SORBONNE UNIVERSITÉ

LIP6 Laboratory

CORIOLIS

User's Guide

Jean-Paul CHAPUT
Jean-Paul.Chaput@lip6.fr

Contents

Credits & License

HURRICANE	Rémy ESCASSUT & Christian MASSON
ETESIAN	Gabriel GOUVINE
STRATUS	Sophie BELLOEIL
KATANA (global)	Damien DUPUIS
KATANA (detailed), UNICORN	Jean-Paul CHAPUT

Others important contributors to CORIOLIS are Christophe ALEXANDRE, Roselyne CHOTIN, Hugo CLEMENT, Marek SROKA and Wu YIFEI.

The KATANA router makes use of the FLUTE software, which is copyright© Chris C. N. CHU from the Iowa State University (`http://home.eng.iastate.edu/~cnchu/`).

Release Notes

Release 1.0.1475

This is the first preliminary release of the CORIOLIS 2 framework.

This release mainly ships the global router KNIK and the detailed router KITE. Together they aim to replace the ALLIANCE NERO router. Unlike NERO, KITE is based on an innovating routing modeling and ad-hoc algorithm. Although it is released under GPL license, the source code will be avalaible later.

Contents of this release:

1. A graphical user interface (viewer only).
2. The KNIK global router.
3. The KITE detailed router.

Supported input/output formats:

- ALLIANCE **vst** (netlist) & **ap** (physical) formats.
- Even if there are some references to the CADENCE LEFDEF format, its support is not included because it depends on a library only available to SI2 affiliated members.

Release 1.0.1963

Release 1963 is alpha. All the tools from CORIOLIS 1 have been ported into this release.

Contents of this release:

1. The STRATUS netlist capture language (GENLIB replacement).
2. The MAUKA placer (still contains bugs).
3. A graphical user interface (viewer only).
4. The KNIK global router.
5. The KITE detailed router.
6. Partially implemented python support for configuration files (alternative to XML).
7. A documentation (imcomplete/obsoleted in HURRICANE's case).

Release 1.0.2049

Release *2049* is Alpha.
Changes of this release:

1. The Hurricane documentation is now accurate. Documentation for the Cell viewer and CRLcore has been added.
2. More extensive Python support for all the components of Coriolis.
3. Configuration is now completly migrated under Python. XML loaders can still be useds for compatibilty.
4. The **`cgt`** main has been rewritten in Python.

Release v2.0.1

1. Migrated the repository from **`svn`** to **`git`**, and release complete sources. As a consequence, we drop the distribution packaging support and give public read-only access to the repository.
2. Deep rewrite of the Katabatic database and Kite detailed router, achieve a speedup factor greater than 20...

Release v2.1

1. Replace the old simulated annealing placer Mauka by the analytical placer Etesian and its legalization and detailed placement tools.
2. Added a Blif format parser to process circuits generated by the Yosys and ABC logic synthetizers.
3. The multiple user defined configuration files are now grouped under a common hidden (dot) directory `.coriolis2` and the file extension is back from `.conf` to `.py`.

Release v2.2

1. Added JSON import/export of the whole Hurricane DataBase. Two save mode are supported: *Cell* mode (standalone) or *Blob* mode, which dump the whole design down and including the standard cells.

Release v2.3

1. Revert to a more standard organisation of the branchs. **devel_anabatic** is closed and we go on with **master** (stable version) and **devel**.
2. Make Katana the default global & detailed router. Put Knik & Kite in the obsolete menues.
3. Finally make uses of PyQt4 widgets. Seems to integrate without problems with the Coriolis own qt widget. The drawback is that to build against qt 5 needs to adjustement from the user.
4. Improved support for whole chip management. The outer part of the chip containing the pad is decoupled from the core. This allow to cleanly separate real pads from the foundry from a symbolic core. But this does not preclude other combinations as fully symbolic or fully real.

   To perform the separation an intermediate hierarchical level `corona` between chip and core has been introduced.

Complete Design Flow & Examples

While Coriolis can be used stand-alone, it is in fact part of a more complete design flow build upon Yosys and Alliance. In addition, a set of demos and examples are supplied in the repository `alliance-check-toolkit`.

- Yosys : http://www.clifford.at/yosys/

  An **rpm** packaged version is available here:

  https://ftp.lip6.fr/pub/linux/distributions/slsoc/soc/7/addons/x86_64/repoview/yosys.html
- Alliance : https://www-soc.lip6.fr/equipe-cian/logiciels/alliance/
- `alliance-check-toolkit` **git** repository:

  https://www-soc.lip6.fr/git/alliance-check-toolkit.git/

Installation

Note

As the sources are being released, the binary packaging is dropped. You may still find (very) old versions here: http://asim.lip6.fr/pub/coriolis/2.0 .

In a nutshell, building source consistis in pulling the **git** repository then running the **ccb** installer.

Note

The documentation is already generated and commited in the **git** tree. You may not install the additional prerequisites for the documentation. By default the documentation is not generated, just installed by **ccb**. If you really want to re-generate it, add the `--doc` flag to **ccb**.

Main building prerequisites:

- cmake
- C++11-capable compiler
- BFD library (provided through `binutils`).
- RapidJSON
- python2.7
- boost
- libxml2
- bzip2
- yacc & lex
- Qt 4 or Qt 5
- PyQt 4 or PyQt 5
- Qwt

Building documentation prerequisites:

- doxygen

- latex
- python-docutils (for reStructuredText)

The following libraries gets directly bundled with Coriolis:

- LEF/DEF (from SI2)
- FLUTE (from Chris C. N. Chu)

For other distributions, refer to their own packaging system.

Fixed Directory Tree

In order to simplify the work of the **ccb** installer, the source, build and installation tree is fixed. To successfully compile Coriolis you must follow it exactly. The tree is relative to the home directory of the user building it (noted `~/` or `$HOME/`). Only the source directory needs to be manually created by the user, all others will be automatically created either by **ccb** or the build system.

Sources	
Sources root	~/coriolis-2.x/src
under git	~/coriolis-2.x/src/coriolis
Architecture Dependant Build	
Linux, SL 7, 64b	~/coriolis-2.x/Linux.el7_64/Release.Shared/build/<tool>
Linux, SL 6, 32b	~/coriolis-2.x/Linux.slsoc6x/Release.Shared/build/<tool>
Linux, SL 6, 64b	~/coriolis-2.x/Linux.slsoc6x_64/Release.Shared/build/<tool>
Linux, Fedora, 64b	~/coriolis-2.x/Linux.fc_64/Release.Shared/build/<tool>
Linux, Fedora, 32b	~/coriolis-2.x/Linux.fc/Release.Shared/build/<tool>
FreeBSD 8, 32b	~/coriolis-2.x/FreeBSD.8x.i386/Release.Shared/build/<tool>
FreeBSD 8, 64b	~/coriolis-2.x/FreeBSD.8x.amd64/Release.Shared/build/<tool>
Windows 7, 32b	~/coriolis-2.x/Cygwin.W7/Release.Shared/build/<tool>
Windows 7, 64b	~/coriolis-2.x/Cygwin.W7_64/Release.Shared/build/<tool>
Windows 8.x, 32b	~/coriolis-2.x/Cygwin.W8/Release.Shared/build/<tool>
Windows 8.x, 64b	~/coriolis-2.x/Cygwin.W8_64/Release.Shared/build/<tool>
Architecture Dependant Install	
Linux, SL 6, 32b	~/coriolis-2.x/Linux.slsoc6x/Release.Shared/install/
FHS Compliant Structure under Install	
Binaries	.../install/bin
Libraries (Python)	.../install/lib
Include by tool	.../install/include/coriolis2/<project>/<tool>
Configuration files	.../install/etc/coriolis2/
Doc, by tool	.../install/share/doc/coriolis2/en/html/<tool>

Note

Alternate build types: the `Release.Shared` means an optimized build with shared libraries. But there are also available `Static` instead of `Shared` and `Debug` instead of `Release` and any combination of them.
`Static` do not work because I don't know yet to mix statically linked binaries and Python modules (which must be dynamic).

Building Coriolis

The actively developed branch

The **devel_anabatic** branch is now closed and we go back to a more classical scheme where **master** is the stable version and **devel** the development one.

The CORIOLIS **`git`** repository is https://www-soc.lip6.fr/git/coriolis.git

Note

Again, the **devel_anabatic** branch is now closed. Please revert to **devel** or **master**.

Note

As it is now possible to mix PYQT widget with CORIOLIS ones, it is simpler for us to revert to QT 4 only. Our reference OS being RHEL 7, there is no compatible PYQT5 build compatible with their QT 5 version (we fall short of one minor, they provides QT 5.9 were we need at least QT 5.10).

Note

Under RHEL 7 or clones, they upgraded their version of QT 4 (from 4.6 to 4.8) so the *diagonal line* bug no longer occur. So we can safely use the default system QT again.

Installing on REDHAT or compatible distributions

1. Install or check that the required prerequisites are installeds :

   ```
   dummy@lepka:~> yum install -y git cmake bison flex gcc-c++ libstdc++-devel   \
                                 binutils-devel                                 \
                                 boost-devel boost-python boost-filesystem      \
                                 boost-regex  boost-wave                        \
                                 python-devel libxml2-devel bzip2-devel         \
                                 qt-devel qwt-devel                             #
   ```

   Note, that the `Qwt` packages are directly availables from the standart distribution when using QT 4.

2. Install the unpackaged prerequisites. Currently, only RapidJSON.

   ```
   dummy@lepka:~> mkdir -p ~/coriolis-2.x/src/support
   dummy@lepka:support> cd ~/coriolis-2.x/src/support
   dummy@lepka:support> git clone http://github.com/miloyip/rapidjson
   ```

3. Create the source directory and pull the **`git`** repository:

   ```
   dummy@lepka:~> mkdir -p ~/coriolis-2.x/src
   dummy@lepka:src> cd ~/coriolis-2.x/src
   dummy@lepka:src> git clone https://www-soc.lip6.fr/git/coriolis.git
   ```

4. Build & install:

```
dummy@lepka:src> cd coriolis
dummy@lepka:coriolis> git checkout devel_anabatic
dummy@lepka:coriolis> ./bootstrap/ccb.py --project=support  \
                                         --project=coriolis \
                                         --make="-j4 install"
```

Note

Pre-generated documentation will get installed by the previous command. Only if you did made modifications to it you need to regenerate it with:

```
dummy@lepka:coriolis> ./bootstrap/ccb.py --project=support  \
                                         --project=coriolis \
                                         --doc --make="-j1 install"
```

We need to perform a separate installation of the documentation because it does not support to be generated with a parallel build. So we compile & install in a first stage in `-j4` (or whatever) then we generate the documentation in `-j1`

Under RHEL6 or clones, you must build using the **`devtoolset`**, the version is to be given as argument:

```
dummy@lepka:coriolis> ./bootstrap/ccb.py --project=coriolis \
                                         --devtoolset=8 --make="-j4 install"
```

If you want to uses Qt 5 instead of Qt 4 modify the previous steps as follow:

- At **step 1**, do not install the QT 4 related development package (`qt4-devel`), but instead:

  ```
  dummy@lepka:~> yum install -y qt5-qtbase-devel qt5-qtsvg-devel             #
  ```

  The package `qwt-qt5-devel` and it's dependency `qwt-qt5` are not provided by any standard repository (like EPEL). You may download them from the LIP6 Addons Repository Then run:

  ```
  dummy@lepka:~> yum localinstall -y qwt-qt5-6.1.2-4.fc23.x86_64.rpm  \
                                     qwt-qt5-6.1.2-4.fc23.x86_64.rpm  # Qwt for
  ```

- At **step 4**, add a `--qt5` argument to the `ccb.py` command line.

- The PYTHON scripts that makes uses of PYQT in `crlcore` and `cumulus` must be edited to import `PyQt5` instead of `PtQt4` (should find a way to automatically switch between the two of them).

The complete list of **`ccb`** functionalities can be accessed with the `--help` argument. It also may be run in graphical mode (`--gui`).

Building a Debug Enabled Version

The `Release.Shared` default version of the CORIOLIS is build stripped of symbols and optimized so that it makes analysing a core dump after a crash difficult. In the (unlikely) case of a crash, you may want to build, alongside the optimized version, a debug one which allow forensic examination by **`gdb`** (or **`valgrind`** or whatever).

