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_EMSS-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

Index of Figures

Index of Tables

1. Summary

1.1. Introduction

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

1.2. 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.

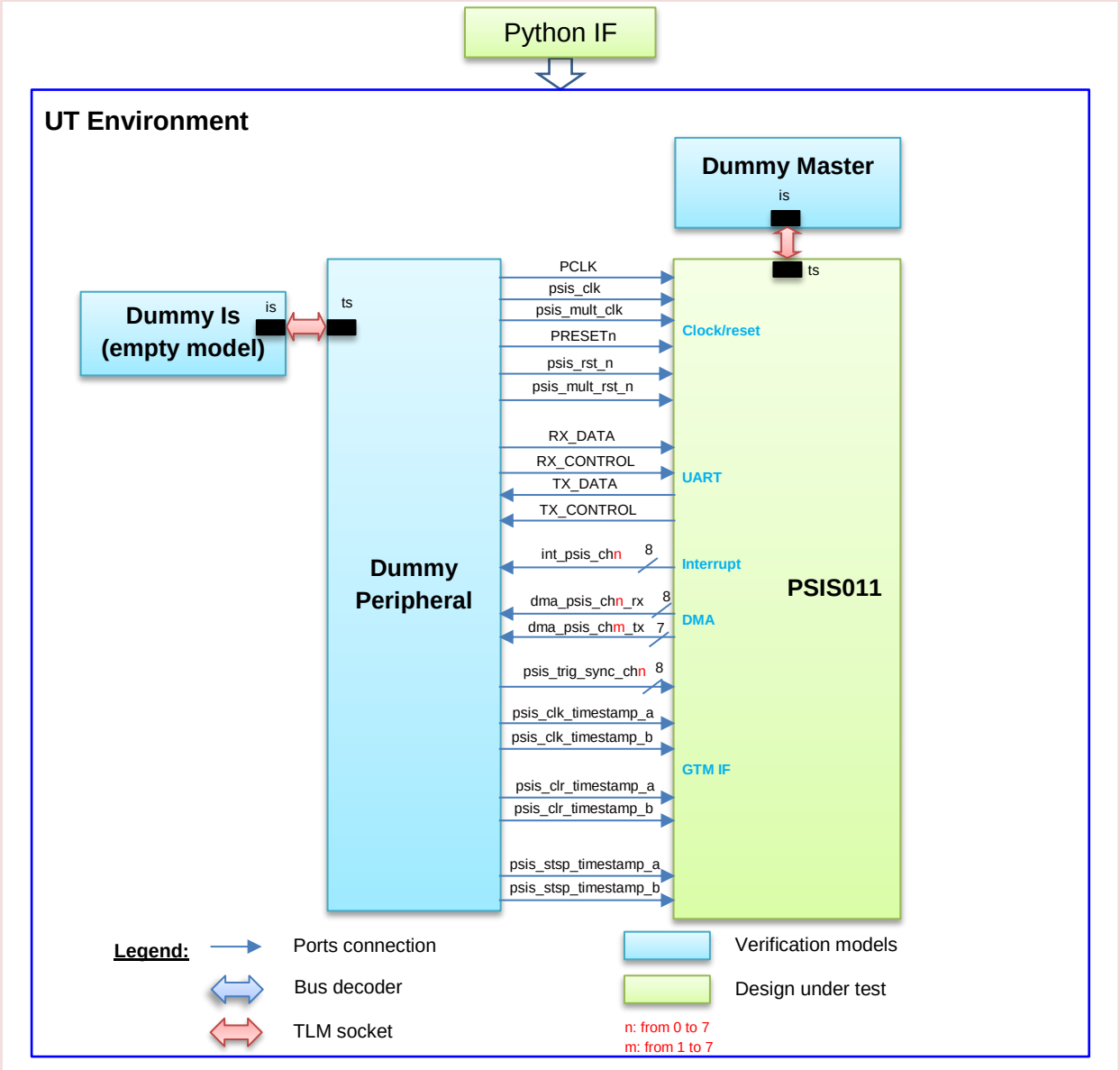


Figure 1.1: Block diagram of Unit Test environment

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.

