 0
#
# Modem_BPSK::demodulate(const float32 Y_N1[8], float32 Y_N2[8])
# {IN} Y_N1 = [-0.29, -0.24, 1.55, -0.58, -0.33, -0.51, 0.80, -2.88]
# {OUT} Y_N2 = [-0.73, -0.61, 3.91, -1.45, -0.84, -1.28, 2.01, -7.26]
# Returned status: 0
#
# Decoder_repetition_std::decode_siho(const float32 Y_N[8], int32 V_K[4])
# {IN} Y_N = [-0.73, -0.61, 3.91, -1.45, -0.84, -1.28, 2.01, -7.26]
# {OUT} V_K = [ 1, 1, 0, 1]
# Returned status: 0
#
# Monitor_BFER::check_errors(const int32 U[4], const int32 V[4])
# {IN} U = [ 1, 1, 0, 1]
# {IN} V = [ 1, 1, 0, 1]
```

(continues on next page)

(continued from previous page)

```
# Returned status: 0
# [...]
```

Note: By default, the debug mode runs the simulation on one thread. It is strongly advise to remove the `--sim-threads`, `-t` parameter from your command line if you use it.

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

`--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.

```
aff3ct -C "REP" -K 4 -N 8 -m 1.0 -M 1.0 --sim-dbg-hex
# [...]
# Modem_BPSK::modulate(const int32 X_N1[8], float32 X_N2[8])
# {IN} X_N1 = [0x1, 0x1, 0x0, 0x1, 0x1, 0x1, 0x0, 0x1]
# {OUT} X_N2 = [-0x1p+0, -0x1p+0, 0x1p+0, -0x1p+0, -0x1p+0, -0x1p+0, 0x1p+0, -0x1p+0]
# Returned status: 0
#
# Channel_AWGN_LL2R::add_noise(const float32 X_N[8], float32 Y_N[8])
# {IN} X_N = [-0x1p+0, -0x1p+0, 0x1p+0, -0x1p+0, -0x1p+0, -0x1p+0, 0x1p+0, -0x1p+0]
# {OUT} Y_N = [-0x1.28be1cp-2, -0x1.ec1ec8p-3, 0x1.8d242cp+0, -0x1.268a8p-1, -0x1.
↪ 54c3ccp-2, -0x1.04df9ap-1, 0x1.9905f8p-1, -0x1.71132cp+1]
# Returned status: 0
# [...]
```

`--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
# [...]
# Modem_BPSK::modulate(const int32 X_N1[8], float32 X_N2[8])
# {IN} X_N1 = [ 1, 1, 0, ...]
# {OUT} X_N2 = [-1.00, -1.00, 1.00, ...]
# Returned status: 0
#
# Channel_AWGN_LL2R::add_noise(const float32 X_N[8], float32 Y_N[8])
```

(continues on next page)

(continued from previous page)

```
# {IN} X_N = [-1.00, -1.00, 1.00, ...]
# {OUT} Y_N = [-0.29, -0.24, 1.55, ...]
# Returned status: 0
# [...]
```

--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 *--sim-inter-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
# [...]
# Modem_BPSK::modulate(const int32 X_N1[8x8], float32 X_N2[8x8])
# {IN} X_N1 = [f1( 1, 1, 0, 1, 1, 1, 0, 1),
#             f2( 0, 1, 1, 0, 0, 1, 1, 0),
#             f3( 1, 0, 1, 1, 1, 0, 1, 1),
#             f4( 1, 0, 0, 0, 1, 0, 0, 0),
#             f5->f8(...)]
# {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_LLNR::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.

```
aff3ct -C "REP" -K 4 -N 8 -m 1.0 -M 1.0 --sim-dbg-prec 4
# [...]
# Modem_BPSK::modulate(const int32 X_N1[8], float32 X_N2[8])
# {IN} X_N1 = [ 0, 0, 1, 1, 0, 0, 1, 1]
# {OUT} X_N2 = [ 1.0000, 1.0000, -1.0000, -1.0000, 1.0000, 1.0000, -1.0000, -1.0000]
# Returned status: 0
#
# Channel_AWGN_LLRL::add_noise(const float32 X_N[8], float32 Y_N[8])
# {IN} X_N = [ 1.0000, 1.0000, -1.0000, -1.0000, 1.0000, 1.0000, -1.0000, -1.0000]
# {OUT} Y_N = [ 1.4260, 0.4301, -1.5119, 0.1559, 0.0784, 1.6980, -1.6501, -0.0769]
# Returned status: 0
# [...]
```

--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.

--sim-stats

Display statistics for each task. Those statistics are shown after each simulated SNR point.

```
aff3ct -C "POLAR" -K 1723 -N 2048 -m 4.2 -M 4.2 -t 1 --sim-stats
# [...]
# -----||-----||-----
# Statistics for the given task || Basic statistics ||
# Measured throughput || Measured latency
# ('*' = any, '-' = same as previous) || on the task ||
# considering the last socket || considering the last socket
# -----||-----||-----
# -----||-----||-----||-----||-----||-----||-----
# |-----|-----|-----|-----|-----|-----|-----|
# |-----|-----|-----|-----|-----|-----|-----|
# |-----|-----|-----|-----|-----|-----|-----|
```

(continues on next page)

(continued from previous page)

#	MODULE	TASK	TIMER	CALLS	TIME	PERC	AVERAGE
→	MINIMUM	MAXIMUM	AVERAGE	MINIMUM	MAXIMUM		
#						(s)	(%)
→	(Mb/s)	(Mb/s)	(us)	(us)	(us)		(Mb/s)
#	-----	-----	-----	-----	-----	-----	-----
→	-----	-----	-----	-----	-----	-----	-----
#	Channel	add_noise	*	14909	0.72	37.59	42.17
→	20.75	45.52	48.56	44.99	98.69		
#	Source	generate	*	14909	0.60	31.13	42.84
→	8.72	44.34	40.22	38.86	197.63		
#	Encoder	encode	*	14909	0.37	19.06	83.17
→	16.00	86.10	24.62	23.79	127.97		
#	Decoder	decode_siho	*	14909	0.22	11.32	117.80
→	36.67	126.75	14.63	13.59	46.99		
#	Monitor	check_errors	*	14909	0.01	0.42	3186.81
→	120.63	3697.42	0.54	0.47	14.28		
#	Modem	demodulate	*	14909	0.00	0.25	6350.57
→	160.24	7876.92	0.32	0.26	12.78		
#	Modem	modulate	*	14909	0.00	0.23	6962.61
→	184.84	8291.50	0.29	0.25	11.08		
#	-----	-----	-----	-----	-----	-----	-----
→	-----	-----	-----	-----	-----	-----	-----
#	TOTAL	*	*	14909	1.93	100.00	13.34
→	3.38	14.10	129.19	122.21	509.42		
#	[...]						

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.

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 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 exceeds the size of the CPU caches and it becomes less interesting to use a large number of threads.

`--sim-inter-fra, -F`

Type integer

Default 1

Examples `--sim-inter-fra 3`

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 during the execution of a task.

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 will automatically affect the IFL level in all the simulation modules (c.f. Fig. 3.6).

Fig. 3.6: 3-way inter frame level in the communication chain.

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).

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

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.

`--sim-crc-earlyt`

Enable to use the CRC to early terminate (if possible) the turbo decoding process. It should reduce the simulation time in high SNR zones.

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-sequence-path`

Type file

Rights write

Default `sequence.dot`

Examples `--sim-sequence-path sequence.dot`

Export the simulated sequence in Graphviz format at the given path.

`--sim-max-fra, -n`

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

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 σ .

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 USER_BIN**Default** RAND**Examples** --src-type AZCW

Method used to generate the K information bits.

Description of the allowed values:

Value	Description
AZCW	Set all the information bits to 0.
RAND	Generate randomly the information bits based on the MT 19937 (Mersenne Twister 19937) PRNG [Src-MN98].
USER	Read the information bits from a given file, the path can be set with the <code>--src-path</code> parameter.
USER_BIN	Read the information bits from a given file in binary mode , the path can be set with the <code>--src-path</code> parameter.

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.

Note: For the USER_BIN type, raw data is read from the file that is open in binary mode. When the end of the file is reached, the source goes back to the beginning of the file and continues to generate bits from there.

--src-implement**Type** text**Allowed values** FAST STD**Default** STD**Examples** --src-implement FAST

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

Description of the allowed values:

Value	Description
STD	Standard implementation working for any source type.
FAST	Fast implementation, only available for the RAND source type.

--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 (informations bits), to use with the USER or USER_BIN source type.

With the USER_BIN source type, raw binary data is directly read from the file.

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-no-reset

Do not reset the source (start to the beginning) if the file reach EOF (End Of File) and pad with zeros after EOF (work only for USER_BIN source type).

--src-fifo

If the FIFO (or pipe) reach the end, then reset the EOF flag and retry to read in loop.

--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 (Cyclic Redundancy Check) 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 [Crc-LST12, Crc-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 `--crc-size` parameter. The type NO deactivates the CRC.

Table 3.1: List of the predefined CRC polynomials.

Type	Polynomial	Size
32-GZIP	0x04C11DB7	32
32-CASTAGNOLI	0x1EDC6F41	32
32-AIXM	0x814141AB	32
32-KOOPMAN	0x32583499	32
30-CDMA	0x2030B9C7	30
24-LTEA	0x864CFB	24
24-RADIX-64	0x864CFB	24
24-FLEXRAY	0x5D6DCB	24
24-5GA	0x864CFB	24
24-5GB	0x800063	24
24-5GC	0xB2B117	24
21-CAN	0x102899	21
17-CAN	0x1685B	17
16-IBM	0x8005	16
16-CCITT	0x1021	16
16-PROFIBUS	0x1DCF	16
16-OPENSAFETY-B	0x755B	16
16-OPENSAFETY-A	0x5935	16
16-DNP	0x3D65	16
16-T10-DIF	0x8BB7	16
16-DECT	0x0589	16
16-CDMA2000	0xC867	16
16-ARINC	0xA02B	16
16-CHAKRAVARTY	0x2F15	16
16-5G	0x1023	16
15-MPT1327	0x6815	15
15-CAN	0x4599	15
14-DARC	0x0805	14
13-BBC	0x1CF5	13
12-CDMA2000	0xF13	12

continues on next page

Table 3.1 – continued from previous page

12-TELECOM	0x80F	12
11-FLEXRAY	0x385	11
11-5G	0x621	11
10-CDMA2000	0x3D9	10
10-ATM	0x233	10
8-WCDMA	0x9B	8
8-SAE-J1850	0x1D	8
8-DARC	0x39	8
8-DALLAS	0x31	8
8-CCITT	0x07	8
8-AUTOSAR	0x2F	8
8-DVB-S2	0xD5	8
7-MVB	0x65	7
7-MMC	0x09	7
6-CDMA2000-A	0x27	6
6-CDMA2000-B	0x07	6
6-DARC	0x19	6
6-ITU	0x03	6
5-ITU	0x15	5
5-EPC	0x09	5
5-USB	0x05	5
4-ITU	0x3	4
1-PAR	0x1	1

--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:

Value	Description
STD	The standard implementation is generic and support any size of CRCs (Cyclic Redundancy Checks). On the other hand the throughput is limited.
FAST	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.
INTER	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.

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:

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

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:

```
# '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}
```

--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:

Value	Description
CHASE	Select the Chase decoder from [Cmn-Cha72].
ML	Select the perfect ML (Maximum Likelihood) decoder.

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

`--dec-implement`

Type text

Allowed values NAIVE STD

Examples `--dec-implement STD`

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
NAIVE	Select the naive implementation (very slow and only available for the ML decoder).
STD	Select the standard implementation.

`--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 .

$N = 2^{m \vee 1}$, 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:

Value	Description
BCH	Select the standard BCH encoder.
AZCW	See the common <i>--enc-type</i> parameter.
COSET	See the common <i>--enc-type</i> parameter.
USER	See the common <i>--enc-type</i> parameter.

--enc-simd**Type** text**Allowed values** INTER**Examples** --enc-simd INTER

Select the SIMD strategy.

Description of the allowed values:

Value	Description
INTER	Select the inter-frame strategy.

Note: Be aware that running the simulator with the *--sim-inter-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.

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:

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

--dec-implement**Type** text**Allowed values** FAST GENIUS STD**Default** STD**Examples** --dec-implement FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
STD	A standard implementation of the BCH.
FAST	Select the fast implementation optimized for SIMD architectures.
GENIUS	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.

