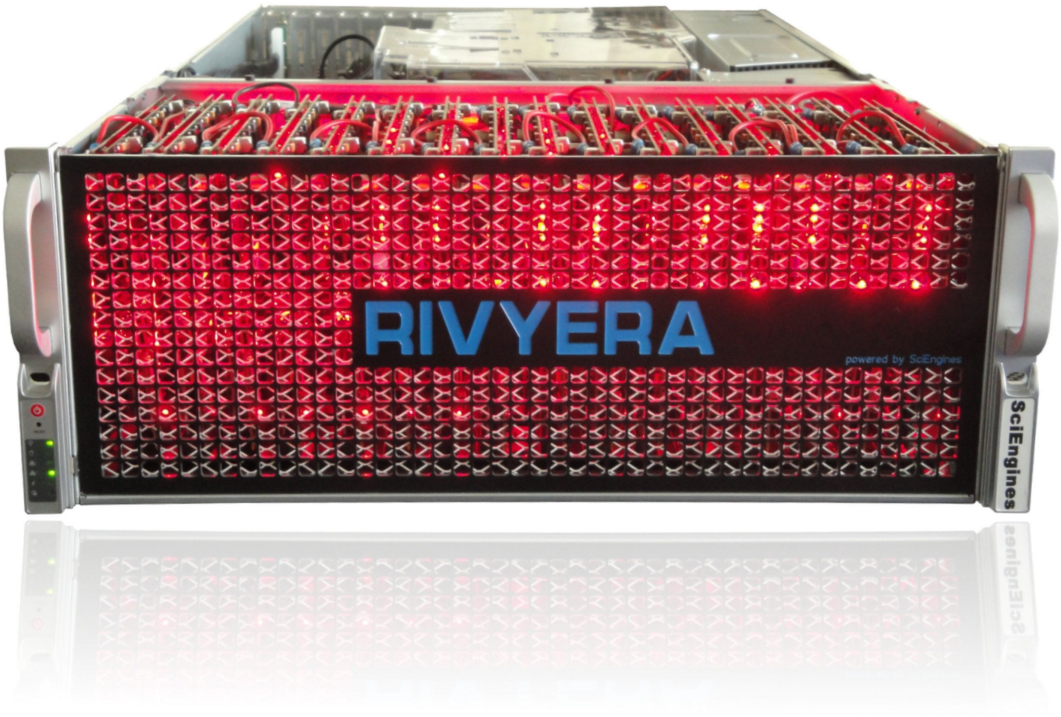
SciEngines
massively parallel computing

RIVYERA API

SciEngines RIVYERA API Documentation

API Simulation Guide

Version 1.95.00



SciEngines GmbH

Revision: 1326 1.95.00 March 9, 2022

Released version

SciEngines RIVYERA API Documentation

API Simulation Guide
Version 1.95.00

SciEngines GmbH

SciEngines GmbH
Am-Kiel-Kanal 2
24106 Kiel
Germany

Public

Released version

Prepared for SciEngines GmbH, Am-Kiel-Kanal 2, 24106 Kiel, Germany
Under No Contract Number

Abstract: This introduction offers a brief overview of the SciEngines RIVYERA computer. It describes the physical and structural details from the programmers' point of view.

The main purpose of the RIVYERA API is to interface with single and multiple FPGAs in a massively parallel architecture as simply and easily as possible. We intend to provide an infrastructure for your FPGA designs which allows you to leverage the benefits of a massively parallel architecture without raising the complexity of your design.
Therefore, we provide a simple interface hiding the idiosyncratic implementation details of the physical layers while permitting a high-level view of your RIVYERA computer.

Contents

Figures and Tables

Figures

Tables

0.1 Introduction

SciEngines API-Simulator is a tool to speedup the development of software for the RIVYERA computers. It directly simulates the VHDL-design and communication between host and RIVYERA without the need to spend hours until ready-to-use bitfiles are generated or the need to write custom VHDL test-benches for your designs. All RIVYERA API commands are available for the simulation of application code. Thus, either the host-software or `se_mon` utility are able to interface with the VHDL design in the usual way. The integration into the SciEngines ProjectCompass provides a comfortable way to test even high complexity VHDL designs at every phase of the development process. Because of that, writing any time-consuming test-bench with simplified synthetic data becomes redundant.

