Institut
Mines-Telecom

Paris Sorbonne
University

Using TTool with SoCLib

Daniela Genius, Ludovic Apvrille
daniela.genius@lip6.fr
ludovic.apvrille@telecom-paristech.fr

Users' Guide to the
SoCLib extension
of **TTool/AVATAR**

Virtual Prototyping of AVATAR Diagrams

Diagram Editor

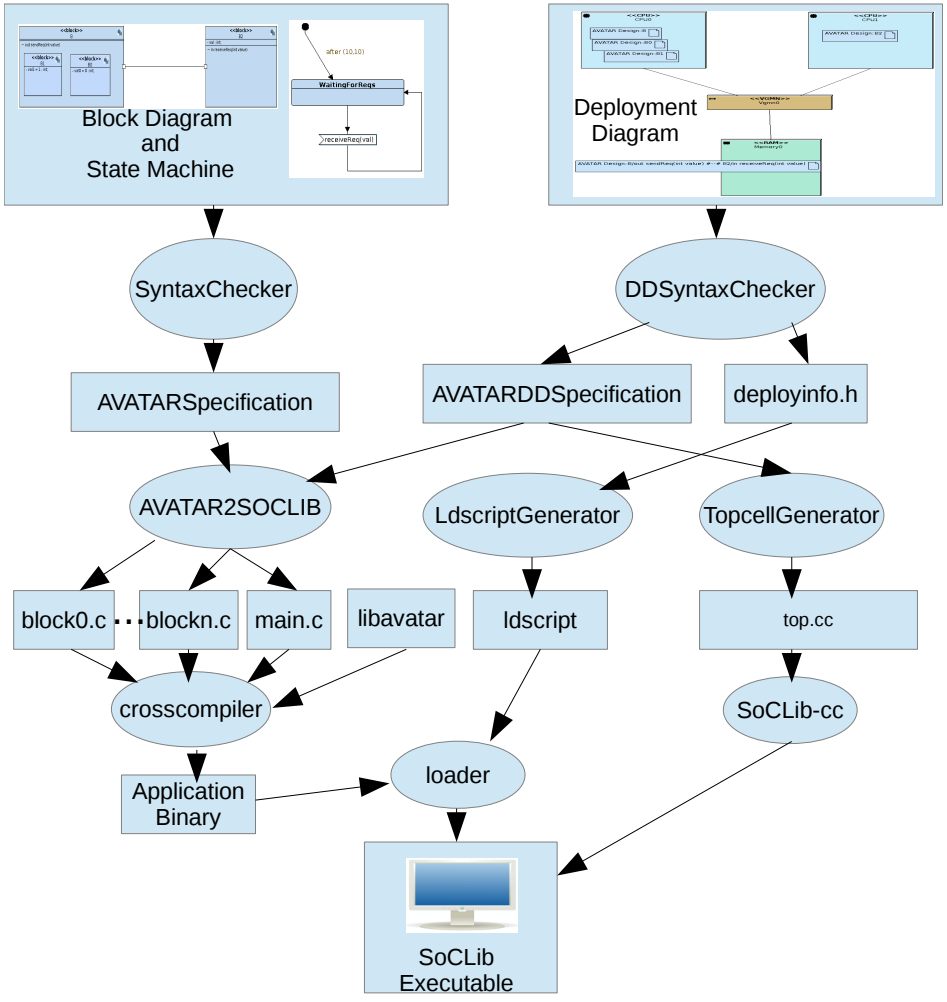
- AVATAR Diagrams (existing)
- Deployment Diagrams (new)
- We do not yet consider security aspects

Generate Code for MPSoC platform

- Tasks and Main
- Ldscript
- Topcell

Compilation Tool Chain

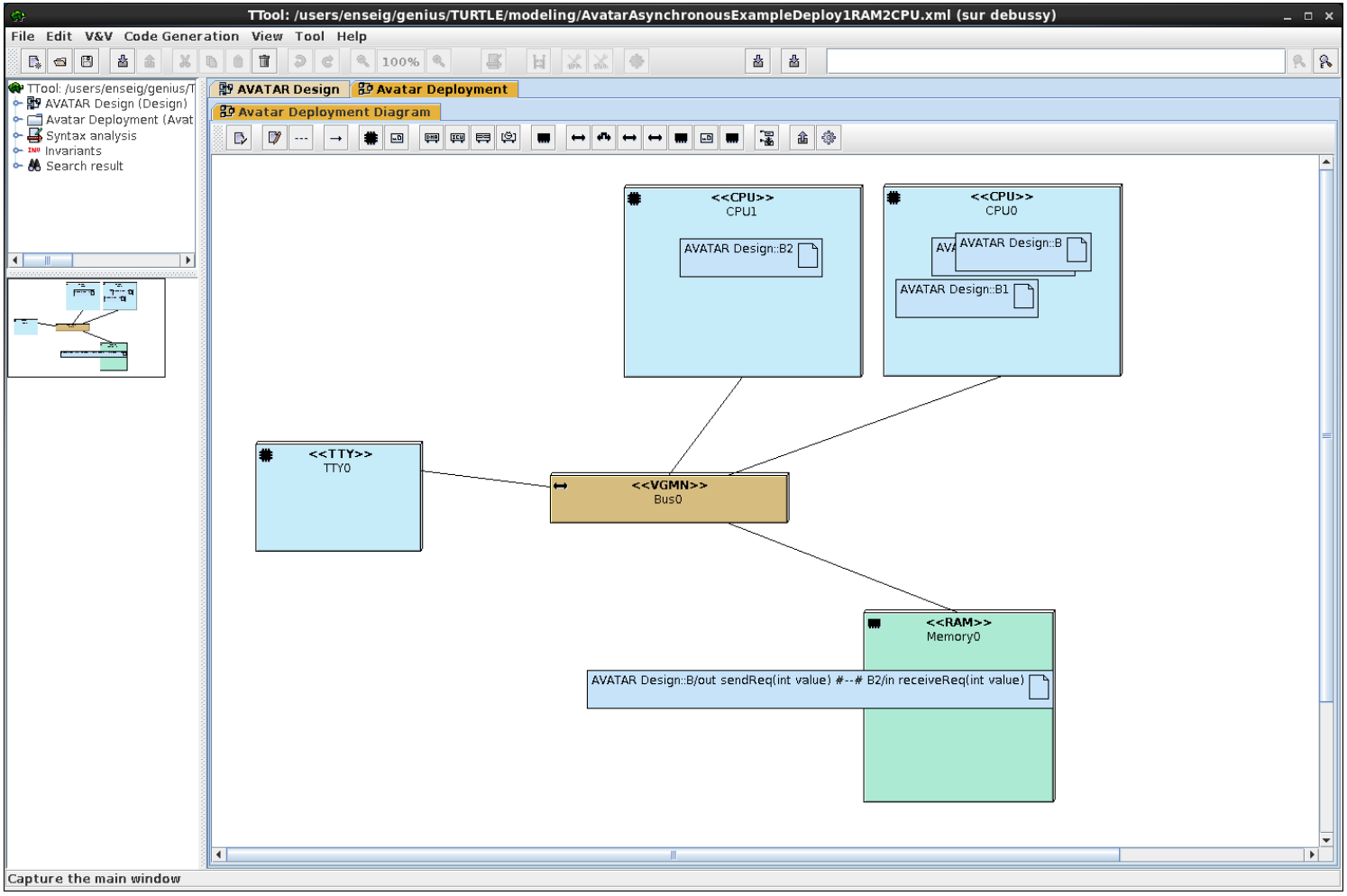
1. **DDSyntaxChecker** checks syntax of deployment diagrams and identifies their elements
2. **Libavatar** Runtime for SoCLib, implements AVATAR operators
3. **AVATAR2SOCLIB** translates AVATAR blocks into C POSIX tasks, generates main program
4. **TopcellGenerator** generates SystemC top cell
5. **LdscriptGenerator** generates linker script



