Foundation

The basis of this tutorial are all documents and descriptions of easier-uvm from Doulos.

https://www.doulos.com/knowhow/systemverilog/uvm/easier-uvm/

This tutorial shows only the changes and enhancements in the version written in Python.

We assume that we have a module or entity description.

In the tutorial, this is the VHDL entity of a master-slave-config_control component.

```
library IEEE;
use IEEE.std_Logic_1164.all;
use IEEE.numeric_std.all;

entity ms_cfg_ctrl is
  generic (
     g_data_width : integer:= 8;
     g_conf_width : integer:= 16
    );
  port (
    clock          : in  std_logic ;
    reset          : in  std_logic ;
    --master
    ma_get_data    : in  std_logic_vector(g_data_width-1 downto 0) ;
    ma_send_data   : out std_logic_vector(g_data_width-1 downto 0) ;
    o_sel          : out std_logic_vector(3 downto 0)              ;
    o_enable       : out std_logic ;
    --slave
    sl_get_data    : in  std_logic_vector(g_data_width-1 downto 0) ;
    sl_send_data   : out std_logic_vector(g_data_width-1 downto 0) ;
    sel            : in  std_logic_vector(3 downto 0)              ;
    enable         : in  std_logic ;
    --target       :
    o_config_data  : out std_logic_vector(g_conf_width-1 downto 0) ;
    reg_data       : in  std_logic_vector(7 downto 0)
  );
end;
```

Design Description

First, let's create a file named entity_desc.txt

The easiest way is to make a copy of the entity and then reformat it.

```
ENTITY = ms_cfg_ctrl

DEV = d_data_width   8
DEV = d_conf_width   16

PAR = g_data_width   8
PAR = g_conf_width   16

!master active

  ma_get_data    ma_get_data    in  [`d_data_width-1:0]  := '0
  ma_send_data   ma_send_data   out [`d_data_width-1:0]  := '0
  o_sel          ma_sel              out [3:0]  := '0
  o_enable       ma_enable           out        := '0

!slave active

  sl_get_data    sl_get_data    in  [`d_data_width-1:0]  := '0
  sl_send_data   sl_send_data   out [`d_data_width-1:0]  := '0
  sel            sl_sel            in  [3:0]  := '0
  enable         sl_enable         in         := '0


!target passive

o_config_data   config_data    out [`d_conf_width-1:0]  := '0

!register active
```

```
reg_data        config_reg     in  [7:0]   := '0

!clock_reset active reset

  clock         clk
  reset         rst_n
```

The format is an extension of the pin list of easier-uvm.

The pinlist (pinlist.txt) is generated!

ENTITY = <entity name>

DEV = <define-name> <value>

PAR = <generic-name> <value>

!<AGENT name> [passive | active] [reset]

passive or active determines whether the agent has a 'driver' and 'sequencer' or only a 'monitor'.
If input signals are driven, active must be selected.

Reset can only be specified for a single active agent that drives the clock and reset. If it is a bottom-up design, it is advantageous later to have your own agent and not to mix it with another functional block.

If clock_reset is selected, it will be generated!

<VHDL-port-name> <SV-interface-name> {in|out|inout} [range start : range end] := <reset value> ;

<port-name> is the name of the signal in the entity.

<SV-interface-name> the name of the variable in the virtual interface. It is possible to use the same names within an agent as for another agent, e.g. with the same functionality. However, reuse at the top level can lead to complications. Therefore, independent and unique names are to be preferred.

Specify one of the signal directions in|out|inout

The <reset value> is used for outputs during reset checking and for inputs in the driver and sequencer.

The file entity_desc.txt for the first description of the test environment will no longer be used later, but can be used for a modified regeneration.

Template TPL Generation

Second, we call the script uvm_setup.py or gen_uvm.py --setup.

See the script gen.cmd in the example directory.

The script generates a directory named uvm_<entity-name>. In our example, uvm_ms_cfg_ctrl.

Within the directory there is now a template file <agent>.tpl and the standard testbench template common.tpl for each agent.

gen_tb.com as an example for the further call.

pinlist.txt for port mapping.

wave.do as a template for the waveforms.

The DUT directory contains an SV package for the reset values as a template

and the file files.f with the "Design File List" as a template.

In the example, the VHDL file ms_cfg_ctrl.vhd must now be copied to DUT/ or the path to the source code in file.f must be added.

In the example, the file is common_defines.sv with the contents

'define d_data_width 8

'define d_conf_width 16

applied.

The file will be viewed later under

'include'.. /dut/common_defines.sv"

involved. The path must be adjusted and compared with the definitions in common.tpl.

Easier UVM

Easier UVM can now be used.

>perl .. /easier_uvm_gen.pl master.tpl slave.tpl target.tpl register.tpl clock_reset.tpl

The script requires an include directory with all defined files.

The warning is issued

WARNING! SPECIFIED INCLUDE FILE ms_cfg_ctrl_inc_test.sv NOT FOUND

Continue? (y/n) [n]

The required include files must now be created.

gen_uvm (in development version 0.1.0)

The python script gen_uvm.py creates all files:

.. /.. /uvm_scripts/gen_uvm.py master.tpl slave.tpl target.tpl register.tpl clock_reset.tpl

The first query for <entity-name>_pkg.sv must be answered with y. The file can then be compared or merged with <entity-name>_pkg.svh in the DUT directory.

Now all files for the agents and the top level are created under include/.

