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

|  |
| --- |
| **Development of SPIDCTL model**  **for multiple products**  (v1.1) |

|  |
| --- |
| **Summary:** |
| This document describes the verification methodology and verification procedure used to verify SPIDCTL 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-20001\_SPIDCTL (\*) | - | Detail requirement of SPID model  (***File***: REQ-MCS-20001\_SPIDCTL.xlsx) | **SVN:**  http://172.29.139.78/mcu\_modeling\_01/release/docs/REQ/2020/20001\_U2A\_SPIDCTL |
| 4 | VRF-MCS-20001-01\_SPIDCTL (\*) | - | Checklist of SPID model  (***File***: VRF-MCS-20001-01\_SPIDCTL.xls) | **SVN:**  http://172.29.139.78/mcu\_modeling\_01/release/docs/VRF/2020/20001\_U2A\_SPIDCTL |

***Note****: (\*) Refer to DEV-MCS-20001\_SPIDCTL* *for version number of REQ-MCS-20001\_SPIDCTL and VRF-MCS-20001-01\_SPIDCTL*

Table of Contents

[1. Summary 6](#_Toc87961443)

[1.1. Introduction 6](#_Toc87961444)

[1.2. Block diagram of SC-HEAP environment 6](#_Toc87961445)

[1.3. Dummy Master model specification 7](#_Toc87961446)

[1.4. Dummy Peripheral model specification 9](#_Toc87961447)

[2. Environment Structure 14](#_Toc87961448)

[2.1. How to verify on SC-HEAP environment 15](#_Toc87961449)

[2.2. Verification environment on Linux 15](#_Toc87961450)

[2.3. Verification environment on Windows 16](#_Toc87961451)

[2.4. How to connect Verification Environment 17](#_Toc87961452)

[3. Verification conditions 18](#_Toc87961453)

[4. Verification requirements 19](#_Toc87961454)

**Index of Figures**

[Figure 1.1: Block diagram of SC-HEAP verification environment 6](#_Toc87961455)

[Figure 1.2: Block diagram of Dummy Master model 7](#_Toc87961456)

[Figure 1.3: Operation flow of Dummy Master model 9](#_Toc87961457)

[Figure 1.4: Block diagram of Dummy Peripheral model 10](#_Toc87961458)

[Figure 1.5: Operation flow of the Dummy Peripheral about receiving input signals 12](#_Toc87961459)

[Figure 1.6: Operation flow of the Dummy Peripheral about issuing output signals 12](#_Toc87961460)

[Figure 2.1: Verification environment structure 15](#_Toc87961461)

[Figure 2.2: Flow chart of verification on Linux 16](#_Toc87961462)

[Figure 2.3: Flow chart of verification on Windows 17](#_Toc87961463)

**Index of Tables**

[Table 1.1: List of Dummy Master’s registers 8](#_Toc87961464)

[Table 1.2: List of Dummy Peripheral’s registers 11](#_Toc87961465)

[Table 2.1: Explanation of verification on Linux 16](#_Toc87961466)

[Table 2.2: Explanation of verification on Windows 17](#_Toc87961467)

[Table 3.1: Verification conditions 18](#_Toc87961468)

[Table 4.1: Verification requirement 19](#_Toc87961469)

# Summary

## Introduction

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

## Block diagram of SC-HEAP environment

In this project, the SC-HEAP environment is employed to verify the SPIDCTL model. Some registers, and basic operations are verified on SC-HEAP to make sure the SPIDCTL model can operate normally on SC-HEAP environment. Figure 1.1 shows the block diagram of the SC-HEAP environment.

Python IF

**SC-HEAP Environment**

**NSMVG4SSV01**

APB Bus

**SPIDCTL**

ts

VCI2APB

I\_BRIDGE

**Dummy**

**Master**

is

**Dummy**

**Peripheral**

ts

PCLK

Reset

**Clock**

**Reset**

ts

BMSPIDOUTn (\*1)

**Legend:**

Ports connection

TLM socket

Verification models

Bus decoder

Design under test

Figure 1.1: Block diagram of SC-HEAP verification environment

***Note***:

1. n = 0 ~ 63

***Explanation***:

* Dummy Master model is used to issue transactions to SPIDCTL model via APB bus (refer to chapter 1.3 in the detail). This model has one initiator socket “is”.
* Dummy Peripheral model is used to control the input signals to SPIDCTL model. Besides, this model receives, stores value of the output signals issued from SPIDCTL model (refer to chapter 1.4 in the detail).

