C Model Design Guide

(Rev.1.2)

Renesas Electronics Corporation
Front-End Design Technology Development Department

2010/09/21 Rev. 1.2

About This Guide

- This guide explains how to describe:
  (1) C++ reference model
  (2) SystemC model for high-speed system simulation
  (3) SystemC model for bus-accurate system simulation

  * Modeling for behavioral synthesis is not covered in this guide.

- To understand code examples in this guide, basic knowledge of C++ and SystemC is required.

- Follow coding rules for better interoperability, reusability and quality.
  - C++/SystemC Coding Rule（Reference [1]）

Guide Organization

- Organization

  Chapter 1. C++ Reference Model
  Chapter 2. SystemC Model for System Simulation
  Chapter 3. Using Model on System Simulators
  Chapter 4. Modeling Techniques
  Appendix. References

- Chapters to read

  For C++ reference model developers:
  -> Chapters 1 and 4
  For SystemC model developers:
  -> Chapters 2, 3 and 4

C Model Design Guide

Chapter 1.
C++ Reference Model

Renesas Electronics Corporation
Front-End Design Technology Development Department

2010/09/21 Rev. 1.2

Chapter 1 Organization

- Organization
  - 1.1 C++ Reference Model Outline
  - 1.2 C++ Reference Model Rules
  - 1.3 C++ Reference Model Sample

1-1. Introduction

- C++ Reference Model
  - Used for functional verification and/or expected value generation
  - Reusable in SystemC model for system simulation by following some rules described in this chapter

- Notations

☐	C++ class
🟪	Address Mapped I/O (Memory, Register)
🟦	Class member function
➡	Function call

1-1. C++ Reference Model

- Implement C++ reference model as a C++ class which includes computation part and communication part.

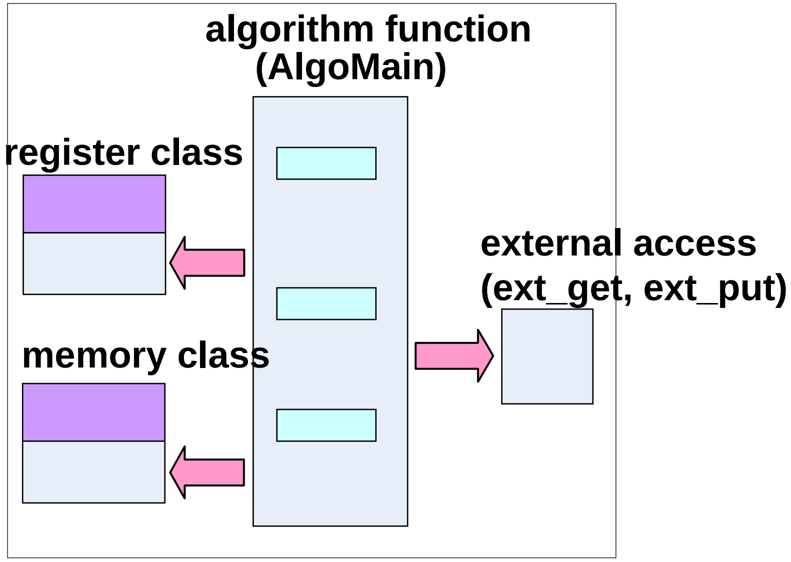
- C++ reference model is used as a reference model for IP algorithm verification, and reused as an IP model in system simulation.

- There are two model categories.
  - Target: accessed from external resource
  - Initiator: accesses external resource

- Model at higher abstraction level as much as possible to reduce development amount and simulation time.

1-1. C++ Reference Model Structure

- Memory, register and external access are modeled separately from algorithm function
  - memory and register to be accessed easily from outside
  - external access to be replaced easily in system simulation



C++ Reference Model Structure

```
#include "mem_core.h"
#include "reg_core.h"

class ip00{
public:
  mem_core  m_mem;
  reg_core  m_reg;

  ip00(){ }

  void AlgoMain();
  void mem_get( MemData* );
  void mem_put( MemData* );
  virtual void ext_get( ExtData* );
  virtual void ext_put( ExtData* );
};
```

memory

register

member functions
(ext_get()/ext_put()
defined only in initiator)

C++ Reference Model Declaration Example

1-2. C++ Reference Model Common Rules

- Common rules

(1) Avoid using global variables **Recommended**

(1) Avoid using global variables

Using global variables makes model maintenance and reuse difficult, as we need to look for wider area where they are read or written. Avoid using global variables as much as possible.

1-2. Algorithm Function Rules (1)

- Algorithm function rules

(1) Avoid describing behavior in main function	**Recommended**
(2) Use pre-defined member functions for external access	MUST
(3) Prepare function for each pipeline stage	Information
(4) Use arguments to pass data to/from pipeline stage function	Information

1-2. Algorithm Function Rules (2)

(1) Avoid describing behavior in main function

Describe detailed behavior in subroutines, instead of main function. This helps looking over whole algorithm and having subroutines easy to reuse.

```
void dct::AlgoMain()
{
    if (reg_dct["START"] == 1) {
        reg_dct["START"] = 0;

        JINT32 work[DCTSIZE2];
        read_data_from_mem(reg_mem_addr, (int*)work, DCTSIZE2);
        dct_one_block(work);
        write_data_to_mem(reg_mem_addr, (int*)work, DCTSIZE2);

        reg_dct["END"] = 1;
    }
}
```

Algorithm main function example

1-2. Algorithm Function Rules (3)

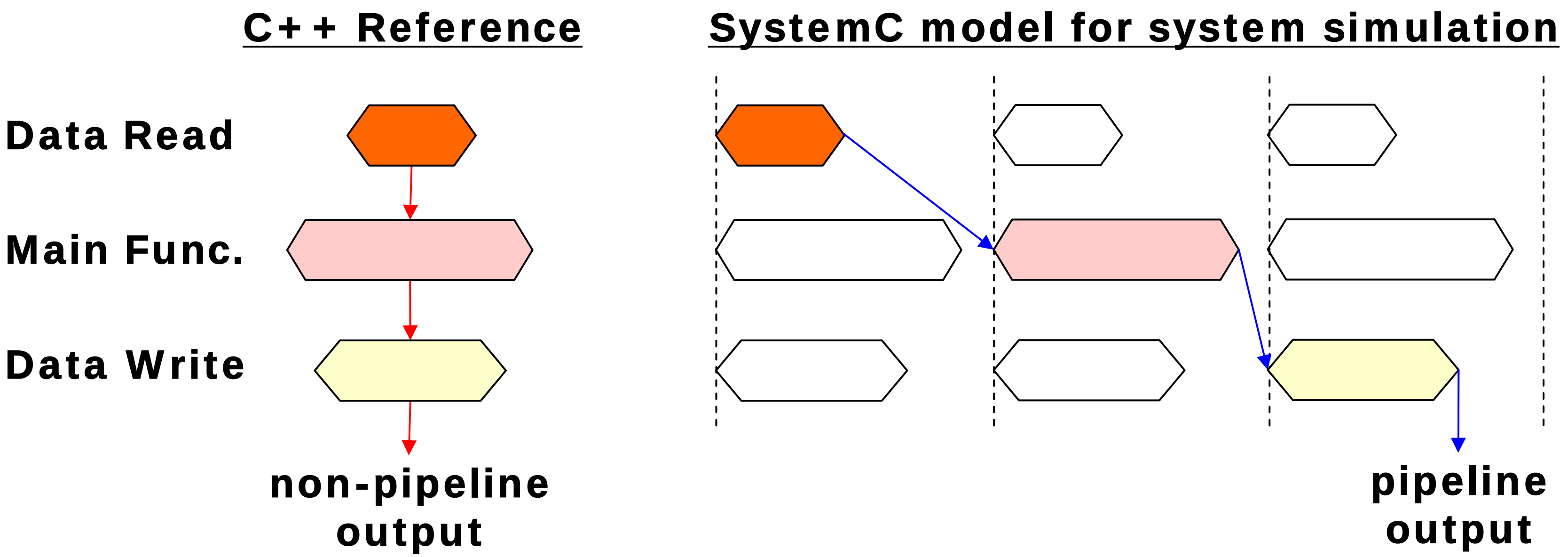
(2) Use pre-defined member functions or operators for external access

Use virtual to declare external access functions ext_put() / ext_get() so that we can easily replace these functions in system simulation. (example: file-access in algorithm verification, bus-access API in system simulation)

1-2. Algorithm Function Rules (4)

(3) Prepare function for each pipeline stage
Prepare pipeline stage functions so that we can modify the model for pipeline system simulation easily, as each pipeline stage behavior is threaded and runs in parallel.

(4) Use arguments to pass data to/from pipeline stage function
Pass data as arguments so that we can switch where to pass pipeline stage output easily.



1-2. Register Class Rules (1)

- Register class rules

(1) Use register class to separate register model from algorithm function	MUST
(2) Use common register class (re_register)	**Recommended**

1-2. Register Class Rules (2)

(1) Use register class to separate register model from algorithm function

Registers are accessed by testbench or system simulator as well as by algorithm function. Using register class can simplify register access implementation.

(2) Use common register class (re_register)

Common register class (re_register) is available from C model library (Reference [2]). The class is equipped with basic register functions and access operators.

1-2. Memory Class Rules (1)

- Memory class rules

(1) Use memory class to separate memory model from algorithm function	MUST
(2) Memory class consists of array and its get/put access functions	**Recommended**
(3) Do not directly call get/put access functions from algorithm function	**Recommended**

1-2. Memory Class Rules (2)

(1) Use memory class to separate memory model from algorithm function

- Memory located outside the IP

  Use memory class so that we can easily replace memory access functions with external access functions in system simulation.

- Memory accessed by outside the IP

  Use memory class so that we can easily implement memory access by outside the IP.

- Memory located inside the IP and not accessed by outside the IP

  It is ok to use simple array.

1-2. Memory Class Rules (3)

(2) Memory class consists of array and its get/put access functions
Using get/put access functions can simplify memory access monitor.

(3) Do not directly call get/put access functions from algorithm function
Using mem_get/mem_put which calls get/put helps easily replace mem_get/mem_put with ext_get/ext_put in system simulation.

```
class mem_core {
public:
  mem_core()  { }

  virtual void get (MemData* d) {
    d->data = array[d->addr];
  }
  virtual void put (MemData* d) {
    array[d->addr] = d->data;
  }

private:
  unsigned int  array[MEM_CORE_SIZE];
};
```

Memory access functions (memory class)

```
struct MemData {
  unsigned int  addr;
  unsigned int  data;
};
```

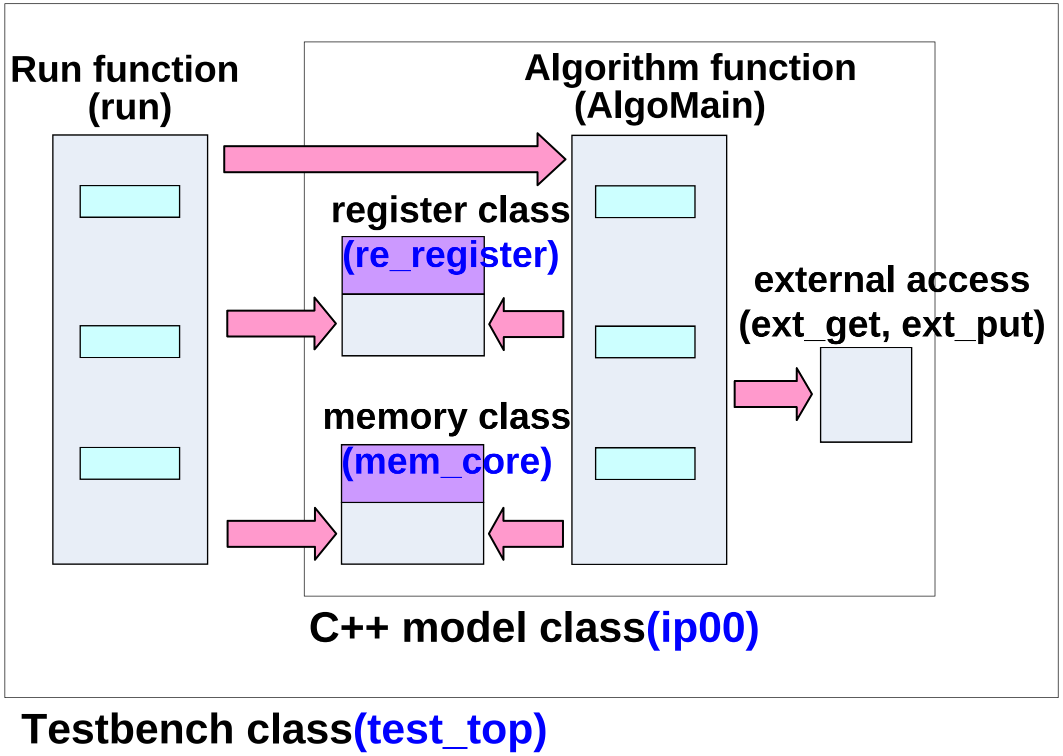
Memory access struct (memory class)

```
void ip00::mem_get( MemData* d ){
  m_mem.get( d );
}
void ip00::mem_put( MemData* d ){
  m_mem.put( d );
}
```

Memory access functions (algorithm function)

1-3. C++ Reference Model Sample(1)

- Sample structure



C++ model class
ip00.h
ip00.cpp

Testbench class
test_top.h
test_top.cpp

Memory class
mem_core.h

Register class
re_register.h
reg_super.h
re_register.cpp

1-3. C++ Reference Model Sample(2)

- Sample behavior
  - Testbench initializes memory and registers
  - Testbench calls algorithm function in C++ model class
  - Algorithm function updates memory data depending on reg values
  - Testbench displays updated memory data

Register name	Size	Description
reg00	4	operation start (1:start、0:stopped)
reg01	4	number of data

Memory name	Size	Description
m_mem	4096	data array of 1byte data

1-3. C++ Reference Model Sample(3)

- C++ reference model class

ip00.h

```
#ifndef _IP00_H_
#define _IP00_H_

#include "mem_core.h"
#include "re_register.h"

struct ExtData {
  unsigned int  addr;
  unsigned int  data;
};

/* class definition */
class ip00
: public vpcl::reg_super{
public:
  /* memory and register */
  mem_core  m_mem;
  vpcl::re_register  reg00;
  vpcl::re_register  reg01;

/* constructor */
  ip00():
    reg00(0, this, "reg00")
  , reg01(4, this, "reg01")
  { }

  /* function prototypes */
  void AlgoMain();
  void mem_get( MemData* );
  void mem_put( MemData* );
  virtual void ext_get( ExtData* );
  virtual void ext_put( ExtData* );
};

#endif // _IP00_H_
```

Register declarations using common register class (re_register)

ip00.cpp

```
#include <cstdio>
#include "ip00.h"

void ip00::AlgoMain() {
  unsigned int  i;
  MemData  d;
  if( reg00 ) {
    for( i=0; i<reg01; i++ ) {
      d.addr = i;
      mem_get( &d );
      if (i%2) {
        d.data++;
      } else {
        d.data = 0;
      }
      mem_put( &d );
    }
  }
}

void ip00::mem_get( MemData* d ) {
  m_mem.get( d ) ;
}

void ip00::mem_put( MemData* d ){
  m_mem.put( d ) ;
}

void ip00::ext_get( ExtData* d ) {
}

void ip00::ext_put( ExtData* d ) {
}
/* eof */
```

Algorithm function : update mem data (addr 0 to reg01-1)
odd addr -> increment
even addr -> zero

1-3. C++ Reference Model Sample(4)

Testbench class

test_top.h

```
#ifndef _TEST_TOP_H
#define _TEST_TOP_H

#define TB_USE_SIZE 8
#include "ip00.h"

class test_top {
public:
  ip00  ip00_i;

  test_top()  { }

  /* function prototype */
  void  run();
};

#endif  // _TEST_TOP_H
```

test_top.cpp

```
#include <cstdio>
#include "test_top.h"

void testtop::run() {
  unsigned int  i;
  MemData  d;

  printf("TB to set REG and MEM initial values¥n");
  ip00_i.reg00 = 1;
  ip00_i.reg01 = TB_USE_SIZE;
  for( i=0; i<TB_USE_SIZE; i++ ) {
    d.addr = i ;
    d.data = i ;
    ip00_i.mem_put( &d ) ;
    printf("  mem[%x] set to %x¥n", d.addr, d.data);
  }

  printf("TB to run AlgoMain() and update MEM¥n");
  ip00_i.AlgoMain();

  printf("TB to dump results¥n");
  for( i=0; i<TB_USE_SIZE; i++ ) {
    d.addr = i ;
    ip00_i.mem_get( &d ) ;
    printf("  mem[%x] is %x¥n", d.addr, d.data);
  }
}
```

Initialize registers and memory

Call algorithm function

Display results

```
int main()
{
  test_top  test_top_i;
  test_top_i.run();
  return (0);
}
```

1-3. C++ Reference Model Sample(5)

- Memory class

mem_core.h

```
#ifndef _MEM_CORE_H_
#define _MEM_CORE_H_
#define MEM_SIZE  4096
#define MEM_MASK  0x00000FFF

struct MemData {
  unsigned int  addr;
  unsigned int  data;
};

class mem_core {
public:
  mem_core()  {}

  virtual void get (MemData* d)  {
    d->data = array[d->addr & MEM_MASK];
  }
  virtual void put (MemData* d)  {
    array[d->addr & MEM_MASK]  = d->data;
  }

private:
  unsigned int  array[MEM_SIZE];
};

#endif // _MEM_CORE_H_
```

- Register class

re_register.h
reg_super.h
re_register.cpp

You can download the Register class files from C model library
（Reference ［2］）

1-3. C++ Reference Model Sample(6)

Compile and Run

```
bs -os SUSE9_0  g++ ./ip00.cpp ./test_top.cpp ¥          # build
  ./re_register.cpp -o run.x
bs -os SUSE9_0  run.x                                    # run
```

Result (log)

```
TB to set REG and MEM values
mem[0] set to 0
mem[1] set to 1
mem[2] set to 2
mem[3] set to 3
mem[4] set to 4
mem[5] set to 5
mem[6] set to 6
mem[7] set to 7
TB to run AlgoMain() and update MEM
TB to dump results
mem[0] is 0
mem[1] is 2
mem[2] is 0
mem[3] is 4
mem[4] is 0
mem[5] is 6
mem[6] is 0
mem[7] is 8
```

C Model Design Guide

Chapter 2.
SystemC Model for System Simulation

Renesas Electronics Corporation
Front-end Design Technology Development Department

2010/09/06 Rev. 1.2

Chapter 2 Organization

- Organization
  - 2-1. Introduction
  - 2-2. TLM Common Class
  - 2-3. Initiator High-speed Model
  - 2-4. Initiator Bus-accurate Simple Model
  - 2-5. Initiator Bus-accurate Model
  - 2-6. Target High-speed Model
  - 2-7. Target Bus-accurate Simple Model
  - 2-8. Target Bus-accurate Model
  - 2-9. Model Coding Samples

- Sections 2-4, 2-5, 2-7, and 2-8 are omitted in this English edition.

2-1. Introduction

- This chapter explains how to describe SystemC model for system level design. Here, SystemC model is classified by the application as “High-speed model” or “Bus-accurate simple model” or “Bus-accurate model”.   Applications and features of each model are shown below.

- **High-speed model**: This model type is applied to embedded software development, system performance evaluation which does not require detailed timing, or functional verification of SoC. The feature is high speed simulation.
- **Bus-accurate simple model**: This model type is applied to system performance evaluation which requires detailed timing and functional verification of SoC. The feature is accurate timing simulation.
- **Bus-accurate model**: This model type is applied to system performance evaluation which requires detailed timing and functional verification of SoC. The feature is accurate timing simulation with complicated bus access.

- SystemC model for system level design (“High-speed model”, “Bus-accurate simple model”, and “Bus-accurate model”) is described based on TLM (Transaction Level Modeling) which is one of the modeling styles in transaction level. Transaction level is that communication data between modules are combined as “transaction” and communication is executed by API call.

2-2. TLM Common Class (Contents)

2-2-1. Features

- TLM common class is a bus access interface prepared in Renesas to increase efficiency of model development. Features of TLM common class are shown below.

- **Standardize Communication Interface**

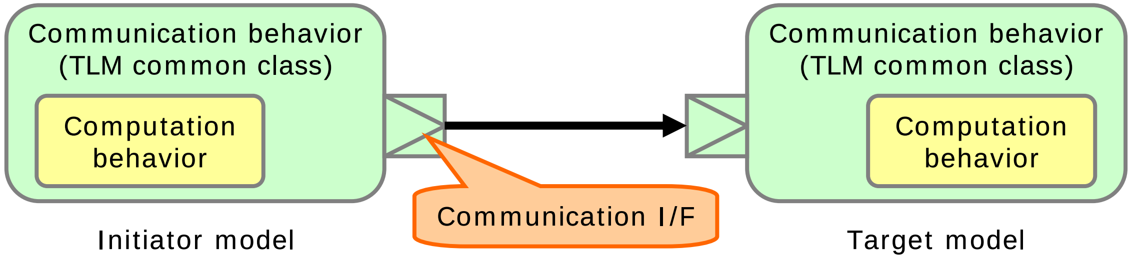
  By using OSCI TLM2.0 which define communication interface and data structure between models, interoperability of models is increased. For example, embedding models in each EDA vender tools and replacement of bus model in the system is easy. By using TLM common class, study of TLM2.0 and modeling based on TLM2.0 are unnecessary.

- **Separate Computation and Communication Behavior**

  By separating computation and communication behavior, reusability of models and ease of refinement are increased. By using TLM common class, consideration of the separation method and implementation of communication behavior are unnecessary.

- **Support Multiple Type of Models**

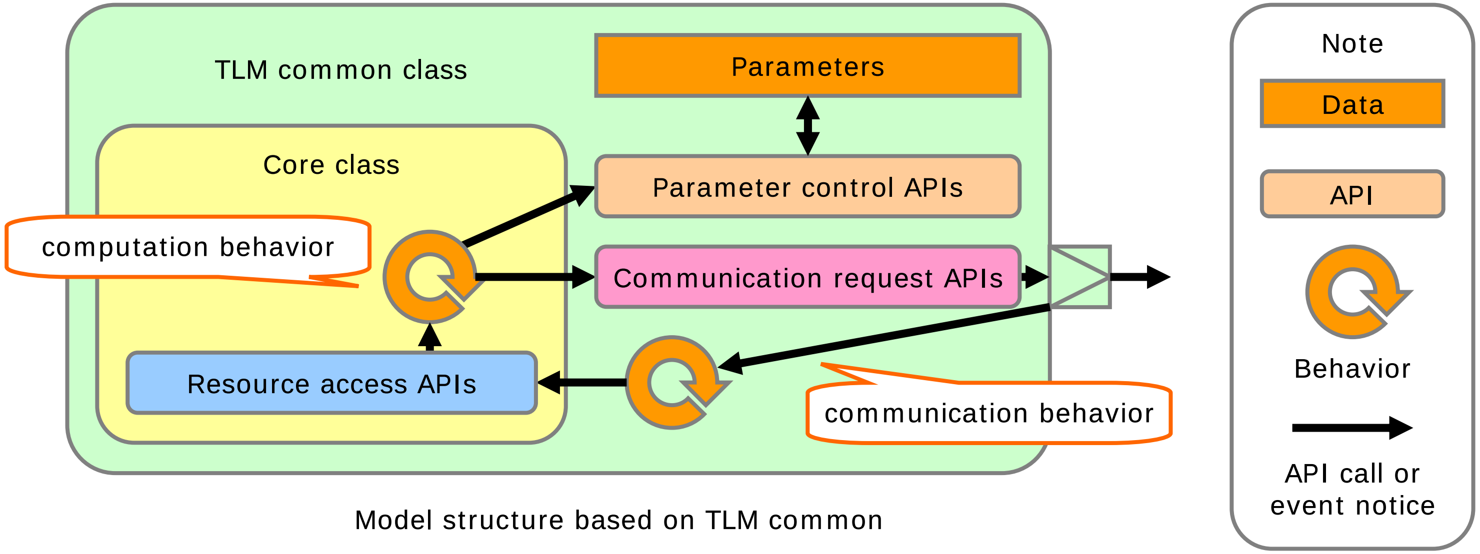
  In TLM common class, communication method can be switched according to application. TLM common class supports high-speed model, bus-accurate simple model, and bus-accurate model.



2-2-2. Model Structure

- This subsection explains model structure based on TLM common class.

- **TLM common class**: Communication behavior is implemented in this class.
  - Parameter control APIs: called by core class to control parameters.
  - Communication request APIs: called by core class to communicate with the other models.
- **Core class**: Computation behavior is implemented in this class.
  - Resource access APIs: called by TLM common class to communicate with core class.

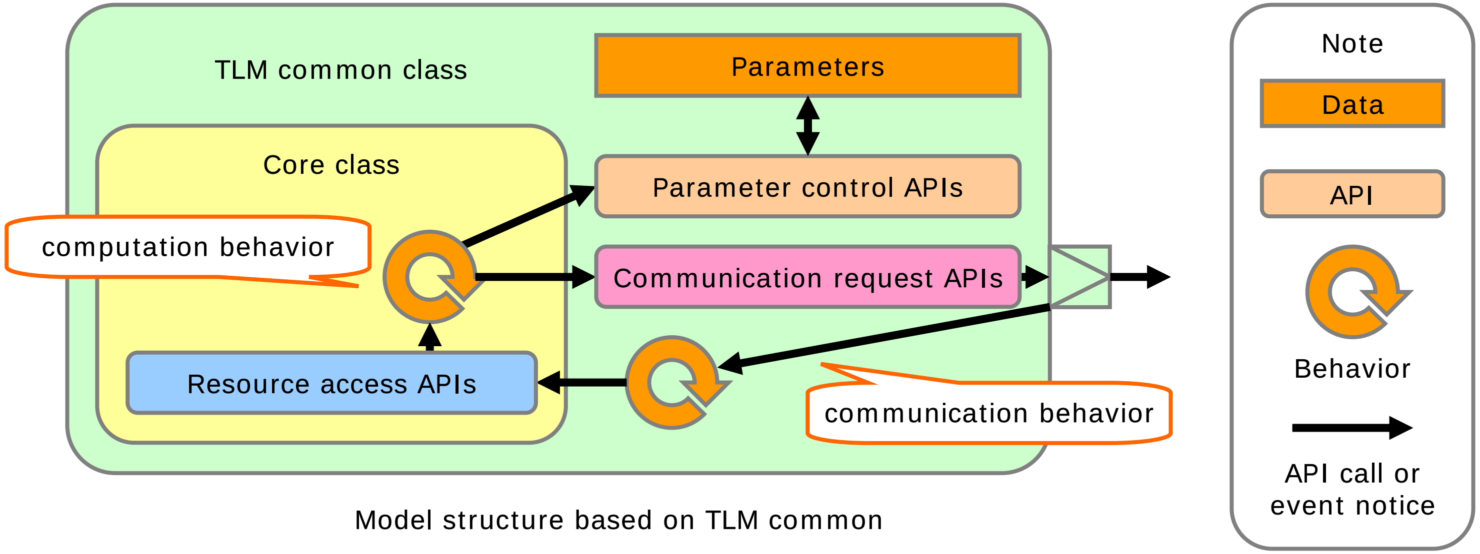


Model structure based on TLM common

2-2-3. Recommendations for TLM modeling

- This subsection explains outline of how to describe SystemC model. SystemC model developers using TLM common class describe only core class.

