



Institut
Mines-Telecom



Paris Sorbonne
University

Using TTool with SoCLib

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

Tutorial to the
SoCLib extension
of **TTool/AVATAR**



Requirements for Linux and MacOSX

1. TTool <http://ttool.telecom-paristech.fr>
2. The Prog arborescence containing among others MutekH sources and a SoCLib version available from <http://ttool.telecom-paristech.fr> or <https://www-soc.lip6.fr/trac/avatarsoclib>
3. Your system has to feature a gcc compiler and a gdb
 - ▶ Linux : gcc 4.7. was tested with success
 - ▶ MacOSX : there are some difficulties with the Xcode compiler, we recommend the homebrew gcc-4.8 compiler
4. A valid SystemC compiler to compile the SoCLib platform: accelera.org/downloads/standards/systemc
 - ▶ Linux : systemc-2.2.0 was tested
 - ▶ MacOSX : systemc-2.3.1 from Accellera was tested with success; here is the link : accelera.org/downloads/standards/systemc

Alternatives and Add-ons

- ▶ Alternatively, SystemCASS [2] is a simulation engine from Lip6 for cycle accurate level which is 10 to 20 times faster than SystemC event-based simulation:

<https://www-soc.lip6.fr/trac/systemcass>

To be installed under /opt

Some tools cannot be distributed by us. If you wish to

- ▶ Make proofs of safety properties from AVATAR models, you need to install and configure UPPAAL. Here is some help :
http://ttool.telecom-paristech.fr/installation_companion.html#uppaal
- ▶ Make proofs of security properties from AVATAR models, you need to install and configure ProVerif. Here is some help :
http://ttool.telecom-paristech.fr/installation_companion.html#proverif

Generating the Crosscompilers

Prog/crossgen.mk fetches and generates the crosscompilers. The crosscompiler will serve to compile your application (task and main file produced by TTool) for the desired architecture

To generate your crosscompilers, you have to execute
./crossgen.mk which is part of MutekH and found in the
Prog/mutekh directory This is usually done by typing
./crossgen.mk all

To be installed per default under /opt/mutekh

Available targets for crosscompiler generation:

mipsel, powerpc, arm, i686, x86_64, nios2, sparc,
avr, lm32, microblaze, avr32



Processor Cores Available

- ▶ Currently the complete toolchain is only available for PowerPC, but it is easy to add other CPU as MutekH allows heterogeneous processors
- ▶ The topcell currently can be used with ISS for the following processor cores:
 - ▶ PowerPc 405
 - ▶ Nios II
 - ▶ Mips 32
 - ▶ Arm 7
 - ▶ Sparc v8
 - ▶ LM 32
- ▶ The mapping table generation and calculation of addresses in `/src/ddtranslatorSoclib/toTopcell` will have to be extended (requests to daniela.genius@lip6.fr)

Additions to the TTool Arborescence

Generation of POSIX code for local workstation in directory
TURTLE/executablecode [1]

Generation of code for MPSoC in directory TURTLE/MPSoC
containing the following subdirectories:

- ▶ generated_src: generated task code for AVATAR blocks and main code spawning the POSIX threads
- ▶ src: the runtime for MPSoC platforms
- ▶ generated_topcell: topcell and mapping information to generate the Idscript

Additions to the TTool Arborescence(2)

Central Makefile `src/MPSoC/Makefile.forsoclib` which copies generated code from the TURTLE into the Prog arborescence as follows:

- ▶ `generated_src` directory:
 - ▶ `*.c` and `*.h` copied to `Prog/mutekh/examples/avatar`
 - ▶ `src_soclib` to `Prog/mutekh/libavatar`
- ▶ `generated_topcell` directory:
 - ▶ `top.cc` to `/Prog/soclib/soclib/platform/topcells/caba-vgm-mutekh_kernel_tutorial/top.cc`
 - ▶ `deployinfo.h` and `deployinfo_map.h` to `/Prog/mutekh/arc/soclib`

they are used by `ldscript.cpp`, a preprocessor generating the `ldscript`

Additions to the TTool Arborescence(3)

