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

|  |
| --- |
| **Development of INTC2G model for E2x**  (v1.3) |

|  |
| --- |
| **Summary:** |
| This document describes the Detail Design Specification of INTC2 Guard (INTC2G) for E2x. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 Modeling guideline (Rev. 4.00) | IDF-14-010278-01 | This document describes the Guideline for peripheral macro  development which is connected to SC-HEAP\_E3 simulator  (***File***: SC-HEAP\_E3\_Modeling\_Guideline.pdf) |
| 3 | SC-HEAP\_E3 BUS I/F outline | LLWEB-00010925  ZSG-F31-12-0029-01 | The document describes the outline of bus I/F applied to SC-HEAP\_E3  (***File***: scheap\_e3\_bus\_if\_outline\_E.pdf) |
| 4 | SC-HEAP\_E3 PYTHON I/F function specification (v2.0) | LLWEB-00105192  MSS-SG-12-0062-02 | The document describes how to use python interface  (***File***: SC-HEAP\_E3 Python IF\_t.pdf) |
| 5 | RH850/E2x-FCC1  Hardware User’s Manual | RH850/E2x-FCC1  (Rev.0.50) | Hardware User’s Manual of Renesas microcontroller RH850 Family  (***File:***r01uh0641ej0050-rh850e2x-fcc1(E2ML).pdf) | - |
| 6 | RH850/E2x-FCC1 Hardware User’s Manual | RH850/E2x-FCC1  (Rev.0.80) | Hardware User’s Manual of Renesas microcontroller RH850 Family  (***File:***r01uh0641ej0080-rh850e2x-fcc1(E2ML).pdf) | - |
| 7 | REQ-MCS-17002\_INTCG(\*) | REQ-MCS-17002\_INTCG | Detail requirement of INTC2G model  (***File***: REQ-MCS-17002\_INTC2G.xlsx) | **DMS:** Documents/010\_ENG/140\_FrontEnd/Project/01\_SLD/2\_SLD\_Project/Model\_Documents/02\_MCS\_Project/REQ |
| 8 | INTC2\_GUARD\_20161025 | - | (***File***: INTC2\_Guard\_20161025.xlsx) |  |
| 9 | RH850/E2x-FCC2 Target spec | - | E38\_Functional Safety\_fcc2\_mod04.docx |  |

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

Table of Contents

[1. Model summary 6](#_Toc518484585)

[2. Supported features 6](#_Toc518484586)

[3. Block diagram 7](#_Toc518484587)

[4. List of implemented registers 8](#_Toc518484588)

[5. Port behavior 14](#_Toc518484589)

[5.1. List of implemented ports 14](#_Toc518484590)

[5.2. Reset signals 15](#_Toc518484591)

[6. Direction for users 16](#_Toc518484592)

[6.1. File structures 16](#_Toc518484593)

[6.2. Input/Output file 17](#_Toc518484594)

[6.3. How to connect Verification Environment 17](#_Toc518484595)

[6.4. Commands and parameters 19](#_Toc518484596)

[6.5. Message style 21](#_Toc518484597)

[6.6. Defined macro and template 26](#_Toc518484598)

[7. Flow diagram 27](#_Toc518484599)

[7.1. Sequence flow 27](#_Toc518484600)

[7.2. State diagram 28](#_Toc518484601)

[7.3. Validating transactions 30](#_Toc518484602)

[7.4. Key Code Protected 31](#_Toc518484603)

[7.5. Reporting error transactions 32](#_Toc518484604)

[7.6. Sending error interrupt request 33](#_Toc518484605)

[7.7. Decoding from transaction address into the protected address range 33](#_Toc518484606)

[7.8. Python IF and Heap configuration operation flow 35](#_Toc518484607)

[7.9. Reset flow 38](#_Toc518484608)

[8. Functions description 39](#_Toc518484609)

[8.1. INTC2G class 39](#_Toc518484610)

[8.2. INTC2G\_Func Class 42](#_Toc518484611)

[8.3. INTC2G\_CommonGuardAgentControllerIf Class 43](#_Toc518484612)

[8.4. INTC2G\_CommonGuard Class 43](#_Toc518484613)

[9. Limitation 45](#_Toc518484614)

**Index of Figures**

[Figure 3‑1: General block diagram for INTC2G model 7](#_Toc518484615)

[Figure 6‑1: File structure 16](#_Toc518484616)

[Figure 6‑2: Example of instantiating INTC2G objects and connecting them to other models 18](#_Toc518484617)

[Figure 7‑1: Sequence diagram 27](#_Toc518484618)

[Figure 7‑2: State diagram 28](#_Toc518484619)

[Figure 7‑3: Validating transactionsactivity diagram 30](#_Toc518484620)

[Figure 7‑4: Key Code Protected flow diagram 31](#_Toc518484621)

[Figure 7‑5: Reporting error transactions activity diagram 32](#_Toc518484622)

[Figure 7‑6: Sending error interrupt request acitivity diagram 33](#_Toc518484623)

[Figure 7‑7: Python IF and Heap configuration operation flow Part 1 35](#_Toc518484624)

[Figure 7‑8: Python IF and Heap configuration operation flow Part 2 36](#_Toc518484625)

[Figure 7‑9: Python IF and Heap configuration operation flow Part 3 36](#_Toc518484626)

[Figure 7‑10: Python IF and Heap configuration operation flow Part 4 37](#_Toc518484627)

[Figure 7‑11: Python IF and Heap configuration operation flow Part 5 37](#_Toc518484628)

[Figure 7‑12: Reset flow 38](#_Toc518484629)

[Figure 7‑13: Relationship between software reset and hardware reset 39](#_Toc518484630)

**Index of Tables**

[Table 2‑1: Feature of INTC2G model 6](#_Toc518484631)

[Table 4‑1: List of INTC2G's registers 8](#_Toc518484632)

[Table 5‑1: List of INTC2G’s ports 14](#_Toc518484633)

[Table 6‑1: File description 16](#_Toc518484634)

[Table 6‑2: List of parameters 19](#_Toc518484635)

[Table 6‑3: List of commands 20](#_Toc518484636)

[Table 6‑4: An example of python interface usage 21](#_Toc518484637)

[Table 6‑5: Dumping Register RW message description 21](#_Toc518484638)

[Table 6‑6: Dumping parameter help message description 21](#_Toc518484639)

[Table 6‑7: Dumping command help message description 22](#_Toc518484640)

[Table 6‑8: Error and debugging message description 22](#_Toc518484641)

[Table 6‑9: Error and debug messages 23](#_Toc518484642)

[Table 8‑1: Public Member Functions of INTC2G class 39](#_Toc518484643)

[Table 8‑2: Public Data Fields of INTC2G class 41](#_Toc518484644)

[Table 8‑3: Private Member Functions of INTC2G class 41](#_Toc518484645)

[Table 8‑4: Public Member Functions of INTC2G\_Func class 42](#_Toc518484646)

[Table 8‑5: Private Attributes of INTC2G\_Func class 43](#_Toc518484647)

[Table 8‑6: Public Member Functions of INTC2G\_CommonGuardAgentControllerIf class 43](#_Toc518484648)

[Table 8‑7: Public Member Functions of INTC2G\_CommonGuard class 43](#_Toc518484649)

[Table 8‑8: Private Member Functions of INTC2G\_CommonGuard class 44](#_Toc518484650)

[Table 8‑9: Private Attributes of INTC2G\_CommonGuard class 44](#_Toc518484651)

# Model summary

* INTC2 Guard (INTC2G), one of the slave guards, controls the access to these INTC2 module. INTC Guard can protect respectively the read and write access against INTC2 registers of each error source. If INTC2G detects any illegal access, a guard error notification is signaled to ECM.
* INTC2G model has one target socket (“ts” socket) of APB bus to read/write its own registers.
* INTC2G model also incorporates one target socket (“tsp” socket) and one initiator socket (“isp”); it receives transactions send to INTC2 via tsp socket, then checks these transactions before sending them to INTC2 via isp socket.
* Both loosely time mode (LT) and approximately time (AT) mode are supported.
* This model supports little endian mode as the endian of APB bus interface.

# Supported features

Table 2‑1: Feature of INTC2G model

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Description** | | **HWM reference** |
| **Hardware** | **Model** |
| Max frequency of clock | - | Unlimited frequency. There is no setting condition | *-* |
| Read/Write registers | - | Use TLM target socket to receive the data from APB bus | *-* |
| Main function | INTC2 Guard is slave guards can protect respectively the read and write access against INTC2 registers of each error source. Notify guard error to ECM if INTC2 Guard detects illegal access | ← | *38.4.5.1 Overview* |
| The protection is configurable by setting- and control-registers. These registers, in turn, are also protected by key-code | ← | *-* |

***Notes****:*

*- The symbol “←” means that these features are supported as description in the hardware manual.*

# Block diagram

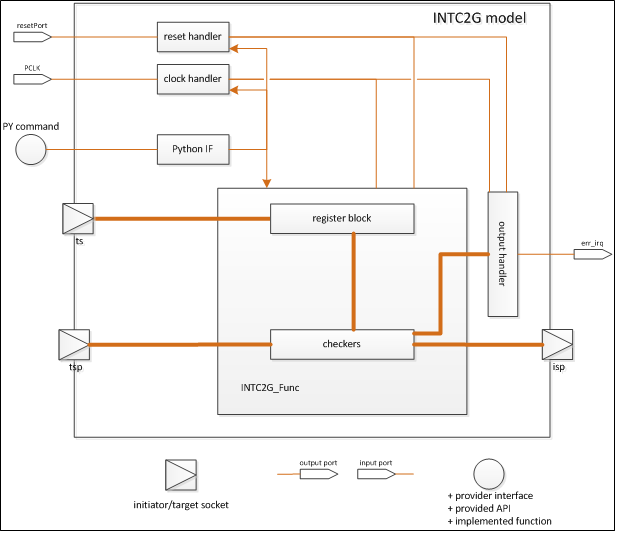


Figure 3‑1: General block diagram for INTC2G model

***Explanation***:

* INTC2G model has an APB target socket “ts” used for read/write access into the INTC2G's registers.
* INTC2G model also incorporates one target socket (“tsp” socket) and one initiator socket (“isp”); it receives transactions sent to INTC2 model via tsp socket, then checks these transactions before sending them to the protected peripherals (INTC2) via isp socket.
* INTC2G model includes the following blocks:
* The “Clock handler” block handles input clocks.
* The “Reset handler” block handles input reset signals.
* The “Output handler” block controls output ports & interrupts.
* Users can use commands and parameters via "PythonIF" block for debug and control model purpose.
* The “INTC2G\_Func” block stores registers of INTC2G& controls INTC2G's operation.

