The OSVVM Simulator Script Library

The OSVVM Simulator Script Library provides a simple way to create and activate libraries, compile designs, and run simulations.

The intent of this scripting approach is to:

- Run the same scripts on any simulator
- Be as easy to read as a compile order list.
- Know the directory the script is in, so it does not have to be passed.
- Simplify integration of other libraries that use the same approach

This is an evolving approach. So it may change in the future. Input is welcome.

Clone the OSVVM-Libraries directory

Lets start by doing. The library OSVVM-Libraries contains all of the OSVVM libraries as submodules. Download the entire OSVVM model library using git clone with the `--recursive` flag:

```
$ git clone --recursive https://github.com/osvvm/OsvvmLibraries
```

The Script Files

- Startup.tcl
  - Detects the simulator running and calls the VendorScript???.tcl. Also calls OsvvmProjectScripts.tcl and OsvvmScriptDefaults.tcl
- VendorScript???.tcl
  - TCL procedures that do simulator specific actions.
  - ??? = one of (ActiveHDL, GHDL, Mentor, RivieraPro, VSimSA)
  - VSimSA is the one associated with ActiveHDL.
- OsvvmProjectScripts.tcl
  - TCL procedures that do common simulator and project build tasks.
- OsvvmScriptDefaults.tcl
  - Default settings for the OSVVM Script environment.

Create a Sim directory

Create a simulation directory. Generally I name this "sim" or "sim<simulator name>". Creating a simulation directory means that cleanup before running regressions is just a matter of deleting the sim directory and recreating a new one.

The following assumes you have created a directory named "sim" in the Osvvm-Libraries directory.

Alternately, you can run simulations out of the Scripts, but cleanup is a mess as a simulator tends to create numerous temporaries.

Preparation

Edit StartUp.tcl and adjust LIB_BASE_DIR to be appropriate for your project. LIB_BASE_DIR determines where libraries are created – note that OSVVM uses a separate named library for different families of verification components. This directory can be your sim directory you created in the previous step, however, I prefer that it goes into a directory that is not backed up. Such as:

```
set LIB_BASE_DIR C:/tools/sim_temp
```

Initialization

RivieraPRO, QuestaSim, ModelSim

Initialize the OSVVM Script environment by doing:

```
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

Want to avoid doing this every time? In Aldec RivieraPro, set the environment variable, ALDEC_STARTUPTCL to StartUp.tcl (including the path information). Similarly in Mentor QuestaSim/ModelSim, set the environment variable, MODELSIM_TCL to StartUp.tcl (including the path information).

ActiveHDL

Before doing this in ActiveHDL and VSimSA (ActiveHDL's command window) instead do:

```
scripterconf -tcl
do -tcl <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

Want to avoid doing this every time? For ActiveHDL, edit /script/startup.do and add above to it. Similarly for VSimSA, edit /BIN/startup.do and add the above to it. Note, with 2021.02, you no longer need to set the "Start In" directory to the OSVVM Scripts directory.

GHDL

I currently run GHDL using MSYS2 64 bit under windows. The scripts must run under tcl (tclsh). As a result, to start the OSVVM scripting environment, in a shell window do:

```
winpty rlwrap tclsh
source <path-to-OsvvmLibraries>/OsvvmLibraries/Scripts/StartUp.tcl
```

To simplify this, I put the `source .../StartUp.tcl` in my `.tclshrc` file and as a result I do not have to do the source command.
I have added a short cut that includes `C:\tools\msys64\mingw64.exe winpty rlwrap tclsh`. I added the short cut to my start menu. With these two, one click and you are running in the OSVVM tcl execution environment.

Alternately, if you are not running in windows, create the `.tclshrc` as above and then in your `.bashrc` create the alias `alias gsim='winpty rlwrap tclsh'` to simplify starting tclsh. From there, at the command line type gsim and you are running ghdl in the OSVVM environment.

Project Files

A project file is a script that allows the specification of basic tasks to run a simulation:

- library - Make this library the active library. Create it if it does not exist.
- analyze - Compile the design into the active library.
- Simulate - Simulate the design using the active library.
- include – include another project script
- build – include + start a new log file for this task

The above tasks are TCL procedures. Hence, a project file is actually a TCL file, and when necessary, TCL can be used, however, the intent is to keep it simple. The naming of the project file is of the form <Name>.pro.

The following is an excerpt from OsvvmLibraries/AXI4/Axi4/Axi4.pro. It first activates the library osvvm_axi4. Next it compiles all of the files in the src directory.

```
library osvvm_axi4
analyze ./src/Axi4MasterComponentPkg.vhd
analyze ./src/Axi4ResponderComponentPkg.vhd
analyze ./src/Axi4MemoryComponentPkg.vhd
analyze ./src/Axi4MonitorComponentPkg.vhd
analyze ./src/Axi4Context.vhd
analyze ./src/Axi4Master.vhd
analyze ./src/Axi4Monitor_dummy.vhd
analyze ./src/Axi4Responder_Transactor.vhd
analyze ./src/Axi4Memory.vhd
```