In contrast to easier_uvm, a separate directory is created for each agent and top level.

If the files are available, the generation is skipped in each case.

The testbench is then generated under <project or enity name>_tb/

Individual automation

Global variables are stored in the header_cfg.py file.

The values must be adjusted:

PROJECT_NAME = Name of the entity or a project

Data that represents something like an imprint:

copyright	= company or person
author	= Who created the TB
email	= of the author
tel	= the author, the company
dept	= Department, Department, Company
company	= Company
year	= Date of Project Begin
version	= of whatever

clock	= port name of the clock - important even if not available
reset	= port name of the reset - important even if not available
clock_reset	= Name of the agent module for the clock-reset driver

json_enable	= 0 if no database image is to be written, 1 if it is
script_path	= "..." complete or indirect path to the uvm scripts. In the case of an indirect path, the directory of the call must be assumed.

tool = "perl" or sys.executable if python is used

genscript = "easier_uvm_gen.pl" or "gen_uvm.py"

script_path = environ.get("GEN_UVM_PATH", join("..","..",""uvm_scripts"))

The environment variable GEN_UVM_PATH should be set otherwise the script will be set to .. /.. /uvm_scripts/ expected.

compatible = 1 # 0 -> not compatible : 1 -> compatible to easier_uvm

Example

Start cmd.exe or Terminal

Command python .. Run /uvm_scripts/uvm_setup.py. Attention Python version 3.x

```
---------- pinlist ----------
generate TOP
Interface: top
Interface: master_if_0
Interface: slave_if_0
```

```
Interface: target_if_0
Interface: register_if_0
Interface: clock_reset_if_0
generate shell script gen_tb.cmd with :call "C:\Program Files\Python310\python.exe"
"D:/github.com/gen_uvm/uvm_scripts\gen_uvm.py" master.tpl slave.tpl target.tpl
register.tpl clock_reset.tpl
Generation done!
-----------------------------------------------
```

The directory uvm_ms_cfg_ctrl was created and the JSON file was written uvm_ms_cfg_ctrl.json.

```
.../example/uvm_ms_cfg_ctrl
                |- dut
                | common_defines.sv
                |- files.f
                | ms_cfg_ctrl_pkg.svh
                | clock_reset.tpl
                |- common.tpl
                | gen_tb.cmd
                |- master.tpl
                | pinlist.txt
                |- register.tpl
                |- slave.tpl
                |- target.tpl
                | uvm_ms_cfg_ctrl.json
                | wave.do
```

When these first steps are successfully completed, the generated script should be present gen_tb.cmd or gen_tb.sh. These are now always to be called up or their content. Before that, copy the file ms_cfg_ctrl.vhd to dut from the example directory.

The tpl files are just one example of which files and settings could be used.

All possibilities are shown in the examples of easier_uvm of Doulos.

A more detailed list for gen_uvm will follow!

With EasierUVM, the encoding of the defined include files is now the first task of the verifier.

Template Generation

gen_uvm has a built-in template class, "class uvm_template(object):". This generates all include files that have been defined in the TPL files. The content is:

```
//-----------------------------------------------------------------
// AGENT NAME: {agent_name}
// PATH : {ref["project"]+"/tb/include/"+ref['agent_name']}
// GENERATOR : {me} for {name}
//-----------------------------------------------------------------
```

If a file uvm_template.py exists, it is used as a template generator. With a project-related file e.g myproject_template.py the generation can be customized.

In the configuration file header.cfg.py, the entry for this is

tmplt_include_file = 'myproject_template'

The template_class file can have any <name>[.py]. It is named header_cfg.py in the configuration (tmplt_include_file = "myproject_template") or in the TPL file common.tpl (tmplt_include_file = uvm_template)

If it is not possible to load the template class, the internal class uvm_template is used.

The file uvm_template.py exists as a template. When creating a project-related class, copy lines 1 to 75 and develop your own subroutines.

The following definitions are available as working modules:

```
template_prototype
driver_inc_inside_class
driver_inc_after_class
if_inc_before_interface
if_inc_inside_interface
monitor_inc_inside_class
monitor_inc_after_class
trans_inc_before_class
trans_inc_inside_class
trans_inc_after_class
agent_seq_inc
tb_inc_before_run_test
agent_scoreboard_inc_inside_class
agent_scoreboard_inc_after_class
common_define
agent_copy_config_vars
top_env_scoreboard_inc_inside_class
top_env_scoreboard_inc_after_class
top_env_append_to_connect_phase
agent_cover_inc_inside_class
test_inc_inside_class
```

QuestaSim Support

The simulator QuestaSim (ModelSim) is especially supported because it was used in development.

A wave.do file is created once and then copied to .../<project>_tb/sim each time it is called. Make edits only to this file.

run_batch.cmd or run_batch.sh start a simulation in batch mode.

r un_gui.cmd or run_gui.sh start the simulation in the GUI

In the directory .../<project>_tb/sim, the files are updated every time gen_uvm.py is called. These are:

```
batch.do     controls batch simulation
common.tcl   TCL proc() collection for GUI/batch simulation
compile.do   only controls generation and compilation
compile.tcl  compiles the design and testbench
gui.do       controls GUI simulation
vgui.cmd     starts GUI Simulation (WIN)
vgui.do      used by vgui.cmd or vgui.sh
vgui.sh      starts GUI Simulation (LINUX)
wave.do      wave forms, copy from .. /.. /
```

All simulation results can also be found in the directory .../<project>_tb/sim.