|  |
| --- |
| **Internal Specification** |

|  |
| --- |
| **Unit verification environment**  **without CPU**  (v1.3) |

|  |
| --- |
| **Summary:** |
| This document describes the internal specification of new unit verification environment without CPU. |

. Blue characters: Current design (v2018\_11\_14) supports these updated points.

. Red characters: Current design (v2018\_11\_14) have not supported these updated points yet.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference Manuals** | | | | |
| **No.** | **Title name** | **Document number** | **Description** | **Path** |
| 1 | Simulation Guide (Rev1.5) | USR-SLD-08009 | Simulation Guide  (***File***: USR-SLD-08009-Simulation\_guide.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/USR/2009 |
| 2 | Test pattern creation Guide (Rev1.4) | Test pattern creation guide  (***File***: USR-SLD-08009-Test\_pattern\_creation\_guide.odt/.pdf) |
| 3 | Environment Generator Guide (Rev1.01) | Verification Environment Generator Guide  (***File***: USR-SLD-08009-Gen\_Env\_Guide.odt/.pdf) |
| 4 | Top Module Generation Script of TEST (Rev 1.0) | USR-SLD-10002 | User Manual of Top Module Generation Script of RESL-X  (***File***:  USR-SLD-10002.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/USR/2010 |
| 5 | Internal specification of TEST environment | INT-SLD-08009 | Internal specification of simulation environment  (***File***:  INT-SLD-08009.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/USR/2009 |
| 6 | Improvement specification for TEST Top Module Generator (Rev 1.1) | VRF-SLD-11014 | Verification Specification of Top Module Generation Script of RESL-X  (***File***:  VRF-SLD-11014.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/VRF/2011 |
| 7 | Model Verification Environment (Rev 1.3) | INT-SLD-15003  (D-SLD-CMN-0032-01) | Internal Specification of Model Verification Environment (uses Tracer)  (***File***: INT-SLD-15003.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/INT/2015 |
| 8 | Transactor Enhancement (Rev 1.1) | INT-SLD-14003 | Internal Specification of Transactor model  (***File***: INT-SLD-14003.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/INT/2014 |
| 9 | Support 64bit mode building (Rev1.0) | REQ-SLD-13027 | The requirement for Unit verification environment without CPU (Support 64bit mode building)  (***File***: REQ-SLD-13027\_ASIC-PF\_64bit.pptx) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/REQ/2013 |
| 10 | Unit verification environment without CPU phase 2 (Rev1.1) | REQ-SLD-17004 | The requirement for Unit verification environment without CPU (phase 2)  (***File***: REQ-SLD-17004\_UnitVerificationEnv.pptx) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/REQ/2017 |
| 11 | Support 64 bit mode building ASIC models (Rev 1.0) | VRF-SLD-13027 | Verification specification for Unit verification environment without CPU (Support 64bit mode building)  (***File***: VRF-SLD-13023.odt/.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/VRF/2013 |
| 12 | Unit verification environment without CPU phase 2.1 | REQ-SLD-17008 | The requirement for Unit verification environment without CPU (phase 2.1)  (***File***: REQ-SLD-17008\_UT env\_wo CPU ph2.1.pdf) | **DMS:**  Documents/1. General Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/01\_Project\_Document\_Management/REQ/2017 |
| 13 | Idea of new Unit verification environment | Idea of new UT env | The idea for new UT environment  (***File***: Idea of new UT env.pptx) | **Email:**  [prj-sld:24451] Re: About UT env  **Storage**:  [\\rvc-vnas-01\\ipp\project\2017\rel\17008\_New\_Unit\_Env\_p2.1\01\_Input](file:///\\rvc-vnas-01\\ipp\project\2017\rel\17008_New_Unit_Env_p2.1\01_Input) |

Table of Contents

[1. Summary 6](#_Toc530959715)

[1.1. Introduction 6](#_Toc530959716)

[1.2. Block diagram of new Unit test environment skeleton 9](#_Toc530959717)

[2. Implementation of unit environment 12](#_Toc530959718)

[2.1. Environment skeleton structure 12](#_Toc530959719)

[2.2. Source code copying script 13](#_Toc530959720)

[2.3. Test pattern preparation (“pat” folder) 13](#_Toc530959721)

[2.4. Model implementation (“tb” folder) 23](#_Toc530959722)

[2.5. Environment configuration setting (“scripts” folder) 27](#_Toc530959723)

[2.6. Top module generation (“gen\_env” folder) 30](#_Toc530959724)

[2.7. Simulation structure preparation (“gen\_sim” folder) 31](#_Toc530959725)

[2.8. Simulation result report (“check\_result” folder) 32](#_Toc530959726)

[2.9. Report generation (“gen\_report” folder) 32](#_Toc530959727)

[2.10. Run all script (“run\_all” folder) 32](#_Toc530959728)

[2.11. UT environment generation script 33](#_Toc530959729)

[3. Others 33](#_Toc530959730)

[3.1. Data location 33](#_Toc530959731)

[3.2. Notes 33](#_Toc530959732)

**Index of Figures**

[Figure 1.1: Block diagram of model development improvement 7](#_Toc530959733)

[Figure 1.2: Block diagram of new UT environment 9](file:///F:\Users\yennguyen\02_SLD\ipd2_sld_17008_unit_env\02_Output\DOC\INT-SLD-17008.docx#_Toc530959734)

[Figure 2.1: Environment skeleton structure 12](#_Toc530959735)

[Figure 2.2: copy\_release\_file.csh 13](file:///F:\Users\yennguyen\02_SLD\ipd2_sld_17008_unit_env\02_Output\DOC\INT-SLD-17008.docx#_Toc530959736)

[Figure 2.3: Skeleton of test pattern file 14](#_Toc530959737)

[Figure 2.4: Example of test pattern created by verifier 14](#_Toc530959738)

[Figure 2.5: “setTimeResolution” function 15](#_Toc530959739)

[Figure 2.6: “setTimeResolution” function 16](#_Toc530959740)

[Figure 2.7: “OpenFile” function 16](#_Toc530959741)

[Figure 2.8: “getAddress” function 17](#_Toc530959742)

[Figure 2.9: “wait” function 17](#_Toc530959743)

[Figure 2.10: “end\_tm” function 17](#_Toc530959744)

[Figure 2.11: “pass\_bp” function 17](#_Toc530959745)

[Figure 2.12: “fail\_bp” function 18](#_Toc530959746)

[Figure 2.13: “msg\_set” function 18](#_Toc530959747)

[Figure 2.14: “msg\_chk” function 18](#_Toc530959748)

[Figure 2.15: “reg\_acc” function 19](#_Toc530959749)

[Figure 2.16: “port\_acc” function 20](#_Toc530959750)

[Figure 2.17: Example of user defined content in common.py script 20](#_Toc530959751)

[Figure 2.18: Example of user defined content in Python iodefine model file (current) 20](#_Toc530959752)

[Figure 2.19: Example of user defined content in in Python iodefine model file (enhancement) 21](#_Toc530959753)

[Figure 2.20: MSG variable in chk\_msg\_py.skl file 22](#_Toc530959754)

[Figure 2.21: MSG variable in chk\_msg.py file 22](#_Toc530959755)

[Figure 2.22: Example of message check report 23](#_Toc530959756)

[Figure 2.23: Example of constructor of control port model for table of ports 24](#_Toc530959757)

[Figure 2.24: IssueTransFromFile thread of control port model 25](#_Toc530959758)

[Figure 2.25: Python I/F model header file 25](#_Toc530959759)

[Figure 2.26: Python I/F model implementation file 26](#_Toc530959760)

[Figure 2.27: test.vcxproj skeleton file 31](#_Toc530959761)

**Index of Tables**

[Table 1.1: Feature list of new unit verification environment 6](#_Toc530959762)

[Table 2.1: REL environment setting 27](#_Toc530959763)

[Table 2.2: RVC environment setting 28](#_Toc530959764)

[Table 2.3: Environment setting 29](#_Toc530959765)

[Table 2.4: Table of updated variables 30](#_Toc530959766)

[Table 2.5: Table of updated functions 30](#_Toc530959767)

# Summary

## Introduction

The RESLX verification environment has the following issues:

1. The verifier have to check the huge simulation log file by visual confirmation. So, it takes much time to check the result and it may make mistake.
2. For verification on 64bit environment, we replaced SH4A ISS with the transactor model. So, we have to run the simulation with 32bit environment which has SH4A ISS to get the transaction information beforehand. Or we have to write the transaction information by hand.
3. The transactor model reads the transaction information one by one and issues transaction to DUT. However, we can't describe loop and branch structure in the transaction information.
4. Racing issue: the DUT model output value after delta time of checking output value => unexpected result while it is correct. (This issue is detected during 64-bit regression test)

The new unit verification environment is developed to solve above issues. It is developed based on current RESLX environment. Only updated/modification points is described in this documents. The following table is supported feature list.