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:

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

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:

Value	Description
IDENTITY	Generate an identity on H to get the parity part.
LU_DEC	Generate a hollow G thanks to the LU decomposition with a guarantee to have the systematic identity. Do not work with irregular matrices.

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.

LDPC Decoder parameters**--dec-h-path****Type** file**Rights** read only**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} [...] d_VN_{nVN}
# list of the degrees for each check nodes
d_CN_{1} d_CN_{2} [...] 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} [...] VN_{1}_CN_{idx_d_VN_{1}}
# variable node '2'
VN_{2}_CN_{idx_1} [...] VN_{2}_CN_{idx_d_VN_{2}}
[...]
# variable node 'nVN'
VN_{nVN}_CN_{idx_1} [...] 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} [...] CN_{1}_VN_{idx_d_CN_{1}}
# check node '2'
CN_{2}_VN_{idx_1} [...] CN_{2}_VN_{idx_d_CN_{2}}
[...]
# check node 'nCN'
CN_{nCN}_VN_{idx_1} [...] 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
# 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}
```

(continues on next page)

(continued from previous page)

```
[...]
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:

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

--dec-imlem**Type** text**Allowed values** STD GALA GALB GALE WBF MWBF PPBF SPA LSPA AMS MS NMS OMS**Default** SPA**Examples** --dec-imlem AMS

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
STD	Select the STD (Standard) implementation.
GALA	Select the GALA (Gallager A) algorithm [Ldpc-RL09].
GALB	Select the GALB (Gallager B) algorithm [Ldpc-DDB14] with majority vote.
GALE	Select the GALE (Gallager E) algorithm [Ldpc-DDB14] with extended alphabet.
PPBF	Select the PPBF (Probabilistic Parallel Bit-Flipping) algorithm [Ldpc-LGK+19].
WBF	Select the WBF (Weighted Bit Flipping) algorithm [Ldpc-WNY+10].
MWBF	Select the MWBF (Modified Weighted Bit Flipping) algorithm [Ldpc-WNY+10].
SPA	Select the SPA (Sum-Product Algorithm) update rules [Ldpc-MN95].
LSPA	Select the LSPA (Logarithmic Sum-Product Algorithm) update rules [Ldpc-MN95].
AMS	Select the AMS (Approximate Min-Star) update rule.
MS	Select the MS (Min-Sum) update rule [Ldpc-FMI99].
NMS	Select the NMS (Normalized Min-Sum) update rule [Ldpc-CF02].
OMS	Select the OMS (Offset Min-Sum) update rule [Ldpc-CF02].

Table 3.2 shows the different decoder types and their corresponding available implementations.

Table 3.2: LDPC decoder types and available implementations.

De-coder	STD	GALA	GALB	GALE	PPBF	WBF	MWBF	SPA	LSPA	AMS	MS	NMS	OMS
BF					✓	✓	✓						
BP-P	✓												
BP-F		✓	✓	✓				✓ ^{***+}	✓ ^{**}	✓ ^{**}	✓ ^{**}	✓ ^{**}	✓ ^{**}
BP-HL								✓ ^{**}	✓ ^{**}	✓ ^{**}	✓ [*]	✓ [*]	✓ [*]
BP-VL								✓ ^{**}	✓ ^{**}	✓ ^{**}	✓ ^{**}	✓ ^{**}	✓ ^{**}

* / **: 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, INTRA

Examples `--dec-simd INTER`

Select the SIMD strategy.

Table 3.2 shows the decoders and implementations that support SIMD.

Description of the allowed values:

Value	Description
INTRA	Select the intra-frame strategy.
INTER	Select the inter-frame strategy.

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 `--sim-inter-fra, -F` parameter. Be aware that running the simulator with the `--sim-inter-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:

Value	Description
ASC	Reorder from the smallest CNs degree to the biggest CNs degree.
DSC	Reorder from the biggest CNs degree to the smallest CNs degree.
NONE	Do not change the order.

`--dec-ite, -i`

Type integer

Default 10

Examples `--dec-ite 30`

Set the maximal number of iterations in the LDPC decoder.

Note: 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:

Value	Description
MINS	$\min^*(a, b) = \min(a, b) + \log(1 + \exp(-(a+b))) - \log(1 + \exp(- a-b)).$
MINL	$\min^*(a, b) \approx \min(a, b) + \text{corr}(a+b) - \text{corr}(a-b)$ with $\text{corr}(x) = \begin{cases} 0 & \text{if } x \geq 2.625 \\ -0.375x + 0.6825 & \text{if } x < 1.0 \\ -0.1875x + 0.5 & \text{else} \end{cases}.$
MIN	$\min^*(a, b) \approx \min(a, b).$

MINS 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 Bernoulli 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 [Ldpc-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:

Value	Description
NO	Disable the puncturer.
LDPC	Puncture the LDPC codeword.

--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:

Value	Description
POLAR	Select the standard Polar encoder.
AZCW	See the common <code>--enc-type</code> parameter.
COSET	See the common <code>--enc-type</code> parameter.
USER	See the common <code>--enc-type</code> parameter.

`--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:

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

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 $\sigma = 0.435999$:

```
8
awgn
0.435999
7 6 5 3 4 2 1 0
```

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 σ 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:

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

--dec-implement

Type text

Allowed values NAIVE FAST

Default FAST

Examples --dec-implement FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
NAIVE	Select the naive implementation which is typically slow (not supported by the A-SCL decoders).
FAST	Select the fast implementation, available only for the SC, SCL, SCL-MEM, A-SCL and A-SCL-MEM decoders.

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

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

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

--dec-simd

Type text

Allowed values INTER INTRA

Examples --dec-simd INTER

Select the SIMD strategy.

Description of the allowed values:

Value	Description
INTER	Select the inter-frame strategy, only available for the SC FAST decoder (see [Plrd-CAL+16, Plrd-CLeGalL+15, Plrd-LeGalLJego15]).
INTRA	Select the intra-frame strategy, only available for the SC (see [Plrd-CAL+16, Plrd-CLeGalL+15]), SCL and A-SCL decoders (see in [Plrd-LeonardonCL+17]).

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 `--sim-inter-fra, -F` parameter. Be aware that running the simulator with the `--sim-inter-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 [Plrd-ABSB14].

--dec-lists, -L

Type integer

Default 8

Examples --dec-lists 1

Set the number of lists to maintain in the SCL and A-SCL decoders.

--dec-partial-adaptive

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 [Plrd-CLeGalL+15, Plrd-SGV+14]. 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`

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:

Value	Description
NO	Disable the puncturer.
SHORTLAST	Select the short last puncturing strategy from [Plrp-Mil15, Plrp-NCL13, Plrp-WL14].

References

Codec Polar MK (Multi-Kernel)

Polar MK Encoder parameters

`--enc-type`

Type text

Allowed values POLAR_MK AZCW COSET USER

Default POLAR_MK

Examples `--enc-type AZCW`

Select the encoder type.

Description of the allowed values:

Value	Description
POLAR_MK	Select the standard Polar MK encoder.
AZCW	See the common <code>--enc-type</code> parameter.
COSET	See the common <code>--enc-type</code> parameter.
USER	See the common <code>--enc-type</code> parameter.

`--enc-sys`

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

`--enc-pc-kernel`

Type list of (list of (boolean:including set={0|1}):limited length [1;inf]), elements of same length

Default "10,11"

Examples `--enc-pc-kernel "111,101,011"`

Set the polar code kernel (squared matrix only).

The Kronecker powers of the kernel matrix T_s defines the G generator matrix. The frame size N can only be a power of the matrix size s . For traditional Polar codes $s = 2$.

Note: This parameter can only be used to define mono-kernel polar codes.

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

Set the path to a file containing the polar code description (kernels definition and stages).

An ASCII file is expected:

```
# number of different kernels
2

# dimension of kernel 0
2
# kernel 0
1 0
1 1

# dimension of kernel 1
3
# kernel 1
1 1 1
1 0 1
0 1 1

# number of stages
5
# type of kernel per stage
0 0 0 0 1
```

The previous file describes a Polar MK code which is build from the following Kronecker product: $G_{48} = T_2 \otimes T_2 \otimes T_2 \otimes T_2 \otimes T_3$ with $N = 2 \times 2 \times 2 \times 2 \times 3 = 48$.

Note: When this parameter is used it overrides the `--enc-pc-kernel` parameter.

Note: This parameter is the only way to define multi-kernel polar codes.

--enc-fb-gen-method**Type** text**Default** GA**Allowed values** GA GAA FILE**Examples** --enc-fb-gen-method FILE

Select the frozen bits generation method.

Description of the allowed values:

Value	Description
GA	Select the GA method from [Pmke-Tri12].
GAA	Select the GA method from [Pmke-Tri12] (works only for Arikan mono-kernel polar codes).
FILE	Read the best channels from an external file, to use with the <code>--enc-fb-awgn-path</code> parameter.

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

`--enc-fb-awgn-path`

Type file

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 channel where the noise variance is $\sigma = 0.435999$:

```
8
awgn
0.435999
7 6 5 3 4 2 1 0
```

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.

`--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 GAA 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.

References**Polar MK Decoder parameters****--dec-type, -D****Type** text**Allowed values** SC SCL ASCL CHASE ML**Default** SC**Examples** --dec-type SCL

Select the decoder algorithm.

Description of the allowed values:

Value	Description
SC	Select the original SC algorithm from [Plrd-Ari09].
SCL	Select the SCL algorithm from [Pmkd-TV11], also support the improved CA-SCL algorithm.
ASCL	Select the A-SCL algorithm from [Pmkd-LST12], only the PA-SCL variant is available.
CHASE	See the common <i>--dec-type, -D</i> parameter.
ML	See the common <i>--dec-type, -D</i> parameter.

At this time, the SC, SCL and ASCL decoders support only a subset of polar kernels listed below.

$$T2_{Arikan} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

$T2_{Arikan}$ is the original 2×2 kernel proposed by Arikan [Pmkd-Ari09]. This matrix is invertible and can be used for systematic encoding/decoding schemes.

$$T3_{Huawei1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

$T3_{Huawei1}$ is a 3×3 kernel proposed in [Pmkd-GBLB17]. This matrix is **not** invertible and **cannot** be used for systematic encoding/decoding schemes.

$$T3_{Huawei2} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

$T3_{Huawei2}$ is a 3×3 kernel proposed in [Pmkd-BL18]. This matrix is invertible and can be used for systematic encoding/decoding schemes.

$$T4_{Huawei} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

$T4_{Huawei}$ is a 4×4 kernel proposed in [Pmkd-BL18]. This matrix is invertible and can be used for systematic encoding/decoding schemes.

$$T4_{Huawei} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

$T5_{Huawei}$ is a 5×5 kernel proposed in [Pmkd-BL18]. This matrix is invertible and can be used for systematic encoding/decoding schemes.

`--dec-implement`

Type text

Allowed values NAIVE

Default NAIVE

Examples `--dec-implement NAIVE`

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
NAIVE	Select the naive implementation which is typically slow.

`--dec-lists, -L`

Type integer

Default 8

Examples `--dec-lists 1`

Set the number of lists to maintain in the SCL decoder.

`--dec-node-type`

Type text

Allowed values MS SPA

Default MS

Examples `--dec-node-type SPA`

Select the type of computations to make in the decoding functions.

Description of the allowed values:

Value	Description
MS	$L_a \boxplus L_b \simeq \text{sign}(L_a) \cdot \text{sign}(L_b) \cdot \min(L_a , L_b)$.
SPA	$L_a \boxplus L_b = 2 \tanh^{-1}(\tanh(\frac{L_a}{2}) \cdot \tanh(\frac{L_b}{2}))$.

References

Codec RA (Repeat and Accumulate)

RA Encoder parameters

--enc-cw-size, -N

Type integer

Examples --enc-cw-size 1

Set the codeword size N .

--enc-info-bits, -K

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:

Value	Description
RA	Select the standard RA encoder.
AZCW	See the common --enc-type parameter.
COSET	See the common --enc-type parameter.
USER	See the common --enc-type parameter.

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:

Value	Description
RA	Select the RA decoder based on the MS update rule in the CNs.
CHASE	See the common <i>--dec-type</i> , <i>-D</i> parameter.
ML	See the common <i>--dec-type</i> , <i>-D</i> parameter.

--dec-implement

Type text

Allowed values STD

Default STD

Examples --dec-implement STD

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
STD	Select the STD implementation.

--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 1Set 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:

Value	Description
REP	Select the standart repetition decoder.
AZCW	See the common <i>--enc-type</i> parameter.
COSET	See the common <i>--enc-type</i> parameter.
USER	See the common <i>--enc-type</i> parameter.