Run again `ccb.py`, adding the `--debug` argument:

```
dummy@lepka:coriolis> ./bootstrap/ccb.py --project=support  \
                                         --project=coriolis \
                                         --make="-j4 install" --debug
```

As **`cgt`** is a PYTHON script, the right command to run **`gdb`** is:

```
dummy@lepka:work> gdb python core.XXXX
```

Installing on Debian 9, Ubuntu 18 or compatible distributions

First, install or check that the required prerequisites are installeds :

```
dummy@lepka:~> sudo apt install -y build-essential binutils-dev
                                   git cmake bison flex gcc python-dev
                                   libboost-all-dev libboost-python-dev
                                   libbz2-dev libxml2-dev rapidjson-dev libbz2-de
                                   qt4-dev-tools libqwt5-qt4-dev
                                   qtbase5-dev libqt5svg5-dev libqwt-qt5-dev
                                   doxygen dvipng graphviz python-sphinx
                                   texlive-fonts-extra texlive-lang-french
```

Second step is to create the source directory and pull the **git** repository:

```
dummy@lepka:~> mkdir -p ~/coriolis-2.x/src
dummy@lepka:src> cd ~/coriolis-2.x/src
dummy@lepka:src> git clone https://www-soc.lip6.fr/git/coriolis.git
```

Third and final step, build & install:

```
dummy@lepka:src> cd coriolis
dummy@lepka:coriolis> git checkout devel_anabatic
dummy@lepka:coriolis> ./bootstrap/ccb.py --project=coriolis \
                                         --make="-j4 install"
```

Additionnal Requirement under MacOS

Coriolis make uses of the **boost::python** module, but the MacPorts **boost** seems unable to work with the Python bundled with MacOS. So you have to install both of them from MacPorts:

```
dummy@macos:~> port install boost +python27
dummy@macos:~> port select python python27
dummy@macos:-> export DYLD_FRAMEWORK_PATH=/opt/local/Library/Frameworks
```

The last two lines tell MacOS to use the Python from MacPorts and *not* from the system.
Then proceed with the generic install instructions.

Packaging Coriolis

Packager should not use **ccb**, instead `bootstrap/Makefile.package` is provided to emulate a top-level `autotool` makefile. Just copy it in the root of the Coriolis git repository (`~/corriolis-2.x/src/coriolis/`) and build.

Sligthly outdated packaging configuration files can also be found under `bootstrap/`:

- `bootstrap/coriolis2.spec.in` for **rpm** based distributions.
- `bootstrap/debian` for Debian based distributions.

Hooking up into Alliance

Coriolis relies on Alliance for the cell libraries. So after installing or packaging, you must configure it so that it can found those libraries.

The easiest way is to setup the Alliance environment (i.e. sourcing `.../etc/profile.d/alc_env.{sh,csh}` **before** setting up Coriolis environment (see the next section). To understand how Coriolis find/setup Alliance you may have look to the Alliance Helper.

Setting up the Environment (coriolisEnv.py)

To simplify the tedious task of configuring your environment, a helper is provided in the `bootstrap` source directory (also installed in the directory `.../install/etc/coriolis2/`):

```
~/coriolis-2.x/src/coriolis/bootstrap/coriolisEnv.py
```

Use it like this:

```
dummy@lepka:~> eval `~/coriolis-2.x/src/coriolis/bootstrap/coriolisEnv.py`
```

Note

Do not call that script in your environement initialisation. When used under RHEL6 or clones, it needs to be run in the **`devtoolset`** environement. The script then launch a new shell, which may cause an infinite loop if it's called again in, say **`~/.bashrc`**.
Instead you may want to create an alias:

```
alias c2r='eval "`~/coriolis-2.x/src/coriolis/bootstrap/coriolisEnv.py`"'
```

Coriolis Configuration & Initialisation

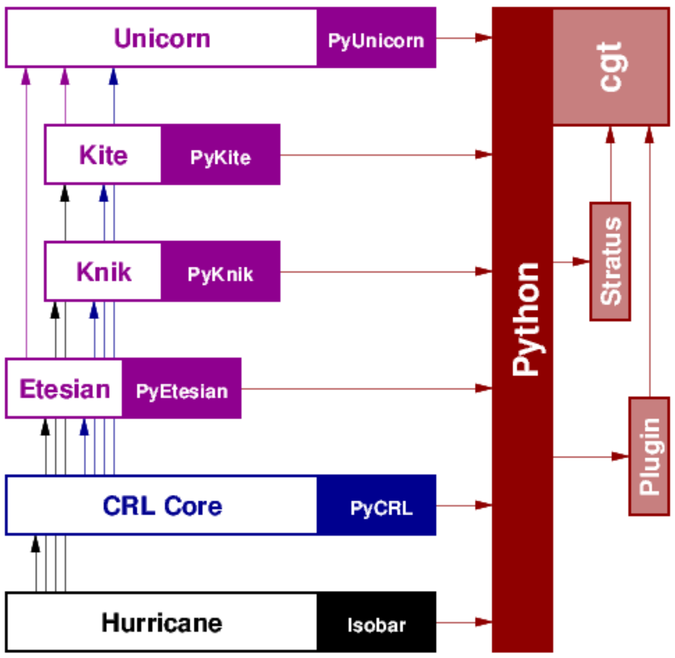
General Software Architecture

CORIOLIS has been built with respect of the classical paradigm that the computational instensive parts have been written in C++, and almost everything else in PYTHON. To build the PYTHON interface we used two methods:

- For self-contained modules **`boost::python`** (mainly in **`vlsisapd`**).
- For all modules based on HURRICANE, we created our own wrappers due to very specific requirements such as shared functions between modules or C++/PYTHON secure bi-directional object deletion.

Note

Python Documentation: Most of the documentation is related to the C++ API and implemetation of the tools. However, the PYTHON bindings have been created so they mimic *as closely as possible* the C++ interface, so the documentation applies to both languages with only minor syntactic changes.



All configuration & initialization files are Python scripts, despite their `.conf` extention. From a syntactic point of view, there is no difference between the system-wide configuration files and the user's configuration, they use the same Python helpers.

Configuration is done in two stages:

1. Selecting the technology.
2. Loading the complete configuration for the given technology and the user's settings.

First Stage: Technology Selection

The initialization process is done by executing, in order, the following file(s):

Order	Meaning	File
1	The system setting	`/etc/coriolis2/techno.conf`
2	The user's global setting	`${HOME}/.coriolis2/techno.py`
3	The user's local setting	`<CWD>/.coriolis2/techno.py`

Those files must provide only one variable, the name of the technology. Each technology will provide configuration for both the symbolic part and the real part. CORIOLIS can work with purely symbolic technology (`symbolic/cmos`) in that case, the real technology part is a dummy one.

For example, to use MOSIS 180nm:

```
# -*- Mode:Python -*-

technology = '180/scn6m_deep_09'
```

Second Stage: Technology Configuration Loading

The **`technology`** variable is set by the first stage and it's the name of the technology. A directory of that name, with all the configuration files, must exist in the configuration directory (**`/etc/coriolis2`**). In addition to the technology-specific directories, a **`common/`** directory is there to provide a trunk for all the identical data across the various technologies. The initialization process is done by executing, in order, the following file(s):

Order	Meaning	File
1	The system initialization	`/etc/coriolis2/<technology>/<TOOL>.conf`
2	The user's global initialization	`${HOME}/.coriolis2/settings.py`
3	The user's local initialization	`<CWD>/.coriolis2/settings.py`

Note

The loading policy is not hard-coded. It is implemented at Python level in **`/etc/coriolis2/coriolisInit.py`**, and thus may be easily amended to whatever site policy.
The truly mandatory requirement is the existence of **`coriolisInit.py`** which *must* contain a **`coriolisConfigure()`** function with no argument.
The **`coriolisInit.py`** script execution is triggered by the *import* of the `CRL` module:

```
import sys
import os.path
import Cfg
import Hurricane
import CRL          # Triggers execution of "coriolisInit.py".
import Viewer
```

Configuration Helpers

To ease the writing of configuration files, a set of small helpers is available. They allow to setup the configuration parameters through simple assembly of tuples. The helpers are installed under the directory:

```
<install>/etc/coriolis2/
```

Where **<install>/** is the root of the installation.

Alliance Helper

The configuration file must provide an **allianceConfig** tuple as shown below. Like all the Coriolis configuration file, it is to be executed through Python, so we can use it to perform a not so dumb search of the Alliance installation directory. Our default policy is to try to read the ALLIANCE_TOP environment variable, and if not found, default to /soc/alliance.

```
import os
from helpers.Alliance import AddMode
from helpers.Alliance import Gauge

allianceTop = None
if os.environ.has_key('ALLIANCE_TOP'):
  allianceTop = os.environ['ALLIANCE_TOP']
  if not os.path.isdir(allianceTop):
    allianceTop = None

if not allianceTop: allianceTop = '/soc/alliance'

cellsTop = allianceTop+'/cells/'


allianceConfig = \
    ( ( 'CATALOG'            , 'CATAL')
    , ( 'WORKING_LIBRARY'    , '.')
    , ( 'SYSTEM_LIBRARY'     , ( (cellsTop+'sxlib'    , AddMode.Append)
                               , (cellsTop+'dp_sxlib', AddMode.Append)
                               , (cellsTop+'ramlib'   , AddMode.Append)
                               , (cellsTop+'romlib'   , AddMode.Append)
                               , (cellsTop+'rflib'    , AddMode.Append)
                               , (cellsTop+'rf2lib'   , AddMode.Append)
                               , (cellsTop+'pxlib'    , AddMode.Append)
                               , (cellsTop+'padlib'   , AddMode.Append) ) )
    , ( 'IN_LO'              , 'vst')
    , ( 'IN_PH'              , 'ap')
    , ( 'OUT_LO'             , 'vst')
    , ( 'OUT_PH'             , 'ap')
    , ( 'POWER'              , 'vdd')
    , ( 'GROUND'             , 'vss')
    , ( 'CLOCK'              , '.*ck.*|.*nck.*')
    , ( 'BLOCKAGE'           , '^blockage[Nn]et*')
    , ( 'PAD'                , '.*_px$')
    )
```

The example above shows the system configuration file, with all the available settings. Some important remarks about those settings:

- In its configuration file, the user does not need to redefine all the settings, just the one he wants to change. In most of the cases, the `SYSTEM_LIBRARY`, the `WORKING_LIBRARY` and the special net names (at this point there is not much alternatives for the others settings).
- `SYSTEM_LIBRARY` setting: Setting up the library search path. Each library entry in the tuple will be added to the search path according to the second parameter:
  - **`AddMode::Append`**: append to the search path.
  - **`AddMode::Prepend`**: insert in head of the search path.
  - **`AddMode::Replace`**: look for a library of the same name and replace it, whithout changing the search path order. If no library of that name already exists, it is appended.

  A library is identified by its name, this name is the last component of the path name. For instance: `/soc/alliance/sxlib` will be named `sxlib`. Implementing the ALLIANCE specification, when looking for a *Cell* `name`, the system will browse sequentially through the library list and returns the first *Cell* whose name match.
- For `POWER`, `GROUND`, `CLOCK` and `BLOCKAGE` net names, a regular expression (GNU regexp) is expected.

A typical user's configuration file would be:

```
import os

homeDir = os.getenv('HOME')

allianceConfig = \
    ( ('WORKING_LIBRARY'    , homeDir+'/worklib')
    , ('SYSTEM_LIBRARY'     , ( (homeDir+'/mylib', Environment.Append) ) )
    , ('POWER'              , 'vdd.*')
    , ('GROUND'             , 'vss.*')
    )
```

Tools Configuration Helpers

All the tools use the same helper to load their configuration (a.k.a. *Configuration Helper*). Currently the following configuration system-wide configuration files are defined:

- **`misc.conf`**: common settings or not belonging specifically to a tool.
- **`etesian.conf`**: for the ETESIAN tool.
- **`kite.conf`**: for the KITE tool.
- **`stratus1.conf`**: for the STRATUS1 tool.