Table 1.1: Feature list of new unit verification environment

|  |  |  |
| --- | --- | --- |
| **No.** | **Feature** | **Implementation** |
| 1 | Verification mode | |
| 1.1 | 32-bit compilation & simulation | Use transactor instead of CPU |
| 1.2 | 64-bit compilation & simulation |
| 2 | Test pattern | |
| 2.1 | Regression test: Format is transaction information which is generated by monitor model in 32 bit mode | Keep current transactor model |
| 2.2 | New verification: Format is supported for loop & branch structure and supported for checking result automatically | ~~Revise gen\_trans script to generate from new format (easy to write for verifier) to current format (easy to read for transactor)~~  Use Python interpreter to write Python patterns. When running simulation, this patterns will generate transaction log file for the simulation. |
| 2.3 | Read/Write 64-bit port value | Supported by verification model (control port model) by ~~using two 32-bit registers (\*)~~ generating port\_acc log to store all port accessing with supported type. |
| 2.4 | Model Python I/F |  |
| 3 | Check result | |
| 3.1 | Register value | Supported (takes 2 clock cycles for each check) |
| 3.2 | Port value | Supported via control port model |
| 3.3 | Message | ~~Revise gen\_trans~~ Python test pattern will generate chk\_msg.py script to generate expected message from test pattern |
| 3.4 | Timing | Simulation time when writing register is different between AT/LT mode, so the timing check |
| 4 | Coverage | |
| 4.1 | 64-bit coverage | Current dummyins.x cannot be run on 64-bit server 🡺 ~~should find out the source code to compile for 64-bit~~. RVC fixed and released to RVC /common/appl |

~~(\*): Transactor can store model pointers to invoke its APIs for port read/write or handeCommand~~

The new unit environment will be used for different models. So the unit environment described in this document is the skeleton environment. One script is implemented to generate the corresponding environment for model.

The Figure 1.1 show the whole view of model development improvement.

**B. Environment preparation**

**(create\_unit\_env.csh)**

Input data

Environment skeleton

E. Register address map preparation

New unit verification environment

D. Driver preparation

C. Control\_port preparation

[Input]

**A. Code preparation**

Verification models

**Transactor**

DUT

**SHwy 32 bits**

**UT Environment**

ShPythonAPI

32 bits

Red

: New develop

Purple

: Enhance

White

: Reuse

Green

: Inputs by user

Figure 1.1: Block diagram of model development improvement

***Explanation***:

* The whole view of model development improvement includes:
  1. Code preparation
  2. Environment preparation
  3. Control port model preparation
  4. Driver preparation
  5. Register address map preparation
* **Code preparation**: Designer will prepare code by using some supported tools such as: register I/F generator, handle Command generator, FSM generator. A common generator should be considered to generate ports, data members, TLM I/F, common code (declaration, initialization, reset, clock setting, …) also. This content will be consider in other project.
* **Environment preparation**: Verification environment includes 2 parts:
  + The first one is common part applied for all models (such as scripts, folder structures, skeleton files, …) is the script used to generate executable verification environment from input of user and environment skeleton. This part is called as Unit test environment skeleton.
  + The second one is the specified part for specified model (such as model name, test patterns, top.sci.skl. This part will be inputted by verifier.
  + A generation script (create\_unit\_env.csh) is created to generate executable environment (environment is ready to run simulation and debug) from the two above parts.
* **Control port preparation**: Verifier will prepare a control model used to verify target model. Its legacy functions are:
  1. Control model input ports and recognize changes of model output ports.
  2. Invoke handleCommand during simulation.
  3. Issue TLM transaction which cannot be used by using verification environment with CPU.
  4. Reuse for SC-HEAP environment.

A control port generation script was created to generate control port model for a), b) & d) functions. It uses register declaration to configure the port and handle Command request information. Its limitations are:

* + Not support for ‘double’ type.
  + Multiple-steps (commands) for setting ports/handleCommand. For example, to write a 54-bits port, user should write to 3 registers: its value to 2 registers, and write to control register to notify a port writing.

During this project, the control port model is considered to use port information file input (similar to transactor) to read/write port. Above limitations are eliminated.

* **Driver preparation**: Driver preparation is used to declare APIs to read/write registers/ports. It is necessary for the verification because of the following reasons:
  + Easy to understand test patterns
  + Easy to modify if the setting specification is changed. In this case, we just need to modify the driver only, we can reduce workload to modify all related test patterns.

However, current driver is depended on verifier. It should be considered as required generation tool for common usage.

* **Register address map preparation**: In verification environment with CPU, its implementation is in iodefine\_<model\_name>.h file. This file is generated by register I/F generator. However, there are two issues with this implementation:
  + The register address is not checked carefully by applying this way. Eg: issue with BIST\_U2A model related to address offset.
  + In new unit environment without CPU, test pattern is Python script, so the register address map should be implemented in Python language also.

To reduce manual cost, it should be considered to generate also from the I/F specification (not register I/F info file).

This specification is focused to describe the environment preparation.

Next section describes block diagram of new Unit test environment skeleton.

## Block diagram of new Unit test environment skeleton

The Figure 1.2 show the block diagrams of new verification environment skeleton.

Figure .: Block diagram of new UT environment

**UT Environment**

**SHwy 32 bits**

DUT

**Transactor**

Verification models

common.py

chk\_msg\_py.skl

<tm\_name>.py

[pat]

**2. Test Pattern Preparation**

32 bits

TM\_path.txt

[pat]

[results]

results

[log]

log

\*\_info.txt

top.sci.skl

result.txt

[results]

reports

[reports]

gen\_sim.pl

[scripts\_linux/

gen\_sim]

[scripts\_linux/

gen\_env]

[scripts\_linux/

gen\_env/Input]

gen\_report.pl

[scripts/gen\_report]

check\_result.py

[scripts\_linux/

check\_result]

run\_sim.sh

[sim]

run\_sim.sh

**4. Simulation result report**

**1. Top module generation**

[sim/category1/…]

chk\_msg.py

top\_gen.pl

[pat/0\_common\_files]

ShPythonAPI

**3. Environment simulation**

Red

: New develop

Purple

: Enhance

White

: Reuse

Green

: Inputs by user

transaction\_log.txt

port\_acc.txt

***Explanation***:

* The new unit environment includes 4 main parts:

1. Top module generation
2. Test pattern preparation
3. Environment simulation
4. Simulation result report.

* **Top module generation**: generate top module (test bench) of UT environment. Top module generator was implemented by Python language (at /shsv/sld/ipp/Repository/R-IP/Common/TopModGen.git). This script is reused and put at “scripts/gen\_env” in unit verification environment. Main script is top\_gen.py. Its input files are:
  + <model>\_info.txt: model port information file.
  + top.sci.skl: system configuration file in which system connection, address map, SHwy bus setting are described.
  + test.vcxprj.skl: VC++ project skeleton file.
  + It is enhanced in this project for Python I/F support. Please refer to section 2.6 for detail enhancement.
* **Test pattern preparation**: test patterns include:
  + Python test pattern (.py): this is main test pattern (as “.cpp” pattern in RESLX environment). It is newly developed in this project and is implemented by using Python language for user convenience. It is described for registers, ports, timing, message verification. Common APIs are developed in “common.py” file. Register address declaration file (iodefine\_<model>.py) and drivers (<model>\_drv.py) are prepared based on model specification. It is considered to be generated from the I/F specification also.
  + Handle Command pattern (.cmd): this file is used to verify handle Command as same as RESLX environment.
  + Please refer to section 2.3 for the detail.
* **Environment simulation**: environment generation script is revised from RESLX environment ~~without modification~~ (gen\_sim.pl & create\_batch.pl) to use for both Linux and Windows. It is used to generated simulation structure and scripts to run simulation.
  + When simulation, Python test pattern will be run and generate transaction\_log.txt for transactor to issue transactions to read/write register, and to dump messages. Please refer to section 2.4 for the detail for transactor enhancement.
  + Python test pattern also generate message check script chk\_msg.py to check message after simulation.
  + After simulation, ~~expected message~~ message check script (~~.msg~~ chk\_msg.py) ~~for each test pattern will be generated for message checking when checking result~~ is invoked to check if the message is correct and report the result in chk\_msg.rpt file.
  + Message check script is kept in “sim” folder as <pattern\_name>\_chk\_msg.py and message report is stored in “results” folders.
* **Simulation result report**: there are two kinds of reports:
  + Report result of each test pattern: After simulation of each test pattern, the check result script will read expected message (generated by test pattern) with simulation log file to judge the result. ~~Check result script (check\_result.pl) was developed in RESLX verification environment. However, it is revised in this project to check message in log file (check\_log.pl) if expected log file is not existed and corresponding expected message (.msg) of each test patterns is existed.~~ Check result script is implemented by Python to check the simulation result and message result based on log file and message report file. Please refer to section 2.7 for detail description.
  + Report result of all test patterns: gen\_report (gen\_report.pl) script is used to generate summary report of all test patterns. This script is enhanced from RESLX verification to support both Linux and Windows.