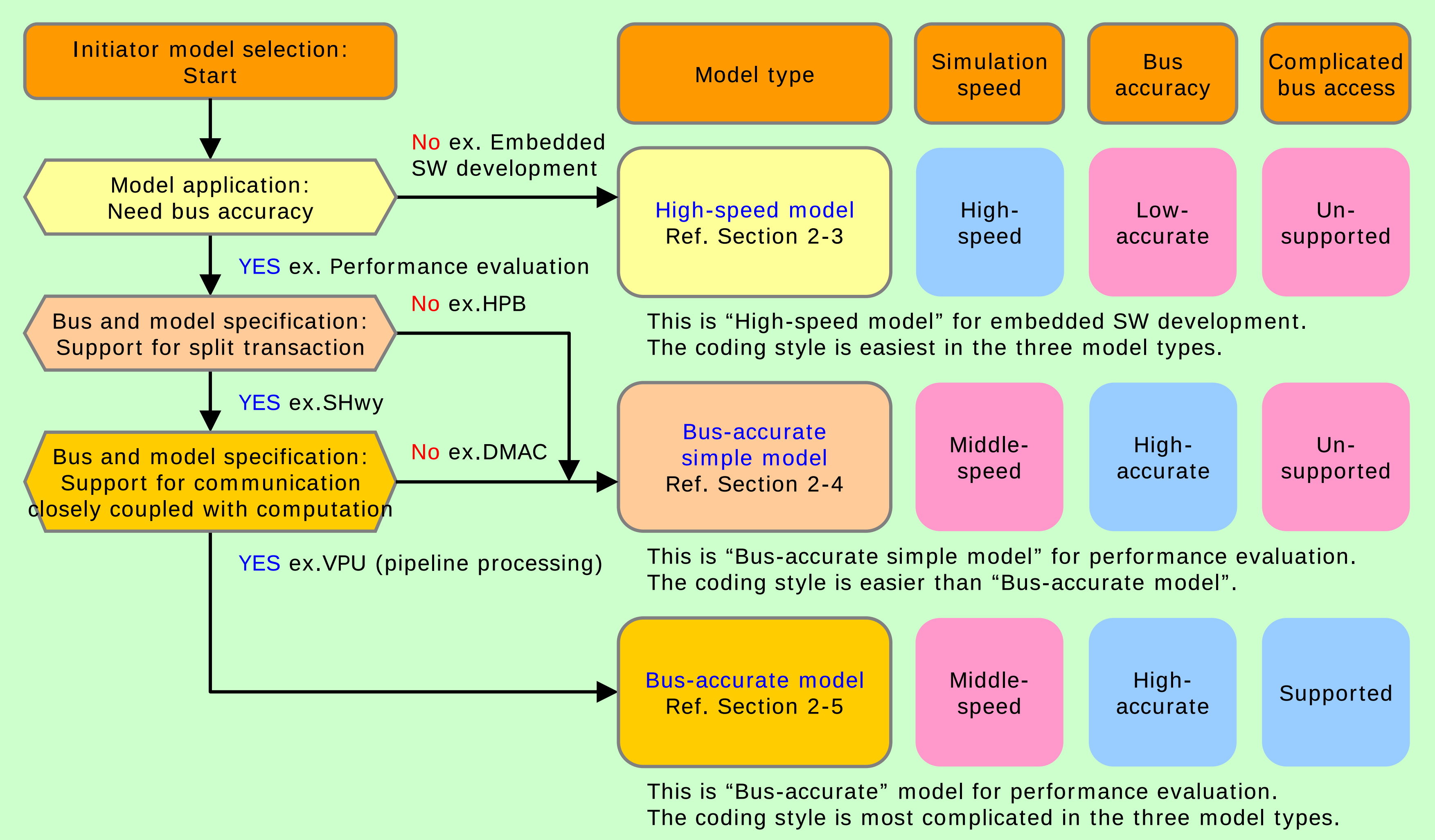
1. Core class inherit TLM common class to use communication behavior. Use initiator (target) I/F to describe initiator (target) model.
2. Implement computation behavior in core class.
3. Use (call) parameter control APIs to set parameters (delay time etc.).
4. Use (call) communication request APIs to communicate with the other models.
5. Implement (override) resource access APIs in core class depending on the model type (details in the following slides).



Model structure based on TLM common

2-2-4. Classification of Initiator Models

- This subsection shows classification of initiator models and reference sections.



Initiator model selection: Start

Model application: Need bus accuracy

No ex. Embedded SW development → High-speed model

YES ex. Performance evaluation

Bus and model specification: Support for split transaction

No ex.HPB → Bus-accurate simple model

YES ex.SHwy

Bus and model specification: Support for communication closely coupled with computation

No ex.DMAC → Bus-accurate simple model

YES ex.VPU (pipeline processing) → Bus-accurate model

Model type	Simulation speed	Bus accuracy	Complicated bus access
High-speed model Ref. Section 2-3	High-speed	Low-accurate	Un-supported
Bus-accurate simple model Ref. Section 2-4	Middle-speed	High-accurate	Un-supported
Bus-accurate model Ref. Section 2-5	Middle-speed	High-accurate	Supported

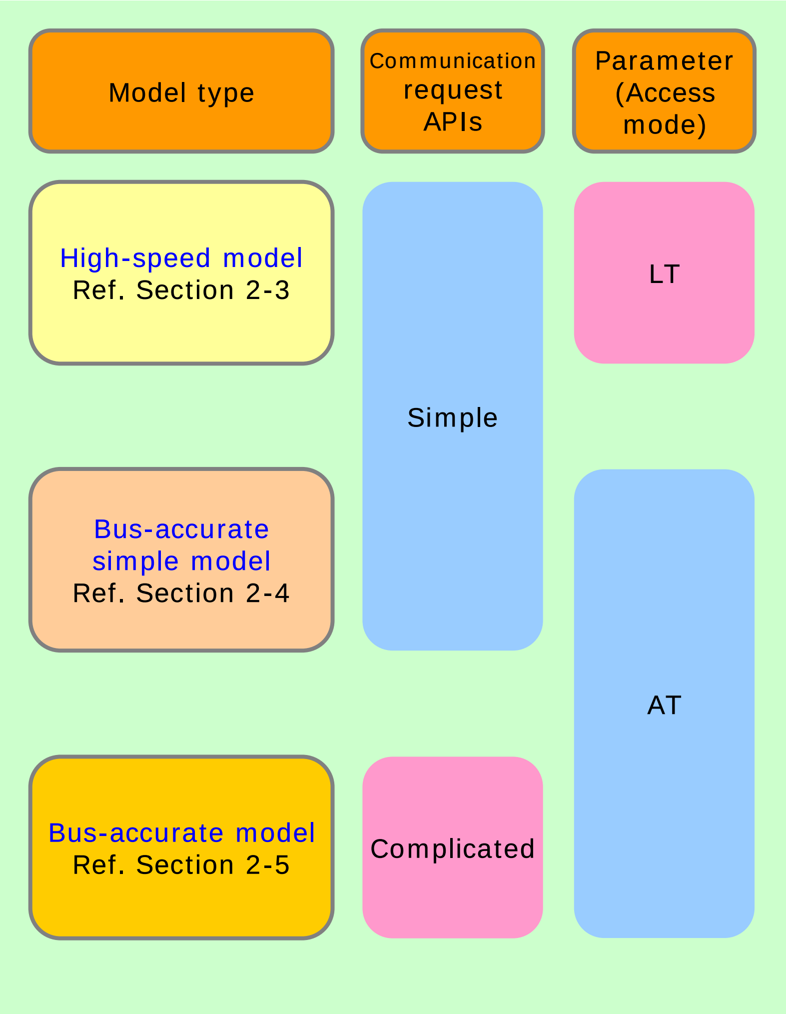
This is “High-speed model” for embedded SW development.
The coding style is easiest in the three model types.

This is “Bus-accurate simple model” for performance evaluation.
The coding style is easier than “Bus-accurate model”.

This is “Bus-accurate” model for performance evaluation.
The coding style is most complicated in the three model types.

2-2-5. Difference of Initiator Models

- This subsection shows difference of initiator models.



Model type	Communication request APIs	Parameter (Access mode)
High-speed model Ref. Section 2-3	Simple	LT
Bus-accurate simple model Ref. Section 2-4		AT
Bus-accurate model Ref. Section 2-5	Complicated	

- **High-speed vs. Bus-accurate simple**
  - APIs : Same
  - Parameter : Different

=> Common source code is available in the both models. Dynamic model type change is possible by switching parameter.

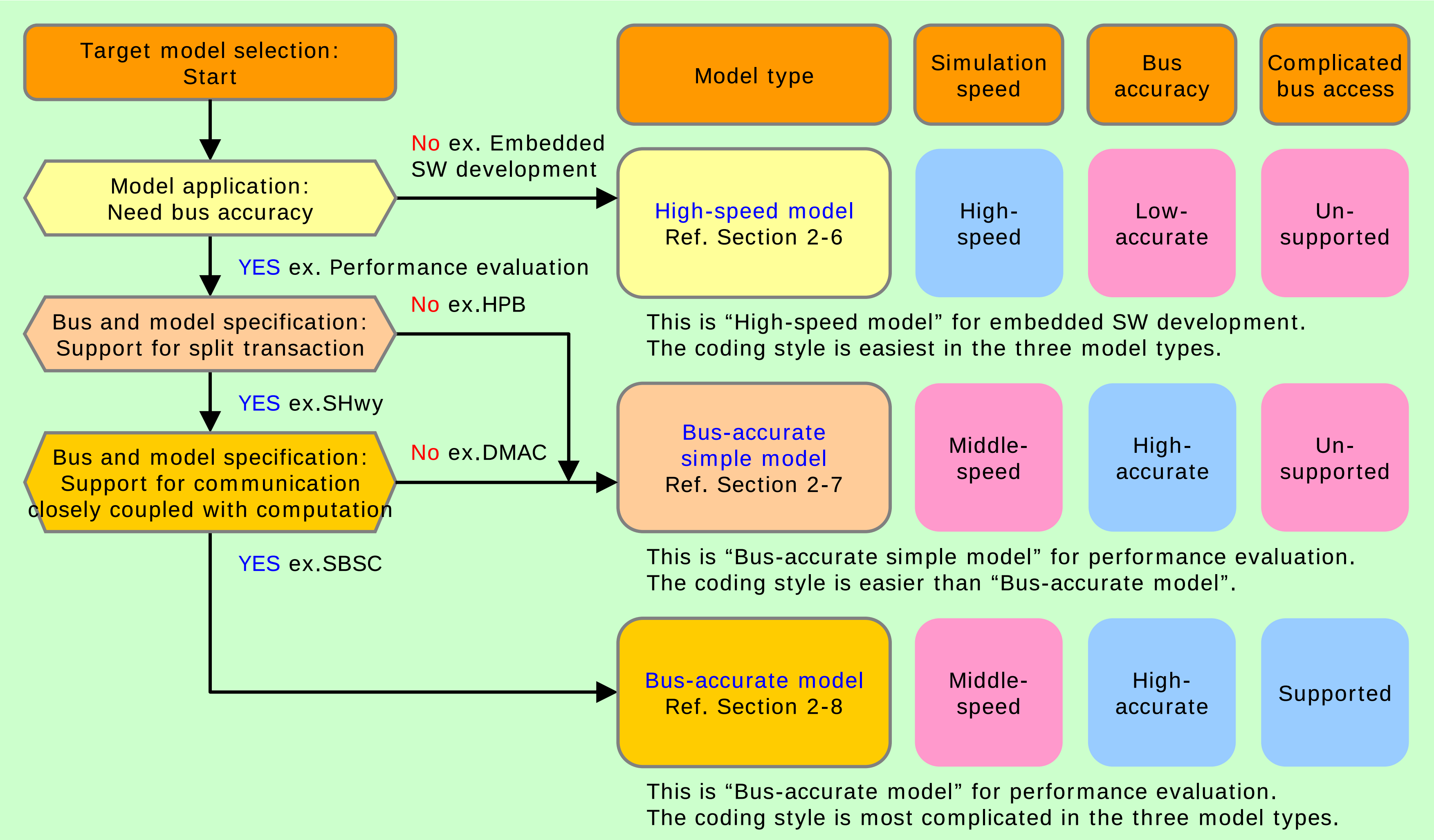
- **High-speed vs. Bus-accurate**
  - APIs : Different
  - Parameter : Different

=> Dynamic model type change is possible by source code description which switches API calls depending on parameter setting.

- **Bus-accurate simple vs. Bus-accurate**
  - APIs : Different
  - Parameter : Same

2-2-6. Classification of Target Models

- This subsection shows classification of target models and reference sections.



Target model selection: Start

Model application: Need bus accuracy

No ex. Embedded SW development → High-speed model

YES ex. Performance evaluation

Bus and model specification: Support for split transaction

No ex.HPB → Bus-accurate simple model

YES ex.SHwy

Bus and model specification: Support for communication closely coupled with computation

No ex.DMAC → Bus-accurate simple model

YES ex.SBSC → Bus-accurate model

Model type	Simulation speed	Bus accuracy	Complicated bus access
High-speed model Ref. Section 2-6	High-speed	Low-accurate	Un-supported
Bus-accurate simple model Ref. Section 2-7	Middle-speed	High-accurate	Un-supported
Bus-accurate model Ref. Section 2-8	Middle-speed	High-accurate	Supported

This is “High-speed model” for embedded SW development.
The coding style is easiest in the three model types.

This is “Bus-accurate simple model” for performance evaluation.
The coding style is easier than “Bus-accurate model”.

This is “Bus-accurate model” for performance evaluation.
The coding style is most complicated in the three model types.

2-2-7. Difference of Target Models

- This subsection shows difference of target models.

Model type	Communication request APIs	Parameter (Phase mode)
High-speed model Ref. Section 2-6	Simple	Single
Bus-accurate simple model Ref. Section 2-7		Multi
Bus-accurate model Ref. Section 2-8	Complex	

- **High-speed vs. Bus-accurate simple**
  - APIs : Same
  - Parameter : Different

=> Common source code is available in the both models. Dynamic model type change is possible by switching parameter.

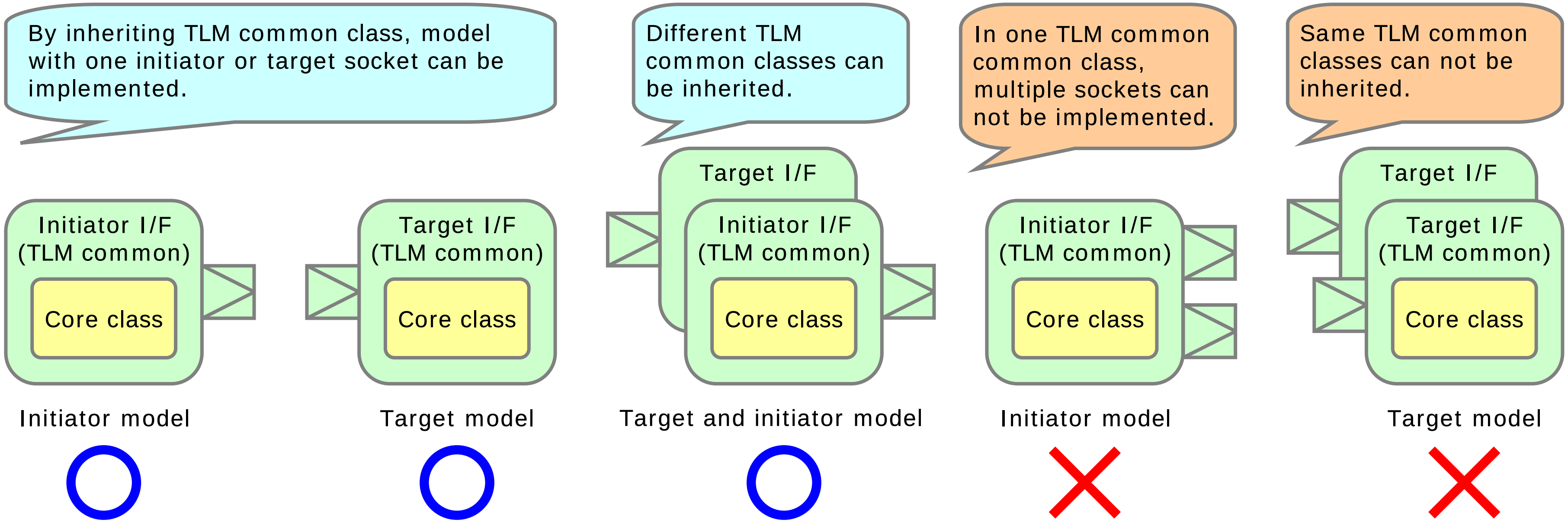
- **High-speed vs. Bus-accurate**
  - APIs : Different
  - Parameter : Different

=> Dynamic model type change is possible by source code description which switches API calls depending on parameter setting.

- **Bus-accurate simple vs. Bus-accurate**
  - APIs : Different
  - Parameter : Same

2-2-8. Restrictions

- This subsection explains restrictions of using TLM common class.
- Models with multiple initiator sockets or target sockets are not supported. For example, a model with two initiator sockets can not be implemented. However, a model with one initiator socket and one target socket can be implemented.
- It is assumed that bus-accurate simple model and bus-accurate model of initiator or target are connected with "SHwy bus", so timing annotation is considered based on specification of "SHwy bus". Models based on TLM common class are able to connect with the other bus model or directly with initiator or target model, but appropriate timing may not be annotated.



2-3. Initiator High-speed Model (Contents)

2-3-1. Initiator High-speed Model: Outline

- **Initiator high-speed model outline** - details explained in the succeeding slides

Initiator Model Behaviors

No.	Behavior
1	Communication sequence
2	Log output
3	Timing annotation
4	Transaction extensions

Initiator Model Parameters

No.	Parameter	Name
1	Name of core instance	name
2	Bus width	bus_width
3	Access mode *	access_mode
4	Source id	src_id
5	File pointer of log output	p_log_file
6	Switch of write access log	wr_log
7	Switch of read access log	rd_log

Parameter Control APIs

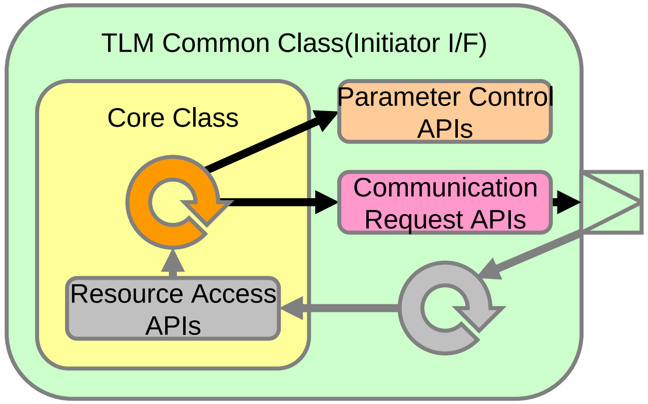
No.	Function	Name
1	Initialize parameter	ini_init_param
2	Set parameter	ini_set_param
3	Get parameter	ini_get_param

Communication Request APIs

No.	Function	Name
1	Write request	ini_wr
2	Read request	ini_rd

Resource Access APIs

No.	Function	Name
-	-	-

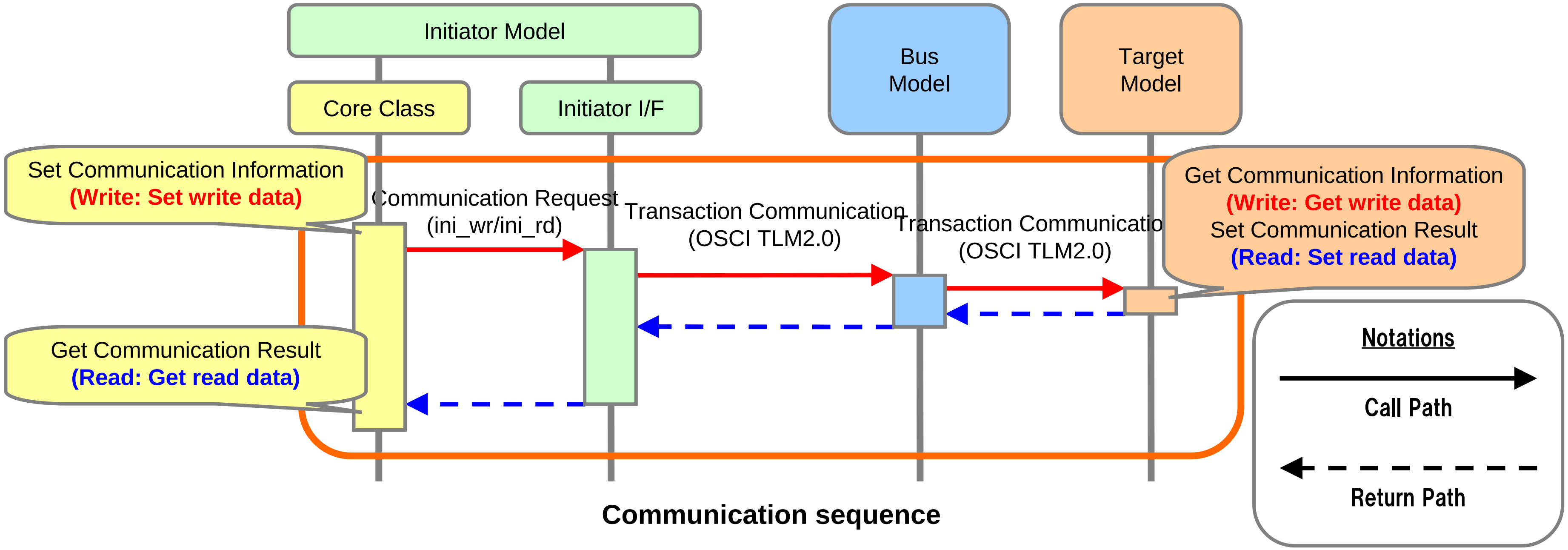


Model Block Diagram

- * Access mode parameter determines initiator model type.

2-3-2.(a) Behavior: Communication Sequence

- Initiator high-speed model behavior: communication sequence
- A single communication request call (ini_wr or ini_rd) completes a transaction.
  -> [Reference] 2-3-4.(d) API:Write, 2-3-4(e) API:Read
- This sequence cannot model complicated bus transaction such as split-transaction or outstanding. Timing accuracy is not guaranteed for such transactions.
- Once an initiator calls a request, the calling process can do nothing but just wait till the end of the transaction.

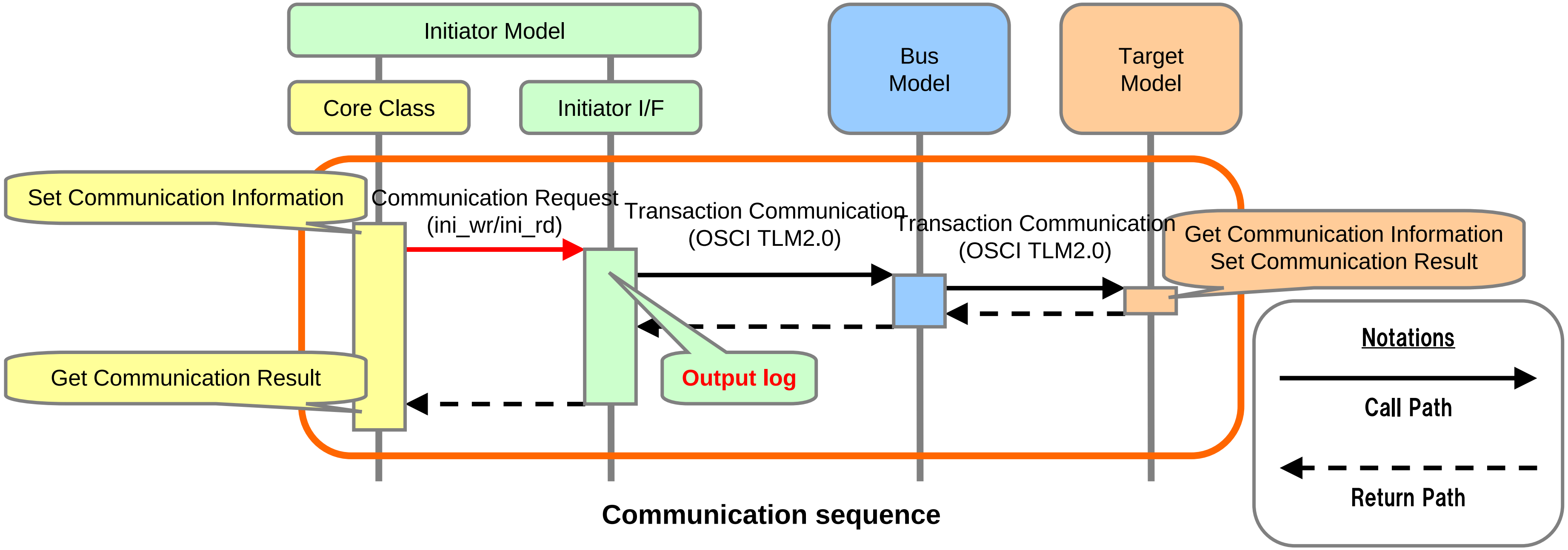


Communication sequence

2-3-2.(b) Behavior: Log Output(1/2)

- Initiator high-speed model behavior: log output
- Output transaction log after every communication request call (ini_wr/ini_rd).
- Disable log output to avoid simulation slowdown except for test/debug purpose.
- Set parameters (wr_log, rd_log) to enable log output.

  -> [Reference] 2-3-3.(a) Structure:Parameters, 2-3-4.(b) API:Set parameters



Communication sequence

2-3-2.(b) Behavior: Log Output(2/2)

■ Initiator high-speed model behavior: log output format

Log output format and example

Item	Simulation time	[Instance]	I(source ID)-> T	Command	(Address)	Data
ex1	213 ns :	[ini_mod]	I(42)-> T	WR	(00001000)	01234567 89ABCDEF
note	At simulation time 213[ns], initiator (instance "ini_mod", source ID=0x42) requests a transaction (Write, addr=0x00001000, data_in_big_endian=01234567_89ABCDEF). (*)					
ex2	593 ns :	[ini_mod]	I(42)-> T	RD	(00001000)	-------- --------
note	At simulation time 593[ns], initiator (instance "ini_mod", source ID=0x42) requests a transaction (Read, addr=0x00001000).					

(*) data is displayed in big-endian format, even though data is stored in little endian by OSCI TLM2 specification.