Here is the contents of **`etesian.conf`**:

```
# Etesian parameters.
parametersTable = \
    ( ('etesian.aspectRatio'     , TypePercentage, 100    , { 'min':10, 'max':1000
    , ('etesian.spaceMargin'     , TypePercentage, 5       )
    , ('etesian.uniformDensity' , TypeBool      , False   )
    , ('etesian.routingDriven'   , TypeBool      , False   )
    , ("etesian.effort"          , TypeEnumerate , 2
      , { 'values':( ("Fast"     , 1)
                   , ("Standard", 2)
                   , ("High"     , 3)
                   , ("Extreme" , 4) ) }
      )
    , ("etesian.graphics"        , TypeEnumerate , 2
      , { 'values':( ("Show every step"   , 1)
                   , ("Show lower bound" , 2)
                   , ("Show result only" , 3) ) }
      )
    )

layoutTable = \
    ( (TypeTab    , 'Etesian', 'etesian')

    , (TypeTitle , 'Placement area')
    , (TypeOption, "etesian.aspectRatio"    , "Aspect Ratio, X/Y (%)", 0 )
    , (TypeOption, "etesian.spaceMargin"    , "Space Margin"          , 1 )
    , (TypeRule  ,)
    , (TypeTitle , 'Etesian - Placer')
    , (TypeOption, "etesian.uniformDensity", "Uniform density"       , 0 )
    , (TypeOption, "etesian.routingDriven" , "Routing driven"        , 0 )
    , (TypeOption, "etesian.effort"         , "Placement effort"      , 1 )
    , (TypeOption, "etesian.graphics"       , "Placement view"        , 1 )
    , (TypeRule  ,)
    )
```

Taxonomy of the file:

- It must contain, at least, the two tables:
  - `parametersTable`, defines & initialises the configuration variables.
  - `layoutTables`, defines how the various parameters will be displayed in the configuration window (The Settings Tab).
- The `parametersTable`, is a tuple (list) of tuples. Each entry in the list describes a configuration parameter. In its simplest form, it's a quadruplet **`(TypeOption, 'paramId', ParameterType, DefaultValue)`** with:
  1. `TypeOption`, tells that this tuple describes a parameter.
  2. `paramId`, the identifier of the parameter. Identifiers are defined by the tools. The list of parameters is detailed in each tool section.
  3. `ParameterType`, the kind of parameter. Could be:
     - `TypeBool`, boolean.
     - `TypeInt`, signed integer.
     - `TypeEnumerate`, enumerated type, needs extra entry.
     - `TypePercentage`, percentage, expressed between 0 and 100.

   - `TypeDouble`, float.
   - `TypeString`, character string.
4. `DefaultValue`, the default value for that parameter.

Hacking the Configuration Files

Aside from the symbols that get used by the configuration helpers like **`allianceConfig`** or **`parametersTable`**, you can put pretty much anything in **`<CWD>/.coriolis2/settings.py`** (that is, written in PYTHON).

For example:

```
# -*- Mode:Python -*-

defaultStyle = 'Alliance.Classic [black]'

# Regular Coriolis configuration.
parametersTable = \
    ( ('misc.catchCore'           , TypeBool      , False  )
    , ('misc.info'                , TypeBool      , False  )
    , ('misc.paranoid'            , TypeBool      , False  )
    , ('misc.bug'                 , TypeBool      , False  )
    , ('misc.logMode'             , TypeBool      , True   )
    , ('misc.verboseLevel1'       , TypeBool      , False  )
    , ('misc.verboseLevel2'       , TypeBool      , True   )
    , ('misc.minTraceLevel'       , TypeInt       , 0      )
    , ('misc.maxTraceLevel'       , TypeInt       , 0      )
    )

# Some ordinary Python script...
import os

print '        o  Cleaning up ClockTree previous run.'
for fileName in os.listdir('.'):
  if fileName.endswith('.ap') or (fileName.find('_clocked.') >= 0):
    print '          - <%s>' % fileName
    os.unlink(fileName)
```

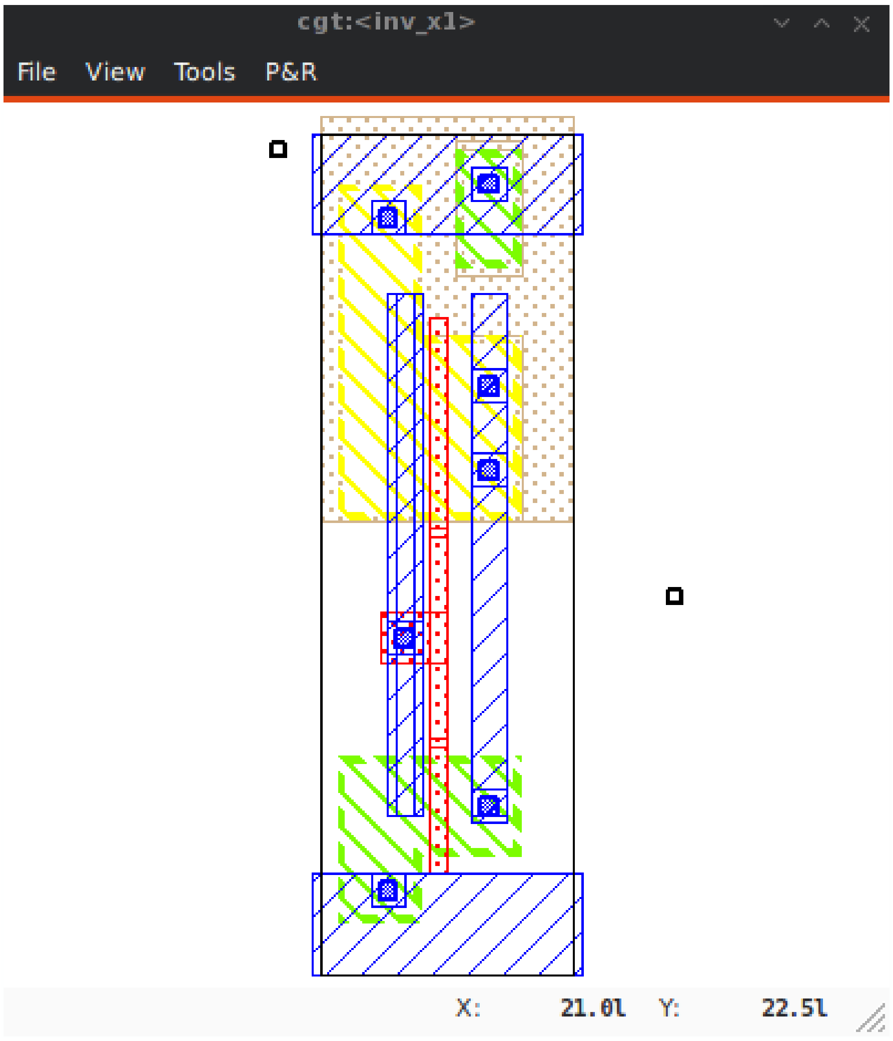
See Python Interface to Coriolis for more details those capabilities.

CGT - The Graphical Interface

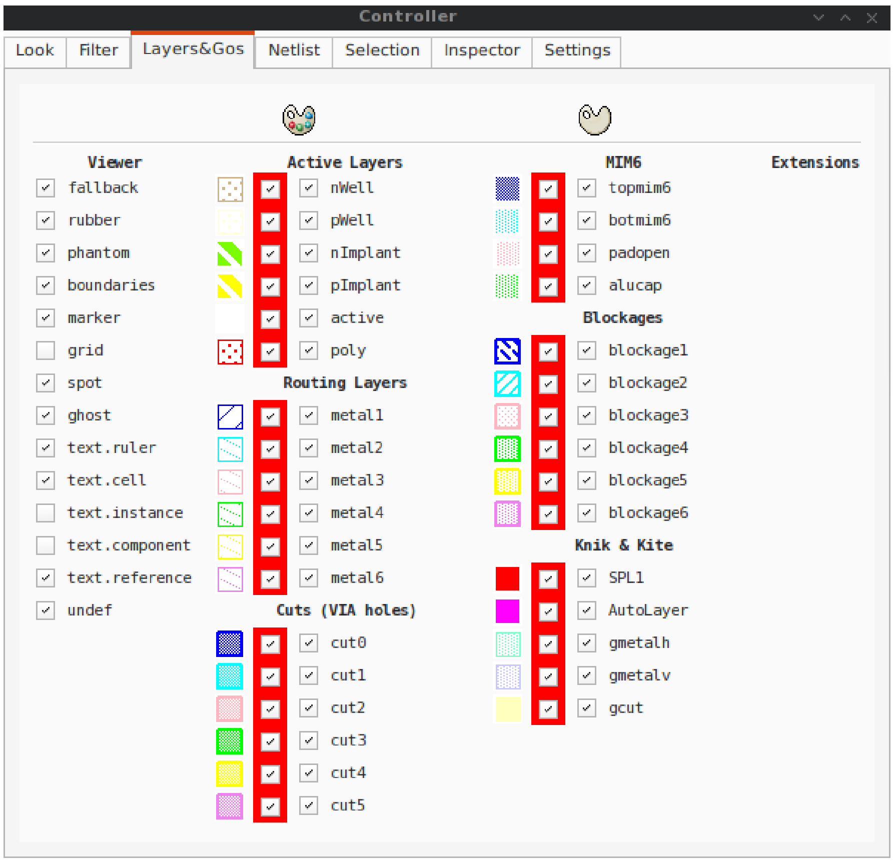
The CORIOLIS graphical interface is split up into two windows.

- The **Viewer**, with the following features:
   - Basic load/save capabilities.
   - Display the current working cell. Could be empty if the design is not yet placed.
   - Execute Stratus Scripts.
   - Menu to run the tools (placement, routage).

Features are detailed in Viewer & Tools.



- The **Controller**, which allows:
  - Tweak what is displayer by the *Viewer*. Through the *Look*, *Filter* and *Layers&Gos* tabs.
  - Browse the *netlist* with eponym tab.
  - Show the list of selected objects (if any) with *selection*
  - Walk through the Database, the Cell or the Selection with *Inspector*. This is an advanced feature, reserved for experimented users.
  - The tab *Settings* which give access to all the settings. They are closely related to Configuration & Initialisation.



Viewer
- ✓ fallback
- ✓ rubber
- ✓ phantom
- ✓ boundaries
- ✓ marker
- ☐ grid
- ✓ spot
- ✓ ghost
- ✓ text.ruler
- ✓ text.cell
- ☐ text.instance
- ☐ text.component
- ✓ text.reference
- ✓ undef

Active Layers
- ✓ nWell
- ✓ pWell
- ✓ nImplant
- ✓ pImplant
- ✓ active
- ✓ poly

Routing Layers
- ✓ metal1
- ✓ metal2
- ✓ metal3
- ✓ metal4
- ✓ metal5
- ✓ metal6

Cuts (VIA holes)
- ✓ cut0
- ✓ cut1
- ✓ cut2
- ✓ cut3
- ✓ cut4
- ✓ cut5

MIM6
- ✓ topmim6
- ✓ botmim6
- ✓ padopen
- ✓ alucap

Blockages
- ✓ blockage1
- ✓ blockage2
- ✓ blockage3
- ✓ blockage4
- ✓ blockage5
- ✓ blockage6

Knik & Kite
- ✓ SPL1
- ✓ AutoLayer
- ✓ gmetalh
- ✓ gmetalv
- ✓ gcut

Extensions

Viewer & Tools

Stratus Netlist Capture

Stratus is the replacement for GenLib procedural netlist capture language. It is designed as a set of Python classes, and comes with it's own documentation (Stratus Documentation)

The Hurricane Data-Base

The Alliance flow is based on the MBK data-base, which has one data-structure for each view. That is, **`Lofig`** for the *logical* view and **`Phfig`** for the *physical* view. The place and route tools were responsible for maintaining (or not) the coherency between views. Reflecting this weak coupling between views, each one was stored in a separate file with a specific format. The *logical* view is stored in a **`vst`** file in VHDL format and the *physical* in an **`ap`** file in an ad-hoc format.

The Coriolis flow is based on the Hurricane data-base, which has a unified structure for *logical* and *physical* view. That data structure is the *Cell* object. The *Cell* can have any state between pure netlist and completly placed and routed design. Although the memory representation of the views has deeply changed we still use the Alliance files format, but they now really represent views of the same object. The point is that one must be very careful about view coherency when going to and from Coriolis.