1.3. 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.

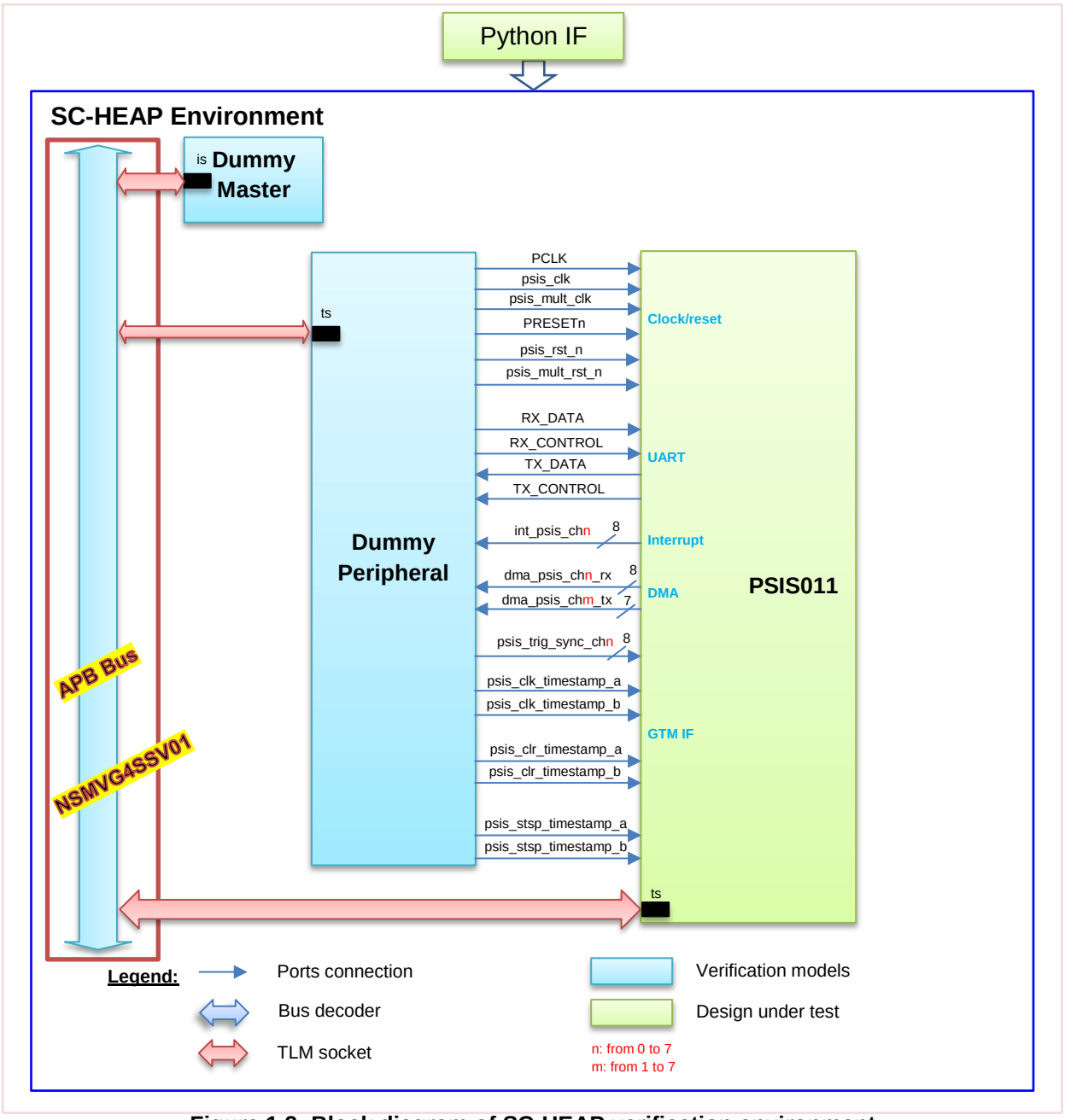


Figure 1.2: Block diagram of SC-HEAP verification environment

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).