## Dummy Master model specification

### Summary

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

****

**Dummy Master**

**Legend:**

TLM socket

Verification models

Bus decoder

**Register handler**

Python IF

**is**

APB Bus

**NSMVG4SSV01**

**ts**

**VCI2APB**

Figure 1.2: 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 SPIDCTL 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[0-7] : PEID  - Bit[8-12] : SPID  - Bit[13] : VM  - Bit[14] : UM  - Bit[16-18] : VCID  - Bit[20-23] : VCIREQ  - Bit[24-29] : TCID  - Bit[31] : SEC |
| 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 SPID model

[CTRL\_REG == 1]

[else]

Figure 1.3: Operation flow of Dummy Master model

***Explanation***:

* When user writes value “0x1” to CTRL\_REG register, a transaction will be issued to the SPIDCTL 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 SPIDCTL model. Besides, this model receives and store value of the output signals of SPIDCTL model for checking value. It is implemented as the DummyPeripheralRvc class.

Python IF

sdfj

**SC-HEAP Environment**

APB Bus

**NSMVG4SSV01**

**VCI2APB**

ts

**SPIDCTL**

**Dummy**

**Peripheral**

**Port handler**

PCLK

Reset

**Clock**

**Reset**

**Register handler**

BMSPIDOUTn

(\*1)

Figure 1.4: Block diagram of Dummy Peripheral model

***Note***:

1. n = 0 ~ 63

***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 SPIDCTL model.
* This model provides clocks (namely PCLK) to the SPIDCTL model.
* Besides, this model can assert/de-assert reset ports (namely Reset) for verifying reset operation of the SPIDCTL model.
* This model issues signals to the SPIDCTL input ports and receives the output signals from the SPIDCTL 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 |
| RESET\_REG | 0x04 | 0x1 | 0 | R|W | Store the values of output reset ports: + Bit 0: Value of Reset |
| PCLK\_LOW\_REG | 0x08 | 0x0 | 0 - 31 | R|W | The 32-bit lower part of 64-bit frequency values of the clock sources output to the DUT. |
| PCLK\_HIGH\_REG | 0x0C | 0x0 | 0 - 31 | R|W | The 32-bit higher part of 64-bit frequency values of the clock sources output to the DUT. |
| BMSPIDOUT\_REGn  (\*1) | 0x10 | 0x0 + n  (\*1) | 0-4 | R | Store the value of input port BMSPIDOUTn (\*1) |

***Note***:

1. n = 0 ~ 63

### Operation

#### Receiving input signals

Notify input port changed

Write port value into corresponding registers

[Input port is triggered]

[else]

Figure 1.5: 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 SPIDCTL model, 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 SPIDCTL model.

#### Issuing output signals

Write value to registers

User task

Model task

**Legend**

Write the value to corresponding ports

Notify issuing signal

Figure 1.6: Operation flow of the Dummy Peripheral about issuing output signals

***Explanation***:

* Dummy Peripheral model provides clock signals to SPIDCTL model via PCLK output ports.
* For reset operation, Dummy Peripheral model issues reset signals to SPIDCTL model via Reset output port.
* Besides, this model issues output signals to SPIDCTL model to verify main operation of SPIDCTL 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

**SPIDCTL\_Output\_64bit**

Store simulation execution log files

Store the test patterns

**log**

**pat**

Store the test patterns tp\*.py

Store the test patterns tp\*.c

**pat\_1\_\***

**pat\_2\_\***

Store simulation results

Store simulation reports

Store all generated files used for simulation

**results**

**reports**

**sim**

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

**scripts\_linux**

**check\_result**

**run\_all**

setup\_\*.csh

run\_all\_\*.csh

check\_result\_\*.pl

Store expected verification result on Windows

**scripts\_windows**

setup\_\*.bat

compile.bat

run.bat

check\_results.bat

run\_all\_\*.bat

**check\_result**

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 SC-HEAP G4 environment core

Store SC-HEAP compilation

Store files for compiling the SC-HEAP environment

Store the models

The user modeling environment

**tb**

**scheapCompile**

**build**

**models\***

**pltfrmCompile**

**Legend:**

**Folder**

**File**

The instruction file

Store the source code of developed target model SPIDCTL

Store simulation environment

readme.txt

**src**

**ENV**

Figure 2.1: Verification environment structure

## 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 2.2. The scripts “run\_all\_osci.csh” and “run\_all\_usk.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 2.3. The scripts “run\_all\_osci.bat”/“run\_all\_usk.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.csh/

setup\_usk.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.csh/ setup\_usk.csh)*  - Compile environment | Setting for SC-HEAP environment by edit and source *setup\_osci.csh/ setup\_usk.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.*

