# TTool-AMS

Integration of SystemC-AMS Simulation Platforms into TTool

Author: Rodrigo Cortés Porto (University of Kaiserslautern)

Co-Authors: Dr. D. Genius (Sorbonne University, LIP6)

Irina Lee (Sorbonne University, Master SESI)

Prof. L. Apvrille (Télécom Paris)

## Contents

# 1 Getting started

## 1.1 Virtual Machine

In the *Virtual Machine* version, tested under VirtualBox, TTool, as well as SoClib and MutekH are preinstalled. You will require to fetch SystemC-AMS from the Accelera website, which requires registration:

`www.accellera.org`

Figure 1.1 shows the relevant part of the Virtual Machine directory tree. The Virtual Machine uses bash and automatically configures all paths on invoking the script `systemc-env.sh`.

```
$HOME/TTool/
  MPSoC/
  SysCAMSGeneratedCode/
bin/
  systemc-env.sh
  .soclib/
    global.conf
.bashrc
```

**Listing 1.1:** Virtual machine directory tree.

## 1.2 Configuration under Linux

The following subsection describes an installation under Linux [1]. Before installing TTool, a `global.config` file should be created under `$HOME/.soclib/`. This file can found in Listing 1.3[2].

In order to use TTool and SystemC-AMS for co-simulation, you need to install SystemC-AMS on your machine. SoCLib and MutekH are provided as part of the TTool free software distribution.

SystemC-AMS is free but requires registration and is available at the following url:

`www.accellera.org`

Then, you require the systemc-env.sh file (by Torsten Maehne) This file can found in Listing 1.4 to set up the paths for compilation [3].

[1]Tested under Scientific Linux 6

[2] The file is provided in the Virtual Machine version.

[3] The file is provided in the Virtual Machine version, in the bin directory.

```python
# -*- python -*-
import os
syscams = os.getenv('SYSTEMC_AMS')
assert syscams, ValueError("Must set $SYSTEMC_AMS")
sysc = os.getenv('SYSTEMC')
assert sysc, ValueError("Must set $SYSTEMC")

config.systemc_ams = Config(
        base = config.systemc,
        cflags = ['-Iinclude',
              '-I'+sysc+'/include',
              '-I'+syscams+'/include'
          ],
        libs = ['-Wl,-rpath='+sysc+'/lib-linux64', '-L'+sysc+'/lib-linux64',
            '-Wl,-rpath='+syscams+'/lib-linux64', '-L'+syscams+'/lib-linux64',
        '-lsystemc-ams', '-lsystemc', '-lm'
        ],
        )
config.ams = Config(
    base = config.default,
    systemc = config.systemc_ams,
    repos = "./obj/soclib-cc",
    )
config.default = config.ams
```

