SoCLib's compilation helper

Why ?

We use a lot of tricky things things

- Different SystemC backends (SystemC-OSCI, SystemCASS, SoCView) each of them is a different implementation of the same LRM, and yields incompatible objects
- Templated classes. The usual way of using templated code is to put all code in .h, having template code emitted at use in main C++ file.

  This is good for small utilities, but SystemC modules may be more than 1000 lines-long, and more that 40 of them may be used in a topcell. This may yield a single translation unit with more than 50000 lines of code, heavily templated. This implies some usage issues (compiler getting out of memory, unreasonnable compile times)

  Therefore we need two features:

- Separate implementation: Put template class definition and implementation in two separate files. Compile them separately.

  This implies template code must be explicitely instanciated with some `template class Foo<parameters>;` code. It has to be done automatically.

- Object reuse: Once modules built separately, we can put objects in a global repository and use the in a cached way.
- Different build modes (debugging, profiling release, others ??)

Reuse of current tools ??

This could be seen as reimplementing make, or even SCons, and this is not totally false. This is all about flexibility, and user-input readibility.

There is no build tool known out the which does object caching and template instantiation at the same time. Even if current build tools may be enhanced to do the job, this is not an easy task.

This very tool has been implemented as a `make` wrapper before, generated Makefiles were unreadable (all templates parameters in the middle, ...), and it did not work so well.

Configuration

In order to compile SoCLib objects, we need:

- a working SystemC installation
- a working GNU-C++ compiler

Soclib-cc processes three files in order:

1. *soclib-dir]*`/etc/soclib.conf`
2. `~/.soclib/global.conf`
3. `./soclib.conf`

These files contain multiple concurrent configurations for building SoCLib. One of them will be chosen (explicitely) as the default one. Others may be used on demand (through command-line or local configuration file)

- File [1] is installation-global. It should be modified by the administrator for a network-wide configuration.
- File [2] is useful for a developper's own configuration. This allows to use a local development branch of a local SystemC, ...
- File [3] is directory-local, this allows to choose different flavours of previously declared configurations

See SoclibConf for a usage guide to these files.

Usage

Soclib-cc may be used two ways:

- As a compiler wrapper. It will just be a CXX wrapper, handling compilation or linkage on demand. This can be useful for external Makefile integration.
- As a complete platform compiler. From an ad-hoc platform definition (wrappers can be written to accept other formats), the complete simulator will be compiled.

Try running `soclib-cc -h`.

As a compiler

The usual way:

```
$ soclib-cc -c -o obj.o file.cc
$ soclib-cc -o sim obj.o ...
```

As a platform compiler

```
$ soclib-cc -p platform_def
```

Global flags

-v
: Print command lines

-q
: Dont say anything

-m MODE
: Change compilation mode (release, debug, prof) This changes in an homogenous way the building

Object repository

As objects can be reused between builds, or even between platforms, we may want to place objects in a global repository.

Default repository is in current directory, in '$(SOCLIB)/repos/'. If you want, you can specify another absolute path in configuration.