# **Implementation of unit environment**

## Environment skeleton structure



Figure 2.1: Environment skeleton structure

Detail explanation of each folder is described in the bellow sections:

## Source code copying script

* This script is used to copy source code into 2 folders to release to REL/SWMCS:
  + <MODEL\_NAME> folder and
  + PYTHON\_IF folder
* The following figure is the content of the script:

Figure 2.2: copy\_release\_file.csh

#! /bin/csh -f

set IPDIR=*<MODEL\_NAME>*

set PYTHONDIR=PYTHON\_IF

set model\_prefix=*<model\_name>*

mkdir $PYTHONDIR

mkdir $IPDIR

cp src/PY\_\*.cpp src/PY\_\*.h $PYTHONDIR

cp src/\*${model\_prefix}\*.cpp src/\*${model\_prefix}\*.h $IPDIR

cp release\_note.txt $IPDIR/HISTORY.txt

***Explanation***:

* This script is skeleton file. Italic contents in script will be updated after running create\_unit\_env.csh to generate executable environment.
* release\_note.txt should be prepared in released environment. (Note for user)

## Test pattern preparation (“pat” folder)

Python test pattern will be executed when running simulation and generate the following files:

* transaction\_log.txt file: this text file contains all commands for transactor, such as to RD, WR, DR, DW, WAIT, MSG. It is input file of transactor model to read/write registers
* port\_acc.txt file: this text file contains all command for control port model, such as RD, WR, WAIT. It is input file of control port model to read/write ports.
* chk\_msg.py: this is Python script used to check message after simulation. This script is invoked by check\_result.py after running simulation.

To support generating these three files, the common functions are implemented in common.py script. The following sections will describes template of test pattern, common APIs and message check specification.

### Template of test pattern

The template of test pattern is described in the following figure:

|  |
| --- |
| import sys  Import common.py script  import os  import traceback  sys.path.append('./')  User declaration  from common import \*  ### variable declaration  ### common setting function call. Eg: setTimeResolution, setClockPeriod  Test pattern content  try:  ### Content of TMs  ### driver functions, common APIs can be invoked  end\_tm()  except Exception:  traceback.print\_exc()  print ("SYNTAX ERROR in Python pattern. Please check and fix pattern first.\n")  os.remove("transaction\_log.txt")  sys.exit() |

Figure 2.3: Skeleton of test pattern file

The following figure is an example of test pattern (.py) file

|  |
| --- |
| from common import \*  Python syntax:  variable declaration  scan\_group = 5  channel\_num = 2  setClockPeriod(10)  Python common setting functions  setTimeResolution("ns")  try:  wait(10)  CPSetClock()  wait(10\*clock\_period)  w\_val = [0xFFFFFFFF, 0x00000000, 0x55555555, 0xAAAAAAAA]  r\_val\_01= [0x00000000, 0x00000000, 0x00000000, 0x00000055, 0x0000002A]  r\_val = [0x00000000, 0x0000002A, 0x00000000, 0x00000055, 0x0000002A]  msg\_id = 0  for channel in range (0,channel\_num):  for index in range (0,scan\_group):  for x in range (0,4):  for y in range (0,4):  msg\_set ("=== MSG ID " + "{0:05d}".format(msg\_id) + " Invalid check")  addr = getAddress("ADC[" + str(channel) + "].ADENDP[" + str (index) + "]") + x  reg\_acc ( "WR", addr, 1, (w\_val[y] >> (x\*8)) &0xFF)  msg\_chk("=== MSG ID " + "{0:05d}".format(msg\_id) + " Invalid check", "Invalid access address", 2\*clock\_period, 1 ,False)  msg\_id += 1  wait((10 + clock\_period) \* 2)  port\_acc("WR", "preset\_n", 0)  msg\_chk("port preset\_n", "Initialize", -1, 784, True)  end\_tm() |

Figure 2.4: Example of test pattern created by verifier

***Explanation***:

* Test pattern will import common APIs file which is put in “pat/ 0\_common\_files/” folder. Its specification is described in the next section
* The syntax of test pattern is Python syntax.
* Handle Command can be invoked in test pattern as Python I/F. However, it cannot be issued during simulation time because it is not invoked in a SC thread.

### Common Python APIs

This Python file (common.py) is put in “pat/ 0\_common\_files/” folder. This common Python file will be imported in test pattern for general purposes. Common variables and functions are declared as following:

* Common variables :
  + CurTime: current simulation time variable. Its initial value is “0”.
  + resolution: Time resolution: this variable is used to set up time resolution for test pattern. It is “ns” by default.
  + clock\_period: clock period value, this value is used to increase the current simulation to (2\*clock\_period) after reading/writing register. It is “20” by default.
  + model\_base\_addr: dictionary table to store model based address value (hex value) based on model name (string). It is “{'ICLK' : 0xFF4B0000 }” by default.
  + model\_base\_reg: dictionary table to store register offset address value (hex value) based on register name (string). It is “{}” by default. Refer to section 2.3.3 for the detail.
  + SC\_ZERO\_TIME: variable to notify SC\_ZERO\_TIME for waiting.
  + tm: transaction log file pointer. Python test pattern will generate transaction\_log.txt as input file of transactor model.
  + ptm: port access file pointer. Python test pattern will generate port\_acc.txt as input file of control port model
  + chk\_msg\_info: information for message check (string type). Python test pattern will generate chk\_msg.py file based on this variable content and chk\_msg\_py.skl file.
  + ~~Common message table (MessageTable): the table of message ID and its simulation time. For example in Figure 2.4, this table is: MessageTable[“m1”] = 0. It is empty table at the beginning.~~
  + ~~Register address map table (RegisterTable): table of register and its address.~~
* Common functions:
  + setClockPeriod(period): this function is used to set clock period (change ‘clock\_period’ variable value. If it is not 20, user should call this API in Python test pattern to change the clock period.
    - period: integer number for clock period

|  |
| --- |
| def setClockPeriod(period):  global clock\_period  clock\_period = period |

Figure 2.5: “setTimeResolution” function

* + setTimeResolution(res): this function is used to set time resolution (change ‘resolution’ variable value. If it is not “ns”, user should call this API in Python test pattern to change the time resolution.
    - res: string defined time resolution. Its value should be “fs”, “ps”, “ns”, “us”, “ms” or “s”.

|  |
| --- |
| def setTimeResolution(res):  global resolution  resolution = res |

Figure 2.6: “setTimeResolution” function

* + OpenFile(Filename, Option): this function is used to open file and return the file pointer.
    - Filename: string defined file name
    - Option: string defined access permission. Eg: “r” (read only) or “w” (write only), …

|  |
| --- |
| def OpenFile (Filename, Option):  if ((not os.path.exists(Filename)) and re.search("(r|a\+)", Option) != None):  print("Cannot open " + Filename + " file.\n")  os.remove("transaction\_log.txt")  return None  FilePointer = None  FilePointer = open(Filename, Option)  return FilePointer |