0.1.1 Features

- Simulate whole VHDL side API and user core behavior without writing one line of testbench code
- Develop your application without access to SciEngines RIVYERA allowing RIVYERA to be running while building new applications in parallel
- Maps complete API functionality and communication
- Full support for `se_mon` utility
- See real-time status of the FPGA's internal states and all signals in Xilinx ISim waveform
- Use simulation file like a normal .bit file in your host application to test the whole workflow
  - No need to write a testbench for each VHDL design
  - Early design error termination
- Advanced (long time) debugging with real data
- Step by step design simulation inside normal Xilinx ISim environment

0.2 Functional Description of the API Simulation

On real RIVYERA hardware the communication flow is like shown in figure 1. In general the flow is split into two parts, the host side seen in figure 2 and the RIVYERA FPGA side.

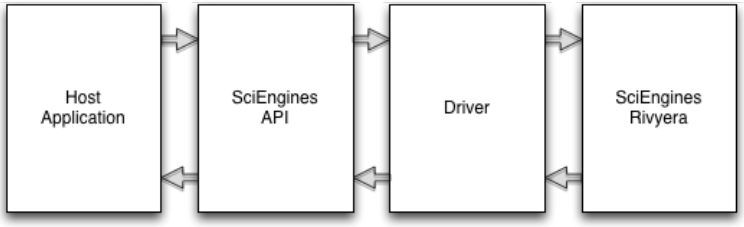


Figure 1. API communication flow

The API-Simulator is a tool to speedup the software development for the RIVYERA. Therefore the API Simulation is fully integrated into the host side part of the API, enabling usage of all API functions and locking mechanisms to exclusively use a machine. It also synchronizes the response timeout handling with the ISim instance for better hardware behavior. So from the user's point of view, the host side communication part remains untouched (figure 2).

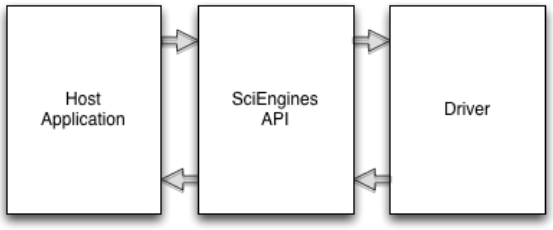


Figure 2. API host communication flow

As seen in figure 3, only the communication between host and RIVYERA is replaced by the API-Simulator to keep the expected control flow.

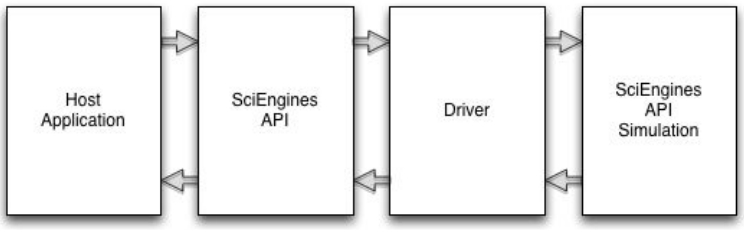


Figure 3. API Simulation communication flow

The API-Simulator maps the communication between host and RIVYERA to a named pipe file structure and uses Xilinx ISim to handle the VHDL design using ISE behavioral models. This makes it possible to directly pass test data and responses between the host side part of the application or `se_mon` and the designed VHDL core.