Deployment Diagrams

- SysML representation of hardware components, their interconnection, tasks and channels
- A valid platform must contain at least one CPU, one memory bank and one terminal for observing the progress
- Some details (interrupt controller, simulation aides) currently left transparent to the user

Deployment Diagrams Screen



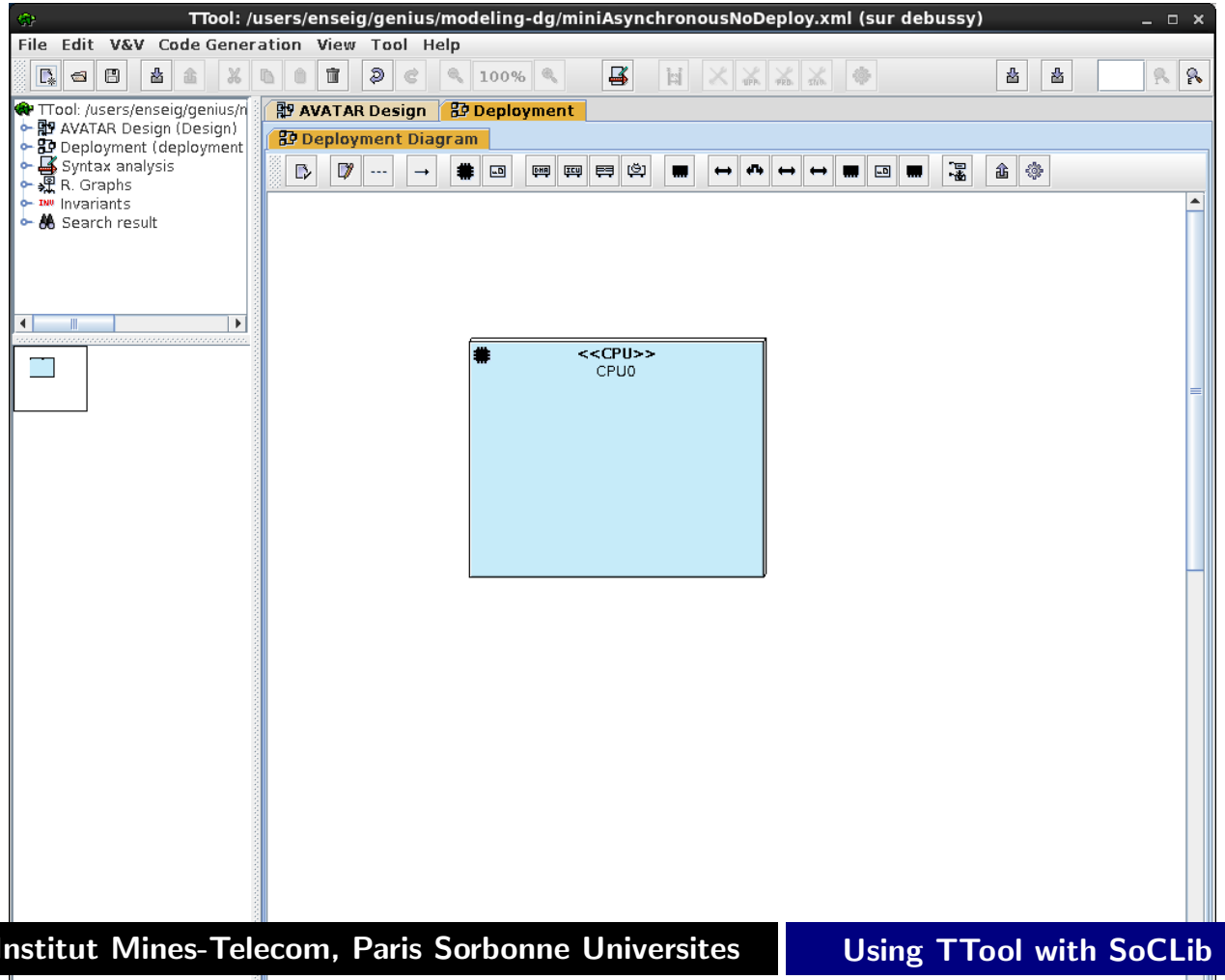
Deployment Diagrams Toolbar

From left to right:

- Edit
- Comment
- UML connector
- Link two nodes
- Add a CPU
- Map task to CPU
- DMA: not yet implemented
- ICU: currently added automatically in all platforms
- COPROCESSOR: not yet implemented
- TIMER: not yet implemented
- TTY
- BUS
- BRIDGE: not yet implemented
- VGMN
- CROSSBAR
- RAM
- Map channel to RAM : map channels onto memory banks
- ROM
- Show/hide attributes
- Extract attributes
- Generate code

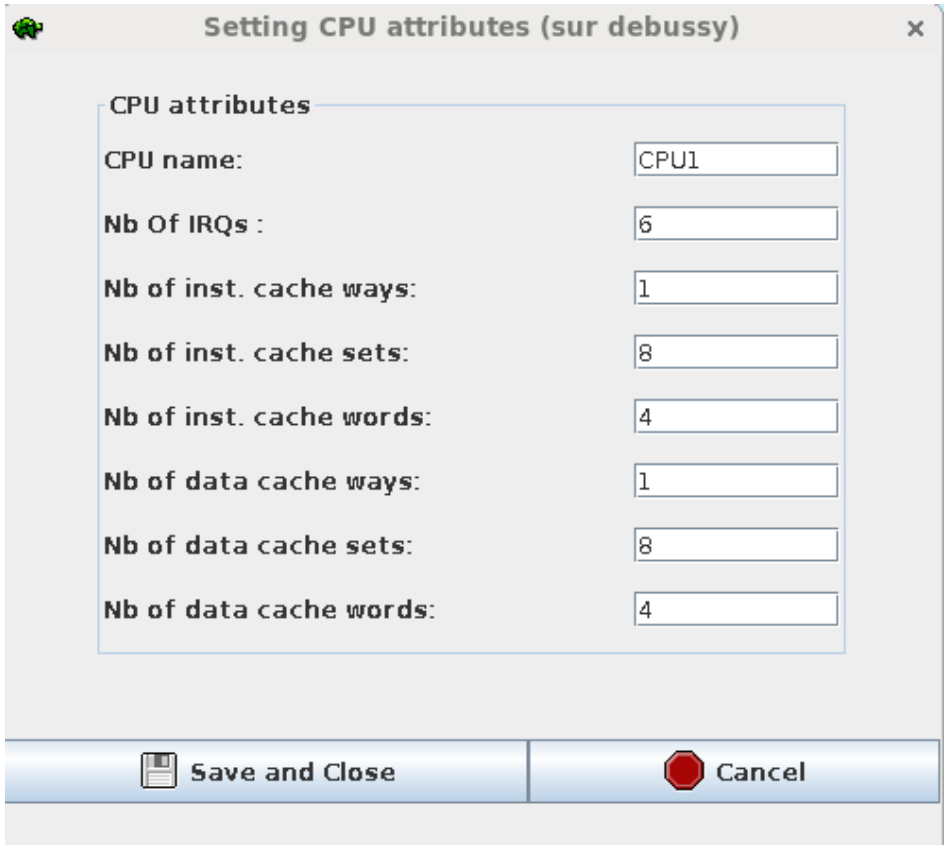
Add a Hardware Module

- Left click on the desired module on the toolbar, then click on screen.
- In the example, we add a CPU.



Modify Hardware Attributes

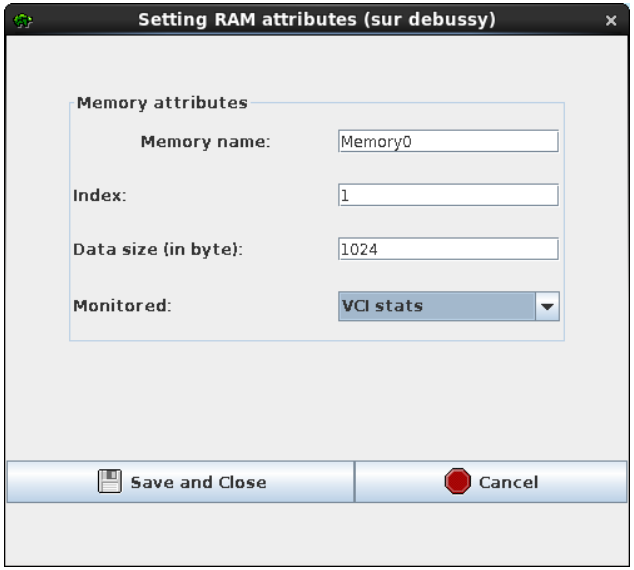
- Right click on components then select **edit** allows to modify attributes : e.g. specify number of cache lines, the size of memory bank



Editing hardware parameters

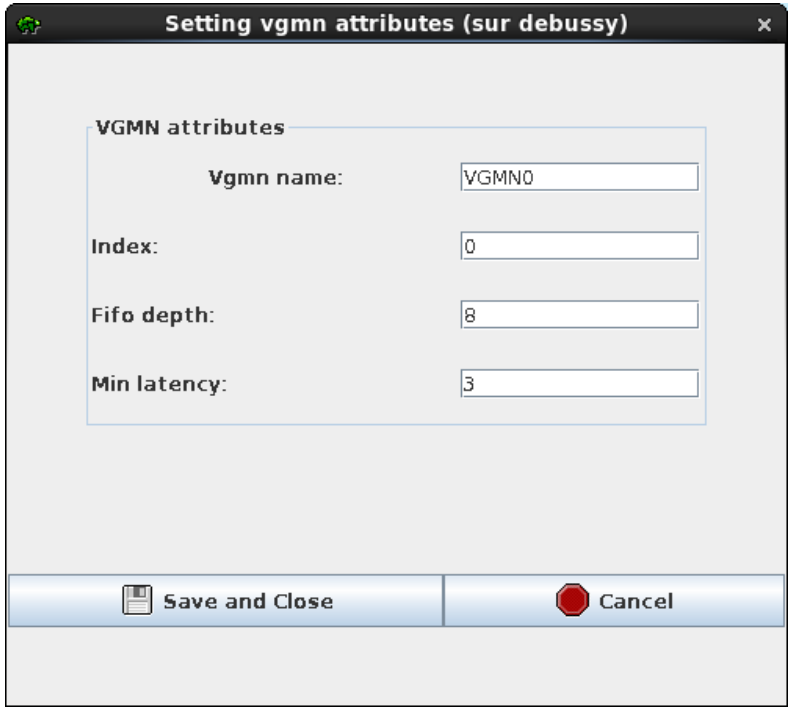
Modify Hardware Attributes (cont.)

