|  |
| --- |
| **Verification Specification** |

|  |
| --- |
| **Development of PSIS011 model**  **for E2x-FCC2**  (v1.2) |

|  |
| --- |
| **Summary:** |
| This document describes the verification methodology and verification procedure used to verify PSIS011 model. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference Manuals** | | | | |
| **No.** | **Title name** | **Document number** | **Description** | **Path** |
| 1 | SC-HEAP\_E3 common requirement (v1.0) | - | The common requirement  (***File***: Common\_Requirement\_RVC.pdf) | **DMS:** Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/02\_MCS\_Project/From\_MCS |
| 2 | SC-HEAP E3 Platform functional specification | LLWEB-00009484\_E  MSS-SG-12-0061-02 \_E | The document describes about SC-HEAP E3 Platform functional  (***File***: SC-HEAP\_E3\_platform\_E\_t.pdf) |
| 3 | REQ-MCS-17013\_PSIS011 (\*) | - | Detail requirement of PSIS011 model  (***File***: REQ-MCS-17013\_PSIS011.xlsx) | **DMS:** Documents/1. General Documents/010\_ENG/050\_Software/5\_SW Design Qualification Management/MCU\_Modeling/REQ |
| 4 | VRF-MCS-17013-01\_PSIS011 (\*) | - | Checklist of PSIS011 model  (***File***: VRF-MCS-17013-01\_PSIS011.xls) | **DMS:** Documents/1. General Documents/010\_ENG/050\_Software/5\_SW Design Qualification Management/MCU\_Modeling/VRF |

***Note****: (\*) Refer to TRA-MCS-17013\_PSIS011 and DEV-MCS-17013\_PSIS011 for version number of REQ-MCS-17013\_PSIS011 and VRF-MCS-17013-01\_PSIS011*

Table of Contents

[1. Summary 6](#_Toc501442671)

[1.1. Introduction 6](#_Toc501442672)

[1.2. Block diagram of Unit Test environment 6](#_Toc501442673)

[1.3. Block diagram of SC-HEAP environment 8](#_Toc501442674)

[1.4. Dummy Master model specification 9](#_Toc501442675)

[1.5. Dummy Peripheral model specification 11](#_Toc501442676)

[2. Environment Structure 16](#_Toc501442677)

[2.1. How to verify on UT 17](#_Toc501442678)

[2.2. How to verify on SC-HEAP environment 17](#_Toc501442679)

[2.3. Verification environment on Linux 17](#_Toc501442680)

[2.4. Verification environment on Windows 19](#_Toc501442681)

[3. Verification conditions 20](#_Toc501442682)

[4. Verification requirements 20](#_Toc501442683)

**Index of Figures**

[Figure 1.1: Block diagram of Unit Test environment 6](#_Toc501442684)

[Figure 1.2: Block diagram of SC-HEAP verification environment 8](#_Toc501442685)

[Figure 1.3: Block diagram of Dummy Master model 9](#_Toc501442686)

[Figure 1.4: Operation flow of Dummy Master model 10](#_Toc501442687)

[Figure 1.5: Block diagram of Dummy Peripheral model 11](#_Toc501442688)

[Figure 1.6: Operation flow of the Dummy Peripheral about receiving input signals 14](#_Toc501442689)

[Figure 1.7: Operation flow of the Dummy Peripheral about issuing output signals 14](#_Toc501442690)

[Figure 2.1: Verification environment structure 16](#_Toc501442691)

[Figure 2.2: Flow chart of verification on Linux 18](#_Toc501442692)

[Figure 2.3: Flow chart of verification on Windows 19](#_Toc501442693)

**Index of Tables**

[Table 1.1: List of Dummy Master’s registers 9](#_Toc501442694)

[Table 1.2: List of Dummy Peripheral’s registers 12](#_Toc501442695)

[Table 2.1: Explanation of verification on Linux 18](#_Toc501442696)

[Table 2.2: Explanation of verification on Windows 19](#_Toc501442697)

[Table 3.1: Verification conditions 20](#_Toc501442698)

[Table 4.1: Verification requirement 20](#_Toc501442699)

# Summary

## Introduction

The purpose of this document is to describe a verification methodology and verification procedure used to verify PSIS011 model.