1.4. Dummy Master model specification

1.4.1. 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.

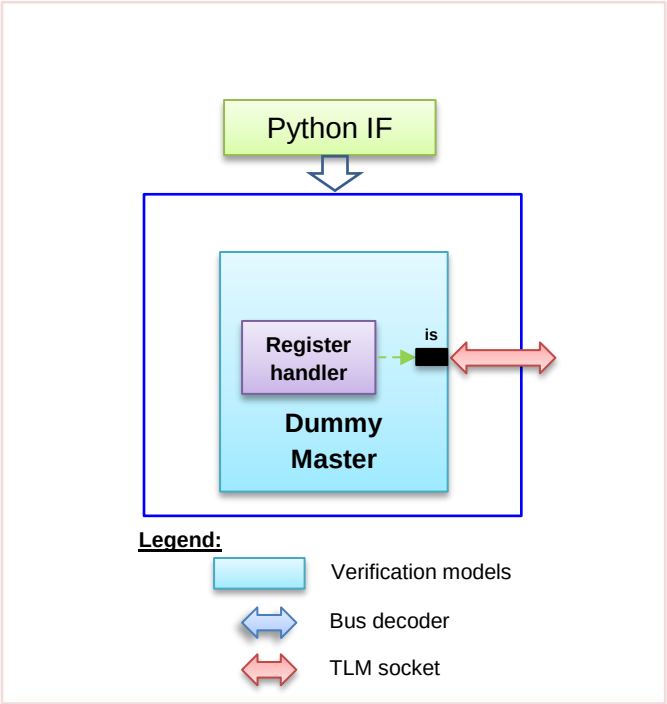


Figure 1.3: Block diagram of Dummy Master model

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”.

1.4.2. 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

Register	Address offset	Initial value	Bit	Access	Description
					- 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

1.4.3. Operation

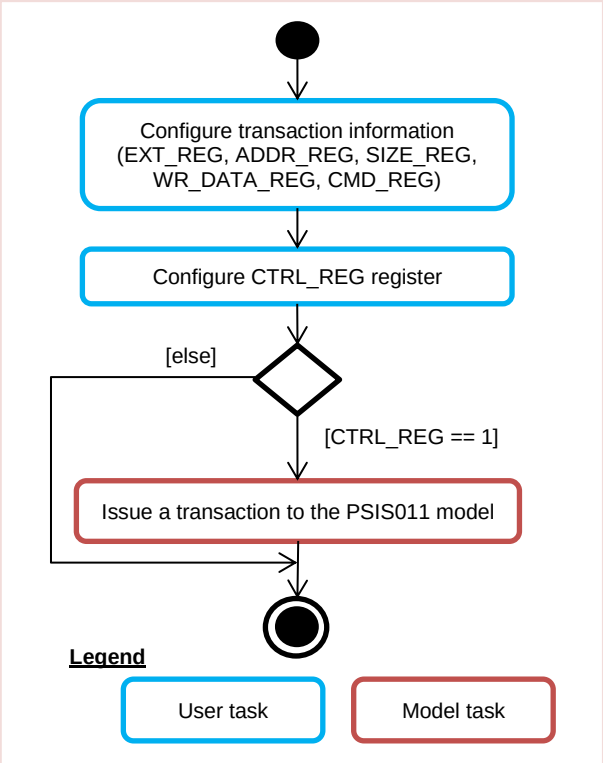


Figure 1.4: Operation flow of Dummy Master model

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.

1.5. Dummy Peripheral model specification

1.5.1. 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.