[1-byte word size] -> data = 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF
[2-byte word size] -> data = 0x2301, 0x6745, 0xAB89, 0xEFCD
[4-byte word size] -> data = 0x67452301, 0xEFCDAB89
[8-byte word size] -> data = 0xEFCDAB89_67452301

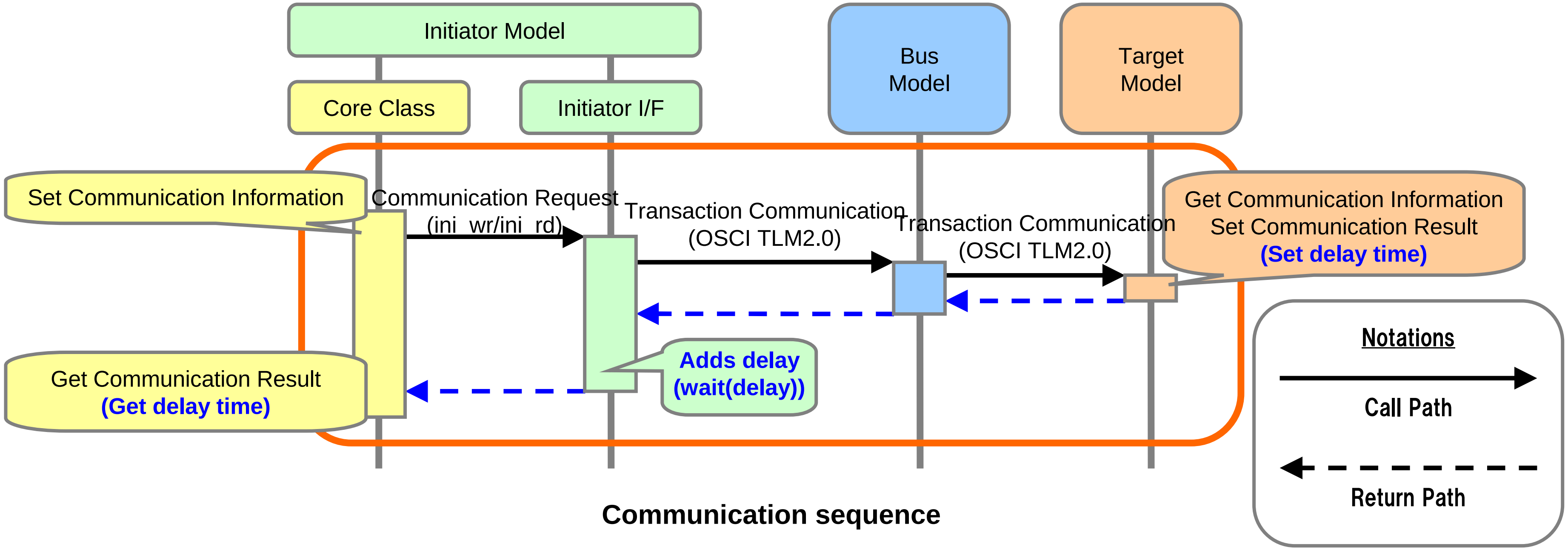
Log output file example

```
0 s :       [ini_mod ]   I(42)-> T   WR    (00000000)   11000000
105 ns :    [ini_mod ]   I(42)-> T   WR    (00000004)   22000000 33000000
210 ns :    [ini_mod ]   I(42)-> T   WR    (0000000C)   44000000 55000000 66000000 77000000
530 ns :    [ini_mod ]   I(42)-> T   RD    (0000000C)   -------- -------- -------- --------
640 ns :    [ini_mod ]   I(42)-> T   RD    (00000004)   -------- --------
750 ns :    [ini_mod ]   I(42)-> T   RD    (00000000)   --------
```

2-3-2.(c) Behavior: Timing Annotation

- Initiator high-speed model behavior: timing annotation
- Target model sets delay time information.
- Bus model or common initiator I/F adds delay; core class does not have to.
- Initiator model can get delay time information from API argument p_ext.

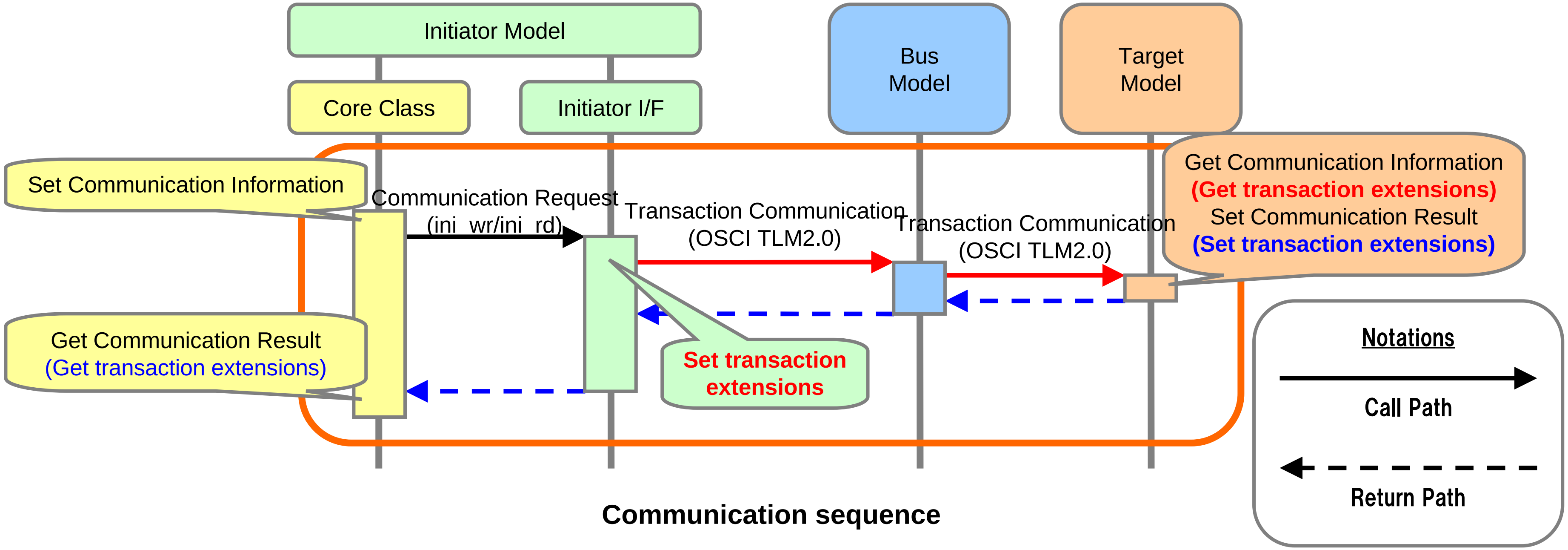
  -> [Reference] 2-3-4.(d) API:Write, 2-3-4.(e) API:Read



Communication sequence

2-3-2.(d) Behavior: Transaction extensions

- Initiator high-speed model behavior: transaction extensions
- Transaction extensions can carry additional information other than basic transaction information such as address/data/size. Use/no-use is optional.
- Common initiator I/F automatically sets its initiator I/F parameters to transaction extensions. Core class does not have to set.
- Target(initiator) model can get initiator(target) I/F parameters. -> [Reference] 2-3-3.(b) Structure:Transaction extensions, 2-3-4.(d) API:Write, 2-3-4.(e) API:Read



Communication sequence

2-3-3.(a) Structure: Parameters

Name	Structure for initiator I/F parameters	Structure name	vpcl::tlm_if_ini_parameter

No.	Parameter	Data type	Variable	Value	Description
1	Name of core instance	std::string	name	(fixed to constructor argument)	*1
2	Bus width	unsigned int	bus_width	(fixed to template argument)	*2
3	Access mode	vpcl::tlm_if_access	access_mode	vpcl::TLM_IF_LT_ACCESS(default) vpcl::TLM_IF_AT_ACCESS	High-speed access *3 Bus-accurate access(Prohibited)
4	Source ID	unsigned int	src_id	0(default) non-negative integer	-
5	File pointer of log output	FILE *	p_log_file	NULL stdout(default) file pointer	No log output *4 Output to stdout Output to file
6	Switch of write access log	bool	wr_log	false(default) true	Not output Output
7	Switch of read access log	bool	rd_log	false(default) true	Not output Output

- *1 Read only: fixed to constructor argument
- *2 Read only: fixed to template argument
- *3 Use vpcl::TLM_IF_LT_ACCESS for initiator high-speed model
- *4 wr_log and rd_log are ignored when p_log_file is set to NULL

2-3-3.(b) Structure: Transaction extensions

Name	Structure for transaction extensions	Structure name	vpcl::tlm_if_extension

No.	Structure member	Data type	Variable	Value	Description
1	Initiator I/F use	bool	ini_if_use	false(default) true	Common initiator I/F not used Common initiator I/F used *5
2	Initiator I/F parameters	vpcl::tlm_if_ini_parameter	ini_if_param	(See 2-3-3.(a))	(See 2-3-3.(a))
3	Target I/F use	bool	tgt_if_use	false(default) true	Common target I/F not used *6 Common target I/F used
4	Target I/F parameters	vpcl::tlm_if_tgt_parameter	tgt_if_param	(See 2-6-3.(a))	(See 2-6-3.(a)) *7
5	Pointer to user extension	void *	p_user_ext	NULL(default) pointer to user extension	No user extension User extension *8

- *5 Automatically set to true by common initiator I/F.
- *6 Automatically set to true by target model if it uses common target I/F; false otherwise.
- *7 tgt_if_param data is invalid when tgt_if_use is false
- *8 Arbitrary data type can be used as user extension, as long as both initiator and target use same data type.

2-3-4.(a) API: Initialize parameters

Name	Initialize initiator I/F parameters	API name	ini_init_param

API	void ini_init_param(void)		
Arguments	void	-	-
Return	void	-	-
Description	Initialize initiator I/F parameters to default values		
Steps	1. Call this API without any argument		
Notes	1. See 2-3-3.(a) for initial values		
Example	this->ini_init_param(void); // step 1		

2-3-4.(b) API: Set parameters

Name	Set initiator I/F parameters	API name	ini_set_param

API	bool ini_set_param(vpcl::tlm_if_ini_parameter *p_param)		
Arguments	vpcl::tlm_if_ini_parameter *p_param	In	Initiator I/F parameters to be set
Return	true	Out	Succeeded
	false	Out	Failed
Description	Set initiator I/F parameters		
Steps	1. Allocate an instance of initiator I/F parameters (vpcl::tlm_if_ini_parameter) 2. Set appropriate values to parameters in the instance 3. Call this API with pointer to the instance 4. Check status by return value if necessary		
Notes	1. Allocate an instance of initiator I/F parameters before calling this API 2. Get parameters and modify only ones to be updated, and call this API 3. Name of core instance and bus width cannot be modified		

Example	`vpcl::tlm_if_ini_parameter param;  // step 1, note 1`
	`this->ini_get_param(&param);  // note 2`
	`param.access_mode = vpcl::TLM_IF_LT_ACCESS;  // step 2`
	`param.src_id = 0x42;  // step 2`
	`param.p_log_file = stdout;  // step 2`
	`param.wr_log = true;  // step 2`
	`param.rd_log = true;  // step 2`
	`this->ini_set_param(&param);  // step 3`		

2-3-4.(c) API: Get parameters

Name	Get initiator I/F parameters	API name	ini_get_param

API	bool ini_get_param(vpcl::tlm_if_ini_parameter *p_param)		
Arguments	vpcl::tlm_if_ini_parameter *p_param	Out	Initiator I/F parameters
Return	true	Out	Succeeded
	false	Out	Failed
Description	Get initiator I/F parameters		
Steps	1. Allocate an instance of initiator I/F parameters (vpcl::tlm_if_ini_parameter) 2. Call this API with pointer to the instance 3. Get initiator I/F parameters from p_param 4. Check status by return value if necessary		
Notes	1. Allocate an instance of initiator I/F parameters before calling this API		

Example	vpcl::tlm_if_ini_parameter param; // step 1, note 1
	this->ini_get_param(&param); // step 2		

2-3-4.(d) API: Write

Name	Write	API name	ini_wr

API	bool ini_wr(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext = NULL)		
Arguments	unsigned int addr	In	Write address
	unsigned char *p_data	In	Write data (pointer)
	unsigned int size	In	Write size [byte]
	vpcl::tlm_if_extension *p_ext	Out	Transaction extensions (pointer)
Return	true	Out	Succeeded
	false	Out	Failed
Description	Issue a write request to target		
Steps	1. Allocate an array for write data (unsigned char []) and an instance of transaction extensions(vpcl::tlm_if_extension), and then set write data to the array 2. Set arguments(addr, p_data, size, p_ext) with write access request and call this API 3. Check status by return value if necessary 4. Get transaction extension information (such as target I/F parameters) from p_ext if necessary		
Notes	1. Allocate the data array and the transaction extensions instance before calling this API 2. Argument p_ext is possible to omit if it is not necessary 3. Target I/F parameters(tgt_if_param) in transaction extensions is available only if target uses common target I/F		

Example	unsigned char data[4]; // step 1, note 1
	vpcl::tlm_if_extension ext; // step 1, note 1
	memset(data, 0xFF, 4); // step 1
	this->ini_wr(0x1000, data, 4, &ext); // step 2		

2-3-4.(e) API: Read

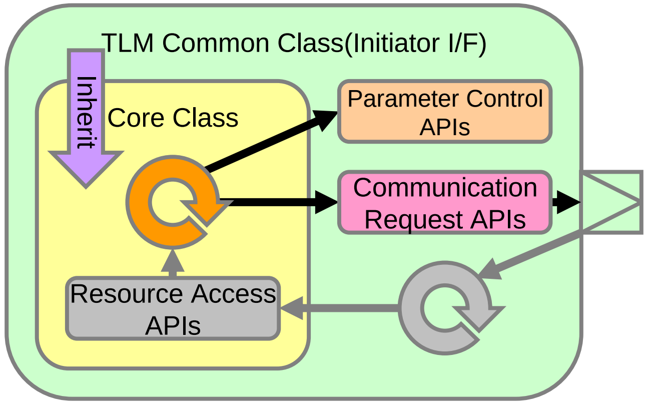
Name	Read	API name	ini_rd

API	bool ini_rd(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext = NULL)		
Arguments	unsigned int addr	In	Read address
	unsigned char *p_data	Out	Read data (pointer)
	unsigned int size	In	Read size [byte]
	vpcl::tlm_if_extension *p_ext	Out	Transaction extensions (pointer)
Return	true	Out	Succeeded
	false	Out	Failed
Description	Issue a read request to target		
Steps	1. Allocate an array for read data (unsigned char []) and an instance of transaction extensions(vpcl::tlm_if_extension) 2. Set arguments(addr, size, p_ext) with read access request and call this API 3. Check status by return value if necessary 4. Get read data from p_data 5. Get transaction extension information (such as target I/F parameters) from p_ext if necessary		
Notes	1. Allocate the data array and the transaction extensions instance before calling this API 2. Argument p_ext is possible to omit if it is not necessary 3. Target I/F parameter(tgt_if_param) in transaction extensions is available only if target uses common target I/F		

Example	`unsigned char p_data[4];  // step 1, note 1`
	`vpcl::tlm_if_extension ext;  // step 1, note 1`
	`this->ini_rd(addr, p_data, size, &ext);  // step 2`		

2-3-5.(a) Modeling: Classes and Files

- Initiator high-speed model: class and files
- This model consists of TLM common class(initiator I/F class) and core class, and the core class inherits the initiator I/F class.
- Use prepared library(tlm_if.h, tlm_ini_if.h) without any modification for initiator I/F class.
- Develop only core class descriptions.



Model Block Diagram

Classes and Files of Initiator Model

Class	Class name	Filename	Notes
Initiator I/F class	tlm_ini_if	tlm_if.h *1 tlm_ini_if.h	Use prepared library
Core class *2	Arbitrary (example) ini_module	Arbitrary (example) ini_module.h ini_module.cpp	Develop this

- *1 Core class does not have to explicitly include "tlm_if.h" as it is included in "tlm_ini_if.h"
- *2 Core class can use arbitrary class name and filenames

2-3-5.(b) Modeling: Descriptions

- Initiator high-speed model: descriptions
- Procedures to use common initiator I/F (No.1-3)
- Initializations of common initiator I/F (No.4-7)
- Communication request API calls (No.8)
- See section 2-9 for an initiator model sample.

Initiator high-speed model descriptions

No.	Level	File	Description
1	MUST	ini_module.h	Include header file of common initiator I/F (tlm_ini_if.h) at the top of core class header file (ini_module.h)
2	MUST	ini_module.h	Inherit common initiator I/F class (tlm_ini_if) in core class declaration
3	MUST	ini_module.h	Set instance name of core class and source ID(optional) as constructor arguments of common initiator I/F class
4	Option	ini_module.h	Set initiator I/F parameters using parameter control APIs in core class constructor; Be sure to describe "this->" before the API name
5	Option	ini_module.cpp or higher module	Call parameter control APIs in initiator core class or higher hierarchical level module which instantiates initiator class such as testbench
6	MUST	higher module	Instantiate initiator with core instance name and bus width in higher hierarchical level module
7	MUST	higher module	Connect socket of common initiator I/F class (m_ini_socket) to target socket
8	MUST	ini_module.cpp	Call communication request APIs in initiator core class to issue communication request to target; Be sure to describe "this->" before the API name -> [Reference] 2-3-4.(d) API: Write, 2-3-4.(e) API: Read

2-6. Target High-speed Model (Contents)

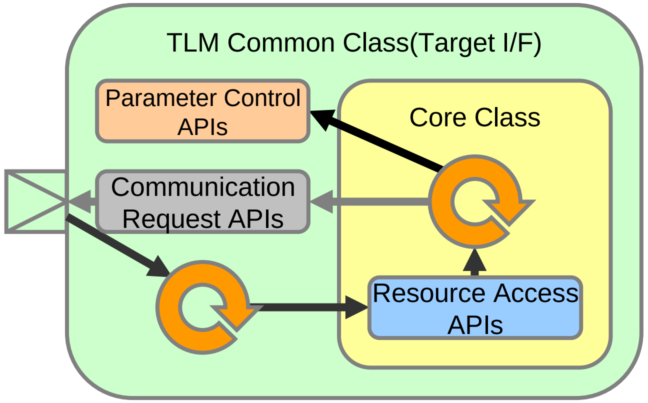
- 2-6-1. Outline
- 2-6-2. Behaviors
  - (a)Communication sequence
  - (b)Log output
  - (c)Timing annotation
  - (d)Transaction extensions
- 2-6-3. Structures
  - (a)Parameters
  - (b)Transaction extensions
- 2-6-4. Functions
  - (a)Initialize parameters
  - (b)Set parameters
  - (c)Get parameters
  - (d)Write CB
  - (e)Read CB
  - (f)DebugWrite CB
  - (g)DebugRead CB
- 2-6-5. Modeling
  - (a)Classes and Files
  - (b)Descriptions

2-6-1. Target High-speed Model: Outline

- Target high-speed model outline - details explained in the succeeding slides

Target Model Behaviors

No.	Behavior
1	Communication sequence
2	Log output
3	Timing annotation
4	Transaction extensions



Model Block Diagram

Target Model Parameters

No.	Parameter	Name
1	Name of core instance	name
2	Bus width	bus_width
3	Phase mode *1	phase_mode
4	Write latency	wr_latency
5	Read latency	rd_latency
6	File pointer of log output	p_log_file
7	Switch of write access log	wr_log
8	Switch of read access log	rd_log

Parameter Control APIs

No.	Function	Name
1	Initialize parameter	tgt_init_param
2	Set parameter	tgt_set_param
3	Get parameter	tgt_get_param

Communication Request APIs

No.	Function	Name
-	-	-

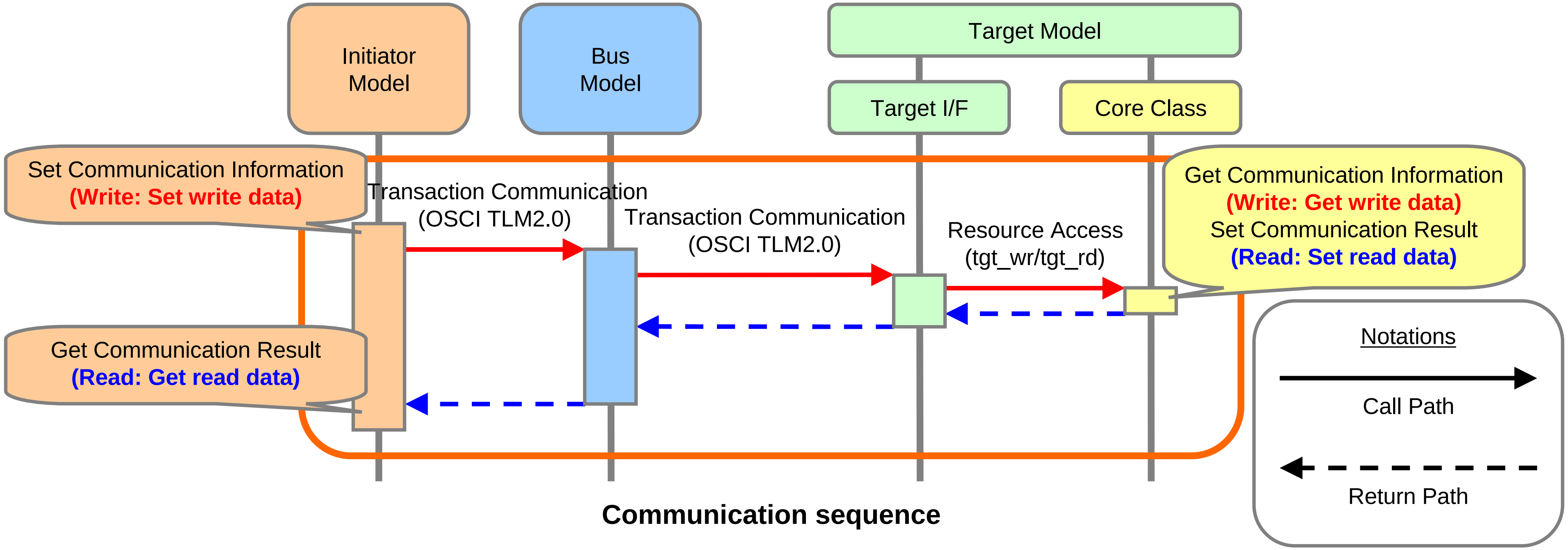
Resource Access APIs *2

No.	Function	Name
1	Write CB	tgt_wr
2	Read CB	tgt_rd
3	DebugWrite CB	tgt_wr_dbg
4	DebugRead CB	tgt_rd_dbg

- *1 Phase mode parameter determines target model type. Common target I/F checks this parameter and automatically sets communication mode with initiator.
- *2 All resource access APIs are callback functions which should be implemented in the core class.

2-6-2.(a) Behavior: Communication Sequence

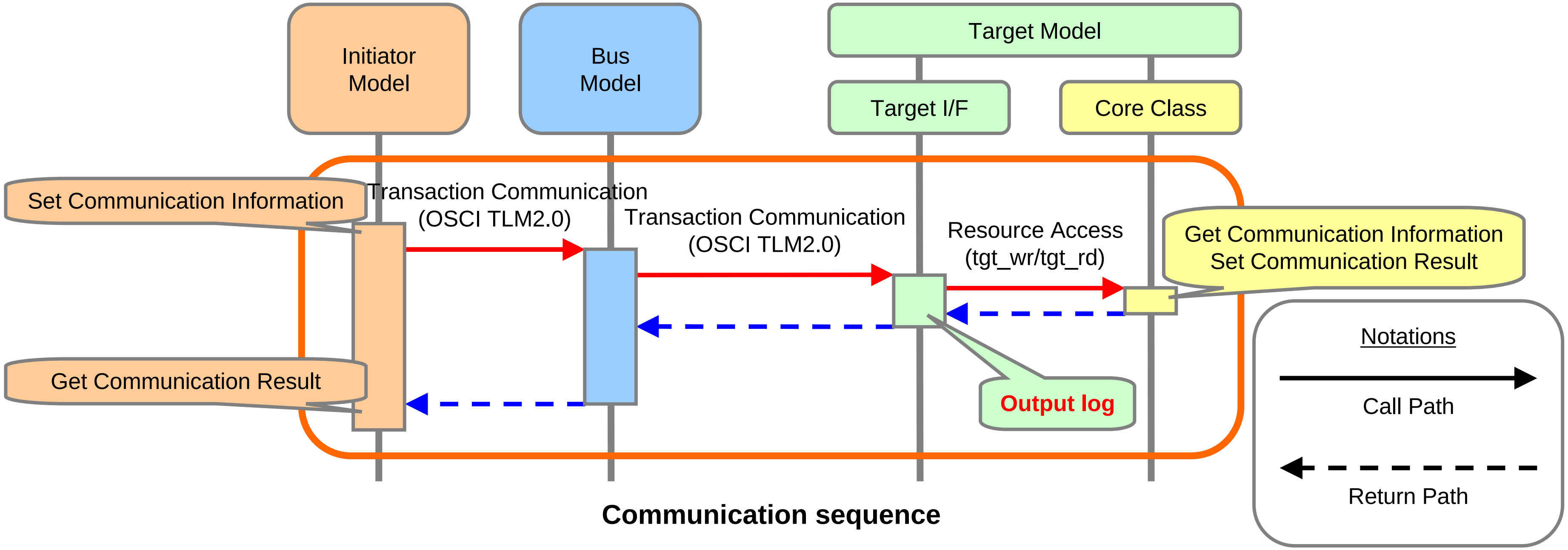
- Target high-speed model behavior: communication sequence
- A single resource access callback (tgt_wr/tgt_rd) implements a transaction.
  -> [Reference] 2-6-4.(f) API:Write CB, 2-6-4.(g) API:Read CB
- This sequence cannot model complicated bus transaction such as split-transaction or outstanding. Timing accuracy is not guaranteed for such transactions.



Communication sequence

2-6-2.(b) Behavior: Log Output(1/2)

- Target high-speed model behavior: log output
- Output transaction log after every resource access call (write/read).
- Set parameters (wr_log, rd_log) to enable log output.
  -> [Reference] 2-6-3.(a) Structure:Parameters, 2-6-4.(b) API:Set parameters



Communication sequence

2-6-2.(b) Behavior: Log Output(2/2)

Target high-speed model behavior: log output format

Log output format and example

Item	Simulation time	[Instance]	I(source ID)<- T	Command	(Address)	Data
ex1	213 ns :	[tgt_mod]	I(42)<- T	WR	(00001000)	01234567 89ABCDEF
note	At simulation time 213[ns], target (instance "tgt_mod") processes a transaction (Write, addr=0x00001000, data_in_big_endian=01234567_89ABCDEF). (*)					
ex2	593 ns :	[tgt_mod]	I(42)<- T	RD	(00001000)	01234567 89ABCDEF
note	At simulation time 593[ns], target (instance "tgt_mod") processes a transaction (Read, addr=0x00001000, data_in_big_endian=01234567_89ABCDEF). (*)					

(*) data is displayed in big-endian format, even though data is stored in little endian by OSCI TLM2 specification.