## Block diagram of Unit Test environment

In this project, the Unit Test environment (UT for short) is mainly employed to verify the PSIS011 model. All registers, operations, and Python IF features are verified in Unit Test. Figure 1.1 shows the block diagram of the Unit Test environment.

**Dummy Master**

is

**PSIS011**

**Dummy**

**Peripheral**

**UT Environment**

Python IF

n: from 0 to 7

m: from 1 to 7

**Legend:**

Ports connection

TLM socket

Verification models

Bus decoder

Design under test

psis\_mult\_clk

psis\_mult\_rst\_n

RX\_CONTROL

TX\_DATA

TX\_CONTROL

int\_psis\_chn

dma\_psis\_chn\_rx

dma\_psis\_chm\_tx

psis\_trig\_sync\_chn

psis\_clk\_timestamp\_a

**UART**

**Interrupt**

**DMA**

**Clock/reset**

**GTM IF**

psis\_clk\_timestamp\_b

psis\_clr\_timestamp\_a

8

8

7

8

psis\_clr\_timestamp\_b

psis\_stsp\_timestamp\_a

psis\_stsp\_timestamp\_b

psis\_clk

PCLK

PRESETn

psis\_rst\_n

RX\_DATA

ts

ts

**Dummy Is**

**(empty model)**

is

Figure 1.1: Block diagram of Unit Test environment

***Explanation***:

* Dummy Master model is used to issue transactions to PSIS011 model via one initiator socket (refer to chapter 1.4 in the detail). This model has one initiator socket “is”.
* Dummy Peripheral model is used to control the input signals to PSIS011 model. Besides, Dummy Peripheral receives and confirms the value of output signals issued from PSIS011 model (refer to chapter 1.5 in the detail).
* Both UT and SC-HEAP environments utilize the same Dummy Peripheral whose target socket is unused in UT (as UT mainly uses Python IF to control/confirm verification models’ ports – it doesn’t use target socket to access register). Therefore, Dummy Is which only owns a TLM initiator socket is employed to treat unused target socket of Dummy Peripheral.

## Block diagram of SC-HEAP environment

In this project, the SC-HEAP environment is employed to verify the PSIS011 model. Some registers, and basic operations are verified on SC-HEAP to make sure the PSIS011 model can operate normally on SC-HEAP environment. Figure 1.2 shows the block diagram of the SC-HEAP environment.

**Dummy Master**

**APB Bus**

**PSIS011**

**Dummy**

**Peripheral**

**SC-HEAP Environment**

Python IF

n: from 0 to 7

m: from 1 to 7

**Legend:**

Ports connection

TLM socket

Verification models

Bus decoder

Design under test

psis\_mult\_clk

psis\_mult\_rst\_n

RX\_CONTROL

TX\_DATA

TX\_CONTROL

int\_psis\_chn

dma\_psis\_chn\_rx

dma\_psis\_chm\_tx

psis\_trig\_sync\_chn

psis\_clk\_timestamp\_a

**UART**

**Interrupt**

**DMA**

**Clock/reset**

**GTM IF**

psis\_clk\_timestamp\_b

psis\_clr\_timestamp\_a

8

8

7

8

psis\_clr\_timestamp\_b

psis\_stsp\_timestamp\_a

psis\_stsp\_timestamp\_b

psis\_clk

PCLK

PRESETn

psis\_rst\_n

RX\_DATA

**NSMVG4SSV01**

ts

ts

is

Figure 1.2: Block diagram of SC-HEAP verification environment

***Explanation***:

* Dummy Master model is used to issue transactions to PSIS011 model via APB bus (refer to chapter 1.4 in the detail). This model has one initiator socket “is”.
* Dummy Peripheral model is used to control the input signals to PSIS011 model. Besides, this model receives and stores value of the output signals issued from PSIS011 model (refer to chapter 1.5 in the detail).

## Dummy Master model specification

### Summary

Dummy Master model is used to issue a transaction to the PSIS011 model directly or indirectly via APB Bus. It is implemented as DummyMasterRvc class.