Figure 2.7: “OpenFile” function

* + getAddress(reg): this function is used to get address from register string. It will split register name to calculate model based address and register offset to get the full register address.
    - reg: string to define the register name. Eg: ADC[0].ADENDP[0]

|  |
| --- |
| def getAddress (reg):  instances = reg.split ('.')  level = 0  base\_addr = 0  reg\_addr = 0  inst = ""  for x in instances:  if (x[-1] == "]"):  index = re.sub(r'.\*\[', '', x[:-1])  if (level == 0):  inst = re.sub(r'\[.\*','', x)  base\_addr = model\_base\_addr[inst] + (int(index) \* ADC\_OFFSET)  else:  reg\_name = re.sub(r'\[.\*','', x)  reg\_addr = model\_base\_reg[inst][reg\_name][1] + (int(index) \* model\_base\_reg[inst][reg\_name][2])  else:  if (level == 0):  inst = x  base\_addr = model\_base\_addr[x]  else:  reg\_addr = model\_base\_reg[inst][x][1]  level += 1  return (base\_addr + reg\_addr) |

Figure 2.8: “getAddress” function

* + wait(time): this wait function will increase the CurTime variable the setting wait time.
    - time: number defined the wait time. When it is ‘0’, it will issue WAIT command to transactor to wait SC\_ZERO\_TIME.

|  |
| --- |
| def wait (time):  global CurTime  if (time < 0):  print ("(common.py) ERROR: Negative simulation time!")  sys.exit ()  elif (time == SC\_ZERO\_TIME):  ret\_str = str(CurTime) + " "+ resolution + " WAIT " + str(time) + "\n"  tm.write(ret\_str)  ptm.write(ret\_str)  CurTime = CurTime + time |

Figure 2.9: “wait” function

* + end\_tm(): this function will generate all output files (transaction\_log.txt, port\_acc.txt and chk\_msg.py) and display “End of test pattern” message.

|  |
| --- |
| def end\_tm ():  msg\_set ("End of test pattern")  tm.close()  ptm.close()  # Generate chk\_msg.py (message check)  try:  f\_chk\_msg\_skl = OpenFile("chk\_msg\_py.skl", "r")  skl\_lines = f\_chk\_msg\_skl.readlines()  f\_chk\_msg\_skl.close()  f\_chk\_msg\_py = OpenFile("chk\_msg.py", "w")  for line in skl\_lines:  if (re.match("%%MESSAGE\_INFO%%", line) != None):  f\_chk\_msg\_py.write(chk\_msg\_info)  else:  f\_chk\_msg\_py.writelines(line)  f\_chk\_msg\_py.close()  except:  traceback.print\_exc()  sys.exit () |

Figure 2.10: “end\_tm” function

* + pass\_bp(): this function will display “TM is PASS” message and end of test pattern.

|  |
| --- |
| def pass\_bp ():  msg\_set (“TM is PASS”)  end\_tm() |

Figure 2.11: “pass\_bp” function

* + fail\_bp(): this function will display “TM is FAIL” message and end of test pattern.

|  |
| --- |
| def fail\_bp ():  msg\_set ("TM is FAIL")  end\_tm() |

Figure 2.12: “fail\_bp” function

* + msg\_set(msg): this function is used to display the specified message at specified simulation time.
    - msg: specified message to display during simulation

|  |
| --- |
| def msg\_set ( msg):  ret\_str = str(CurTime) + " "+ resolution + " MSG \"" + msg + "\"\n"  tm.write(ret\_str)  return ret\_str |

Figure 2.13: “msg\_set” function

* + msg\_chk(mark\_label, msg\_content, sim\_time\_period, count, accuracy = False): this function is used to generate message check information (%%MESSAGE\_INFO%% field in chk\_msg\_py.skl file, refer to section 2.3.3) for chk\_msg.py script.
    - mark\_label: identification marked point. Identification marked point can be a regular expression.
    - msg\_content: the expected message content. Message content can contains regular expression.
    - sim\_time\_period: the timing period whenever the message occurs from the start point. If it is “0”, check message at the time of start point. If it is “-1”, check message in the rest of simulation log file.
    - count: number of times that expected message is dumped. To check no message, ‘0’ is set for this argument.
    - accuracy: It is optional and valid when timing from start point is message ID or a setting time number. If accuracy is “True”, check message at the same time of message ID or at the setting time. If accuracy is “False”, check message between the time of start point and the time of message ID/setting time. Default value is “True”.

|  |
| --- |
| def msg\_chk (mark\_label, msg\_content, sim\_time\_period, count, accuracy = False):  global chk\_msg\_info  chk\_msg\_info += " MessageItem(\"" + mark\_label + "\", \"" + msg\_content + "\", " + str(sim\_time\_period) + ", " + str(count) + ", " + str(accuracy) + "),\n" |

Figure 2.14: “msg\_chk” function

* + reg\_acc(acc\_type, reg, size, val=0, low\_bit\_id = 0, max\_bit\_id = 31): this function will issue register R/W transaction request into transaction\_log.txt file and increase the CurTime (current simulation time) to 2\*clock\_period. The reason of increasing the current simulation time to 2\*clock\_period is to avoid the issue caused by unexpected simulation time of register R/W transaction in AT mode (non – blocking TLM, the next transaction can be invoked while the current transaction is not finished yet).
    - acc\_type: access type, support following types:
      * “RD”: normal read with checking expected value
      * “RO” normal read return value without checking expected value.
      * “WR”: normal write
      * “DR”: debug read
      * “DW”: debug write
    - reg: register name includes 2 fields: one is model indicator, one is register indicator. The separator is “.” character. For example:
      * DUT.REG\_A: DUT is model name which is defined in top.sci.skl. This function will decode address of register based on model\_base\_addr or model\_base\_reg variables. Refer to section 2.3.3 for the detail.
      * In case that register is defined with factor index, the corresponding factor can be declared as array accessing: DUT.REG\_A[1].
    - size: access size. It must be integer number. Eg: 1, 2, 4.
    - val: written value if it is write transaction, expected value if it is read transaction. The simulation will be stopped abnormally if return read value is different from expected value.
    - low\_bit\_id, max\_bit\_id: integer number used to specify the masked bit index in 32-bit value. The ‘val’ argument is corresponding to this setting. By the default, whole 32-bit value is calculated. This specification is new and is not implemented yet.

|  |
| --- |
| def reg\_acc (acc\_type, reg, size, val=0):  addr = 0  if isinstance(reg,int):  addr = reg  elif isinstance(reg, long):  addr = reg  else:  addr = getAddress(reg)  global CurTime  global resolution  ret\_str = (str(CurTime) + " "+ resolution + " " + acc\_type + " " + re.sub(r'L$', '', str(hex(addr))) + " " + str(size) + " " +re.sub(r'L$', '', str(hex(val))) + "\n")  CurTime += 2\*clock\_period  tm.write (ret\_str)  return ret\_str |

Figure 2.15: “reg\_acc” function

* + port\_acc (acc\_type, port, val = 0): this function will issue register R/W transaction request into port\_acc.txt file:
    - acc\_type: support following types:
      * “RD”: read port with checking expected value
      * “WR”: write port
    - port: port name which is declared in model info file. Port array is also supported. Port name is the hierarchy name
    - val: written port value if direction is “read”, expected port value if direction is “write”. The simulation will be stopped abnormally if return read port value is different from expected value.

|  |
| --- |
| def port\_acc (acc\_type, port, val = 0):  global CurTime  global resolution  if (acc\_type == "RD" or acc\_type == "WR"):  ret\_str = (str(CurTime) + " "+ resolution + " " + acc\_type + " " + port + " " + re.sub(r'L$', '', str(hex(val))) + "\n")  ptm.write(ret\_str)  else:  print ("(common.py) ERROR: Invalid access port type! Support only RD/WR")  sys.exit()  return "" |

Figure 2.16: “port\_acc” function

* User defined content:
  + User defined content is marked as “%%PY\_IMPORT%%” in common.py file. It is used to import specified Python script for register addresses and drivers as following example:

|  |
| --- |
| from iodefine\_control\_port import \*  from iodefine\_adc import \*  from iodefine\_sarad import \*    from control\_port\_drv import \*  from adc\_drv import \* |

Figure 2.17: Example of user defined content in common.py script

### Register address definition

Register address definition is defined in “iodefine\_<model>.py” file. It declares model\_base\_addr and model\_base\_reg variables:

* Current implementation:
  + Model base address is defined by model\_base\_addr variable (dictionary which key is string defined model name, and value is hex number defined model base address) and model offset value. If there are many model instances in test bench, model offset will be used to access model instance base address as an array. Following is an example:

|  |
| --- |
| global ADC\_OFFSET  ADC\_OFFSET = 0x1000  model\_base\_addr['ADC'] = 0xFFF91000  model\_base\_reg ['ADC'] = {  'VCR' : [ 32 , 0x000 , 0x4 ],  'PWDVCR' : [ 32 , 0x180 , 0 ],  …  }  //Address of ADC[0].VCR[1] is 0xFFF91000 + 0x000 + 1 \* 0x4 ~ 0xFFF91004  //Address of ADC[1].PWDVCR is 0xFFF91000 + 0x1000 + 0x180 ~ 0xFFF92180 |

Figure 2.18: Example of user defined content in Python iodefine model file (current)

* + Register base address is defined by model\_base\_reg variable. It is a dictionary which key is string defined register name, value is an array 3 elements:
    - Register access size (bits)
    - Register base address (hex number)
    - Register factor step (bytes)
* Enhanced implementation:
  + Model base address is defined by model\_base\_addr variable (dictionary which key is string defined model name, and value is an array with 2 elements:
    - hex number defined model base address, and
    - model offset value.
  + Register base address is defined by model\_base\_reg variable. It is a dictionary which key is string defined model name, value is a dictionary which key is register name, value is a dictionary which name is bit name. The first element is whole register information, next element is bit information. Data member of CRegInfo class is as following:
    - Register/Bit name: string indicates register name or bit name.
    - Register address offset: hex value indicates register address offset.
    - Register access size: number (bytes) (default is 4)
    - Register/Bit min bit index: number from 0 to 31 (default is 0)
    - Register/Bit max bit index: number from 0 to 31 (default is 31)
    - Register factor step: number (bytes): default is 0 (optional)
  + The following is an example.

|  |
| --- |
| class CBitInfo:  def \_\_init\_\_(self, name, min\_id = 0, max\_id = 31):  self.mName = name  self.mMinID = min\_id  self.mMaxID = max\_id  class CRegInfo:  def \_\_init\_\_(self, name, offset, acc\_size = 4, bit\_info = {}, step = 0):  self.mName = name  self.mOffset = offset  self.mAccSize = acc\_size  self.mBitInfo = bit\_info  self.mStep = step    model\_base\_addr['ADC'] = [0xFFF91000, 0x1000]  model\_base\_reg ['ADC'] = {  'VCR' : CRegInfo(‘VCR’, 0x000, 4, {  ‘WTTS’: CBitInfo (‘WTTS’, 24,27),  ‘GCTRL’: CBitInfo (‘GCTRL’, 0, 5),  } , 0x4 ),  'PWDVCR' : CRegInfo(‘PWDVCR', 0x180, 4),  …  }  //Address of ADC[0].PWDVCR[1] is 0xFFF91000 + 0x180 + 1 \* 0x4 ~ 0xFFF91004  //Address of ADC[1].PWDVCR[0] is 0xFFF91000 + 0x1000 + 0x180 ~ 0xFFF92180 |

Figure 2.19: Example of user defined content in in Python iodefine model file (enhancement)

### Message check specification

* The message check script (chk\_msg.py) is generated from the skeleton file (chk\_msg\_py.skl) which is stored on “sim/pat/0\_common\_files” folder.
* Main function in this script is: chk\_msg (input\_file). This function will parse each line of log file and check each message checking rule if it is same as the expectation and return the result as number. Message checking rules are created based on test patterns and they are stored in “MSG” variable.
* The meaning of return value is as following:
  + -1: No message checking rules (test pattern does not check message)
  + 0: All message checking rules are checked and they are all PASS
  + 1: All message checking rules are checked and one of them is FAIL
  + 2: Script error related to IO files (log file, result file)
  + 3: Error caused by marked message (cannot find marked message, cannot get simulation time from marked message)
  + 4: Error caused by checked message (cannot get simulation time from checked message)
* “MSG” variable is an array of MessageItem class’s instances. These instances are generated by “msg\_chk” common function (in common.py script)

|  |
| --- |
| MSG = [  %%MESSAGE\_INFO%%  ]  chk\_msg\_py.skl |

Figure 2.20: MSG variable in chk\_msg\_py.skl file

* + The following is an example of %%MESSAGE\_INFO%%

|  |
| --- |
| MSG = [  MessageItem("=== MSG ID 00008 Invalid check", "Invalid access address", 20, 1, False),  MessageItem("1234", "m1", 100, 1, True),  MessageItem("ABC", "DEF", 0, 0, True),  MessageItem("XY.\*Z", "INT assert", -1, 4, True),  chk\_msg.py  ] |

Figure 2.21: MSG variable in chk\_msg.py file

***Explanation***:

* + No.1: check if message “Invalid access address” is dumped **1** time **from** the simulation time that message “=== MSG ID 00008 Invalid check” is detected **to** 20 time unit.
  + No.2: check if message “m1” is dumped 1 time only **at** 100 time unit after the simulation time that “1234” is detected.
  + No.3: check if message “DEF” is **NOT** dumped at the simulation time that “1234” is detected.
  + No.4: check if message “INT assert” is dumped 4 times **from** the simulation that message “XY,\*Z” is detected **until the message “End of test pattern”**.
* MessageItem class has following data members:
  + mark\_label: identification marked point (marked message). Identification marked point can be a regular expression. (Note for users: This marked message must be unique to assure the accuracy of checking message).
  + msg\_content: the expected message content (checked message). Message content can contains regular expression.
  + msg\_dump\_cnt\_exp: expected number of checked messages which is dumped in expected time.
  + msg\_dump\_time\_exp: the timing period whenever the message occurs from the start point. If it is “0”, check message at the time of start point. If it is “-1”, check message in the rest of simulation log file.
  + accurate\_dump\_time: accuracy of expected message’s timing check. If it is “True”, check message at msg\_dump\_time\_exp setting time. If accuracy is “False”, check message from the time of mark\_label to msg\_dump\_time\_exp setting time. Default value is “False”.
  + msg\_check\_enabled: Boolean flag to enable checking message. Its initial value is “False”.
  + msg\_dump\_cnt: number of checked message at/during checking time/period. Its initial value is 0
  + msg\_dump\_starttime: simulation time of the marked message (integer type). Its initial value is 0.
  + msg\_pass: message result (string type). Its initial value is “PASS”.
* The message report file is as following:

|  |
| --- |
| NO. MARK LABEL MESSAGE COUNT COUNT\_EXP RESULT  1 '=== MSG ID 00008 Invalid check' 'Invalid access address' 1 1 PASS  2 '1234 ' 'm1 ' 1 1 PASS  msg\_chk.rpt |

Figure 2.22: Example of message check report

## Model implementation (“tb” folder)

### Transactor model

* Transactor is enhanced the following points:
  + Use TLM Bus width 32 bit.
  + Fix issues that wait\_time is negative number (simulation time of the next transaction is smaller than current transaction).
  + Current supported actions are: “RD”, “WR”, “DR”, “DW”, new “WAIT” and “MSG” action are added. “WAIT 0” means “wait(SC\_ZERO\_TIME).
  + Add 2 new arguments for “RD”, “WR”, “DR”, “DW” actions to declare the bit accessing. If lower masked bit is different from 0 or higher masked bit is different from 31, a read transaction will be called to get value to mask it before writing.

### Control port model

* Control port model is enhanced the following points to read/write port:
  + Add new data members as a sorted associative container of port instances (key is port name string):

std::map<std::string, sc\_in <port\_type>\* > m<Direction>Ports<Portttype>.