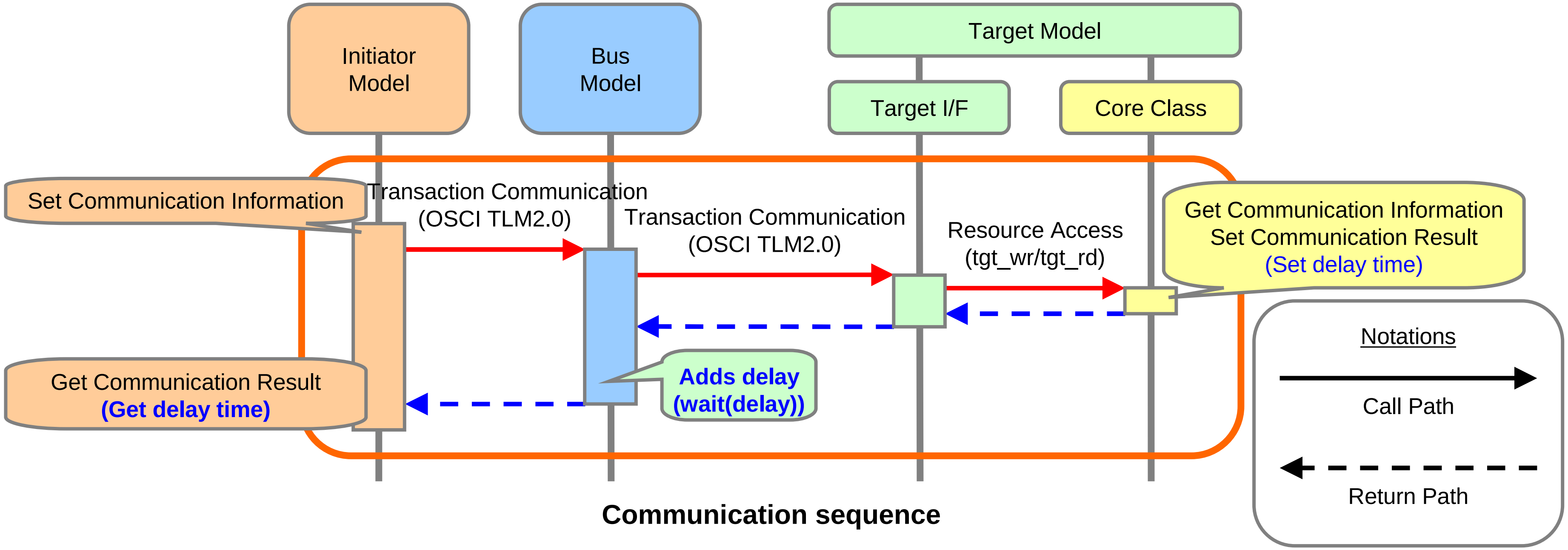
[1-byte word size] -> data = 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF
[2-byte word size] -> data = 0x2301, 0x6745, 0xAB89, 0xEFCD
[4-byte word size] -> data = 0x67452301, 0xEFCDAB89
[8-byte word size] -> data = 0xEFCDAB89_67452301

Log output file example

```
0 s :       [tgt_mod ]    I(42)<- T   WR   (00000000)    11000000
105 ns :    [tgt_mod ]    I(42)<- T   WR   (00000004)    22000000 33000000
210 ns :    [tgt_mod ]    I(42)<- T   WR   (0000000C)    44000000 55000000 66000000 77000000
530 ns :    [tgt_mod ]    I(42)<- T   RD   (0000000C)    44000000 55000000 66000000 77000000
640 ns :    [tgt_mod ]    I(42)<- T   RD   (00000004)    22000000 33000000
750 ns :    [tgt_mod ]    I(42)<- T   RD   (00000000)    11000000
```

2-6-2.(c) Behavior: Timing Annotation

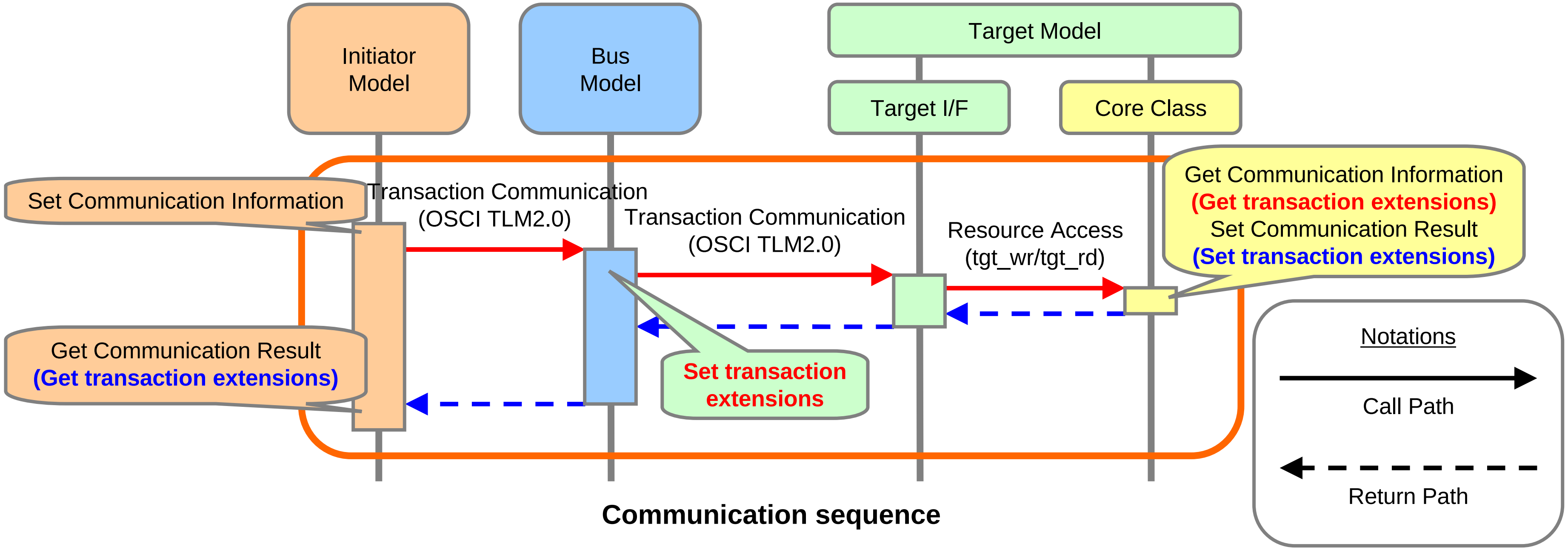
- Target high-speed model behavior: timing annotation
- Common target I/F automatically sets delay time information.
- Delay time values are defined by parameters(wr_latency, rd_latency) in target model.
  -> [Reference] 2-6-3.(a) Structure:Parameters, 2-6-4.(g) API:Set parameters
- Bus model or initiator model adds delay; see specifications of bus model and initiator model to know which model does.



Communication sequence

2-6-2.(d) Behavior: Transaction extensions

- Target high-speed model behavior: transaction extensions
- Transaction extensions can carry additional information other than basic transaction information such as address/data/size. Use/no-use is optional.
- Common target I/F automatically sets its target I/F parameters to transactions. Core class does not have to set.
- Initiator(target) model can get parameters set by target(initiator).
  -> [Reference] 2-6-3.(b) Structure:Transaction extensions, 2-6-4.(d) API:Write CB, 2-6-4.(e) API:Read CB



Communication sequence

2-6-3.(a) Structure: Parameters

Name	Structure for target I/F parameters	Structure name	vpcl::tlm_if_tgt_parameter

No.	Parameter	Data type	Variable	Value	Description
1	Name of core instance	std::string	name	(fixed to constructor argument)	*1
2	Bus width	unsigned int	bus_width	(fixed to template argument)	*2
3	Phase mode	vpcl::tlm_if_phase	phase_mode	vpcl::TLM_IF_SINGLE_PHASE(default) vpcl::TLM_IF_MULTI_PHASE	Single phase *3 Multi phase(Prohibited)
4	Write latency -> [Reference] 2-6-4.(d)	sc_time	wr_latency	SC_ZERO_SEC(default) non-negative number	0[sec] latency
5	Read latency -> [Reference] 2-6-4.(e)	sc_time	rd_latency	SC_ZERO_SEC(default) non-negative number	0[sec] latency
6	File pointer of log output	FILE *	p_log_file	NULL stdout(default) file pointer	No log output *4 Output to stdout Output to file
7	Switch of write access log	bool	wr_log	false(default) true	Not output Output
8	Switch of read access log	bool	rd_log	false(default) true	Not output Output

*1 Read only: fixed to constructor argument
*2 Read only: fixed to template argument
*3 Use vpcl::TLM_IF_SINGLE_PHASE for target high-speed model
*4 wr_log and rd_log are ignored when p_log_file is set to NULL

2-6-3.(b) Structure: Transaction extensions

Name	Structure for transaction extensions	Structure name	vpcl::tlm_if_extension

No.	Structure member	Data type	Variable	Value	Description
1	Initiator I/F use	bool	ini_if_use	false(default) true	Common initiator I/F not used *5 Common initiator I/F used
2	Initiator I/F parameters	vpcl::tlm_if_ini_parameter	ini_if_param	(See 2-3-3.(a))	(See 2-3-3.(a)) *6
3	Target I/F use	bool	tgt_if_use	false(default) true	Common target I/F not used Common target I/F used *7
4	Target I/F parameters	vpcl::tlm_if_tgt_parameter	tgt_if_param	(See 2-6-3.(a))	(See 2-6-3.(a))
5	Pointer to user extension	void *	p_user_ext	NULL(default) pointer to user extension	No user extension User extension *8

- *5 Automatically set to true by initiator model if it uses common initiator I/F; false otherwise
- *6 ini_if_param data is invalid when ini_if_use is false.
- *7 Automatically set to true by common target I/F.
- *8 Arbitrary data type can be used as user extension, as long as both initiator and target use same data type.

2-6-4.(a) API: Initialize parameters

Name	Initialize target I/F parameters	API name	tgt_init_param

API	void tgt_init_param(void)		
Arguments	void	-	-
Return	void	-	-
Description	Initialize target I/F parameters to default values		
Steps	1. Call this API without any argument		
Notes	1. See 2-6-3.(a) for initial values		
Example	this->tgt_init_param(void); // step1		

2-6-4.(b) API: Set parameters

Name	Set target I/F parameters	API name	tgt_set_param

API	bool tgt_set_param(vpcl:tlm_if_tgt_parameter *p_param)		
Arguments	vpcl::tlm_if_tgt_parameter*p_param	In	Target I/F parameters to be set
Return	true	Out	Succeeded
	false	Out	Failed
Description	Set target I/F parameters		
Steps	1. Allocate an instance of target I/F parameters (vpcl::tlm_if_tgt_parameter) 2. Set appropriate values to parameters in the instance 3. Call this API with pointer to the instance 4. Check status by return value if necessary		
Notes	1. Allocate an instance of target I/F parameters before calling this API 2. Get parameters and modify only ones to be updated, and call this API 3. Name of core instance and bus width cannot be modified		

Example	vpcl::tlm_if_tgt_parameter param; // step 1, note 1
	this->tgt_get_param(&param); // note 2
	param.phase_mode = vpcl::TLM_IF_SINGLE_PHASE; // step 2
	param.wr_latency = sc_time(10, SC_NS); // step 2
	param.rd_latency = sc_time(10, SC_NS); // step 2
	param.p_log_file = stdout; // step 2
	param.wr_log = true; // step 2
	param.rd_log = true; // step 2
	this->tgt_set_param(&param); // step 3		

2-6-4.(c) API: Get parameters

Name	Get target I/F parameters	API name	tgt_get_param

API	bool tgt_get_param(vpcl::tlm_if_tgt_parameter *p_param)		
Arguments	vpcl::tlm_if_tgt_parameter *param	Out	Target I/F parameters
Return	true	Out	Succeeded
	false	Out	Failed
Description	Get target I/F parameters		
Steps	1. Allocate an instance of target I/F parameters (vpcl::tlm_if_tgt_parameter) 2. Call this API with pointer to the instance 3. Get target I/F parameters from p_param 4. Check status by return value if necessary		
Notes	1. Allocate an instance of target I/F parameters before calling this API		

Example	vpcl::tlm_if_tgt_parameter param; // step 1, note 1
	this->tgt_get_param(&param); // step 2		

2-6-4.(d) API: Write CB

Name	Write CB	API name	tgt_wr

API	bool tgt_wr(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time)		
Arguments	unsigned int addr	In	Write address
	unsigned char *p_data	In	Write data (pointer)
	unsigned int size	In	Write size [byte]
	vpcl::tlm_if_extension *p_ext	In	Transaction extensions (pointer)
	sc_time *p_time	In/Out	Write latency (pointer)
Return	true	Out	Succeeded
	false	Out	Failed
Description	Process a write request from initiator		
Steps	This API is a callback function called by common target I/F. 1. Extract write request information from arguments (addr, p_data, size, p_time) and transaction extensions from argument (p_ext), and process the write request 2. Set delay time information (p_time) if necessary (wr_latency value has been set to p_time by common target I/F) 3. Return true if this process succeeded, false if failed		
Notes	1. Initiator I/F parameters(ini_if_param) in transaction extensions is available only if initiator uses common initiator I/F 2. Do not use wait() statement in this API 3. Do not call initiator access in this API; invoke separate process which calls initiator access instead (see subsection 2-9-5 for example)		
Example			

```
bool tgt_module<BUSWIDTH>::
tgt_wr(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time) {
  memcpy(m_mem + addr&0xff, p_data, size);  // step 1
  return true;  // step 3
}
```

2-6-4.(e) API: Read CB

Name	Read CB	API name	tgt_rd

API	bool tgt_rd(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time)		
Arguments	unsigned int addr	In	Read address
	unsigned char *p_data	Out	Read data (pointer)
	unsigned int size	In	Read size [byte]
	vpcl::tlm_if_extension *p_ext	In	Transaction extensions (pointer)
	sc_time *p_time	In/Out	Read latency (pointer)
Return	true	Out	Succeeded
	false	Out	Failed
Description	Process a read request from initiator		
Steps	This API is a callback function called by common target I/F. 1. Extract read request information from arguments (addr, size, p_time) and transaction extensions from argument (p_ext), and process the read request 2. Set read data to argument (p_data) 3. Set delay time information (p_time) if necessary (rd_latency value has been set to p_time by common target I/F) 4. Return true if this process succeeded, false if failed		
Notes	1. Initiator I/F parameters(ini_if_param) in transaction extensions is available only if initiator uses common initiator I/F 2. Do not use wait() statement in this API 3. Do not call initiator access in this API; invoke separate process which calls initiator access instead (see subsection 2-9-5 for example)		
Example			

```
bool tgt_module<BUSWIDTH>::
tgt_rd(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time) {
  memcpy(p_data, m_mem + addr&0xff, size);  // step 1, 2
  return true;  // step 4
}
```

2-6-4.(f) API: DebugWrite CB

Name	DebugWrite CB	API name	tgt_wr_dbg

API	bool tgt_wr_dbg(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext)		
Arguments	unsigned int addr	In	Write address
	unsigned char *p_data	In	Write data (pointer)
	unsigned int size	In	Write size [byte]
	vpcl::tlm_if_extension *p_ext	In	Transaction extensions (pointer)
Return	true	Out	Succeeded
	false	Out	Failed
Description	Process a write request from debugger		
Steps	This API is a callback function called by common target I/F. 1. Extract write request information from arguments (addr, p_data, size) and transaction extensions from argument (p_ext), and process the write request 2. Return true if this process succeeded, false if failed		
Notes	1. Initiator I/F parameters(ini_if_param) in transaction extensions is available only if initiator uses common initiator I/F 2. This API is called only by debugger; do not describe target behavior other than data copy to memory or register 3. Do not use wait() statement in this API 4. Do not call initiator access in this API; invoke separate process which calls initiator access instead (see subsection 2-9-5 for example)		
Example			

Example:

```
bool tgt_module<BUSWIDTH>::
tgt_wr_dbg(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext) {
  memcpy(m_mem + addr&0xff, p_data, size);  // step 1
  return true;  // step 2
}
```

2-6-4.(g) API: DebugRead CB

Name	DebugRead CB	API name	tgt_rd_dbg

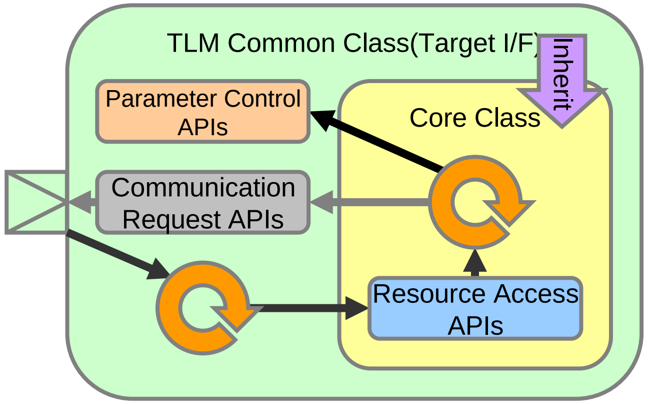
API	bool tgt_rd_dbg(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext)		
Arguments	unsigned int addr	In	Read address
	unsigned char *p_data	Out	Read data (pointer)
	unsigned int size	In	Read size [byte]
	vpcl::tlm_if_extension *p_ext	In	Transaction extensions (pointer)
Return	true	Out	Succeeded
	false	Out	Failed
Description	Process a read request from debugger		
Steps	This API is callback function called by common target I/F. 1. Extract read request information from arguments (addr, size) and transaction extensions from argument (p_ext), and process the read request 2. Set read data to argument (p_data) 3. Return true if this process succeeded, false if failed		
Notes	1. Initiator I/F parameters(ini_if_param) in transaction extensions is available only if initiator uses common initiator I/F 2. This API is called only by debugger; do not describe target behavior other than data copy to memory or register 3. Do not use wait() statement in this API 4. Do not call initiator access in this API; invoke separate process which calls initiator access instead (see subsection 2-9-5 for example)		
Example			

Example:

```
bool tgt_module<BUSWIDTH>::
tgt_rd_dbg(unsigned int addr, unsigned char *p_data, unsigned int size, vpcl::tlm_if_extension *p_ext) {
  memcpy(p_data, m_mem + addr&0xff, size);  // step 1, 2
  return true;  // step 3
}
```

2-6-5.(a) Modeling: Classes and Files

- Target high-speed model: class and files
- This model consists of TLM common class(target I/F class) and core class, and the core class inherits the target I/F class.
- Use prepared library(tlm_if.h, tlm_tgt_if.h) without any modification for target I/F class.
- Develop only core class descriptions.



Model Block Diagram

Classes and Files of Target Model

Class	Class name	Filename	Notes
Target I/F class	tlm_tgt_if	tlm_if.h *1 tlm_tgt_if.h	Use prepared library
Core class *2	Arbitrary (example) tgt_module	Arbitrary (example) tgt_module.h tgt_module.cpp	Develop this

- *1 Core class does not have to explicitly include "tlm_if.h" as it is included in "tlm_tgt_if.h"
- *2 Core class can use arbitrary class name and filenames

2-6-5.(b) Modeling: Descriptions

- Target high-speed model: descriptions
- Procedures to use common target I/F (No.1-3)
- Initializations of common target I/F (No.4-7)
- Describe resource access APIs (No.8-9)
- See section 2-9 for a target model sample.

Target high-speed model descriptions

No.	Level	File	Description
1	MUST	tgt_module.h	Include header file of common target I/F (tlm_tgt_if.h) at the top of core class header file (tgt_module.h)。
2	MUST	tgt_module.h	Inherit common target I/F class (tlm_tgt_if) in core class declaration
3	MUST	tgt_module.h	Set instance name of core class as constructor argument of common target I/F class
4	Option	tgt_module.h	Set target I/F parameters using parameter control APIs in core class constructor; Be sure to describe "this->" before the API name
5	Option	tgt_module.cpp or higher module	Call parameter control APIs in target core class or higher hierarchical level module which instantiates target class such as testbench
6	MUST	higher module	Instantiate target wirh core instance name and bus width in higher hierarchical level module
7	MUST	higher module	Connect socket of common target I/F class (m_tgt_socket) to initiator socket
8	MUST	tgt_module.h	Override resource access APIs (tgt_rd_dbg, tgt_rd_dbg, tgt_wr, tgt_rd)
9	MUST	tgt_module.cpp	Implement resource access APIs (tgt_rd_dbg, tgt_rd_dbg, tgt_wr, tgt_rd) -> [Reference] 2-6-4. API

2-9. Modeling Samples (Contents)

- 2-9-1. Build Environment and How to Execute
- 2-9-2. Initiator High-speed Model (Sec.2-3) vs. Target High-speed Model (Sec.2-6)
- 2-9-3. Initiator Bus-accurate Simple Model (Sec.2-4) vs. Target Bus-accurate Simple Model (Sec.2-7)
- 2-9-4. Initiator Bus-accurate Model (Sec.2-5) vs. Target Bus-accurate Model (Sec.2-8)
- 2-9-5. Target and Initiator High-speed Model

- Subsections 2-9-2, 2-9-3, and 2-9-4 are omitted in this English edition.

- Source code of the modeling samples introduced in this section is uploaded to “EDA Home Page (EDA TOOLS information)”. The samples are freely download to reuse them. URL: http://www.hoku.renesas.com/EDA/tools-out/index_en.htm

2-9-1. Build Environment and How to Execute (1/2)

- This subsection explains build environment and how to execute modeling samples introduced in this chapter.

* The samples have been tested with following environment. They may be available in other environments.

- First, setup following build environment and directory and file configuration.
- Second, execute a build and execution script “run”. The script builds source files by makefile “Makefile”, then executes generated execution file to simulate. “run” and “Makefile” are shown in the next page.

* Following directory and file configuration and “Makefile” in the next page are example of subsection 2-9-5.

Build environment

Category	Item	Version
Machines	Linux/RHEL	3.0
	Linux/SLES	10.0
Compiler	GNU/gcc	4.2.2
Simulator	OSCI/SystemC	2.2
	OSCI/TLM	2.0

Directory and file configuration

Directory and file			Content
src/	tlm_if.h		Common header
	tlm_ini_if.h		Initiator I/F
	tlm_tgt_if.h		Target I/F
guide_sample_000/	src/	main.cpp	Top hierarchy
		ini_module.h, ini_module.cpp	Initiator module
		tgt_module.h, tgt_module.cpp	Target module
		bin_module.h, bin_module.cpp	Target and initiator module
	run		Build and execution script
	Makefile		Makefile

2-9-1. Build Environment and How to Execute (2/2)

run

```
#! /bin/csh -f

set path = (/common/appl/Renesas/SystemC/compiler/gcc4.2.2/bin $path)
setenv LD_LIBRARY_PATH "/common/appl/Renesas/SystemC/compiler/gcc4.2.2/lib":$LD_LIBRARY_PATH

gmake clean
gmake

tlm_test.exe
```

Setting of gcc version 4.2.2

Execution of build

Execution of simulation

Makefile

```
SYSTEMC_HOME = /common/appl/Renesas/SystemC/SystemC-2.2
TLM_HOME     = /common/appl/Renesas/SystemC/TLM-2008-06-09
TARGET_ARCH  = linux
SYSTEMC_KIND = -gcc422-m32

LD = g++ -m32
CC = g++ -m32

LDFLAGS = -L$(SYSTEMC_HOME)/lib-$(TARGET_ARCH) -lsystemc$(SYSTEMC_KIND)
CFLAGS  = -Wall -g

INCDIR  = -I$(SYSTEMC_HOME)/include -I$(TLM_HOME)/include/tlm -I../src -I./src
OBJS    = ./src/main.o ./src/ini_module.o ./src/tgt_module.o ./src/bin_module.o

all: tlm_test.exe

tlm_test.exe: $(OBJS)
	$(LD) -o $@ $(OBJS) $(LDFLAGS)

%.o :: %.cpp
	$(CC) $(CFLAGS) $(INCDIR) -c $< -o $@

clean:
	rm -f ./src/*.o ./*.exe
```

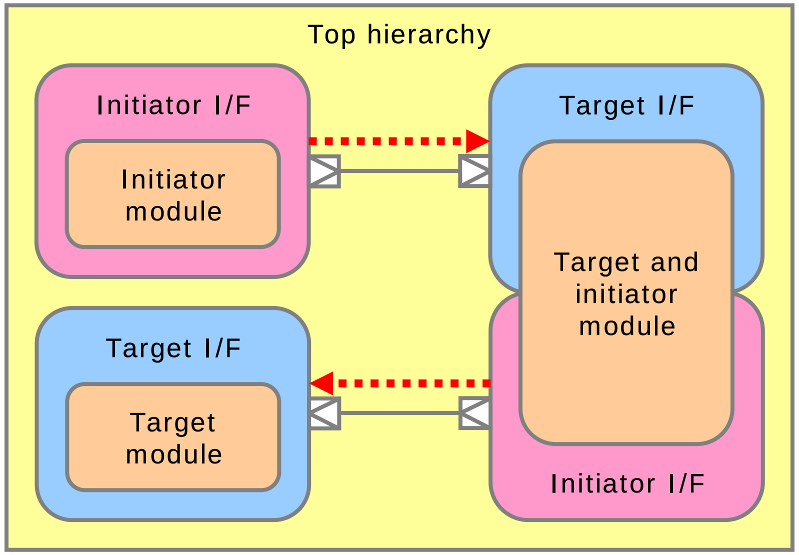
Setting of SystemC and TLM2.0

Setting of gcc version 4.2.2

2-9-5. Target and Initiator High-speed Model (1/2)

System configuration

No.	Name	Class	File
1	Top hierarchy	top_system	main.cpp
2	Initiator module	ini_module	ini_module.h ini_module.cpp
3	Target module	tgt_module	tgt_module.h tgt_module.cpp
4	Target and initiator module	bin_module	bin_module.h bin_module.cpp



Block diagram of system configuration

Parameter setting of initiator module

No.	Name	Variable	Value
1	Name of core instance	name	ini_mod
2	Bus width	bus_width	32
3	Access mode	access_mode	vpcl::TLM_IF_LT_ACCESS
4	Source ID	src_id	0x00
5	File pointer of log output	p_log_file	stdout
6	Switch of log output for write	wr_log	true
7	Switch of log output for read	rd_log	false