**Legend:**

TLM socket

Verification models

Bus decoder

**Dummy**

**Master**

**Register handler**

Python IF

**is**

Figure 1.3: Block diagram of Dummy Master model

***Explanation***:

* Dummy Master is modeled with “Register handler” block. This block stores registers and controls operation of Dummy Master model.
* Besides, this model can issue transactions to the PSIS011 model directly or indirectly via APB Bus thanks to an initiator socket “is”.

### Registers

The registers of Dummy Master model are described in the Table 1.1.

Table 1.1: List of Dummy Master’s registers

| **Register** | **Address offset** | **Initial value** | **Bit** | **Access** | **Description** |
| --- | --- | --- | --- | --- | --- |
| CTRL\_REG | 0x00 | 0x0 | 0 - 15 | R/W | Control the transaction to slave  - 0x1: Issue a transaction to slave  - Other values: Ignored |
| DEBUG\_MODE\_REG | 0x04 | 0x0 | 0 | R/W | Store transaction mode  - 0x0: Normal transaction  - 0x1: Debug transaction |
| EXT\_REG | 0x08 | 0x0 | 0 - 31 | R/W | Store the value of TlmG3mExtension  - Bit[0] : VM  - Bit[1] : UM  - Bit[4-6] : PEID  - Bit[8-12] : SPID  - Bit[16-18]: VCID  - Bit[24-29]: TCID |
| ADDR\_REG | 0x0C | 0x0 | 0 - 31 | R/W | Store the transaction address |
| SIZE\_REG | 0x10 | 0x0 | 0 - 7 | R/W | Store transaction size |
| CMD\_REG | 0x14 | 0x0 | 0 | R/W | Store the transaction command  - 0x0: Read transaction  - 0x1: Write transaction |
| WR\_DATA\_REG | 0x18 | 0x0 | 0 - 31 | R/W | Store data of write transaction |
| RD\_DATA\_REG | 0x1C | 0x0 | 0 - 31 | R | Store data of read transaction |

### Operation

User task

Model task

**Legend**

Configure transaction information

(EXT\_REG, ADDR\_REG, SIZE\_REG, WR\_DATA\_REG, CMD\_REG)

Configure CTRL\_REG register

Issue a transaction to the PSIS011 model

[CTRL\_REG == 1]

[else]

Figure 1.4: Operation flow of Dummy Master model

***Explanation***:

* When user writes value “0x1” to CTRL\_REG register, a transaction will be issued to the PSIS011 model via an initiator socket “is”. Otherwise, there is no transaction issued.

## Dummy Peripheral model specification

### Summary

Dummy Peripheral model is used to control the input signals of PSIS011 model. Besides, this model receives and store value of the output signals of PSIS011 model for checking value. It is implemented as the DummyPeripheralRvc class.

**SC-HEAP Environment**

**Dummy**

**Peripheral**

**Port handler**

Python IF

sdfj

**Register handler**

**Legend:**

Ports connection

TLM socket

Verification models

Bus decoder

Design under test

Internal controller

n: from 0 to 7

m: from 1 to 7

psis\_mult\_clk

psis\_mult\_rst\_n

RX\_CONTROL

TX\_DATA

TX\_CONTROL

int\_psis\_chn

dma\_psis\_chn\_rx

dma\_psis\_chm\_tx

psis\_trig\_sync\_chn

psis\_clk\_timestamp\_a

**UART**

**Interrupt**

**DMA**

**Clock/reset**

**GTM IF**

psis\_clk\_timestamp\_b

psis\_clr\_timestamp\_a

8

8

7

8

psis\_clr\_timestamp\_b

psis\_stsp\_timestamp\_a

psis\_stsp\_timestamp\_b

psis\_clk

PCLK

PRESETn

psis\_rst\_n

RX\_DATA

**PSIS011**

APB Bus

**NSMVG4SSV01 (for SC-HEAP environment only)**

ts

Figure 1.5: Block diagram of Dummy Peripheral model

***Explanation***:

* Dummy Peripheral is modeled with 2 blocks: “Register handler” stores registers and controls the operation of this model according register setting; and “Port handler” controls issuing/receiving input/output signals to/from the PSIS011 model.
* This model provides clocks (namely PCLK, psis\_clk, psis\_mult\_clk) to the PSIS011 model.
* Besides, this model can assert/de-assert reset ports (namely PRESETn, psis\_rst\_n, psis\_mult\_rst\_n) for verifying reset operation of the PSIS011 model.
* This model issues signals to the PSIS011's input ports and receives the output signals from the PSIS011 model for verifying operation of this model.
* For SC-HEAP environment, Dummy Peripheral’s TLM target socket “ts” is connect to bus model. Users can access read/write the Dummy Peripheral's registers through this target socket. On the other hands, in UT, this target socket “ts” is connect to Dummy Is model & unused.