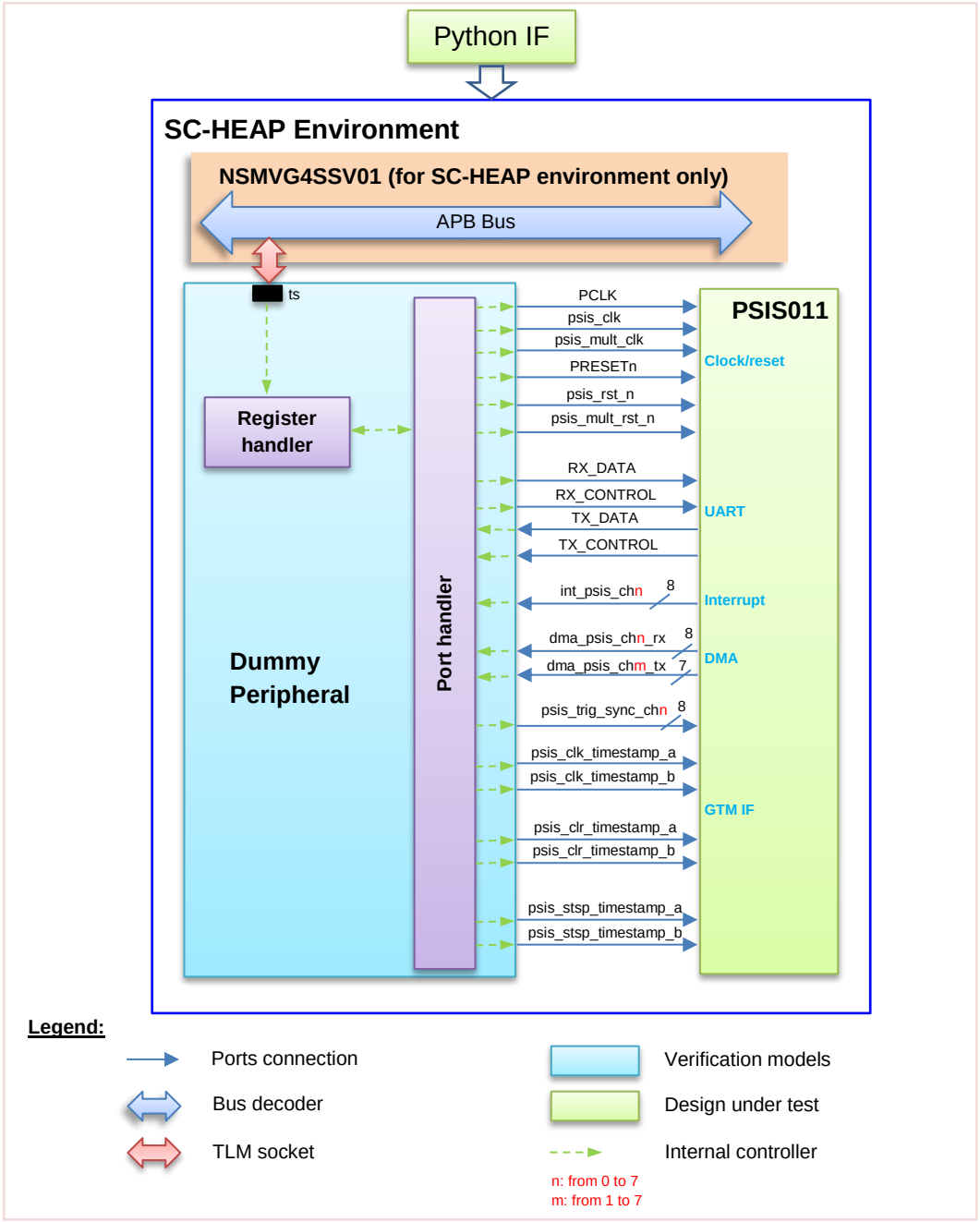


Figure 1.5: Block diagram of Dummy Peripheral model

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.

