Everywhere you imagine.

Front End Design Group

System Level Design Team

Internal Training

TLM 2.0 Library

Renesas Design Viet Nam Co., Ltd.

Design Engineering Division

July 2009

v1.2

Le Nguyen Khang

Confidential

Rev. 1.2

Outline

1. Transaction Level Modeling (TLM)
2. Objects in TLM2.0
3. TLM2.0 library
4. Connection
5. Communication
6. TLM2.0 with Forest
7. Topics

Outline

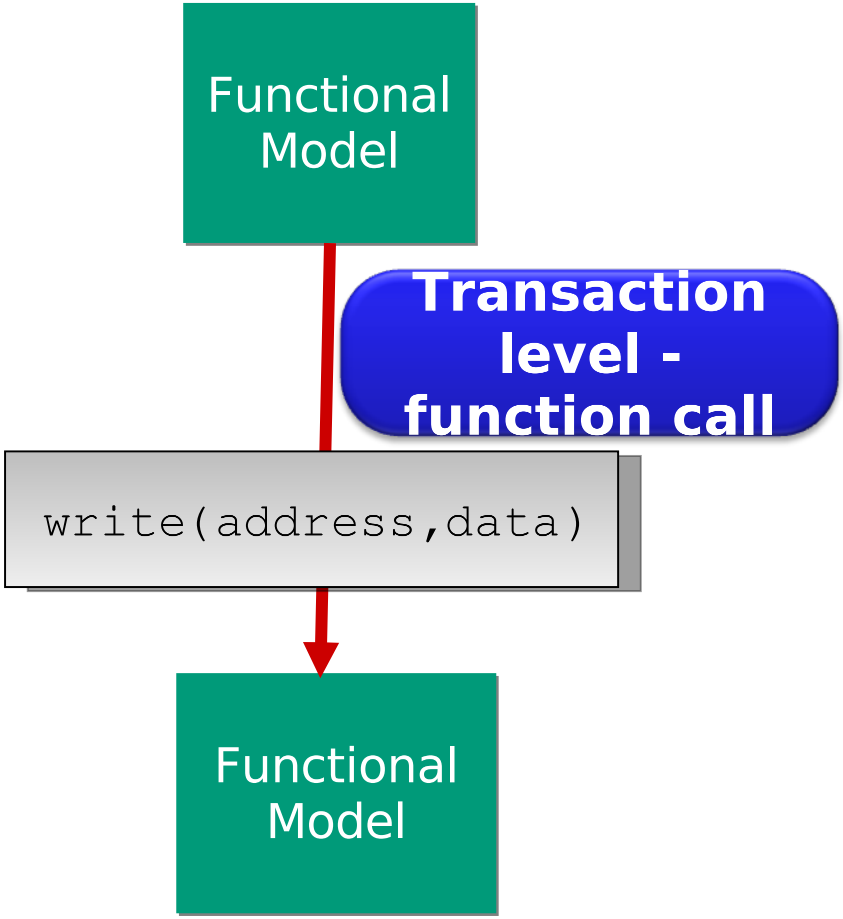
1. Transaction Level Modeling (TLM)
2. Objects in TLM2.0
3. TLM2.0 library
4. Connection
5. Communication
6. TLM2.0 with Forest
7. Topics

1. Transaction Level Modeling (TLM) (1/3)

TLM is a concept without precise definition. A working group of **Open SystemC Initiative** (OSCI) is currently defining a set of terminology for TLM and developing TLM standards.

TLM is used to solve these problems:

- Providing an early platform for software development.
- Aiding software/hardware integration.
- Enabling software performance analysis.
- System Level Design architecture analysis.
- Functional hardware verification.



1. Transaction Level Modeling (TLM) (2/3)

Reasons of using TLM

- Architectural exploration, performance modelling
- Software execution on virtual model of hardware platform
- Golden model for hardware functional verification
- Available before RTL

