Grundlage

Die Grundlage dieses Tutorial sind alle Dokumente und Beschreibungen von easier-uvm von Doulos.

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

Dieses Tutorial zeigt nur die Änderungen und Erweiterungen in der in Python geschriebenen Version.

Wir gehen davon aus eine Modul- oder Entity Beschreibung vorliegen zu haben.

In dem Tutorial ist dies die VHDL Entity einer master-slave-config_control Komponente.

```vhdl
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 Beschreibung

Als erstes erstellen wir eine Datei mit dem Namen entity_desc.txt

Am einfachsten eine Kopie der Entity und dann umformatieren.

```
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
```

Das Format ist eine Erweiterung der Pin Liste von easier-uvm.

Die Pinlist (pinlist.txt) wird generiert!

ENTITY = <entity name>

DEV = <define-name> <value>

PAR = <generic-name> <value>

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

passive oder active bestimmt ob der Agent einen ‚Driver' und ‚Sequencer' oder nur einen ‚Monitor' hat.
Werden Eingangssignale getrieben muss active gewählt werden.

Reset kann nur bei einen einzigen aktiven Agent angegeben werden der den clock und reset treibt. Wenn es sich um ein bottom-up Design handelt ist es später von Vorteil dafür einen eigenen Agent zu haben und nicht mit einem andern funktionalen Block zu vermischen.

Wird clock_reset gewählt, wird dieser generiert!

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

<port-name> ist der Name des Signals in der Entity.

<SV-interface-name> der Name der Variablen im virtuellen Interface. Es ist möglich innerhalb eines Agent die selben Namen wie bei einem anderen Agent zu verwenden, zB. bei gleicher Funktionalität. Bei Wiederverwendung auf Top-Level kann es aber zu Komplikationen kommen. Deshalb sind eigenständige und eindeutigen Namen zu bevorzugen.

Es ist eine der Signalrichtungen in|out|inout anzugeben

Der <reset value> wird bei Ausgängen bei der Reset Überprüfung und bei Eingängen im Treiber und Sequenzer verwendet.

Die Datei entity_desc.txt zur ersten Beschreibung der Testumgebung wird später nicht mehr verwendet, kann aber für eine geänderte Neugenerierung benutzt werden.

Template TPL Generierung

Als zweites rufen wir das script uvm_setup.py auf oder gen_uvm.py --setup.

Siehe im Verzeichnis example das script gen.cmd .

Das Script generiert ein Verzeichnis mit dem Namen uvm_<entity-name>. In unserem Beispiel uvm_ms_cfg_ctrl.

Innerhalb des Verzeichnisses ist nun für jeden Agent eine Template Datei <agent>.tpl und das Standard Testbench Template common.tpl vorhanden.

gen_tb.com	als Beispiel für den weitern Aufruf.
pinlist.txt	für das Portmapping.
wave.do	als Vorlage für die Waveforms.

Das Verzeichnis DUT beinhaltet ein SV-Package für die reset-Werte als Template

und das File files.f mit der „Design File List“ ebenfals als Template.

Im Beispiel muss nun noch die VHDL Datei ms_cfg_ctrl.vhd nach DUT/ kopiert oder der Pfad zum Sourcecode in file.f ergänzt werden.

Im Beispiel ist die Datei common_defines.sv mit dem Inhalt

```
`define d_data_width 8

`define d_conf_width 16
```

angelegt.

Die Datei wird später unter

```
`include "../dut/common_defines.sv"
```

eingebunden. Der Pfad ist anzupassen und mit den Definitionen in common.tpl abzugleichen.

Easier UVM

Easier UVM kann nun benutzt werden.

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

Das Script setzt ein include-Verzeichnis voraus mit allen definierten Dateien.

Es wird die Warning ausgegeben