1.5.2. 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

Register	Address offset	Initial value	Bit	Access	Description
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:

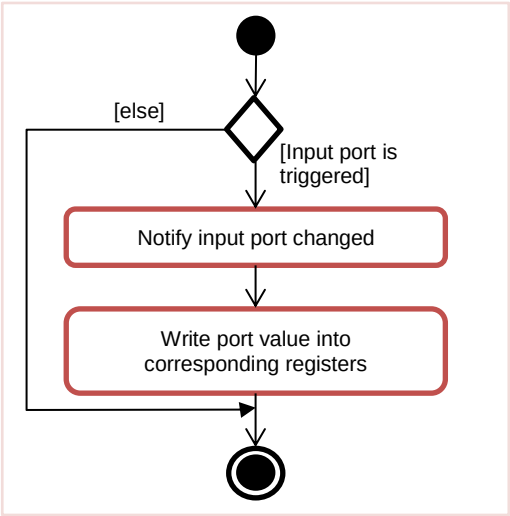
n is from 0 to 7

m is from 0 to 6

1.5.3. Operation

1.5.3.1. Receiving input signals

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.

1.5.3.2. Issuing output signals

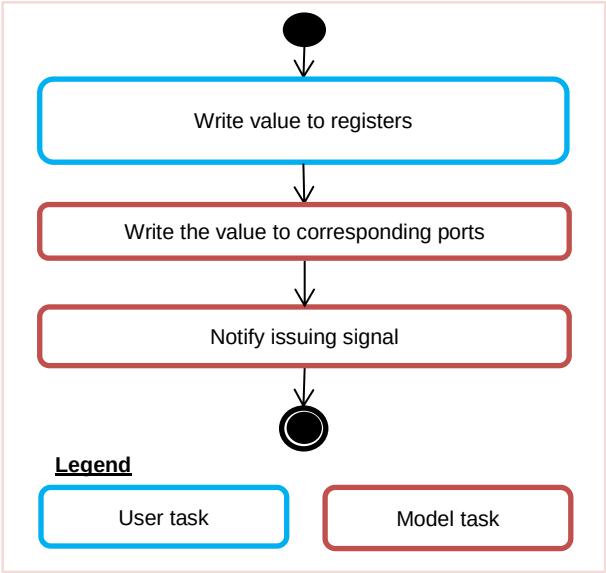


Figure 1.7: Operation flow of the Dummy Peripheral about issuing output signals

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.