## 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.bat/

setup\_usk.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.bat/setup\_usk.bat*) | Edit the script to set all the environment variables to specify options for simulation, including.  The requirement mode for the SC-HEAP environment is the “Release” mode. |
| Compile the environment  (*compile.bat*) | Compile the Visual C++ solution which includes the SC-HEAP G4 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.*

## How to connect Verification Environment

1. The following steps should be done to connect SPIDCTL into environment.
   1. Prepare config\_file for SPID master with format [SPID\_BUS\_MASTER] = (n, m). n is SPID register location, m is SPID initial value

Example:

// GTM, BM10SPID

[SPID\_BUS\_MASTER] = (10, 10)

// RHSIF1, BM27SPID

[SPID\_BUS\_MASTER] = (27, 18)

* 1. Declare an instance of the SPIDCTL class in environment.

Example:

spidctl = new SPIDCTL("spidctl", 0, 0, config\_file);

* 1. Connect the target socket of SPIDCTL instance to corresponding initiator socket.
  2. Connect reset/clock port of SPIDCTL to corresponding ports.
  3. Connect corresponding output ports to corresponding ports.

1. The following steps should be done to connect Dummy Peripheral into environment.
   1. Declare an instance of the DummyPeripheralRvc class in environment

Example:

dummy\_peripheral = new DummyPeripheralRvc("dummy\_peripheral", 0, 0);

* 1. Connect the target socket of dummy\_peripheral instance to corresponding initiator socket.
  2. Connect reset/clock port of dummy\_peripheral to corresponding ports.
  3. Connect corresponding interrupt output ports to corresponding ports.

1. The following steps should be done to connect Dummy Master into environment.
   1. Declare an instance of the DummyMasterRvc class in environment.

Example:

dummy\_master = new DummyMasterRvc("dummy\_master", 0, 0, G4SS->mIssMode);

* 1. Connect the target/initiator socket of dummy\_master instance to corresponding target/initiator socket.
  2. Connect reset/clock port of dummy\_master corresponding to ports.

# 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-4.9.3-64bit:  + for SC-HEAP and SystemC.  + for cforestg4 and ForestUtilScIf. |
| - Microsoft Visual Studio Express 2015 for Windows Desktop (64 bits) |
| Style checker | 1Team:System 1.16.7 |
| Code coverage | - gcov v4.9.3 - dummyins.x ver1.0.1 tool for C1 coverage |
| Memory check | Valgrind v3.11.0 |
| Python I/F | Python 2.7.3 (64bit) |
| Embedded software tool | GHS MULTI 6.1.4 |
| Library | System C, TLM | OSCI SystemC v2.3.1a (gcc4.9.3 64bit) |
| VWorks OSCAR (USK) Vlab2.5.16 (64bit) |
| Environment | SC-HEAP | - SCHeapG4\_Rev370 |
| - There are 3 models in environment: Dummy Master model, Dummy Peripheral model, SPIDCTL model  - Dummy Master model issues TLM transaction to SPIDCTL model.  - Dummy Peripheral model issues/checks input/output ports of SPIDCTL. |

# 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 | Branch coverage (C1) is 100% Reason for non-coverage report must be specific with a suitable reason. |
| Functional coverage | 100% on traceability table |
| Style check | Run 1TeamSystem with option template=Renesas/Modeling” |
| Memory check | No error and warning about target source code |
| Test pattern | Refer to VRF-MCS-20001-01\_SPIDCTL.xls |
| Performance | Performance is not checked in this model because there is no timer, no output clock. |
| Simulation mode | Both loosely time mode (LT) and approximately time mode (AT) are supported |

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Modified points** | **Approver** | **Checker** | **Author** |
| 1.0 | - New creation | Chuong Le  Jan/15/2020 | Anh Nguyen  Jan/09/2020 | Khoa Nguyen  Dec/31/2019 |
| 1.1 | - *Block diagram of SC-HEAP environment,* *Dummy Peripheral model specification:* update to test 64 SPID ports (maximum register allocation)  - *Verification conditions:* update the latest tool version base on U2B 2.0 project plan  - *How to connect Verification Environment*: Add guideline for configuring SPID initial value base on bus master via config\_file, update SPIDCTL constructor base on REQ  - *Table 4.1: Verification requirement:* Add Simulation mode information  - Change Reqtify ID phase name from DD to UD | Chuong Le  Nov/17/2021 | Minh Ha  Nov/16/2021 | Duong Phan  Nov/08/2021 |

Copyright(c) 2020-2021 Renesas Electronics Corporation. All rights reserved.