- For a RAM, you have to specify the size, which is 0 per default
- You also have to determine in a menu one among three choices of monitoring
  - VCD trace (default): generates a VCD trace that can be viewed e.g. with gtkwave. File `mytrace.vcd`.
  - VCI Logger: monitors all traffic on the VCI (attention traces are very big). Output on tty
  - VCI stats: extraction tool based on the VCI logger that traces actions on channels. File `mwmr.log`.



Modify Hardware Attributes (cont.)

- For a VGMN interconnect, you have to set the latency to at least 3
- The number of initiators and targets can be set, but is currently calculated automatically (as we have modules, interrupt and simulation infrastructure added transparently)



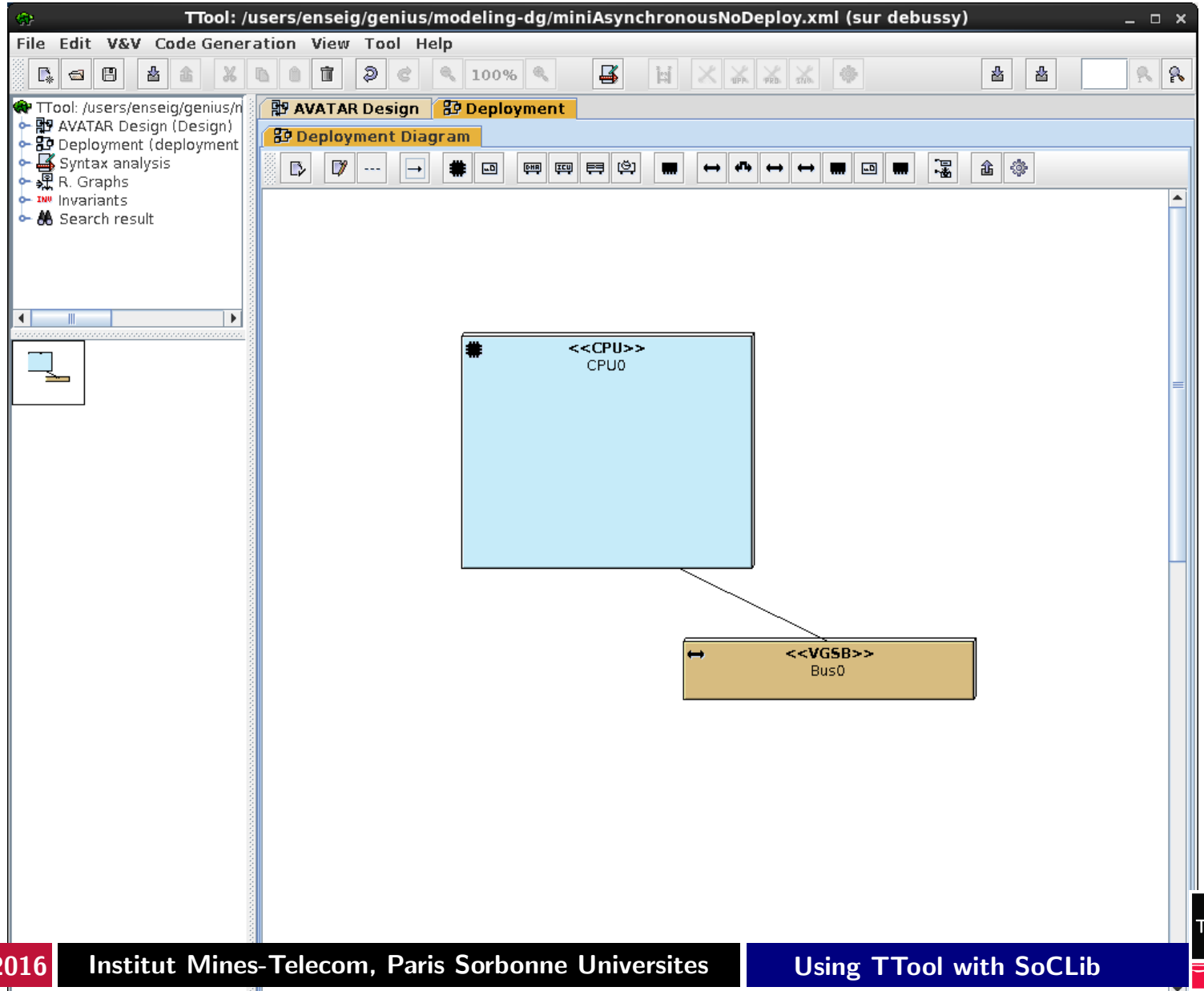
Editing hardware parameters

Connect Hardware Modules

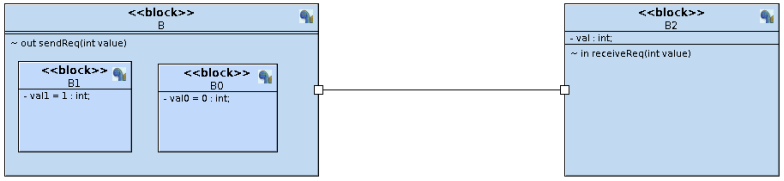
We added a second module (an interconnect). Now we wish to connect the two. For this :

- Left click on arrow in the tool bar.
- Left click on starting point which becomes embossed (a module has 16 connecting points around its rim)
- Left click on ending point point which becomes embossed when the cursor approaches.
- In the example, we connected a Bus and the CPU.
- Important: you have to chose the other module first, the bus or interconnect second.