**Listing 1.2:** Code for the `soclib.conf` Configuration file.

```python
   # -*- python -*-

# SOCLIB environment definition
def mkname():
        try:
                import os
                import pwd
                return pwd.getpwuid(os.getuid())[0]
        except OSError:
                try:
                        import os
                        return os.environ["LOGNAME"]
                except KeyError:
                        return 'unknown'

config.toolchain_64 = Toolchain(
      parent        = config.toolchain,
      max_processes = 3,
      cflags        = config.toolchain.cflags+['-m64'],
      )

config.systemc_22_64 = Library(
      parent = config.systemc,
      dir    = "/opt/systemcams/systemc-2.3.1/",
      cflags = config.systemc.cflags,
      os     = "linux64",
      )

config.systemc_ams = Library(
        parent = config.systemc,
        dir    = "/opt/systemcams/systemc-ams-2.1",
        cflags = ['-Iinclude',
              '-I/opt/systemcams/systemc-ams-2.1/include'
          ],
        libs = ['-Wl,-rpath=/opt/systemcams/systemc-2.3.1/lib-linux64',
        '-L/opt/systemcams/systemc-2.3.1/lib-linux64',
        '-Wl,-rpath=/opt/systemcams/systemc-ams-2.1/lib-linux64',
        '-L/opt/systemcams/systemc-ams-2.1/lib-linux64',
        '-lsystemc-ams', '-lsystemc', '-lm'
        ],
        )

# Definition of a new build environments, which can be referenced with 'soclib-cc -t'

# SystemC 64bits environment
config.systemc_64 = BuildEnv(
      parent = config.build_env,
      repos = "/home/%s/tmp/soclib_repos_64"%mkname(),
      toolchain = config.toolchain_64,
      libraries = [config.systemc_22_64, config.systemc_ams],
```

```bash
#!/bin/bash
######################################################################
# systemc-env.sh: Set up the environment for SystemC
######################################################################
# Author:  Torsten MAEHNE <Torsten.Maehne@lip6.fr>
# Created: 2014-01-30
######################################################################
#
# This shell script prepares the shell environment for building or
# using the SystemC libraries. You can either source it from the command
# line or in your .bashrc:
#
# . ./systemc-env.sh [--build-version=2.2|2.3.0|2.3.1]
#                    [--build-type=Debug|Release] \
#                    [-h|--help] \
#                    [-v|--version] \
#
######################################################################

######################################################################
# Print help message
######################################################################
function systemc_environment_help_message() {
    cat <<EOF
This script should be sourced to prepare your environment for SystemC:
. $0 [--build-version=2.2|2.3.0|2.3.1] [--build-type=Debug|Release]
EOF
}

######################################################################
# Print version message
######################################################################
function systemc_environment_version_message() {
    cat <<EOF
systemc-env.sh: Set up the environment for SystemC
Author:  Torsten MAEHNE <Torsten.Maehne@lip6.fr>
Version: unreleased
EOF
}
```

```
###########################################################################
# Function to manipulate the path variable
###########################################################################
path_append ()  { path_remove $1; export PATH="$PATH:$1"; }
path_prepend () { path_remove $1; export PATH="$1:$PATH"; }
path_remove ()  { export PATH=`echo -n $PATH | awk -v RS=: -v ORS=: '$0 != "'$1'"' | sed 's/:$//'`; }

# Default SystemC version (either 2.2, 2.3.0, or 2.3.1)
SYSTEMC_VERSION=2.3.1
# Default SystemC build type (either Release or Debug)
SYSTEMC_BUILD_TYPE=Release
# Process the command line arguments
for i in $*
do
    case $i in
        --build-version=*)
            SYSTEMC_VERSION=${i#*=}
            ;;
            --build-type=*)
            SYSTEMC_BUILD_TYPE=${i#*=}
            ;;
        -h | --help)
            systemc_environment_help_message
            exit 0
            ;;
        -v | --version)
            systemc_environment_version_message
            exit 0
            ;;
            *)
            # unknown option
            echo "Error: Unknown option \"$i\"" 1>&2
            exit 1
            ;;
    esac
done

case $SYSTEMC_BUILD_TYPE in
    Debug)
        SYSTEMC_BUILD_SUFFIX=dbg
    ;;
    Release)
        SYSTEMC_BUILD_SUFFIX=rel
    ;;
    *)
        echo "Error: Unknown build type \"$BUILD_TYPE\". Use either \"Debug\’’ or \"Release\"." 1>&2
esac

case $SYSTEMC_VERSION in
    2.3.1)
        export BOOST_ROOT=/usr
        export SYSTEMC_ARCH=linux64
        export TARGET_ARCH=
             export SYSTEMC_ROOT=/opt/systemcams/systemc-2.3.1
        export SYSTEMC_HOME=$SYSTEMC_ROOT
             export SYSTEMC_AMS_HOME=/opt/systemcams/systemc-ams-2.1
              export SYSTEMC_INCLUDE_DIRS="-I$SYSTEMC_ROOT/include -I$SYSTEMC_AMS_HOME/include"
        export SCV_HOME=$SYSTEMC_ROOT
        export TLM_HOME=$SYSTEMC_ROOT
        export SYSTEMC_LIBRARY_DIRS="-Wl,-rpath=$SYSTEMC_HOME/lib-$SYSTEMC_ARCH
        -L$SYSTEMC_HOME/lib-$SYSTEMC_ARCH -Wl,-rpath=$SYSTEMC_AMS_HOME/lib-$SYSTEMC_ARCH
        -L$SYSTEMC_AMS_HOME/lib-$SYSTEMC_ARCH"
            export PATH=$PATH:/opt/soclib/utils/bin:/opt/mipsel-unknown-elf/bin
        ;;
    *)
        echo "Error: Unknown SystemC version \"$SYSTEMC_VERSION\". 1>&2
        ;;
esac

export SYSTEMC=$SYSTEMC_HOME
export TLM2=$TLM_HOME
export SYSTEMC_AMS=$SYSTEMC_AMS_HOME
```

**Listing 1.4:** Code for `systemc-env.sh`

Finally, you need to work under bash and set up your `.bashrc` as follows. As an example, the `.bashrc` file in the Virtual machine looks as follows, you might have to adapt it to your system configuration (Listing 1.5).

```bash
# .bashrc
# Source global definitions
 if [ -f /etc/bashrc ]; then . /etc/bashrc; fi

# User specific aliases and functions
export PATH=$PATH:$HOME/bin:$HOME/TTool/MPSoC/soclib/utils/bin
export PATH=$PATH:/opt/mutekh/bin
export PATH=$PATH:/opt/jdk1.8.0_73/bin
export PATH=$PATH:$HOME/cxtools/gcc_mips/obj/bin/
export LD_LIBRARY_PATH=$HOME/TTool/MPSoC/mutekh/lib
```

**Listing 1.5:** Example for `.bashrc`

## 1.3 Installing and executing TTool

In order to install and execute TTool, run the following commands under the `$HOME/TTool/` directory:

```
> make ttool
> make install
> ./ttool.exe
```

After opening TTool, go to File>New Model. Right click on the design area and select "New SystemC-AMS Block Diagram". A new SystemC-AMS panel will open. Right click on the panel and select New SystemC-AMS Diagram. A new SystemC-AMS Component Diagram panel will open. In the same way, several SystemC-AMS Component Diagrams can be created inside the SystemC-AMS panel.