# List of implemented registers

Table 4‑1: List of INTC2G's registers

.

| **Register name** | **Model** | **Address Offset** | **R/W** | **Initial value** | **Register size (bit)** | **Write Access size (bit)** | **Read Access size (bit)** | **Bit name** | **Bit position** | **Explanation** | **Support** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **INTC2 Guard Error Clear Register** | INTC2GERRCLR | INTC2\_GUARD\_base + H'04 | R|W | 0000 0000H | 32 | 32 | 8|16|32 | CLRO | 1 | Clear the INTC2GERRSTAT.OVF bit | Yes |
| CLRE | 0 | Clear the INTC2GERRSTAT.ERR bit | Yes |
| **INTC2 Guard Error Status Register** | INTC2GERRSTAT | INTC2\_GUARD\_base + H'08 | R | 0000 0000H | 8|32 | 8|16|32 | 8|16|32 | OVF | 1 | Overflow status flag | Yes |
| ERR | 0 | Error status flag | Yes |
| **INTC2 Guard Error Address Register** | INTC2GERRADDR | INTC2\_GUARD\_base + H'0C | R | 0000 0000H | 32 | 8|16|32 | 8|16|32 | ADDR | [31:0] | Access address to the target slave when a guard error occurred | Yes |
| **INTC2 Guard Error Access Information Register** | INTC2GERRTYPE | INTC2\_GUARD\_base + H'10 | R | 0000 0000H | 16|32 | 8|16|32 | 8|16|32 | SEC | 13 | Access attribute of SEC to the target slave when a guard error occurred | Yes |
| DBG | 12 | Access attribute of DBG to the target slave when a guard error occurred | Yes |
| UM | 11 | Access attribute of UM to the target slave when a guard error occurred | Yes |
| SPID | [10:6] | Access attribute of SPID to the target slave when a guard error occurred | Yes |
| WRITE | 0 | Access type of read or write to the target slave when a guard error occurred | Yes |
| **INTC2 Guard Key Code Protection Register** | INTC2GKCPROT | INTC2\_GUARD\_base + H'0 | R|W | 0000 0000H | 32 | 32 | 8|16|32 | KCPROT | [31:1] | Enable or disable modification of the KCE bit. The value written is not retained. These bits are always read as 0. | Yes |
| KCE | 0 | Key Code Enable bit 0: Disables write access of protected registers 1: Enables write access of protected registers | Yes |
| **INTC2 Guard SPID Setting Register** | INTC2GMPIDn(n=0-7) | INTC2\_GUARD\_base + H'40 + H'4 \* n | R|W | 0000 0000H | 32 | 8|32 | 8|16|32 | SPID | [4:0] | R/W enable setting by SPID | Yes |
| **INTC2 protection control register** | INTC2GPROT\_GR | INTC2\_GUARD\_base + H'F0 | R|W | 0000 0000H | 32 | 8|16|32 | 8|16|32 | MPID | [23:16] | R/W enable setting by MPID MPID is a list of bits each representing one MPID value. Multiple MPID values are enabled simultaneously by setting multiple bits. For example, setting MPID to 0101B enables access with MPID = 0 and MPID = 2. 0: Read / Write with MPID = m depends on the RG and WG bit setting 1: Enables Read / Write with MPID = m | Yes |
| GEN | 8 | Enables / disables guard setting 0: Disables the guard setting 1: Enables the guard setting | Yes |
| DBG | 6 | R/W enable setting for debug master 0: Depends on other enable / disable settings 1: Enables R/W | Yes |
| UM | 4 | R/W disable setting in user mode 0: R/W disabled 1: R/W depends on other enable / disable settings | Yes |
| WG | 1 | Write Global Enable 0: During write, GMPIDn is used as a judgment condition. 1: During write, GMPIDn is not used as a judgment condition. | Yes |
| RG | 0 | Read Global Enable 0: During read, GMPIDn is used as the judgment condition. 1: During read, GMPIDn is not used as a judgment condition. | Yes |
| **INTC2 Guard Control Register** | INTC2GPROTn(n=32 to 511) | INTC2\_GUARD\_base + H'100 + H'4 \* n | R|W | 0000 0000H | 32 | 8|16|32 | 8|16|32 | MPID | [23:16] | R/W enable setting by MPID MPID is a list of bits each representing one MPID value. Multiple MPID values are enabled simultaneously by setting multiple bits. For example, setting MPID to 0101B enables access with MPID = 0 and MPID = 2. 0: Read / Write with MPID = m depends on the RG and WG bit setting 1: Enables Read / Write with MPID = m | Yes |
| GEN | 8 | Enables / disables guard setting 0: Disables the guard setting 1: Enables the guard setting | Yes |
| DBG | 6 | R/W enable setting for debug master 0: Depends on other enable / disable settings 1: Enables R/W | Yes |
| UM | 4 | R/W disable setting in user mode 0: R/W disabled 1: R/W depends on other enable / disable settings | Yes |
| WG | 1 | Write Global Enable 0: During write, GMPIDn is used as a judgment condition. 1: During write, GMPIDn is not used as a judgment condition. | Yes |
| RG | 0 | Read Global Enable 0: During read, GMPIDn is used as the judgment condition. 1: During read, GMPIDn is not used as a judgment condition. | Yes |

***Notes***:

*There are 481 channels incorporatedinsideINTC2G:*

* *INTC2GPROT\_n (n from 32 to 511): 480 channels*
* *INTC2GPROT\_GR: 1 channel*

# Port behavior

## List of implemented ports

Table 5‑1: List of INTC2G’s ports

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Port name** | **I/O** | **Type** | **Initial** | **Active** | **Synchronous clock** | **Description** | **Support** |
| ***AMBA APB I/F (access register)*** | | |  |  |  |  |  |
| PCLK | In | sc\_in<sc\_dt::uint64> | - | - | - | APB bus clock | Yes |
| resetPort | In | sc\_in<bool> | - | false | - | Asynchronous reset signal | Yes |
| ts | In | tlm::tlm\_target\_socket | - | - | PCLK | Target socket of APB bus interface.  This socket is dedicated to operation configuration of the INTC2G | Yes |
| ***Sockets for exchanging transaction*** | | | | | | | |
| tsp | In | tlm::tlm\_target\_socket | - | - | PCLK | Target socket of APB bus interface.  This socket is dedicated to receive transactions. These transactions will be checked by settings of INTC2G configuration registers | Yes |
| isp | Out | tlm::tlm\_initiator\_socket | - | - | PCLK | Initiator socket of APB bus interface.  This socket is dedicated to send out transaction to the protected peripherals. | Yes |
| ***Internal I/F*** | | | | | | | |
| err\_irq | Out | sc\_out<bool> | false | high | PCLK | Notify error to ECM module | Yes |

.

## Reset signals

* “resetPort” is active-low asynchronous reset.
* When “resetPort” is active, all registers are initialized, all attributes are initialized, Operation of INTC2G is stopped.

# Direction for users

## File structures

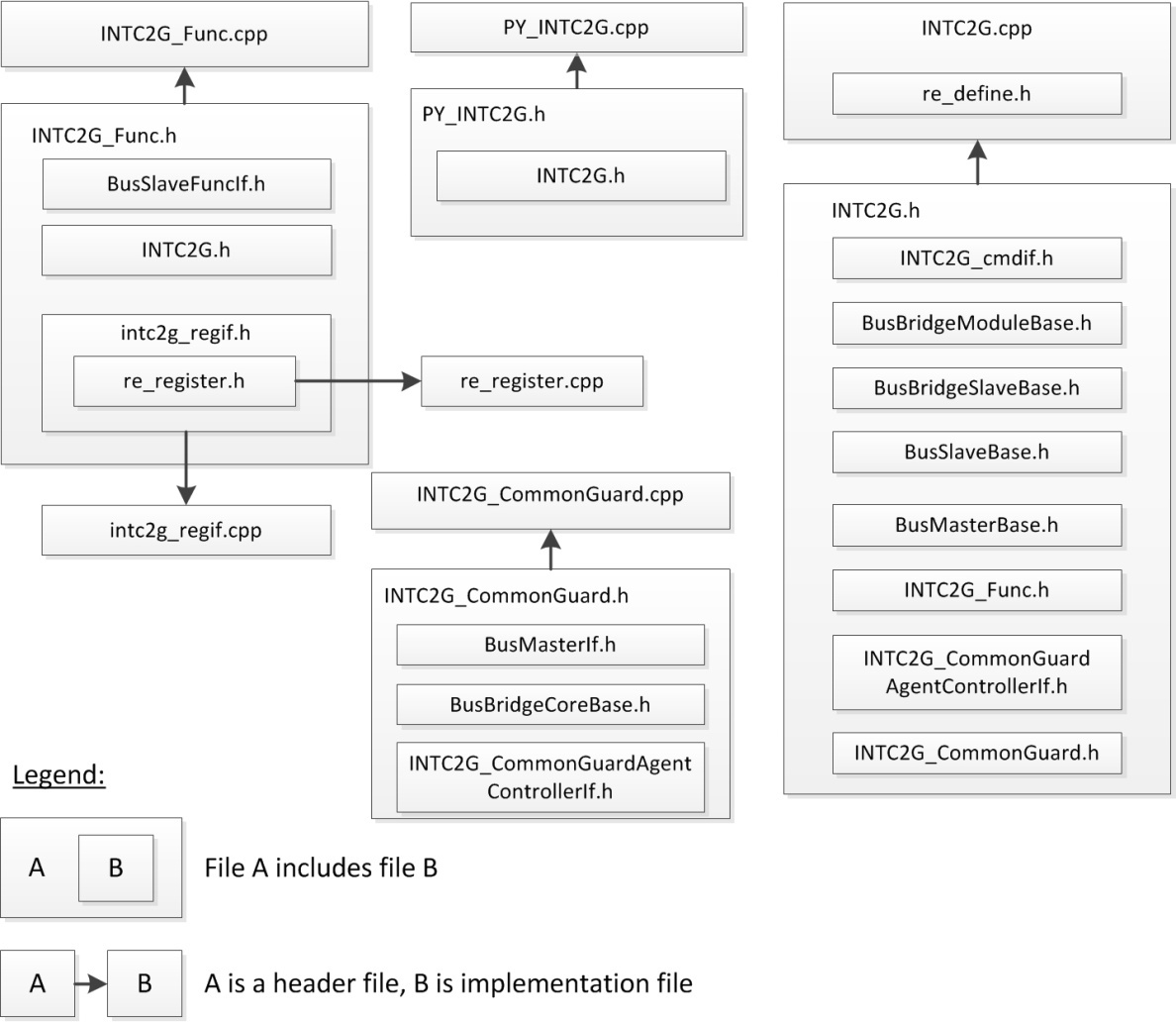


Figure 6‑1: File structure

.