Essential

Parameter setting of target module

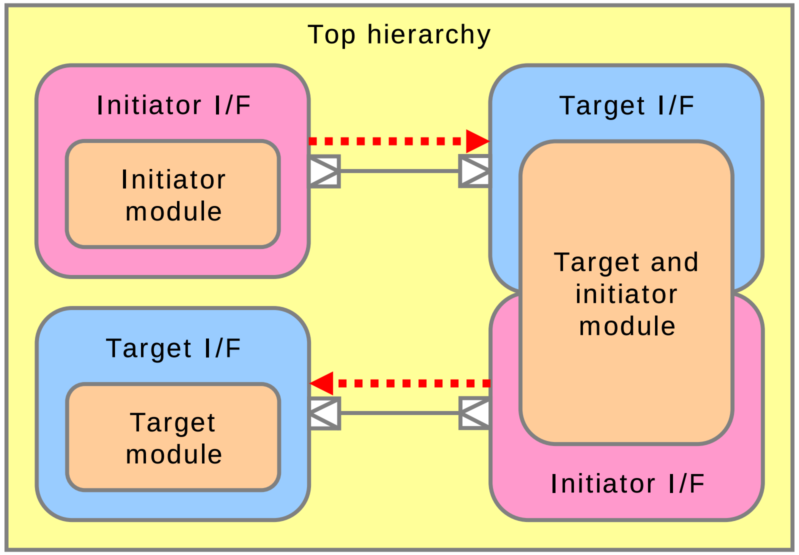
No.	Name	Variable	Value
1	Name of core instance	name	tgt_mod
2	Bus width	bus_width	32
3	Phase mode	phase_mode	TLM_IF_SINGLE_PHASE
4	Fixed write latency	wr_latency	sc_time(10, SC_NS)
5	Fixed read latency	rd_latency	sc_time(20, SC_NS)
6	File pointer of log output	p_log_file	stdout
7	Switch of log output for write	wr_log	false
8	Switch of log output for read	rd_log	true

Essential

2-9-5. Target and Initiator High-speed Model (2/2)

System configuration

No.	Name	Class	File
1	Top hierarchy	top_system	main.cpp
2	Initiator module	ini_module	ini_module.h ini_module.cpp
3	Target module	tgt_module	tgt_module.h tgt_module.cpp
4	Target and initiator module	bin_module	bin_module.h bin_module.cpp



Block diagram of system configuration

Parameter setting of target and initiator module (initiator side)

No.	Name	Variable	Value
1	Name of core instance	name	bin_mod
2	Bus width	bus_width	32
3	Access mode	access_mode	vpcl::TLM_IF_LT_ACCESS
4	Source ID	src_id	0x01
5	File pointer of log output	p_log_file	stdout
6	Switch of log output for write	wr_log	true
7	Switch of log output for read	rd_log	false

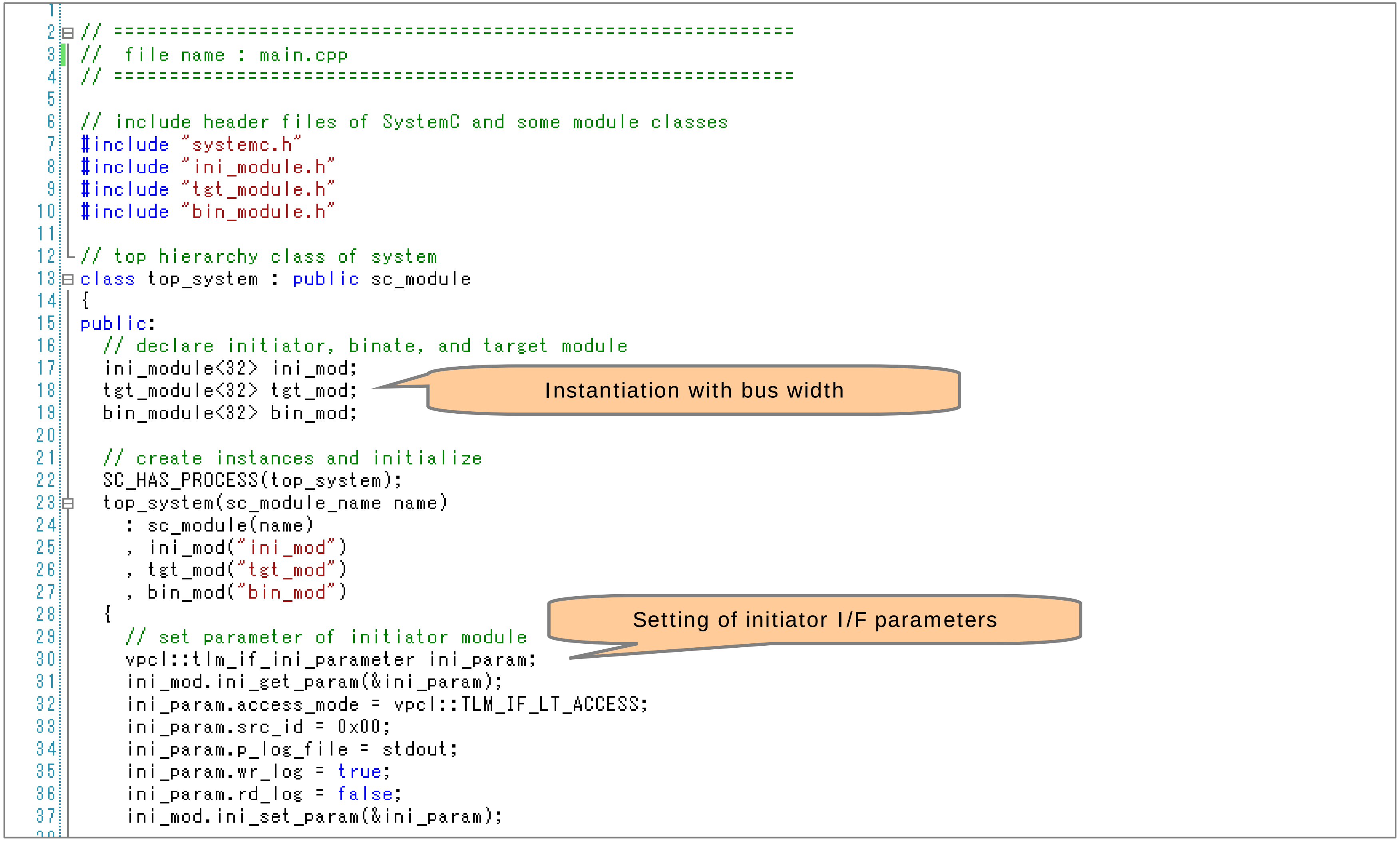
Essential

Parameter setting of target and initiator module (target side)

No.	Name	Variable	Value
1	Name of core instance	name	bin_mod
2	Bus width	bus_width	32
3	Phase mode	phase_mode	TLM_IF_SINGLE_PHASE
4	Fixed write latency	wr_latency	sc_time(10, SC_NS)
5	Fixed read latency	rd_latency	sc_time(20, SC_NS)
6	File pointer of log output	p_log_file	stdout
7	Switch of log output for write	wr_log	false
8	Switch of log output for read	rd_log	true

Essential

2-9-5. main.cpp (1/3)

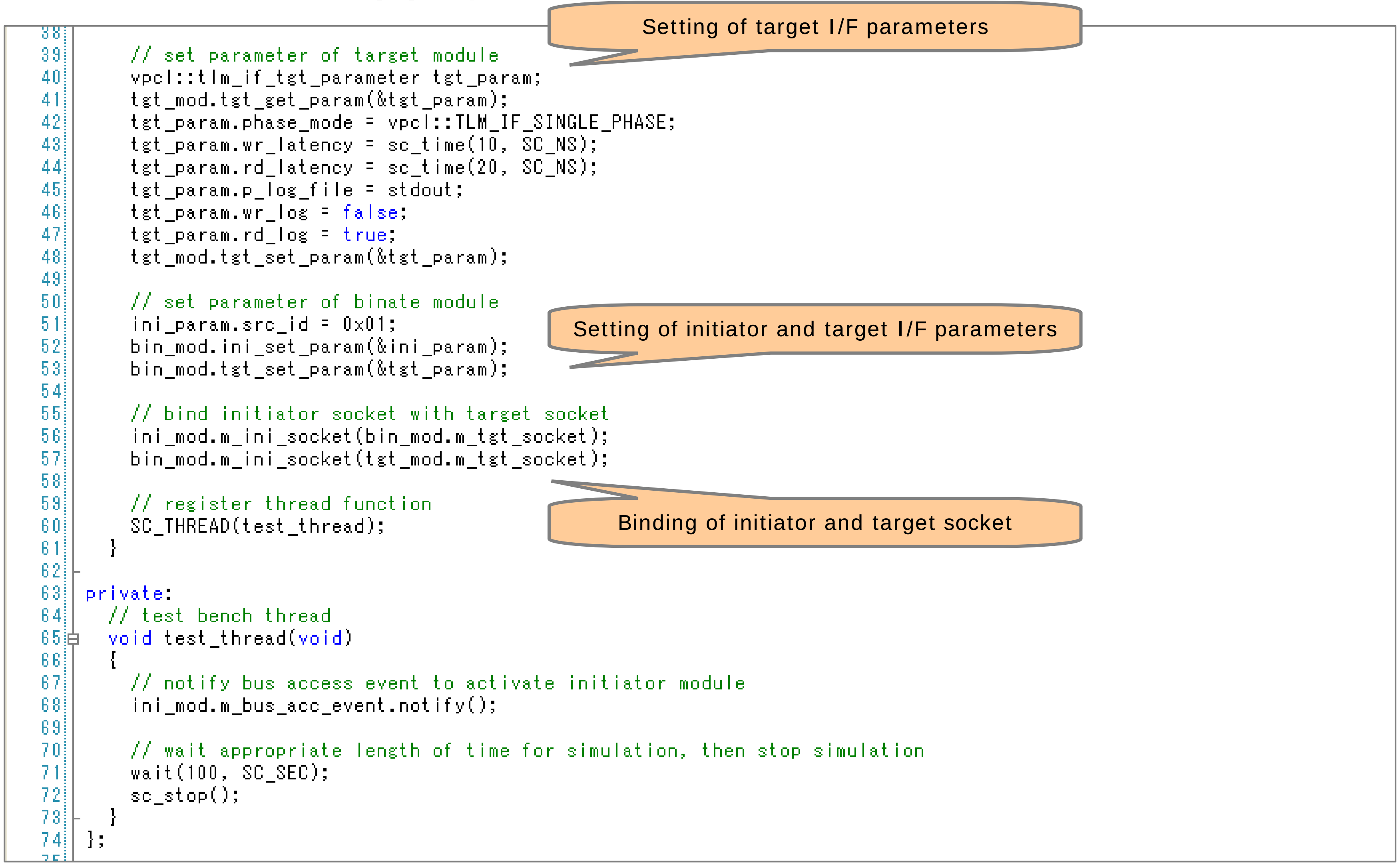


```
// =============================================================
//  file name : main.cpp
// =============================================================

// include header files of SystemC and some module classes
#include "systemc.h"
#include "ini_module.h"
#include "tgt_module.h"
#include "bin_module.h"

// top hierarchy class of system
class top_system : public sc_module
{
public:
  // declare initiator, binate, and target module
  ini_module<32> ini_mod;
  tgt_module<32> tgt_mod;
  bin_module<32> bin_mod;

  // create instances and initialize
  SC_HAS_PROCESS(top_system);
  top_system(sc_module_name name)
    : sc_module(name)
    , ini_mod("ini_mod")
    , tgt_mod("tgt_mod")
    , bin_mod("bin_mod")
  {
    // set parameter of initiator module
    vpcl::tlm_if_ini_parameter ini_param;
    ini_mod.ini_get_param(&ini_param);
    ini_param.access_mode = vpcl::TLM_IF_LT_ACCESS;
    ini_param.src_id = 0x00;
    ini_param.p_log_file = stdout;
    ini_param.wr_log = true;
    ini_param.rd_log = false;
    ini_mod.ini_set_param(&ini_param);
```

Instantiation with bus width

Setting of initiator I/F parameters

2-9-5. main.cpp (2/3)



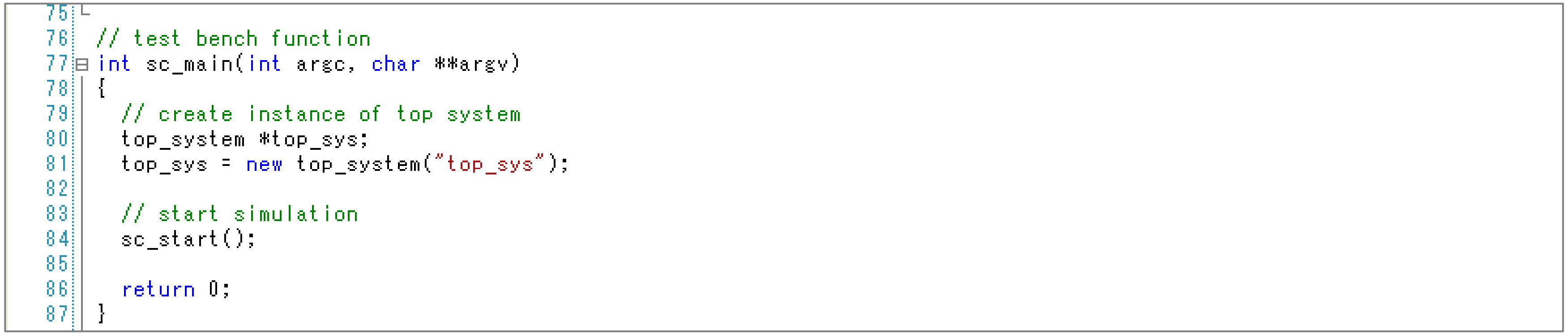
```
    // set parameter of target module
    vpcl::tlm_if_tgt_parameter tgt_param;
    tgt_mod.tgt_get_param(&tgt_param);
    tgt_param.phase_mode = vpcl::TLM_IF_SINGLE_PHASE;
    tgt_param.wr_latency = sc_time(10, SC_NS);
    tgt_param.rd_latency = sc_time(20, SC_NS);
    tgt_param.p_log_file = stdout;
    tgt_param.wr_log = false;
    tgt_param.rd_log = true;
    tgt_mod.tgt_set_param(&tgt_param);

    // set parameter of binate module
    ini_param.src_id = 0x01;
    bin_mod.ini_set_param(&ini_param);
    bin_mod.tgt_set_param(&tgt_param);

    // bind initiator socket with target socket
    ini_mod.m_ini_socket(bin_mod.m_tgt_socket);
    bin_mod.m_ini_socket(tgt_mod.m_tgt_socket);

    // register thread function
    SC_THREAD(test_thread);
  }

private:
  // test bench thread
  void test_thread(void)
  {
    // notify bus access event to activate initiator module
    ini_mod.m_bus_acc_event.notify();

    // wait appropriate length of time for simulation, then stop simulation
    wait(100, SC_SEC);
    sc_stop();
  }
};
```

Setting of target I/F parameters

Setting of initiator and target I/F parameters

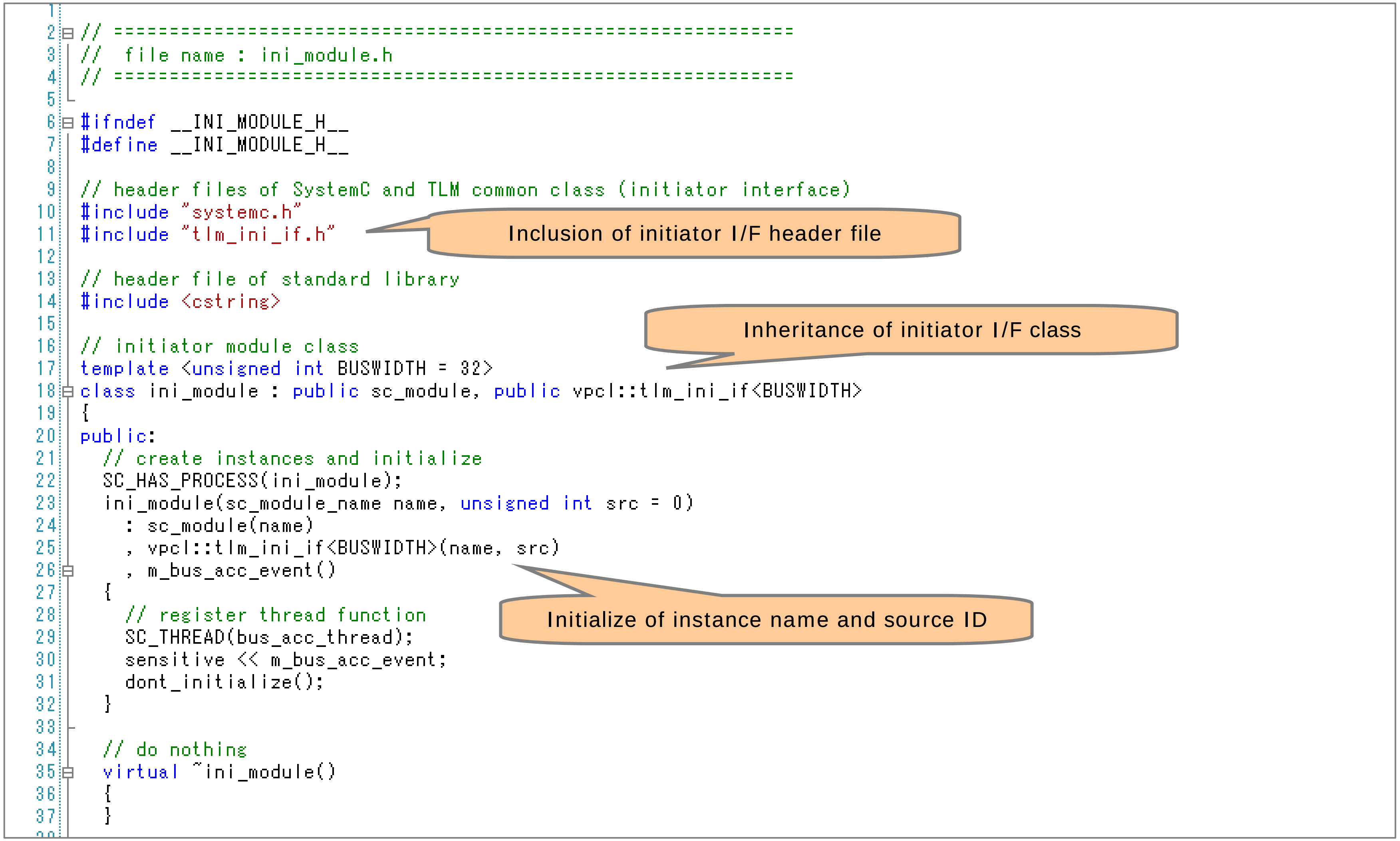
Binding of initiator and target socket

2-9-5. main.cpp (3/3)



```
// test bench function
int sc_main(int argc, char **argv)
{
  // create instance of top system
  top_system *top_sys;
  top_sys = new top_system("top_sys");

  // start simulation
  sc_start();

  return 0;
}
```

2-9-5. ini_module.h (1/2)



```
// =============================================================
//  file name : ini_module.h
// =============================================================

#ifndef __INI_MODULE_H__
#define __INI_MODULE_H__

// header files of SystemC and TLM common class (initiator interface)
#include "systemc.h"
#include "tlm_ini_if.h"

// header file of standard library
#include <cstring>

// initiator module class
template <unsigned int BUSWIDTH = 32>
class ini_module : public sc_module, public vpcl::tlm_ini_if<BUSWIDTH>
{
public:
  // create instances and initialize
  SC_HAS_PROCESS(ini_module);
  ini_module(sc_module_name name, unsigned int src = 0)
    : sc_module(name)
    , vpcl::tlm_ini_if<BUSWIDTH>(name, src)
    , m_bus_acc_event()
  {
    // register thread function
    SC_THREAD(bus_acc_thread);
    sensitive << m_bus_acc_event;
    dont_initialize();
  }

  // do nothing
  virtual ~ini_module()
  {
  }
```

Inclusion of initiator I/F header file

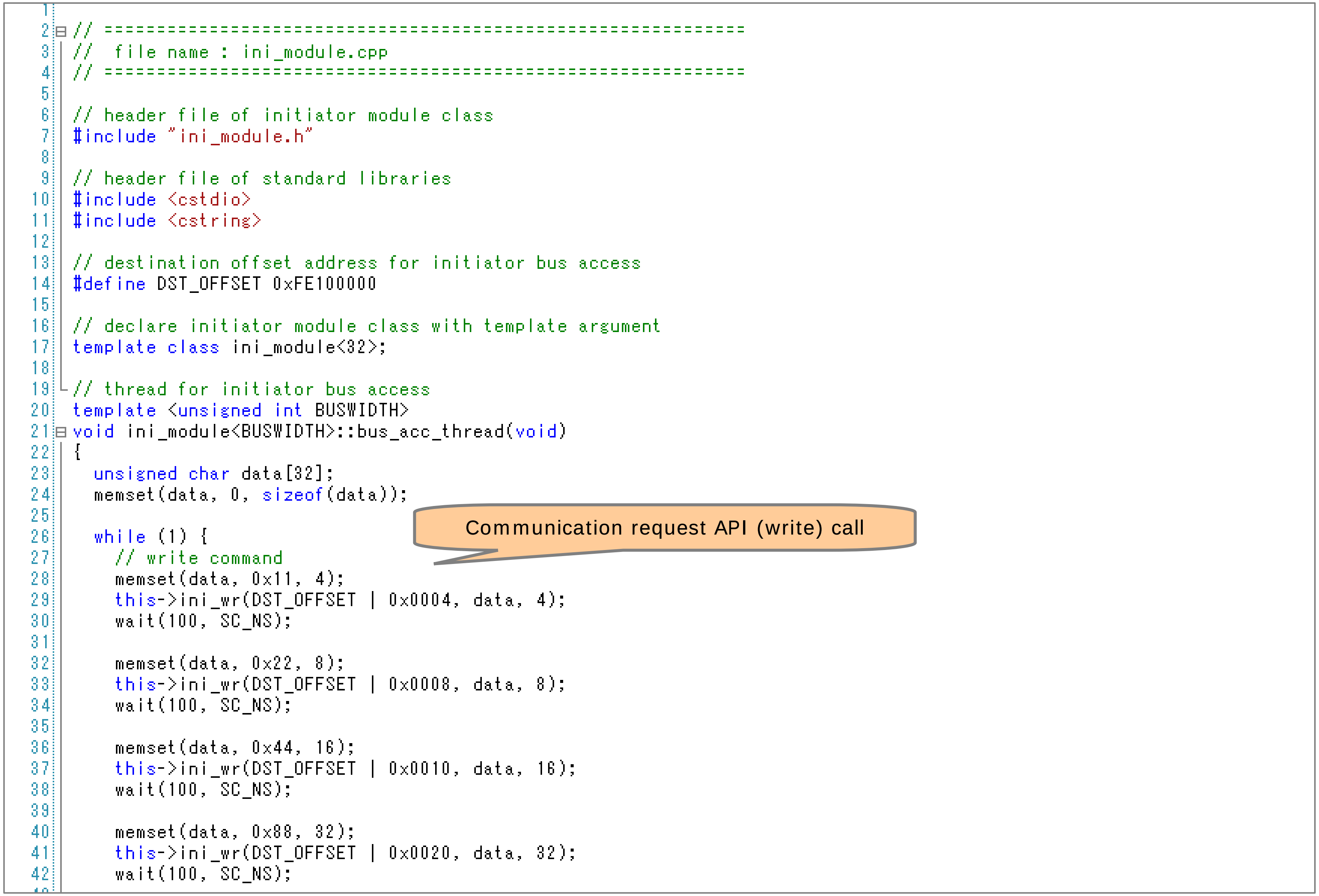
Inheritance of initiator I/F class

Initialize of instance name and source ID

2-9-5. ini_module.h (2/2)

```
  // event for initiator bus access thread
  sc_event m_bus_acc_event;

private:
  // function for initiator bus access thread
  void bus_acc_thread(void);
};

#endif
```

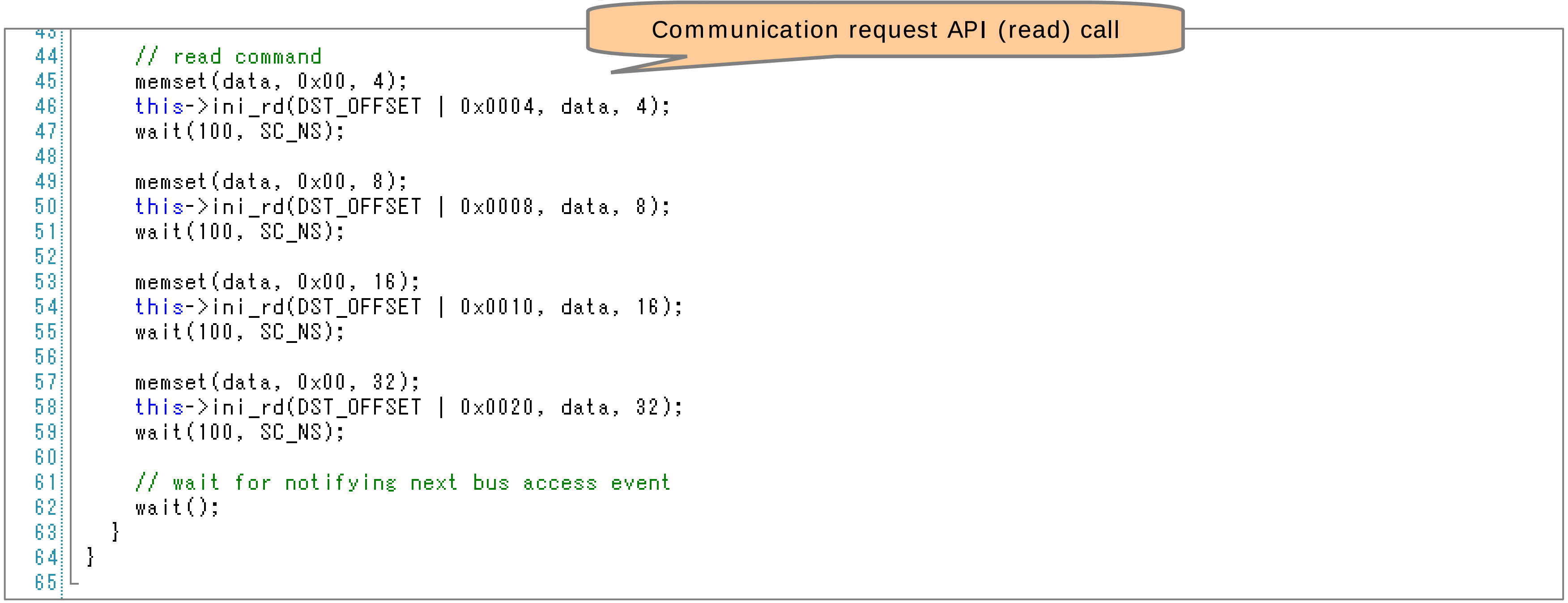
2-9-5. ini_module.cpp (1/2)



```

// ============================================================
//  file name : ini_module.cpp
// ============================================================

// header file of initiator module class
#include "ini_module.h"

// header file of standard libraries
#include <cstdio>
#include <cstring>

// destination offset address for initiator bus access
#define DST_OFFSET 0xFE100000

// declare initiator module class with template argument
template class ini_module<32>;

// thread for initiator bus access
template <unsigned int BUSWIDTH>
void ini_module<BUSWIDTH>::bus_acc_thread(void)
{
  unsigned char data[32];
  memset(data, 0, sizeof(data));

  while (1) {
    // write command
    memset(data, 0x11, 4);
    this->ini_wr(DST_OFFSET | 0x0004, data, 4);
    wait(100, SC_NS);

    memset(data, 0x22, 8);
    this->ini_wr(DST_OFFSET | 0x0008, data, 8);
    wait(100, SC_NS);

    memset(data, 0x44, 16);
    this->ini_wr(DST_OFFSET | 0x0010, data, 16);
    wait(100, SC_NS);

    memset(data, 0x88, 32);
    this->ini_wr(DST_OFFSET | 0x0020, data, 32);
    wait(100, SC_NS);
```