**Remark:** It is also possible to compile and run TTool under gradle (recommended if unitary tests should be run). This is not part of the Virtual Machine implementation; the proceding is explained on the TTool web site.

# 2 TTool's usage scenario

For this usage scenario, the TDF model shown in Figure 2.1 will be modeled and simulated in TTool. Module **A** will write a value of 2 to module **B**. Module **B** will read that value, multiply it by the last value received from the GPIO2VCI component, and transmit the result to the GPIO2VCI component which will be connected to the SoCLib interconnect component of an SoC platform.

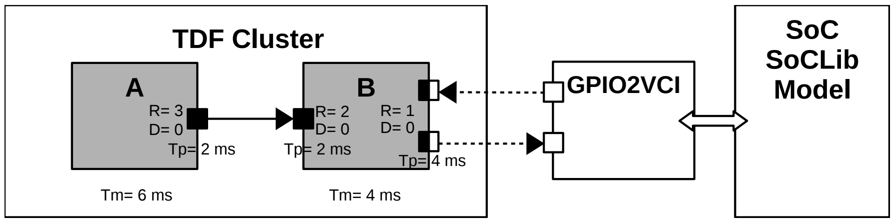


**Figure 2.1:** TDF Cluster model

Inside the SystemC-AMS Component Diagram panel TDF clusters can be created. To create a TDF cluster click on the "Cluster" button, number 1 of Figure 2.2, and click anywhere inside the SystemC-AMS Component Diagram panel to place the TDF Cluster block. Double-click to change the name of the TDF cluster. The size of the TDF cluster can be adjusted.

To add TDF module blocks, click on the "TDF Block" button, number 2 of Figure 2.2, and click anywhere inside the TDF Cluster block to place the TDF Module block. To add a DE module block follow the same procedure, just start by clicking on the "DE Block" button, number 3 of Figure 2.2. To add a GPIO2VCI block, click on the "GPIO2VCI block" button, number 4 of Figure 2.2. GPIO2VCI blocks should be placed outside of the TDF Cluster block.

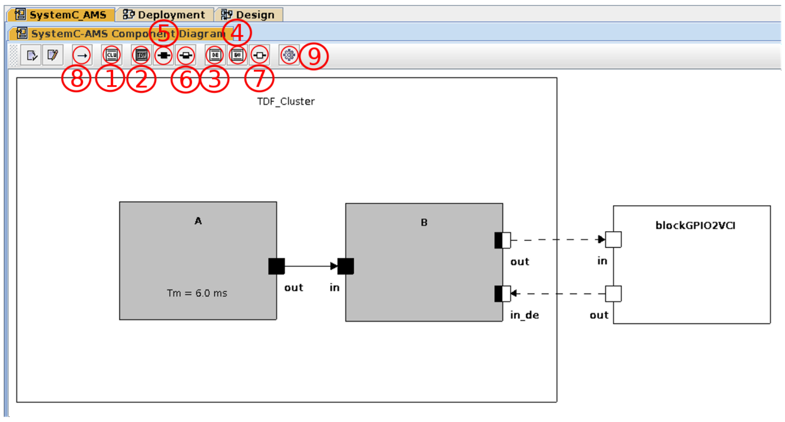
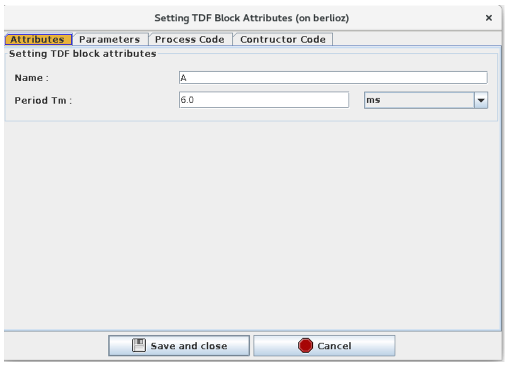


**Figure 2.2:** TDF Cluster creation in the SystemC-AMS Component Diagram panel.

The properties of the TDF module blocks can be set by double-clicking the block. A new window will open, as shown in Figure 2.3. In the Attributes panel the name and module timestep (Tm) including time units can be set, as Figure 2.3a shows. In the Parameters panel, seen in Figure 2.3b, the parameters of a TDF module such as its internal variables or template parameters can be also set up. In the Process Code panel, the `processing()` function of the module can be set, as Figure 2.3c shows. Finally, if constructor code needs to be added, it can be done in the Constructor Code panel. The attributes of the DE module blocks can be modified in the same way. The GPIO2VCI block has no attributes to be modified.