### Registers

The registers of Dummy Peripheral model are described in the Table 1.2.

Table 1.2: List of Dummy Peripheral’s registers

| **Register** | **Address offset** | **Initial value** | **Bit** | **Access** | **Description** |
| --- | --- | --- | --- | --- | --- |
| JUDGE\_REG | 0x00 | 0x0 | 0 | R|W | Store the simulation result  - JUDGE[0]: Judge bit  + 0x0 : Pass  + 0x1 : Fail |
| Clock/reset | | | | | |
| RESET\_REG | 0x04 | 0x3 | 0 - 1 | R|W | Store the values of output reset ports  - BUSRST[0] for value PRESETn port |
| PSIS\_RST\_N | 0x08 | 0x3 | 0 - 1 | R|W | Store the values of output reset ports  - BUSRST[0] for value psis\_rst\_n port |
| PSIS\_MULT\_RST\_N | 0x0C | 0x3 | 0 - 1 | R|W | Store the values of output reset ports  - BUSRST[0] for value psis\_mult\_rst\_n port |
| PCLK\_REG | 0x10 | 0x0 | 0 - 31 | R|W | Store the value of output port “PCLK” |
| PSIS\_CLK | 0x18 | 0x0 | 0 - 31 | R|W | Store the value of output port “psis\_clk” |
| PSIS\_MULT\_CLK | 0x20 | 0x0 | 0 - 31 | R|W | Store the value of output port “psis\_mult\_clk” |
| UART | | | | | |
| RX\_DATA\_REG | 0x30 | 0x0 | 0 - 31 | R|W | Store the value of output port RX\_DATA |
| RX\_CONTROL\_REG | 0x34 | 0x0 | 0 - 31 | R|W | Store the value of output port RX\_CONTROL |
| TX\_DATA\_REG | 0x38 | 0x0 | 0 - 31 | R | Store the value of input port TX\_DATA |
| TX\_CONTROL \_REG | 0x3C | 0x0 | 0 - 31 | R | Store the value of input port TX\_CONTROL |
| Interrupts | | | | | |
| INT\_PSIS\_CHn \_REG | 0x40 + 4\*n | 0x0 | 0 - 31 | R | Store the value of input ports int\_psis\_chn |
| DMA | | | | | |
| DMA\_PSIS\_CHn\_RX \_REG | 0x60 + 4\*n | 0x0 | 0 - 31 | R | Store the value of input ports dma\_psis\_chn\_rx |
| DMA\_PSIS\_CHm\_TX \_REG | 0x80 + 4\*m | 0x0 | 0 - 31 | R | Store the value of input ports dma\_psis\_chm\_tx |
| GTM IF | | | | | |
| PSIS\_TRIG\_SYNC\_CHn \_REG | 0xA0 + 4\*n | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_trig\_sync\_chn |
| PSIS\_CLK\_TIMESTAMP\_A \_REG | 0xC0 | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_clk\_timestamp\_a |
| PSIS\_CLK\_TIMESTAMP\_B \_REG | 0xC4 | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_clk\_timestamp\_b |
| PSIS\_CLR\_TIMESTAMP\_A \_REG | 0xC8 | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_clr\_timestamp\_a |
| PSIS\_CLR\_TIMESTAMP\_B \_REG | 0xCC | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_clr\_timestamp\_b |
| PSIS\_STSP\_TIMESTAMP\_A \_REG | 0xD0 | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_stsp\_timestamp\_a |
| PSIS\_STSP\_TIMESTAMP\_B \_REG | 0xD4 | 0x0 | 0 - 31 | R|W | Store the value of output ports psis\_stsp\_timestamp\_b |
| PSIS\_FRAME\_TYPE\_REG | 0xD8 | 0x0 | 0 – 31 | R|W | Store the value of frame type |
| PSIS\_DDSR\_DATA\_REG | 0xDC | 0x0 | 0 – 31 | R|W | Store the DDSR transmit data |
| PSIS\_PARITY\_REG | 0xE4 | 0x0 | 0 – 31 | R|W | Store the value of parity |
| PSIS\_TX\_DATA\_REG | 0xE8 | 0x0 | 0 – 31 | R|W | Store the value of TX data |
| PSIS\_EXP\_REG | 0xEC | 0x0 | 0 - 31 | R|W | Store the expected value |