--enc-no-buff

Disable the buffered encoding.

Without the buffered encoding, considering K information bits $U_0, U_1, \dots, U_{K-1}$, the corresponding sequence of bits in the codeword is organized as follow: $X_0^0, X_0^1, \dots, X_0^{rep-1}, X_1^0, X_1^1, \dots, X_1^{rep-1}, \dots, X_{K-1}^0, X_{K-1}^1, \dots, X_{K-1}^{rep-1}$, with $rep = N/K$.

With the buffered encoding, considering K information bits $U_0, U_1, \dots, U_{K-1}$, the corresponding sequence of bits in the codeword is organized as follow: $X_0^0, X_1^0, \dots, X_{K-1}^0, X_0^1, X_1^1, \dots, X_{K-1}^1, \dots, X_0^{rep-1}, X_1^{rep-1}, \dots, X_{K-1}^{rep-1}$, 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:

Value	Description
REPETITION	Select the repetition decoder.
CHASE	See the common <i>--dec-type, -D</i> parameter.
ML	See the common <i>--dec-type, -D</i> parameter.

--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:

Value	Description
STD	Select the STD implementation.
FAST	Select the fast implementation, much more faster without the <i>--enc-no-buff</i> parameter.

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:

Value	Description
RS	Select the standard RS encoder.
AZCW	See the common <i>--enc-type</i> parameter.
COSET	See the common <i>--enc-type</i> parameter.
USER	See the common <i>--enc-type</i> parameter.

RS Decoder parametersThe RS decoder was described by Reed and Solomon in 1960 [[Rs-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:

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

--dec-implement**Type** text**Allowed values** STD GENIUS**Default** STD**Examples** --dec-implement GENIUS

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
STD	A standard implementation of the RS.
GENIUS	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.

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

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:

Value	Description
RSC	Select the standard RSC encoder.
AZCW	See the common <i>--enc-type</i> parameter.
COSET	See the common <i>--enc-type</i> parameter.
USER	See the common <i>--enc-type</i> parameter.

--enc-no-buff

Disable the buffered encoding.

Without the buffered encoding, considering the following sequence of K information bits: $U_0, U_1, [\dots], U_{K-1}$, the encoded bits will be organized as follow: $X_0^s, X_0^p, X_1^s, X_1^p, [\dots], X_{K-1}^s, X_{K-1}^p, X_0^t, X_0^p, X_1^s, X_1^p, [\dots], X_{\log_2(ts)-1}^s, X_{\log_2(ts)-1}^p$, 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: $U_0, U_1, [\dots], U_{K-1}$, the encoded bits will be organized as follow: $X_0^s, X_1^s, [\dots], X_{K-1}^s, X_0^t, X_1^t, [\dots], X_{\log_2(ts)-1}^t, X_0^p, X_1^p, [\dots], X_{K-1}^p, X_0^t, X_1^t, [\dots], X_{\log_2(ts)-1}^t$, 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:

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

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:

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

--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:

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

--dec-simd**Type** text**Allowed values** INTER INTRA**Examples** --dec-simd INTER

Select the SIMD strategy.

Description of the allowed values:

Value	Description
INTER	Select the inter-frame strategy, only available for the BCJR STD, FAST and VERY_FAST implementation (see [Rsc-CTL+16]).
INTRA	Select the intra-frame strategy, only available for the BCJR STD and FAST implementations (see [Rsc-WWY+13]).

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 *--sim-inter-fra, -F* parameter. Be aware that running the simulator with the *--sim-inter-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 $\max^*$ operator used in the trellis decoding.

Description of the allowed values:

Value	Description
MAXS	$\max^*(a, b) = \max(a, b) + \log(1 + \exp(- a - b))$.
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. 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

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:

Value	Description
RSC_DB	Select the standard RSC DB encoder.
AZCW	See the common <i>--enc-type</i> parameter.
COSET	See the common <i>--enc-type</i> parameter.
USER	See the common <i>--enc-type</i> parameter.

--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:

Value	Description
DVB-RCS1	Select the configuration of the DVB-RCS ₁ (Digital Video Broadcasting - Return Channel via Satellite 1) standard.
DVB-RCS2	Select the configuration of the DVB-RCS ₂ (Digital Video Broadcasting - Return Channel via Satellite 2) standard.

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:

Value	Description
BCJR	Select the BCJR DB decoder [Rsdb-BCJR74].
CHASE	See the common <i>--dec-type, -D</i> parameter.
ML	See the common <i>--dec-type, -D</i> parameter.

--dec-implement**Type** text**Allowed values** GENERIC DVB-RCS1 DVB-RCS2**Default** DVB-RCS1**Examples** --dec-implement 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:

Value	Description
MAXS	$\max^*(a, b) = \max(a, b) + \log(1 + \exp(- a - b))$.
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. 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**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 AZCW COSET USER

Default TURBO

Examples `--enc-type AZCW`

Select the encoder type.

Description of the allowed values:

Value	Description
TURBO	Select the standard Turbo encoder.
AZCW	See the common <code>--enc-type</code> parameter.
COSET	See the common <code>--enc-type</code> parameter.
USER	See the common <code>--enc-type</code> parameter.

`--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 **slowdown 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: $U_0, U_1, \dots, U_{K-1}$, the encoded bits will be organized as follow: $X_0^{sn}, X_0^{pn}, X_0^{pi}, \dots, X_{K-1}^{sn}, X_{K-1}^{pn}, X_{K-1}^{pi}, X_0^{sn^t}, X_0^{pn^t}, \dots, X_{\log_2(ts)-1}^{sn^t}, X_{\log_2(ts)-1}^{pn^t}, X_0^{si^t}, X_0^{pi^t}, \dots, X_{\log_2(ts)-1}^{si^t}, X_{\log_2(ts)-1}^{pi^t}$, 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 ts the *trellis size*.

With the buffered encoding, considering the following sequence of K information bits: $U_0, U_1, \dots, U_{K-1}$, the encoded bits will be organized as follow: $X_0^{sn}, \dots, X_{K-1}^{sn}, X_0^{sn^t}, \dots, X_{\log_2(ts)-1}^{sn^t}, X_0^{pn}, \dots, X_{K-1}^{pn}, X_0^{pn^t}, \dots, X_{\log_2(ts)-1}^{pn^t}, X_0^{si^t}, \dots, X_{\log_2(ts)-1}^{si^t}, X_0^{pi^t}, \dots, X_{K-1}^{pi^t}, X_0^{pi}, \dots, X_{K-1}^{pi}$, 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 ts the *trellis size*.

--enc-sub-poly

Please refer to the RSC [--enc-poly](#) parameter.

--enc-sub-std

Type text

Allowed values CCSDS LTE

Examples --enc-sub-std CCSDS

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

Description of the allowed values:

Value	Description
CCSDS	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).
LTE	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).

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:

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

--dec-implement**Type** text**Allowed values** STD FAST**Default** FAST**Examples** --dec-implement FAST

Select the implementation of the decoder algorithm.

Description of the allowed values:

Value	Description
STD	Select the STD implementation.
FAST	Select the fast implementation from [Trb-CTL+16].

--dec-sub-type, -D

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

--dec-sub-implement

Please refer to the RSC *--dec-implement* 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 [Trb-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 [Trb-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 [Trb-VF00].

Description of the allowed values:

Value	Description
ADAPTIVE	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.
ARRAY	Select an hard-coded array of SFs (Scaling Factors) (c.f. Table 3.3).
CST	Set the same SF to be applied for each iterations.
LTE	Select a 0.75 SF.
LTE_VEC	Select a 0.75 vectorized SF (faster than LTE), used in [Rsc-CTL+16] .

Table 3.3: Hard-coded array of SFs.

Iteration	Value
1	0.15
2	0.25
3	0.30
4	0.40
5	0.70
6	0.80
7	0.90
8	0.95

`--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:

Value	Description
NO	Disable the puncturer.
TURBO	Enable the puncturing patterns.

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:

Value	Description
TURBO_DB	Select the standard Turbo DB encoder.
AZCW	See the common <code>--enc-type</code> parameter.
COSET	See the common <code>--enc-type</code> parameter.
USER	See the common <code>--enc-type</code> parameter.

--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:

Value	Description
DVB-RCS1	Set the DVB-RCS1 trellis and select the DVB-RCS1 interleaver (see the <i>--itl-type</i> parameter).
DVB-RCS2	Set the DVB-RCS2 trellis and select the DVB-RCS2 interleaver (see the <i>--itl-type</i> parameter).

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:

Value	Description
TURBO_DB	Select the standard Turbo decoder.
CHASE	See the common <i>--dec-type, -D</i> parameter.
ML	See the common <i>--dec-type, -D</i> parameter.

--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:

Value	Description
STD	Select the STD implementation.

--dec-sub-type, -D

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

--dec-sub-implement

Please refer to the RSC DB *--dec-implement* 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 [Trdb-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 [Trdb-VF00].

Description of the allowed values:

Value	Description
ADAPTIVE	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.
ARRAY	Select an hard-coded array of SFs (c.f. Table 3.3).
CST	Set the same SF to be applied for each iterations.
LTE	Select a 0.75 SF.
LTE_VEC	Select a 0.75 vectorized SF (faster than LTE).

Table 3.4: Hard-coded array of SFs.

Iteration	Value
1	0.15
2	0.25
3	0.30
4	0.40
5	0.70
6	0.80
7	0.90
8	0.95

--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:

Value	Description
NO	Disable the puncturer.
TURBO_DB	Enable the puncturer.

`--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`

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**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:

Value	Description
TPC	The TPC encoder.
AZCW	See the common <i>--enc-type</i> parameter.
COSET	See the common <i>--enc-type</i> parameter.
USER	See the common <i>--enc-type</i> parameter.

--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_{i+1}^c is the $N \times N$ *a posteriori* matrix computed by this decoder after the i^{th} iteration on the columns. Initially, $R_0^c = C$.

Let's say R_{i+1}^r is the $N \times N$ *a posteriori* matrix computed by this decoder after the i^{th} iteration on the rows, with $R_i^r = R_{i+1}^c$.

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

$$R_{i+1}^c = \alpha_{2i+0}.W_i^c + C$$

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:

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

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

Parameter *alpha* is set with the argument `--dec-alpha`.

`--dec-sub-cw-size, -N`

Type integer

Examples `--dec-sub-cw-size 1`

Set the codeword size N .

`--dec-sub-info-bits, -K`

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:

Value	Description
CP	Decode with the Chase-Pyndiah algorithm of the TPC
CHASE	See the common <code>--dec-type, -D</code> parameter.
ML	See the common <code>--dec-type, -D</code> parameter.

The CP algorithm is the implementation of [Trbp-Pyn98] 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^s the competitor with the smallest metric C_m that have $C_j^s \neq D_j$.

* when C^s exists:

$$F_j = b.D_j.[C_m - D_m]$$

* when C^s does not exist and if `--dec-beta` is given:

$$F_j = D_j.beta$$

* else:

$$F_j = D_j \cdot \left[\sum_{i=0}^e P_i - c.D_m + d.|R_j| \right] \text{ where } P \text{ is considered sorted, } 0 < e < p, \text{ and when } e == 0 \implies e = p - 1.$$

- 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:

Value	Description
STD	A standard implementation

--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`

Select the decoder algorithm.

Description of the allowed values:

Value	Description
NONE	Select the NONE decoder.
CHASE	See the common <code>--dec-type, -D</code> parameter.
ML	See the common <code>--dec-type, -D</code> parameter.

`--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:

Value	Description
HARD_DECISION	Take the hard decision on the input LLRs.

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:

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

Fig. 3.8: Interleaver NO.

Fig. 3.9: Interleaver COL_ROW.

Fig. 3.10: Interleaver ROW_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:

Fig. 3.11: Interleaver RANDOM.

Fig. 3.12: Interleaver RAND_COL.

```
# 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
```

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 bellow):

Fig. 3.13: Interleaver USER.

Value	Description
TOP_LEFT	Read is down from the top left corner to the bottom right corner.
TOP_RIGHT	Read is down from the top right corner to the bottom left corner.
BOTTOM_LEFT	Read is down from the bottom left corner to the top right corner.
BOTTOM_RIGHT	Read is down from the bottom right corner to the top left corner.

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

Fig. 3.14: Interleaver COL_ROW read orders.

Fig. 3.15 depicts the read order options on the ROW_COL interleaver.