Setting TDF Block Attributes (on berlioz)

Attributes | Parameters | Process Code | Contructor Code

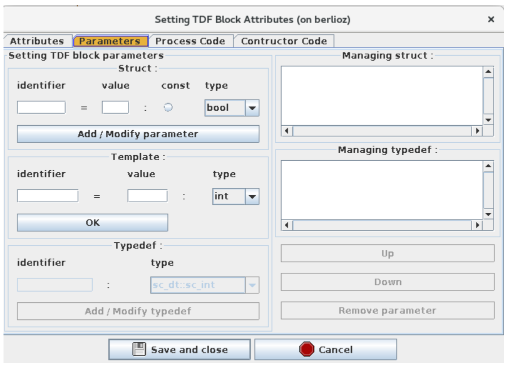
Setting TDF block attributes

Name : A

Period Tm : 6.0 ms

Save and close | Cancel

(a) Attributes panel



Setting TDF Block Attributes (on berlioz)

Attributes | Parameters | Process Code | Contructor Code

Setting TDF block parameters

Struct :

identifier = value : const type bool

Add / Modify parameter

Template :

identifier = value : type int

OK

Typedef :

identifier : type sc_dt::sc_int

Add / Modify typedef

Managing struct :

Managing typedef :

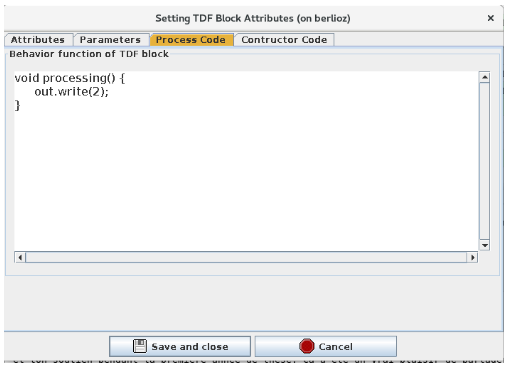
Up

Down

Remove parameter

Save and close | Cancel

(b) Parameters panel



Setting TDF Block Attributes (on berlioz)

Attributes | Parameters | Process Code | Contructor Code

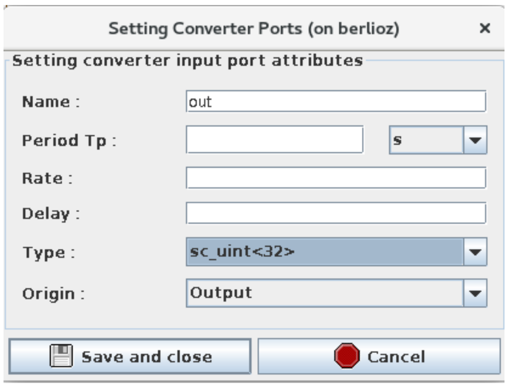
Behavior function of TDF block

```
void processing() {
    out.write(2);
}
```

Save and close | Cancel

(c) Process Code panel

**Figure 2.3:** TDF module block attributes window.


Setting Converter Ports (on berlioz)

Setting converter input port attributes

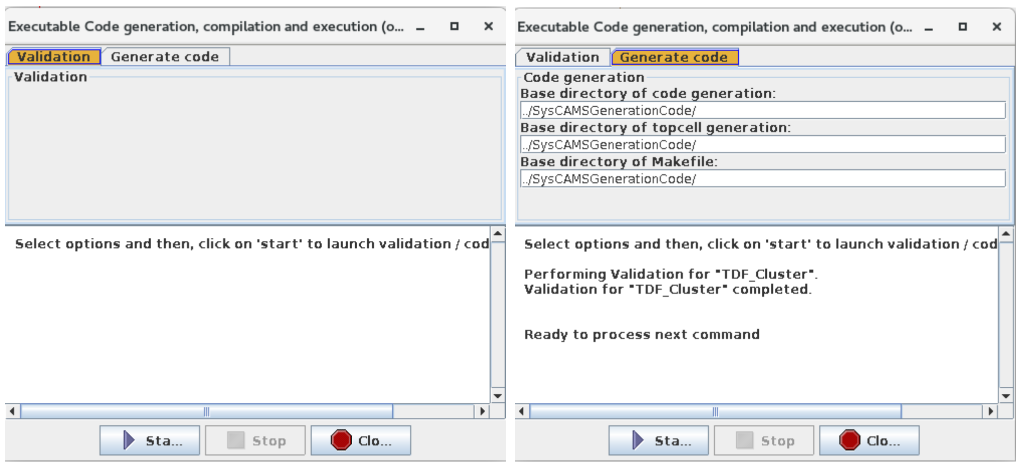
| | | |
|---|---|---|
| Name : | out | |
| Period Tp : | | s |
| Rate : | | |
| Delay : | | |
| Type : | sc_uint<32> | |
| Origin : | Output | |