0.3 File structure of the API Simulation

In general only the files `ProjectName.sim`, `.sim.run` and `isim.cmd` are relevant to the user. They are all generated automatically by ProjectCompass.

```
ProjectName
`-- fpga
     `-- vhdl
```

```
          |-- isim.cmd
          |-- ProjectName.sim
          |-- .sim.run
```

There are also files generated by .sim.run file inside the user folder for each simulation device. The number of simulation devices is configured inside /etc/sciengines.conf file (see Installation User Guide on how to configure such devices). The three FIFOs help handling the communication flow between API and ISim in a hardware near manner. The other two files are needed by ISim directly. Below is an example file tree with two simulation machines.

```
/home/UserName/.sciengines
|-- 0
|   |-- fifo_host_to_isim
|   |-- fifo_host_to_isim_ack
|   |-- fifo_isim_to_host
|   |-- fuse.out
|   `-- sim.prj
`-- 1
    |-- fifo_host_to_isim
    |-- fifo_host_to_isim_ack
    |-- fifo_isim_to_host
    |-- fuse.out
    `-- sim.prj
```

0.3.1 ProjectName.sim file

The `ProjectName.sim` is used to program a simulation machine in the same manner as .rbt and .bit files on real hardware using the API function `se_program()`. It only contains the FPGA type, the project is designed for, and the path to the `.sim.run` file.

```
\$ cat  ./ProjectName/fpga/vhdl/ProjectName.sim
6slx150fgg676
.sim.run
```

0.3.2 .sim.run file

Inside this file the whole work is done. In general there are four main parts executed:

```
main() {
   loadConfig
   setup_variables \$@
   setup_files
   run_simulation

   return 0
}
```

In first step the general /etc/sciengines.conf file is loaded. For more information about this file see the Installation User Guide.

```
loadConfig() {
   if [ -f "/etc/sciengines.conf" ]; then
      source /etc/sciengines.conf
      BASE_DIR=$(dirname "$PATH_SE_HOME")
      return 0
   else
      perror "ERROR! Unable to access \"/etc/sciengines.conf\"."
      return 1
   fi
}
```

The setup_variables function sets all required ISim testbench variables :

```
setup_variables() {
   # set environment variables
   PROJECT_PATH="$(cd ${1} 2> /dev/null && pwd)"
   ISIM_DEVICE="${2}"
   CLK_PERIOD=10ns
   API_I_FILEPATH="${HOME}/.sciengines/${ISIM_DEVICE}/fifo_host_to_isim"
   API_I_ACK_FILEPATH="${HOME}/.sciengines/${ISIM_DEVICE}/fifo_host_to_isim_ack"
   API_O_FILEPATH="${HOME}/.sciengines/${ISIM_DEVICE}/fifo_isim_to_host"
   PRJ_FILE="${HOME}/.sciengines/${ISIM_DEVICE}/sim.prj"
   WDB_FILE="${HOME}/.sciengines/${ISIM_DEVICE}/sim.wdb"
   FUSE_OUT="${HOME}/.sciengines/${ISIM_DEVICE}/fuse.out"
   XISE_FILE="${PROJECT_PATH}/ProjectName.xise"
   TB_FILE="work.ProjectName_main_tb"
   TCL_FILE="${PROJECT_PATH}/isim.cmd"
   ISIM_VIEW="${PROJECT_PATH}/ProjectName_main_tb.wcfg"
   PATH="/opt/Xilinx/latest/ISE_DS/ISE/bin/lin64:$PATH"

   # check if path exists and is accessible
   if [ -z "${PROJECT_PATH}" ]; then
      perror "Unable to enter directory ${PROJECT_PATH}"
      return 1
   else
      return 0
   fi
}
```

The third step handles generation of needed files inside user folder :

```
setup_files() {
   # remove old files and setup directory
   rm -rf "${HOME}/.sciengines/${ISIM_DEVICE}" > /dev/null 2>&1 \
      || perror "Error removing directory
          ${HOME}/.sciengines/${ISIM_DEVICE}"
   mkdir -p "${HOME}/.sciengines/${ISIM_DEVICE}" > /dev/null 2>&1 \
      || perror "Error creating directory
          ${HOME}/.sciengines/${ISIM_DEVICE}"

   # create prj file
   "${PATH_SE_HOME}/bin/se_prjgen" "${XISE_FILE}" -o "${PRJ_FILE}" \
      || perror "Error creating prj file ${PRJ_FILE} from xise file
          ${XISE_FILE}"

   # initialize fifos
   mkfifo -m 700 "${API_I_FILEPATH}" > /dev/null 2>&1 \
      || perror "Error creating FIFO ${API_I_FILEPATH}"
   mkfifo -m 700 "${API_I_ACK_FILEPATH}" > /dev/null 2>&1 \
      || perror "Error creating FIFO ${API_I_ACK_FILEPATH}"
   mkfifo -m 700 "${API_O_FILEPATH}" > /dev/null 2>&1 \
      || perror "Error creating FIFO ${API_O_FILEPATH}"

   return 0
}
```