Fig. 3.15: Interleaver 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:

Value	Description
BPSK	Select a BPSK (Bit Phase-Shift Keying) modulation.
CPM	Select a Continuous Phase Modulation (CPM (Continuous Phase Modulation)) [Mdm-ARS81 , Mdm-AS81].
OOK	Select an On-Off Keying (OOK (On-Off Keying)) modulation.
PAM	Select a Pulse-Amplitude Modulation (PAM (Pulse-Amplitude Modulation)).
PSK	Select a Phase-Shift Keying (PSK (Phase-Shift Keying)) modulation.
QAM	Select a rectangular Quadrature-Amplitude Modulation (QAM (Quadrature Amplitude Modulation)).
SCMA	Select a Sparse Code Multiple Access (SCMA) modulation [Mdm-NB13].
USER	Select a user defined constellation (to use with the <i>--mdm-const-path</i> parameter).

--mdm-implement**Type** text**Allowed values** FAST STD**Default** STD**Examples** --mdm-implement FAST

Select the MODEM implementation.

Description of the allowed values:

Value	Description
STD	Select a standard implementation working for any MODEM.
FAST	Select a fast implementation.

--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):

```
# 0000
3 3
# 0001
-3 -3
# 0010
-1 3
# 0011
1 -3
# 0100
-3 1
# 0101
3 -1
# 0110
1 1
# 0111
-1 -1
# 1000
1 -1
# 1001
-1 1
# 1010
-3 -1
# 1011
3 1
# 1100
-1 -3
# 1101
1 3
# 1110
3 -3
# 1111
-3 3
```

Note: 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 MAXS MAXSS 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	$\max^*(a, b) \approx \max(a, b) + d$ with $d = \begin{cases} 0 & \text{if } d \geq 37 \\ \exp(- a - b) & \text{if } 9 \leq d < 37. \\ \log(1 + \exp(- a - b)) & \text{else} \end{cases}$
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 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 σ^2 in the demodulator where σ 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:

Value	Description
GRAY	Gray code switching only one bit at a time from a symbol to the following.
NATURAL	The natural binary code incrementing the value from a symbol to the next one.

--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:

Value	Description
GMSK	Gaussian Minimum Shift Keying.
RCOS	Raised COSinus.
REC	RECTangular.

--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:

Value	Parameter	Value	Description
GSM	<i>--mdm-bps</i>	1	Bit per symbol.
	<i>--mdm-cpm-upf</i>	5	Upsampling factor.
	<i>--mdm-cpm-k</i>	1	Modulation index numerator.
	<i>--mdm-cpm-p</i>	2	Modulation index denominator.
	<i>--mdm-cpm-L</i>	3	Memory depth.
	<i>--mdm-cpm-map</i>	NATURAL	Mapping layout.
	<i>--mdm-cpm-ws</i>	GMSK	Wave shape.

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

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

Set the number of iterations in the SCMA demodulator.

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

Select the ψ function used in the SCMA demodulator.

Description of the allowed values:

Value	Description
PSI0	$\psi_0 = \exp\left(-\frac{ d }{n_0}\right)$
PSI1	$\psi_1 \approx \psi_0 \approx \frac{1}{ d +n_0}$
PSI2	$\psi_2 \approx \psi_0 \approx \frac{1}{8 \cdot d ^2 + n_0}$
PSI3	$\psi_3 \approx \psi_0 \approx \frac{1}{4 \cdot d ^2 + n_0}$

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

See the `--mdm-no-sig2` parameter to disable the division by σ^2 .

`--mdm-cb-path`

Type file

Rights read/write

Examples `--mdm-cb-path conf/mod/SCMA/CS1.cb`

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 $2M$ columns (real and imaginary parts):

V	K	M
Re(User 1, Resource 1, Code 1)	Im(User 1, Resource 1, Code 1)	...
↪ Resource 1, Code M	Im(User 1, Resource 1, Code M)	Re(User 1, ↪
...	...	...
↪	...	↪
Re(User 1, Resource K, Code 1)	Im(User 1, Resource K, Code 1)	...
↪ Resource K, Code M	Im(User 1, Resource K, Code M)	Re(User 1, ↪
Re(User 2, Resource 1, Code 1)	Im(User 2, Resource 1, Code 1)	...
↪ Resource 1, Code M	Im(User 2, Resource 1, Code M)	Re(User 2, ↪
...	...	...
↪	...	↪
Re(User 2, Resource K, Code 1)	Im(User 2, Resource K, Code 1)	...
↪ Resource K, Code M	Im(User 2, Resource K, Code M)	Re(User 2, ↪
...	...	...
↪	...	↪
Re(User V, Resource 1, Code 1)	Im(User V, Resource 1, Code 1)	...
↪ Resource 1, Code M	Im(User V, Resource 1, Code M)	Re(User V, ↪
...	...	...
↪	...	↪
Re(User V, Resource K, Code 1)	Im(User V, Resource K, Code 1)	...
↪ Resource K, Code M	Im(User V, Resource K, Code M)	Re(User V, ↪

Descriptions of the codebooks of the configuration files

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

Codebook Set	Description
CS1	From [Scma-Pro]
CS2	LDS based on QPSK constellation
CS3	Based on [Scma-CWC15] and own optimization for AWGN channel
CS4	From [Scma-ZXX+16]
CS5	From [Scma-SWC17]
CS6	From [Scma-KS17]
CS7	From [Scma-KS17]
CS8	From [Scma-WZC15]

The simulation results for **CS1-CS7** (AWGN and Rayleigh fading channels) can be found in [Scma-KS17]. The simulation results for **CS8** can be found in [Scma-KS16] (defined as **CS2** in the paper).

--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).

--chn-type

Type text

Allowed values NO BEC BSC AWGN RAYLEIGH RAYLEIGH_USER OPTICAL USER USER_ADD
USER_BEC USER_BSC

Default AWGN

Examples --chn-type AWGN

Select the channel type.

Description of the allowed values:

Value De-		scrip-
		tion
3.2. Parameters	NO	Dis- able the chan- nel noise: $Y = X$.
	BEC	Se- lect the Bi- nary Era- sure Chan- nel (BEC): $Y_i = \begin{cases} \text{erased} & \text{if } e = 1 \\ X_i & \text{else} \end{cases}$, with $P(e = 1) = p_e$ and $P(e = 0) = 1 - p_e$.
	BSC	Se- lect the Bi- nary Sym- met- ric Chan- nel (BSC (Bi- nary Sym- met- ric Chan- nel))): $Y_i = \begin{cases} !X_i & \text{if } e = 1 \\ X_i & \text{else} \end{cases}$, with $P(e = 1) = p_e$ and $P(e = 0) = 1 - p_e$.
		AWGN
		Se- lect the Ad

Where:

- σ 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-implement

Type text

Allowed values STD FAST GSL MKL

Default STD

Examples `--chn-implement FAST`

Select the implementation of the algorithm to generate the noise.

Description of the allowed values:

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

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

Frame size	STD	FAST	MKL	GSL
16	31,80	99,12	29,80	41,39
32	30,05	134,94	53,72	46,76
64	30,78	165,92	93,80	52,79
128	31,24	188,06	148,31	54,77
256	31,41	199,14	204,53	55,38
512	31,52	199,43	267,74	55,49
1024	31,79	199,71	331,71	56,15
2048	31,61	200,16	342,06	56,32
4096	31,88	198,88	343,40	57,42
8192	30,43	195,78	342,59	56,92

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

Frame size	STD	FAST	MKL	GSL
16	36,18	114,87	104,1	58,92
32	40,28	170,64	184,99	69,14
64	42,84	223,78	319,65	77,26
128	43,28	252,41	474,18	87,78
256	43,42	272,82	624,92	93,71
512	43,51	273,22	738,72	95,36
1024	43,64	275,41	865,25	97,84
2048	43,63	272,78	996,88	97,25
4096	42,78	274,71	1109,13	97,65
8192	43,67	272,71	1116,41	98,47

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

Frame size	STD	FAST	MKL	GSL
16	6,69	7,56	6,71	6,53
32	9,89	10,98	10,28	9,56
64	12,67	14,30	14,05	12,15
128	14,40	16,33	16,33	13,88
256	15,82	17,74	18,22	15,07
512	16,52	18,46	18,29	15,79
1024	17,18	19,14	19,31	16,19
2048	16,96	18,76	20,30	16,42
4096	17,19	18,65	20,29	16,47
8192	17,26	18,98	20,58	16,63

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** NO FRAME 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.

References

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 `--sim-prec`, `-p` parameter).

Warning: Some decoders are not *fixed-point ready*, please refer to the decoders documentation for more details.

`--qnt-type`

Type text

Allowed values CUSTOM POW2

Default POW2

Examples `--qnt-type CUSTOM`

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):

Value	Description
CUSTOM	$Q_i = \begin{cases} +v_{sat} & \text{when } v_i > +v_{sat} \\ -v_{sat} & \text{when } v_i < -v_{sat}, \text{ with } v_i = \lfloor \frac{Y_i}{\Delta} \rfloor \text{ and } v_{sat} = 2^{p_b-1} - 1 \text{ and } \Delta = \frac{ p_r }{v_{sat}} \\ v_i & \text{else} \end{cases}$
POW2	$Q_i = \begin{cases} +v_{sat} & \text{when } v_i > +v_{sat} \\ -v_{sat} & \text{when } v_i < -v_{sat}, \text{ with } v_i = \lfloor Y_i * F \rfloor \text{ and } v_{sat} = 2^{p_b-1} - 1 \text{ and } F = 2^{p_d} \\ v_i & \text{else} \end{cases}$

Where p_r , p_b and p_d are respectively given through the `--qnt-range`, `--qnt-bits` and `--qnt-dec` parameters.

`--qnt-implement`

Type text

Allowed values STD FAST

Default STD

Examples `--qnt-implement FAST`

Select the implementation of the quantizer.

Description of the allowed values:

Value	Description
STD	Select a standard implementation.
FAST	Select a fast implementation, only available for the POW2 quantizer.

--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.

Code	Value
LDPC	6
POLAR	6
REP	6
RSC	6
RSC_DB	6
TPC	6 on 8-bit and 8 on 16-bit
TURBO	6
TURBO_DB	6

--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.

Code	Value
LDPC	2
POLAR	1
REP	2
RSC	1 on 8-bit and 3 on 16-bit
RSC_DB	1 on 8-bit and 3 on 16-bit
TPC	2 on 8-bit and 3 on 16-bit
TURBO	2 on 8-bit and 3 on 16-bit
TURBO_DB	2 on 8-bit and 3 on 16-bit

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**Rights** write only**Default** ./hist**Examples** --mnt-err-hist-path my/histogram/root/path/name

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.

Note: This parameter is not available if the code has been compiled with MPI.

--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:

Value	Description
STD	Select the standard format.

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 (σ) 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}>
```

```
    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
```

(continues on next page)

(continued from previous page)

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

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}>

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-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

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::p+cde-type,C]
```

Select the channel code family to simulate.

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

```
[factory::Launcher::p+prec,p]
```

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

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

```
[factory::Launcher::p+type]
```

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

Other parameter(s):

```
--Help, -H
```

```
[factory::Launcher::Help,H]
```

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

```
--help, -h
```

```
[factory::Launcher::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
```

(continues on next page)

(continued from previous page)

<code>[factory::Launcher::no-colors]</code>
Disable the colors in the shell.
<code>--version, -v</code>
<code>[factory::Launcher::version,v]</code>
Print informations about the version of the source code and compilation options.

--except-no-bt

Disable the backtrace display when running an exception.

--no-legend

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

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.