**Figure 2.4:** Setting port attributes.

When the required modules have been created they need to be connected through their ports. The TDF ports and converter ports can be added to the TDF module blocks. Click on the “TDF port” button, number 5 of Figure 2.2, to add a TDF port. Click on the “Converter port” button, number 6 of Figure 2.2, to add a TDF converter port. DE ports can be added to the DE blocks and to the GPIO2VCI block by clicking on the DE port button, number 7 of Figure 2.2. The attributes of the ports can be modified by double-clicking a port, as shown in Figure 2.4. The name, timestep (Tp) along with the time units, rate, delay, type and origin of the port can be modified. Note that if a TDF module or a DE module will be connected to the GPIO2VCI component, the type `sc_uint<32>` should be selected as shown in Figure 2.4. For DE ports, the port can be added to the sensitivity list of the module by enabling the Sensitive field and selecting if the port will be sensitive to a positive or negative edge of the incoming signal or `null` for any incoming signal change. To connect the blocks, click the “Connector” button, number 8 of Figure 2.2, and then click an output port to connect it with an input port.

Once a TDF cluster model has been created. The next step is to validate the correctness of the model. This is done by clicking on the “Generate SystemC-AMS code” button, as shown in number 9 from Figure 2.2. This will open a new window, where validation of the model and code generation can be made. Click on the “Start” button to start the validation of the model, as shown in Figure 2.5a. The Validation panel will display a message stating if there is an error with the model and make suggestions on how to fix it. If the model is valid, then a success message will be displayed and the Generate Code panel will open, as shown in Figure 2.5b. Click on the “Start” button again to generate the SystemC-AMS code for the model.

In parallel, the Software Design and the Deployment Diagrams can be created. Right click on the tabs section of the design area and select “New Design” to create a new Software Design panel. A Block diagram can be created there, as shown in Figure 2.6. Click on the “Block” button, number 1 of Figure 2.6, to add a new block. Note that a new panel is created automatically, with the name of the block. Go to the **Block0** panel. Here, state machine diagrams that allow to design the software can be created, as shown in Figure 2.7. For this model, one state will be added by clicking on the “State” button, number 1 of Figure 2.7, and placing it in the panel. A stop block can be added by clicking the “Stop” button, number 2 of Figure 2.7. Finally the states should be connected by clicking the “Connect” button, number 3 of Figure 2.7.



Validation

Select options and then, click on 'start' to launch validation / cod

Code generation
Base directory of code generation:
../SysCAMSGenerationCode/
Base directory of topcell generation:
../SysCAMSGenerationCode/
Base directory of Makefile:
../SysCAMSGenerationCode/

Select options and then, click on 'start' to launch validation / cod

Performing Validation for "TDF_Cluster".
Validation for "TDF_Cluster" completed.

Ready to process next command

**(a)** Validation panel **(b)** Generate Code panel

**Figure 2.5:** Validation and code generation window.

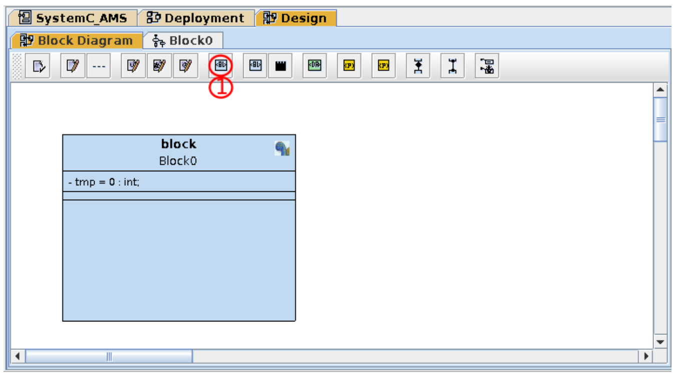


**Figure 2.6:** Software design Block Diagram panel.

By double-clicking the state block, C code can be entered manually in the Prototyping tab. Here is where the functions to communicate to the GPIO2VCI component can be added as shown in Figure 2.8. For the software of this model, a value of 5 will be written to the GPIO2VCI component. This value will be transmitted to the TDF cluster components. Then the output from the TDF cluster will be read and printed to the TTY component of the model. The code is shown in Listing 2.1.

```
tmp = read_gpio2vci("TDF_Cluster");
printf("Value read from TDF Cluster: %d\n", tmp);
write_gpio2vci(5, "TDF_Cluster");
tmp = read_gpio2vci("TDF_Cluster");
printf("Value read from TDF Cluster: %d\n", tmp);
```

**Listing 2.1:** State block code.

Note that the code is using a variable `tmp`. To create the variable in the Block Diagram panel, double click **Block0** to open the attributes window for the block, as shown in Figure 2.9. In the Attributes panel, new variables can be added by giving an identifier name, an initial value and a type.

