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

|  |
| --- |
| **Development of INTC2G model for E2x**  (v1.6) |

|  |
| --- |
| **Summary:** |
| This document describes the verification methodology and verification procedure used to verify interrupt controller Guard (INTC2G) model. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reference Manuals** | | | | |
| **No.** | **Title name** | **Document number** | **Description** | **Path** |
| 1 | SC-HEAP\_G3 common requirement (v1.0) |  | The common requirement  (***File***: Common\_Requirement\_RVC.pdf) | **DMS:** |
| 2 | SC-HEAP G4 Platform functional specification |  | The document describes about SC-HEAP G4 Platform functional  (***File***: SC-HEAP\_G4\_platform\_G\_t.pdf) |
| 3 | REQ-MCS-17002\_INTC2G(\*) | REQ-MCS-17002\_INTC2G | Detail requirement of INTC2G model  (***File***: REQ-MCS-17002\_INTC2G.xlsx) | **DMS:** |
| 4 | VRF-MCS-17002-01\_INTC2G(\*) | VRF-MCS-17002-01\_INTC2G | Checklist of INTC2G model  (***File***: VRF-MCS-17002-01\_INTC2G.xls) |  |

***Note****: (\*) Refer to TRA-MCS-17002\_INTC2G and DEV-MCS-17002\_INTC2G for version number of REQ-MCS-17002\_INTC2G and VRF-MCS-17002-01\_INTC2G.*

Table of Contents

[1. Summary 6](#_Toc518397415)

[1.1. Introduction 6](#_Toc518397416)

[1.2. Block diagram 6](#_Toc518397417)

[1.3. Dummy Master model specification 7](#_Toc518397418)

[1.4. Dummy Peripheral model specification 11](#_Toc518397419)

[1.5. Dummy Slave model specification 14](#_Toc518397420)

[2. Environment Structure 18](#_Toc518397421)

[2.1. How to verify 19](#_Toc518397422)

[2.2. Verification environment on Linux 19](#_Toc518397423)

[2.3. Verification environment on Windows 21](#_Toc518397424)

[3. Verification conditions 22](#_Toc518397425)

[4. Verification requirements 22](#_Toc518397426)

**Index of Figures**

[Figure 1‑1: Block diagram of verification environment 6](#_Toc518022736)

[Figure 1‑2: Block diagram of Dummy Master model 8](#_Toc518022737)

[Figure 1‑3: Operation flow of the Dummy Master model 10](#_Toc518022738)

[Figure 1‑4: Block diagram of Dummy Peripheral 11](#_Toc518022739)

[Figure 1‑5: Operation flow of the Dummy Peripheral about receiving input signals 13](#_Toc518022740)

[Figure 1‑6: Operation flow of the Dummy Peripheral about issuing output signals 14](#_Toc518022741)

[Figure 1‑7: Block diagram of Dummy Slave model 15](#_Toc518022742)

[Figure 1‑8: Operation flow of the Dummy Slave model 17](#_Toc518022743)

[Figure 2‑1: Verification environment structure 18](#_Toc518022744)

[Figure 2‑2: Flow chart of verification on Linux 19](#_Toc518022745)

[Figure 2‑3: Flow chart of verification on Windows 21](#_Toc518022746)

**Index of Tables**

[Table 1.1: List of Dummy Master’s registers 9](#_Toc518022747)

[Table 1.2: List of Dummy Peripheral’s registers 12](#_Toc518022748)

[Table 1.3: List of Dummy Slave’s registers 16](#_Toc518022749)

[Table 2.1: Explanation of verification on Linux 20](#_Toc518022750)

[Table 2.2: Explanation of verification on Windows 21](#_Toc518022751)

[Table 3.1: Verification conditions 22](#_Toc518022752)

[Table 4.1: Verification requirement 22](#_Toc518022753)

# Summary

## Introduction

The purpose of this document is to describe a verification methodology and verification procedure used to verify INTC2G model.

## Block diagram

In this project, the SC-HEAP environment is reused and modified to verify the INTC2G model. The Figure 1‑1 shows the block diagram of the verification environment.

Python IF

**SC-HEAP Environment**



Figure 1‑1: Block diagram of verification environment

***Explanation***:

* Dummy Master model is used to issue transactions to INTC2G model via APB bus (refer to chapter 1.3 in the detail). This model has two initiator sockets “is” and “isp” (they are connected to APB bus). Beside, this model has one target socket “ts” also connected to APB bus.
* Dummy Peripheral model is used to control the input signals of INTC2G model. Besides, this model receives and stores values of the output signals issued from INTC2G model (refer to chapter 1.4 in detail). This model has one target socket (“ts” socket) and it is connected to APB bus.
* Dummy Slave model is used to confirm whether the transaction is transferred through INTC2G model successfully (refer to chapter 1.5 in detail). This model has one target socket “ts”, through which users can access read/write into this model’s registers. Besides, this model has other target socket (“tsp” socket) which receives transactions from INTC2G model.