Earlier

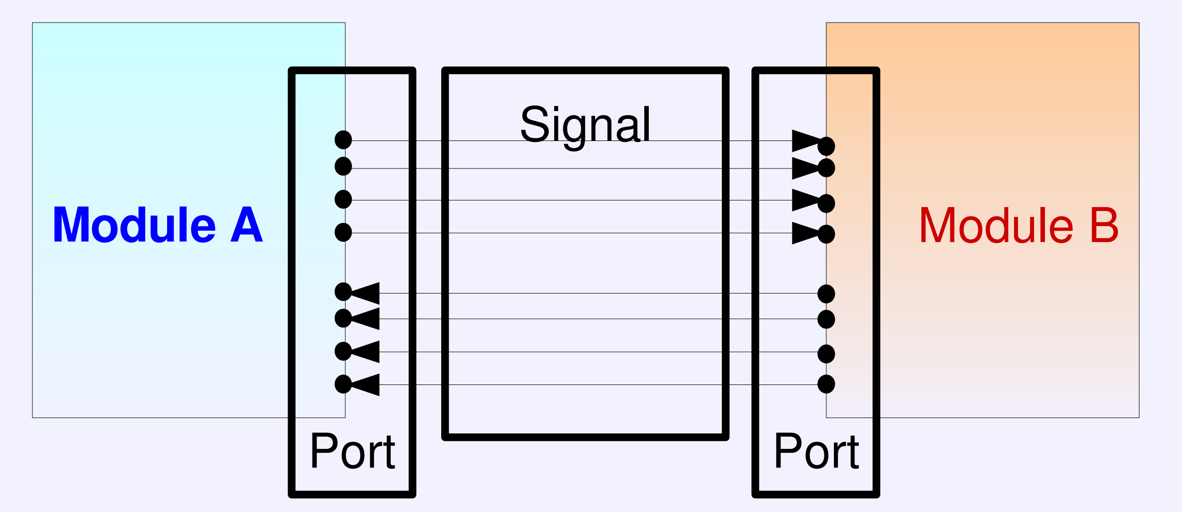
- Simulation much faster than RTL (100-10.000 times)

Faster

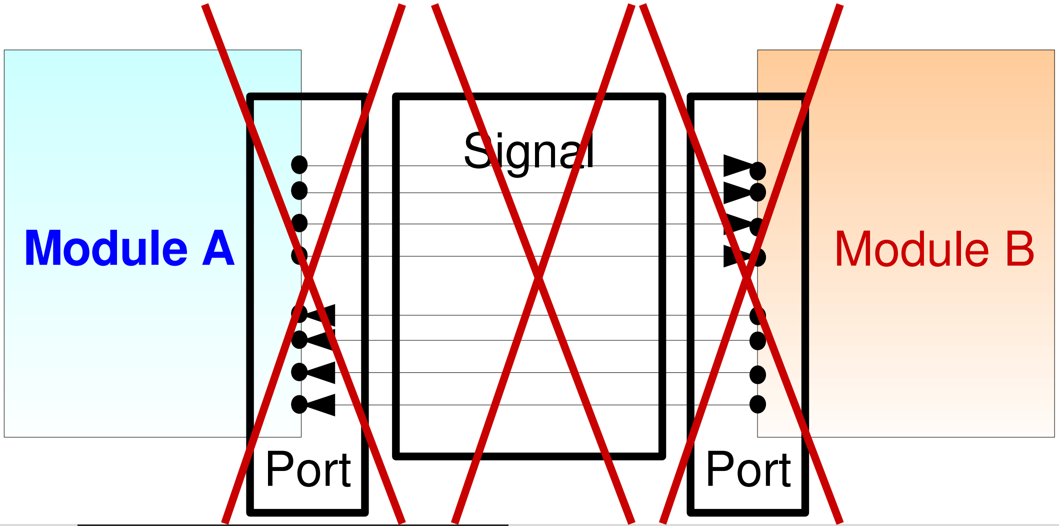
TLM 2.0

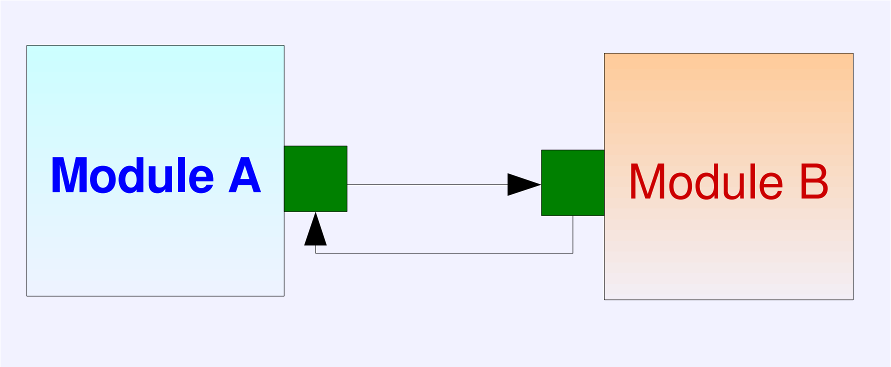
- A library which built on SystemC library
- Consists of a set of core interfaces, objects and base protocol, and utilities to enable TLM concept