Table 6‑1: File description

| **File name** | **CVS tag** | **Developed/Reused** | **Description** |
| --- | --- | --- | --- |
| INTC2G.h | - | Developed | Header file of INTC2G class. |
| INTC2G.cpp | - | Developed | Implementation of INTC2G class. |
| INTC2G\_Func.h | - | Developed | Header file of INTC2G function. |
| INTC2G\_Func.cpp | - | Developed | Implementation files of INTC2G function. |
| PY\_INTC2G.h | - | Developed | Header file of INTC2G python interface. |
| PY\_INTC2G.cpp | - | Developed | Implementation files of INTC2G python interface. |
| INTC2G\_CommonGuard.h | - | Developed | Header file of INTC2G\_CommonGuard class. |
| INTC2G\_CommonGuard.cpp | - | Developed | Implementation of INTC2G\_CommonGuard class. |
| INTC2G\_CommonGuardAgentControllerIf.h | - | Developed | Header file of the interface INTC2G\_CommonGuardAgentControllerIf |
| intc2g\_regif.h | - | Generated (\*) | Header file of INTC2G register interface. |
| intc2g\_regif.cpp | - | Generated (\*) | Implementation file of INTC2G register interface. |
| INTC2G\_cmdif.h | - | Generated (\*) | Implementation file of message level and re\_printf function. |
| re\_register.h | v2013\_12\_17  (Modified to  add function UpdateBitInitValue()) | Reused | Header file of the re\_register class. |
| re\_register.cpp | Reused | Implement the attributes and the operations of common register class. |
| BusBridgeModuleBase.h | - | Reused | Header file of BusBridgeModuleBase class. |
| BusBridgeSlaveBase.h | - | Reused | Header file of BusBridgeSlaveBase class. |
| BusSlaveBase.h | - | Reused | Header file of BusSlaveBase class. |
| BusMasterBase.h | - | Reused | Header file of BusMasterBase class. |
| BusSlaveFuncIf.h | - | Reused | Header file of BusSlaveFuncIf class. |
| re\_define.h | - | Reused | Define common define macro, enum and so on |

***Note***:

*- (\*) Files intc2g\_regif.h/.cppare generated from Register IF Generator (v2014\_10\_07). FilesINTC2G\_cmdif.hare generated from Command IF Generator (v2014\_03\_13).*

## Input/Output file

There is no input or output file.

## How to connect Verification Environment

There are 4 basic steps to connect INTC2G model to a verification environment.

* Step 1: Declare instances of INTC2G class
* Step 2: Bind the target sockets, namely “ts” and “tsp”, and then bind the initiator socket “isp”.
* Step 3: Bind the input/output signals
* Step 4: Set protected address ranges via Python interface.



Figure 6‑2: Example of instantiating INTC2G objects and connecting them to other models

***Explanation***:

* Error notification signals “err\_irq” are connected into ECM module.
* In INTC2G model, it has 481 channels:
  + channels which are controlled by registers INTC2GPROT\_n (n from 32 to 511): 480 channels
  + a channel which is controlled by registerINTC2GPROT\_GR: 1 channel
* The code snippet bellow illustrates how to instantiate INTC2G class with correct setting parameter:

1. ...
2. INTC2G\* intc2g;
3. ...
4. intc2g = new INTC2G(“INTC2G”,0, 0, G4SS->mIssMode);

* Before the INTC2Gmodel can be executed in a normal manner, the protected address ranges should be set via Python Interface. The code snippet bellow illustrates how to set protected address ranges for INTC2Gobjects protecting registers of INTC2 module. Please note that:
  + The channelIDs of the corresponding register INTC2GPROT\_n (n from 32-511) are fixed from 32 to 511 with same order.
  + The channelID of the corresponding register INTC2GPROT\_GR is fixed 512.
  + All address range must not be overlapped, model will be reject settings with overlapping address.
* INTC2G model support a channel can protect multi address ranges.

1. ...
2. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 32, 0x8fff0000, 0x4) //corresponding INTC2GPROT\_n. “id” = 32
3. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 33, 0x8fff0100, 0x40)
4. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 33, 0x8fff0200, 0x50)
5. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 34, 0x8fff0300, 0x60)
6. ...
7. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 510, 0x8fffA100, 0x60)
8. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 511, 0x8fffA200, 0x60)
9. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 511, 0x8fffA300, 0x60)//corresponding INTC2GPROT\_n. “id” = 511
10. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 512, 0x8fffA400, 0x60) //corresponding INTC2GPROT\_GR “id” = 512
11. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 512, 0x8fffA500, 0x60)
12. SCHEAP.INTC2G\_SetAreaAddr(“RH850.INTC2G”, 512, 0x8fffA600, 0x60)
13. ...

For details of usage, see the next sectionCommand help messages.

## Commands and parameters

Table 6‑2: List of parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **Type** | **Default** | **Configure method** | **Description** |
| INTC2G  \_MessageLevel | string | fatal|error | Python IF | Select debug message level (“fatal”, “error”, “warning” and “info”).  One or more than levels can be connected by vertical bar (Example “fatal|error”)  Note: The setting of this parameter is not affected when REGIF\_SC\_REPORT is defined |
| INTC2G  \_DumpRegisterRW | string | false | Python IF | Enable/disable dumping access register.  + “false”: Not dump register access information  + “true”: Dump register access information  Note: The setting of this parameters affects messages belong to dumping register information. |
| INTC2G  \_DumpInterrupt | string | false | Python IF | Dump interrupt information when interrupt is assert. This message is info level.  + “false” : Not dump interrupt information  + “true” : Dump interrupt information |

Table 6‑3: List of commands

| **Command** | **Type** | **Argument** | **Configure method** | **Description** |
| --- | --- | --- | --- | --- |
| INTC2G  \_AssertReset | void | rst\_name, start-time, period | Python IF | Assert and deassert reset signal  + std::string <rst\_name>: name of reset signal (“resetPort”)  + double <start-time>: the time until asserting reset signal from current time. The unit is “ns”  + double <period>: the time from asserting reset signal to de-assert it. The unit is “ns” |
| INTC2G  \_SetCLKFreq | void | clk\_name,  freq,  unit | Python IF | Set clock frequency value  + std::string <clk\_name>: name of signal (“PCLK”)  + sc\_dt::uint64 <freq>: clock frequency  + std::string <unit>: frequency unit (“Hz”, “KHz”, “MHz” or “GHz”) |
| INTC2G  \_GetCLKFreq | void | clk\_name | Python IF | Get clock frequency value  + std::string <clk\_name>: name of signal (“PCLK”) |
| INTC2G  \_ForceRegister | void | reg\_name,  value | Python IF | ForceINTC2G’s common registers with setting value  + std::string <reg\_name>: name of register  + unsigned int <value>: value which is set to register |
| INTC2G  \_ReleaseRegister | void | reg\_name | Python IF | Release INTC2G’s common registers from force value  + std::string <reg\_name>: name of register |
| INTC2G  \_WriteRegister | void | reg\_name,  value | Python IF | Write a value to INTC2G’s common registers  + std::string <reg\_name>: name of register  + unsigned int <value>: value which is set to register |
| INTC2G  \_ReadRegister | void | reg\_name | Python IF | Read value from INTC2G’s common registers  + std::string <reg\_name>: name of register |
| INTC2G  \_ListRegister | void | - | Python IF | Dump register names of INTC2G model |
| INTC2G  \_Help | void | type | Python IF | Dump the direction how to use python interface parameters and commands  + std::string <type>: “parameters” or “commands” |
| INTC2G\_SetAreaAddr | void | Channel\_ID,  start\_address,  address\_size | Python IF | Set range of address to be protected by INTC2G  + unsigned int<Channel\_ID>: id of the channel in model (channel\_ID of channels which are controlled by register INTC2GPROT\_n (n from 32 to 511) is from 32 to 511. Channel\_ID of channel which is controlled by register INTC2GPROT\_GR is 512)  + std::hex<start\_address>: Start of protected area  + std::hex<address\_size>: Size of area will be protected |
| INTC2G\_DumpStatusInfo | void | - | Python IF | Dump the status information of IBG model. |

***How to use***: Below example describes how to use commands/parameters of python interface.

Table 6‑4: An example of python interface usage

|  |
| --- |
| SCHEAP.setFreq(100,”MHz”)  SCHEAP.INTC2G\_MessageLevel(“RH850.INTC2G”,”info|error|warning|fatal”)  SCHEAP.INTC2G\_SetCLKfreq(“RH850.INTC2G”,”PCLK”,2, “MHz”)  SCHEAP.INTC2G\_GetCLKfreq(“RH850.INTC2G”,”PCLK”)  SCHEAP.INTC2G\_WriteRegister(“RH850.INTC2G”,”INTC2GKCPROT32”,0xFFFF)  SCHEAP.INTC2G\_WriteRegister(“RH850.INTC2G”,”INTC2GKCPROT511”,0xFFFF)  SCHEAP.INTC2G\_WriteRegister(“RH850.INTC2G”,”INTC2GKCPROT\_GR”,0xFFFF)  SCHEAP.INTC2G\_WriteRegister(“RH850.INTC2G”,”INTC2GMPID0”,0xFF)  SCHEAP.INTC2G\_WriteRegister(“RH850.INTC2G”,”INTC2GMPID7”,0xAA)  SCHEAP.INTC2G\_ReadRegister(“RH850.INTC2G”,”INTC2GKCPROT32”)  SCHEAP.sc\_start(1000) |

See details of registers in section List of implemented registers.