As for the second release, Coriolis can be used only for three purposes :

- **Placing a design**, in which case the *netlist* view must be present.
- **Routing a design**, in that case the *netlist* view and the *layout* view must be present and *layout* view must contain a placement. Both views must have the same name. When saving the routed design, it is advised to change the design name otherwise the original unrouted placement in the *layout* view will be overwritten.

- **Viewing a design**, the *netlist* view must be present, if a *layout* view is present it still must have the same name but it can be in any state.

Synthetizing and loading a design

CORIOLIS supports several file formats. It can load all file format from the ALLIANCE toolchain (.ap for layout, behavioural and structural vhdl .vbe and .vst), BLIF netlist format as well as benchmark formats from the ISPD contests.

It can be compiled with LEF/DEF support, although it requires acceptance of the SI2 license and may not be compiled in your version of the software.

Synthesis under Yosys You can create a BLIF file from the YOSYS synthetizer, which can be imported under Coriolis. Most libraries are specified as a .lib liberty file and a .lef LEF file. YOSYS opens most .lib files with minor modifications, but LEF support in Coriolis relies on SI2. If Coriolis hasn't been compiled against it, the library is given in ALLIANCE .ap format. Some free libraries already provide both .ap and .lib files.

Once you have installed a common library under YOSYS and Coriolis, just synthetize your design with YOSYS and import it (as Blif without the extension) under Coriolis to perform place&route.

Synthesis under Alliance ALLIANCE is an older toolchain but has been extensively used for years. Coriolis can import and write Alliance designs and libraries directly.

Etesian -- Placer

The ETESIAN placer is a state of the art (as of 2015) analytical placer. It is within `5%` of other placers' solutions, but is normally a bit worse than ePlace. This CORIOLIS tool is actually an encapsulation of COLOQUINTE which *is* the placer.

Note

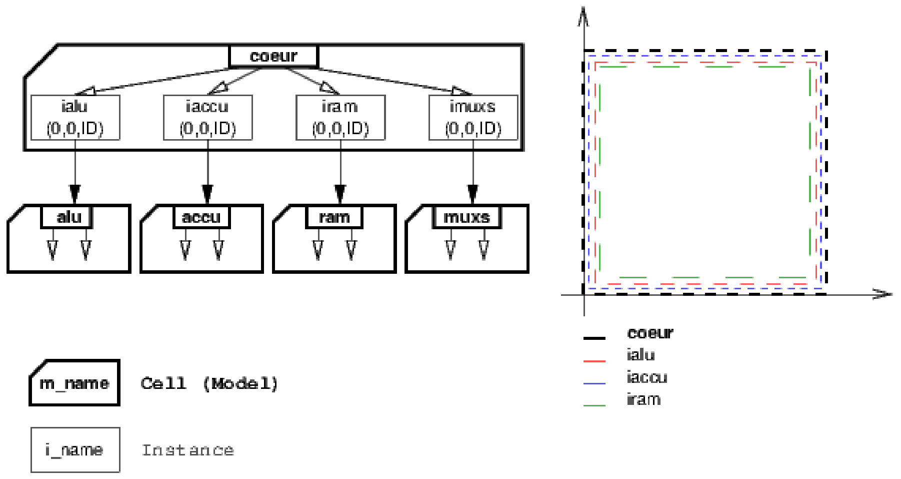
Instance Uniquification: a same logical instance cannot have two different placements. So, if you don't supply a placement for it, it will be uniquified (cloned) and you will see the copy files appears on disk upon saving.

Hierarchical Placement

The placement area is defined by the top cell abutment box.

When placing a complete hierarchy, the abutment boxes of the cells (models) other than the top cell are sets identical to the one of the top cell and their instances are all placed at position `(0,0,ID)`. That is, all the abutments boxes, whatever the hierarchical level, defines the same area (they are exactly superposed).

We choose this scheme because the placer will see all the instances as virtually flattened, so they can be placed anywhere inside the top-cell abutment box.



Computing the Placement Area

The placement area is computed using the `etesian.aspectRatio` and `etesian.spaceMargin` parameters only if the top-cell has an empty abutment box. If the top-cell abutment box has to be set, then it is propagated to all the instances models recursively.

Reseting the Placement

Once a placement has been done, the placer cannot reset it (will be implemented later). To perform a new placement, you must restart **`cgt`**. In addition, if you have saved the placement on disk, you must erase any **`.ap`** file, which are automatically reloaded along with the netlist (**`.vst`**).

Limitations

Etesian supports standard cells and fixed macros. As for the Coriolis 2.1 version, it doesn't support movable macros, and you must place every macro beforehand. Timing and routability analysis are not included either, and the returned placement may be unroutable.

Etesian Configuration Parameters

Parameter Identifier	Type	Default
Etesian Parameters		
`etesian.aspectRatio`	TypePercentage	**100**
	Define the height on width `H/W` aspect ratio, can be comprised between 10 and 1000	
`etesian.spaceMargin`	TypePercentage	**5**
	The extra white space added to the total area of the standard cells	
`etesian.uniformDensity`	TypeBool	**False**
	Whether the cells will be spread envenly across the area or allowed to form denser clusters	
`etesian.effort`	TypeInt	**2**
	Sets the balance between the speed of the placer and the solution quality	
`etesian.routingDriven`	TypeBool	**False**
	Whether the tool will try routing iterations and whitespace allocation to improve routability; to be implemented	
`etesian.graphics`	TypeInt	**2**
	How often the display will be refreshed More refreshing slows the placer. • `1` shows both upper and lower bounds • `2` only shows lower bound results • `3` only shows the final results	

Katana -- Global Router

The quality of KATANA global routing solutions are equivalent to those of FGR 1.0. For an in-depth description of KATANA algorithms, you may download the thesis of D. DUPUIS avalaible from here~: Knik Thesis (KNIK has been rewritten as part of KATANA, the algorithms remains essentially the same).

The global router is now deterministic.

Katana -- Detailed Router

KATANA no longer suffers from the limitations of NERO. It can route big designs as its runtime and memory footprint is almost linear (with respect to the number of gates). It has successfully routed design of more than *150K* gates.

> **Note**
>
> **Slow Layer Assignment.** Most of the time, the layer assignment stage is fast (less than a dozen seconds), but in some instances it can take more than a dozen *minutes*. This is a known bug and will be corrected in later releases.

After each run, KATANA displays a set of *completion ratios* which must all be equal to *100%* or (`NNNN+0`) if the detailed routing has been successfull. In the event of a failure, on a saturated design, you may tweak the three following configuration parameters:

1. `katana.hTrackReservedLocal`, the number of track reserved for local routing, that quantity is substracted from the edge capacities (global routing) to give a sense of the cluttering inside the GCells.
2. `katana.vTrackReservedLocal`, same as above.
3. `etesian.spaceMargin`, increase the free area of the overall design so the routing density decrease.

The idea is to increase the horizontal and vertical local track reservation until the detailed router succeed. But in doing so we make the task of the global router more and more difficult as the capacity of the edges decrease, and at some point it will fail too. So this is a balance.

Routing a design is done in four ordered steps:

1. Detailed pre-route **P&R** $\rightarrow$ **Step by Step** $\rightarrow$ **Detailed PreRoute**
2. Global routing **P&R** $\rightarrow$ **Step by Step** $\rightarrow$ **Global Route**
3. Detailed routing **P&R** $\rightarrow$ **Step by Step** $\rightarrow$ **Detailed Route**
4. Finalize routing **P&R** $\rightarrow$ **Step by Step** $\rightarrow$ **Finalize Route**

It is possible to supply to the router a complete wiring for some nets that the user's wants to be routed according to a specific topology. The supplied topology must respect the building rules of the ANABATIC database (contacts must be, *terminals*, *turns*, *h-tee* & *v-tee* only). During the first step `Detailed Pre-Route` the router will solve overlaps between the segments, without making any dogleg. If no pre-routed topologies are present, this step may be ommited. Any net routed at this step is then fixed and become unmovable for the later stages.

After the detailed routing step the KATANA data-structure is still active (the Hurricane wiring is decorated). The finalize step performs the removal of the KATANA data-structure, and it is not advisable to save the design before that step.

You may visualize the density (saturation) of either the edges (global routing) or the GCells (detailed routing) until the routing is finalized. Special layers appears to that effect in the The Layers&Go Tab.

Katana Configuration Parameters The Anabatic parameters control the layer assignment step.

All the defaults value given below are from the default Alliance technology (**`cmos`** and **`SxLib`** cell gauge/routing gauge).

Parameter Identifier	**Type**	**Default**
Anabatic Parameters		
`anabatic.topRoutingLayer`	TypeString	**`METAL5`**
	Define the highest metal layer that will be used for routing (inclusive).	
`anabatic.globalLengthThreshold`	TypeInt	**`1450`**
	This parameter is used by a layer assignment method which is no longer used (did not give good results)	
`anabatic.saturateRatio`	TypePercentage	**`80`**
	If `M(x)` density is above this ratio, move up feedthru global segments up from depth `x` to `x+2`	
`anabatic.saturateRp`	TypeInt	**`8`**
	If a GCell contains more terminals (**`RoutingPad`**) than that number, force a move up of the connecting segments to those in excess	
Katana Parameters		
`katana.hTracksReservedLocal`	TypeInt	**`3`**
	To take account the tracks needed *inside* a GCell to build the *local* routing, decrease the capacity of the edges of the global router. Horizontal and vertical locally reserved capacity can be distinguished for more accuracy.	
`katana.vTracksReservedLocal`	TypeInt	**`3`**
	cf. `kite.hTracksReservedLocal`	
`katana.eventsLimit`	TypeInt	**`4000002`**
	The maximum number of segment displacements, this is a last ditch safety against infinite loop. It's perhaps a little too low for big designs	
`katana.ripupCost`	TypeInt	**`3`**
	Differential introduced between two ripup cost to avoid a loop between two ripped up segments	
`katana.strapRipupLimit`	TypeInt	**`16`**
	Maximum number of ripup for *strap* segments	
`katana.localRipupLimit`	TypeInt	**`9`**
	Maximum number of ripup for *local* segments	
`katana.globalRipupLimit`	TypeInt	**`5`**
	Maximum number of ripup for *global* segments, when this limit is reached, triggers topologic modification	
`katana.longGlobalRipupLimit`	TypeInt	**`5`**
	Maximum number of ripup for *long global* segments, when this limit is reached, triggers topological modification	

Executing Python Scripts in Cgt

Python/Stratus scripts can be executed either in text or graphical mode.

Note

How Cgt Locates Python Scripts: **`cgt`** uses the Python `import` mechanism to load Python scripts. So you must give the name of your script whitout `.py` extention and it must be reachable through the `PYTHONPATH`. You may uses the dotted module notation.

A Python/Stratus script must contains a function called `ScriptMain()` with one optional argument, the graphical editor into which it may be running (will be set to `None` in text mode). The Python interface to the editor (type: **`CellViewer`**) is limited to basic capabilities only.

Any script given on the command line will be run immediatly *after* the initializations and *before* any other argument is processed.

For more explanation on Python scripts see Python Interface to Coriolis.

Printing & Snapshots

Printing or saving into a PDF is fairly simple, just uses the **File -> Print** menu or the `CTRL+P` shortcut to open the dialog box.

The print functionality uses exactly the same rendering mechanism as for the screen, beeing almost *WYSIWYG*. Thus, to obtain the best results it is advisable to select the `Coriolis.Printer` look (in the *Controller*), which uses a white background and much suited for high resolutions `32x32` pixels patterns

There is also two mode of printing selectable through the *Controller* **Settings -> Misc -> Printer/Snapshot Mode**:

Mode	DPI (approx.)	Intended Usage
Cell Mode	150	For single `Cell` printing or very small designs. Patterns will be bigger and more readable.
Design Mode	300	For designs (mostly commposed of wires and cells outlines).

Note

The pdf file size Be aware that the generated PDF files are indeed only pixmaps. So they can grew very large if you select paper format above `A2` or similar.

Saving into an image is subject to the same remarks as for PDF.

Memento of Shortcuts in Graphic Mode

The main application binary is **`cgt`**.