The src/ddtranslatorSoclib directory

- ▶ contains the code for analyzing the deployment diagrams
- ▶ contains subdirectories toSoclib and toTopCell generating task/main code and topcell/information for the Idscript and main, respectively

In TURTLE/bin/config.xml add the following two lines :

- ▶ `<AVATARMPSocCodeDirectory data=" /TURTLE/MPSoc/" />`
- ▶ `<AVATARMPSocCompileCommand data="make -C /TURTLE/MPSoc updategeneratedcode compilesoclib" />`

Make sure that you use the script ./ttool.exe containing the line

```
java -Xmx1024m -Djavax.net.ssl.trustStore=ServerKeyStore
-Djavax.net.ssl.trustStorePassword=123456 -jar ttool.jar
-config config.xml -experimental -debug -avatar -uppaal
-launcher
```


Virtual Prototyping of AVATAR Diagrams

Diagram Editor

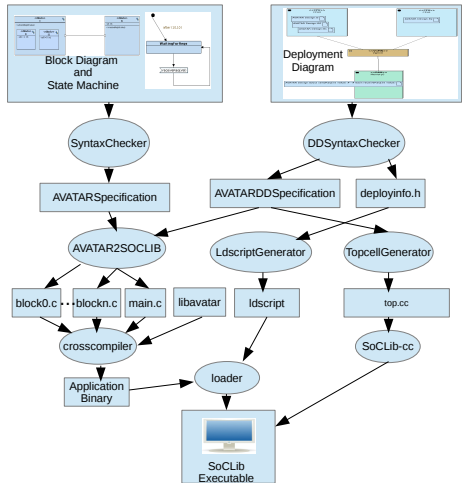
- ▶ AVATAR [4] Diagrams (existing)
- ▶ Deployment Diagrams [3] (new)
- ▶ We do not yet consider security aspects

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

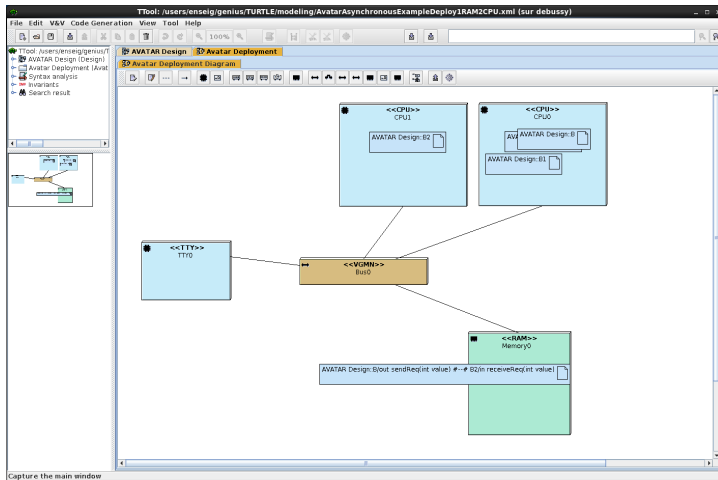




Deployment Diagrams

- ▶ 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

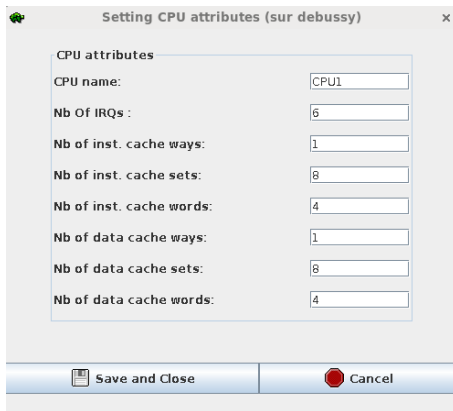


From left to right:

- ▶ 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
- ▶ VGMMN
- ▶ CROSSBAR
- ▶ RAM
- ▶ Map channel to RAM : map channels onto memory banks
- ▶ ROM
- ▶ Show/hide attributes
- ▶ Extract attributes
- ▶ Generate code

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

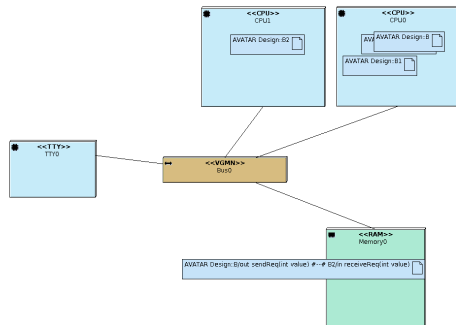


Editing hardware parameters

Example



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



Mapping on
two processors
and two RAM



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 Prog/mutekh/examples/avatar
 - ▶ updatruntime: runtime copied into Prog/mutekh/libavatar

The AVATAR Runtime (2)

Implemented 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`: random functions
- ▶ `request`: describes one request
- ▶ `request_manager`: central manager of requests
- ▶ `syncchannel`: SoCLib implementation of asynchronous 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 `Prog/soclib/soclib/platform/topcells/caba-vgm-mutekh_kernel_tutorial/`
 - ▶ `nbproc`: number of processors extracted from Deployment Diagram used for Idscript generation
 - ▶ `deployinfo.h`: information on channel mapping extracted from Deployment Diagram used for Idscript generation
- ▶ `generated_src`: code for tasks and main program, to be copied into `Prog/mutekh/exaples/avatar`

SystemC Top Cell Generation

The topcell is generated and contains several parts, listed in 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:

- Instantiation 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);
```

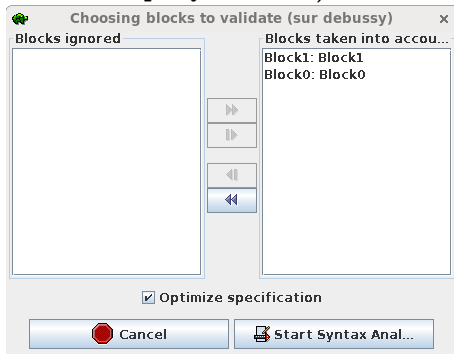
Linker Script

- ▶ Defines the memory layout
- ▶ Associates each entry section to an output section
- ▶ /Prog/mutekh/arch/soclib contains generator
- ▶ copied into /Prog/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

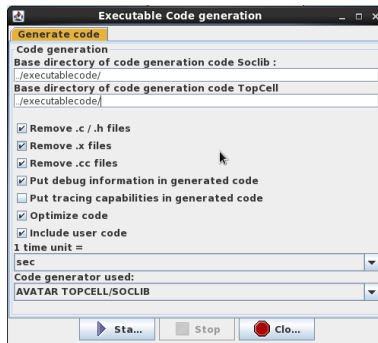
Click on the AVATAR Design tab (this cannot be invoked from the AVATAR Deployment tab)



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

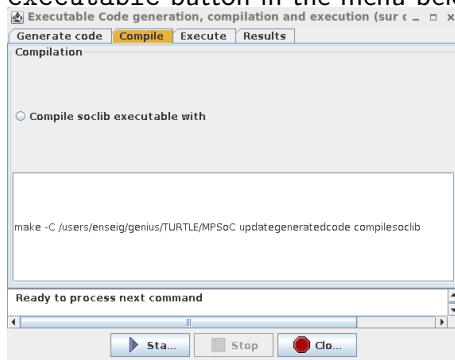
Code Generation Dialogue

- ▶ 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:

```
/Prog/mutekh/examples/avatar
```

```
Type cd /Prog/mutekh;
```

```
make CONF=examples/avatar/config
```

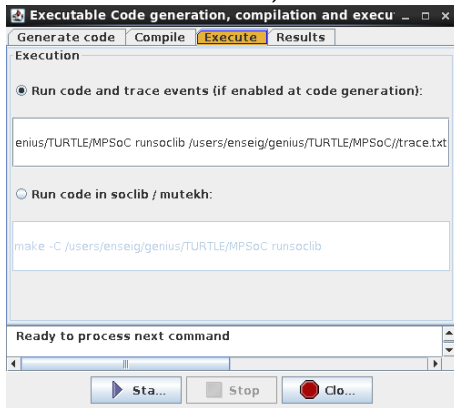
```
BUILD=soclib-$(MUTEKH_CPU):pf-tutorial
```

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:

```
Prog/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)  
/Prog/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

```
vci_multi_tty (sur debussy)

#7 times1,001919616 block=DGRSC_Management type=state_entering state=__StartState
#8 times1,000284800 block=NeighbourhoodTableManagement type=state_entering state=__StartState
#9 times1,000204416 block=NeighbourhoodTableManagement type=variable_modification variable=listOfNodes_id0 setTo=0
#10 times1,000901248 block=NeighbourhoodTableManagement type=variable_modification variable=listOfNodes_id1 setTo=0
#11 times1,000237952 block=NeighbourhoodTableManagement type=variable_modification variable=listOfNodes_id2 setTo=0
#12 times1,001185048 block=CorrectnessChecking type=state_entering state=__StartState
#13 times1,002134144 block=PTC type=state_entering state=__StartState
#14 times0,999859456 block=DrivingPowerReductionStrategy type=state_entering state=__StartState
#15 times1,001779072 block=BCU type=state_entering state=__StartState
[]
```

SoCLib simulation window

```
SystemC 2.2.0 --- Jun 8 2009 14:24:38
Copyright (c) 1996-2006 by all Contributors
ALL RIGHTS RESERVED

Initializing memories with 5a
caba-vgnm-mutekh kernel tutorial SoCLib simulator for Mutekh
Initializing memories with 5a
Initializing memories with 5a
***q
[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 simulation on trap), E (exit on fault)
=> See http://www.soclib.fr/trac/dev/wiki/Tools/GdbServer
[GDB] Listening on port 2350
***i
- Building VciXicu : vci_xicu
=> segment vci_xicu / base = d2200000 / size = 1000

Warning: (MS05) object already exists: vcirtimer.ctrl. Latter declaration will be renamed to vcirtimer.ctrl_0
In file: sc_object.cpp:196
- Building VciMultiTty TTY0
=> segment vci_multi_tty / base = d0200000 / size = 10
vcirtimer: Unable to Setup Tap interface, check privileges. (try: sudo setcap cap_net_admin=elp ./system.x)
Loading at 0x60000000 size 0x100000: text .rodata .except .cpudata .contextdata .data
Loading at 0x60000000 size 0x100000: text .rodata .except .cpudata .contextdata .data
Loading at 0xfffff800 size 0x800: .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`



L. Apvrille and A. Becoulet.

Prototyping an embedded automotive system from its UML/SysML models.

In *ERTSS'2012*, Toulouse, Feb. 2012.



R. Buchmann and A. Greiner.

A fully static scheduling approach for fast cycle accurate SystemC simulation of MPSoCs.

In *Proc. ICEEC*, pages 35–39, Cairo, Egypt, Dec. 2007. IEEE.



D. Genius and L. Apvrille.

Virtual yet precise prototyping : An automotive case study.

In *ERTSS'2016*, Toulouse, Jan. 2016.



G. Pedroza, D. Knorreck, and L. Apvrille.

AVATAR: A SysML environment for the formal verification of safety and security properties.

In *The 11th IEEE Conference on Distributed Systems and New Technologies (NOTERE'2011)*, Paris, France, May 2011.