## Message style

### Register RW messages

Table 6‑5: Dumping Register RW message description

|  |  |
| --- | --- |
| **Condition** | This message is output when INTC2G registers are accessed and parameter INTC2G\_DumpRegisterRW is set “true”. |
| **Output** | This message's kind is printed to standard output (console). |
| **Format**: Info: <hier\_instance\_name>: [<time>ps] REG [<reg\_name>] R Size = <size> Addr = <reg\_address> Data = <reg\_value>  Info: <hier\_instance\_name>: [<time>ps] REG [<reg\_name>] W Size = <size> Addr = <reg\_address> Data = <reg\_value> : <old\_value> =><new\_value>  **Example**:  Info: INTC2G: [ 5030000 ps] REG [INTC2GERRSTAT ] R Size= 1 Addr= 0xFFC7000B Data= 0x00000000  Info: INTC2G: [ 36420000 ps] REG [INTC2GERRSTAT ] W Size= 4 Addr= 0xFFC70008 Data= 0x00000000 : 0x00000000 => 0x00000000 | |
| **Tag name** | **Description** |
| hier\_instance\_name | Hierarchy instance name of INTC2G model is being used. |
| time | Simulation time |
| reg\_name | Name of accessed register. |
| size | Register size. |
| address | Register address. |
| value | Register value. |
| old\_value | Register's value before writing. |
| new\_value | Register's value after writing. |

### Help messages

#### Parameter help messages

Table 6‑6: Dumping parameter help message description

|  |  |
| --- | --- |
| **Condition** | This message is dumped out when command INTC2G\_Help is called with “parameters” argument. |
| **Output** | This kind of message is printed to standard output (console).  The help message is used for python interface. |
| --- parameters ---  INTC2G\_MessageLevel (fatal|error|warning|info) Set debug message level (default: fatal|error).  INTC2G\_DumpRegisterRW (true/false) Enable/disable dumping access register (default: false).  INTC2G\_DumpInterrupt (true/false) Enable/disable dumping interrupt information (Default: false). | |

#### Command help messages

Table 6‑7: Dumping command help message description

|  |  |
| --- | --- |
| **Condition** | This message is dumped out when command INTC2G\_Help is called with “commands” argument. |
| **Output** | This message's kind is printed to standard output (console).  The help message is used for python interface. |
| --- commands ---  INTC2G\_AssertReset (rst\_name, start\_time, period) Assert and de-assert reset signal (“resetPort”).  INTC2G\_SetCLKFreq (clk\_name, freq, unit) Set clock frequency to model.  INTC2G\_GetCLKFreq (clk\_name) Get clock frequency of model.  INTC2G\_ForceRegister (reg\_name, value) Force INTC2G’s common register with setting value.  INTC2G\_ReleaseRegister (reg\_name) Release INTC2G’s common register from force value.  INTC2G\_WriteRegister (reg\_name, value) Write a value to INTC2G’s common register.  INTC2G\_ReadRegister (reg\_name) Read value from INTC2G’s common register.  INTC2G\_ListRegister ()Dump name of INTC2G's registers.  INTC2G\_SetAreaAddr (channelId, start\_addr, size) Set protected address range | |

### Error and debugging messages

#### Error and debugging messages style

Table 6‑8: Error and debugging message description

|  |  |
| --- | --- |
| **Condition** | This message's kind is output when error occurs or some important events occur. It's depended on setting of parameter INTC2G\_MessageLevel.  Detailed conditions are described in the “Description” column ofTable 6‑9. |
| **Output** | This message's kind is printed to standard output (console). |
| **Format**: <severity>: <hier\_instance\_name>: [<time><unit>] <Message content>  **Example**: Warning: INTC2G: [ 0 ps] Cannot write during reset period. | |
| **Tag name** | **Description** |
| severity | Kind of message's severity. |
| hier\_instance\_name | Hierarchy instance name of INTC2G model is being used. |
| time | Simulation time. |
| unit | Simulation time's unit. |
| Message content | Message content (message list is described inTable 6‑9). |

#### List of error and debugging messages

Table 6‑9: Error and debug messages

| **No.** | **Type** | **Severity** | **Message** | **Description** |
| --- | --- | --- | --- | --- |
| 1 | user | error | Cannot find the object of <model name> class | Users call PythonIF with wrong object of <model name>class |
| 2 | user | error | <command name> has too much arguments | Dump this message when number of input arguments is incorrect. |
| 3 | user | error | <command name> command needs an argument [true/false] | Dump this message when input argument is missed. |
| 4 | user | warning | The <model name>\_ListRegister has not any arguments | Users call PythonIF of <model name>\_ListRegister with any argument. The argument should be not input. |
| 5 | user | warning | The arguments of <command name> are wrong | Users call PythonIF of <command name> with wrong arguments |
| 6 | user | warning | The name (%s) of <model name>\_Help argument is wrong (commands or parameters). | Users call <model name>\_Help command with invalid argument. It must be “commands” or “parameters” |
| 7 | user | warning | Register name <register\_name> is invalid | Dump this message when register name is invalid. |
| 8 | user | info | <model name>\_DumpInterrupt %s | This message is dumped when users call <model name>\_DumpInterrupt without argument to check value of it. |
| 9 | user | error | Invalid access size: <%d> bytes | This message is dumped when users access to register with invalid size. |
| 10 | user | error | Writing access size to %s at address 0x%08X is wrong: %d byte(s). | Users write the value to registers with invalid size. |
| 11 | user | error | Reading access size to %s at address 0x%08X is wrong: %d byte(s). | Users read the value from registers with invalid size. |
| 12 | user | error | Invalid access address 0x%08X | Users access the model with invalid address |
| 13 | user | error | Invalid access address <address> with access size <size> bytes | User access to model's register with wrong aligned address |
| 14 | user | warning | Cannot write 1 to reserved bit | Users write the value 1 to reserved bit of registers |
| 15 | user | warning | %s forbids to write 0x%x | Users write the value to bit of registers, but the bit is read-only |
| 16 | user | warning | %s forbids to read | Users read bit of registers, but the bit is write-only |
| 17 | user | warning | Should read all bit in a register | Users read the registers with read access size are smaller than register size. |
| 18 | user | warning | %s is blocked writing from Bus I/F. | Access write to register which it is locked by <model name>\_ForceRegister. |
| 19 | user | warning | Clock name (%s) is invalid. | Users call <model name>\_SetCLKFreq command with clock name is wrong. |
| 20 | user | warning | Frequency unit (%s) is wrong, frequency unit is set as unit Hz default | Users call <model name>\_SetCLKFreq with frequency unit is wrong. The frequency unit must be Hz, Khz, MHz, GHz. |
| 21 | user | info | %s frequency is zero. | The model operates with clock frequency is zero. |
| 22 | user | info | <clock name> frequency is %0.0f %s. | Users call <model name>\_GetCLKFreq with clock name is <clock name> |
| 23 | user | warning | The reset name (%s) is wrong. | Users call <model name>\_AssertReset command with wrong reset name. |
| 24 | user | warning | The software reset of <reset name> is called in the reset operation of the model. So it is ignored | <model name>\_AssertReset command of <reset name> is called during reset operation of <reset name> |
| 25 | user | warning | Cannot write during reset period | Dump this message when users write to register during reset period. |
| 26 | user | warning | Cannot launch call-back function during reset period | Users write the value to the registers during reset period |
| 27 | user | info | The model will be reset (<reset name>) for %f ns after %f ns. | Users call <model name>\_AssertReset command for <reset name> with start reset time and reset period. |
| 28 | user | info | The reset port <reset name> is asserted | Users activate <reset name> port. |
| 29 | user | info | The reset port <reset name> is de-asserted | Users deactivate <reset name> port. |
| 30 | user | info | The model is reset by AssertReset command of <reset name> | Users call <model name>\_AssertReset command for <reset name> and reset operation is accepted after specified time at first argument of <model name>\_AssertReset. |
| 31 | user | info | Reset period of <reset name> is over | Reset period of <reset name> which is set by <model name>\_AssertReset is over. |
| 32 | user | info | Initialize %s (%08x) | This message is dumped when initializing value of register during reset period. |
| 33 | user | info | INT [<Interrupt name>] Assert. | This message is dumped when <model name>\_DumpInterrupt is enabled and <Interrupt name> is asserted. |
| 34 | user | info | INT [<Interrupt name>] De-assert. | This message is dumped when <model name>\_DumpInterrupt is enabled and <Interrupt name> is de-asserted. |
| 35 | user | warning | Prohibited setting of <register name>.<bit name> bit. | This message is dumped when users set prohibited value to <bit name> bit of <register name> register. |
| 36 | user | warning | The given channel id is invalid | Dumped when users give a channel id that is not correct via Python Interface. The correct channel id must not exceed the number of channels as defined in the user-manual. |
| 37 | user | warning | The given address range is invalid | Dumped when users give an address range that is not correct via Python Interface. The correct address range must not be specified by a start address greater than or equal to 0; size of the range must be greater than 0, and it does not overlap other ranges defined before. |
| 38 | user | error | The reading transaction is fail because the reset is active | Dumped when the INTC2G receives a read transaction when it is in reset state. |
| 39 | user | error | The writing transaction is fail because the reset is active | Dumped when the INTC2G receives a write transaction when it is in reset state. |
| 40 | user | warning | The address range overlapped with the previous defined one | Dumped when users give an address range that overlaps a certain address range already inserted before |
| 41 | user | warning | The address of the received transaction does not belong to any protected address range | When the protected address ranges are not continuous, a transaction sent to the INTC2G but it does not belong to any protected address range. |
| 42 | user | info | The bus guard is disabled | Inform users that a received transaction is discard, the first reason is that the bus guard is in disabled state |
| 43 | user | info | The guard forwarded the debug transaction because of debug enable feature | Inform users that the received transaction is forwarded to the protected address, because it is an debug transaction |
| 44 | user | info | The guard discarded the user-mode transaction because the user-mode enable is off | Inform users the reason why the received transaction is discarded or forwarded |
| 45 | user | info | The guard forwarded the read transaction due to global-read feature | Inform users the reason why the received transaction is discarded or forwarded |
| 46 | user | info | The guard forwarded the write transaction due to global-write feature | Inform users the reason why the received transaction is discarded or forwarded |
| 47 | user | info | The guard forwarded the transaction due to the matched SPID | Inform users the reason why the received transaction is discarded or forwarded |
| 48 | user | info | The guard discarded the transaction due to the unmatched SPID | Inform users the reason why the received transaction is discarded or forwarded |
| 49 | user | warning | Can not write to register block during zero clock | Users write the value to the registers during zero clock |
| 50 | user | warning | abnormal access. the address of the received transaction does not belong to protected range, but access range of transaction is belong protected range | Inform user an abnormal case: When address of transaction (trans.get\_address) do not belong to protected address range but access range of transaction (trans.get\_address + trans.get\_data\_length) is belong to protected address range |
| 51 | debug | warning | Could not found the information of guard setting | To detect the problem when common guard gets settings from register block. |
| 52 | Debug | info | finish instantiating the common guard | To debug only, confirm that the constructor of the INTC2G\_CommonGuard class returns successfully |
| 53 | Debug | info | copy G3m extension side-band info | To debug only, confirm that G3m extension exists in the received transaction |
| 54 | Debug | info | copy Vci extension side-band info | To debug only, confirm that VCI extension exists in the received transaction |
| 55 | Debug | info | start processing a read transaction | To debug only, confirm that the guard has already received a read transaction successfully and the corresponding handler (read method) is invoked |
| 56 | Debug | info | forward the read transaction | To debug only, confirm that the read transaction is passed permission checking |
| 57 | Debug | info | forward a read transaction in LT mode (or debug transactions) | To debug only, confirm that the read transaction is passed permission checking, and then transfer in LT mode branch. |
| 58 | Debug | info | forward a read transaction in AT mode (or non-debug transactions) | To debug only, confirm that the read transaction is passed permission checking, and then transfer in AT mode branch. |
| 59 | Debug | info | discard the read transaction | To debug only, confirm that the read transaction is discarded because the permission check returns fail |
| 60 | Debug | info | finish processing a read transaction | To debug only, confirm that the method handlers received read transaction executed completely. |
| 61 | Debug | info | start processing a write transaction | To debug only, confirm that the guard has already received a write transaction successfully and the corresponding handler (write method) is invoked |
| 62 | Debug | info | forward the write transaction | To debug only, confirm that the write transaction is passed permission checking |
| 63 | Debug | info | forward a write transaction in LT mode (or debug transactions) | To debug only, confirm that the write transaction is passed permission checking, and then transfer in LT mode branch. |
| 64 | Debug | info | forward a write transaction in AT mode (or debug transactions) | To debug only, confirm that the write transaction is passed permission checking, and then transfer in AT mode branch. |
| 65 | Debug | info | discard the write transaction | To debug only, confirm that the write transaction is discarded because the permission check returns fail |
| 66 | Debug | info | finish processing a write transaction | To debug only, confirm that the method handlers received write transaction executed completely. |
| 67 | Debug | info | try to get G3m extension side-band info | To debug only, confirm that the checking process is reach to step get G3m extension |
| 68 | Debug | error | could not get G3m extension side-band info | To debug only, confirm that the extension G3m is unavailable in the received transaction |
| 69 | Debug | info | try to get address info | To debug only, confirm that the checking process is reach to step get address information |
| 70 | Debug | info | validate address of the received transaction | To debug only, confirm that the checking process is reach to step validate address information |
| 71 | Debug | info | from the accessed address range, get validating settings | To debug only, confirm that the checking process is reach to step get checking parameters from the register block |
| 72 | Debug | info | check if the INTC2G is enable or not | To debug only, confirm that the checking process is reach to step checking status enable/disable of the INTC2G guard |
| 73 | Debug | info | check if the received transaction is debug and debug-enable mode is disable or not | To debug only, confirm that the checking process is reach to step checking debug mode setting |
| 74 | Debug | info | check if the the received transaction is in user mode and user mode disable is on or not | To debug only, confirm that the checking process is reach to step checking user mode setting |
| 75 | Debug | info | check if the global-read is enabled for all read transactions or not | To debug only, confirm that the checking process is reach to step checking global read setting |
| 76 | Debug | info | check if the global-write is enabled for all write transactions or not | To debug only, confirm that the checking process is reach to step checking global write setting |
| 77 | Debug | info | check if the SPID field of the transaction is allowed or not | To debug only, confirm that the checking process is reach to step checking SPID setting |
| 78 | Debug | info | default result is false, the guard blocks all others | To debug only, confirm that the checking process is reach to the point after the last checking step |
| 79 | Debug | info | copy Axi extension side-band info | To debug only, confirm that AXI extension exists in the received transaction |
| 80 | Debug | info | copy Vpi extension side-band info | To debug only, confirm that VPI extension exists in the received transaction |
| 81 | Debug | info | copy Ahb extension side-band info | To debug only, confirm that AHB extension exists in the received transaction |
| 82 | Debug | info | copy Apb extension side-band info | To debug only, confirm that APB extension exists in the received transaction |
| 83 | user | info | PROFILE(%s): INTC2G: INTC2GERRSTAT bits: 0x%08X\n | dump status info of registers |