1. Transaction Level Modeling (TLM) (3/3)



Without Applying TLM2.0



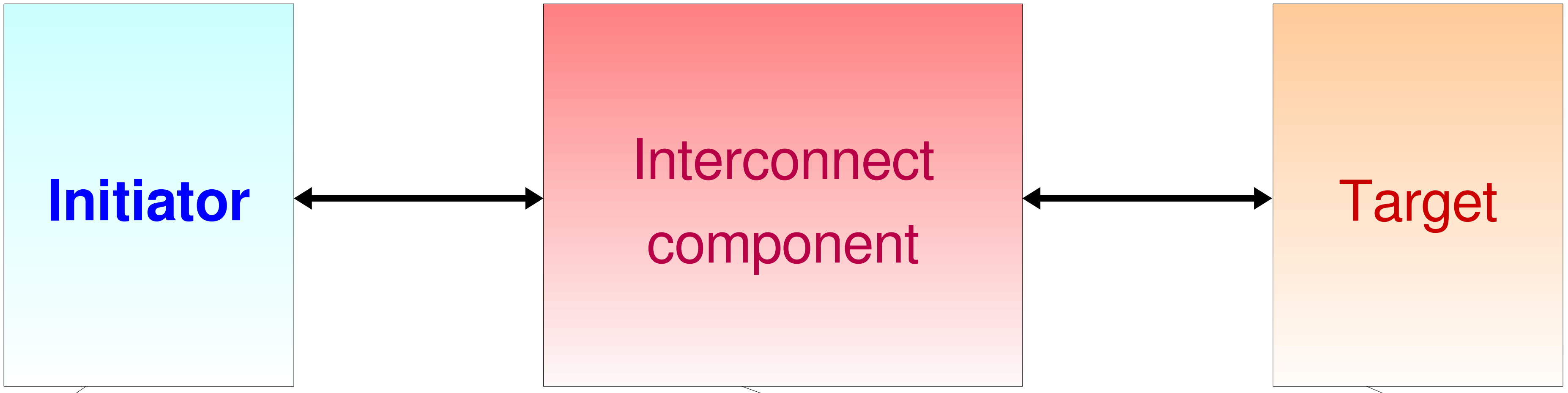


Applying TLM2.0

Outline

1. Transaction Level Modeling (TLM)
2. Basic Objects in TLM2.0
   - 2.1. Initiator/Target
   - 2.2. Socket
   - 2.3. Path
   - 2.4. Interface
3. TLM2.0 library
4. Connection
5. Communication
6. TLM2.0 with Forest
7. Topics