2. Environment Structure

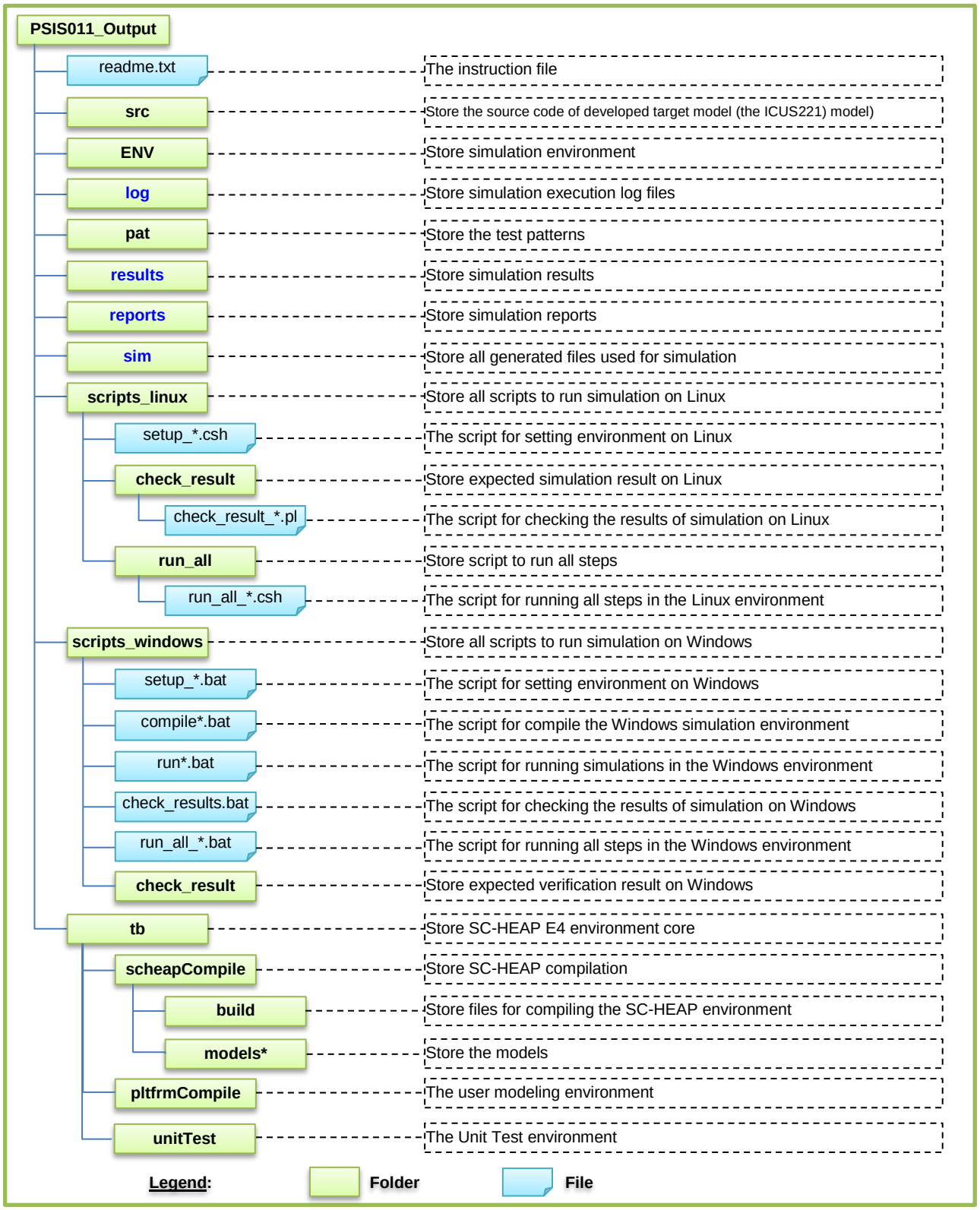


Figure 2.1: Verification environment structure

2.1. 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.

2.2. 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.

2.3. Verification environment on Linux

2.3.1. Verification steps

The verification flowchart on Linux is shown in Figure 2.2.