.

## Defined macro and template

* There is no template used in INTC2G model.
* The following are macros used in the model: REGIF\_SC\_REPORT, REGIF\_NOT\_USE\_SYSTEMC.
* If users define the macro REGIF\_SC\_REPORT, the SC\_REPORT function is used. Otherwise, the “printf” function is used. This macro should be not defined if users defined REGIF\_NOT\_USE\_SYSTEMC.
* Users can define macro REGIF\_NOT\_USE\_SYSTEMC to remove SystemC part.

# Flow diagram

## Sequence flow

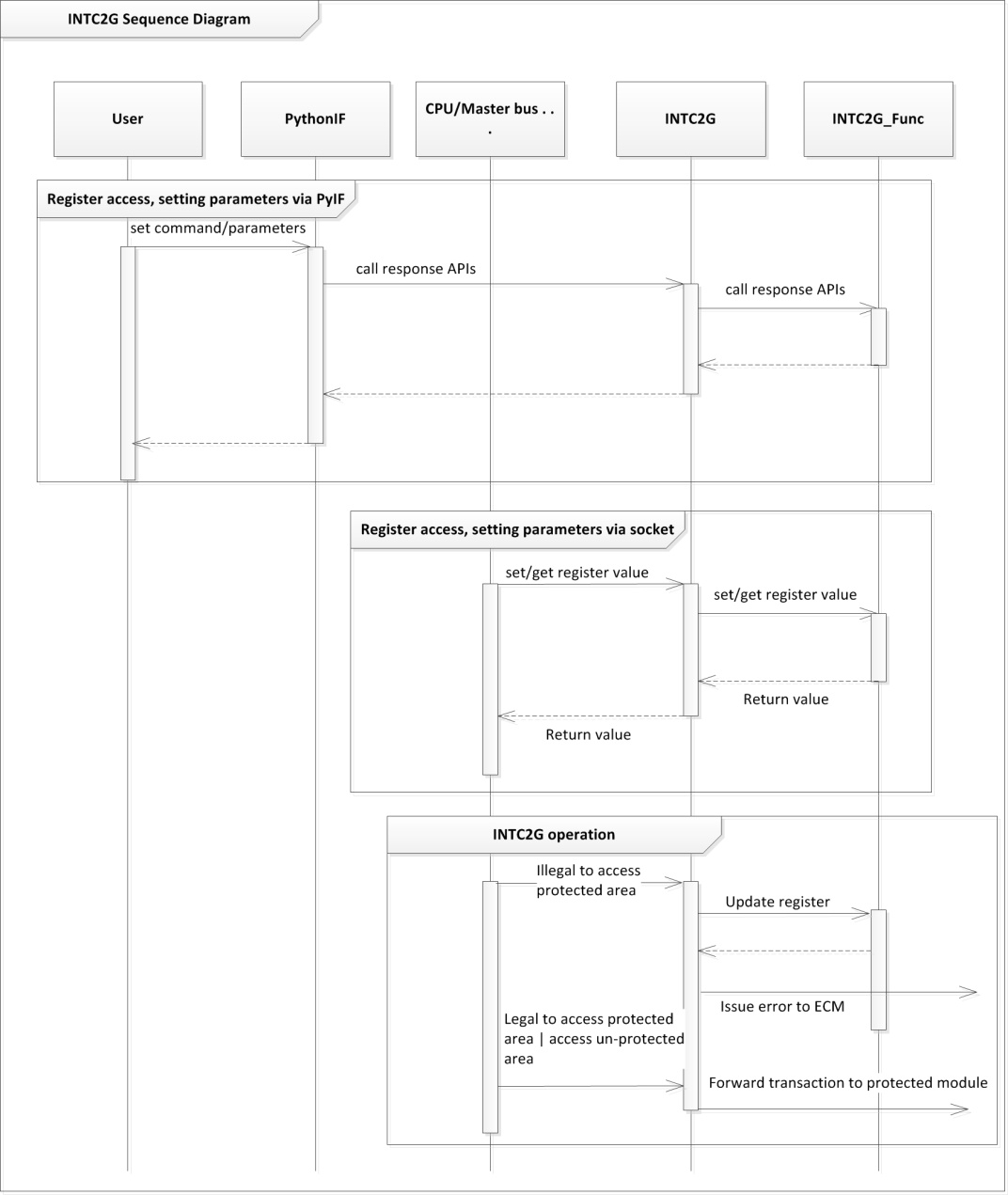


Figure 7‑1: Sequence diagram

***Explanation***:

* CPU/ other bus master model access the registers of model via the target socket named “ts”.
* User interacts with parameters/commands of model via Python IF.

.