The following is an excerpt from OsvvmLibraries/AXI4/Axi4/testbench/testbench.pro. It first activates the library osvvm_TbAxi4. Next it compiles the entity for the testbench sequencer (TestCtrl_e.vhd), the test harness (TbAxi4.vhd), and the test architectures (TbAxi4_RandomReadWrite.vhd and TbAxi4_MemoryBurst.vhd). Finally it simulates the test TbAxi4_MemoryBurst by calling its configuration (which follows the test architecture in the same file).

```
library osvvm_TbAxi4
analyze TestCtrl_e.vhd
analyze TbAxi4.vhd
analyze TbAxi4_RandomReadWrite.vhd
analyze TbAxi4_MemoryBurst.vhd

# simulate TbAxi4_RandomReadWrite
simulate TbAxi4_MemoryBurst
```

Building and Running OSVVM Testbenches

To build all of the OSVVM Libraries, run the script, OsvvmLibraries.pro. In your simulator do the following. This will make you ready to run any of the testbenches.

```
cd <OsvvmLibraries directory>/sim
build ../OsvvmLibraries.pro
```

Now lets run the AXI4 testbench by doing the following in your simulator. You might note that the ".pro" extension was left off. When this is done and the last name is a directory, it looks for a file in that directory of the form <directory-name>.pro – hence here testbench.pro.

```
build ../AXI4/Axi4/testbench
```

Note in the AXI4 testbench.pro script, the test, TbAxi4_RandomreadWrite, was not run. Lets run it now. After running the testbench.pro script, the active library is still osvvm_TbAxi4. From the simulator command line, you can run the TbAxi4_RandomreadWrite test by typing the following:

```
simulate TbAxi4_RandomReadWrite
```

All OSVVM verification components include a testbench. You can learn much about how to use a model in a test by reading the testbenches. Run the other OSVVM verification components by doing the following. .. code-block:: tcl

```
build ../AXI4/Axi4/testbench
build ../AXI4/Axi4Lite/testbench
build ../AXI4/AxiStream/testbench
build ../UART/testbench
```

Commands

Command	Description
library <library>	Make the library the active library. If the library does not exist, create it and create a mapping to it. Libraries are created either in the path specified by LIB_BASE_DIR in Scripts/StartUp.tcl.
analyze <file>	Compile the file. A path name specified is relative to the location of the current <file>.pro directory location. Library is the one specified in the previous library command.
simulate <design-unit>	Start a simulation on the design unit. Often best if this is a configuration name. Library is the one specified in the previous library command.
include <directory>	Include accepts either a file or a directory.
include <path>/<file>	If it is a file and its extension is .pro, .tcl, or .do, the file will be sourced. If it is a file and its extension is .files or .dirs it will be handled in a manner consistent with revision 1 of the scripts. If it is a directory, then files whose base name matches the directory name and that have the extensions .pro, .dirs, .files, .tcl, and .do are searched for (in this order) and any found will be processed as above if they exist. A path name specified is relative to the location of the current <file>.pro directory location.

Command	Description
build <directory>	Re-initializes the working directory to
build <path>/<file>	the script directory, opens a transcript
	file, and calls include. A path name
	specified is relative to the location of
	the current <file>.pro directory
	location.
map <library> [<path>]	Create a mapping to a library
RemoveAllLibraries	Delete all of the working libraries.
SetVHDLVersion	Set VHDL analyze version
	Valid values = (2008, 2019, 1993, 2002)
	OSVVM libraries require 2008 or newer
GetVHDLVersion	Return the current VHDL Version
SetSimulatorResolution	Set Simulator Resolution
	Any value supported by the simulator is ok
GetSimulatorResolution	Return the current Simulator Resolution
LinkLibrary	Link libraries that are in the LibraryDirectory
	LibraryDirectory is the directory that contains
	an OSVVM created VHDL_LIBS directory
Undocumented Procedures	Any undocumented procedure is in development
	and may change in a future revision

Deprecated Descriptor Files

The scripts still have support for older forms of the scripts. These are intended to be deprecated.

If you run your scripts using by just specifying the <DirectoryPathName>, then you want to avoid having files of the form <DirectoryName>.tcl, <DirectoryName>.dirs, <DirectoryName>.files, and <DirectoryName>.sim.

<DirectoryName>.tcl is a tcl script file name. We abandoned pure TCL scripting because different simulators did not provide a convenient way to dealing with path names when the scripts were located.

<DirectoryName>.dirs is a directory descriptor file and and contains a list of directories to search for other descriptor files.

<DirectoryName>.files is a file descriptor file and contains a list of names that are either .vhd for vhdl, .v for Verilog or .lib for current working library. If the file descriptor file contains a .lib, it sets the current working library to that name without the .lib extension and that library will be used until it is changed or the end of file is reached. The .vhd and .v files will simply be compiled.

<DirectoryName>.sim is a simulation descriptor file and it contains a list of names that are either .lib, .vhd, or no extension. The .lib and .vhd extensions are handled in the same fashion as in a file descriptor. If a name does not contain an extension, it is assumed to be a library unit for simulation. Currently any text following the library unit is passed to the simulator.

Release History

For the release history see, CHANGELOG.md

Participating and Project Organization

The OSVVM project welcomes your participation with either issue reports or pull requests. For details on how to participate see

You can find the project Authors here and Contributors here.

More Information on OSVVM

OSVVM Forums and Blog: http://www.osvvm.org/
SynthWorks OSVVM Blog: http://www.synthworks.com/blog/osvvm/
Gitter: https://gitter.im/OSVVM/Lobby
Documentation: Documentation for the OSVVM libraries can be found here

Copyright and License

See the License for the specific language governing permissions and limitations under the License.