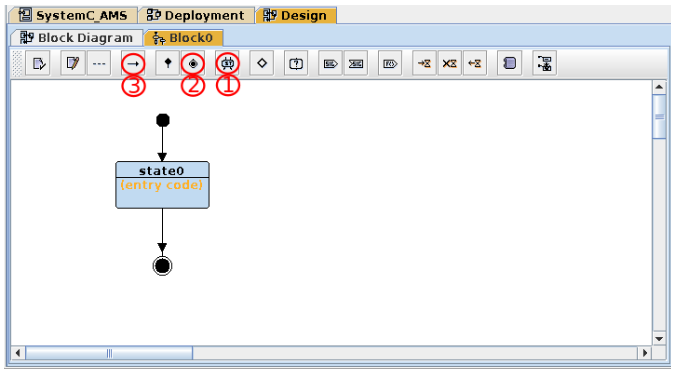
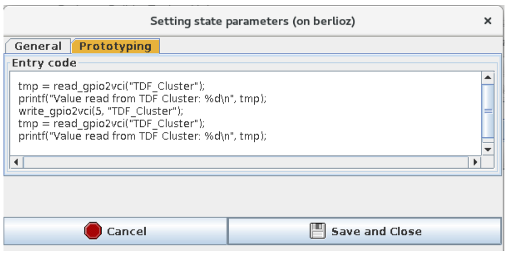


**Figure 2.7:** Software design State Machine Diagram panel.

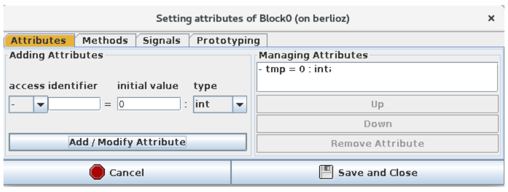


```
tmp = read_gpio2vci("TDF_Cluster");
printf("Value read from TDF Cluster: %d\n", tmp);
write_gpio2vci(5, "TDF_Cluster");
tmp = read_gpio2vci("TDF_Cluster");
printf("Value read from TDF Cluster: %d\n", tmp);
```

**Figure 2.8:** State block Prototyping panel .

Once that the software design is complete, the MPSoC model needs to be created in the Deployment Diagram. Here, the user can insert SoCLib components and the TDF clusters. To insert a CPU click the "CPU" button, number 1 of Figure 2.10. Double click the CPU block and setup the necessary attributes. To add a RAM memory click on the "RAM" button, number 2 of Figure 2.10. Double click the RAM block and set up its attributes. To add a TTY console click on the "TTY" button, number 3 of Figure 2.10. Finally an interconnect component needs to be added, by clicking the "VGMN" button, number 4 of Figure 2.10. To map the software blocks from the Block Diagram into a specific CPU, click the "Map and AVATAR block" button, number 5 of Figure 2.10, and place it under the CPU. Double click the block inside the CPU and select the name of the block that is mapped to that CPU.

In order to include the TDF clusters into the MPSoC model, they need to be added as SystemC-AMS Cluster blocks in the Deployment Diagram. To add a new SystemC-AMS Cluster block, click on the "Cluster" button, number 6 of Figure 2.10, and place the block in the Deployment Diagram panel. The name of the SystemC-AMS Cluster block should be the same name provided in the SystemC-AMS Component diagram. All the blocks should be connected to a SoCLib interconnect component using a connector, number 7 of Figure Figure 2.10. Once the necessary SystemC-AMS Cluster blocks have been added, the topcell from the Deployment Diagram model can be generated. In the Deployment Diagram Panel, click on the "Syntax analysis" button, number 8 of Figure 2.10. This will open a new window to verify the syntax of the model, as shown in Figure 2.11. Click on the "Check syntax" button. If there are any syntax errors, a message will be displayed, otherwise we can proceed to the generation of the topcell.


Setting attributes of Block0 (on berlioz)

Attributes | Methods | Signals | Prototyping

Adding Attributes

access identifier: -
initial value: 0
type: int

Add / Modify Attribute

Managing Attributes

- tmp = 0 : int;