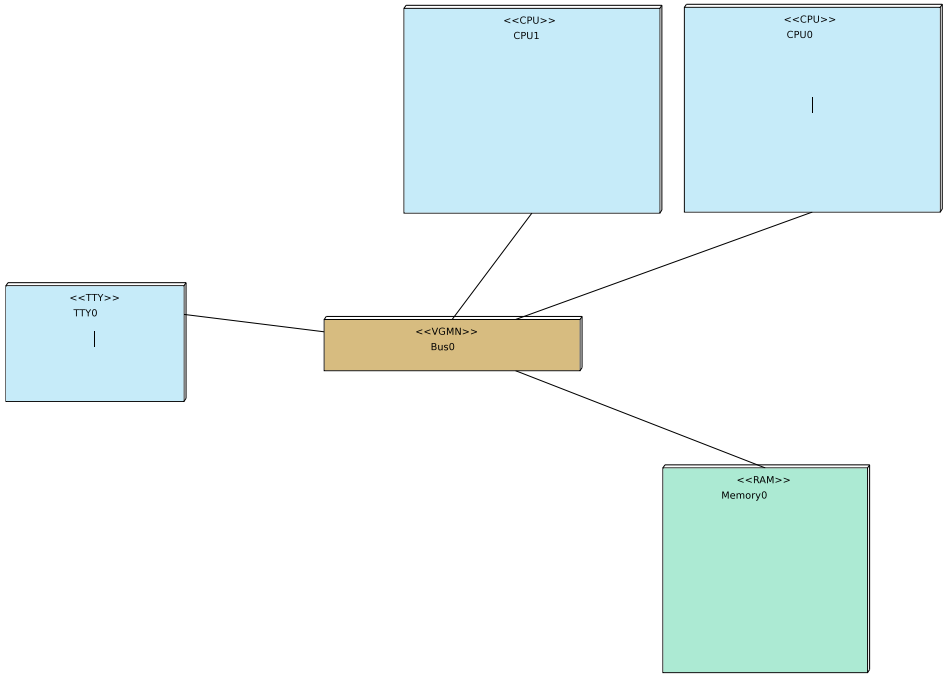
Connect Hardware Modules (Cont.)



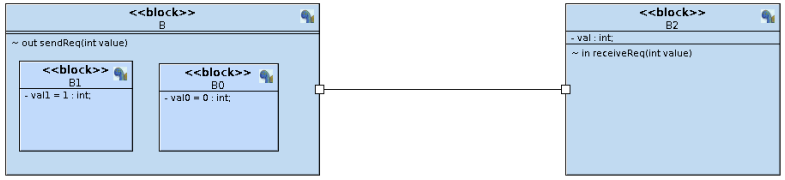
Example

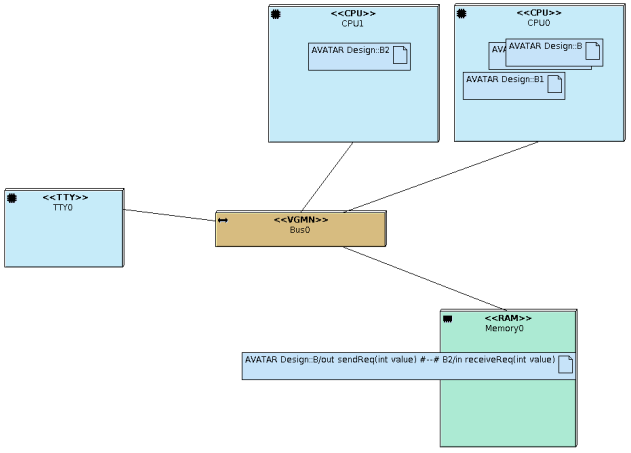


AVATAR example
featuring 4 blocks,
two of which nested,
and one channel



Example





To map tasks on processors, click on task icon  . Then, select Edit and choose the name of the task.

To map channels on memory banks, click on channel icon  (which looks the same but is located more to the right of the toolbar). Then, select Edit and choose the name of the channel. There will be a warning on the terminal if not all tasks or channels are mapped, listing those missing.

The AVATAR Runtime

Library of functions which capture the semantics of the AVATAR operators that appear in the code of the tasks implemented using MutekH `http://www.mutekh.org` primitives

- Directory `TURTLE/MPSoC/src` contains the runtime
- `Makefile.forsoclib` with targets
  - `updategeneratedcode`: generated code copied into MPSoC/mutekh/examples/avatar
  - `updateruntime`: runtime copied into MPSoC/mutekh/libavatar

The AVATAR Runtime (2)

Implements the semantics of AVATAR operators

Operator	**Semantics**
Asynchronous read	Blocking
Asynchronous write	Blocking or non blocking
Synchronous read/write	Blocking
Delay [min..max]	Processor is suspended
Complexity [min..max]	Active execution
Test	Depends on Boolean condition
Non deterministic choice	The first message in the queues triggers the transition or a branch is randomly taken

The AVATAR Runtime (3)

- `asyncchannel`: SoCLib implementation of asynchronous channel
- `debug`: debug functions
- `message`: describes one message
- `myerrors`: error treatment
- `mytimelib`: time functions library
- `random`: ramdom functions
- `request`: describes one request
- `request_manager`: central manager of requests
- `syncchannel`: SoCLib implementation of synchronous channel
- `tracemanager`: managing traces (cycle precise traces not yet implemented)

Code Generation

Generation of MPSoC code in TURTLE/MPSoC directory's subdirectories

- `generated_topcell`: contains three files:
  - `top.cc`: generated topcell, to be copied into `MPSoC/soclib/soclib/platform/topcells/caba-vgmn-mutekh_kernel_tutorial/`
  - `nbproc`: number of processors extracted from Deployment Diagram used for ldscript generation
  - `deployinfo.h`: information on channel mapping extracted from Deployment Diagram used for ldscript generation
- `generated_src`: code for tasks and main program, to be copied into `MPSoC/mutekh/exaples/avatar`

