SoCLib's compilation helper

What do we want ?

We want a tool which can build a complete virtual prototype (aka SystemC simulator) from:

- a topcell provided by user
- a list of parametrable components, and their parameters

Therefore we need the following features:

- Component indexation in the library tree
- Parametrization of source files (about C++, it means templated code)
- Build of all these files
- compile all these with different SystemC implementations
- compile all these with different compilation modes (debug, release, profiling, ?)
- without having to change build files

Additional features:

- Parallel compilation (virtual prototypes may be huge)
- Object caching (users nearly always use the same parameters ? ccache is be an option)

Why ?

Reworded, we need:

- Different SystemC backends (SystemC-OSCI, SystemCASS, SoCView) each of them is a different implementation of the same LRM, and yields incompatible objects, thus objects have to be in separate directories in order to be able to reuse objects without conflicts
- Metadata-based component definition, including used source files. Components are not necessarily implemented in an unique file, but may be scattered through different files, metadata does the glue.
- 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 libraries (like STL), 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 more features:
  - Separate implementation: Put template class definition (header) and implementation (.cpp) in two separate files. Compile them separately.
    This implies that the C++ templates must be explicitely instanciated with some `template class Foo<parameters>;` code. It has to be done automatically.
  - Object reuse: Once modules are built separately, we can put objects in a global repository and use the in a cached way.
- Different build modes (debugging, profiling release, others ??)

Why not reuse existing tools ?

There is no known build tool out there 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. Moreover, resulting code would be a kludge. Soclib-cc has been implemented as a `make` wrapper before, it was not usable:

- generated Makefiles were unreadable (all templates parameters in the middle, ?)
- it did not work so well (make interprets `:` a special way, and escaping is nearly unusable)
- the code generator was a big ugly piece of software
- we still had to do `.sd` file indexation
- we still had to emit template instantiation code

Soclib-cc could be seen as reimplementing make, or even SCons, and this is not totally wrong. But we added other features:

- Template instantiation
- Separate source enumeration
- Object caching

This is all about flexibility, and user-input readability.

Compilation flow

The resulting compilation flow is as follows:

Design

Usage

Soclib-cc may be used three 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. (the `-c` option)
- As a component compiler (the `-l` option)
- 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. (the `-p` option)

Try running `soclib-cc -h`.