Up
Down
Remove Attribute

Cancel
Save and Close

**Figure 2.9:** Block diagram's Block attributes.

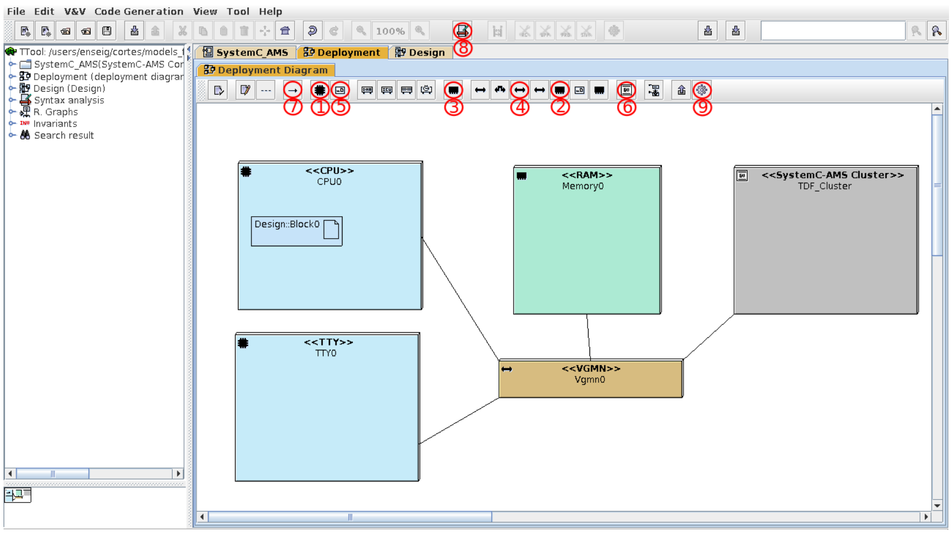
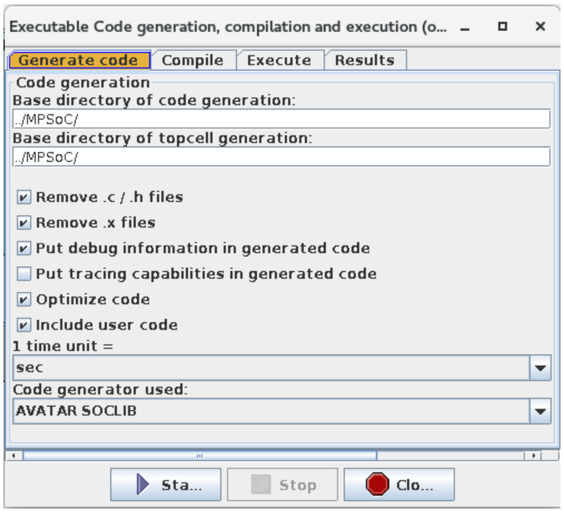


**Figure 2.10:** Adding SystemC-AMS Clusters to the Deployment Diagram.


Executable Code generation, compilation and execution (o...

Generate code | Compile | Execute | Results

Code generation

Base directory of code generation:
../MPSoC/

Base directory of topcell generation:
../MPSoC/

☑ Remove .c / .h files
☑ Remove .x files
☑ Put debug information in generated code
☐ Put tracing capabilities in generated code
☑ Optimize code
☑ Include user code

1 time unit =
sec

Code generator used:
AVATAR SOCLIB

Sta... | Stop | Clo...

**Figure 2.12:** Code generation window.

Click on the "Generate Deploy SoCLib" button in the Deployment Diagram panel, number 9 of Figure 2.10. A new window will be opened where the topcell code can be generated, compiled and executed. In Figure 2.12, the Generate Code panel is shown, where several option can be chosen, including the tracing capabilities and debugging information. Click "Start" to generate the topcell `top.cc` code and the software code. Then in the Compile panel, click "Start" to compile the code. Finally in the Execute panel, click "Start" to begin the simulation of the virtual prototype of the model.

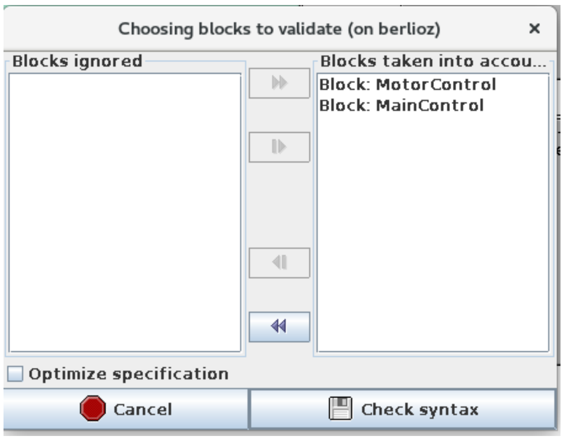
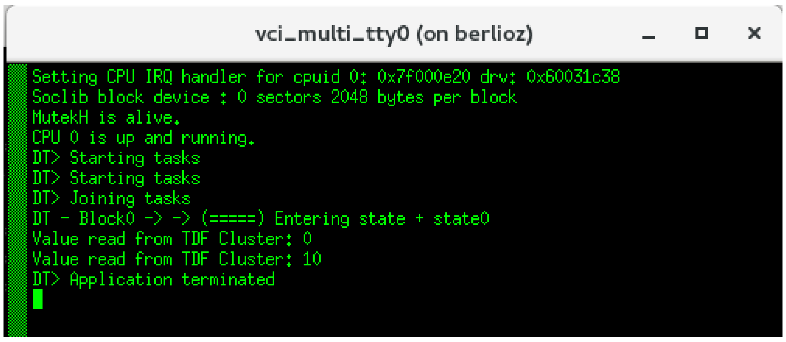