Category	**Keys**	**Action**
Moves	`Up`, `Down` `Left`, `Right`	Shift the view in the according direction
Fit	`f`	Fit to the Cell abutment box
Refresh	`CTRL+L`	Triggers a complete display redraw
Goto	`g`	*apperture* is the minimum side of the area displayed around the point to go to. It's an alternative way of setting the zoom level

... continued on next page

Category	Keys	Action
Zoom	z, m	Respectively zoom by a 2 factor and *unzoom* by a 2 factor
	Area Zoom	You can perform a zoom to an area. Define the zoom area by *holding down the left mouse button* while moving the mouse.
Selection	Area Selection	You can select displayed objects under an area. Define the selection area by *holding down the right mouse button* while moving the mouse.
	Toggle Selection	You can toggle the selection of one object under the mouse position by pressing CTRL and pressing down *the right mouse button*. A popup list of what's under the position shows up into which you can toggle the selection state of one item.
	S	Toggle the selection visibility
Controller	CTRL+I	Show/hide the controller window. It's the Swiss Army Knife of the viewer. From it, you can fine-control the display and inspect almost everything in your design.
Rulers	k, ESC	One stroke on k enters the ruler mode, in which you can draw one ruler. You can exit the ruler mode by pressing ESC. Once in ruler mode, the first click on the *left mouse button* sets the ruler's starting point and the second click the ruler's end point. The second click exits automatically the ruler mode.
	K	Clears all the drawn rulers
Print	CTRL+P	Currently rather crude. It's a direct copy of what's displayed in pixels. So the resulting picture will be a little blurred due to anti-aliasing mechanism.
Open/Close	CTRL+O	Opens a new design. The design name must be given without path or extention.
	CTRL+W	Close the current viewer window, but do not quit the application.
	CTRL+Q	*CTRL+Q* quit the application (closing all windows).
Hierarchy	CTRL+Down	Go one hierarchy level down. That is, if there is an *instance* under the cursor position, load it's *model* Cell in place of the current one.
	CTRL+Up	Go one hierarchy level up. if we have entered the current model through CTRL+Down reload the previous model (the one in which this model is instanciated).

Cgt Command Line Options

Appart from the obvious `--text` options, all can be used for text and graphical mode.

Arguments	**Meaning**
-t|--text	Instruct **`cgt`** to run in text mode.
-L|--log-mode	Disable the uses of ANSI escape sequence on the **`tty`**. Useful when the output is redirected to a file.
-c <cell>|--cell=<cell>	The name of the design to load, without leading path or extention.
-m <val>|--margin=<val>	Percentage *val* of white space for the placer (ETESIAN).
--events-limit=<count>	The maximal number of events after which the router will stops. This is mainly a failsafe against looping. The limit is sets to 4 millions of iteration which should suffice to any design of *100K*. gates. For bigger designs you may wants to increase this limit.
-G|--global-route	Run the global router (KATANA).
-R|--detailed-route	Run the detailed router (KATANA).
-s|--save-design=<routed>	The design into which the routed layout will be saved. It is strongly recommanded to choose a different name from the source (unrouted) design.
--stratus-script=<module>	Run the Python/Stratus script `module`. See Python Scripts in Cgt.

Some Examples :

- Run both global and detailed router, then save the routed design :

```
> cgt -v -t -G -R --cell=design --save-design=design_r
```

Miscellaneous Settings

Parameter Identifier	Type	Default
Verbosity/Log Parameters		
`misc.info`	TypeBool	**`False`**
	Enable display of *info* level message (**`cinfo`** stream)	
`misc.bug`	TypeBool	**`False`**
	Enable display of *bug* level message (**`cbug`** stream), messages can be a little scarry	
`misc.logMode`	TypeBool	**`False`**
	If enabled, assume that the output device is not a `tty` and suppress any escaped sequences	
`misc.verboseLevel1`	TypeBool	**`True`**
	First level of verbosity, disable level 2	
`misc.verboseLevel2`	TypeBool	**`False`**
	Second level of verbosity	
Development/Debug Parameters		
`misc.minTraceLevel`	TypeInt	0
`misc.maxTraceLevel`	TypeInt	0
	Display trace information *between* those two levels (**`cdebug`** stream)	
`misc.catchCore`	TypeBool	**`False`**
	By default, **`cgt`** do not dump core. To generate one set this flag to **`True`**	

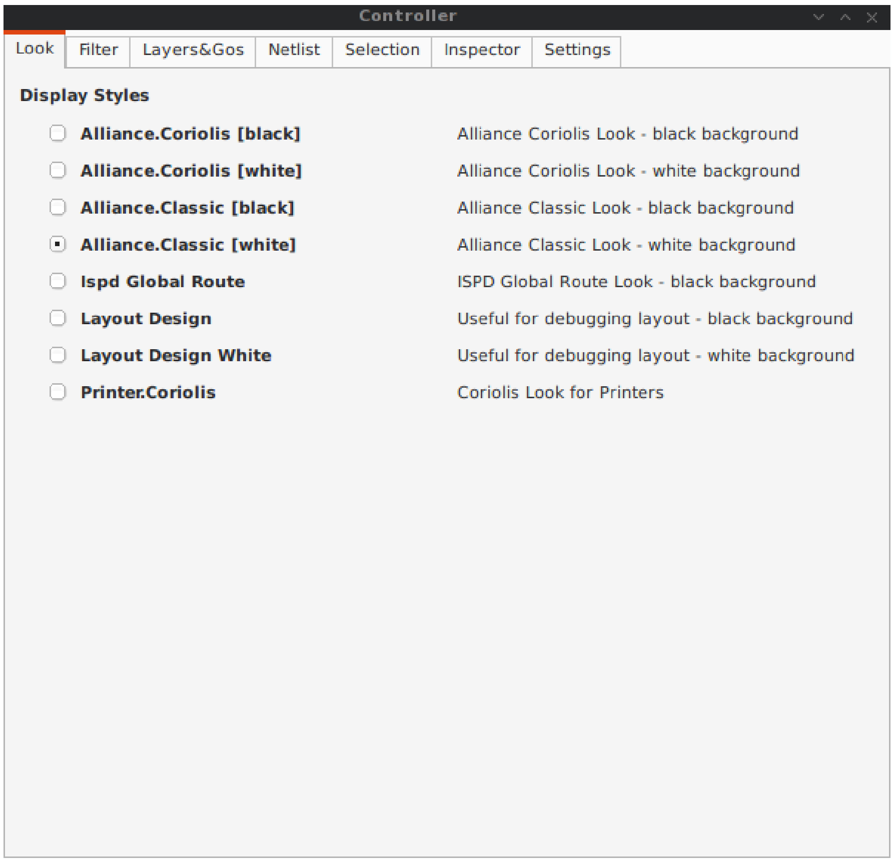
The Controller

The *Controller* window is composed of seven tabs:

1. The Look Tab to select the display style.
2. The Filter Tab the hierarchical levels to be displayed, the look of rubbers and the dimension units.
3. The Layers&Go Tab to selectively hide/display layers.
4. The Netlist Tab to browse through the *netlist*. Works in association with the *Selection* tab.
5. The Selection Tab allow to view all the currently selected elements.
6. The Inspector Tab browse through either the DataBase, the Cell or the current selection.
7. The Settings Tab access all the tool's configuration settings.

The Look Tab

You can select how the layout will be displayed. There is a special one `Printer.Coriolis` specifically designed for Printing & Snapshots. You should select it prior to calling the print or snapshot dialog boxes.



Display Styles

☐	Alliance.Coriolis [black]	Alliance Coriolis Look - black background
☐	Alliance.Coriolis [white]	Alliance Coriolis Look - white background
☐	Alliance.Classic [black]	Alliance Classic Look - black background
●	Alliance.Classic [white]	Alliance Classic Look - white background
☐	Ispd Global Route	ISPD Global Route Look - black background
☐	Layout Design	Useful for debugging layout - black background
☐	Layout Design White	Useful for debugging layout - white background
☐	Printer.Coriolis	Coriolis Look for Printers

The Filter Tab

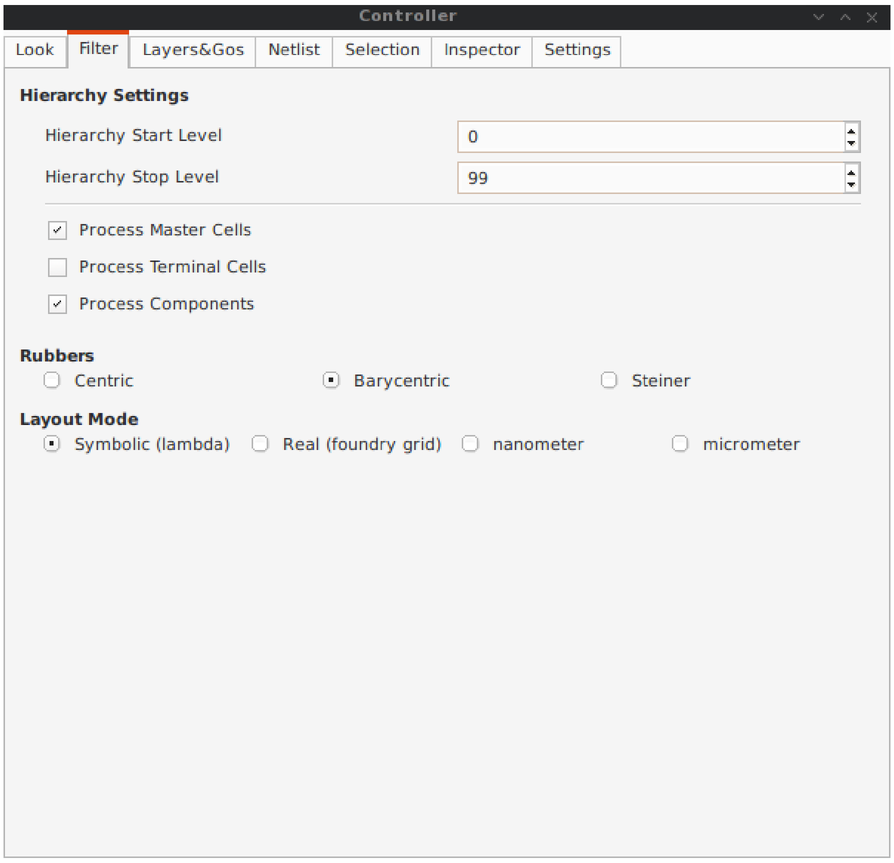
The filter tab let you select what hierarchical levels of your design will be displayed. Hierarchy level are numbered top-down: the level 0 correspond to the top-level cell, the level one to the instances of the top-level Cell and so on.

There are also check boxes to enable/disable the processing of Terminal Cell, Master Cells and Compnents. The processing of Terminal Cell (hierarchy leaf cells) is disabled by default when you load a hierarchical design and enabled when you load a single Cell.

You can choose what kind of form to give to the rubbers and the type of unit used to display coordinates.

Note

What are Rubbers: HURRICANE uses *Rubbers* to materialize physical gaps in net topology. That is, if some wires are missing to connect two or more parts of net, a *rubber* will be drawn between them to signal the gap.
For example, after the detailed routing no *rubbers* should remains. They have been made *very* visibles as big violet lines...