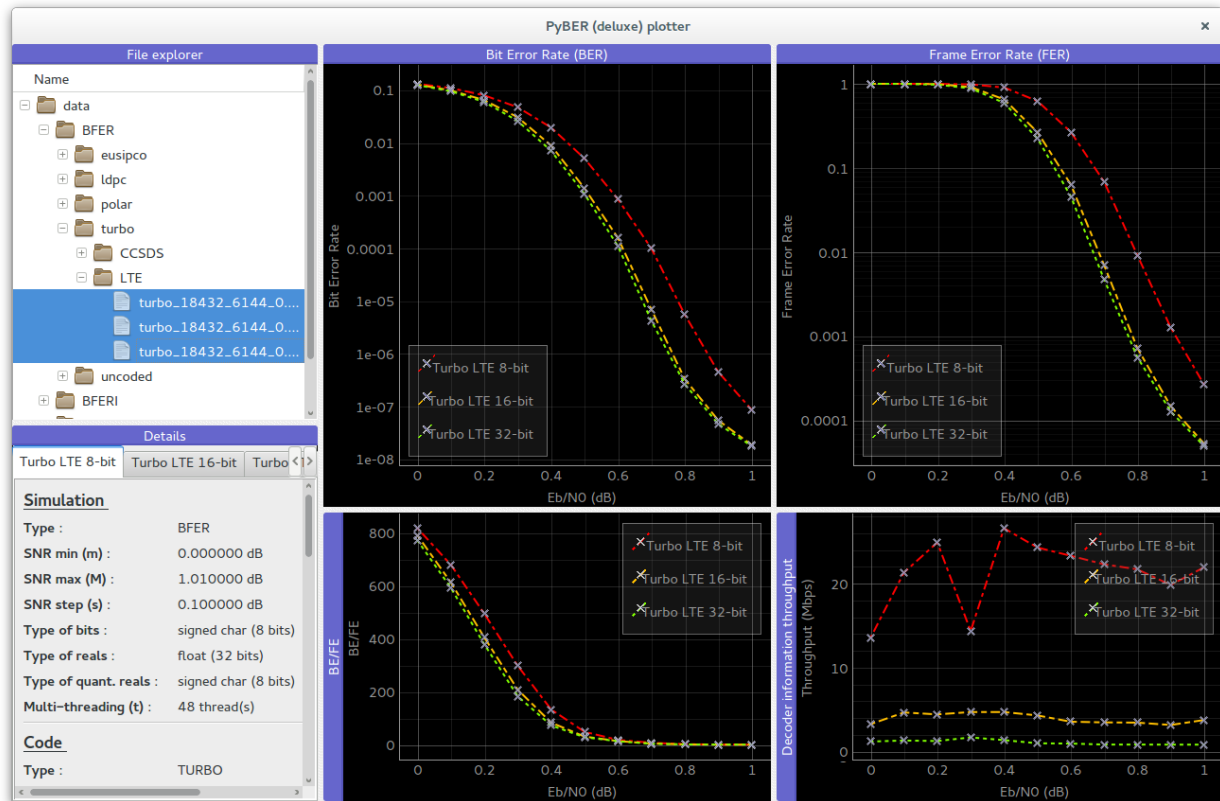
--no-colors

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 &
```


LIBRARY EXAMPLES

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.

Fig. 4.1: Simulated communication chain.

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

```
1  #include <aff3ct.hpp>
2  using namespace aff3ct;
3
4  int main(int argc, char** argv)
5  {
6      params1 p; init_params1 (p ); // create and initialize the parameters defined
    ↳ by the user
7      modules1 m; init_modules1(p, m); // create and initialize modules
8      buffers1 b; init_buffers1(p, b); // create and initialize the buffers required
    ↳ by the modules
9      utils1 u;  init_utils1 (m, u); // create and initialize utils
10
```

(continues on next page)

(continued from previous page)

```

11 // display the legend in the terminal
12 u.terminal->legend();
13
14 // loop over SNRs range
15 for (auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
16 {
17     // compute the current sigma for the channel noise
18     const auto esn0 = tools::ebn0_to_esn0 (ebn0, p.R);
19     const auto sigma = tools::esn0_to_sigma(esn0 );
20
21     u.noise->set_values(sigma, ebn0, esn0);
22
23     // update the sigma of the modem and the channel
24     m.modem ->set_noise(*u.noise);
25     m.channel->set_noise(*u.noise);
26
27     // display the performance (BER and FER) in real time (in a separate
    ↪ thread)
28     u.terminal->start_temp_report();
29
30     // run the simulation chain
31     while (!m.monitor->fe_limit_achieved())
32     {
33         m.source ->generate      ( b.ref_bits );
34         m.encoder->encode      ( b.ref_bits, b.enc_bits );
35         m.modem ->modulate      ( b.enc_bits, b.symbols );
36         m.channel->add_noise     ( b.symbols, b.noisy_symbols);
37         m.modem ->demodulate     ( b.noisy_symbols, b.LLRs );
38         m.decoder->decode_siho ( b.LLRs, b.dec_bits );
39         m.monitor->check_errors(b.dec_bits, b.ref_bits );
40     }
41
42     // display the performance (BER and FER) in the terminal
43     u.terminal->final_report();
44
45     // reset the monitor for the next SNR
46     m.monitor->reset();
47     u.terminal->reset();
48 }
49
50 return 0;
51 }

```

Listing 4.1 gives an overview of what can be achieved with the AFF3CT library. The firsts 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 `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 `int32_t` or `float`.

Listing 4.2: Bootstrap: parameters

```

1 struct params1
2 {
3     int    K        = 32;    // number of information bits
4     int    N        = 128;   // codeword size
5     int    fe       = 100;   // number of frame errors
6     int    seed     = 0;     // PRNG seed for the AWGN channel
7     float  ebn0_min  = 0.00f; // minimum SNR value
8     float  ebn0_max  = 10.01f; // maximum SNR value
9     float  ebn0_step = 1.00f; // SNR step
10    float  R;             // code rate (R=K/N)
11 };
12
13 void init_params1(params1 &p)
14 {
15     p.R = (float)p.K / (float)p.N;
16 }

```

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

```

1 struct modules1
2 {
3     std::unique_ptr<module::Source_random<>>    source;
4     std::unique_ptr<module::Encoder_repetition_sys<>> encoder;
5     std::unique_ptr<module::Modem_BPSK<>>        modem;
6     std::unique_ptr<module::Channel_AWGN_LLR<>>  channel;
7     std::unique_ptr<module::Decoder_repetition_std<>> decoder;
8     std::unique_ptr<module::Monitor_BFER<>>      monitor;
9 };
10
11 void init_modules1(const params1 &p, modules1 &m)
12 {
13     m.source = std::unique_ptr<module::Source_random<>>(new module::Source_
14     ↪ random          <>(p.K      ));
15     m.encoder = std::unique_ptr<module::Encoder_repetition_sys<>>(new
16     ↪ module::Encoder_repetition_sys<>(p.K, p.N  ));
17     m.modem   = std::unique_ptr<module::Modem_BPSK<>>(new module::Modem_
18     ↪ BPSK          <>(p.N      ));
19     m.channel = std::unique_ptr<module::Channel_AWGN_LLR<>>(new
20     ↪ module::Channel_AWGN_LLR    <>(p.N, p.seed));
21     m.decoder = std::unique_ptr<module::Decoder_repetition_std<>>(new
22     ↪ module::Decoder_repetition_std<>(p.K, p.N  ));
23     m.monitor = std::unique_ptr<module::Monitor_BFER<>>(new
24     ↪ module::Monitor_BFER        <>(p.K, p.fe ));
25 }

```

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

```

1 struct buffers1
2 {
3     std::vector<int> > ref_bits;
4     std::vector<int> > enc_bits;
5     std::vector<float> symbols;
6     std::vector<float> noisy_symbols;
7     std::vector<float> LLRs;
8     std::vector<int> > dec_bits;
9 };
10
11 void init_buffers1(const params1 &p, buffers1 &b)
12 {
13     b.ref_bits      = std::vector<int> >(p.K);
14     b.enc_bits      = std::vector<int> >(p.N);
15     b.symbols       = std::vector<float>(p.N);
16     b.noisy_symbols = std::vector<float>(p.N);
17     b.LLRs          = std::vector<float>(p.N);
18     b.dec_bits      = std::vector<int> >(p.K);
19 }

```

[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

```

1 struct utils1
2 {
3     std::unique_ptr<tools::Sigma<>> noise;    // a sigma noise type
4     std::vector<std::unique_ptr<tools::Reporter>> reporters; // list of reporters
5     ↪ displayed in the terminal
6     std::unique_ptr<tools::Terminal_std> terminal; // manage the output
7     ↪ text in the terminal
8 };
9
10 void init_utils1(const modules1 &m, utils1 &u)
11 {
12     // create a sigma noise type
13     u.noise = std::unique_ptr<tools::Sigma<>>(new tools::Sigma<>());
14     // report the noise values (Es/N0 and Eb/N0)
15     u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_noise
16     ↪ <>(*u.noise)));
17     // report the bit/frame error rates
18     u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_BFER<
19     ↪ (*m.monitor)));
20     // report the simulation throughputs
21     u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_
22     ↪ throughput<>(*m.monitor)));

```

(continues on next page)

(continued from previous page)

```

18 // create a terminal that will display the collected data from the reporters
19 u.terminal = std::unique_ptr<tools::Terminal_std>(new tools::Terminal_std(u.
20 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

```

# -----|-----|-----|-----|
# -----
# Signal Noise Ratio || Bit Error Rate (BER) and Frame Error Rate (FER) ||
# Global throughput
# (SNR) || and
# elapsed time
# -----|-----|-----|-----|
# -----
# -----|-----|-----|-----|-----|-----|-----|-----|
# --|-----
# Es/N0 | Eb/N0 || FRA | BE | FE | BER | FER || SIM_
# THR | ET/RT
# (dB) | (dB) || | | | | | | (Mb/
# s) | (hh:mm:ss)
# -----|-----|-----|-----|-----|-----|-----|
# --|-----
# -6.02 | 0.00 || 108 | 262 | 100 | 7.58e-02 | 9.26e-01 || 2.
# 382 | 00h00'00
# -5.02 | 1.00 || 125 | 214 | 100 | 5.35e-02 | 8.00e-01 || 4.
# 813 | 00h00'00
# -4.02 | 2.00 || 136 | 179 | 100 | 4.11e-02 | 7.35e-01 || 3.
# 804 | 00h00'00
# -3.02 | 3.00 || 210 | 135 | 100 | 2.01e-02 | 4.76e-01 || 4.
# 516 | 00h00'00
# -2.02 | 4.00 || 327 | 122 | 100 | 1.17e-02 | 3.06e-01 || 5.
# 157 | 00h00'00
# -1.02 | 5.00 || 555 | 112 | 100 | 6.31e-03 | 1.80e-01 || 4.
# 703 | 00h00'00
# -0.02 | 6.00 || 1619 | 108 | 100 | 2.08e-03 | 6.18e-02 || 4.
# 110 | 00h00'00
# 0.98 | 7.00 || 4566 | 102 | 100 | 6.98e-04 | 2.19e-02 || 4.
# 974 | 00h00'00
# 1.98 | 8.00 || 15998 | 100 | 100 | 1.95e-04 | 6.25e-03 || 4.
# 980 | 00h00'00
# 2.98 | 9.00 || 93840 | 100 | 100 | 3.33e-05 | 1.07e-03 || 5.
# 418 | 00h00'00
# 3.98 | 10.00 || 866433 | 100 | 100 | 3.61e-06 | 1.15e-04 || 4.
# 931 | 00h00'05

```

Note: The full source code is available here: https://github.com/aff3ct/my_project_with_aff3ct/blob/master/examples/bootstrap/src/main.cpp.

4.2 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

```

1  #include <aff3ct.hpp>
2  using namespace aff3ct;
3
4  int main(int argc, char** argv)
5  {
6      params1 p; init_params1 (p ); // create and initialize the parameters defined
    ↪ by the user
7      modules1 m; init_modules2(p, m); // create and initialize modules
8      // the 'init_buffers1' function is not required anymore
9      utils1 u; init_utils1 (m, u); // create and initialize the utils
10
11     // display the legend in the terminal
12     u.terminal->legend();
13
14     // 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)
15     using namespace module;
16     (*m.encoder)[enc::sck::encode      ::U_K ].bind((*m.source )[src::sck::generate
    ↪ ::U_K ]);
17     (*m.modem )[mdm::sck::modulate     ::X_N1].bind((*m.encoder)[enc::sck::encode
    ↪ ::X_N ]);
18     (*m.channel)[chn::sck::add_noise   ::X_N ].bind((*m.modem )[mdm::sck::modulate
    ↪ ::X_N2]);
19     (*m.modem )[mdm::sck::demodulate    ::Y_N1].bind((*m.channel)[chn::sck::add_noise
    ↪ ::Y_N ]);
20     (*m.decoder)[dec::sck::decode_siho ::Y_N ].bind((*m.modem
    ↪ [mdm::sck::demodulate ::Y_N2]);
21     (*m.monitor)[mnt::sck::check_errors::U   ].bind((*m.encoder)[enc::sck::encode
    ↪ ::U_K ]);
22     (*m.monitor)[mnt::sck::check_errors::V   ].bind((*m.decoder)[dec::sck::decode_
    ↪ siho::V_K ]);
23
24     // loop over the range of SNRs
25     for (auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
26     {
27         // compute the current sigma for the channel noise
28         const auto esn0 = tools::ebn0_to_esn0 (ebn0, p.R);
29         const auto sigma = tools::esn0_to_sigma(esn0 );
30

```

(continues on next page)