SystemC Top Cell Generation

The topcell is generated and contains several parts, liste din alphabetical order:

- Code.java: some fixed code (gdb call etc)
- Declaration.java: Declaration of hardware components
- Header.java: some fixed code
- Loader.java: call to the loader
- MappingTable.java: declaration of the memory segments (shared memory architecture)
- NetList.java: netlist, connecting signals to ports
- Signal.java: declaration of signals used in netlist
- Simulation.java: call to simulation engine
- TopCellGenerator.java: main class

SystemC Top Cell Generation (2)

The topcell is generated and contains several parts, each corresponding to part of its code. Example lines from a generated topcell:

- Instanciation of components:

```
soclib::caba::VciRam<vci_param>Memory0("Memory0", IntTab(2), maptab);
```

- memory segments:

```
maptab.add(Segment("data", 0x7f000000, 0x01000000, IntTab(2), false));
```

- charging of code:

```
data_ldr.load_file(std::string(kernel_p) +
  ";.data;.channel0;.cpudata;.contextdata");
```

- Netlist:

```
vgsb.p_to_target[2](signal_vci_vciram0);
```

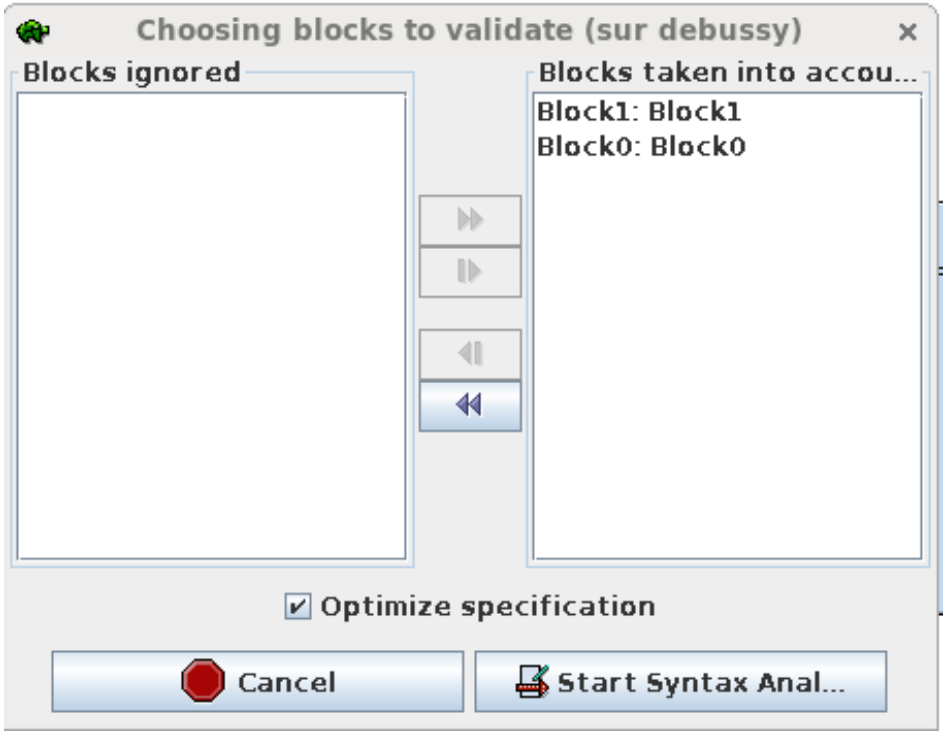
Linker Script

- Defines the memory layout
- Associates each entry section to an output section
- /MPSoC/mutekh/arch/soclib contains generator
- copied into /MPSoC/mutekh/obj-avatar-soclib-ppc/arch
- implements the mapping of channels by forcing them on the memory bank specified in the Deployment Diagram

```
MEMORY
{
     mem_ram (RWAL): ORIGIN = 0x7f000000, LENGTH = 0x01000000
...
}
SECTIONS
{
...
 .data : {
  __data_start = ABSOLUTE(.);
  *(.sdata*)
  *(.data*)
  *(.cpuarchdata*)
 } > mem_ram AT>mem_rom
 .channel0 : { *(section_channel0)} > mwmrd_ram
...
}
```

Syntax Checking

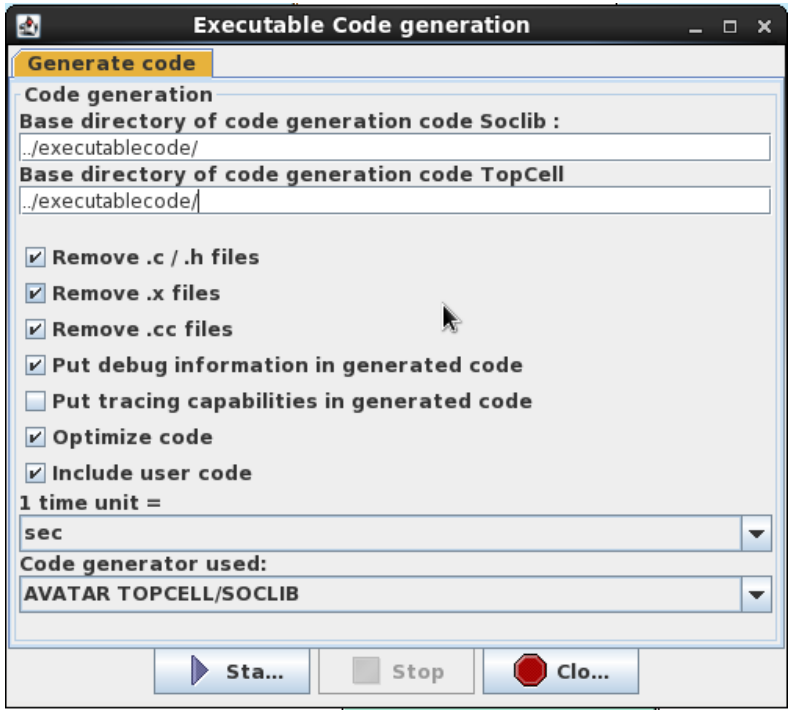
Click on the `AVATAR Design` tab (this cannot be invoked from the `AVATAR Deployment` tab)



You should get a message indicating there are no errors; otherwise refer to the general TTool documentation.