Hierarchy Settings

Hierarchy Start Level: 0
Hierarchy Stop Level: 99

[✓] Process Master Cells
[ ] Process Terminal Cells
[✓] Process Components

Rubbers

Centric [✓ Barycentric] Steiner

Layout Mode

[✓ Symbolic (lambda)] Real (foundry grid) nanometer micrometer

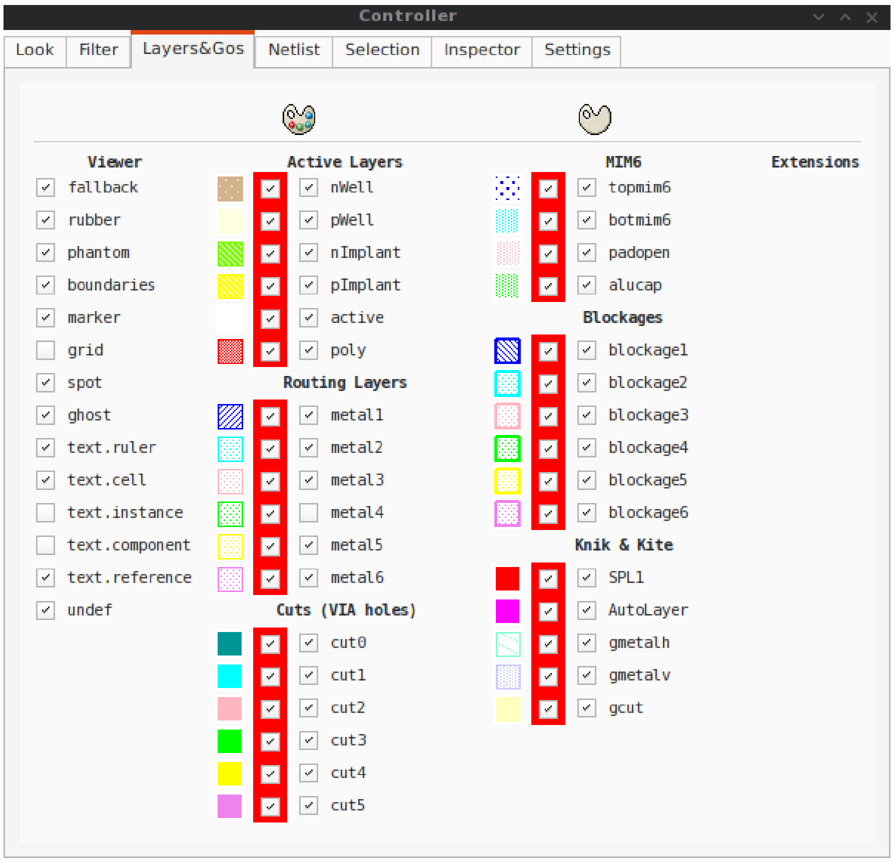
The Layers&Go Tab

Control the individual display of all *layers* and *Gos*.

- *Layers* correspond to a true physical layer. From a HURRICANE point of view they are all the *BasicLayers* (could be matched to GDSII).
- *Gos* stands from *Graphical Objects*, they are drawings that have no physical existence but are added by the various tools to display extra information. One good exemple is the density map of the detailed router, to easily locate congested areas.

For each layer/Go there are two check boxes:

- The normal one triggers the display.
- The red-outlined allows objects of that layer to be selectable or not.



Viewer
- ✓ fallback
- ✓ rubber
- ✓ phantom
- ✓ boundaries
- ✓ marker
- ☐ grid
- ✓ spot
- ✓ ghost
- ✓ text.ruler
- ✓ text.cell
- ☐ text.instance
- ☐ text.component
- ✓ text.reference
- ✓ undef

Active Layers
- ✓ ✓ nWell
- ✓ ✓ pWell
- ✓ ✓ nImplant
- ✓ ✓ pImplant
- ✓ ✓ active
- ✓ ✓ poly

Routing Layers
- ✓ ✓ metal1
- ✓ ✓ metal2
- ✓ ✓ metal3
- ✓ ☐ metal4
- ✓ ✓ metal5
- ✓ ✓ metal6

Cuts (VIA holes)
- ✓ ✓ cut0
- ✓ ✓ cut1
- ✓ ✓ cut2
- ✓ ✓ cut3
- ✓ ✓ cut4
- ✓ ✓ cut5

MIM6
- ✓ ✓ topmim6
- ✓ ✓ botmim6
- ✓ ✓ padopen
- ✓ ✓ alucap

Blockages
- ✓ ✓ blockage1
- ✓ ✓ blockage2
- ✓ ✓ blockage3
- ✓ ✓ blockage4
- ✓ ✓ blockage5
- ✓ ✓ blockage6

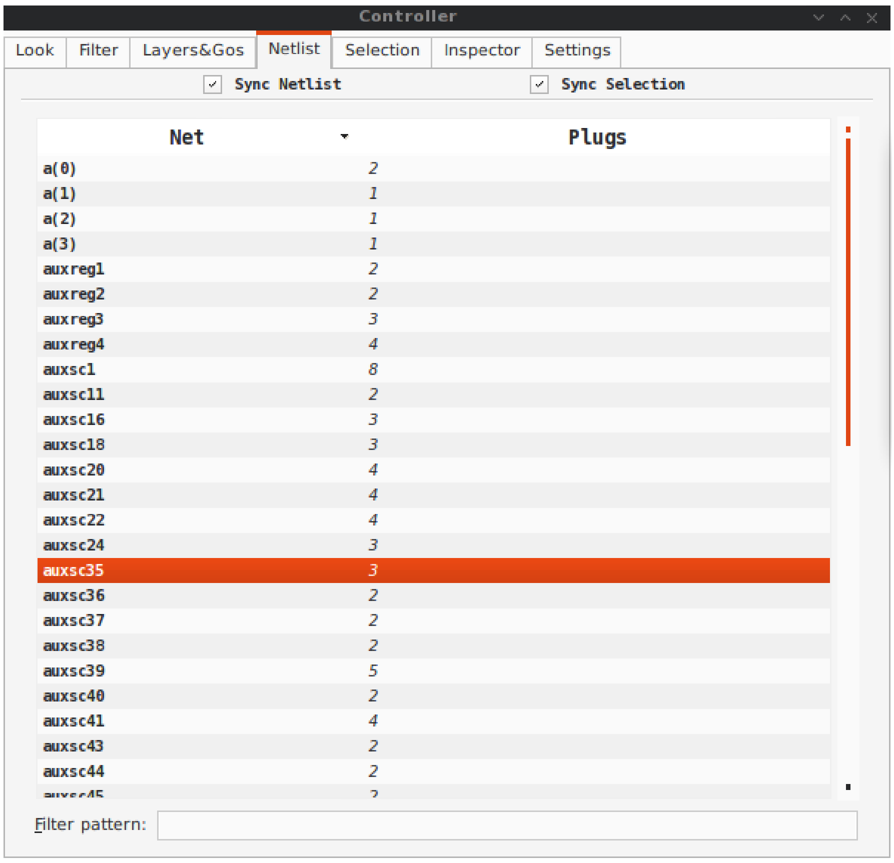
Knik & Kite
- ✓ ✓ SPL1
- ✓ ✓ AutoLayer
- ✓ ✓ gmetalh
- ✓ ✓ gmetalv
- ✓ ✓ gcut

Extensions

The Netlist Tab

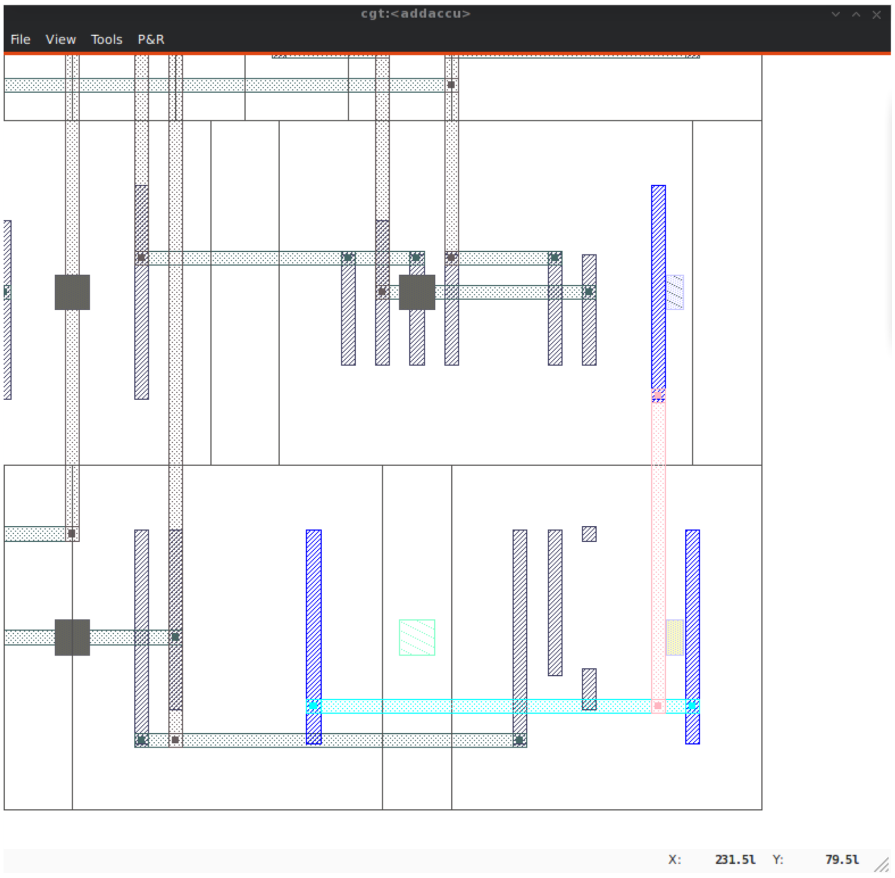
The *Netlist* tab shows the list of nets... By default the tab is not *synched* with the displayed Cell. To see the nets you must check the **Sync Netlist** checkbox. You can narrow the set of displayed nets by using the filter pattern (supports regular expressions).

An very useful feature is to enable the **Sync Selection**, which will automatically select all the components of the selected net(s). You can select multiple nets. In the figure the net `auxsc35` is selected and is highlited in the *Viewer*.



Net	Plugs
a(0)	2
a(1)	1
a(2)	1
a(3)	1
auxreg1	2
auxreg2	2
auxreg3	3
auxreg4	4
auxsc1	8
auxsc11	2
auxsc16	3
auxsc18	3
auxsc20	4
auxsc21	4
auxsc22	4
auxsc24	3
auxsc35	3
auxsc36	2
auxsc37	2
auxsc38	2
auxsc39	5
auxsc40	2
auxsc41	4
auxsc43	2
auxsc44	2
auxsc45	2

Filter pattern:

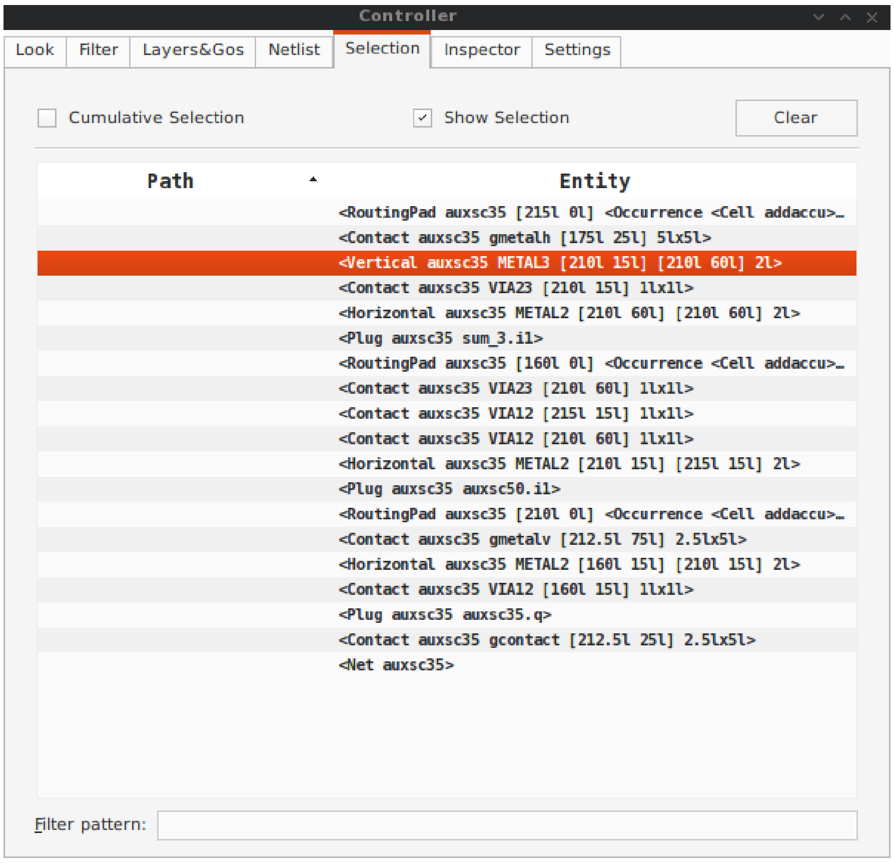


The Selection Tab

The *Selection* tab list all the components currently selecteds. They can be filtered thanks to the filter pattern.

Used in conjunction with the *Netlist* **Sync Selection** you will all see all the components part of *net*.