Communication request API (write) call

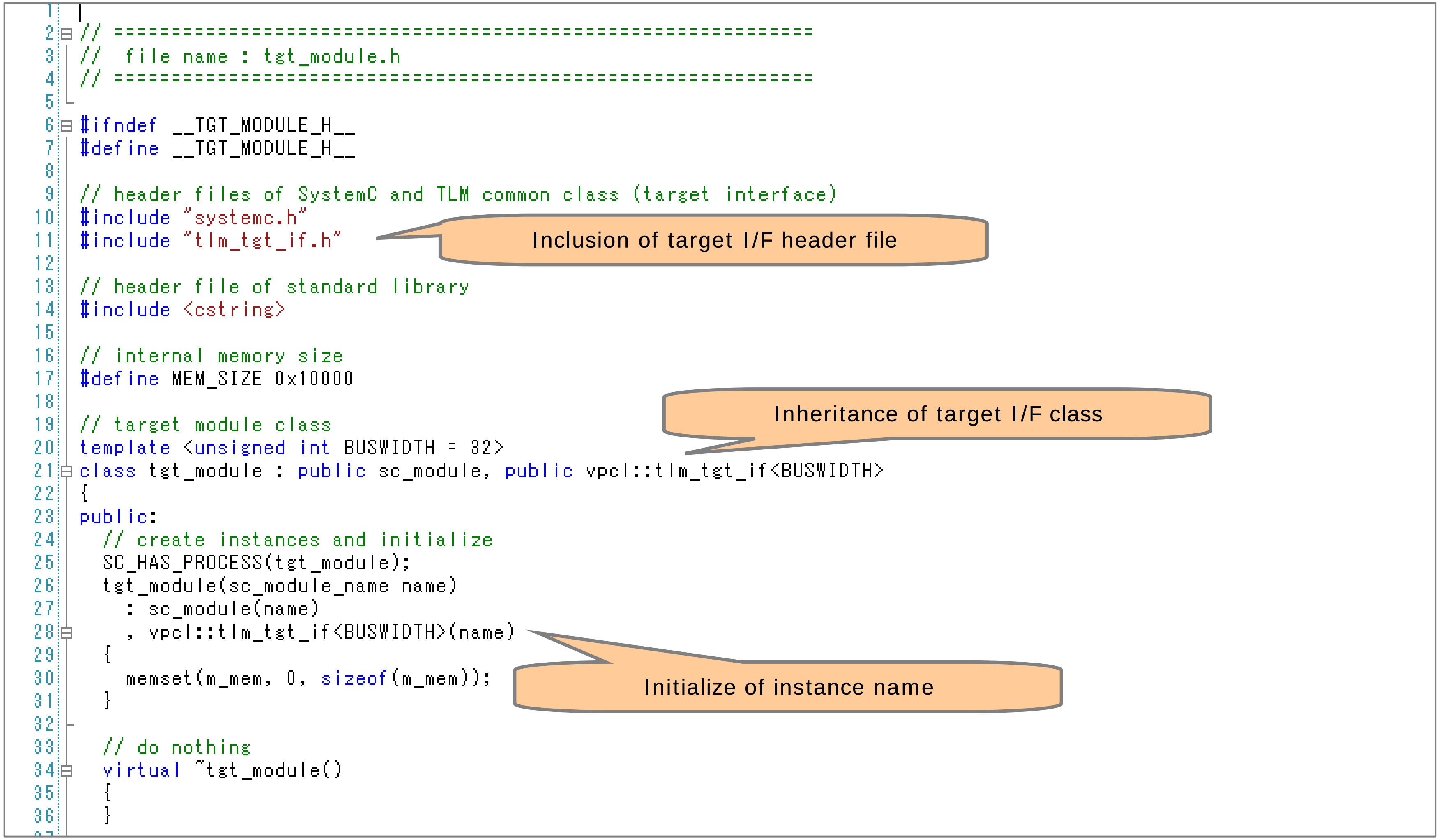
2-9-5. ini_module.cpp (2/2)



Communication request API (read) call

```
    // read command
    memset(data, 0x00, 4);
    this->ini_rd(DST_OFFSET | 0x0004, data, 4);
    wait(100, SC_NS);

    memset(data, 0x00, 8);
    this->ini_rd(DST_OFFSET | 0x0008, data, 8);
    wait(100, SC_NS);

    memset(data, 0x00, 16);
    this->ini_rd(DST_OFFSET | 0x0010, data, 16);
    wait(100, SC_NS);

    memset(data, 0x00, 32);
    this->ini_rd(DST_OFFSET | 0x0020, data, 32);
    wait(100, SC_NS);

    // wait for notifying next bus access event
    wait();
  }
}
```

2-9-5. tgt_module.h (1/2)



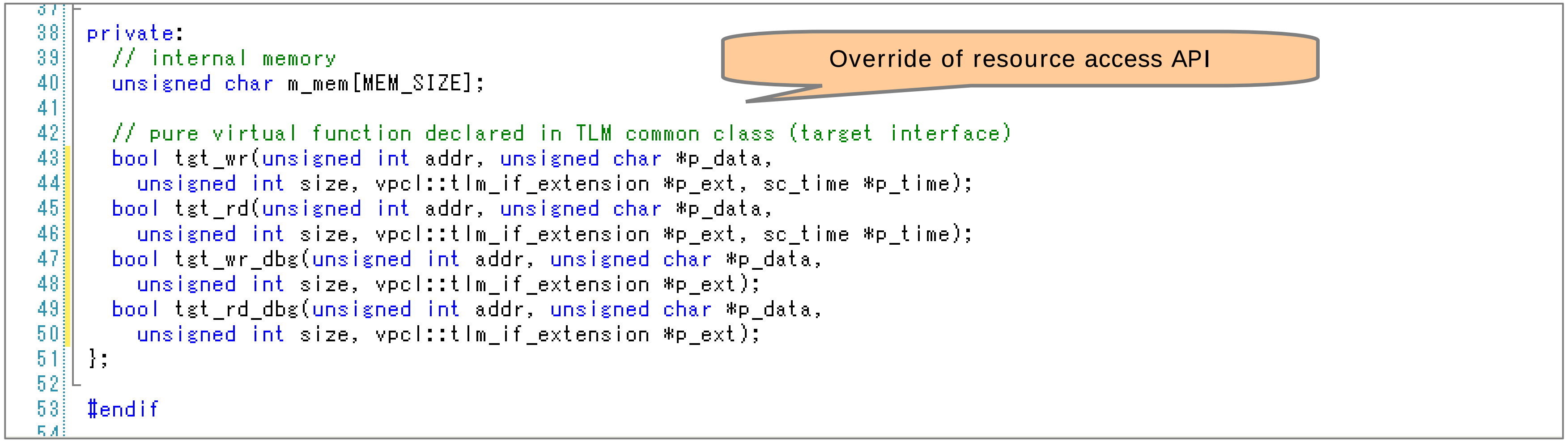
```

// ============================================================
//  file name : tgt_module.h
// ============================================================

#ifndef __TGT_MODULE_H__
#define __TGT_MODULE_H__

// header files of SystemC and TLM common class (target interface)
#include "systemc.h"
#include "tlm_tgt_if.h"

// header file of standard library
#include <cstring>

// internal memory size
#define MEM_SIZE 0x10000

// target module class
template <unsigned int BUSWIDTH = 32>
class tgt_module : public sc_module, public vpcl::tlm_tgt_if<BUSWIDTH>
{
public:
  // create instances and initialize
  SC_HAS_PROCESS(tgt_module);
  tgt_module(sc_module_name name)
    : sc_module(name)
    , vpcl::tlm_tgt_if<BUSWIDTH>(name)
  {
    memset(m_mem, 0, sizeof(m_mem));
  }

  // do nothing
  virtual ~tgt_module()
  {
  }
```

Inclusion of target I/F header file

Inheritance of target I/F class

Initialize of instance name

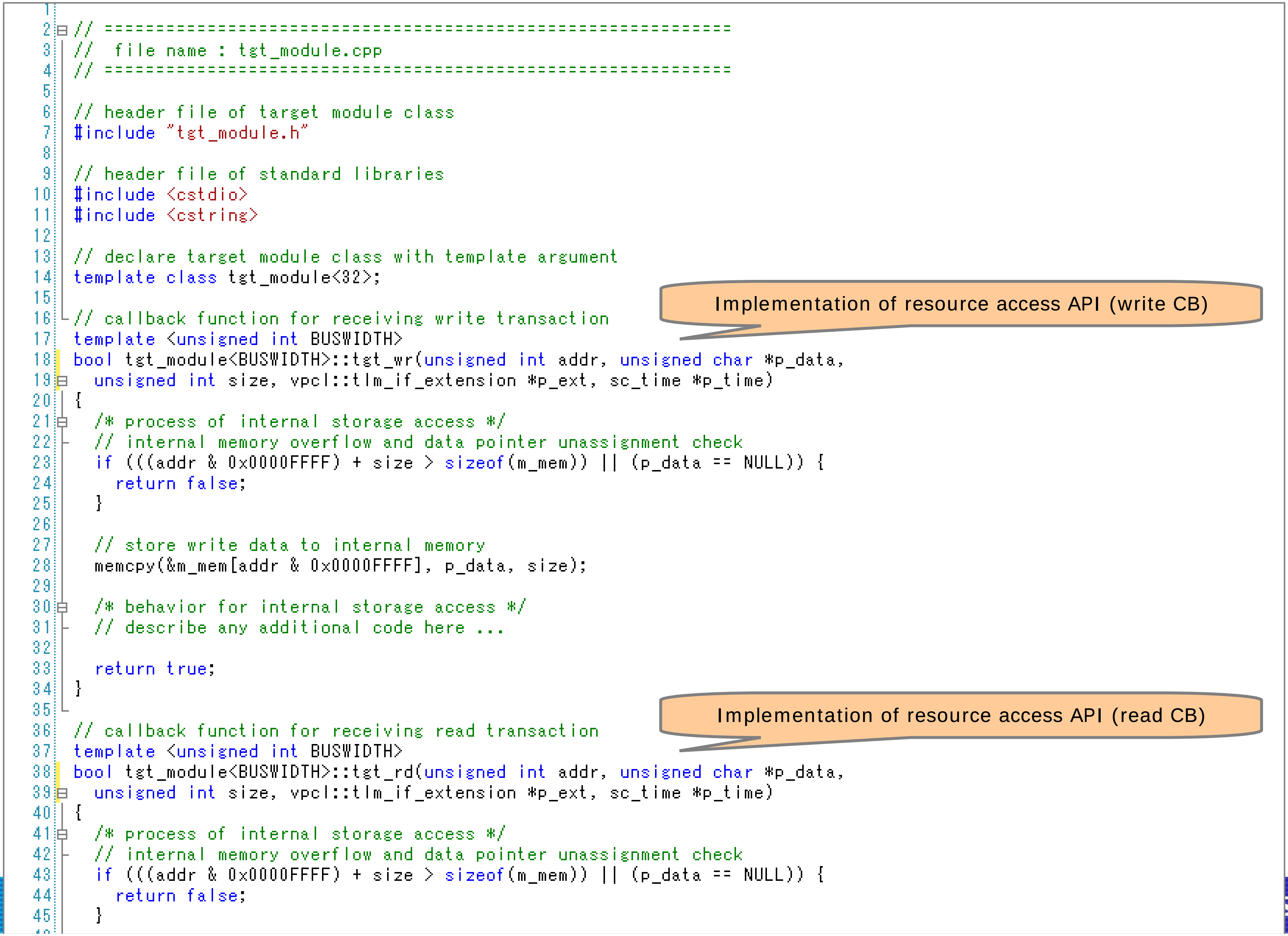
2-9-5. tgt_module.h (2/2)



```
private:
  // internal memory
  unsigned char m_mem[MEM_SIZE];

  // pure virtual function declared in TLM common class (target interface)
  bool tgt_wr(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time);
  bool tgt_rd(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time);
  bool tgt_wr_dbg(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext);
  bool tgt_rd_dbg(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext);
};

#endif
```

Override of resource access API

2-9-5. tgt_module.cpp (1/2)

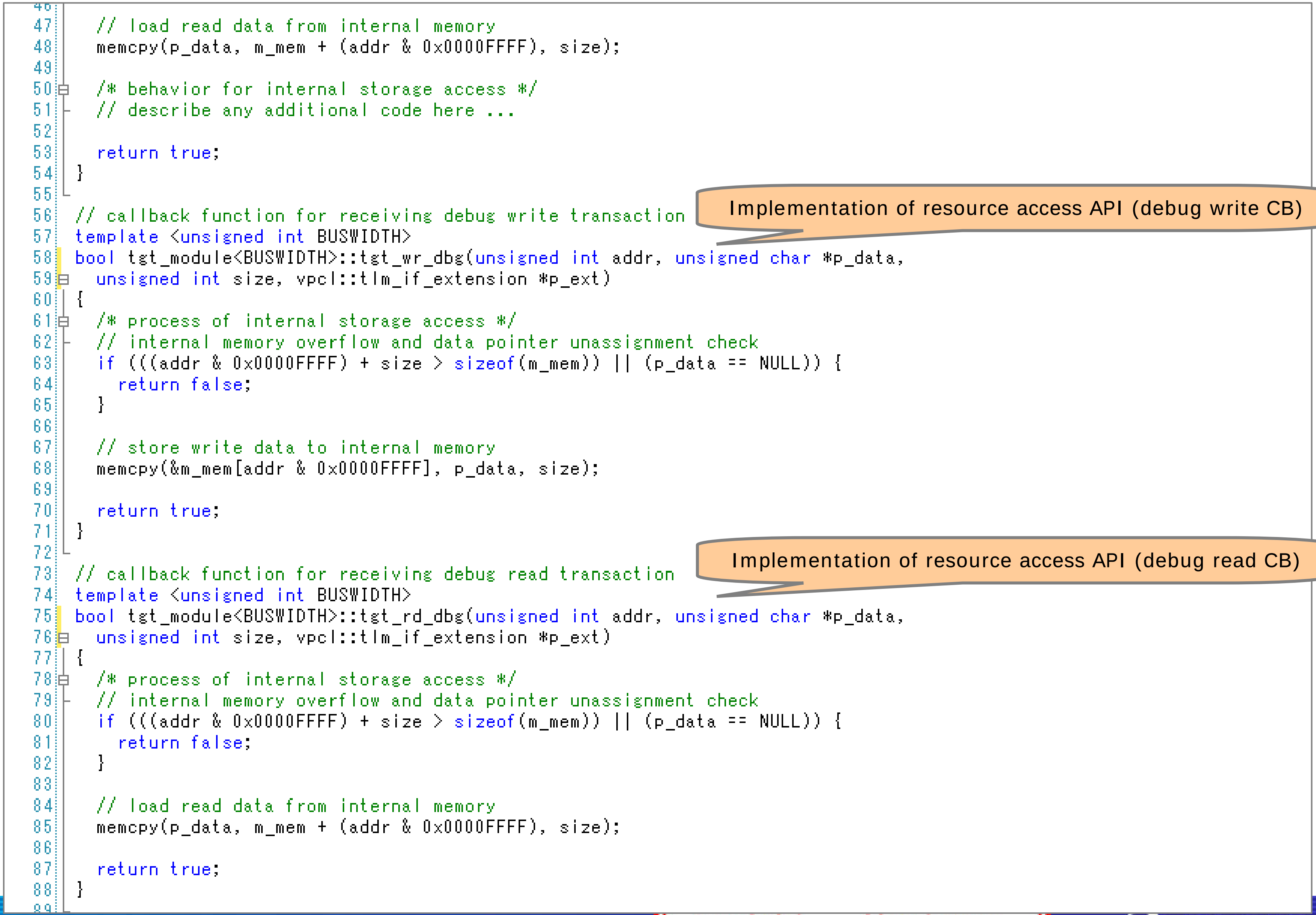


Implementation of resource access API (write CB)

Implementation of resource access API (read CB)

```

// ============================================================
//  file name : tgt_module.cpp
// ============================================================

// header file of target module class
#include "tgt_module.h"

// header file of standard libraries
#include <cstdio>
#include <cstring>

// declare target module class with template argument
template class tgt_module<32>;

// callback function for receiving write transaction
template <unsigned int BUSWIDTH>
bool tgt_module<BUSWIDTH>::tgt_wr(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }

  // store write data to internal memory
  memcpy(&m_mem[addr & 0x0000FFFF], p_data, size);

  /* behavior for internal storage access */
  // describe any additional code here ...

  return true;
}

// callback function for receiving read transaction
template <unsigned int BUSWIDTH>
bool tgt_module<BUSWIDTH>::tgt_rd(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }
```

2-9-5. tgt_module.cpp (2/2)

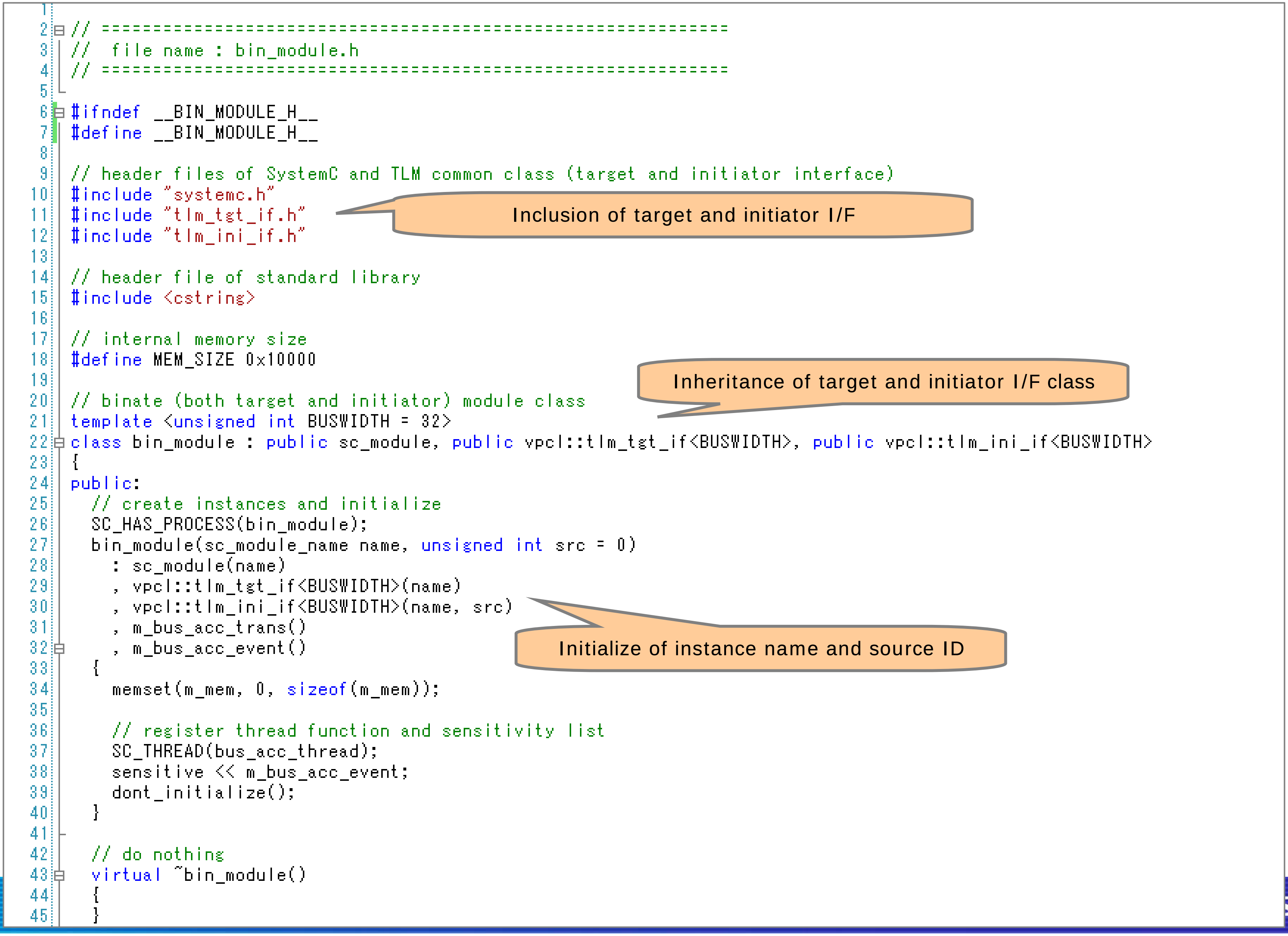


```
  // load read data from internal memory
  memcpy(p_data, m_mem + (addr & 0x0000FFFF), size);

  /* behavior for internal storage access */
  // describe any additional code here ...

  return true;
}

// callback function for receiving debug write transaction
template <unsigned int BUSWIDTH>
bool tgt_module<BUSWIDTH>::tgt_wr_dbg(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }

  // store write data to internal memory
  memcpy(&m_mem[addr & 0x0000FFFF], p_data, size);

  return true;
}

// callback function for receiving debug read transaction
template <unsigned int BUSWIDTH>
bool tgt_module<BUSWIDTH>::tgt_rd_dbg(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }

  // load read data from internal memory
  memcpy(p_data, m_mem + (addr & 0x0000FFFF), size);

  return true;
}
```

Implementation of resource access API (debug write CB)

Implementation of resource access API (debug read CB)

2-9-5. bin_module.h (1/2)



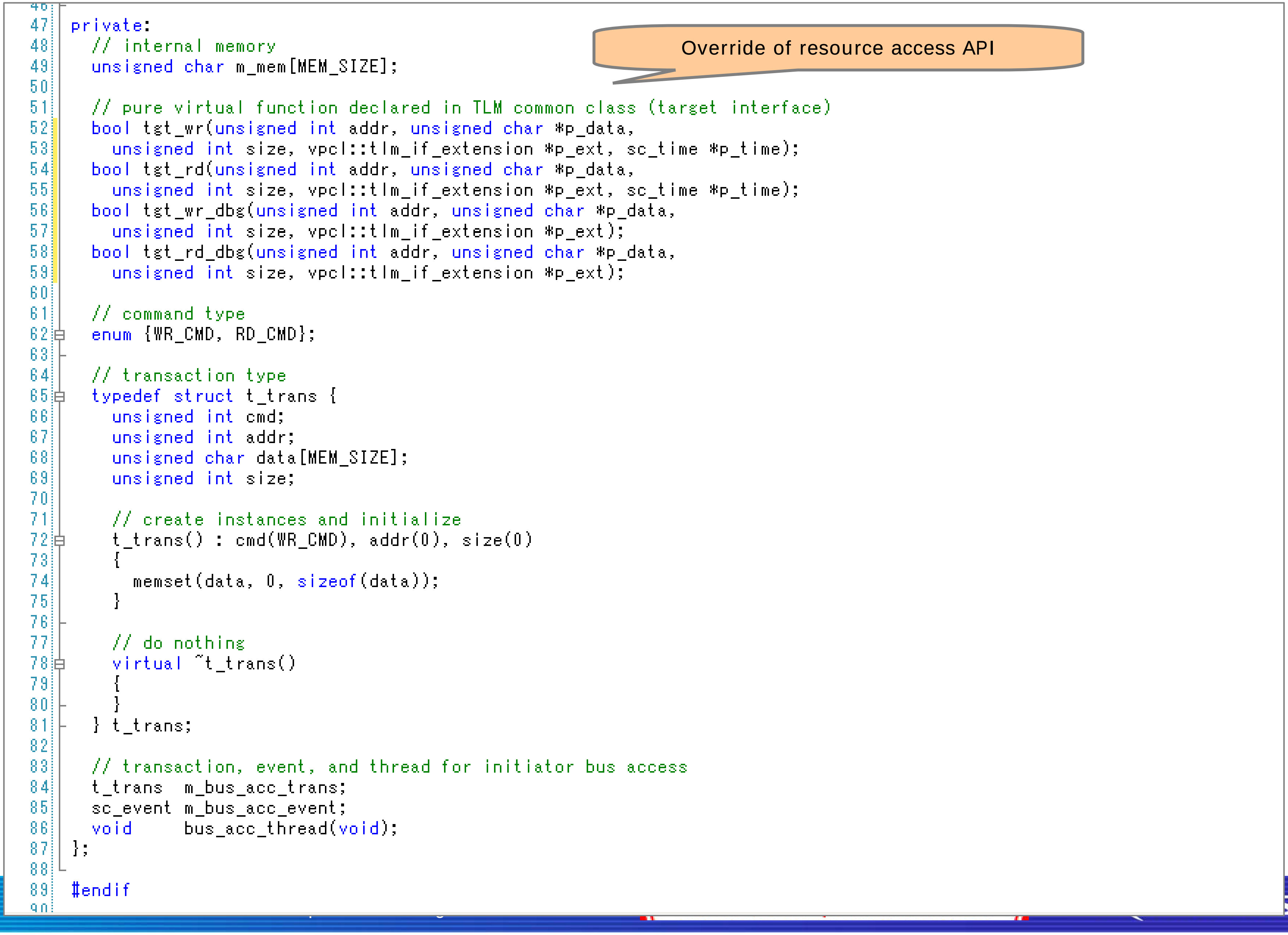
```

// =============================================================
//  file name : bin_module.h
// =============================================================

#ifndef __BIN_MODULE_H__
#define __BIN_MODULE_H__

// header files of SystemC and TLM common class (target and initiator interface)
#include "systemc.h"
#include "tlm_tgt_if.h"
#include "tlm_ini_if.h"

// header file of standard library
#include <cstring>

// internal memory size
#define MEM_SIZE 0x10000

// binate (both target and initiator) module class
template <unsigned int BUSWIDTH = 32>
class bin_module : public sc_module, public vpcl::tlm_tgt_if<BUSWIDTH>, public vpcl::tlm_ini_if<BUSWIDTH>
{
public:
  // create instances and initialize
  SC_HAS_PROCESS(bin_module);
  bin_module(sc_module_name name, unsigned int src = 0)
    : sc_module(name)
    , vpcl::tlm_tgt_if<BUSWIDTH>(name)
    , vpcl::tlm_ini_if<BUSWIDTH>(name, src)
    , m_bus_acc_trans()
    , m_bus_acc_event()
  {
    memset(m_mem, 0, sizeof(m_mem));

    // register thread function and sensitivity list
    SC_THREAD(bus_acc_thread);
    sensitive << m_bus_acc_event;
    dont_initialize();
  }

  // do nothing
  virtual ~bin_module()
  {
  }
```