## Dummy Master model specification

### Summary

Dummy Master model is used to issue a transaction to the APB bus. It is implemented as DummyMasterRvc class.

**SC-HEAP Environment**



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.
* This model has an APB target socket “ts”. Users can access read/write the Dummy Master's registers through this target socket.
* Besides, this model can issue transaction to the APB bus domain through initiator sockets are “is” and “isp”.

### 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 - 16 | R/W | Control the transaction to slaves  - 0x1: Issue a transaction to INTC2G for check model operation  - 0x2: Issue a transaction for debug register area  - 0x3: Issue a transaction with all extensions to INTC2G for check model operation.  - Bit[16] : ISHAVEEXT  ISHAVEEXT = 1: not set extension  ISHAVEEXT = 0: set extension  - 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



Figure 1‑3: Operation flow of the Dummy Master model

***Explanation***:

* The Dummy Master model receives the data from TLM target IF via target socket “ts” to setup the transaction information into registers such as EXT\_REG, ADDR\_REG, SIZE\_REG, WR\_DATA\_REG, CMD\_REG.
* When users write value “0x1” to CTRL\_REG register, a transaction will be issued to slaves in APB bus domain via initiator sockets “isp” or “is” depending on the value of ADDR\_REG. Otherwise, no transaction is issued.

## Dummy Peripheral model specification

### Summary

Dummy Peripheral model is used to control the input signals of INTC2G model. Besides, this model receives and stores value of the output signals of INTC2G model for checking values. It is implemented as the DummyPeripheralRvc class.

**SC-HEAP Environment**



Figure 1‑4: Block diagram of Dummy Peripheral

***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 INTC2G model.
* This model provides clock “PCLK” to INTC2G model.
* About reset component, the “resetPort” port is used to verify reset operation of INTC2G model. Beside, user can only write INTC2G’s registers when “resetPort” is not activated.
* This model issues signals to INTC2G's input ports and receives the output signals from INTC2G’s output port for verifying operation of this model.
* Dummy Peripheral model has an APB target socket “ts”. Users can access read/write the Dummy Peripheral's registers through this target socket.

### 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 resetPort  + Bit 0: Value of resetPort |
| CLK\_PCLK\_REG | 0x08 | 0x0 | 0-31 | R/W | Store the value of output port “PCLK” |
| ERR\_IRQ\_REG | 0x10 | 0x0 | 0 | R/W | Store the value of input port “err\_irq” |
| COUNT\_ERR\_IRQ\_REG | 0x14 | 0x0 | 0 | R/W | Store the value of the number error interrupt is issued |

### Operation

#### Receiving input signals



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 INTC2G 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 INTC2G model.

#### Issuing output signals



Figure 1‑6: Operation flow of the Dummy Peripheral about issuing output signals

***Explanation***:

* Dummy Peripheral model provides clock signals to INTC2G model via “PCLK” output port.
* For reset operation, Dummy Peripheral model issues reset signal to INTC2G model via “resetPort” output port.
* When users write value to register via “ts” socket, this value is written to corresponding output ports. (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.

## Dummy Slave model specification

### 1.5.1. Summary

* Dummy Slave model is used to confirm whether the transaction is transferred through INTC2G model successfully. It is implemented as DummySlaveRvc class.

**SC-HEAP Environment**



Figure 1‑7: Block diagram of Dummy Slave 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 receiving output signals from INTC2G model.
* Dummy Slave model is a target model which receives the transaction “tsp” from INTC2G model. After receiving the transaction, detail transaction information such as the address, command, and size are stored for checking the result.
* Dummy Slave model has an APB target socket "ts". Users can access read/write theDummy Slave's registers through this target socket.

### 1.5.2. Register

The registers of Dummy Slave model are described in the Table 1.3

Table 1.3: List of Dummy Slave’s registers

| **Register** | **Address offset** | **Initial value** | **Bit** | **Access** | **Description** |
| --- | --- | --- | --- | --- | --- |
| EXT\_REG | 0x00 | 0x0 | 0 - 31 | R | 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 | 0x04 | 0x0 | 0 - 31 | R | Store the transaction address |
| SIZE\_REG | 0x08 | 0x0 | 0 - 7 | R | Store transaction size |
| CMD\_REG | 0x0C | 0x0 | 0 | R | Store the transaction command  - 0x0: Read transaction  - 0x1: Write transaction |
| DATA\_REG | 0x10 | 0x0 | 0 - 31 | R | Store data of read transaction |

### 1.5.3. Operation



Figure 1‑8: Operation flow of the Dummy Slave model

***Explanation:***

* The Dummy Slave model receives the transaction from INTC2G model. After that, the transaction information will be stored into registers listed in Table 1.3.

Users can get value of those registers to check transaction result.