In this list, you can toggle individually the selection of component by pressing the `t` key. When unselected in this way a component is not removed from the the selection list but instead displayed in red italic. To see where a component is you may make it blink by repeatedly press the `t` key...



Path	Entity
	<RoutingPad auxsc35 [215l 0l] <Occurrence <Cell addaccu>…
	<Contact auxsc35 gmetalh [175l 25l] 5lx5l>
	<Vertical auxsc35 METAL3 [210l 15l] [210l 60l] 2l>
	<Contact auxsc35 VIA23 [210l 15l] 1lx1l>
	<Horizontal auxsc35 METAL2 [210l 60l] [210l 60l] 2l>
	<Plug auxsc35 sum_3.i1>
	<RoutingPad auxsc35 [160l 0l] <Occurrence <Cell addaccu>…
	<Contact auxsc35 VIA23 [210l 60l] 1lx1l>
	<Contact auxsc35 VIA12 [215l 15l] 1lx1l>
	<Contact auxsc35 VIA12 [210l 60l] 1lx1l>
	<Horizontal auxsc35 METAL2 [210l 15l] [215l 15l] 2l>
	<Plug auxsc35 auxsc50.i1>
	<RoutingPad auxsc35 [210l 0l] <Occurrence <Cell addaccu>…
	<Contact auxsc35 gmetalv [212.5l 75l] 2.5lx5l>
	<Horizontal auxsc35 METAL2 [160l 15l] [210l 15l] 2l>
	<Contact auxsc35 VIA12 [160l 15l] 1lx1l>
	<Plug auxsc35 auxsc35.q>
	<Contact auxsc35 gcontact [212.5l 25l] 2.5lx5l>
	<Net auxsc35>

The Inspector Tab

This tab is very useful, but mostly for CORIOLIS developpers. It allows to browse through the live DataBase. The *Inspector* provide three entry points:

- **DataBase**: Starts from the whole HURRICANE DataBase.
- **Cell**: Inspect the currently loaded Cell.
- **Selection**: Inspect the object currently highlited in the *Selection* tab.

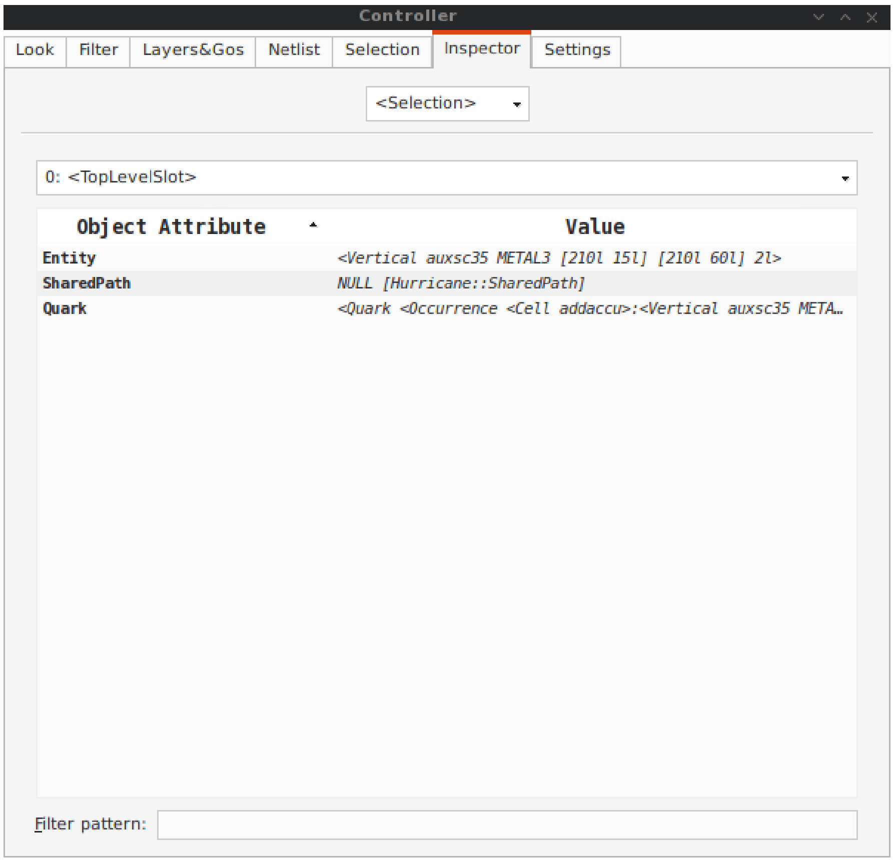
Once an entry point has been activated, you may recursively expore all it's fields using the right/left arrows.

Note

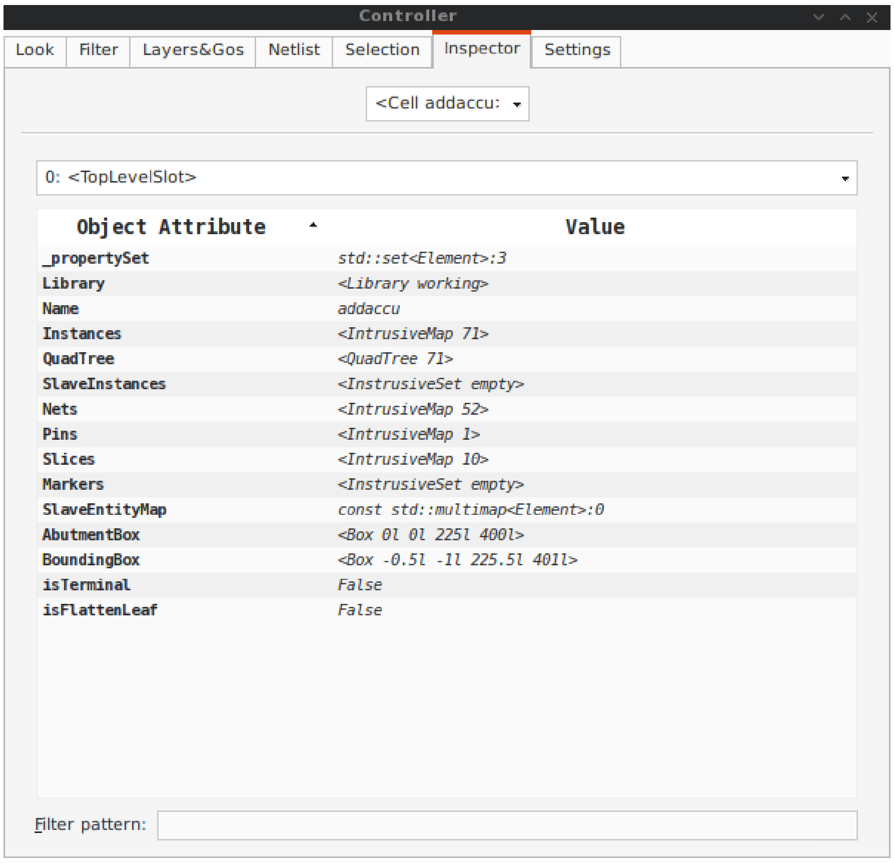
Do not put your fingers in the socket: when inspecting anything, do not modify the DataBase. If any object under inspection is deleted, you will crash the application...

Note

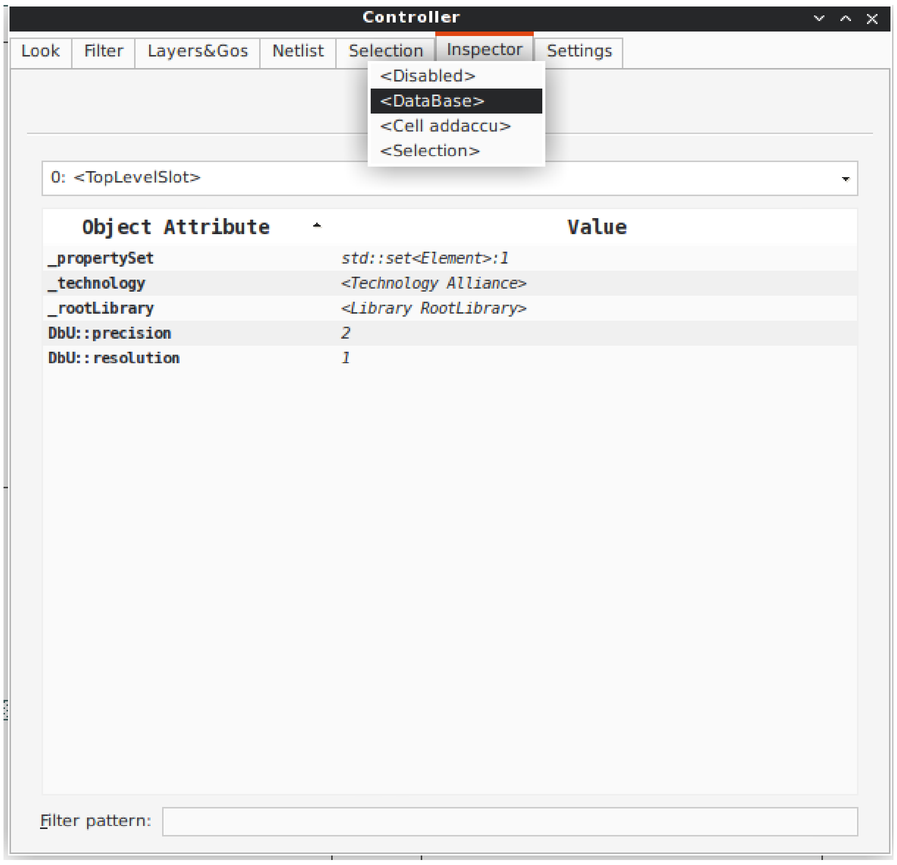
Implementation Detail: the inspector support is done with `Slot`, `Record` and `getString()`.



Object Attribute	Value
Entity	<Vertical auxsc35 METAL3 [210l 15l] [210l 60l] 2l>
SharedPath	NULL [Hurricane::SharedPath]
Quark	<Quark <Occurrence <Cell addaccu>:<Vertical auxsc35 META…



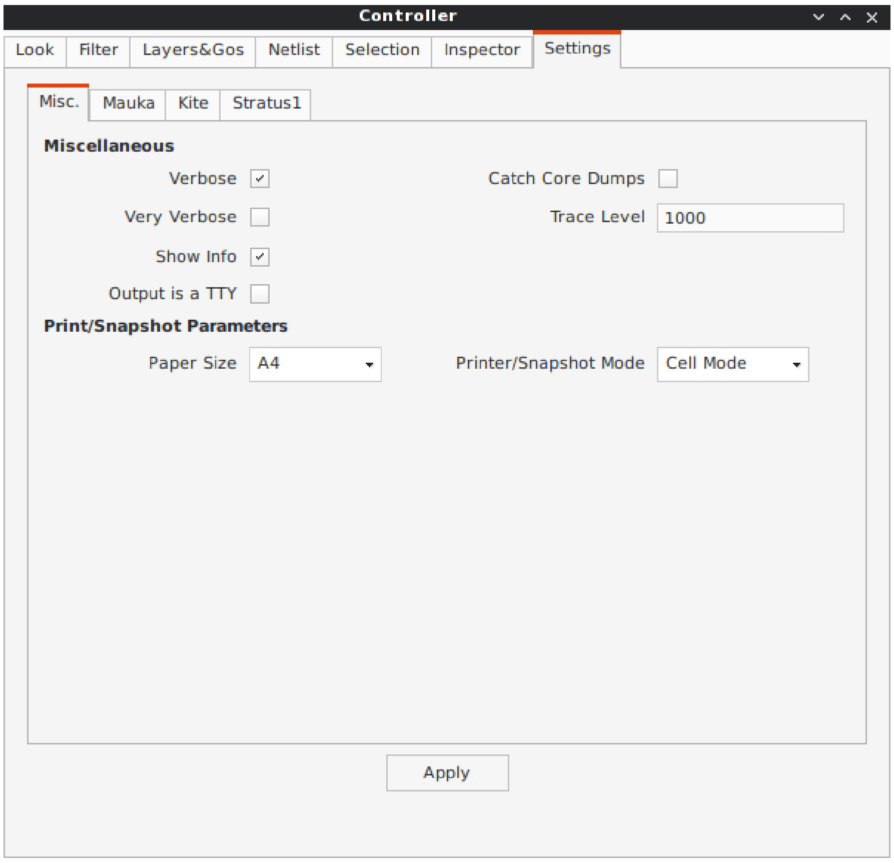
Object Attribute	Value
_propertySet	std::set<Element>:3
Library	<Library working>
Name	addaccu
Instances	<IntrusiveMap 71>
QuadTree	<QuadTree 71>
SlaveInstances	<InstrusiveSet empty>
Nets	<IntrusiveMap 52>
Pins	<IntrusiveMap 1>
Slices	<IntrusiveMap 10>
Markers	<InstrusiveSet empty>
SlaveEntityMap	const std::multimap<Element>:0
AbutmentBox	<Box 0l 0l 225l 400l>
BoundingBox	<Box -0.5l -1l 225.5l 401l>
isTerminal	False
isFlattenLeaf	False



Object Attribute	Value
_propertySet	std::set<Element>:1
_technology	<Technology Alliance>
_rootLibrary	<Library RootLibrary>
DbU::precision	2
DbU::resolution	1

The Settings Tab

Here comes the description of the *Settings* tab.