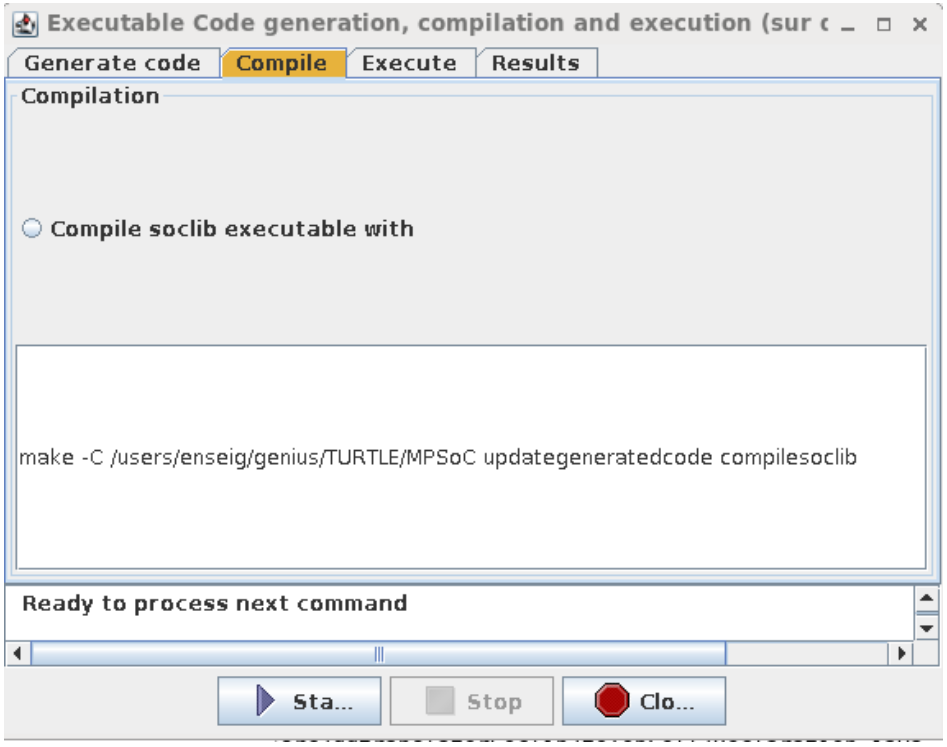
Code Generation Dialogue

- Click on AvatarDeployment (right tab) Click on the gear at the right of the lower toolbar (Deployment Diagram Toolbar)
- Select trace and:or debug mode if required
- Select **Compile soclib executable** in the compile menu
- Select **Run code in soclib/mutekh** in the **Execute** menu



Compilation Dialogue

Normally, this is done by pushing the `Compile soclib executable` button in the menu below.



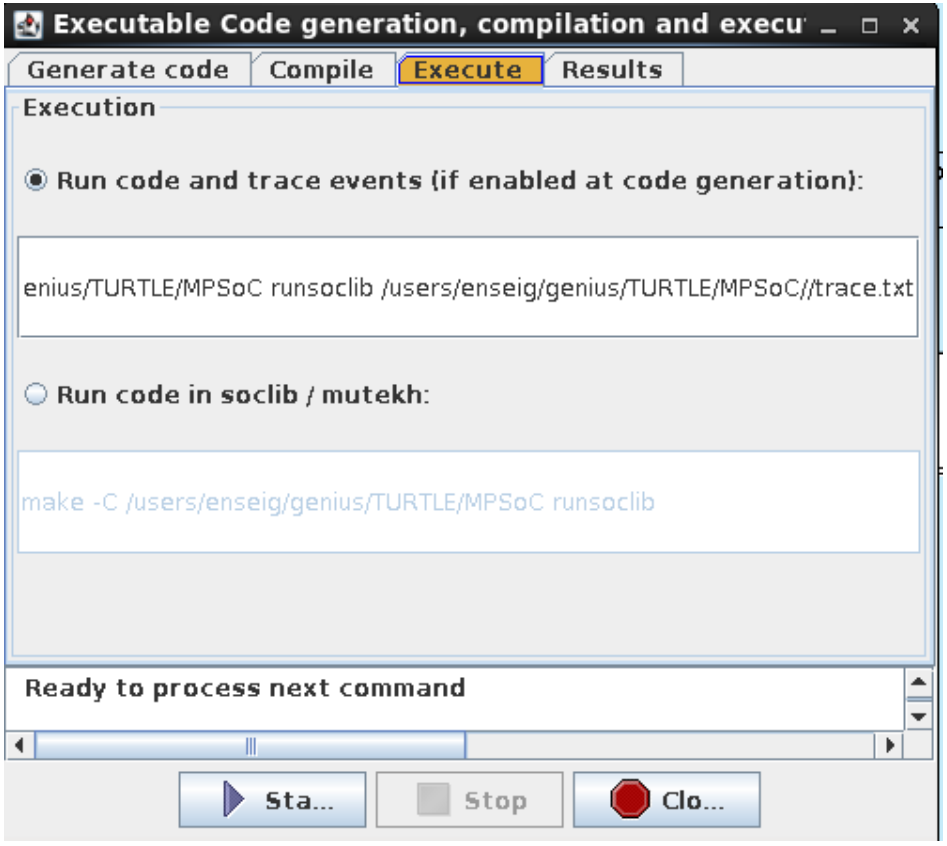
Compilation (2)

Alternatively, for test and debug purposes, you may wish to manually modify the code of task and main files. The code is copied into the following directory:
`/MPSoC/mutekh/examples/avatar`
Type `cd /MPSoC/mutekh;`
`make CONF=examples/avatar/config BUILD=soclib-$(MUTEKH_CPU):pf-tutorial`
Where `MUTEKH_CPU` is your CPU (currently ppc)

Be careful when you modify generated code, your changes will be lost at next code generation!