**Notes:**

n is from 0 to 7

m is from 0 to 6

### Operation

#### Receiving input signals

Notify input port changed

Write port value into corresponding registers

[Input port is triggered]

[else]

Figure 1.6: Operation flow of the Dummy Peripheral about receiving input signals

***Explanation***:

* If the input port changes, an info message is dumped to inform the receiving input signal from PSIS011 model and the value is stored into the corresponding register (refer to Table 1.2 for relationship between registers and corresponding input ports).
* Users can get the value of corresponding register above to check values notified from PSIS011 model.

#### Issuing output signals

Write value to registers

User task

Model task

**Legend**

Write the value to corresponding ports

Notify issuing signal

Figure 1.7: Operation flow of the Dummy Peripheral about issuing output signals

***Explanation***:

* Dummy Peripheral model provides clock signals to PSIS011 model via PCLK, psis\_clk, and psis\_mult\_clk output ports.
* For reset operation, Dummy Peripheral model issues reset signals to PSIS011 model via PRESETn, psis\_rst\_n, and psis\_mult\_rst\_n output ports.
* Besides, this model issues output signals to PSIS011 model to verify main operation of PSIS011 model.
* In SC-HEAP environment, when users write value to register via “ts” socket, this value is written to corresponding output port right after the value of register is changed. (refer to Table 1.2 for relationship between registers and corresponding output ports).
* When output port is written, an info message is dumped to inform the issuing output signal.

# **Environment Structu**re

**PSIS011\_Output**

**scripts\_windows**

setup\_\*.bat

compile\*.bat

run\*.bat

check\_results.bat

run\_all\_\*.bat

**check\_result**

readme.txt

**src**

**ENV**

**log**

**pat**

**results**

**reports**

**sim**

**scripts\_linux**

**check\_result**

**run\_all**

setup\_\*.csh

run\_all\_\*.csh

check\_result\_\*.pl

**tb**

**scheapCompile**

**build**

**models\***

**pltfrmCompile**

The instruction file

Store the source code of developed target model (the ICUS221) model)

Store simulation environment

Store simulation execution log files

Store the test patterns

Store simulation results

Store simulation reports

Store all generated files used for simulation

Store all scripts to run simulation on Linux

The script for setting environment on Linux

Store expected simulation result on Linux

The script for checking the results of simulation on Linux

Store script to run all steps

The script for running all steps in the Linux environment

Store all scripts to run simulation on Windows

The script for setting environment on Windows

The script for compile the Windows simulation environment

The script for running simulations in the Windows environment

The script for checking the results of simulation on Windows

The script for running all steps in the Windows environment

Store expected verification result on Windows

Store SC-HEAP E4 environment core

Store SC-HEAP compilation

Store files for compiling the SC-HEAP environment

Store the models

The user modeling environment

The Unit Test environment

**unitTest**

**Legend:**

**Folder**

**File**

Figure 2.1: Verification environment structure

## How to verify on UT

* Verification on Linux should be done first before moving to Windows verification although there is no need to compile Python TMs.
* Following are verification steps:

1. *Verify on Linux*: The flow of verification on Linux is explained in chapter 2.3. The scripts “run\_all\_osci\_ut\_64bit.csh” and “run\_all\_usk\_ut\_64bit.csh” in “scripts\_linux/run\_all” folder can be used to perform all steps automatically right after "Setup environment" step.
2. *Moving to Windows*: The environment after verifying on Linux should be used for verifying on Windows. The “sim” folder is required.
3. *Verify on Windows*: The flow of verification on Windows is explained as in chapter 2.4. The scripts “run\_all\_osci\_ut\_64bit.bat”/“run\_all\_usk\_ut\_64bit.bat” in “scripts\_windows” folder can be used to perform all steps automatically.

