# Cosimulation using SoCLib components and RTL models

You may use CABA models together with RTL models using ModelSim. This needs the following parts:

- a set of SystemC models
- a set of Verilog/VHDL models
- glue wrappers where needed, exporting a RTL model to SystemC or SystemC to RTL (scgenmod to export RTL model to SystemC but this is not covered in this guide)
- a SystemC clock driver (we had some issues with vhdl clock driver), i.e. a module bagotting clock signal

SystemC modules are SoCLib ones and are usually compiled with SoCLib-cc. They come with pure-c++ dependancies which must be linked together with the modules.

Due to its simulator core design, ModelSim has to compile SystemC modules a special way, and has a dedicated tool to compile SystemC/C++ files: `sccom`.

Soclib-cc has three main jobs:

- Select modules and dependancies from a platform description file,
- Explicitly instantiate C++ templates,
- Call the C++ compiler. Only this step is implemented in `sccom`,
- Call the vhdl or verilog compiler: `vcom` and `vlog`.

The flow is as in the picture:

soclib-cc handles most of this automagically if correctly configured. This guide explains how to set things up.

Moreover, the C/C++ only dependancies are not directly compileable with the dedicated ModelSim tool, but can be *injected* at the last time, for the linkage phase (`sccom -link`).

# How to configure SoCLib-cc to call ModelSim compiler driver

Sometimes, the C++-only dependencies of SystemC modules need to Know about SystemC types. Therefore, SystemC includes must be available.

soclib-cc needs new configuration sections for

- the compiler used by sccom
- the path to ModelSim's SystemC implementation
- used flags
- object file names pattern in sccom `work` directory

For all these, we must create 3 new configurations in soclib-cc's configuration file:

- a compiler
- a SystemC library
- a build environment

```
# Definition of the compiler used for ModelSim-usable SoCLib components.
# We use sccom for components compilation and linkage, gcc/g++ for utilities
config.modelsim_toolchain = Toolchain(
        parent = config.toolchain,
        # Must use this.
        tool_SCCOM_CC = 'sccom',
        tool_SCCOM_CXX = 'sccom',
        tool_CC = '/users/soft/mentor/modelsim-6.5c/modeltech/bin/gcc',
        tool_CXX = '/users/soft/mentor/modelsim-6.5c/modeltech/bin/g++',
        tool_CC_LINKER = 'sccom',
        tool_CXX_LINKER = 'sccom',
        tool_VHDL = 'vcom',
        tool_VERILOG = 'vlog',
        # Modelsim cant do parallel builds :'(
        max_processes = 1,
        # No cflags are needed, sccom forces them
        cflags = ['-m32'],
        # Special features, it has a -link invocation needed at end...
        libs = ['-link'],
        # VHDL compilation flags
        vflags = ['-05', '-quiet', '-check-synthesis'],
)

# Definition of the ModelSim SystemC implementation. Must modify the
# path according to the ModelSim current installation.
config.libsystemc_modelsim = Library(
        name = 'systemc',
        # This special vendor attributes enables some quirks in soclib-cc
        vendor = 'modelsim',
        # This is the path of the produced .o files when compiled with sccom.
        # You have to try it by hand, and adapt
        sc_workpath = "work/_sc/linux_gcc-4.1.2",
        # Empty useless variables
        libs = [],
        # cflags have to be deducted from actual invocation
        # Try using sccom -v by hand
        cflags = ['-I/users/soft/mentor/modelsim-6.5c/modeltech/include/systemc',
                  '-I/users/soft/mentor/modelsim-6.5c/modeltech/include',
                  '-I'+config.path+'/soclib/lib/include'],
        )

# Definition of a new build environment, which can be referenced with 'soclib-cc -t'
config.modelsim = BuildEnv(
        parent = config.build_env,
        toolchain = config.modelsim_toolchain,
        libraries = [config.libsystemc_modelsim],
        # Where temporary files lies, beware that if you set a global path,
        # you'll need a mechanism to make user-unique directories.
        repos = "/tmp/",
        )
```

## SystemC modules in ModelSim limitations

All modules that may be used from the outside of the SystemC-part (from RTL or from GUI) have to be declared with a special macro (`SC_MODULE_EXPORT`).

There is no `sc_main()` function in modelsim-based simulators. The top module must be a `sc_module` with no interfaces. This probably needs a rewrite of your netlists. (Note: If you use DSX-generated netlists, this is done transparently)

## Component Metadata

RTL modules, exactly like their caba counterparts, have their own metadata files. Caba modules can use RTL ones as dependancies.

## Usage

Now we configured soclib-cc, we can compile a complete SystemC system.

Let's have an example system with two basic components communicating through a fifo.

We'll use

- SoCLib SystemC Fifo Ports,
- a VHDL `fifo_gen` component,
- a VHDL-SystemC `fifo_gen_wrapper` wrapper,
- a SystemC `fifo_wrapper` hosting a pure-C++.

This basic system has to be modeled as the following tree:

It contains:

`rtl:fifo_gen`
: The VHDL component writing to the Fifo

`caba:fifo_gen`
: The VHDL/SystemC wrapper to export `fifo_gen` to the SystemC world

`caba:fifo_reader`
: A SystemC component reading from the fifo

`caba:topcell`
: A SystemC component implementing the topcell

`caba:system_driver`
: A SystemC component controlling `reset` and `clock` signals

In order to simulate this system we need to:

- Reset the work directory, to make sure,

  ```
  $ rm -rf work transcript modelsim.ini vsim.wlf
  ```
- Initialize modelsim `work` directory,

  ```
  $ vlib work
  $ vmap work work
  ```
- Compile the SystemC system driver with soclib-cc, all dependancies are pulled with it. `.sd` metadata are needed (even for the VHDL/SystemC wrapper and VHDL modules), see in tarball.

  ```
  $ soclib-cc -1 caba:system_driver -t modelsim -v -o sccom-link.o
  ```
- Open modelsim with the platform

  ```
  $ vsim -sclib work system_driver
  ```

See the attached tarball for the complete example