Inclusion of target and initiator I/F

Inheritance of target and initiator I/F class

Initialize of instance name and source ID

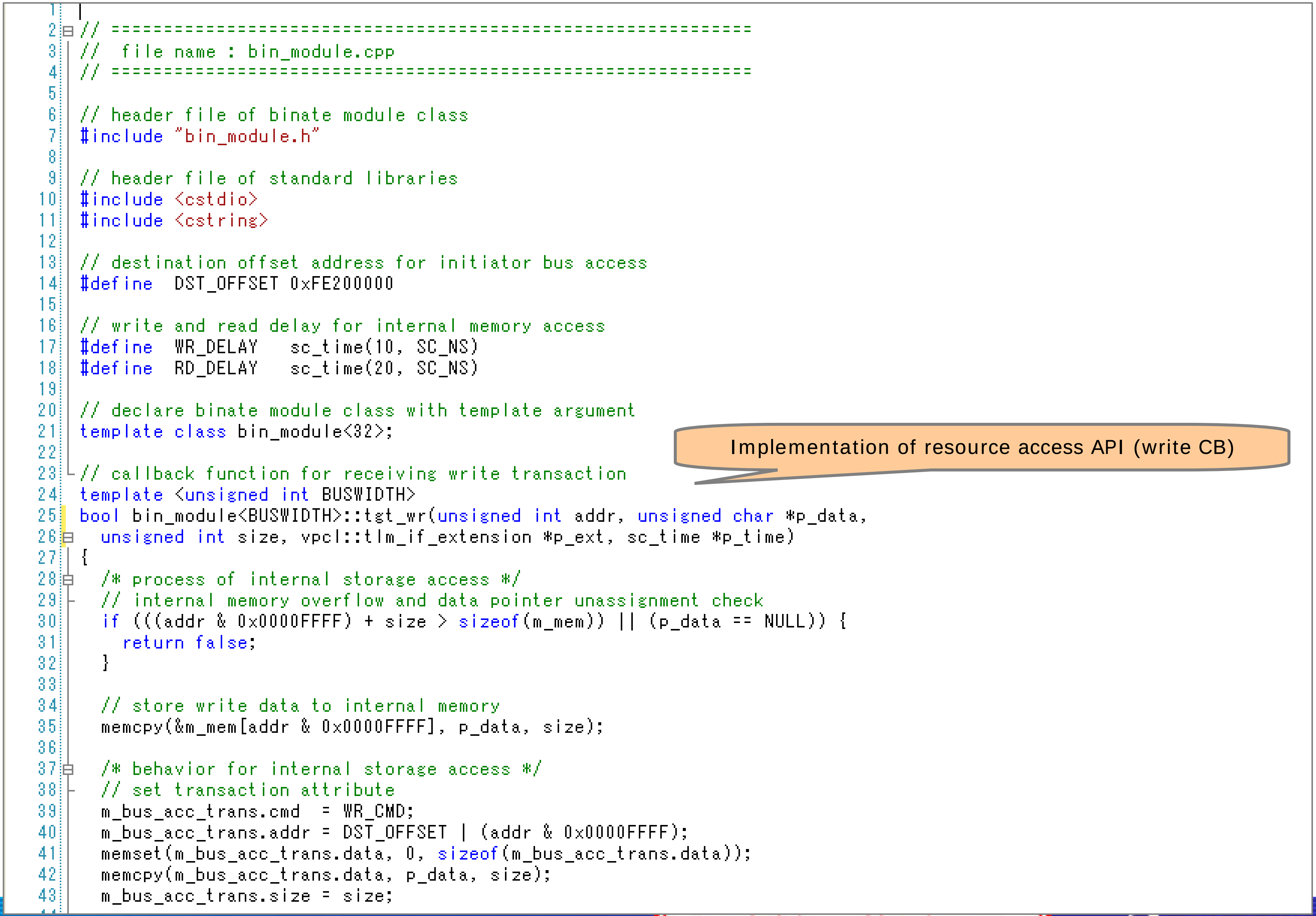
2-9-5. bin_module.h (2/2)



Override of resource access API

```
private:
  // internal memory
  unsigned char m_mem[MEM_SIZE];

  // pure virtual function declared in TLM common class (target interface)
  bool tgt_wr(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time);
  bool tgt_rd(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time);
  bool tgt_wr_dbg(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext);
  bool tgt_rd_dbg(unsigned int addr, unsigned char *p_data,
    unsigned int size, vpcl::tlm_if_extension *p_ext);

  // command type
  enum {WR_CMD, RD_CMD};

  // transaction type
  typedef struct t_trans {
    unsigned int cmd;
    unsigned int addr;
    unsigned char data[MEM_SIZE];
    unsigned int size;

    // create instances and initialize
    t_trans() : cmd(WR_CMD), addr(0), size(0)
    {
      memset(data, 0, sizeof(data));
    }

    // do nothing
    virtual ~t_trans()
    {
    }
  } t_trans;

  // transaction, event, and thread for initiator bus access
  t_trans  m_bus_acc_trans;
  sc_event m_bus_acc_event;
  void     bus_acc_thread(void);
};

#endif
```

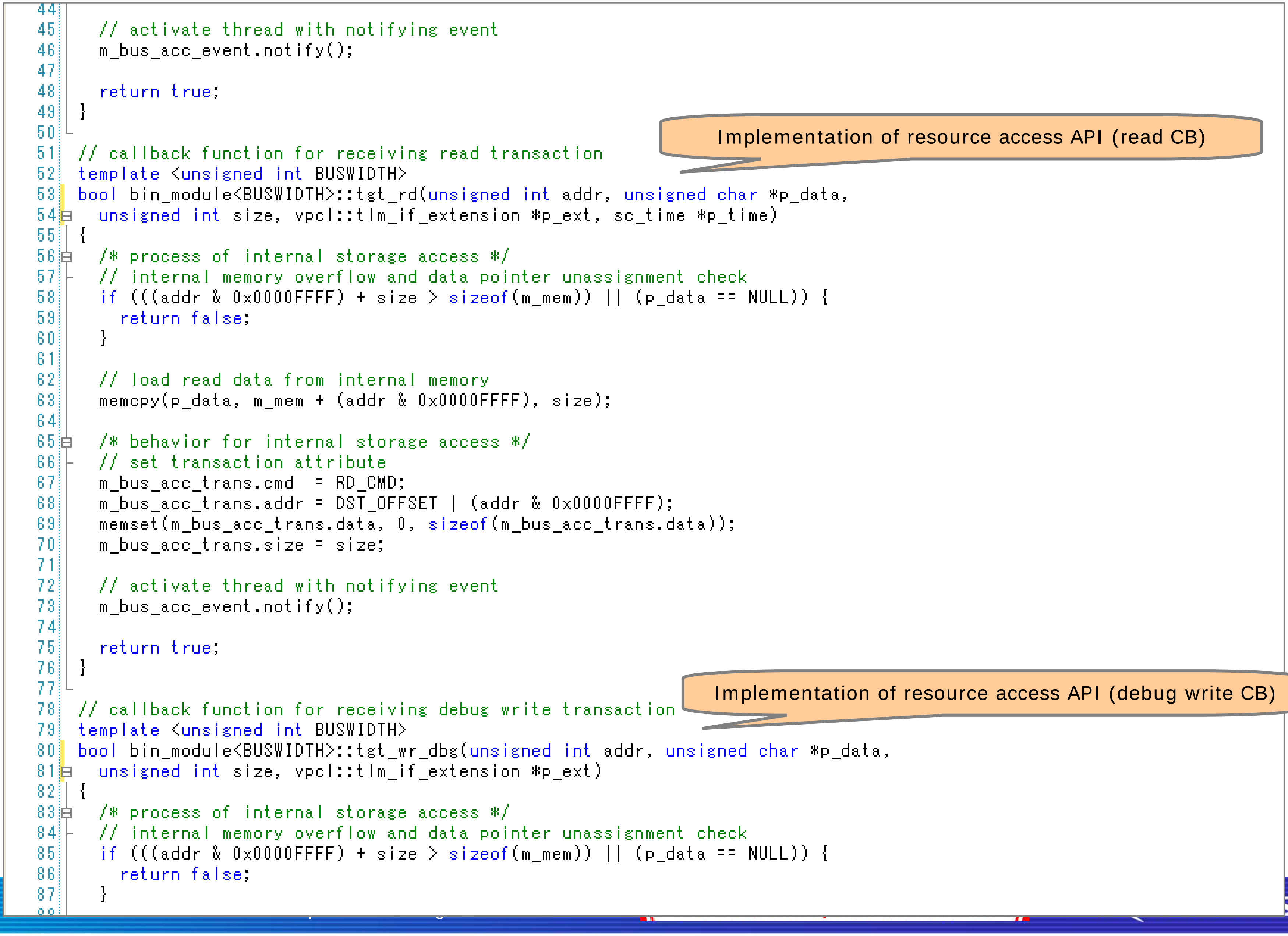
2-9-5. bin_module.cpp (1/3)



```

// =============================================================
//  file name : bin_module.cpp
// =============================================================

// header file of binate module class
#include "bin_module.h"

// header file of standard libraries
#include <cstdio>
#include <cstring>

// destination offset address for initiator bus access
#define  DST_OFFSET 0xFE200000

// write and read delay for internal memory access
#define  WR_DELAY   sc_time(10, SC_NS)
#define  RD_DELAY   sc_time(20, SC_NS)

// declare binate module class with template argument
template class bin_module<32>;

// callback function for receiving write transaction
template <unsigned int BUSWIDTH>
bool bin_module<BUSWIDTH>::tgt_wr(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }

  // store write data to internal memory
  memcpy(&m_mem[addr & 0x0000FFFF], p_data, size);

  /* behavior for internal storage access */
  // set transaction attribute
  m_bus_acc_trans.cmd  = WR_CMD;
  m_bus_acc_trans.addr = DST_OFFSET | (addr & 0x0000FFFF);
  memset(m_bus_acc_trans.data, 0, sizeof(m_bus_acc_trans.data));
  memcpy(m_bus_acc_trans.data, p_data, size);
  m_bus_acc_trans.size = size;
```

Implementation of resource access API (write CB)

2-9-5. bin_module.cpp (2/3)

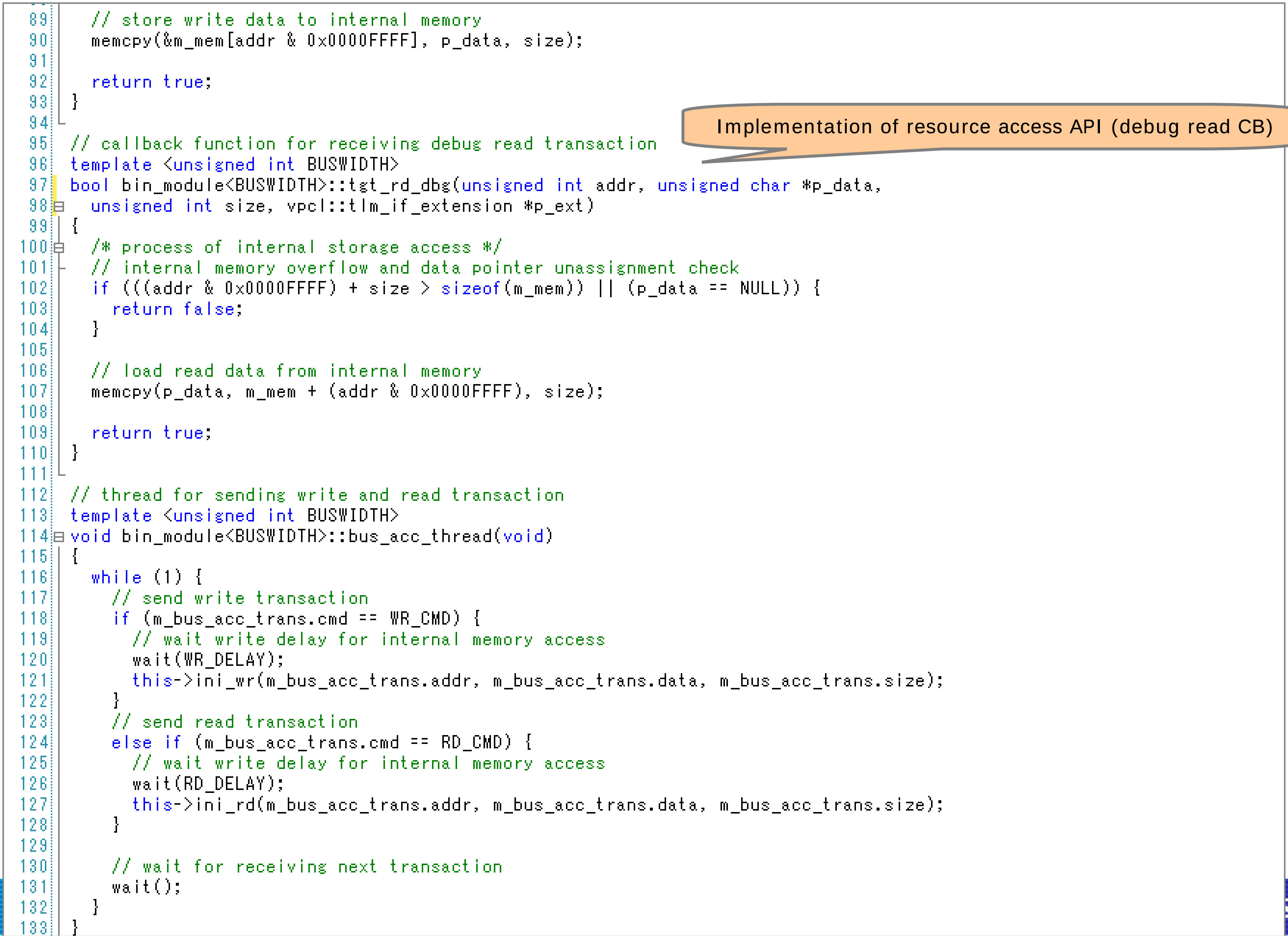


```
  // activate thread with notifying event
  m_bus_acc_event.notify();

  return true;
}

// callback function for receiving read transaction
template <unsigned int BUSWIDTH>
bool bin_module<BUSWIDTH>::tgt_rd(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext, sc_time *p_time)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }

  // load read data from internal memory
  memcpy(p_data, m_mem + (addr & 0x0000FFFF), size);

  /* behavior for internal storage access */
  // set transaction attribute
  m_bus_acc_trans.cmd  = RD_CMD;
  m_bus_acc_trans.addr = DST_OFFSET | (addr & 0x0000FFFF);
  memset(m_bus_acc_trans.data, 0, sizeof(m_bus_acc_trans.data));
  m_bus_acc_trans.size = size;

  // activate thread with notifying event
  m_bus_acc_event.notify();

  return true;
}

// callback function for receiving debug write transaction
template <unsigned int BUSWIDTH>
bool bin_module<BUSWIDTH>::tgt_wr_dbg(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }
```

Implementation of resource access API (read CB)

Implementation of resource access API (debug write CB)

2-9-5. bin_module.cpp (3/3)



```
  // store write data to internal memory
  memcpy(&m_mem[addr & 0x0000FFFF], p_data, size);

  return true;
}

// callback function for receiving debug read transaction
template <unsigned int BUSWIDTH>
bool bin_module<BUSWIDTH>::tgt_rd_dbg(unsigned int addr, unsigned char *p_data,
  unsigned int size, vpcl::tlm_if_extension *p_ext)
{
  /* process of internal storage access */
  // internal memory overflow and data pointer unassignment check
  if (((addr & 0x0000FFFF) + size > sizeof(m_mem)) || (p_data == NULL)) {
    return false;
  }

  // load read data from internal memory
  memcpy(p_data, m_mem + (addr & 0x0000FFFF), size);

  return true;
}

// thread for sending write and read transaction
template <unsigned int BUSWIDTH>
void bin_module<BUSWIDTH>::bus_acc_thread(void)
{
  while (1) {
    // send write transaction
    if (m_bus_acc_trans.cmd == WR_CMD) {
      // wait write delay for internal memory access
      wait(WR_DELAY);
      this->ini_wr(m_bus_acc_trans.addr, m_bus_acc_trans.data, m_bus_acc_trans.size);
    }
    // send read transaction
    else if (m_bus_acc_trans.cmd == RD_CMD) {
      // wait write delay for internal memory access
      wait(RD_DELAY);
      this->ini_rd(m_bus_acc_trans.addr, m_bus_acc_trans.data, m_bus_acc_trans.size);
    }

    // wait for receiving next transaction
    wait();
  }
}
```

Implementation of resource access API (debug read CB)

C Model Design Guide

Chapter 3.
Connecting Models to System Simulator

Renesas Electronics Corporation
Front-End Design Technology Development Department

2010/09/21 Rev. 1.2

Chapter 3 Organization

- Organization
  - 3.1 Connect to CoWare Virtual Platform
  - 3.2 Connect to VaST CoMET
  - 3.3 Connect to GAIO System Simulator

This chapter focuses on how to connect models to each system simulator. Please consult vendor manuals for detailed tool information.

3-1. Connect to CoWare: Outline

- This section explains model description and connection steps when we connect OSCI TLM2 model to CoWare Virtual Platform environment

- Modify OSCI TLM2 model so that the environment can:
  - recognize, observe, and debug registers
  - set model parameters using tcl command

- Use "#ifdef CWR_SYSTEMC ... #endif" area to put CoWare-specific description

(1) Add scml_memory instance declarations	MUST
(2) Modify re_register callback function	MUST
(3) Add parameter interface	**Recommended**
(4) Create TCL script	MUST
(5) Build model system library	MUST

3-1. Connect to CoWare: Model Description(1)

(1) Add scml_memory instance declarations

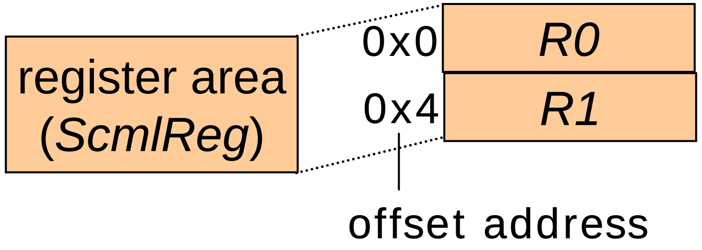
- CoWare vpa debugger can recognize, observe, and debug registers through scml_memory instances

Steps

<1> include scml header file
<2> declare scml_memory instance to cover register area
<3> specify name and size(*) of it in constructor
<4> declare scml_memory instance for each register
<5> specify name, alias, offset(*) and size(*) of them in constructor

* size and offset should be scaled in template variable type size;
see (CoWare-V2010.1.1)/Documentation/docs/pdf/PA_ModelingLib.pdf for details

Example: two 32-bit registers



register area
(*ScmlReg*)

0x0 *R0*

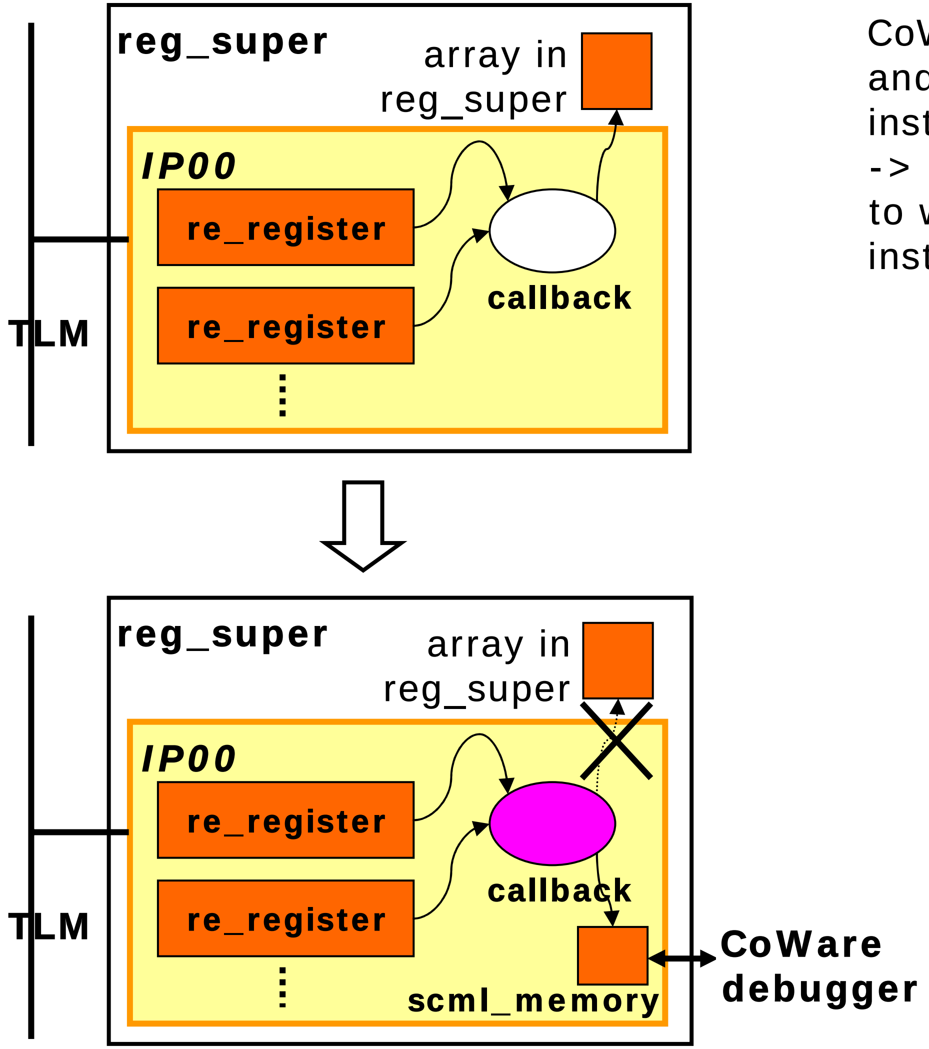
0x4 *R1*

offset address

```
<1> #include <scml.h>
    class IP00 ... {
    ...
<2> scml_memory<unsigned int> ScmlReg;
<4> scml_memory<unsigned int> R0;
    scml_memory<unsigned int> R1;
    ...
<3> , ScmlReg("ScmlReg", scml_memsize(2));
<5> , R0("R0", ScmlReg, 0x0/4, 1);
    , R1("R1", ScmlReg, 0x4/4, 1);
```

3-1. Connect to CoWare: Model Description(2)

(2) Modify re_register callback functions



CoWare vpa debugger can recognize, observe, and debug registers through scml_memory instances
-> modify callback functions (wr_cb and rd_cb) to write/read register values in scml_memory instead of array in reg_super class

Example

```
// write callback function
void wr_cb
  (const unsigned int addr, unsigned int data){
#ifdef  CWR_SYSTEMC
 ScmlReg.put(data, (addr&0xFF)/4, 32, 0);
#else
 array[addr&0xFF] = data;
#endif
}

// read callback function
unsigned int rd_cb(const unsigned int addr){
#ifdef  CWR_SYSTEMC
 return ScmlReg.get((addr&0xFF)/4, 32, 0);
#else
 return (array[addr&0xFF]);
#endif
}
```

3-1. Connect to CoWare: Model Description(3)

(3) Add parameter interface

- CoWare debugger can access model parameters through parameter interface in the model module constructor
  - use SCML_COMMAND_PROCESSOR to indicate which method should be called
  - use SCML_ADD_COMMAND to declare which commands can be handled by the object

```
SCML_COMMAND_PROCESSOR(handleCommand);
```

```
SCML_ADD_COMMAND("load", 1, 2, "load <file> [type]", "Load
parameter info from file");
```

No.	Parameter	Explanation
1	std::string	command
2	unsigned int minParam	minimum number of parameters the command needs
3	unsigned int maxParam	maximum number of parameters the command needs
4	std::string synopsis	short description of the command
5	std::string description	elaborate description of the command

3-1. Connect to CoWare: TCL script(1)

(4) Create TCL script-1

Step	TCL script example
Load OSCI TLM 2.0 library	open_library $env(COWAREHOME)/IP/TLM2_BL/ConvergenSC/TLM2_BL.xml
Initialize variables	clear_systemc_defines clear_systemc_include_path add_to_systemc_include_path . set_update_existing_encaps_flag true * use "add_to_systemc_include_path" to add include path other than current directory.
Load SystemC models	load_all_modules ./*re_register.cpp* load_all_modules ./*ip00.cpp*
Set model system variables	set lib "SYSTEM_LIBRARY" set blk "*Cip00*"

3-1. Connect to CoWare: TCL script(2)

(4) Create TCL script-2

Step	**TCL script example**
Define port parameters	set port "*intreq_ip00*" set_param_value ${lib}:${blk}/${port} "Port Properties" MasterSlaveness *Master* set_param_value ${lib}:${blk}/${port} "Port Properties" Direction *Out* set_param_value ${lib}:${blk}/${port} "Port Properties" Category *Control* set_param_value ${lib}:${blk}/${port}:${blk} "Protocol Common Parameters" data_width *4* * MasterSlaveness : "Master" if output, "Slave" if input * Direction : "Out" if output, "In" if input * Category : "Clock" if clock, "Control" otherwise * data_width : port width (bit) * repeat this with all ports
Export model system library	add_encap_source_file ${lib}:${blk}/${blk} *./re_regiser.cpp* export_system_library ${blk} *ip00.xml*

3-1. Connect to CoWare: Model System Library

(5) Build model system library

- run pcsh with TCL script to export model system library (XML)
- example with the sample TCL script in (4)

```
source  /common/appl/dotfiles/platform_architect.CSHRC_2010.1.1
pcsh  Create_ip00_lib.tcl
```

* Currently this step cannot be done in RVC,
as RVC does not have PA(Platform Architect) license necessary for this step.