## How to verify on SC-HEAP environment

* Verification on Linux should be done first before moving to Windows verification so that the TMs can be compiled (only be done on Linux).
* Following are verification steps:

1. *Verify on Linux*: The flow of verification on Linux is explained in chapter 2.3. The scripts “run\_all\_osci\_64bit.csh” and “run\_all\_usk\_64bit.csh” in “scripts\_linux/run\_all” folder can be used to perform all steps automatically right after "Setup environment" step.
2. *Moving to Windows*: The environment after verifying on Linux should be used for verifying on Windows. The “sim” folder is required.
3. *Verify on Windows*: The flow of verification on Windows is explained as in chapter 2.4. The scripts “run\_all\_osci\_64bit.bat”/“run\_all\_usk\_64bit.bat” in “scripts\_windows” folder can be used to perform all steps automatically.

## Verification environment on Linux

### Verification steps

The verification flowchart on Linux is shown in Figure 2.2.

The detailed explanation is described in Table 3.1

Setup environment

Compile environment

Compile test pattern

setup\_osci\_64bit.csh/

setup\_usk\_64bit.csh

Create run batch

Run simulation

Check result

Make report

gen\_mot.pl

gen\_sim\_osci.pl/

gen\_sim\_usk.pl

run\_all\_osci\*.csh/

run\_all\_usk\*.csh

check\_results.pl

gen\_report\_osci.pl/

gen\_report\_usk.pl

Task name

Script name

**Legend**

Prepare Environment

Prepare

test pattern

Run simulation

Check and report

Figure 2.2: Flow chart of verification on Linux

Table 2.1: Explanation of verification on Linux

|  |  |
| --- | --- |
| **Step** | **Explanation** |
| - Setup environment  (*setup\_osci\_64bit.csh/ setup\_usk\_64bit.csh)*  - Compile environment | Setting for UT, and SC-HEAP environment by edit and source *setup\_osci\_64bit.csh/ setup\_usk\_64bit.csh* environment file. And verification SC-HEAP environment is compiled. |
| - Compile test patterns  (*gen\_mot.pl*) | The *gen\_mot.pl* script is used to compile all test patterns.  Please setup the GHS license before compiling the TMs for the SC-HEAP environment |
| - Create run batch  (*gen\_sim\_osci\*.pl/ gen\_sim\_usk\*.pl)*  - Run simulation | In order to run simulation automatically, run batch is created by *gen\_sim\_osci\*.pl/ gen\_sim\_usk\*.pl* script; then simulation is done by running all created test pattern.  For running the whole environment, please use the *run\_all\_osci\*.csh* or the *run\_all\_usk\*.csh* to run with a compatible library.  The ASTC requires source its setting license before running the environment. |
| - Check result  (*check\_result.pl)*  - Make report  (*gen\_report\_osci.pl/ gen\_report\_usk.pl)* | The results are made by *check\_result.pl* script and reports are created by *gen\_report\_osci.pl/ gen\_report\_usk.pl* script in order to express Pass/Fail information. |

***Note***:

1. *All scripts for verification on Linux are stored in “scripts\_linux” folder. The script “run\_all\_osci\*.csh”/ “run\_all\_usk\*.csh” calls “gen\_mot.pl”, “gen\_sim\_osci\*.pl/gen\_sim\_usk\*.pl”, “check\_result.pl” and “gen\_report\_osci.pl”/ “gen\_report\_usk.pl” to run all steps automatically for verification on Linux.*
2. All the scripts which has “\_ut” suffix are reserved for UT environment.

## Verification environment on Windows

### Verification steps

The verification flowchart on Windows is shown in Figure 2.3.

The detailed explanation is described in Table 2.2.

