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, with separate implementation. In order to reduce compilation time when a file changes, splitting implementation from header is the right thing to do. Unfortunatelly it involves manually instaciating source code for each used template.
- Different build modes (debugging, profiling release, others ??)
- Compiled objects reuse: as we may build lots of SoCs with minor modifications, reusing compiled objects from one build to the other seems interesting (considering templated C++ build time)

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.

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. It had been developped in a week, debugged in two weeks.

Soclib-cc has been totally developped in 16 hours.

Configuration

In order to compile SoCLib objects, we need:

- a working SystemC installation
- a working GNU-Bfd installation (used for loading binaries into the platform)
- 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 'repos/'. If you want, you can specify an absolute path in configuration, enabling a global object repository (per user, per host, per network...).