SystemCASS
Description of the Functionality

Richard Buchmann and Alain Greiner
E-mail: `richard.buchmann@lip6.fr`, `alain.greiner@lip6.fr`

1 SystemCASS Overview

SystemCASS is a simulator that executes models described in **SystemC language** 10x times faster than the SystemC simulation kernel. Its goal is to provide cycle based simulation of systems built upon hardware and software components, in order to evaluate performances (hardware/software partitioning, system validation, early software development).

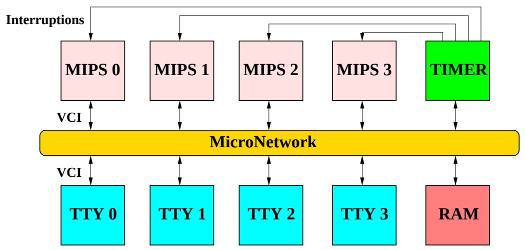

Figure 1. Typical embedded system build around VCI interfaces.

SystemCASS simulates a **SystemC** netlist of predefined and/or user defined components. The components used to validate the methodology are part of the **SoCLIB** library. The SoCLIB library contains efficient models that are VCI, bit and cycle accurate. Efficiency is obtained through the use of the Finite State Machine description (Figure 2).

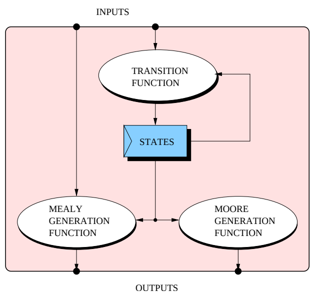

Figure 2. Finite State Machine Modeling

SystemCASS accepts a SystemC language subset including :

- core class :
  sc_module, sc_signal, sc_in, sc_out, sc_inout...
- core functions :
  sc_start, sc_stop, sc_simulation_time...
- basic data types :
  sc_i[u]nt, sc_big[u]int...

SystemCASS doesn't currently include standard channels, methodology-specific channels (master/slave library, verification library) and elementary channels (timer, mutex, semaphore, fifo, etc.).

2 Installation

Set **SYSTEMC** environment variable to the **SystemCASS** base directory. Run the SystemCASS Makefile.

The distribution is as follow :

- *src* : Source files
- *includes* : Header files
- *lib-linux* : Libraries for linux distribution
- *docs* : Documentation
- *examples* :
  - *soclib_date04* : An hardware timer raises some interruptions on 4 Mips R3000 periodically.
  - *soclib_spg* : An specific hardware configures a simple DMA.
  - *soclib_date05* : Gigabit ethernet application using multiprocessor architecture.

3 Simulator Execution

To use SystemCASS simulator, you just need to include the SystemCASS headers and link to SystemCASS library.
To debug, two ways are available :

- link to the debug library and use a C++ debugger.
- generate a trace file by using dedicated functions and use a VCD viewer.

4 Simulator Kernel Compilation

Some macro definitions are available to accelerate the simulation, to check the netlist, variable dependancies, the scheduling to help on debugging and so on.
You need to modify the Options.def file and rebuild the libraries.

5 Options

SystemCASS has two ways to compute the scheduling :

- from the static sensitivity list (default)
- from the port dependancy graph[**?**] (USE_PORT_DEPENDANCY defined) : The component designer need to declare the port dependancies in the constructor. *Syntax : outPort(inPort); DUMP_COMBINATIONAL_LIST2DOT definition generates some DOT files to check dependancies.*

The generated scheduling is written in code-XXX.cc file. KEEP_GENERATED_CODE definition allows to keep the file after simulation execution.

PRINT_SCHEDULE definition prints the scheduling at execution time.
DUMP_SCHEDULE_STATS prints some miscellanous statistics.
NO_STATIC_SCHEDULE disables the static scheduling.