Simulation Dialogue

Normally, this is done by pushing the `Run code in soclib/mutekh button` in the menu (cycle accurate trace facility will be added later on)



Simulation (2)

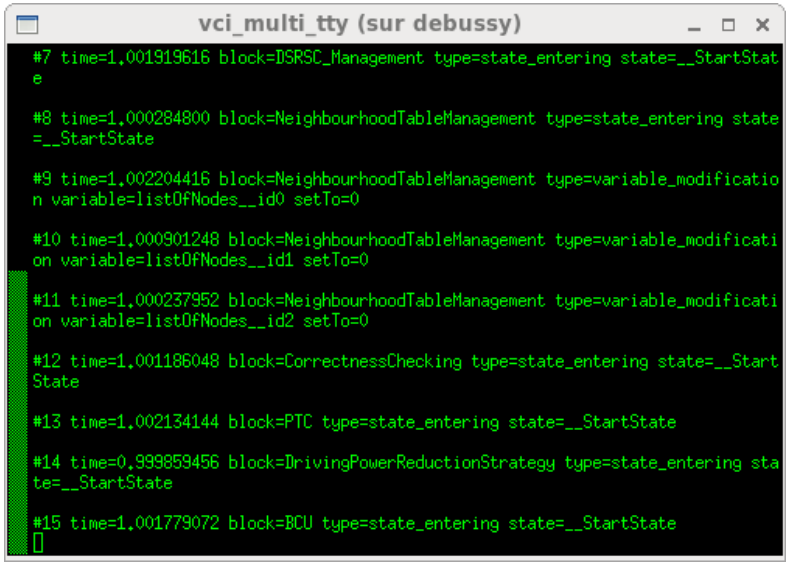
Alternatively, for test and debug, you may wish to manually modify the topcell. in this case, go to the following directory:
`MPSoC/soclib/soclib/platform/topcells/caba-vgmn-mutekh_kernel_tutorial/`
Eventually modify `top.cc`
Type `make`
Then start the platform executable

```
./system.x $(SOCLIB_CPU):$(SOCLIB_CPU_COUNT)
 /MPSoC/mutekh/avatar-soclib-$(MUTEKH_CPU).out
```

Where `SOCLIB_CPU_COUNT` has to correspond to the number of CPUs used in your Deployment Diagram and `MUTEKH_CPU` is the type of CPU (currently ppc)

Invoking SoCLib from TTool

A TTY (green on black) should open; if the debug option was selected, the application progress can be watched and read/write operations on channels can be monitored



SoCLib simulation window

```
          SystemC 2.2.0 --- Jun  8 2009 14:24:30
       Copyright (c) 1996-2006 by all Contributors
                   ALL RIGHTS RESERVED
Initializing memories with 5a
caba-vgmn-mutekh_kernel_tutorial SoCLib simulator for MutekH
Initializing memories with 5a
Initializing memories with 5a
***0
[MemChecker] SOCLIB_MEMCHK env variable may contain the following flag letters:
  R (show region changes),     C (show context ops),  S (show context switch),
  T (raise gdb except on err), I (show iss dump),     A (show access details),
  L (show locks accesses),     E (show checks enable) X (exit simulation on err)
  => See http://www.soclib.fr/trac/dev/wiki/Tools/MemoryChecker
[GDB] SOCLIB_GDB env variable may contain the following flag letters:
  X (dont break on except),      S (wait connect on except),  F (start frozen)
  C (functions branch trace),    Z (functions entry trace),   D (gdb protocol debug),
  W (dont break on watchpoints), T (exit sumilation on trap), E (exit on fault)
  => See http://www.soclib.fr/trac/dev/wiki/Tools/GdbServer
[GDB] listening on port 2350
***1
  - Building VciXicu : vci_xicu
    => segment vci_xicu / base = d2200000 / size = 1000

Warning: (W505) object already exists: vcirttimer.ctrl. Latter declaration will be renamed to vcirttimer.ctrl_0
In file: sc_object.cpp:196
  - Building VciMultiTTy TTY0
    => segment vci_multi_tty / base = d0200000 / size = 10
vcieth: Unable to setup tap interface, check privileges. (try: sudo setcap cap_net_admin=eip ./system.x)
Loading at 0x60000000 size 0x100000: .text .rodata .excep .cpudata .contextdata .data
Loading at 0x60000000 size 0x100000: .text .rodata .excep .cpudata .contextdata .data
Loading at 0x60000000 size 0x100000: .text .rodata .excep .cpudata .contextdata .data
Loading at 0xffffff80 size 0x80: .boot
Loading at 0x802000 size 0x1000: nothing
Loading at 0 size 0x1000: nothing
Loading at 0xbfc00000 size 0x1000: nothing
Loading at 0x80000000 size 0x1000000: nothing
Loading at 0x30200000 size 0x100: nothing
Loading at 0xa0200000 size 0x1000: nothing
Loading at 0x7f000000 size 0x1000000: nothing
```

Invoking SoCLib simulation

Upcoming Extensions

- We will shortly supply a virtual machine for Windows
- Adaptation to SysML-Sec
  `http://sysml-sec.telecom-paristech.fr/`
- Debug mode: very similar to POSIX version
- Trace mode: cycle accurate traces of CABA Simulation
  (proposed as a M2 project see
  `http://www-soc.lip6.fr/offres-demplois/stages/`)

Web Site and Contacts

`http://ttool.telecom-paristech.fr`

or

`https://www-soc.lip6.fr/trac/avatarsoclib`

Contact daniela.genius@lip6.fr,
ludovic.apvrille@telecom-paristech.fr