Finally Isim instance executes :

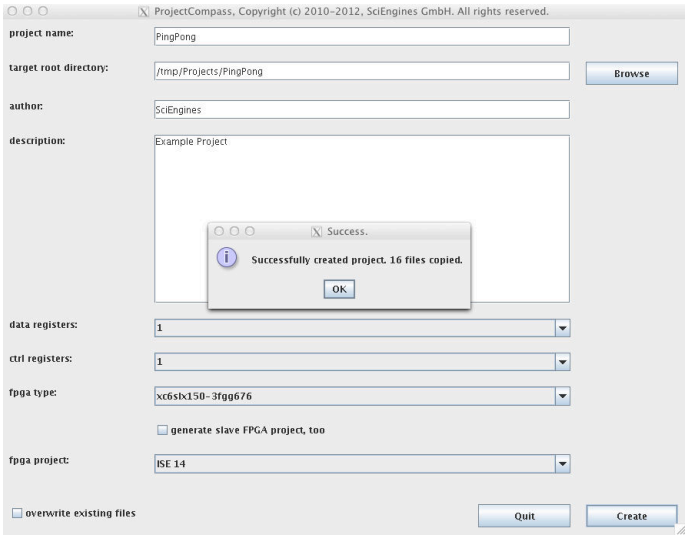
```
run_simulation() {
   cd ${PROJECT_PATH} > /dev/null 2>&1 \
      || perror "Error changing working directory to ${PROJECT_PATH}"
   fuse -intstyle ise -o ${FUSE_OUT} -prj ${PRJ_FILE} ${TB_FILE}
       -generic_top "NUM_LEDS=${NUM_LEDS}" -generic_top
       "CLK_PERIOD=${CLK_PERIOD}" -generic_top
       "API_I_FILEPATH=${API_I_FILEPATH}" -generic_top
       "API_I_ACK_FILEPATH=${API_I_ACK_FILEPATH}" -generic_top
       "API_O_FILEPATH=${API_O_FILEPATH}" > /dev/null \
```

```
      || perror "Error setting up simulation"

   # run simulation
   XILINX="/opt/Xilinx/latest/ISE_DS/ISE/" \
      PLATFORM="lin64" \
      LD_LIBRARY_PATH="${XILINX}/lib/${PLATFORM}" \
      PATH="/opt/Xilinx/latest/ISE_DS/ISE/bin/lin64:${PATH}" \
      $FUSE_OUT -gui -intstyle ise -tclbatch ${TCL_FILE} -view
          $ISIM_VIEW -wdb ${WDB_FILE} &

   return 0
}
```