Setup environment

Compile environment

setup\_osci\_64bit.bat/

setup\_usk\_64bit.bat

Run simulation

Check result

compile\*.bat

check\_results.bat

run\_all\_osci\*.bat/

run\_all\_usk\*.bat

Task name

Scripts name

**Legend**

Prepare Environment

Run simulation

Check results

run\*.bat

Figure 2.3: Flow chart of verification on Windows

Table 2.2: Explanation of verification on Windows

|  |  |
| --- | --- |
| **Step** | **Explanation** |
| Setup environment  (*setup\_osci\_64bit.bat/setup\_usk\_64bit.bat*) | Edit the script to set all the environment variables to specify options for simulation, including.  The requirement mode for the UT, and SC-HEAP environment is the “Release” mode. |
| Compile the environment  (*compile\*.bat*) | Compile the Visual C++ solution which includes the SC-HEAP E4 VC++ project. |
| Run simulation for all test patterns  (*run\*.bat*) | Run all the test patterns. Output log files will be generated and stored in “log” folder. |
| Check the results of simulation  (*check\_results.bat*) | Check the results of simulation by confirm PASS/FAIL number. The results are stored in “results” folder. |

***Note***:

1. *All scripts for verification on Windows are stored in “scripts\_windows” folder. The script “run\_all\_osci\*.bat”/ “run\_all\_usk\*.bat” calls “setup\_osci\*.bat”/ “setup\_usk\*.bat”, “compile\*.bat”, “run\*.bat” and “check\_results.bat” to run all steps automatically for verification on Windows.*
2. All the scripts which has “\_ut” suffix are reserved for UT environment.

# Verification conditions

Verification conditions are described in Table 3.1.

Table 3.1: Verification conditions

|  |  |  |
| --- | --- | --- |
| **Group** | **Target** | **Condition** |
| Machine | Linux | Red Hat Enterprise 6 (64 bits) |
| Windows | Windows 10 (64 bits) |
| Tool | Compiler | gcc 4.9.3 |
| Visual Studio 2015 |
| Style checker (\*) | 1Team:System 1.16.5 |
| Code coverage | gcov in gcc\_4.9.3 |
| Memory check (\*\*) | Valgrind v3.7.0 |
| Python I/F | Python 2.7.3 (64bit) |
| Embedded software tool | GHS MULTI 6.1.4 |
| rteserv2 |
| Library | System C, TLM | OSCI SystemC v2.3.1a |
| VWorks OSCAR for vlab2.3.6 |
| Environment | SC-HEAP | - SC-HEAP G4 ver163 |
| Unit Test | - There are 3 models in this environment: Dummy Master model, Dummy Peripheral model, Dummy Is, and PSIS011 model, where:  - Dummy Master model issues TLM transactions to PSIS011 directly.  - Dummy Peripheral model issues/checks input/output ports of PSIS011.  - Dummy Is model connects unused target sockets of Dummy Peripheral in UT. |

***Note***: *(\*) – 1Team is not ready to be used due to license issue.*

*(\*\*) - Memory leakage is ignored.*

# Verification requirements

Verification requirements are described in Table 4.1.

Table 4.1: Verification requirement

|  |  |
| --- | --- |
| **Requirement** | **Target** |
| Compile | No error and no warning |
| Code coverage | Line coverage is 100% |
| Functional coverage | 100% on traceability table |
| Style check | Run 1TeamSystem with option template=Renesas/Modeling” |
| Memory check | No error and warning bout target source code |
| Test pattern | Refer to VRF-MCS-17013-01\_PSIS011.xls |

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Modified points** | **Approver** | **Checker** | **Author** |
| 1.0 | - Created new | Binh Nguyen  10/10/2017 | ChanLe  10/10/2017 | ChuongLe  09/14/2017 |
| 1.1 | - Add Dummy Is to Figure 1.1  - Add target socket “ts” to Dummy Peripheral to Figure 1.2  - Update registers of Dummy Peripheral in Table 1.2  - Add How to verify on UT into Environment Structure section  - Revise script names in Figure 2.2 for adding UT  - Revise script names in Figure 2.3 for adding UT  - Add Dummy Is into Table 3.1 | Binh Nguyen  12/19/2017 | ChanLe  12/19/2017 | ChuongLe  12/19/2017 |
| 1.2 | - Update [chapter 2](#_Environment_Structure) for new environment 64bit.  - Update Table 3.1 for new environment 64bit. | Chan Le  02/24/2018 | Chuong Le  02/24/2018 | Chan Le  02/24/2018 |