The detailed explanation is described in Table 3.1

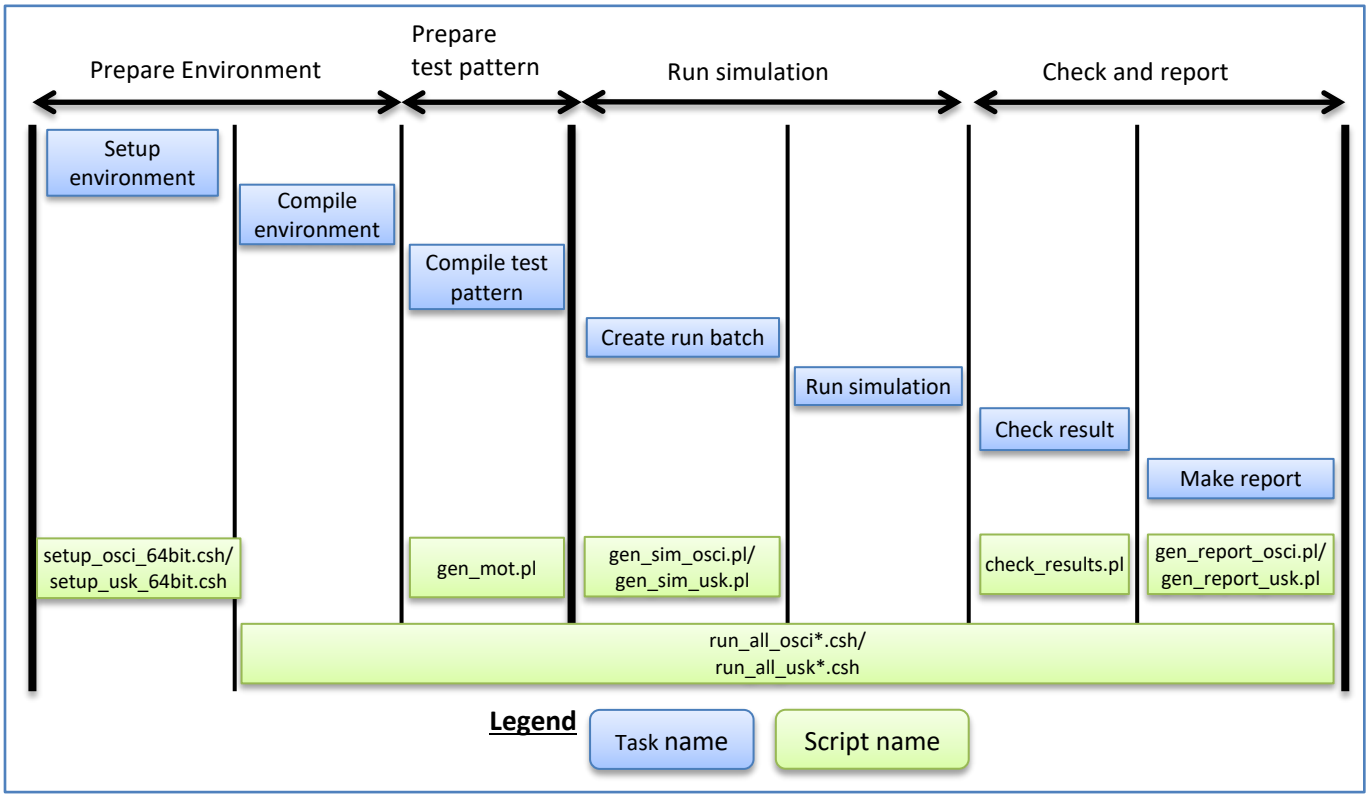


Figure 2.2: Flow chart of verification on Linux

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.

2.4. Verification environment on Windows

2.4.1. Verification steps

The verification flowchart on Windows is shown in Figure 2.3**.**

The detailed explanation is described in Table 2.2.

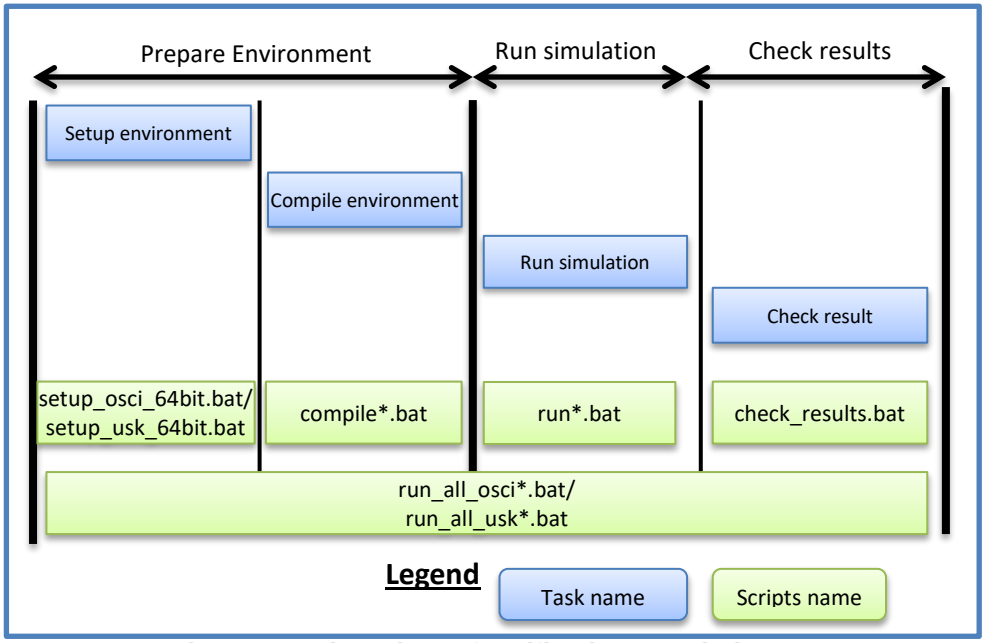


Figure 2.3: Flow chart of verification on Windows

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.

3. 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.

4. 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 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