## State diagram



Figure 7‑2: State diagram

***Explanation***:

During the whole time of simulation, the model can be in one of the following states

* RESET:
  + Transit-in conditions:
    - When reset signal or command is active, the model transits into this state from any state.
  + Transit-out conditions:
    - When reset signal is deasserted or when reset duration is expired, the model transits from this state to DISABLED state
  + Entry actions:
    - The model cancels all internal scheduled events
    - The model initializes all internal variables
    - The model initializes all outputs
  + Do actions:
    - The model discards any transaction sent to it
    - The model ignores all external events on input signals
  + Exit actions:
    - The model does nothing when transiting into other states
* DISABLED:
  + Transit-in conditions:
    - When the simulation starts, the INTC2G in this state
    - When reset signal is deasserted or when reset duration is expired, the model transits from RESET into this state
  + Transit-out conditions:
    - When reset signal or command is active, the model transits into RESET state.
    - When the model in PROTECT state, it is disabled by set GEN bit to 0, then the model transits into this state
  + Entry actions:
    - The model does nothing when transiting into this state
  + Do actions:
    - the model forwards any transaction sent to it with address targets to the protected address ranges
  + Exit actions:
    - The model does nothing when transiting into other states
* PROTECT:
  + Transit-in conditions:
    - When GEN field is set on, the model transits from DISABLED state into this state
  + Transit-out conditions:
    - When reset signal or command is active, the model transits into RESET state.
    - When the model in DISABLE state, it is enabled by set GEN bit to 1, then the model transits into this state
  + Entry actions:
    - The model does nothing when transiting into this state
  + Do actions:
    - The model checks any transaction sent to it with address targets to the protected address ranges, and then it decides whether to transfer those transactions to the targeted protected peripherals or not
  + Exit actions:
    - The model does nothing when transiting into other states

.

## Validating transactions

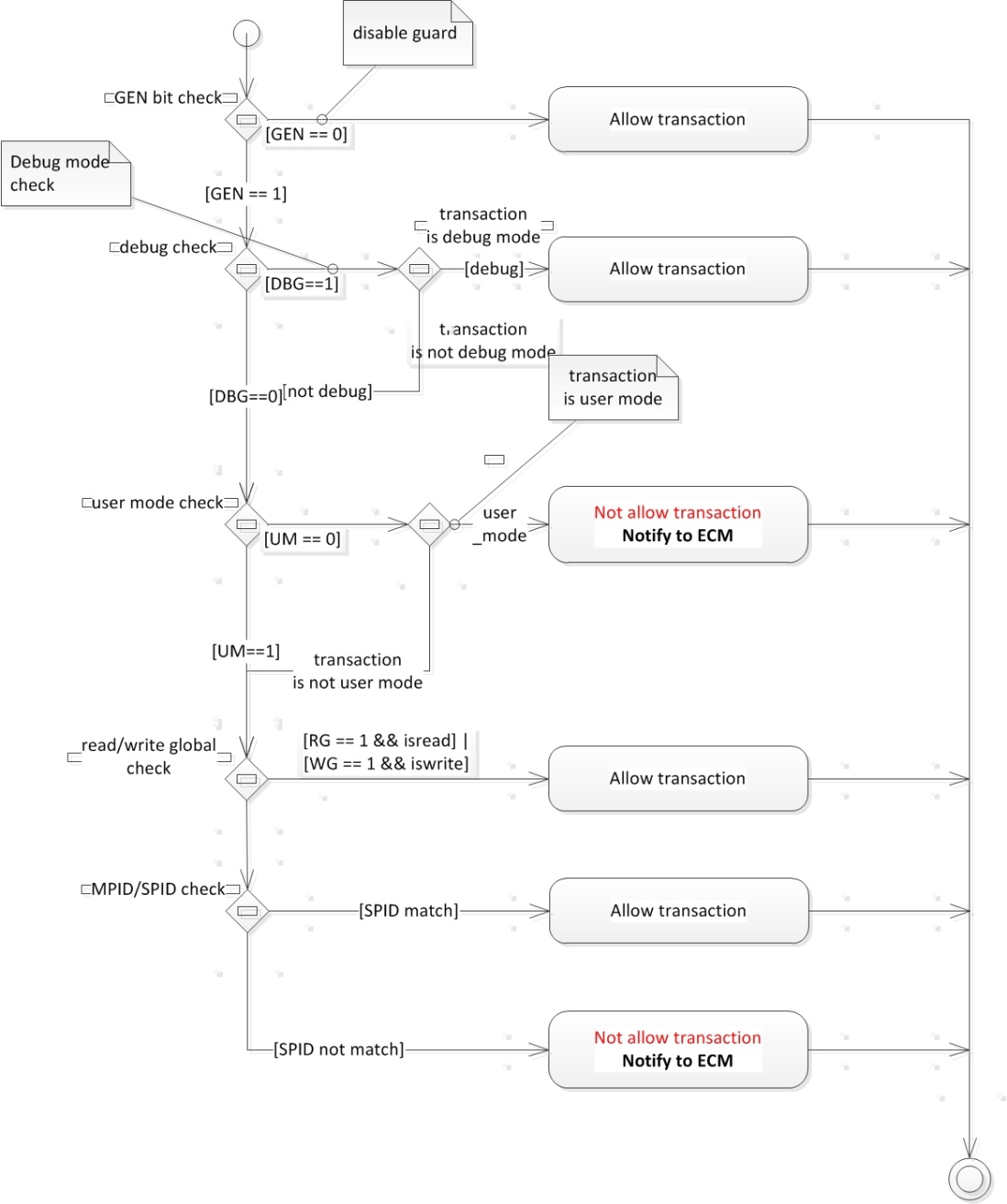


Figure 7‑3: Validating transactionsactivity diagram

***Explanation***:

The activity diagram above describes steps from which a read/write transaction suffers before it is transferred to the target protected peripheral.

* The validation process is triggered when the INTC2G receives a read/write transaction and detects channel ID by using address of transaction. This channel ID will be used to specify the register which is used.
* Firstly, the INTC2G has to be in enable state, indicated by GEN field on INTC2GPROT\_n/GR (correspond with channel ID) register; if not, the read/write transaction will be transferred .
* Next, the field DGB debug-enable in INTC2GPROT\_n/GR register is considered. This ensuresall debug read transactions are passed to the target protected peripheral when debug-enable feature is turned on. In this case, if the received read/write transaction is a debug transaction, indicated by debug field in the payload, it is passed to the target protected peripheral, otherwise, the transaction is passed to the next checks.
* User mode check, in turn, ensures that none of user-mode transaction, indicated by a field in the transaction payload, is passed to the protected peripheral, and only supervisor transactions are allowed when the INTC2G is set to disable user mode. In this case, if the received transaction is a user-mode transaction, it will be discarded with error report.
* Read-global/write-global check is taken place after these checks. The main function of INTC2G is to protect the peripherals from illegal read access and only allows specific initiators to access to a protected address range. When read-global/write-global is turn on, the INTC2G will allow access from any initiator.
* If read global/write field is off, the last check is carried out. At this step, SPID field of the read/write transaction is checked. Setting in bits MPID on register INTC2GPROT\_n/GR will be decoded to SPID values which specify exactly initiators allowed to send read/write transaction; if the SPID of the received read/write transaction is from an initiator in this white-list, the transaction is passed to the target protected area.

## Key Code Protected

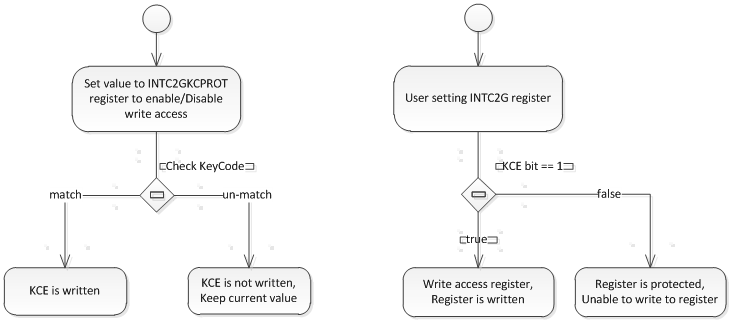


Figure 7‑4: Key Code Protected flow diagram

***Explanation***:

+ Only INTC2GMPIDn, INTC2GPROT\_GR, INTC2GPROT\_m registers are affected by this function

+ The flow diagram above describes steps when write to INTC2G registers above.

+ Write access of INTC2G registers above is protected by KCE bit of register INTC2GKCPROT. And KCE bit, itself is protected by bits INTC2GKCPROT.KCPROT

## Reporting error transactions



Figure 7‑5: Reporting error transactions activity diagram

***Explanation***:

The activity diagram above describes procedure for reporting illegal accesses. These steps might be processed at the same time.

* Error-field ERR in the INTC2GERRSTAT is not turned on then :
  + The address field of the illegal transaction is saved into INTC2GERRADDR register.
  + Additional information, namely debug-mode, user-mode, SPID value, read-write access type of that transaction is saved into INTC2GERRTYPE register.
  + Error-field ERR in the INTC2GERRSTAT must be turned on.
* In the case it has already turned on before, overflow-field OVF in INTC2GGERRSTAT must be turned on.
* Notify an error to ECM

.

## Sending error interrupt request



Figure 7‑6: Sending error interrupt request acitivity diagram

***Explanation***:

The activity diagram above details actions of error reporting thread err\_irq\_scth.

* When an illegal transaction is detected, an err\_irq\_scev event is fired and then handled by this thread err\_irq\_scth. It waits for the next active clock edge and then raises an interrupt request on err\_irq output to notify the ECM. The pulse lasts only 1 clock cycle.
* During the time this thread is waiting to the next clock cycle, if an asynchronous reset is active, this thread is moved to the default wait state.

.

## Decoding from transaction address into the protected address range

This section describes an important part of data structure of the model in detail.

The model makes use of std::map in order to decode the target-address of a received transaction into the identifier of the protected address range; and then by the found id, it gets the corresponding setting parameters to the validation process taken place in further steps. This section focuses on decoding step.

std::map, a template container in C++ standard library, takes two input types: key and value types as required input templates. Furthermore, the key type must support “less-than” comparison operator due to the fact that the power of map data structure is the capability of searching on the sorted list of keys quickly and comparison is a fundamental requirement for sorting those keys.