(continued from previous page)

```

31     u.noise->set_values(sigma, ebn0, esn0);
32
33     // update the sigma of the modem and the channel
34     m.modem ->set_noise(*u.noise);
35     m.channel->set_noise(*u.noise);
36
37     // display the performance (BER and FER) in real time (in a separate
↪ thread)
38     u.terminal->start_temp_report();
39
40     // run the simulation chain
41     while (!m.monitor->fe_limit_achieved())
42     {
43         (*m.source ) [src::tsk::generate   ].exec();
44         (*m.encoder)[enc::tsk::encode     ].exec();
45         (*m.modem  ) [mdm::tsk::modulate   ].exec();
46         (*m.channel)[chn::tsk::add_noise   ].exec();
47         (*m.modem  ) [mdm::tsk::demodulate ].exec();
48         (*m.decoder)[dec::tsk::decode_siho ].exec();
49         (*m.monitor)[mnt::tsk::check_errors].exec();
50     }
51
52     // display the performance (BER and FER) in the terminal
53     u.terminal->final_report();
54
55     // reset the monitor and the terminal for the next SNR
56     m.monitor->reset();
57     u.terminal->reset();
58 }
59
60 // display the statistics of the tasks (if enabled)
61 tools::Stats::show({ m.source.get(), m.encoder.get(), m.modem.get(), m.channel.
↪ get(), m.decoder.get(), m.monitor.get() }, true);
62
63 return 0;
64 }

```

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 4.8: Tasks: modules

```

1 void init_modules2(const params1 &p, modules1 &m)
2 {

```

(continues on next page)

(continued from previous page)

```

3      m.source = std::unique_ptr<module::Source_random><>>(new module::Source_
↳random      <>(p.K      ));
4      m.encoder = std::unique_ptr<module::Encoder_repetition_sys><>>(new_
↳module::Encoder_repetition_sys<>(p.K, p.N  ));
5      m.modem = std::unique_ptr<module::Modem_BPSK><>>(new module::Modem_
↳BPSK      <>(p.N      ));
6      m.channel = std::unique_ptr<module::Channel_AWGN_LL_R><>>(new_
↳module::Channel_AWGN_LL_R      <>(p.N, p.seed));
7      m.decoder = std::unique_ptr<module::Decoder_repetition_std><>>(new_
↳module::Decoder_repetition_std<>(p.K, p.N  ));
8      m.monitor = std::unique_ptr<module::Monitor_BFER><>>(new_
↳module::Monitor_BFER      <>(p.K, p.fe ));
9
10     // configuration of the module tasks
11     std::vector<const module::Module*> modules_list = { m.source.get(), m.encoder.
↳get(), m.modem.get(), m.channel.get(), m.decoder.get(), m.monitor.get() };
12     for (auto& mod : modules_list)
13         for (auto& tsk : mod->tasks)
14             {
15                 tsk->set_autoalloc (true ); // enable the automatic allocation_
↳of data buffers in the tasks
16                 tsk->set_debug      (false); // disable the debug mode
17                 tsk->set_debug_limit(16 ); // display only the 16 first bits_
↳if the debug mode is enabled
18                 tsk->set_stats      (true ); // enable statistics collection
19
20                 // enable fast mode (= disable optional checks in the tasks) if_
↳there is no debug and stats modes
21                 if (!tsk->is_debug() && !tsk->is_stats())
22                     tsk->set_fast(true);
23             }
24 }

```

The beginning of the `init_modules2` function (Listing 4.8) is the same as the `init_module1` function (Listing 4.3). At lines 10–23, 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 Factory

In the previous *Bootstrap* and *Tasks* 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 4.9: Factory: main function

```

1  #include <aff3ct.hpp>
2  using namespace aff3ct;
3
4  int main(int argc, char** argv)
5  {
6      params3 p; init_params3 (argc, argv, p); // create and initialize the
        ↪ parameters from the command line with factories
7      modules3 m; init_modules3(p, m          ); // create and initialize modules
8      utils1 u; init_utils3 (p, m, u        ); // create and initialize utils
9
10     // [...]
11
12     // display the statistics of the tasks (if enabled)
13     tools::Stats::show({ m.source.get(), m.modem.get(), m.channel.get(), m.monitor.
        ↪ get(), m.encoder, m.decoder }, true);
14
15     return 0;
16 }
```

The main function in Listing 4.9 is almost unchanged from the main function in Listing 4.7.

Listing 4.10: Factory: parameters

```

1  struct params3
2  {
3      float ebn0_min = 0.00f; // minimum SNR value
4      float ebn0_max = 10.01f; // maximum SNR value
5      float ebn0_step = 1.00f; // SNR step
6      float R;           // code rate (R=K/N)
7
8      std::unique_ptr<factory::Source> source;
9      std::unique_ptr<factory::Codec_repetition> codec;
10     std::unique_ptr<factory::Modem> modem;
11     std::unique_ptr<factory::Channel> channel;
12     std::unique_ptr<factory::Monitor_BFER> monitor;
13     std::unique_ptr<factory::Terminal> terminal;
14 };
15
16 void init_params3(int argc, char** argv, params3 &p)
17 {
18     p.source = std::unique_ptr<factory::Source>(new factory::Source
        ↪ ());
19     p.codec = std::unique_ptr<factory::Codec_repetition>(new factory::Codec_
        ↪ repetition());
20     p.modem = std::unique_ptr<factory::Modem>(new factory::Modem
        ↪ ());
```

(continues on next page)

(continued from previous page)

```

21     p.channel = std::unique_ptr<factory::Channel>(new factory::Channel
↳   ());
22     p.monitor = std::unique_ptr<factory::Monitor_BFER>(new factory::Monitor_
↳   BFER ());
23     p.terminal = std::unique_ptr<factory::Terminal>(new factory::Terminal
↳   ());
24
25     std::vector<factory::Factory*> params_list = { p.source .get(), p.codec .get(),
↳   p.modem .get(),
26                                                     p.channel.get(), p.monitor.get(),
↳   p.terminal.get() };
27
28     // parse command line arguments for the given parameters and fill them
29     tools::Command_parser cp(argc, argv, params_list, true);
30     if (cp.parsing_failed())
31     {
32         cp.print_help    ();
33         cp.print_warnings();
34         cp.print_errors  ();
35         std::exit(1);
36     }
37
38     std::cout << "# Simulation parameters: " << std::endl;
39     tools::Header::print_parameters(params_list); // display the headers (= print
↳   the AFF3CT parameters on the screen)
40     std::cout << "#" << std::endl;
41     cp.print_warnings();
42
43     p.R = (float)p.codec->enc->K / (float)p.codec->enc->N_cw; // compute the code
↳   rate
44 }

```

The `params3` structure from Listing 4.10 contains some pointers to factory objects (lines 8-13). SNR parameters remain static in this example.

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 `tools::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 4.11: Factory: modules

```

1 struct modules3
2 {
3     std::unique_ptr<module::Source<>>    source;
4     std::unique_ptr<tools::Codec_SIHO<>> codec;
5     std::unique_ptr<module::Modem<>>    modem;
6     std::unique_ptr<module::Channel<>>   channel;
7     std::unique_ptr<module::Monitor_BFER<>> monitor;
8     module::Encoder<>*                  encoder;

```

(continues on next page)

(continued from previous page)

```

9      module::Decoder_SIHO<>* decoder;
10 };
11
12 void init_modules3(const params3 &p, modules3 &m)
13 {
14     m.source = std::unique_ptr<module::Source><>(p.source->build());
15     m.codec = std::unique_ptr<tools::Codec_SIHO><>(p.codec->build());
16     m.modem = std::unique_ptr<module::Modem><>(p.modem->build());
17     m.channel = std::unique_ptr<module::Channel><>(p.channel->build());
18     m.monitor = std::unique_ptr<module::Monitor_BFER><>(p.monitor->build());
19     m.encoder = m.codec->get_encoder().get();
20     m.decoder = m.codec->get_decoder_siho().get();
21
22     // configuration of the module tasks
23     std::vector<const module::Module*> modules_list = { m.source.get(), m.modem.
↳ get(), m.channel.get(), m.monitor.get(), m.encoder, m.decoder };
24     for (auto& mod : modules_list)
25         for (auto& tsk : mod->tasks)
26         {
27             tsk->set_autoalloc (true); // enable the automatic allocation.
↳ of the data in the tasks
28             tsk->set_debug      (false); // disable the debug mode
29             tsk->set_debug_limit(16); // display only the 16 first bits.
↳ if the debug mode is enabled
30             tsk->set_stats      (true); // enable the statistics
31
32             // enable the fast mode (= disable the useless verifs in the
↳ tasks) if there is no debug and stats modes
33             if (!tsk->is_debug() && !tsk->is_stats())
34                 tsk->set_fast(true);
35         }
36 }

```

In Listing 4.11 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 4.10 are used to build the modules (lines 14-18).

Listing 4.12: Factory: utils

```

1 void init_utils3(const params3 &p, const modules3 &m, utils1 &u)
2 {
3     // create a sigma noise type
4     u.noise = std::unique_ptr<tools::Sigma><>(new tools::Sigma<>());
5     // report noise values (Es/N0 and Eb/N0)
6     u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_noise
↳ <>(*u.noise)));
7     // report bit/frame error rates
8     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_
↳ throughput<>(*m.monitor)));

```

(continues on next page)

(continued from previous page)

```

11      // create a terminal object that will display the collected data from the
12      ↪reporters
13      u.terminal = std::unique_ptr<tools::Terminal>(p.terminal->build(u.reporters));
14  }

```

In the Listing 4.12, 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 4.13.

Listing 4.13: 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 4.14.

Listing 4.14: Factory: display the command line parameters

```
./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.

4.4 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 4.15: OpenMP: main function

```

1  int main(int argc, char** argv)
2  {
3      params3 p; init_params3(argc, argv, p); // create and initialize the parameters
4      ↪from the command line with factories
5      utils4 u; // create an 'utils4' structure
6
7      #pragma omp parallel
8      {
9          #pragma omp single
10         {
11             // get the number of available threads from OpenMP
12             const size_t n_threads = (size_t)omp_get_num_threads();
13             u.monitors.resize(n_threads);
14             u.modules.resize(n_threads);
15         }
16     }

```

(continues on next page)

(continued from previous page)

```

15     modules4 m; init_modules_and_utils4(p, m, u); // create and initialize the
↳ modules and initialize a part of the utils
16
17 #pragma omp barrier
18 #pragma omp single
19 {
20     init_utils4(p, u); // finalize the utils initialization
21
22     // display the legend in the terminal
23     u.terminal->legend();
24 }
25
26     // sockets binding (connect the sockets of the tasks = fill the input sockets
↳ with the output sockets)
27     using namespace module;
28     (*m.encoder)[enc::sck::encode      ::U_K ].bind((*m.source )[src::sck::generate
↳ ::U_K ]);
29     (*m.modem )[mdm::sck::modulate     ::X_N1].bind((*m.encoder)[enc::sck::encode
↳ ::X_N ]);
30     (*m.channel)[chn::sck::add_noise   ::X_N ].bind((*m.modem )[mdm::sck::modulate
↳ ::X_N2]);
31     (*m.modem )[mdm::sck::demodulate   ::Y_N1].bind((*m.channel)[chn::sck::add_noise
↳ ::Y_N ]);
32     (*m.decoder)[dec::sck::decode_siho ::Y_N ].bind((*m.modem
↳ [mdm::sck::demodulate ::Y_N2]);
33     (*m.monitor)[mnt::sck::check_errors::U   ].bind((*m.encoder)[enc::sck::encode
↳ ::U_K ]);
34     (*m.monitor)[mnt::sck::check_errors::V   ].bind((*m.decoder)[dec::sck::decode_
↳ siho::V_K ]);
35
36     // loop over the SNRs range
37     for (auto ebn0 = p.ebn0_min; ebn0 < p.ebn0_max; ebn0 += p.ebn0_step)
38     {
39         // compute the current sigma for the channel noise
40         const auto esn0 = tools::ebn0_to_esn0 (ebn0, p.R);
41         const auto sigma = tools::esn0_to_sigma(esn0 );
42
43 #pragma omp single
44     u.noise->set_values(sigma, ebn0, esn0);
45
46     // update the sigma of the modem and the channel
47     m.modem ->set_noise(*u.noise);
48     m.channel->set_noise(*u.noise);
49
50 #pragma omp single
51     // display the performance (BER and FER) in real time (in a separate
↳ thread)
52     u.terminal->start_temp_report();
53
54     // run the simulation chain
55     while (!u.monitor_red->is_done())
56     {
57         (*m.source )[src::tsk::generate      ].exec();

```

(continues on next page)

(continued from previous page)