# **Environment Structu**re

**Output**

**scripts\_windows**

setup\_\*.bat

compile/compile\_64bit.bat

compile.bat

run/run\_64bit.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 timer C 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 G4 environment core

Store SC-HEAP compilation

Store files for compiling the SC-HEAP environment

Store the models

The user modeling environment

**Legend:**

**Folder**

**File**

Figure 2‑1: Verification environment structure

## How to verify

* 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 Linux is explained as in chapter 2.3. The scripts “run\_all\_osci.bat/run\_all\_osci\_64bit.bat”, “run\_all\_usk.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 2.1

Prepare

test pattern

Check and report

Run simulation

Prepare Environment

Setup environment

Compile environment

Compile test pattern

Create run batch

Run simulation

Check result

Make report

setup\_osci.csh/ setup\_osci\_64bit.csh

setup\_usk.csh/

setup\_usk\_64bit.csh

gen\_report\_osci.pl/

gen\_report\_usk.pl

check\_results.pl

gen\_sim\_osci.pl/

gen\_sim\_usk.pl

gen\_mot.pl

run\_all\_osci.csh/

run\_all\_usk.csh

**Legend**

Script name

Task name

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\_osci\_64bit.csh setup\_usk.csh/ setup\_usk\_64bit.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***:

* *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.*
* *\*\_64bit.csh scripts are used for 64bit env*

## 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.

Check results

Run simulation

Prepare Environment

Setup environment

Compile environment

Run simulation

Check result

compile.bat/

compile\_64bit.bat

setup\_osci.bat/ setup\_osci\_64bit.bat

setup\_usk.bat/ setup\_usk\_64bit.bat

run.bat/

run\_64bit.bat

check\_results.bat

run\_all\_osci.bat/run\_all\_osci\_64bit.bat

run\_all\_usk.bat/run\_all\_usk\_64bit.bat

**Legend**

Scripts name

Task name

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\_osci\_64bit.bat*  *setup\_usk.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 SC-HEAP environment is the “Release” mode. |
| Compile the environment  (*compile.bat/compile\_64bit.bat*) | Compile the Visual C++ solution which includes the SC-HEAP G4 VC++ project. |
| Run simulation for all test patterns  (*run.bat/run\_64bit.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***:

* *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.*
* *\*\_64bit.bat scripts are used for 64bit env.*

# 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 |
| Embedded software tool | GHS MULTI 6.1.4  rteserv2 |
| Library | System C, TLM | OSCI SystemC v2.3.1a |
| vlab 2.3.6 |
| Environment | SC-HEAP | SC-HEAP G4 Rev73 |
| Execution mode | LT and AT modes |

***Note***: *(\*) - Coding rule is ignored because 1Team tool cannot execute with SystemC 2.3 library*

*(\*\*) - Memory leakage is ignored because Valgrind tool cannot execute in SCHEAP environment.*

# 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-17002-01\_INTC2G.xls. |

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Modified points** | **Approver** | **Checker** | **Author** |
| 1.0 | - Create new | RVC/DungNguyen  03/01/2017 | FPT/Ha  02/28/2017 | FPT/Truong  02/23/2017 |
| 1.1 | Update to change Debug Master PEID:  - “Reference Manuals”: add DMS path  - Update Table 1.1 (add WR\_DATA\_REG\_0, WR\_DATA\_REG\_1, RD\_DATA\_REG\_0, RD\_DATA\_REG\_1, BIG\_PEID\_REG)  - Update Table 1.3 (add BIG\_PEID\_REG)  - Update Table 3.1 (correct OSCAR version) | RVC/Binh  04/21/2017 | RVC/Hue  04/21/2017 | RVC/Binh  04/20/2017 |
| 1.2 | - Update SCHEAP Env to 106 | RVC/Binh  09/01/2017 | RVC/ChuongLe  09/01/2017 | FPT/Ha  08/17/2017 |
| 1.3 | - Update Table 3.1 (add 64bit env condition) | RVC/Binh  11/22/2017 | RVC/Chan  11/22/2017 | RVC/Binh  11/21/2017 |
| 1.4 | - Update Figure 2‑1, Figure 2‑2, Figure 2‑3, Table 2.1, Table 2.2, [Chapter 2.1](#_How_to_verify) (add scripts for 64bit env) | RVC/Binh  12/08/2017 | RVC/Chan  12/08/2017 | RVC/Binh  12/07/2017 |
| 1.5 | - Update Table 1.1 (add “0x3” setting for description of CTRL\_REG) | RVC/Binh  12/20/2017 | RVC/Chan  12/20/2017 | RVC/Binh  12/19/2017 |
| 1.6 | - Update Table 3.1 (add 64bit env Rev73)  - Update CTRL\_REG in Table 1.1: List of Dummy Master’s registers | Chan Le  07/05/2018 | ThanhLe  07/04/2018 | FPT/Huong  06/18/2018 |