For clear code and short explanation, these data types are redefined as follows:

1. typedef std::pair<uint32\_t,uint32\_t> addrng;
2. typedef uint32\_tchanid;
3. typedef std::map<addrng,chanid>addrmap;

“addrng”, short form of address range, is a pair of higher and lower address of an address range, which in turn are 32-bit unsigned integers. In most of cases, higher address is really greater than lower address. addrng can also specifies a single address; and in this use-case, higher address equals to lower address.

“chanid”, short form of channel identifier, is 32-bit unsigned integer values.

“addrmap” is a std::map; it utilizes “addrng” as key type and “chanid” as value type. As a std::map, it ensures each key is absolutely unique, and consequently, it eliminates the potential overlapsof address ranges, which is valuable build-in error immunity.

To full-fill the requirement of a map, key type addrng supports a “less-than” comparison operator defined as follow:

1. typedef std::pair<uint32\_t, uint32\_t> addrng;
2. bool lessThan(const addrng& leftAddrng, const addrng& rightAddrng) {
3. return (leftAddrng.first < rightAddrng.second);
4. }

The implementation above shows that an address range leftAddrng is less than rightAddrng (leftAddrng < rightAddrng) if and only if the higher address of the range leftAddrng is less than the lower address of the range rightAddrng. The advantage of this definition is to eliminate the overlap of address ranges.

Investigating the following examples is the best ways illustrate how to use these data structures:

Example 1:

1. addrng r1 = addrng(7, 3);
2. addrng r2 = addrng(10, 6);
3. addrng r3 = addrng(10, 9);
4. addrng a1 = addrng(3, 3);
5. addrng a2 = addrng(7, 7);
6. addrng a3 = addrng(4, 4);
7. addrng a4 = addrng(8, 8);
8. addrng a5 = addrng(9, 9);
9. addrng a6 = addrng(11, 11);
10. r1 == r2; //if( (!(r1<r2)) && (!(r2<r1))) else r1 == r2;
11. r2 == r3;
12. r1 < r3; // there is no transitive-relation here!
13. r1 == a1;
14. r1 == a2;
15. r1 == a3;
16. a1 < a2;
17. a2 > a3;
18. a1 < a3; // there is transitive-relation here!
19. r1 < a4;
20. a4 == r2;
21. a4 < r3;
22. r1 < a5;
23. a5 == r3;
24. r3 < a6;

In the example above, r1, r2 and r3 specifies sets of addresses. Intersection of sets r1 and r2 is not an empty set, in other words, they overlaps each other, so they are equal. Set r2 and r3 also are equal because of the same reason. But set r1 and r3 is not overlap, so they are not equal. Notice that here is no transitive relation here.

A single address can be express in the term of set, for example a1, a2 and a3 are single addresses.

When applying these relations into std::map addrmap, the addrmap ignores address range r2 if r1, r2, and r3 are inserted into the map in this order. As a result, the overlaps between r1-r2 and between r2-r3 are eliminated.

The map also accelerates searching operation because of its sorted-order nature and build-in quick-search algorithms on sorted sets. For given addresses extracted from the transactions, for example a1, a2, a3, a4, a5 and a6, the map treats them as search-keys and searches them through the existent keys r1 and r3. The search results should be as follows:

* a1, a2, a3 are mapped to range r1
* a4 is mapped to range r2.
* a5 is mapped to range r3.
* a6 is not found in the map.

By specifying corresponding pair of key addrng and value chanid, the address map addrmap not only simplifies the code, prevents data structure from the risk of address overlap, but also accelerates process of decoding from transaction address into the identifier of address range.

.

## Python IF and Heap configuration operation flow



Figure 7‑7: Python IF and Heap configuration operation flow Part 1

.



Figure 7‑8: Python IF and Heap configuration operation flow Part 2

.



Figure 7‑9: Python IF and Heap configuration operation flow Part 3

.



Figure 7‑10: Python IF and Heap configuration operation flow Part 4

.



Figure 7‑11: Python IF and Heap configuration operation flow Part 5

***Explanation***

* Users set/call the parameters/commands of the INTC2G model via the Python IF and Heap configure. Setting/calling operation is described in figures above in this section.
* The function of the parameters and commands is described in Table 6‑2,
* Table 6‑3

.

## Reset flow



Figure 7‑12: Reset flow

***Explanation***:

* The INTC2G model has 1 hardware reset and 1corresponding software reset, users can reset INTC2G model by software reset via Python IF. The Figure 7‑13 shows the relationship between software reset and hardware reset.
* If resetPortsignal is active, all registers, variables, output ports are initialized.

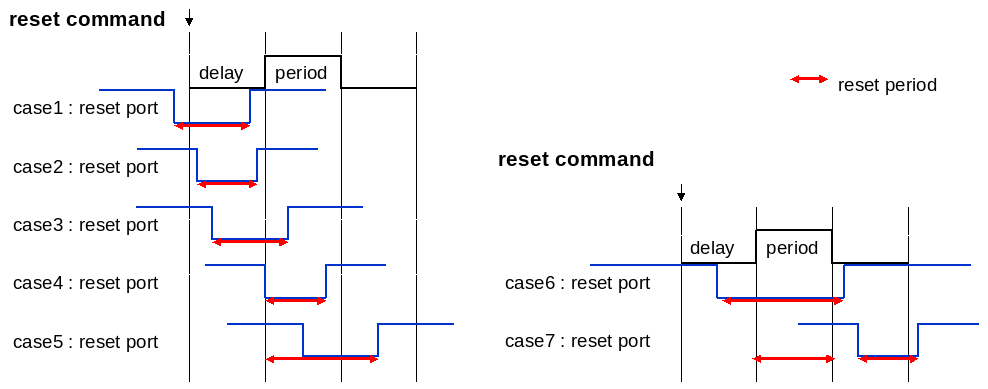


Figure 7‑13: Relationship between software reset and hardware reset

# Functions description

## INTC2G class

Table 8‑1: Public Member Functions of INTC2G class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | virtual  ~INTC2G () | Public destructor |
| 2 | INTC2G( sc\_module\_name name  , uint32\_t rLatency  , uint32\_t wLatency  , SIM\_MODE\_T simmode  ) | Public contructor |
| 3 | void  AssertReset (const std::string reset\_name, const double start\_time, const double period) | Assert and de-assert reset signal. |
| 4 | void  SetCLKFreq (const std::string clock\_name, const sc\_dt::uint64 freq, const std::string unit) | Set frequency value to these blocks. |
| 5 | void  GetCLKFreq (const std::string clock\_name) | Get frequency value of these blocks. |
| 6 | void  ForceRegister (const std::string reg\_name, const unsigned int reg\_value) | Force value to register by the software via Python interface. |
| 7 | void  ReleaseRegister (const std::string reg\_name) | Release forced value after forcing registers by the software via Python interface. |
| 8 | void  WriteRegister (const std::string reg\_name, const unsigned int reg\_value) | Write value to registers by software by the software via Python interface. |
| 9 | void  ReadRegister (const std::string reg\_name) | Read value of register by software by the software via Python interface. |
| 10 | void  ListRegister (void) | List all registers name by software by the software via Python interface. |
| 11 | void  Help (const std::string type) | Dump help message of all parameters or commands. |
| 12 | void  SetMessageLevel (const std::string msg\_lv) | Select debug message level ("fatal", "error", "warning", "info"). |
| 13 | void  DumpRegisterRW (const std::string is\_enable) | Dump register access information when registers are accessed. |
| 14 | void  DumpInterrupt (const std::string is\_enable) | Enable/disable dumping interrupt information. |
| 15 | bool  CheckGuardClock () | Check if period value of the clock clocking the bus guard is greater than 0 or not. |
| 16 | bool  GetGuardResetStatus () | Check status of the combination of hard reset and command reset that controls the bus guard is active or not |
| 17 | void  printMsg (const char \*severity, const char \*msg) | Function allows the bus guard to report diagnosis message for debug |
| 19 | bool  SetAreaAddr (const uint32\_t channelId, const uint32\_t start\_addr, const uint32\_t size) | Add a address range into the map of protected range. |
| 20 | void  TransferErrInfo (const uint32\_t channelId, const uint32\_t addr, const bool isDebug, const bool isUserMode, const bool isWrite, const uint8\_t SPID) | The function API allows the bus guard to report error to the rest of system whenever it detects an illegal transaction |
| 21 | bool  GetGuardSettings (const uint32\_t channelId, bool &GEN, bool &DBG, bool &UM, bool &WG, bool &RG, uint32\_t &SPID) | The APIs allows the bus guard to get information of validation settings from registers in the registers block. |
| 22 | uint32\_t spidDecode(uint8\_t MPID\_value) | Decode SPID (onehot code) from MPID value and register INTC2GMPIDn |
| 23 | bool GetResetStatus() const; | Check status of the combination of hard reset and command reset |
| 24 | void DumpStatusInfo(); | Dump the status information of model |
| 25 | uint32\_t GetINTC2GERRSTAT(); | Get value of INTC2GERRSTAT register |

.

Table 8‑2: Public Data Fields of INTC2G class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | sc\_in< sc\_dt::uint64 >  PCLK | Positive edge active clock PCLK |
| 2 | INTC2G\_ApbTlmTargetSocket\*  ts | For register block of the INTC2G |
| 3 | INTC2G\_ApbTlmTargetSocket \*  tsp | To receive protected transactions |
| 4 | INTC2G\_ApbTlmInitiatorSocket \*  isp | To forward protected transactions |
| 5 | sc\_out< bool >  err\_irq | Interrupt notifies that the bus guard detected an illegal transaction |

.