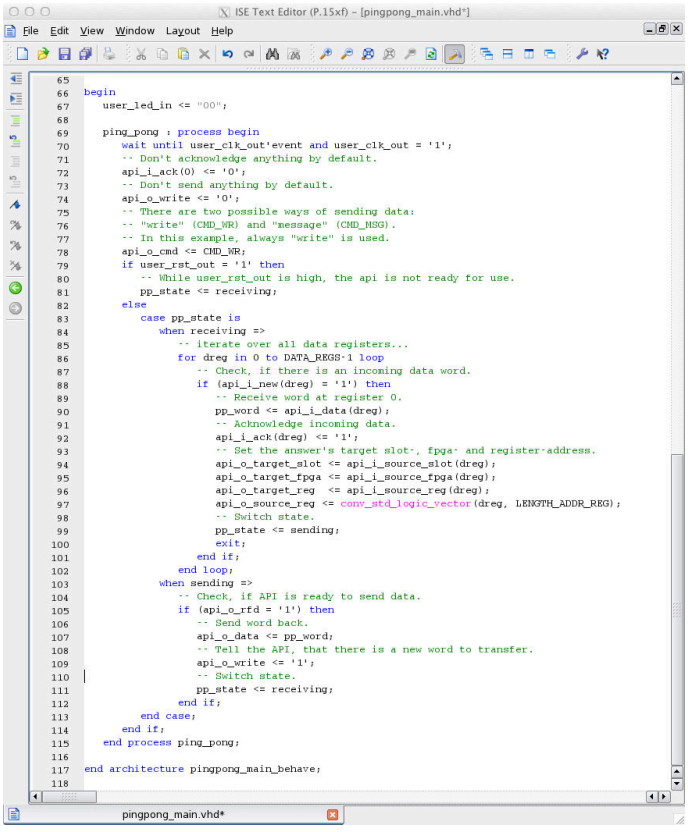
0.4 Example

In this example we simulate the `PingPong` demo without writing any line of testbench code. In short, this example takes incoming words from the API and directly writes these words back to the source. Please have a look at the PingPong documentation within the examples directory to learn more about the PingPong example.

1. Generate a project with SciEngines ProjectCompass (for an in-depth tutorial see the ProjectCompass guide).



2. Open project with ISE and develop your VHDL core. Here, we see the ping_pong example process.



3. Test your design with SciEngines API Simulation and `se_mon`. For this purpose open a terminal and navigate into project dir.

```
# :>cd /tmp/Projects/PingPong/
```

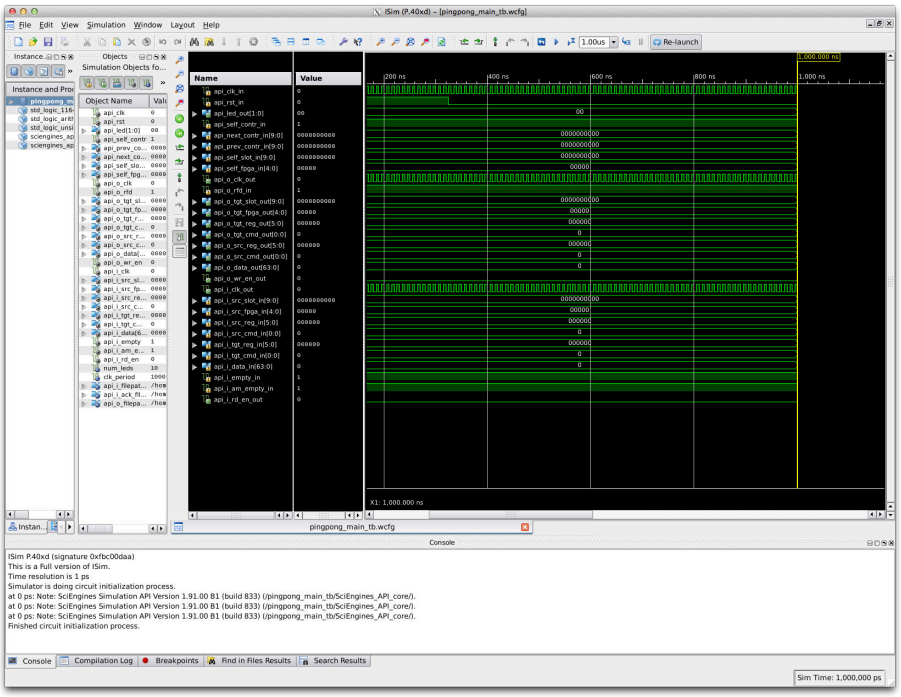
4. Open `se_mon` and allocate one of the simulation machines configured inside SciEngines Configuration File (see Installation User Guide for an explanation on how to add such machines). The configuration file is found in `/etc/sciengines.conf`.

```
# :>se_mon
se_mon version 1.95.08
Copyright (c) 2011-2020, SciEngines GmbH
All rights reserved.

SciEngines RIVYERA Host-API version 1.95.01, build 1362

 -- Enter "help" to get some help. --

Machine=0 Contr=0 Slot=0 FPGA=0 > goto 1 0 0 *
Machine=1 Contr=0 Slot=0 FPGA=* > allocMachine 0
0m0.022s (allocMachine)
```