**Figure 2.11:** Check syntax window.

Figure 2.13 shows the TTY console from the model. In the last lines, the values being read from the TDF cluster are printed. The first value is 0, since nothing have been written to the TDF cluster. The last value is 10, since a value of 5 was written to the TDF cluster, and it is being multiplied by the value 2 generated from the TDF module **A**.



```
Setting CPU IRQ handler for cpuid 0: 0x7f000e20 drv: 0x60031c38
Soclib block device : 0 sectors 2048 bytes per block
MutekH is alive.
CPU 0 is up and running.
DT> Starting tasks
DT> Starting tasks
DT> Joining tasks
DT - Block0 -> -> (=====) Entering state + state0
Value read from TDF Cluster: 0
Value read from TDF Cluster: 10
DT> Application terminated
```

**Figure 2.13:** Simulation output from the TTY component of the model.

# 3 Directory tree of source code and generated files

In this section, the directory tree of all source files modified for this project and all the generated files is shown. Listing 3.1 shows the location of the automatically generated files from TTool. The SystemC-AMS generated files for the TDF clusters are stored under the `generated_CPP/` directory, and the generated files for the TDF modules are stored under the `generated_H/` directory. The generated source code files for the software of the virtual prototype are stored under the `generated_src/` directory. The generated topcell is stored in the `generated_topcell/` directory.

```
$HOME/TTool/
├── SysCAMSGenerationCode/
│   ├── generated_CPP/
│   │   └── *_tdf.h
│   └── generated_H/
│       └── *_tdf.h
└── MPSoC/
    ├── generated_src/
    │   ├── main.c
    │   └── Block0.c
    └── generated_topcell/
        └── top.cc
```

**Listing 3.1:** Generated code files directories.

The GPIO2VCI component was created under the `connectivity_component/` directory, as shown in Listing 3.2

```
$HOME/TTool/MPSoC/soclib/soclib/module/connectivity_component/gpio2vci/caba/
└── metadata/
    ├── gpio2vci.sd
    ├── source/include/
    │   └── gpio2vci.h
    └── source/src/
        └── gpio2vci.cpp
```

**Listing 3.2:** GPIO2VCI component directories.

Listing 3.3 shows the java files that were modified or created as part of the integration of the SystemC-AMS modules and SoCLib modules into TTool.

```
$HOME/TTool/src/main/java/
 ├── ui/
 │   ├── window/
 │   │   ├── JDialogSysCAMSExecutableCodeGeneration.java
 │   │   ├── JDialogSysCAMSBlockDE.java
 │   │   ├── JDialogSysCAMSBlockTDF.java
 │   │   ├── JDialogSysCAMSPortConverter.java
 │   │   ├── JDialogSysCAMSPortDE.java
 │   │   └── JDialogSysCAMSPortTDF.java
 │   └── AvatarDeploymentPanelTranslator.java
 ├── syscamstranslator/
 │   ├── toSysCAMSCluster/
 │   │   ├── ClusterCode.java
 │   │   ├── Header.java
 │   │   ├── PrimitiveCode.java
 │   │   └── TopCellGeneratorCluster.java
 │   ├── SysCAMSTBlockTDF.java
 │   ├── SysCAMSSpecification.java
 │   ├── SysCAMSTPortDE.java
 │   ├── SysCAMSTPortTDF.java
 │   ├── SysCAMSTPortConverter.java
 │   └── SysCAMSValidateException.java
 └── ddtranslatorSoclib/
     ├── toSoclib/
     │   ├── Gpio2VciAddress.java
     │   ├── TaskFileSoclib.java
     │   └── TasksAndMainGenerator.java
     ├── toTopCell/
     │   ├── Declaration.java
     │   ├── Header.java
     │   ├── MappingTable.java
     │   ├── NetList.java
     │   ├── Platforminfo.java
     │   ├── Signal.java
     │   └── TopCellGenerator.java
     ├── AvatarAmsCluster.java
     └── AvatarddSpecification.java
```

**Listing 3.3:** Java files created or modified for the integration of SystemC-AMS and SoCLib modules.

The `libsyscams` library created to provide interface functions for communication with the GPIO2VCI component is shown under Listing 3.4.

```
$HOME/TTool/MPSoC/mutekh/libsyscams/
 ├─ gpio2vci_address.c
 ├─ gpio2vci_address.h
 ├─ gpio2vci_iface.c
 ├─ gpio2vci_iface.h
 ├─ libsyscams.config
 └─ Makefile
```

**Listing 3.4:** `libsyscams` library source files.

Listing 3.5 shows other files that were modified as part of the integration tasks.

```
$HOME/TTool/MPSoC/
 ├─ Makefile.forsoclib
 └─ generated_topcell/
     └─ config_noproc
```

**Listing 3.5:** Other modified files for the integration tasks.