* + - port\_type: C++/SystemC data type (bool, unsigned int, sc\_uint<>, …)
    - Portttype is port\_type without space and special characters, the first character is upper case. Eg: “Bool”, “UnsignedInt”, “Uint64”, “Uint2”, …
    - Direction is “Input” or “Output”
    - Example for constructor:

|  |
| --- |
| mOutputPortsBool["preset\_n"] = &preset\_n;  mOutputPortsBool["RESETAD\_N"] = &RESETAD\_N;  mOutputPortsUint64["pclk"] = &pclk;  mOutputPortsUint64["ADCLK"] = &ADCLK;  mInputPortsBool["INT\_AD00"] = &INT\_AD00; |

Figure 2.23: Example of constructor of control port model for table of ports

* + Add “IssueTransFromFile” thread to parse “port\_acc.txt” file to read/write port. It is similar to transactor model. Supported actions are: RD, WR and WAIT.

|  |
| --- |
| std::string cur\_line;  std::string sim\_time;  std::string resolution;  std::string action;  std::string port\_name;  while (1) {  while(getline(mInputStream, cur\_line)){  std::istringstream sin(cur\_line);  sin >> sim\_time;  sin >> resolution;  sin >> action;  double sim\_time\_double = GetWaitTime(sim\_time, resolution, false);  if (action == "WAIT") {  sin >> sim\_time;  GetWaitTime(sim\_time, resolution, true);  } else {  unsigned int tempdata ;  sin >> port\_name;  sin >> std::hex >> tempdata;    bool find\_success = false;  if (mInputPortsBool.find(port\_name) != mInputPortsBool.end()) {  if (action == "RD") {  unsigned int rd\_val;  rd\_val = mInputPortsBool.find(port\_name) -> second->read();  if (rd\_val != tempdata) {  printf("Port (%s), read value (0x%x), expected value (0x%x). Port read FAIL. \n", …);  }  } else {  printf ("%s [%20s] Cannot write to input port %s\n", this->name(), …);  }  find\_success = true;  } else if … // other port type check  }  if (!find\_success) {  printf("error [%s] (%s) Cannot find port name '%s'.\n", …);  }  }  }  return;  } |

Figure 2.24: IssueTransFromFile thread of control port model

### ShPythonAPI class

#### Overview

This class is Python interpreter as C++ class model. It is used to:

* + Register model Python I/F.
  + Read and execute Python test pattern.

#### Header file description:

The following figure describes Python I/F header source code.

This preprocessor condition is implemented to fix linking issue on Windows Debug mode.

#ifndef SH\_PYTHON\_API\_H\_DEF

#define SH\_PYTHON\_API\_H\_DEF

#ifdef \_DEBUG

#undef \_DEBUG

#include <Python.h>

#define \_DEBUG

#else

#include <Python.h>

#endif

class ShPythonAPI {

public:

static void StartPy (int argc, char\* argv[]);

static void DestructorPy (void);

static void LoadShCommandDefPy ();

};

#endif //#ifndef SH\_PYTHON\_API\_H\_DEF

Figure .: Python I/F model header file

#### Implementation file description:

The following figure describes Python I/F implementation source code.

The following figure describes Python I/F implementation source code.

SCHEAP: Python init module name.

It is declared in generated PY\_<model>.h file.

This name is generated as “SCHEAP” by handle Command I/F generator. Should we change it?

#include "ShPythonAPI.h"

#define PYTHON\_HOME "UT\_PYTHONHOME"

%%INCLUDE\_MODULE\_HEADERS

extern char \*glb\_python\_scr\_file;

void ShPythonAPI::StartPy (int argc, char\* argv[])

{

// Show Python home

char \*py\_home = getenv(PYTHON\_HOME);

if( py\_home ) {

Py\_SetPythonHome( py\_home );

}

Py\_Initialize ();

LoadShCommandDefPy ();

char command\_string[1024];

sprintf ( command\_string, "execfile(\"%s\")", glb\_python\_scr\_file);

PyRun\_SimpleString (command\_string);

Py\_Finalize ();

}

void ShPythonAPI::DestructorPy (void) {}

static PyObject\* sc\_startPy(PyObject\* self, PyObject\* args)

{

unsigned int cycle\_i;

PyArg\_ParseTuple (args, "i", &cycle\_i);

sc\_start();

return Py\_BuildValue ("");

}

// python extended command definition

static PyMethodDef m\_sh\_api\_def[] = {

{"sc\_start", sc\_startPy, METH\_VARARGS, "sc\_start"},

{NULL, NULL, 0, NULL}

};

/// <summary>load python command</summary>

void ShPythonAPI::LoadShCommandDefPy ()

{

PyObject \*tmp = Py\_InitModule("SCHEAP", m\_sh\_api\_def);

%%COMMAND\_HANDLER

static const char\* CODES\_Template = "import SCHEAP\n";

char CODES[16384];

sprintf(CODES, CODES\_Template);

PyRun\_SimpleString(CODES);

}

Figure .: Python I/F model implementation file

***Explanation***:

Red description will be replaced by corresponding code as following:

* + %%INCLUDE\_MODULE\_HEADERS: include model Python I/F header files.  
    Eg:

#include "PY\_SCDS.h"

* + %%COMMAND\_HANDLER: invokes SetPyExtCmd APIs of model Python I/F.  
    Eg:

PY\_SCDS::SetPyExtCmd();

Python I/F model has 3 functions:

* + StartPy function: this is main function. It will initializes, load, and process Python test pattern. It is invoked in transactor model.
  + DestructorPy function: this is destructor function. It is implemented as empty function as default. It is invoked in transactor model after parsing Python test pattern.
  + LoadShCommandDefPy function: this function load Python I/F commands of all Python I/F models in unit environment. It is invoked in StartPy function.

## Environment configuration setting (“scripts” folder)

### Linux environment configuration setting

* For REL: env\_rel\_set.csh <library> [32/64bit mode]
  + Library is “osci” or “astc”
  + 32/64 bit mode: “m32” or “m64”. Default is 64 bit mode (m64)
  + Environment setting is as following table:

Table 2.1: REL environment setting

|  |  |  |  |
| --- | --- | --- | --- |
| Environment variable/Tools | | | Setting |
| ld\_lib | | 32bit: | /common/appl/Renesas/SystemC/compiler/gcc4.1.2-64/lib |
| 64bit: | /common/appl/Renesas/SystemC/compiler/gcc4.9.3/lib64 |
| OSCI | SC\_TYPE | 32bit: | SystemC-2.3.0 |
| 64bit: | SystemC-2.3.1 |
| SYSTEMC | | /common/appl/Renesas/SystemC/${SC\_TYPE} |
| COMMON\_MODEL\_DIR | |  |
|  | 32bit: | /eda04/SystemDesign/vpp\_work/${VP\_NAME}/all\_library |
| 64bit: | /eda04/SystemDesign/vpp\_work/${VP\_NAME}/all\_library\_m64 |
| TLM | 32bit: | /common/appl/Renesas/SystemC/TLM\_in\_Sysc-2.3.0 |
| 64bit: | ${SYSTEMC} |
| ASTC | OSCAR\_HOME | 32bit: | /eda04/SystemDesign2/appl/ASTC/VLAB/vlab-2.1.9 |
| 64bit: | /eda04/SystemDesign2/appl/ASTC/VLAB/vlab-2.3.6 |
| RLM\_LICENSE | | 5053@mlsl110.eda.renesas.com |
| SYSTEMC | | ${OSCAR\_HOME}/include |
| COMMON\_MODEL\_DIR | |  |
|  | 32bit: | /eda04/SystemDesign/vpp\_work/${VP\_NAME}/all\_library\_astc |
| 64bit: | /eda04/SystemDesign/vpp\_work/${VP\_NAME}/all\_library\_astc\_m64 |
| ld\_lib | | ${OSCAR\_HOME}/lib:${ld\_lib} |
| TLM | | ${SYSTEMC} |
| PATH | | 32bit: | "/common/appl/Renesas/SystemC/free\_tool/Python-3.1.2rc1/bin/:/common/appl/Renesas/SystemC/compiler/gcc4.1.2-64/bin:/common/appl/Renesas/SystemC/free\_tool/bin:$PATH" |
| 64bit: | "/common/appl/Renesas/SystemC/free\_tool/Python-3.1.2rc1/bin/:/common/appl/Renesas/SystemC/compiler/gcc4.9.3/bin:/common/appl/Renesas/SystemC/free\_tool/bin:$PATH" |
| gcc\_ver | | | `gcc -dumpversion | sed '1,$s/\.//g'` |
| SYSTEMC\_DEBUG | | | -gcc${gcc\_ver}${buildmode} |
| BS | | | "bs -M 1000 -os RHEL6" |
| PYTHON\_HOME | | | /common/appl/python/python-2.7.3-64bit |
| PYTHON\_LIB | | | /common/appl/Renesas/SystemC/free\_tool/Python-3.1.2rc1/lib |
| PYCHECKER | | | /common/appl/Renesas/SystemC/free\_tool/pychecker-0.8.18/pychecker/ |
| PYTHON3 | | | /common/appl/Renesas/SystemC/free\_tool/Python-3.1.2rc1/bin/python3.1 |
| PYTHON3\_LIB | | | /common/appl/Renesas/SystemC/free\_tool/Python-3.1.2rc1/lib/python3.1 |
| TC\_INSTALL\_DIR | | | /common/appl/Renesas/shc/SHCV90200 |
| CC | | | g++ |
| LD\_LIBRARY\_PATH | | | {ld\_lib}:${LD\_LIBRARY\_PATH} |
| SEARCH\_DIR | | | ${COMMON\_MODEL\_DIR} |
| INCDIR | | | -I${COMMON\_MODEL\_DIR} |
| 1Team | | | /common/appl/dotfiles/1TeamSystem.CSHRC\_1.16.7 |
| SH compiler | | | /common/appl/Renesas/shc/dotfiles/shc.CSHRC\_9.02.00 |