5. Start the simulation by executing .sim-file which is automatically generated by ProjectCompass. This is done inside `se_mon` with the`program` command. The .sim-file in `PROJECTDIR/fpga/VHDL/PROJECTNAME.sim` is automatically generated by ProjectCompass. Because the simulation takes a bit longer for some operations, especially for reading from the simulated FPGA, we set the timeout to 10 seconds.

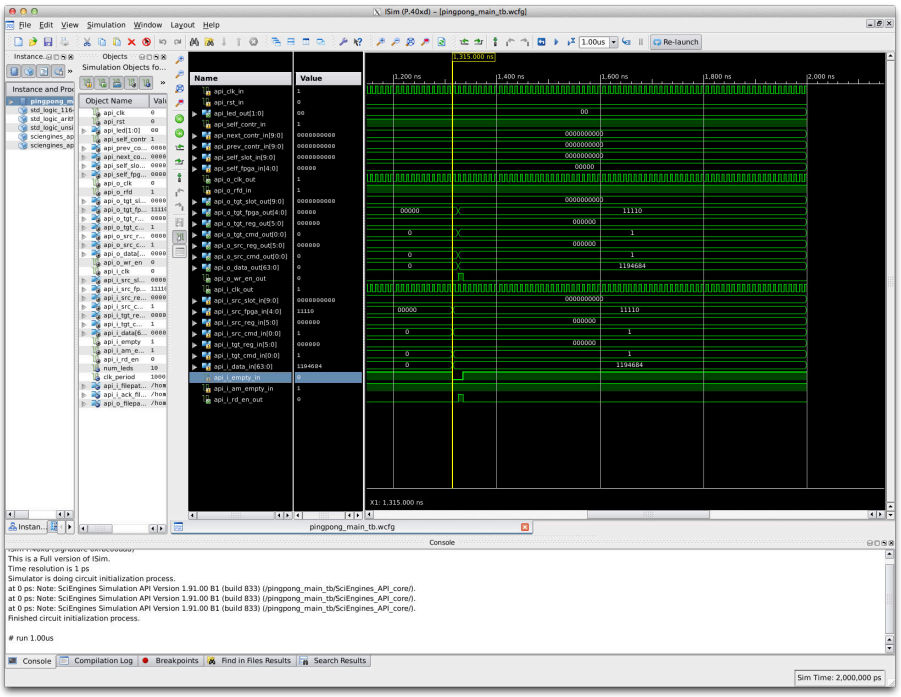
```
Machine=1 Contr=0 Slot=0 FPGA=* > timeout 10000
Machine=1 Contr=0 Slot=0 FPGA=* > program fpga/VHDL/pingpong.sim
Overriding Xilinx file <images/gui-clip-help-support-home.png>
    with local file
</opt/Xilinx/14.3/ISE_DS/ISE/data/images/gui-clip-help-support-home.png>
0m4.835s (program)
```

6. This automatically executes a Xilinx ISim instance. Feel free to use run all, run the simulation for a specified time or the step by step function. You are also able to add additional waves, delete or move existing ones and save the wcfg file to reuse it the next time.



7. Send the word $0x123ABC$ to register 0 and run some iterations in ISim until the word is arrived on `api_i_data`. Also the signal `api_i_empty_in` goes to 1 signalizing new word available for user core. One clock cycle later, the pingpong core sets the `api_i_rd_en_in` signal to 1 telling the API that it noticed the data word.

```
Machine=1 Contr=0 Slot=0 FPGA=* > goto 1 0 0 0
Machine=1 Contr=0 Slot=0 FPGA=0 > write 0 0x123ABC
```