Table 8‑3: Private Member Functions of INTC2G class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | void  PCLKMethod () | Monitor clock |
| 2 | bool  CheckClockPeriod (const std::string clock\_name) | Check if clock period value is greater than 0 or not. |
| 3 | sc\_time  CalculateClockEdge (const std::string clock\_name, const bool is\_pos=true) | Calculate synchronous time. |
| 4 | double  CalculateClockEdge (const std::string clock\_name, const bool is\_pos, const double time\_point) | Calculate synchronous time. |
| 5 | void  resetPortHandleMethod () | Monitor resetPort port |
| 6 | void  resetCmdHandleMethod () | Process reset function when reset command is active. |
| 7 | void  CancelResetCmdMethod () | Cancel reset function when reset command is active. |
| 8 | void  Initialize (void) | Initialize outputs and internal variables, internal states, internal signals |
| 9 | void  CancelEvents (void) | Cancel all scheduled events |
| 10 | void  EnableReset (const bool isActive) | Reset model and its sub-instance if reset is active. |
| 11 | void  GetTimeResolution (sc\_dt::uint64 &resolution\_value, sc\_time\_unit &resolution\_unit) | Adjust the resolution of time |
| 12 | void  SeparateString (std::vector< std::string >&vtr, const std::string msg) | Separate a string to a vector of separated sub-strings. |
| 13 | void  ConvertClockFreq (sc\_dt::uint64 &freq\_out, std::string &unit\_out, sc\_dt::uint64 freq\_in, std::string unit\_in) | Check frequency value and frequency unit. |
| 14 | void  DumpInterruptMsg (const std::string intr\_name, const bool value) | Print out value of boolean signal interrupt |
| 15 | void  AssertErr\_IrqMethod () | Process asserts the err\_irq output according to clock edge |
| 16 | void  DeassertErr\_IrqMethod () | Process de-asserts the err\_irq output according to clock edge |
| 17 | void UpdateErr\_IrqMethod(); | Process to write to err\_irq sinal |

.

## INTC2G\_Func Class

Table 8‑4: Public Member Functions of INTC2G\_Func class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | virtual  ~INTC2G\_Func () | Public destructor |
| 2 | void  RegisterHandler (const std::vector< std::string >&args) | Handle command passed into the register block via Python interface |
| 3 | void  read (uint32\_t offsetAddress, TlmBasicPayload &trans, BusTime\_t \*t, bool debug) | Read access implementation BusSlaveFuncIf::read via BusSlaveFuncIf |
| 4 | void  write (uint32\_t offsetAddress, TlmBasicPayload &trans, BusTime\_t \*t, bool debug) | write access implementation of BusSlaveFuncIf::write via BusSlaveFuncIf |
| 5 | void  cb\_INTC2GERRCLR\_CLRE (RegCBstr str) | Implement a call-back of write access into clear error status field of guard error status clear register |
| 6 | void  cb\_INTC2GERRCLR\_CLRO (RegCBstr str) | Implement a call-back of write access into clear overflow status field of guard error status clear register |
| 7 | void cb\_INTC2GMPID\_SPID(RegCBstr str) | Implement a call-back of write access into INTC2GMPIDn register |
| 8 | void  cb\_INTC2GKCPROT\_KCE (RegCBstr str) | Implement a call-back of write access into key code enable field of key code protect register |
| 9 | void  cb\_INTC2GKCPROT\_KCPROT (RegCBstr str) | Implement a call-back of write access into key code protection field of Key code protect register |
| 10 | void  cb\_INTC2GPROT\_RG (RegCBstr str) | Implement a call-back of write access into write-global enable, user-mode, debug-mode, global-enable fields of protection setting register (INTC2GPROT\_n) |
| 11 | void  cb\_INTC2GPROT\_GR\_RG (RegCBstr str) | Implement a call-back of write access into write-global enable, user-mode, debug-mode, global-enable fields of protection setting register (INTC2GPROT\_GR) |
| 12 | uint32\_t GetINTC2GERRSTAT(); | Get value of INTC2GERRSTAT register |

.

.

Table 8‑5: Private Attributes of INTC2G\_Func class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | const uint32\_t  KEY\_CODE\_PROT | The key code protect illegal write into validating setting registers |
| 2 | INTC2G \*  pWrapper | Implement relationship between the core and the wrapper |
| 3 | string  mModuleName | Name of the function model |

.

## INTC2G\_CommonGuardAgentControllerIf Class

Table 8‑6: Public Member Functions of INTC2G\_CommonGuardAgentControllerIf class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | virtual bool  CheckGuardClock ()=0 | Check if period value of the clock clocking the bus guard is greater than 0 or not. |
| 2 | virtual bool  GetGuardResetStatus ()=0 | Check status of the combination of hard reset and command reset that controls the bus guard is active or not |
| 3 | virtual void  printMsg (const char \*severity, const char \*msg)=0 | Function allows the bus guard to report diagnosis message for debug |
| 4 | virtual bool  SetAreaAddr (const uint32\_t ch\_id, const uint32\_t start\_addr, const uint32\_t size)=0 | Add a address range into the map of protected range. |
| 5 | virtual void  TransferErrInfo (const uint32\_t channelId, const uint32\_t addr, const bool isDebug, const bool isUserMode, const bool isWrite, const uint8\_t SPID) | The function API allows the bus guard to report error to the rest of system whenever it detects an illegal transaction |
| 6 | virtual bool  GetGuardSettings (const uint32\_t channelId, bool &GEN, bool &DBG, bool &UM, bool &WG, bool &RG, uint32\_t &SPID)=0 | The API allows the bus guard to get information of validation settings from registers in the registers block. |

.

## INTC2G\_CommonGuard Class

.

Table 8‑7: Public Member Functions of INTC2G\_CommonGuard class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | INTC2G\_CommonGuard (std::string name, INTC2GCommonGuardBusBridgeModule \*Parent, INTC2GCommonGuardBusBridgeSlaveIf \*SlaveIf, INTC2GCommonGuardBusMasterIf \*MasterIf, SIM\_MODE\_T simmode, INTC2G\_CommonGuardAgentControllerIf \*AgentController\_ptr) | Constructor of the bus guard. |
| 2 | virtual  ~INTC2G\_CommonGuard () | Destructor. |
| 3 | bool  SetAreaAddr (const uint32\_t channelId, const uint32\_t start\_addr, const uint32\_t size) | Add a address range into the map of protected range. |

.

Table 8‑8: Private Member Functions of INTC2G\_CommonGuard class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | void  SetTransBasic (TlmBasicPayload &inTrans, TlmBasicPayload &outTrans) | Copy basic info (address, data, length)from the input transaction into the output transaction. |
| 2 | void  TransferComplete (void) | Notify transaction completion in AT mode. |
| 3 | void  read (unsigned int offsetAddress, TlmBasicPayload &trans, BusTime\_t \*t, bool debug) | Overwrite virtual functions of BusBridgeCoreBase class. |
| 4 | void  write (unsigned int offsetAddress, TlmBasicPayload &trans, BusTime\_t \*t, bool debug) | Overwrite virtual functions of BusBridgeCoreBase class. |
| 5 | void  deleteTrans (void) | Overwrite virtual functions of BusBridgeCoreBase class. |
| 6 | void  notifyTransComp (TlmBasicPayload &trans, BusTime\_t &t) | Overwrite virtual functions of BusBridgeCoreBase class. |
| 7 | bool  CheckAccessPermission (TlmBasicPayload &trans, const bool debug) | Check access ability to peripherals. |

.

Table 8‑9: Private Attributes of INTC2G\_CommonGuard class

|  |  |  |
| --- | --- | --- |
| No | Name | Description |
| 1 | INTC2GCommonGuardBusBridgeSlaveIf \*  mSlaveIf | The bus master cooperates with the bus guard. The bus guard notifies completions of received transactions via this interface |
| 2 | INTC2GCommonGuardBusMasterIf \*  mMasterIf | The bus master cooperates with the bus guard. The bus guard forwards read and write transactions via this interface role |
| 3 | INTC2G\_CommonGuardAgentControllerIf \*  mAgentController | Relationship with the agent controller |
| 4 | std::map< TlmBasicPayload \*,TlmBasicPayload \* >  mRequestMap | Bus request map for AT mode. |
| 5 | std::queue< TlmBasicPayload \* >  mClearTransFifo | Clear transaction list for AT mode. |
| 6 | TlmBasicPayload \*  mTrans | Transaction info. |
| 7 | BusTime\_t  mTransTime | Internal time. |
| 8 | bool  mIsDebug | Is debug transaction. |
| 9 | INTC2G\_CommonGuard\_AddrMap  addrMap | Map with the keys are address ranges and id value of these address |

.

# Limitation

* There is no limitation in the model.

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Modified points** | **Approver** | **Checker** | **Author** |
| 1.0 | Create new | Dung Nguyen  03/01/2017 | FPT/Truong  02/22/2017 | FPT/Ha  02/07/2017 |
| 1.1 | Fix size of INTC2GMPID register Table 4‑1  Fix File structure Figure 6‑1 && Table 6‑1  Add message No 51-> 84 Table 6‑9  Fix “How to use” Table 6‑4  Add function description No21 Table 8‑1  Remove unused functions Table 8‑4  Remove unused functions Table 8‑6 | Dung Nguyen  03/22/2017 | FPT/Truong  03/15/2017 | FPT/Ha  03/09/2017 |
| 1.2 | - Update Table 4‑1 for some registers’ size and initial value: INTC2GERRCLR, INTC2GERRSTAT, INTC2GERRTYPE, INTC2GMPIDn.  - Add reference [8] | RVC/Binh  09/01/2017 | RVC/ChuongLe  09/01/2017 | FPT/Ha  08/17/2017 |
| 1.3 | - Update Table 4‑1 for some registers’ write access size value: INTC2GERRCLR, INTC2GERRSTAT, INTC2GERRTYPE, INTC2GMPIDn.  - Update Figure 7‑5: Reporting error transactions activity diagram changed step save address and fields DBG, UM, SPID, access type.  - Add INTC2G\_DumpStatusInfo function in Table 6‑2: List of parameters  - Modify reporting error transaction operation in Reporting error transactions  - Add DumpStatusInfo() and GetINTC2GERRSTAT() function in Table 8‑1: Public Member Functions of INTC2G class  - Add GetINTC2GERRSTAT() function in Table 8‑4: Public Member Functions of INTC2G\_Func class  - Modify Python IF and Heap configuration operation flow in Figure 7‑7: Python IF and Heap configuration operation flow Part 1  - Modify Reset flow in Figure 7‑12: Reset flow  - Modify result of Example 1 in Decoding from transaction address into the protected address range | Chan Le  07/05/2018 | Thanh Le  07/04/2018 | FPT/Huong  06/08/2018 |