* For RVC: env\_rvc\_set.csh <library> [32/64bit mode]
  + Library is “osci” or “astc”
  + 32/64 bit mode: “m32” or “m64”. Default is 64 bit mode (m64)
  + Environment setting is as following table:

Table 2.2: RVC environment setting

|  |  |  |  |
| --- | --- | --- | --- |
| Environment variable/Tools | | | Setting |
| ld\_lib | | 32bit: | /common/appl/gcc-4.1.2/lib |
| 64bit: | /shsv/sld/Common/05\_Tools/04\_Compilers/gcc-4.9.3-64bit/lib64 |
| OSCI | SC\_TYPE | 32bit: | systemc-2.3.0\_gcc412\_tlm-2.3.0 |
| 64bit: | systemc-2.3.1a\_gcc493\_64bit |
| SYSTEMC | | /shsv/sld/Common/04\_Lib/01\_SystemC/${SC\_TYPE} |
| COMMON\_MODEL\_DIR | |  |
|  | 32bit: | /shsv/sld/Common/04\_Lib/99\_latest\_common\_src/${VP\_NAME}/all\_library\_tmp/OSCI |
| 64bit: | /shsv/sld/Common/04\_Lib/99\_latest\_common\_src/${VP\_NAME}/all\_library\_64bit/OSCI-m64 |
| ASTC | OSCAR\_HOME | 32bit: | /common/appl/ASTC/Vlab\_2.1.9 |
| 64bit: | /common/appl/ASTC/Vlab\_2.3.6-64bit |
| RLM\_LICENSE | | 5053@licedu3 |
| SYSTEMC | | ${OSCAR\_HOME}/include |
| COMMON\_MODEL\_DIR | |  |
|  | 32bit: | /shsv/sld/Common/04\_Lib/99\_latest\_common\_src/${VP\_NAME}/ all\_library\_ tmp/ASTC\_VLAB\_2\_1\_9 |
| 64bit: | /shsv/sld/Common/04\_Lib/99\_latest\_common\_src/${VP\_NAME}/all\_library\_64bit/ASTC\_VLAB\_2\_3\_6-m64 |
| ld\_lib | | ${OSCAR\_HOME}/lib:${ld\_lib} |
| PATH | | 32bit: | "/common/appl/python/python-3.1.2-64bit/bin/":"/common/appl/gcc-4.1.2/bin":"$PATH" |
| 64bit: | "/common/appl/python/python-3.1.2-64bit/bin/":"/shsv/sld/Common/05\_Tools/04\_Compilers/gcc-4.9.3-64bit/bin":"$PATH" |
| gcc\_ver | | | `gcc -dumpversion | sed '1,$s/\.//g'` |
| SYSTEMC\_DEBUG | | | -gcc${gcc\_ver}${buildmode} |
| BS | | | "" |
| PYTHON\_HOME | | | /common/appl/python/python-2.7.3-64bit |
| PYTHON\_LIB | | | /common/appl/Renesas/SystemC/free\_tool/Python-3.1.2rc1/lib |
| PYCHECKER | | | /common/appl/Renesas/SystemC/free\_tool/pychecker-0.8.18/pychecker/ |
| PYTHON3 | | | /common/appl/python/python-3.1.2-64bit/bin/python3.1 |
| PYTHON3\_LIB | | | /common/appl/python/python-3.1.2-64bit/lib/python3.1 |
| TC\_INSTALL\_DIR | | | /common/appl/Renesas/shc/SHCV90200 |
| CC | | | g++ |
| LD\_LIBRARY\_PATH | | | {ld\_lib}:${LD\_LIBRARY\_PATH} |
| SEARCH\_DIR | | | ${COMMON\_MODEL\_DIR} |
| INCDIR | | | -I${COMMON\_MODEL\_DIR} |
| TLM | | | ${SYSTEMC} |
| 1Team | | | /common/appl/Env/SpyGlass/1TeamSystem-1.16.7 |
| SH compiler | | | ${TC\_INSTALL\_DIR}/shc.CSHRC\_9.02.00 |

### Windows environment configuration setting

* Setting file: setup\_<library>.bat
  + Library is “osci” or “astc”
  + Environment setting is as following table:

Table 2.3: Environment setting

|  |  |  |  |
| --- | --- | --- | --- |
| Environment variable/Tools | | | Setting |
| call | | 32bit: | C:\Program Files (x86)\Microsoft Visual Studio 10.0\Common7\Tools\vsvars32.bat |
| 64bit: | C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\x86\_amd64\vcvarsx86\_amd64.bat |
| MODE | | | “Release” or “Debug” or “Release\_MT” or “Debug\_MT”.  In ASTC mode, only “Release” mode is supported. |
| OSCI | SYSC\_VER | 32bit: | systemc-2.3.0 |
| 64bit: | systemc-2.3.1a\_64bit |
| VS\_VER | 32bit: | msvc100 |
| 64bit: | msvc100\_m64 |
| ROOT\_PATH | | C:\Users\%USERNAME%\Projects\Lib |
| SYSTEMC\_PATH | | %ROOT\_PATH%\%SYSC\_VER%\src |
| SYSTEMC\_LIB | 32bit | %ROOT\_PATH%\%SYSC\_VER%\%VS\_VER%\SystemC\%MODE% |
| 64bit | %ROOT\_PATH%\%SYSC\_VER%\%VS\_VER%\SystemC\x64\%MODE% |
| ASTC | COMMON\_PATH | 32bit: | Z:\Common\04\_Lib\99\_latest\_common\_src\M40PF\_all\_library |
| 64bit: | Z:\Common\04\_Lib\99\_latest\_common\_src\M40PF\_all\_library\_64bit\OSCI-m64 |
| OSCAR\_PATH | 32bit: | C:\Program Files (x86)\VLAB Works\VLAB 2.1.2 (vc100) |
| 64bit: | C:\Program Files\VLAB Works\VLAB 2.3.6 (vc140) |
| RLM\_LICENSE | | Same as Linux environment setting |
| SYSTEMC\_PATH | | %OSCAR\_PATH%\include |
| SYSTEMC\_LIB | | %OSCAR\_PATH%\lib |
| PATH | | %OSCAR\_PATH%\lib;%PATH% |
| LIB | | | %SYSTEMC\_LIB%;%COMMON\_PATH%\%VS\_VER%\%MODE%;%LIB% |
| PYTHON\_HOME | | | C:\Python273\_64bit |

## Top module generation (“gen\_env” folder)

Top module generation is updated from Git tag “v2017\_05\_08” to install Python I/F to process Python test pattern. It is updated for two following points:

- Top module (test bench) and sc\_main skeleton files: update to add Python test pattern filename as global variable and remove. Update points are described in section 2.6.1

- Python I/F skeleton file: this file is added to generate Python I/F class. Its specification is described in section 2.4.3

- Top module generator (top\_gen.py/top\_gen\_class.py): this script is updated to generated ShPythonAPI.h & test.vcxproj

### Top module and skeleton files

#### Include files

The Python I/F header file is included in top\_main.cpp.skl:

#include "ShPythonAPI.h"

#### Variables

Table 2.4: Table of updated variables

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Name** | **Type** | **Description** | **File** | **Note** |
| 1 | PAT | macro | Test pattern path | top.h.skl | Remove |
| 2 | m\_patdir | std::string | Test pattern directory | top.h.skl  top\_main.cpp.skl | Remove |
| 3 | glb\_python\_tm | char \* | Python test pattern name | top\_main.cpp.skl | Add |
| 4 | argv | char \*\* | Argument list of main program | top\_main.cpp.skl | Add |
| 5 | mShPythonAPI | ShPythonAPI | Python I/F declaration | top\_main.cpp.skl | Add |