```

57         (*m.encoder)[enc::tsk::encode    ].exec();
58         (*m.modem  )[mdm::tsk::modulate   ].exec();
59         (*m.channel)[chn::tsk::add_noise  ].exec();
60         (*m.modem  )[mdm::tsk::demodulate ].exec();
61         (*m.decoder)[dec::tsk::decode_siho].exec();
62         (*m.monitor)[mnt::tsk::check_errors].exec();
63     }
64
65     // need to wait all the threads here before to reset the 'monitors' and 'terminal' states
66     #pragma omp barrier
67     #pragma omp single
68     {
69         // final reduction
70         u.monitor_red->reduce();
71
72         // display the performance (BER and FER) in the terminal
73         u.terminal->final_report();
74
75         // reset the monitor and the terminal for the next SNR
76         u.monitor_red->reset();
77         u.terminal->reset();
78     }
79 }
80
81 #pragma omp single
82 {
83     // display the statistics of the tasks (if enabled)
84     tools::Stats::show(u.modules_stats, true);
85 }
86 }
87
88     return 0;
89 }

```

Listing 4.15 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 threads gets its own local `m`.

Listing 4.16: OpenMP: modules and utils

```

1  struct modules4
2  {
3      std::unique_ptr<module::Source<>>    source;
4      std::unique_ptr<tools::Codec_SIHO<>> codec;
5      std::unique_ptr<module::Modem<>>    modem;
6      std::unique_ptr<module::Channel<>>   channel;

```

(continues on next page)

(continued from previous page)

```

7         module::Monitor_BFER<>* monitor;
8         module::Encoder<>* encoder;
9         module::Decoder_SIHO<>* decoder;
10    };
11
12    struct utils4
13    {
14        std::unique_ptr<tools::Sigma<>> noise;           // a sigma_
15        ↪noise type
16        std::vector<std::unique_ptr<tools::Reporter>> reporters; // list of_
17        ↪reporters displayed in the terminal
18        std::unique_ptr<tools::Terminal> terminal;       // manage_
19        ↪the output text in the terminal
20        std::vector<std::unique_ptr<module::Monitor_BFER<>>> monitors; // list of_
21        ↪the monitors from all the threads
22        std::unique_ptr<tools::Monitor_BFER_reduction> monitor_red; // main_
23        ↪monitor object that reduce all the thread monitors
24        std::vector<std::vector<const module::Module*>> modules; // list of_
25        ↪the allocated modules
26        std::vector<std::vector<const module::Module*>> modules_stats; // list of_
27        ↪the allocated modules reorganized for the statistics
28    };
29
30    void init_modules_and_utils4(const params3 &p, modules4 &m, utils4 &u)
31    {
32        // get the thread id from OpenMP
33        const int tid = omp_get_thread_num();
34
35        // set different seeds for different threads when the module use a PRNG
36        p.source->seed += tid;
37        p.channel->seed += tid;
38
39        m.source      = std::unique_ptr<module::Source> (<>>(p.source ->build()));
40        m.codec       = std::unique_ptr<tools::Codec_SIHO> (<>>(p.codec ->build()));
41        m.modem       = std::unique_ptr<module::Modem> (<>>(p.modem ->build()));
42        m.channel     = std::unique_ptr<module::Channel> (<>>(p.channel->build()));
43        u.monitors[tid] = std::unique_ptr<module::Monitor_BFER<>> (<>>(p.monitor->build()));
44        m.monitor     = u.monitors[tid].get();
45        m.encoder     = m.codec->get_encoder().get();
46        m.decoder     = m.codec->get_decoder_siho().get();
47
48        // configuration of the module tasks
49        std::vector<const module::Module*> modules_list = { m.source.get(), m.modem.
50        ↪get(), m.channel.get(), m.monitor, m.encoder, m.decoder };
51        for (auto& mod : modules_list)
52            for (auto& tsk : mod->tasks)
53            {
54                tsk->set_autoalloc (true); // enable the automatic allocation_
55        ↪of the data in the tasks
56                tsk->set_debug      (false); // disable the debug mode
57                tsk->set_debug_limit(16); // display only the 16 first bits_
58        ↪if the debug mode is enabled

```

(continues on next page)

(continued from previous page)

```

49         tsk->set_stats      (true ); // enable the statistics
50
51         // enable the fast mode (= disable the useless verifs in the_
↳tasks) if there is no debug and stats modes
52         if (!tsk->is_debug() && !tsk->is_stats())
53             tsk->set_fast(true);
54     }
55
56     u.modules[tid] = modules_list;
57 }

```

In Listing 4.16, there is a change in the `modules4` structure compared to the `modules3` structure (Listing 4.11): 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 4.15).

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 56 a list of the modules is stored in `u`.

Listing 4.17: OpenMP: utils