3-2. Connect to VaST: Outline

- This section explains model description and connection steps when we connect OSCI TLM2 model to VaST CoMET environment

- Modify OSCI TLM2 model to:
  - adapt to CoMET environment specification
  - avoid CoMET environment restrictions

- Use "#ifdef VAST ... #endif" area to put CoWare-specific description

(1) Return TLM_OK_RESPONSE for non-fatal access	MUST
(2) One process can write to an sc_port/sc_signal	MUST
(3) Add dummy socket to model with tlm_common	**Recommended**
(4) Refer to tool parameter of .fmx file	**Recommended**
(5) Modify endian settings on big-endian system	MUST

3-2. Connect to VaST: Model Description(1)

(1) Return TLM_OK_RESPONSE for non-fatal access

- VaST master models quit simulation when they receive a response other than TLM_OK_RESPONSE

(2) One process can write to an sc_port/sc_signal

- In SystemC 2.2, if two separate processes write to a signal an error will be reported
- "setenv SC_SIGNAL_WRITE_CHECK disable" does not work in VaST environment (simulation freezes)

(3) Add dummy socket to model with tlm_common

- VaST tool does not automatically recognize tlm_common socket
- Add dummy socket to help the tool connect the model (see next page for details)

3-2. Connect to VaST: Model Description(2)

(3) Add dummy socket to model with tlm_common

step1. Add dummy socket to model

```
class IP00                                                    [IP00.h]
  : public sc_module
  , public tlm::tlm_fw_transport_if<>  /// target dummy
  , public tlm::tlm_bw_transport_if<>  /// initiator dummy
  , public vpcl::tlm_tgt_if<32>  // tlmcommon target
  , public vpcl::tlm_ini_if<32>  // tlmcommon initiator
{
public:
  tlm::tlm_target_socket<32> m_tgt_socket;  /// target
  tlm::tlm_initiator_socket<32> m_ini_socket;  /// initiator
.....
```

step2. Import the model

VaST tool recognizes dummy socket and generate connection description in _config.cpp file.

step3. Disable auto-update of _config.cpp file

Right-click on the imported project -> Properties -> Builders, disable "VaST SystemC Builder", and press OK.

step4. Comment-out dummy socket

Follow usual import steps after this.

<OR>

step1. Add dummy socket to model

Put dummy socket description in "#ifdef COMET63_DUMMY_SOCKET -- #endif" areas

step2. Import the model with macro setting

Macro "COMET63_DUMMY_SOCKET" defined in Advanced Setting -> Preprocessor Defines(-D).

step3. Disable auto-update of _config.cpp file

step4. Remove macro setting

Right-click on the imported project -> Properties, remove COMET63_DUMMY_SOCKET macro

Follow usual import steps after this.

3-2. Connect to VaST: Model Description(3)

(4) Refer to tool parameter of .fmx file

- Model can refer to tool parameter without recompilation
- Example: parameter *MyDebug*

step1. Declare and use parameter in model

```
class IP00 ..... {                    [IP00.h]
public:
  bool  MyDebug;  // declaration
  .....
}
```

```
.....                                 [IP00.cpp]
if (MyDebug) {  // use
  // debug behavior
} else {
  // normal behavior
}
.....
```

step2. Import the model

.fmx and _config.cpp files are generated automatically

step3. Declare and set parameter in .fmx file

Master/Details TAB -> Right-click on Parameters -> Add Parameter, then declare and set *MyDebug* in Parameter Details

step4. Connect parameter in _config.cpp file

Below /**** Generated Section End. ~ ****/.

[*IP00*_config.cpp]

```
.....
 IP00 *pIP00 = new IP00(pInstance);
.....
  /**** Generated Section End. ~ ****/
  pIP00->MyDebug = pParameter->MyDebug;
.....
```

step5. Project->Build All

Parameter change in system .pcx file will be effective in system simulation without recompilation

3-2. Connect to VaST: Model Connection(1)

(5) Modify endian settings on big-endian system

- VaST environment flips the lower bits of address in big-endian system simulation
- Modify endian settings manually
  both two SetEndianMode() lines in osci_tlm2_adapter.cpp

[osci_tlm2_adapter.cpp]

```
.....
///    stdbus_port()->SetEndianMode(VtaEndianLittle);  /// original
    stdbus_port()->SetEndianMode(VtaEndianBigWord32);  /// when 32bit bus
.....
```

3-2. Connect to VaST: Model Connection(2)

- Model Debugging
  - cout(printf) debug is available
  - VC++ debug is available *except* Express edition

- Where to check when simulation freezes after OSCI TLM2 model access
  - out-of-bound access by memcpy in the model : use (address - offset) for memcpy address
  - if two separate processes write to a signal
  - if initiator access is described in the same process with its trigger target access ("Transaction timed out") : use separate processes for target and initiator access

- How to trace bus transaction
  - set parameter in system .pcx file
  - ModuleInstance -> Top -> (VSP) -> (Bus) -> ParameterOverrides ->CheckingAndTracing -> ProtocolTracing to "Enabled"

3-3. Connect to GAIO: Outline

- This section explains connection steps and a recommendation of model description when we connect OSCI TLM2 model to GAIO No.1 simulator environment

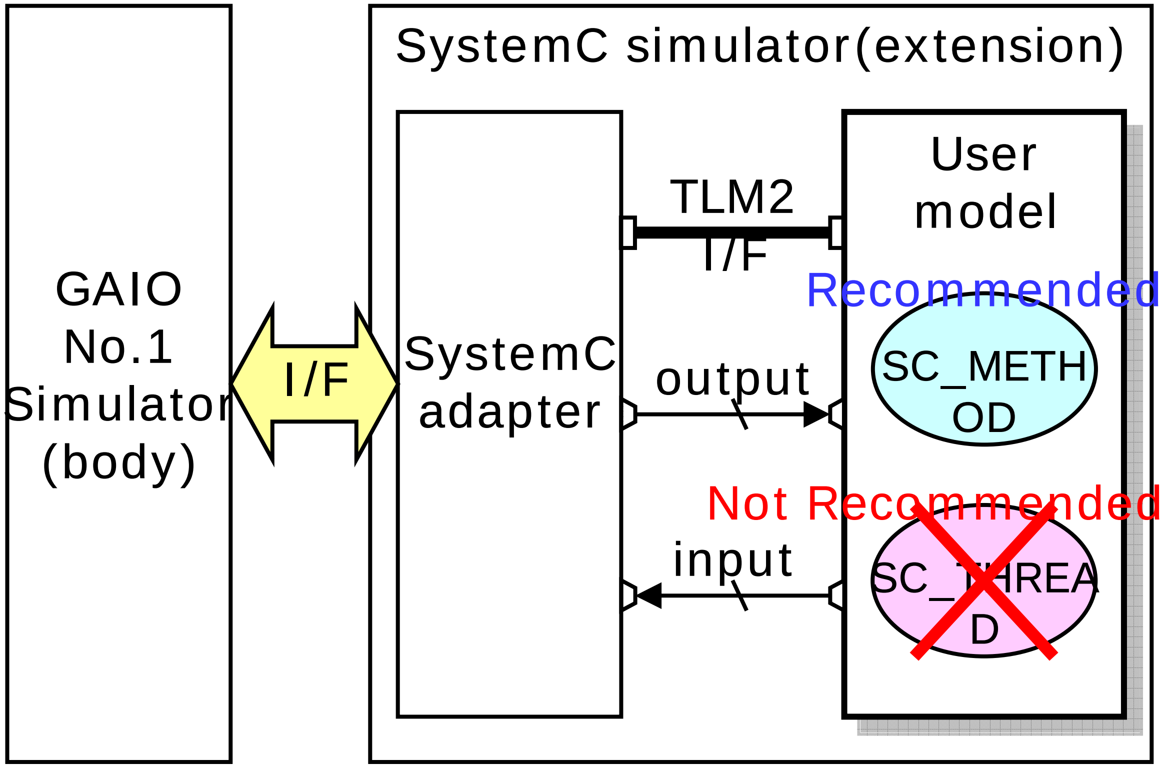
- GAIO No.1 simulator does not automatically connect OSCI TLM2 model; we need to ask GAIO to connect manually

(1) SC_METHOD is preferable for model process **Recommended**

3-3. Connect to GAIO: Model Description

(1) SC_METHOD is preferable for model process

- SC_METHOD runs faster than SC_THREAD in GAIO environment



Connection to GAIO No.1 simulator

```
class Cusr_model : public sc_module
{
public:
  sc_in <sc_uint<8> > in;
  ...

  SC_HAS_PROCESS(Cusr_model);
  Cusr_model(sc_module_ )
  {
    SC_METHOD(method_proc);
    sensitive << in;

  //SC_THREAD(thread_proc);
  //sensitive << in;
  }
private:
  void method_proc(void);
//void thread_proc(void);
};
```

User model example

C Model Design Guide

Chapter 4.
Modeling Methodology

Renesas Electronics Corporation
Front-End Design Technology Development Department

2010/09/21 Rev. 1.2

Chapter 4 Organization

- Organization
  - 4.1 Modeling for Endianness
  - 4.2 Modeling for Efficiency and Speed

4.1 Endianness Introduction

- This guide explains how we should consider endianness in developing models

- Endianness is:
  - how we store multiple-byte data on memory (byte order)
  - two major endian types : little and big
  - Intel x86:little, Sun SPARC:big, Renesas SH/RX:bi-endian
  - example: 4 byte data(0x12345678) on memory address 0x00

4 byte data
(**0x12345678**) ⇔

memory address	00	01	02	03
little endian	78	56	34	12
big endian	12	34	56	78

- Understand and determine endian rules to transfer data correctly among models and tools
  - model : initiator, target, memory, bus, external-I/F, etc.
  - tool : memory load/save, data display, etc.

4.1 Endian Modeling Rules(1)

- The endianness of the real module (modeled endianness)
  - when a model has a multiple-byte port (except OSCI TLM 2.0 bus), the port model should follow the byte-order of the real module
  - if the real module and its model support bi-endian, the model should switch its endian with ***IS_MODELED_ENDIAN_BIG*** macro

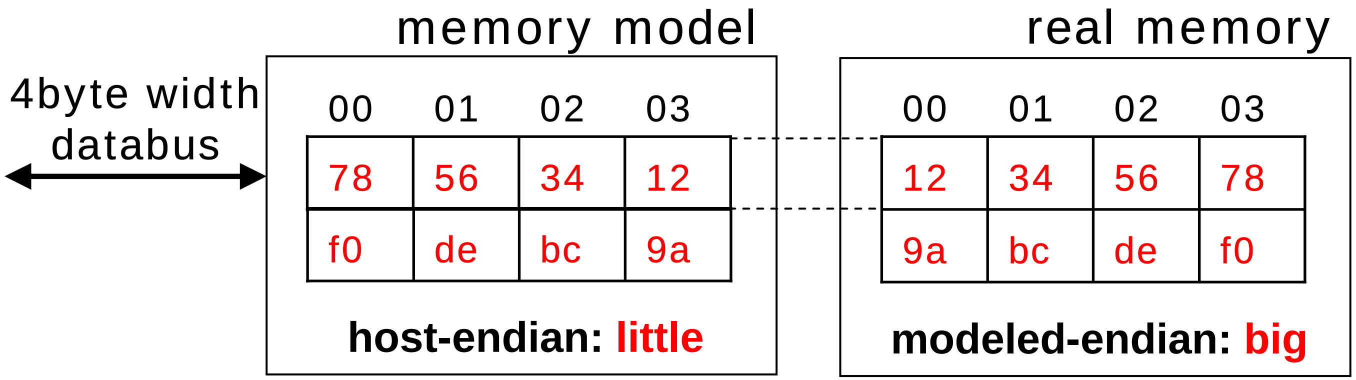
- The endianness of the host machine (host endianness)
  - little if x86, big if SPARC
    - little only host endian support is sufficient, as currently we use models only on x86 EWS/PC
  - memory model specification of CoWare/VaST tools (next page) depends on host endianness

- Match/mismatch of modeled-endian and host-endian
  - refer to "Bus access specification of CoWare/VaST tools" for details

4.1 Endian Modeling Rules(2)

- Memory model specification of CoWare/VaST tools
  - the byte-order within each word shall be **host-endian**
    - does NOT depend on modeled-endian
    - does NOT depend on size of each stored data
    - word size = connected data-bus size(width)

  - Example: 2byte-instruction code data stored in memory
    - starting addr=0 in memory, four instructions 0x1234, 0x5678, 0x9abc, 0xdef0 are stored
    - assumption: modeled-endian=big, host-endian=little



memory model

4byte width databus

00	01	02	03
78	56	34	12
f0	de	bc	9a

host-endian: little

real memory

00	01	02	03
12	34	56	78
9a	bc	de	f0

modeled-endian: big

4.1 Endian Modeling Rules(3)

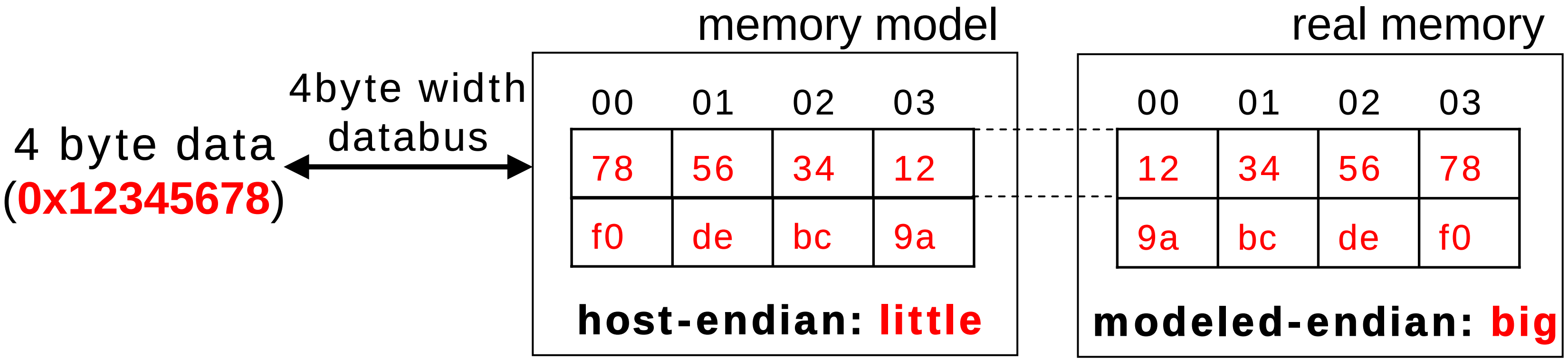
- OSCI TLM 2.0 bus specification (OSCI TLM2.0 User Manual, 6.17):
  - the byte-order within each word shall be **host-endian**
    - the byte-order does NOT depend on modeled-endian
    - word size = connected data-bus size(width)
    - memcpy is available between variable(ex. uint) and data(ex. p_data[]), because these byte-order is same(host-endian)
    - Refer to [Endian modeling code sample] for code sample
  - the address value depends on **match / mismatch** of modeled-endian and host-endian
    - refer to [Bus access specification of CoWare/VaST tools] for details

4.1 Endian Modeling Rules(4)

- Bus access specification of CoWare/VaST tools
  - If modeled-endian and host-endian differ, bus access address should be translated for less-than-word-size bus access
    - lower *n* bits are translated (*n*=log2 word-size-in-byte; 3 if word size is 64bit)
    - invert all *n* bits for byte access, all *n* bits except bit0 for 2byte access, all *n* bits except bit0-1 for 4byte access, etc.
  - In CoWare environment, each model should translate address for less-than-word-size bus access
    - initiator : translate
    - target(register) : reverse-translate
    - target(memory) : reverse-translation NOT necessary
    - bus-bridge(width-change) : reverse-translate and translate
  - In VaST environment, each TLM2 model does nothing because VaST osci-tlm2 adapter can take care of address translation
    - use no-translation setting only for target(memory) adapter

4.1 Endian Modeling Rules(5)

- Bus access specification of CoWare/VaST tools (cont'd)
  - Example : 2byte data read from address=0



4 byte data
(**0x12345678**)

4byte width
databus

memory model

00	01	02	03
78	56	34	12
f0	de	bc	9a

host-endian: little

real memory

00	01	02	03
12	34	56	78
9a	bc	de	f0

modeled-endian: big

– real memory : data=0x1234 is read

– memory model : initiator translates address (0->2) so that data=0x1234 is read

4.1 Endian Modeling Code Sample

4byte data -> memory

```
unsigned int  var = 0x12345678;
unsigned char  p_data[4];

.....
memcpy(p_data, &var, 4);  /* little/big host */
```

```
without memcpy (slow)
for (i=0; i<4; i++) {
  p_data[i] = (var>>(i*8)) & 0xff; /* little host*/
  p_data[3-i] = (var>>(i*8)) & 0xff; /* big host*/
}
```

4byte data(0x12345678)

4byte width databus

		00	01	02	03
host endian	little	78	56	34	12
	big	12	34	56	78

The column headers 00–03 are memory addresses.

memory -> 4byte data

```
unsigned int  var = 0;
unsigned char  p_data[4] =
                    {0xaa, 0xbb, 0xcc, 0xdd};
.....
memcpy(&var, p_data, 4);  /* little/big host */
```

```
without memcpy (slow)
for (i=0; i<4; i++) {
  var += p_data[i]<<(i*8);  /* little host*/
  var += p_data[3-i]<<(i*8);  /* big host*/
}
```

memory	00	01	02	03
	aa	bb	cc	dd

4byte width databus

		4byte data
host endian	little	0xddccbbaa
	big	0xaabbccdd

4.2 Modeling for Efficiency and Speed

- SystemC models are usually used:
  - in early stages of design flow, until real SoC chip comes out
  - to run software which takes too long time in RTL simulation

- Therefore, modeling should follow the policies below:
  - Implement only necessary features for its simulation purpose to reduce development amount
  - Use existing models or prepared common parts such as bus, bus-interface, register or core peripheral models to reduce development amount
  - Model at higher abstraction level than RTL to accelerate simulation speed

4.2 Reducing Development Amount

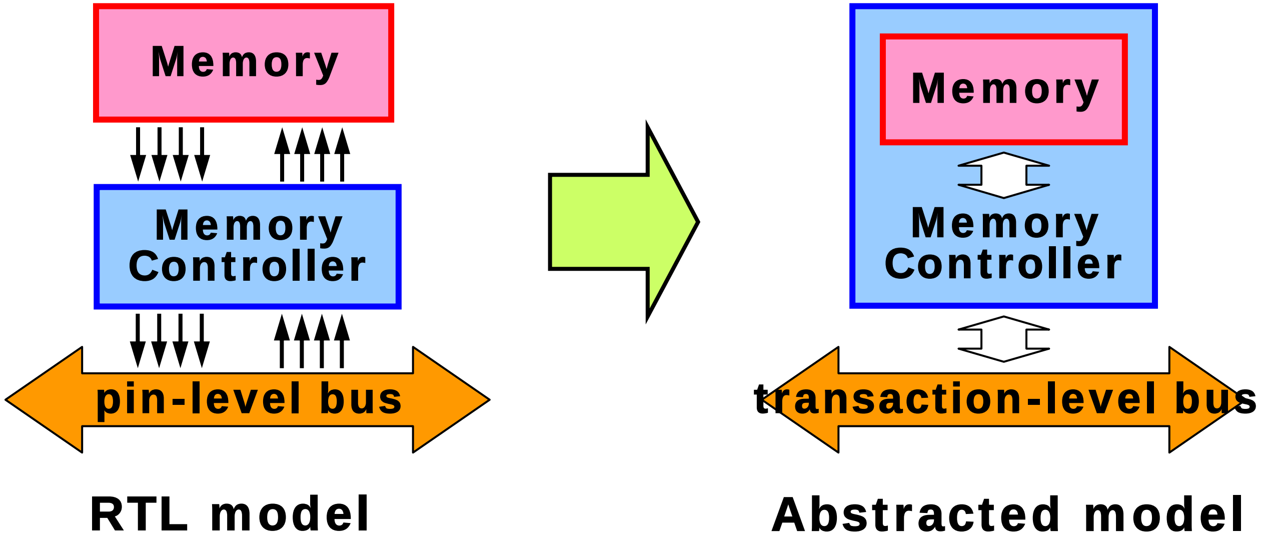
- Implement only necessary features
  - Avoid implementing unnecessary behaviors, registers, ports, ...
    - omit unnecessary behaviors, such as test-mode, error cases, etc.
    - omit unnecessary registers, or use dummy with only read / write behavior
    - omit unnecessary ports, or tie them to fixed values

- Use prepared common parts
  - Bus or bus interface models
  - Register model
  - Core peripheral models
    - e.g. timer, dma, sci, etc.

  -> They are available from “C model library” on the intra-net (Reference [2][3])

4.2 Accelerating Simulation Speed(1)

- Model at higher abstraction level
  - Minimizing number of events is the key for speed-up
  - Cycle accuracy, pin accuracy, structural accuracy are not necessary at all; they are same as RTL and very slow!
    - run multi-cycle-task in one cycle and set cycles forward once
    - use transaction instead of pin-accurate details for communication; or even replace bus-transaction with direct memory access
    - simplify hierarchy



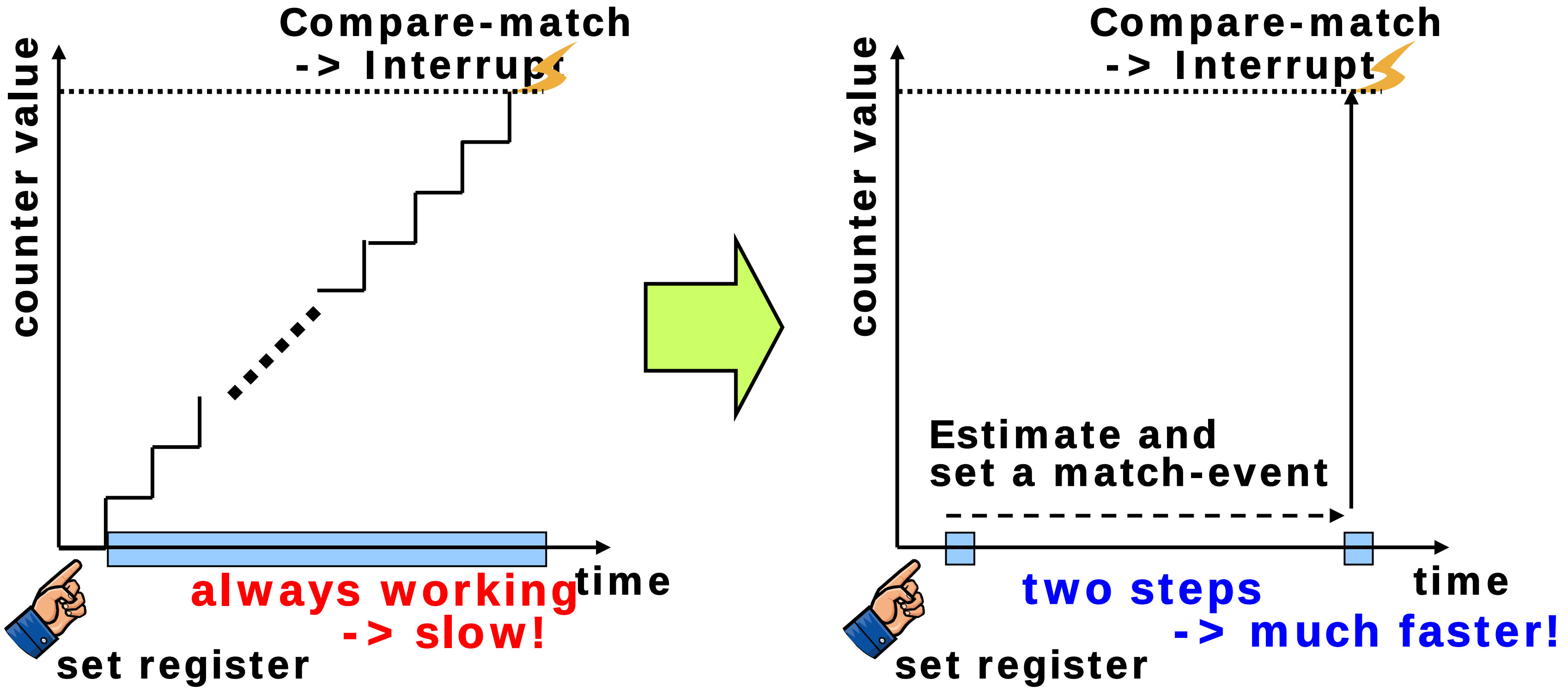
4.2 Accelerating Simulation Speed(2)

- Avoid clock-sensitive threads/methods as much as possible
  - They seriously slow down SystemC simulation
  - Use event-sensitive threads/methods instead
    - ex. register-access sensitive thread

```
void ClassA::xxxThread()
{
  while(true) {
    if( no-job ) {
      wait(TriggerEvent);  // make a thread sleep
    } else {
      do-job;
      wait(Clock.posedge_event());
    }
  }
}
void reg_access()
{
  TriggerEvent.notify();  // wake the thread up
}
```

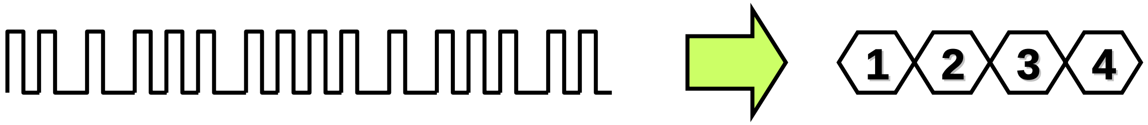
4.2 Modeling Examples(1)

- TIMER module
  - TIMER has counters which are incremented every cycle
    -> do NOT increment counters using clock signals
    - instead, set an alarm (compare-match-event) at the estimated time
    - calculate value of the counter only when it's necessary, i.e. it is accessed



4.2 Modeling Examples(2)

- DMAC (direct memory access controller) module
  - Do NOT precisely model read / write behaviors
  - Collect as many data as possible and transfer them together
    - e.g. 4 bytes x 1024 times -> 4096 bytes x 1 time
  - Set a completion-event at estimated time
- SCI (serial communication interface) module
  - Do NOT precisely model serial communication including start / parity bit, and instead, transfer byte / block-wise data
  - Set a completion-event at estimated time



- Data processing module
  - Do NOT precisely model I/O events or processing behavior
  - Collect as many data as possible to read / write data
  - Execute a series of process in one cycle as large as possible
  - Set a completion-event at estimated time

【《 RENESAS Group CONFIDENTIAL 》】

C Model Design Guide

Appendix.
References

Renesas Electronics Corporation
Front-End Design Technology Development Department

2010/09/21 Rev. 1.2

Related References and Resources

[1] C++/SystemC Coding Rule

http://www.hoku.renesas.com/EDA/tools/data/lv1ww/manual/manual_1710.pdf

[2] C model library (IP Gear 2.1)

http://libg4.mu.renesas.com:8101/scripts/isynch.dll

[3] Common Parts User Guide

(You can download from C model library)

Feedback Request

- Please direct feedback to:

FEgikai/M.Masuda	(masayuki.masuda.gx@renesas.com)
FEgikai/T.Asano	(tetsuya.asano.yj@renesas.com)
FEgikai/S.Hanaki	(shoichi.hanaki.pd@renesas.com)

Revision History

Rev.No	Contents	Approval	Checked	Created
1.1	Translated from Japanese edition Rev.1.1 Updated 3.1 (Connect to CoWare)	Y. Takamine '10/5/26	S.Hanaki '10/05/26	M.Masuda T.Asano '10/05/26
1.2	Translated from Japanese edition Rev.1.2 – Updated 3.1 (Connect to CoWare) – Updated 4.1 (Endianness) – Other explanation updates	Y. Takamine '10/9/24	S.Hanaki '10/9/24	T.Asano '10/9/21

Renesas Electronics Corporation