Miscellaneous

Verbose ✓

Catch Core Dumps

Very Verbose

Trace Level 1000

Show Info ✓

Output is a TTY

Print/Snapshot Parameters

Paper Size A4

Printer/Snapshot Mode Cell Mode

Python Interface for Hurricane / Coriolis

The (almost) complete interface of Hurricane is exported as a Python module and some part of the other components of Coriolis (each one in a separate module). The interface has been made to mirror as closely as possible the C++ one, so the C++ doxygen documentation could be used to write code with either languages.

Summary of the C++ Documentation

A script could be run directly in text mode from the command line or through the graphical interface (see Python Scripts in Cgt).

Aside for this requirement, the python script can contain anything valid in Python, so don't hesitate to use any package or extention.

Small example of Python/Stratus script:

```
from Hurricane import *
from Stratus   import *

def doSomething ():
    # ...
    return

def ScriptMain ( **kw ):
  editor = None
  if kw.has_key('editor') and kw['editor']:
    editor = kw['editor']
    stratus.setEditor( editor )

  doSomething()
  return

if __name__ == "__main__" :
  kw             = {}
  success        = ScriptMain( **kw )
  shellSuccess = 0
  if not success: shellSuccess = 1

  sys.exit( shellSuccess )
      ScriptMain ()
```

This typical script can be executed in two ways:

1. Run directly as a Python script, thanks to the

   ```
   if __name__ == "__main__" :
   ```

   part (this is standart Python). It is a simple adapter that will calls **ScriptMain()**.

2. Through **cgt**, either in text or graphical mode. In that case, the **ScriptMain()** is directly called trough a sub-interpreter. The arguments of the script are passed through the `**kw` dictionnary.

****kw Dictionnary**	
Parameter Key/Name	**Contents type**
`'cell'`	A Hurricane cell on which to work. Depending on the context, it may be `None`. For example, when run from **cgt**, it the cell currently loaded in the viewer, if any.
`'editor'`	The viewer from which the script is run, when lauched through **cgt**.

Plugins

Plugins are PYTHON scripts specially crafted to integrate with **cgt**. Their entry point is a **`ScriptMain()`** method as described in Python Interface to Coriolis. They can be called by user scripts through this method.

Chip Placement

Automatically perform the placement of a complete chip. This plugin, as well as the other P&R tools expect a specific top-level hierarchy for the design. The top-level hierarchy must contains the instances of all the I/O pads and **exactly one** instance named `corona` of an eponym cell `corona`. The `corona` cell in turn containing the instance of the chip's core model.

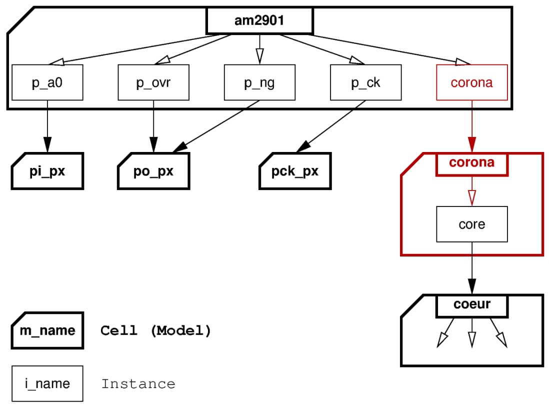
The intermediate `corona` hierarchical level has been introduced to handle the possible dis-coupling between real I/O pads supplied by a foundry and a symbolic core. So the *chip* level contains only real layout and the corona and below only symbolic layer.

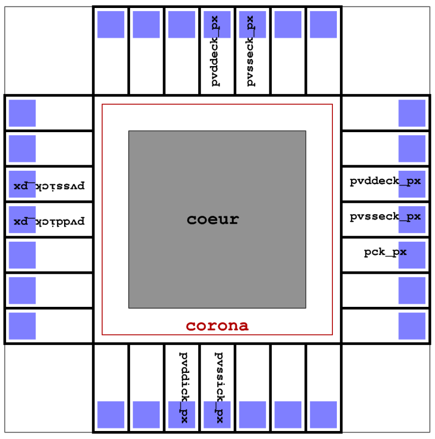
Note

This do not prevent having a design either fully symbolic (pads and core) or fully real.

Note

The `corona` also avoid the router to actually have to manage directly the pads which simplificate it's configuration and accessorily avoid to have the pads stuffed with blockages.





The designer must provide a configuration file that defines the rules for the placement of the top-level hierarchy (that is, the pads and the core). This file must be named after the chip's name, by appending `_ioring.py` (obviously, it is a PYTHON file). For instance if the chip netlist file is called `amd2901.vst`, then the configuration file must be named `amd2901_ioring.vst`.

Example of chip placement configuration file (for `AM2901`):

```
from helpers import l, u, n

chip = \
  { 'pads.ioPadGauge' : 'pxlib'
  , 'pads.south'      : [ 'p_a3'     , 'p_a2'     , 'p_a1'     , 'p_r0'
                        , 'p_vddick0', 'p_vssick0', 'p_a0'     , 'p_i6'
                        , 'p_i8'     , 'p_i7'     , 'p_r3'     ]
  , 'pads.east'       : [ 'p_zero'   , 'p_i0'     , 'p_i1'     , 'p_i2'
                        , 'p_vddeck0', 'p_vsseck0', 'p_q3'     , 'p_b0'
                        , 'p_b1'     , 'p_b2'     , 'p_b3'     ]
  , 'pads.north'      : [ 'p_noe'    , 'p_y3'     , 'p_y2'     , 'p_y1'
                        , 'p_y0'     , 'p_vddeck1', 'p_vsseck1', 'p_np'
                        , 'p_ovr'    , 'p_cout'   , 'p_ng'     ]
  , 'pads.west'       : [ 'p_cin'    , 'p_i4'     , 'p_i5'     , 'p_i3'
                        , 'p_ck'     , 'p_d0'     , 'p_d1'     , 'p_d2'
                        , 'p_d3'     , 'p_q0'     , 'p_f3'     ]
  , 'core.size'       : ( l(1500), l(1500) )
  , 'chip.size'       : ( l(3000), l(3000) )
  , 'chip.clockTree'  : True
  }
```

The file must contain *one dictionnary* named `chip`.

Chip Dictionnary	
Parameter Key/Name	**Value/Contents type**
`'pad.ioPadGauge'`	The routing gauge to use for the pad. Must be given as it differs from the one used to route standard inside the core
`'pad.south'`	Ordered list (left to right) of pad instances names to put on the south side of the chip
`'pad.east'`	Ordered list (down to up) of pad instances names to put on the east side of the chip
`'pad.north'`	Ordered list (left to right) of pad instances names to put on the north side of the chip
`'pad.west'`	Ordered list (down to up) of pad instances names to put on the west side of the chip
`'core.size'`	The size of the core (to be used by the placer)
`'chip.size'`	The size of the whole chip. The sides must be great enough to accomodate all the pads
`'chip.clockTree'`	Whether to generate a clock tree or not. This calls the ClockTree plugin

Configuration parameters, defaults are defined in `etc/coriolis2/<STECHNO>/plugins.conf`.

Parameter Identifier	**Type**	**Default**
Chip Plugin Parameters		
`chip.block.rails.count`	TypeInt	**5**
	The minimum number of rails around the core block. Must be odd and suppérior to 5. One rail for the clock and at least two pairs of power/grounds	
`chip.block.rails.hWidth`	TypeInt	**12 λ**
	The horizontal with of the rails	

... continued on next page

Parameter Identifier	Type	Default
`chip.block.rails.vWidth`	TypeInt	**12 λ**
	The vertical with of the rails	
`chip.block.rails.hSpacing`	TypeInt	**6 λ**
	The spacing, *edge to edge* of two adjacent horizontal rails	
`chip.block.rails.vSpacing`	TypeInt	**6 λ**
	The spacing, *edge to edge* of two adjacent vertical rails	

Note

If no clock tree is generated, then the clock rail is *not* created. So even if the requested number of rails `chip.block.rails.count` is, say 5, only four rails (2* `power`, 2* `ground`) will be generateds.

Clock Tree

Inserts a clock tree into a block. The clock tree uses the H strategy. The clock net is splitted into sub-nets, one for each branch of the tree.

- On **chip** design, the sub-nets are created in the model of the core block (then trans-hierarchically flattened to be shown at chip level).
- On **blocks**, the sub nets are created directly in the top block.
- The sub-nets are named according to a simple geometrical scheme. A common prefix `ck_htree`, then one postfix by level telling on which quarter of plane the sub-clock is located:

  1. `_bl`: bottom left plane quarter.
  2. `_br`: bottom right plane quarter.
  3. `_tl`: top left plane quarter.
  4. `_tr`: top right plane quarter.

  We can have `ck_htree_bl`, `ck_htree_bl_bl`, `ch_htree_bl_tl` and so on.

The clock tree plugin works in four steps:

1. Builds the clock tree: creates the top-block abutment box, compute the required levels of H tree and places the clock buffers.
2. Once the clock buffers are placed, calls the placer (Etesian) to place the ordinary standard cells, whithout disturbing clock H-tree buffers.
3. At this point we know the exact positions of all the DFFs, so we can connect them to the nearest H-tree leaf clock signal.
4. Leaf clock signals that are not connected to any DFFs are removed.

Netlist reorganisation:

- Obviously the top block or chip core model netlist is modified to contain all the clock sub-nets. The interface is *not* changed.
- If the top block contains instances of other models *and* those models contain DFFs that get re-connected to the clock sub-nets (from the top level). Changes both the model netlist and interface to propagate the relevant clock sub-nets to the instanciated model. The new model with the added clock signal is renamed with a `_cts` suffix. For example, the sub-block model `ram.vst` will become `ram_cts.vst`.

Note

If you are to re-run the clock tree plugin on a netlist, be careful to erase any previously generated `_cts` file (both netlist and layout: `rm *_cts.{ap,vst}`). And restart **cgt** to clear its memory cache.

Configuration parameters, defaults are defined in `etc/coriolis2/<STECHNO>/plugins.conf`.

Parameter Identifier	**Type**	**Default**
ClockTree Plugin Parameters		
`clockTree.minimumSide`	TypeInt	**300** λ
	The minimum size below which the clock tree will stop to perform quadri-partitions	
`clockTree.buffer`	TypeString	**buf_x2**
	The buffer model to use to drive sub-nets	

Recursive-Save (RSave)

Performs a recursive top down save of all the models from the top cell loaded in **cgt**. Forces a write of any non-terminal model. This plugin is used by the clock tree plugin after the netlist clock sub-nets creation.

A Simple Example: AM2901

To illustrate the capabilities of CORIOLIS tools and PYTHON scripting, a small example, derived from the ALLIANCE **AM2901** is supplied.

This example contains only the synthetized netlists and the **doChip.py** script which perform the whole P&R of the design.

You can generate the chip using one of the following method:

1. **Command line mode:** directly run the script:

```
dummy@lepka:AM2901> ./doChip -V --cell=amd2901
```

2. **Graphic mode:** launch **cgt**, load chip netlist `amd2901` (the top cell) then run the PYTHON script **doChip.py**.

Note

Between two consecutive run, be sure to erase the netlist/layout generateds:

```
dummy@lepka:AM2901> rm *_cts*.vst *.ap
```