```

1 void init_utils4(const params3 &p, utils4 &u)
2 {
3     // allocate a common monitor module to reduce all the monitors
4     u.monitor_red = std::unique_ptr<tools::Monitor_BFER_reduction>(new_
↳tools::Monitor_BFER_reduction(u.monitors));
5     u.monitor_red->set_reduce_frequency(std::chrono::milliseconds(500));
6     // create a sigma noise type
7     u.noise = std::unique_ptr<tools::Sigma<>>(new tools::Sigma<>());
8     // report noise values (Es/N0 and Eb/N0)
9     u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_noise
↳<>(*u.noise)));
10    // report bit/frame error rates
11    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_BFER<
↳(*u.monitor_red)));
12    // report simulation throughputs
13    u.reporters.push_back(std::unique_ptr<tools::Reporter>(new tools::Reporter_
↳throughput<>(*u.monitor_red)));
14    // create a terminal that will display the collected data from the reporters
15    u.terminal = std::unique_ptr<tools::Terminal>(p.terminal->build(u.reporters));
16
17    u.modules_stats.resize(u.modules[0].size());
18    for (size_t m = 0; m < u.modules[0].size(); m++)
19        for (size_t t = 0; t < u.modules.size(); t++)
20            u.modules_stats[m].push_back(u.modules[t][m]);
21 }

```


In Listing 4.17, 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 4.16).

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 4.15 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.

LICENSE**MIT License**

Copyright (c) 2017-2024 aff3ct

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

BIBLIOGRAPHY

- [Sim-Ari09] E. Arikan. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory (TIT)*, 55(7):3051–3073, July 2009. doi:10.1109/TIT.2009.2021379.
- [Sim-BBGL17] M. Benammar, V. Bioglio, F. Gabry, and I. Land. Multi-kernel polar codes: proof of polarization and error exponents. In *Information Theory Workshop (ITW)*, 101–105. IEEE, November 2017. doi:10.1109/ITW.2017.8277949.
- [Sim-BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo-codes. In *International Conference on Communications (ICC)*, volume 2, 1064–1070 vol.2. IEEE, May 1993. doi:10.1109/ICC.1993.397441.
- [Sim-BRC60] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Springer Information and Control*, 3(1):68 – 79, 1960. doi:10.1016/S0019-9958(60)90287-4.
- [Sim-DHJM98] D. Divsalar, H. Jin, and R. J. McEliece. Coding theorems for "turbo-like" codes. In *Allerton Conference on Communication, Control and Computing*, 201–210. September 1998. URL: <https://pdfs.semanticscholar.org/b5cc/c94d4f9ea6df991190f17359ddd7ac47f005.pdf>.
- [Sim-Gal63] R. G. Gallager. Low-density parity-check codes. 1963. URL: <http://web.mit.edu/gallager/www/pages/ldpc.pdf>.
- [Sim-MN95] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In *IMA International Conference on Cryptography and Coding (IMA-CCC)*, 100–111. UK, December 1995. Springer. doi:10.1007/3-540-60693-9_13.
- [Sim-RS60] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960. doi:10.1137/0108018.
- [Sim-RL09] W. Ryan and S. Lin. *Channel codes: classical and modern*. Cambridge University Press, September 2009. ISBN 978-0-511-64182-4. URL: <http://www.cambridge.org/9780521848688>.
- [Src-MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. doi:10.1145/272991.272995.
- [Crc-LST12] B. Li, H. Shen, and D. Tse. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Communications Letters (COMML)*, 16(12):2044–2047, December 2012. doi:10.1109/LCOMM.2012.111612.121898.
- [Crc-TLLeGal+16] T. Tonnellier, C. Leroux, B. Le Gal, B. Gadat, C. Jégo, and N. Van Wambeke. Lowering the error floor of turbo codes with CRC verification. *IEEE Wireless Communications Letters (WCL)*, 5(4):404–407, August 2016. doi:10.1109/LWC.2016.2571283.
- [Cmn-Cha72] D. Chase. Class of algorithms for decoding block codes with channel measurement information. *IEEE Transactions on Information Theory (TIT)*, 18(1):170–182, January 1972. doi:10.1109/TIT.1972.1054746.

- [Bch-Ber15] E. R. Berlekamp. *Algebraic Coding Theory (Revised Edition)*. World Scientific, May 2015. First edition from 1968. doi:10.1142/9407.
- [Bch-Chi64] R. Chien. Cyclic decoding procedures for bose- chaudhuri-hocquenghem codes. *IEEE Transactions on Information Theory (TIT)*, 10(1):357–363, October 1964. doi:10.1109/TIT.1964.1053699.
- [Bch-Mas69] J. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory (TIT)*, 15(1):122–127, January 1969. doi:10.1109/TIT.1969.1054260.
- [Ldpc-CF02] J. Chen and M. P. C. Fossorier. Density evolution for two improved bp-based decoding algorithms of ldpc codes. *IEEE Communications Letters (COMML)*, 6(5):208–210, May 2002. doi:10.1109/4234.1001666.
- [Ldpc-DDB14] M. Fossorier D. Declercq and E. Biglieri. *Channel Coding: Theory, Algorithms, and Applications*. Academic Press, 2014. ISBN 978-0-12-396499-1. doi:10.1016/C2011-0-07211-3.
- [Ldpc-FMI99] M. P. C. Fossorier, M. Mihaljevic, and H. Imai. Reduced complexity iterative decoding of low-density parity check codes based on belief propagation. *IEEE Transactions on Communications (TCOM)*, 47(5):673–680, May 1999. doi:10.1109/26.768759.
- [Ldpc-LGK+19] K. Le, F. Ghaffari, L. Kessal, D. Declercq, E. Boutillon, C. Winstead, and B. Vasić. A probabilistic parallel bit-flipping decoder for low-density parity-check codes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(1):403–416, Jan 2019. doi:10.1109/TCSI.2018.2849679.
- [Ldpc-MN95] D. J. C. MacKay and R. M. Neal. Good codes based on very sparse matrices. In *IMA International Conference on Cryptography and Coding (IMA-CCC)*, 100–111. UK, December 1995. Springer. doi:10.1007/3-540-60693-9_13.
- [Ldpc-RL09] W. Ryan and S. Lin. *Channel codes: classical and modern*. Cambridge University Press, September 2009. ISBN 978-0-511-64182-4. URL: <http://www.cambridge.org/9780521848688>.
- [Ldpc-Spi01] M.G. Luby ; M. Mitzenmacher ; M.A. Shokrollahi ; D.A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory (TIT)*, 47(2):569 – 584, February 2001. doi:10.1109/18.910575.
- [Ldpc-WNY+10] T. Wadayama, K. Nakamura, M. Yagita, Y. Funahashi, S. Usami, and I. Takumi. Gradient descent bit flipping algorithms for decoding ldpc codes. *IEEE Transactions on Communications (TCOM)*, 58(6):1610–1614, June 2010. doi:10.1109/TCOMM.2010.06.090046.
- [Ldpc-YPNA01] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam. High throughput low-density parity-check decoder architectures. In *Global Communications Conference (GLOBECOM)*, volume 5, 3019–3024 vol.5. IEEE, 2001. doi:10.1109/GLOCOM.2001.965981.
- [Ldpc-ZF02] J. Zhang and M. Fossorier. Shuffled belief propagation decoding. In *Asilomar Conference on Signals, Systems, and Computers (ACSSC)*, volume 1, 8–15 vol.1. IEEE, November 2002. doi:10.1109/ACSSC.2002.1197141.
- [Plre-TV13] I. Tal and A. Vardy. How to construct polar codes. *IEEE Transactions on Information Theory (TIT)*, 59(10):6562–6582, October 2013. doi:10.1109/TIT.2013.2272694.
- [Plre-Tri12] P. Trifonov. Efficient design and decoding of polar codes. *IEEE Transactions on Communications (TCOM)*, 60(11):3221–3227, November 2012. doi:10.1109/TCOMM.2012.081512.110872.
- [Plre-3GPP] 3GPP. TS 38.212, Multiplexing and Channel Coding (Release 15). URL: http://www.3gpp.org/ftp/Specs/archive/38_series/38.212/.
- [Plrd-ABSB14] O. Afisiadis, A. Balatsoukas-Stimming, and A. Burg. A low-complexity improved successive cancellation decoder for polar codes. In *Asilomar Conference on Signals, Systems, and Computers (ACSSC)*, 2116–2120. IEEE, November 2014. doi:10.1109/ACSSC.2014.7094848.
- [Plrd-Ari09] E. Arikan. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory (TIT)*, 55(7):3051–3073, July 2009. doi:10.1109/TIT.2009.2021379.

- [Plrd-CAL+16] A. Cassagne, O. Aumage, C. Leroux, D. Barthou, and B. Le Gal. Energy consumption analysis of software polar decoders on low power processors. In *European Signal Processing Conference (EUSIPCO)*, 642–646. IEEE, August 2016. doi:10.1109/EUSIPCO.2016.7760327.
- [Plrd-CLeGal+15] A. Cassagne, B. Le Gal, C. Leroux, O. Aumage, and D. Barthou. An efficient, portable and generic library for successive cancellation decoding of polar codes. In *International Workshop on Languages and Compilers for Parallel Computing (LCPC)*. Springer, September 2015. doi:10.1007/978-3-319-29778-1_19.
- [Plrd-FB14] U. U. Fayyaz and J. R. Barry. Low-complexity soft-output decoding of polar codes. *IEEE Journal on Selected Areas in Communications (JSAC)*, 32(5):958–966, May 2014. doi:10.1109/JSAC.2014.140515.
- [Plrd-LST12] B. Li, H. Shen, and D. Tse. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Communications Letters (COMML)*, 16(12):2044–2047, December 2012. doi:10.1109/LCOMM.2012.111612.121898.
- [Plrd-LeonardonCL+17] M. Léonardon, A. Cassagne, C. Leroux, C. Jégo, L.-P. Hamelin, and Y. Savaria. Fast and flexible software polar list decoders. *CoRR*, 2017. URL: <http://arxiv.org/abs/1710.08314>, arXiv:1710.08314.
- [Plrd-SGV+14] G. Sarkis, P. Giard, A. Vardy, C. Thibault, and W. J. Gross. Fast polar decoders: algorithm and implementation. *IEEE Journal on Selected Areas in Communications (JSAC)*, 32(5):946–957, May 2014. doi:10.1109/JSAC.2014.140514.
- [Plrd-TV11] I. Tal and A. Vardy. List decoding of polar codes. In *International Symposium on Information Theory (ISIT)*, 1–5. IEEE, July 2011. doi:10.1109/ISIT.2011.6033904.
- [Plrd-LeGalLJego15] B. Le Gal, C. Leroux, and C. Jégo. Multi-Gb/s software decoding of polar codes. *IEEE Transactions on Signal Processing (TSP)*, 63(2):349–359, January 2015. doi:10.1109/TSP.2014.2371781.
- [Plrp-Mil15] V. Miloslavskaya. Shortened polar codes. *IEEE Transactions on Information Theory (TIT)*, 61(9):4852–4865, September 2015. doi:10.1109/TIT.2015.2453312.
- [Plrp-NCL13] K. Niu, K. Chen, and J. R. Lin. Beyond turbo codes: rate-compatible punctured polar codes. In *International Conference on Communications (ICC)*, 3423–3427. IEEE, June 2013. doi:10.1109/ICC.2013.6655078.
- [Plrp-WL14] R. Wang and R. Liu. A novel puncturing scheme for polar codes. *IEEE Communications Letters (COMML)*, 18(12):2081–2084, December 2014. doi:10.1109/LCOMM.2014.2364845.
- [Pmke-Tri12] P. Trifonov. Efficient design and decoding of polar codes. *IEEE Transactions on Communications (TCOM)*, 60(11):3221–3227, November 2012. doi:10.1109/TCOMM.2012.081512.110872.
- [Pmkd-Ari09] E. Arikan. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory (TIT)*, 55(7):3051–3073, July 2009. doi:10.1109/TIT.2009.2021379.
- [Pmkd-BL18] V. Bioglio and I. Land. On the marginalization of polarizing kernels. In *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 1–5. December 2018. doi:10.1109/ISTC.2018.8625378.
- [Pmkd-GBLB17] F. Gabry, V. Bioglio, I. Land, and J. Belfiore. Multi-kernel construction of polar codes. In *International Conference on Communications (ICC)*, 761–765. IEEE, May 2017. doi:10.1109/ICC.2017.7962750.
- [Pmkd-LST12] B. Li, H. Shen, and D. Tse. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check. *IEEE Communications Letters (COMML)*, 16(12):2044–2047, December 2012. doi:10.1109/LCOMM.2012.111612.121898.
- [Pmkd-TV11] I. Tal and A. Vardy. List decoding of polar codes. In *International Symposium on Information Theory (ISIT)*, 1–5. IEEE, July 2011. doi:10.1109/ISIT.2011.6033904.

- [Rs-Ber15] E. R. Berlekamp. *Algebraic Coding Theory (Revised Edition)*. World Scientific, May 2015. First edition from 1968. doi:10.1142/9407.
- [Rs-Chi64] R. Chien. Cyclic decoding procedures for bose- chaudhuri-hocquenghem codes. *IEEE Transactions on Information Theory (TIT)*, 10(4):357–363, October 1964. doi:10.1109/TIT.1964.1053699.
- [Rs-ISR60] G. Solomon I. S. Reed. Polynomial codes over certain finite fields. *Society for Industrial and Applied Mathematics*, 8(2):300–304, June 1960. doi:10.1137/0108018.
- [Rs-Mas69] J. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory (TIT)*, 15(1):122–127, January 1969. doi:10.1109/TIT.1969.1054260.
- [Rsc-BCJR74] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Transactions on Information Theory (TIT)*, 20(2):284–287, March 1974. doi:10.1109/TIT.1974.1055186.
- [Rsc-CTL+16] A. Cassagne, T. Tonnellier, C. Leroux, B. Le Gal, O. Aumage, and D. Barthou. Beyond Gbps turbo decoder on multi-core CPUs. In *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 136–140. IEEE, September 2016. doi:10.1109/ISTC.2016.7593092.
- [Rsc-WWY+13] M. Wu, G. Wang, B. Yin, C. Studer, and J. R. Cavallaro. HSPA+/LTE-A turbo decoder on GPU and multicore CPU. In *Asilomar Conference on Signals, Systems, and Computers (ACSSC)*, 824–828. IEEE, November 2013. doi:10.1109/ACSSC.2013.6810402.
- [Rsdb-BCJR74] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate (corresp.). *IEEE Transactions on Information Theory (TIT)*, 20(2):284–287, March 1974. doi:10.1109/TIT.1974.1055186.
- [Trb-CTL+16] A. Cassagne, T. Tonnellier, C. Leroux, B. Le Gal, O. Aumage, and D. Barthou. Beyond Gbps turbo decoder on multi-core CPUs. In *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 136–140. IEEE, September 2016. doi:10.1109/ISTC.2016.7593092.
- [Trb-Ton17] T. Tonnellier. *Contribution to the Improvement of the Decoding Performance of Turbo Codes : Algorithms and Architecture*. PhD thesis, Université de Bordeaux, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01580476>.
- [Trb-TLLeGal+16] T. Tonnellier, C. Leroux, B. Le Gal, B. Gadat, C. Jégo, and N. Van Wambeke. Lowering the error floor of turbo codes with CRC verification. *IEEE Wireless Communications Letters (WCL)*, 5(4):404–407, August 2016. doi:10.1109/LWC.2016.2571283.
- [Trb-VF00] J. Vogt and A. Finger. Improving the max-log-MAP turbo decoder. *IET Electronics Letters*, 36(23):1937–1939, November 2000. doi:10.1049/el:20001357.
- [Trdb-TLLeGal+16] T. Tonnellier, C. Leroux, B. Le Gal, C. Jégo, B. Gadat, and N. Van Wambeke. Lowering the error floor of double-binary turbo codes: the flip and check algorithm. In *International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, 156–160. IEEE, September 2016. doi:10.1109/ISTC.2016.7593096.
- [Trdb-VF00] J. Vogt and A. Finger. Improving the max-log-MAP turbo decoder. *IET Electronics Letters*, 36(23):1937–1939, November 2000. doi:10.1049/el:20001357.
- [Trbp-Pyn98] R.M. Pyndiah. Near-optimum decoding of product codes: block turbo codes. *IEEE Transactions on Communications (TCOM)*, 46(8):1003–1010, August 1998. doi:10.1109/26.705396.
- [Itl-CLGH99] S. Crozier, J. Lodge, P. Guinand, and A. Hunt. Performance of turbo codes with relative prime and golden interleaving strategies. In *International Mobile Satellite Conference (IMSC)*, 268–275. 1999. URL: <https://www.tib.eu/en/search/id/BLCP%3ACN033129464/Performance-of-Turbo-Codes-with-Relative-Prime/>.
- [Itl-MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. doi:10.1145/272991.272995.

- [Mdm-ARS81] T. Aulin, N. Rydbeck, and C. -. Sundberg. Continuous phase modulation - part ii: partial response signaling. *IEEE Transactions on Communications (TCOM)*, 29(3):210–225, March 1981. doi:[10.1109/TCOM.1981.1094985](https://doi.org/10.1109/TCOM.1981.1094985).
- [Mdm-AS81] T. Aulin and C. Sundberg. Continuous phase modulation - part i: full response signaling. *IEEE Transactions on Communications (TCOM)*, 29(3):196–209, March 1981. doi:[10.1109/TCOM.1981.1095001](https://doi.org/10.1109/TCOM.1981.1095001).
- [Mdm-NB13] H. Nikopour and H. Baligh. Sparse Code Multiple Access. In *International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, volume, 332–336. Sept 2013. doi:[10.1109/PIMRC.2013.6666156](https://doi.org/10.1109/PIMRC.2013.6666156).
- [Scma-CWC15] M. Cheng, Y. Wu, and Y. Chen. Capacity analysis for non-orthogonal overloading transmissions under constellation constraints. In *International Conference on Wireless Communications Signal Processing (WCSP)*, 1–5. Oct 2015. doi:[10.1109/WCSP.2015.7341294](https://doi.org/10.1109/WCSP.2015.7341294).
- [Scma-KS16] V. P. Klimentyev and A. B. Sergienko. Detection of scma signal with channel estimation error. In *Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, 106–112. April 2016. doi:[10.1109/FRUCT-ISPIT.2016.7561515](https://doi.org/10.1109/FRUCT-ISPIT.2016.7561515).
- [Scma-KS17] V. P. Klimentyev and A. B. Sergienko. Scma codebooks optimization based on genetic algorithm. In *European Wireless Conference*, 1–6. May 2017. doi:.
- [Scma-Pro] Altera University Program. The 1st 5g algorithm innovation competition-scma.
- [Scma-SWC17] G. Song, X. Wang, and J. Cheng. Signature design of sparsely spread code division multiple access based on superposed constellation distance analysis. *IEEE Access*, 5:23809–23821, 2017. doi:[10.1109/ACCESS.2017.2765346](https://doi.org/10.1109/ACCESS.2017.2765346).
- [Scma-WZC15] Y. Wu, S. Zhang, and Y. Chen. Iterative multiuser receiver in sparse code multiple access systems. In *IEEE International Conference on Communications (ICC)*, 2918–2923. June 2015. doi:[10.1109/ICC.2015.7248770](https://doi.org/10.1109/ICC.2015.7248770).
- [Scma-ZXX+16] S. Zhang, K. Xiao, B. Xiao, Z. Chen, B. Xia, D. Chen, and S. Ma. A capacity-based codebook design method for sparse code multiple access systems. In *International Conference on Wireless Communications Signal Processing (WCSP)*, 1–5. Oct 2016. doi:[10.1109/WCSP.2016.7752620](https://doi.org/10.1109/WCSP.2016.7752620).
- [Chn-BM+58] G. E. P. Box, M. E. Muller, and others. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, 29(2):610–611, 1958. doi:[10.1214/aoms/1177706645](https://doi.org/10.1214/aoms/1177706645).
- [Chn-MT00] G. Marsaglia and W. W. Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000. doi:[10.18637/jss.v005.i08](https://doi.org/10.18637/jss.v005.i08).
- [Chn-MN98] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. doi:[10.1145/272991.272995](https://doi.org/10.1145/272991.272995).