8. Wait for data on controller 0 in `se_mon` and spend some more time watching signals. As expected inside ISim, on the next clock cycle the user core sets `api_i_rd_en_out` to 0 and the API reacts on the read signal and also sets `api_i_empty_in` to 0. In the same clock cycle the pingpong core sets the output `api_o_data_out` register to $0x123ABC$ and raises the `api_o_wr_en_out` signal which makes the API write the data to the host.

```
Machine=1 Contr=0 Slot=0 FPGA=0 > waitForData
m1 c0 s0 f0 r0 with 1 words
0m3.901s (waitForData)
```

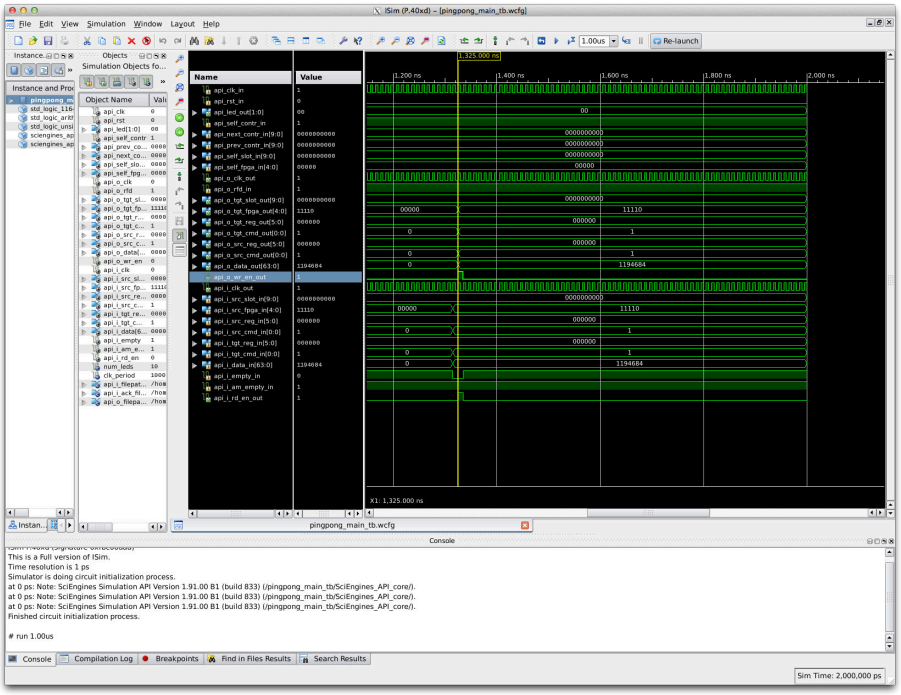
9. Try to read two words from register 0. As expected the VHDL core only generates one word of data and the API returns a timeout error and only returns the word $0x123ABC$.

```
Machine=1 Contr=0 Slot=0 FPGA=0 > readPassive 0 2
No.      decimal  hexadecimal                binary
[    0] 1194684  0000000000123ABC  0...0 00010010 00111010 10111100

ERROR! SciEngines API returned "SeApiReadTimeout" (7)! 1 of 2
    words transferred.
0m10.000s (readPassive)
```

10. Deprogramming the machine automatically stops the current Xilinx ISim. It remains open for further inspection.

```
Machine=1 Contr=0 Slot=0 FPGA=0 > goto *
Machine=1 Contr=0 Slot=0 FPGA=* > deprogram
0m6.191s (deprogram)
```

11. Finally free the machine and quit `se_mon`




```
Machine=1 Contr=0 Slot=0 FPGA=* > freeMachine
Machine=1 Contr=0 Slot=0 FPGA=* > quit
# :>
```

Imprint:

SciEngines GmbH
Am Kiel-Kanal 2
D-24106 Kiel Germany

Phone:	+49(0)431-9086-2000
Fax:	+49(0)431-9086-2009
E-Mail:	info@SciEngines.com
Internet:	www.SciEngines.com

CEO: Gerd Pfeiffer

Commercial Register:	Amtsgericht Kiel
Commercial Register No.:	HR B 9565 KI
VAT-Identification Number:	DE 814955925