#### Functions

Table 2.5: Table of updated functions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Name** | **Description** | **File** | **Note** |
| 1 | constructor  (Ctest) | Remove 2nd argument (patdir) | top.h.skl  top\_main.cpp.skl | Remove |
| 2 | Remove code to allocate memory for simple memory model (because this model is not needed | top.h.skl | Remove |
| 3 | sc\_main | Remove -patdir option | top\_main.cpp.skl | Remove |
| 4 | Add -tm option | Add |
| 5 | Invoke mShPythonAPI.StartPy function | Add |

### VC++ project file generation

Top module generation is enhancement to generate test.vcxproj file for include files and implementation files:

|  |
| --- |
| <ItemGroup>  <ClCompile Include="top\_main.cpp" />  <ClCompile Include="ShPythonAPI.cpp" />  **%%IMPLEMENTATION\_FILES\_WINDOWS**  </ItemGroup>  <ItemGroup>  <ClInclude Include="top.h" />  <ClInclude Include="ShPythonAPI.h" />  **%%INCLUDE\_MODULE\_HEADER\_WINDOWS**  </ItemGroup> |

Figure 2.27: test.vcxproj skeleton file

* + %%IMPLEMENTATION\_FILES\_WINDOWS
    - Implementation files will be generated to compile as following:

<ClCompile Include="sarad.cpp" />

<ClCompile Include="adc.cpp" />

* + %%INCLUDE\_MODULE\_HEADER\_WINDOWS
    - Header files will be generated to include as following:

<ClInclude Include="sarad.h" />

<ClInclude Include="adc.h" />

Beside that, “MS\_NO\_COREDLL” preprocessor is also added to fix linking issue on Windows Debug mode.

## Simulation structure preparation (“gen\_sim” folder)

* This folder contains 3 scripts to prepare for the simulation:
  + gen\_sim.pl: Perl script to generate simulation structure and linked files to prepare for the simulation. It is reused from RESLX environment. It is enhanced to make link for create\_ssc.pl script
  + create\_batch.pl: Perl script is used to generate run\_exec.csh file, which is the main script to run simulation. It is reused from RESLX environment. It is enhanced to generate command in run\_exec.csh file.
  + New script “create\_ssc.pl” is implemented to generate .ssc file for model setting by parameters and handle Command for each test pattern. It will be called when simulation by running run\_exec.csh. This script has 1 argument is the output ssc file name.
* All Perl scripts are implemented to run on both Linux and Windows.

## Simulation result report (“check\_result” folder)

* Check result script is put at “scripts/check\_result”. Check result script in RESLX environment is “check\_result.pl”. It is written by Perl language. For new unit test environment, new Python script is implemented (check\_result.py) to check the simulation result.
* This script has 2 parts:
  + Mandatory check on log file. It will report as compile error if :
    - There is “TM is FAIL \(lack of tm\)” message in log file, or
    - There is no “End of test pattern” message in log file.

It will report Fail result if:

* + - There is no “TM is PASS” message in log file, or
    - There is “FAIL” message at the end of any log line, or
    - There is “Segmentation fault” message in log file, or
    - There is “fatal\(TLM\)” in log file.
  + Message check: this script will call chk\_msg function in “chk\_msg.py” script to check message. It will report the result as return value of this function. Please refer to section 2.3.3 for the detail.
* This script is implemented to use for both Linux and Windows

## Report generation (“gen\_report” folder)

* This folder is reused from RESLX environment to generate report from results stored in “results” folder. Scripts are enhanced to use for both Linux and Windows.

## Run all script (“run\_all” folder)

* This script is used to run all simulation automatically. It is put at “run\_all” folder. Main script is “run\_all\_execution.csh” for Linux and “run\_all\_execution.bat” for Windows

### Linux scripts

* run\_all\_execution.csh script: There are two arguments: <site> <os>
  + site is “rvc” or “rel”.
  + os is “Linux” or “Windows”.
    - If this argument is “Linux”, this script will run 4 modes: OSCI/ASTC lt/at modes (for line coverage), and OSCI/lt mode (for branch coverage). All simulation results are put in sim/All\_simulation\_results folder.
    - If this argument is “Windows”, this script will prepare “sim\_windows” script to run simulation on Windows in parallel with Linux simulation.
* gen\_env.csh: this script is created new by moving the code to generate test bench (top module) out of run\_all.csh script. It can be run independently to prepare Windows data.
* run\_all.csh: this script is reused from RESLX environment to run simulation for one specified mode. It is invoked by run\_all\_execution.csh script

### Windows scripts

* run\_all\_execution.bat script: This script calls run\_all\_<osci/astc>.bat scripts.
* run\_all\_osci.bat: this script is used to run simulation for Release Release\_MT Debug Debug\_MT mode for OSCI library.
* run\_all\_osci.bat: this script is used to run simulation for Release Release\_MT Debug Debug\_MT mode for ASTC library.
* compile.bat: this script is used to run compile VC++ project file in CUI mode.
* compile\_gui.bat: this script is used to run compile VC++ project solution file in GUI mode.
* timecmd.bat: this script is used to calculate CPU time of a command, or a script.

## UT environment generation script

* create\_unit\_env.csh script is used to generate UT environment for a specific model. It has 3 arguments:
  + input directory: the path to the input files (same directory structure as Unit environment skeleton). Input path can contains test patterns, model information file, test bench configuration, model source code, …
  + output directory: the path to generated UT environment for specific model.
  + model name: model name
* This script follows these steps:
  + Confirm if to back up environment if it is existed.
  + Copy UT environment skeleton to output directory.
  + Replace all model\_name and MODEL\_NAME keyword in all files
  + Copy input directory data to output directory.

# **Others**

## Data location

Git repository for regression test is changed from:

/shsv/sld/ipp/Repository/R-IP/Common/UnitEnv.git

to:

/shsv/sld/ipp/Repository/R-IP/Common/Regression64bitEnv.git

This new unit environment will be committed as new repository on GIT LAB server:

https://gitlab.rvc.renesas.com/FED2/ipd2\_sld\_17008\_unit\_env.git

## Notes

To run Perl script on Windows, please install Strawberry Perl (recommended version is from 5.22.1.3-64bit)

As Figure 1.1, there are some steps can be considered to implement to reduce the model development time.

A. Code preparation: DUT model I/F should be considered to generate in future also. Sockets, ports, registers, variables, initialization, port tracing (for waveform check) are generated automatically to avoid some mistakes in initialization and to reduce workload of designer. There is port generation script are created before for similar purpose. It can be enhanced for this target.

C, D, E also should be considered to develop.

By the way, when the unit environment skeleton is updated, all models should update latest environment, too. This point is not considered in this specification yet.

**sRevision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Modified points** | **Agreement by customer** | **Checker** | **Author** |
| 1.0 | - Created new | - | Duc Duong  05/08/2018 | Yen Nguyen  05/04/2018 |
| 1.1 | Update to use test pattern as Python script and use Python I/F  - Figure 1.2: Change block diagram  - Figure 2.4: Change test pattern format to Python script.  - Chapter 3: Add new for repository info and notes. |  | Duc Duong  05/22//2018 | Yen Nguyen  05/22/2018 |
| 1.2 | Update to fix REL comments (#36973)  - Figure 1.2: Unify the words in the figures, the explanations, chapter names and so on  - Add explanation for .cmd  - Section 3.1: revise to use new git. |  | Duc Duong  05/28/2018 | Yen Nguyen  05/28/2018 |
| 1.3 | - Add Figure 1.1 : whole view of model development  Figure 1.2: Change “.msg” to “chk\_msg.py”, change check\_results.pl to check\_results.py, add common.py, <model>\_drv.py  - Table 1.1: Add red description for pattern, port and message check.  - Add section 2.4.2 and 2.4.3 for control port model and ShPythonAPI class  - Add 2.6.2 for VC++ project file generation  - Add 2.7and 2.9 for gen\_sim and gen\_report updating.  - Update 2.8 and 2.10 for check\_result and run\_all scripts.  - Section 3.1: Update path of GIT LAB data and note for Strawberry Perl on Windows.  - Table 2.1, Table 2.2 and Table 2.3: Change VLAB version from 2.3.6-rc1 to 2.3.6 |  | Duc Duong  11/20/2018 | Yen Nguyen  11/20/2018 |