```
WARNING! SPECIFIED INCLUDE FILE ms_cfg_ctrl_inc_test.sv NOT FOUND
```

Continue? (y/n) [n]

Es müssen nun die erforderlichen include dateien angelegt werden.

gen_uvm (in der Entwicklung Version 0.1.0)

Das python script gen_uvm.py legt alle Dateien an:

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

Die erste Abfrage nach <entity-name>_pkg.sv ist mit y zu beantworten. Die Datei kann dann mit <entity-name>_pkg.svh im Verzeichnis DUT verglichen bzw. zusammengeführt werden.

Nun werden alle Dateien für die Agents und dem Top Level unter include/ erstellt.

Im Unterschied zu easier_uvm wird für jeden Agent und Toplevel ein eigenes Verzeichnis angelegt.

Sind die Dateien vorhanden wird die Generierung jeweils übersprungen.

Anschließend wird die Testbench unter <project oder enity name>_tb/ generiert

Individuelle Automatisation

Globale Variablen sind in der Datei header_cfg.py hinterlegt.

Die Werte sind anzupassen:

PROJECT_NAME = Name der Entity oder eines Projects

Daten die so etwas wie ein Impressum darstellen:

copyright	=	Firma oder Person
author	=	Wer hat die TB erstellt
email	=	des Autors
tel	=	des Autors, der Firma
dept	=	Department, Abteilung, Firma
company	=	Firma
year	=	Datum von Projekt Begin
version	=	von was auch immer

clock	=	Portname des Taktes - wichtig auch wenn nicht vorhanden
reset	=	Portname des Reset - wichtig auch wenn nicht vorhanden
clock_reset	=	Name des Agent Moduls für den clock-reset Treiber
json_enable	= 0	wenn kein Datenbank-Image geschrieben werden soll, 1 wenn doch
script_path	= „…“	vollständiger oder indirekter Pfad zu den uvm-scripten. Bei indirektem Pfad ist vom Verzeichnis des Aufrufes auszugehen.

tool = "perl" oder sys.executable wenn python verwendet wird

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

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

Die Environment Variable GEN_UVM_PATH sollte gesetzt sein ansonsten wird das script in ../../uvm_scripts/ erwartet.

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

Beispiel

cmd.exe oder Terminal starten

Befehl `python ../uvm_scripts/uvm_setup.py` ausführen. Achtung 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!
-----------------------------------------------
```

Es wurde das Verzeichnis uvm_ms_cfg_ctrl erstellt und die JSON Datei uvm_ms_cfg_ctrl.json geschrieben.

```
.../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
```

Wenn diese ersten Schritte mit Erfolg beendet wurden sollte das generierte script gen_tb.cmd oder gen_tb.sh vorhanden sein. Diese sind nun immer aufzurufen bzw deren Inhalt. Vorher ist aus dem Verzeichnis example die Datei ms_cfg_ctrl.vhd nach dut zu kopieren.

Die tpl Dateien sind nur ein Beispiel welche Dateien und Einstellungen verwendet werden könnten.

Alle Möglichkeiten sind in den Beispielen von easier_uvm von Doulos aufgezeigt.

Eine genauere Aufstellung für gen_uvm folgt!

Bei EasierUVM ist nun die Kodierung der definierten include-files die erste Aufgabe des Verifizierers.

Template Generierung

gen_uvm hat eine eingebaute template class, „class uvm_template(object):“. Diese generiert alle include-Dateien die in den TPL-Dateien definiert wurden. Der Inhalt ist:

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

Ist eine Datei uvm_template.py vorhanden wird diese als Templategenerator verwendet. Mit einer Projektbezogenen Datei z.B myproject_template.py kann die Generierung angepasst werden.

In der Konfigurationsdatei header.cfg.py ist der Eintrag dafür

```
tmplt_include_file = 'myproject_template'
```

Die Datei der template_class kann jeden <namen>[.py] besitzen. Sie ist in der Konfiguration header_cfg.py benannt (tmplt_include_file = "myproject_template") oder in der TPL Datei common.tpl (tmplt_include_file = uvm_template)

Ist das Laden der template class nicht möglich wird die interne class uvm_template verwendet.

Es ist die Datei uvm_template.py als Vorlage vorhanden. Bei der Erstellung einer projectbezogenen Klasse ist die Zeile 1 bis 75 zu kopieren und die eigenen Unterprogramme entwickeln.

Folgende Definitionen sind als funktionierende Module vorhanden:

```
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 Unterstützung

Der Simulator QuestaSim (ModelSim) wird besonders unterstützt da er in der Entwicklung benutzt wurde.

Es wird eine wave.do Datei einmalig erstellt und dann bei jedem aufruf nach …/<project>_tb/sim kopiert. Bearbeitungen nur an dieser Datei vornehmen.

run_batch.cmd oder run_batch.sh starten eine Simulation im Batch-Mode.

run_gui.cmd oder run_gui.sh starten die Simulation in der GUI

Im Verzeichnis …/<project>_tb/sim werden bei jedem Aufruf von gen_uvm.py die Dateien upgedated. Dies sind:

```
batch.do                    kontrolliert die Batch-Simulation
common.tcl                  TCL proc() Sammlung für die GUI/Batch Simulation
compile.do                  kontrolliert nur Generation und Kompilation
compile.tcl                 kompiliert das Design und die Testbench
gui.do                      kontrolliert die GUI-Simulation
vgui.cmd                    startet GUI Simulation (WIN)
vgui.do                     von vgui.cmd bzw. vgui.sh verwendet
vgui.sh                     startet GUI Simulation (LINUX)
wave.do                     wave forms, kopie aus ../../
```

Alle Simulationsergebnisse sind ebenfalls im Verzeichnis …/<project>_tb/sim zu finden.