2.1 Initiator/Target



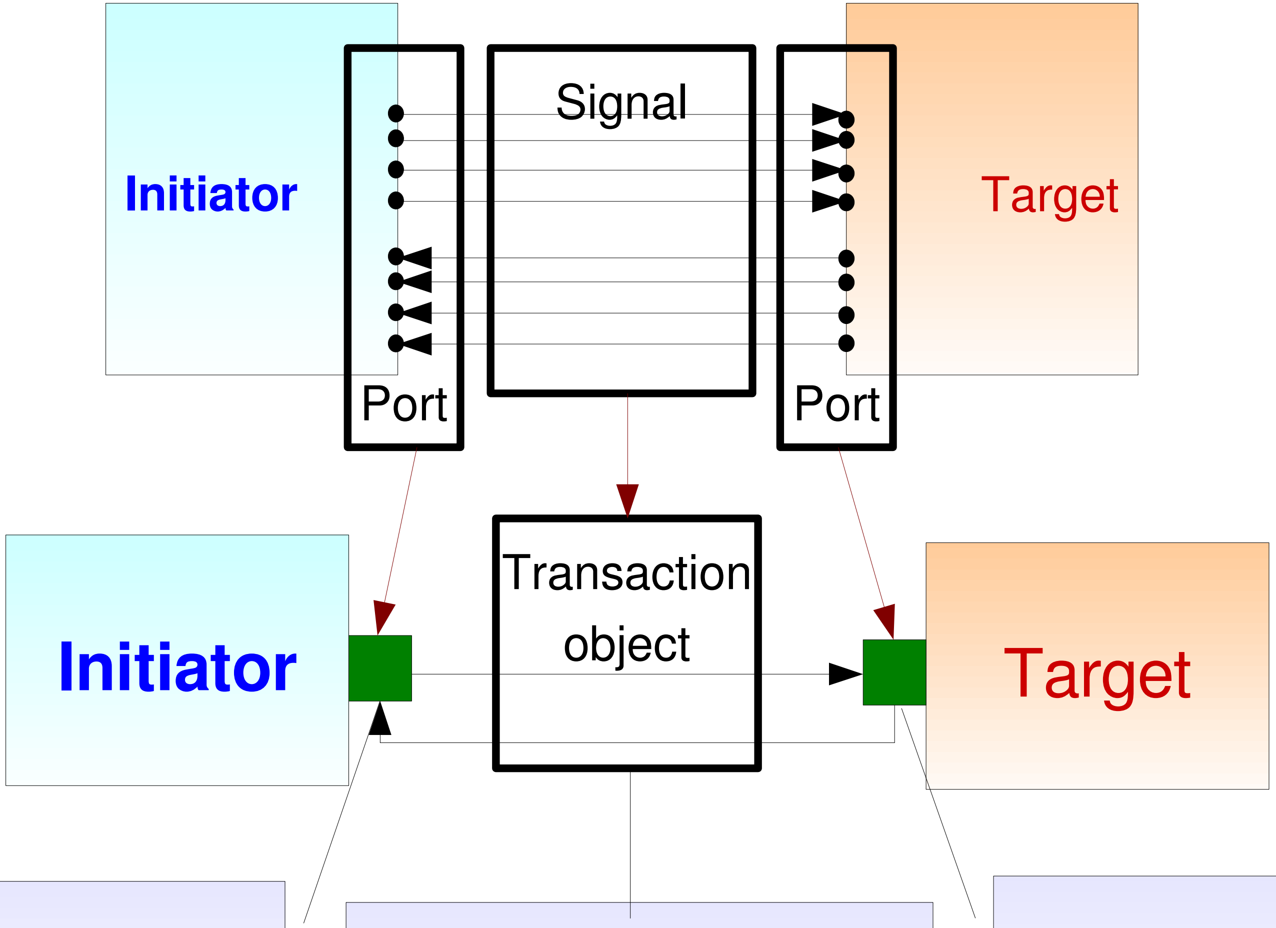
Module that can **initiate** transactions, create transaction objects

Module that accesses a transaction but does **act as an initiator or a target** for that transaction

Module that acts as the **final destination** for a transaction

A module applied TLM2.0 must be the one of three above types. DON'T apply TLM2.0 if you don't know this important information

2.2 Socket

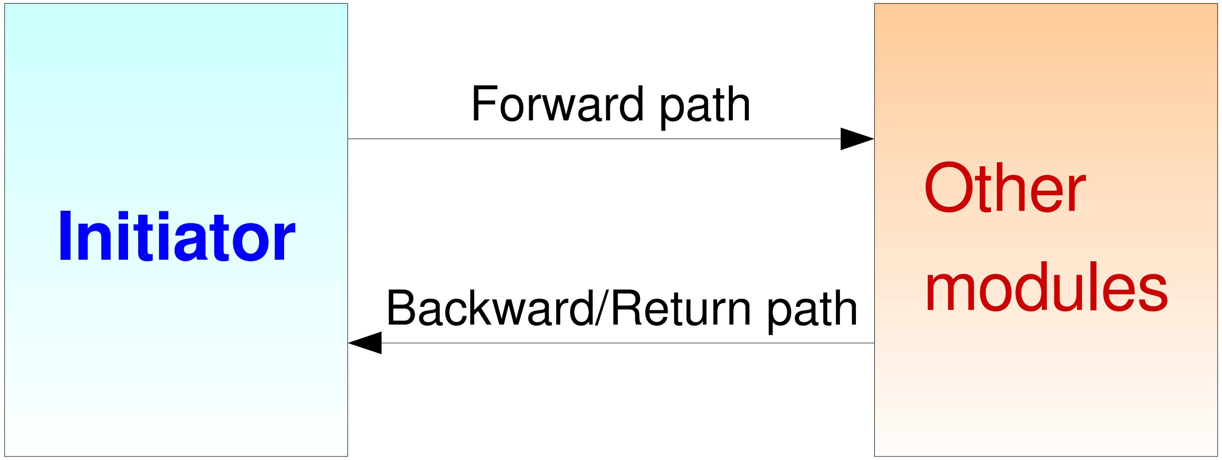


Initiator socket: connect initiator module to others

Stores Information transferred among modules

Target socket: connect Target module to others

2.3 Path



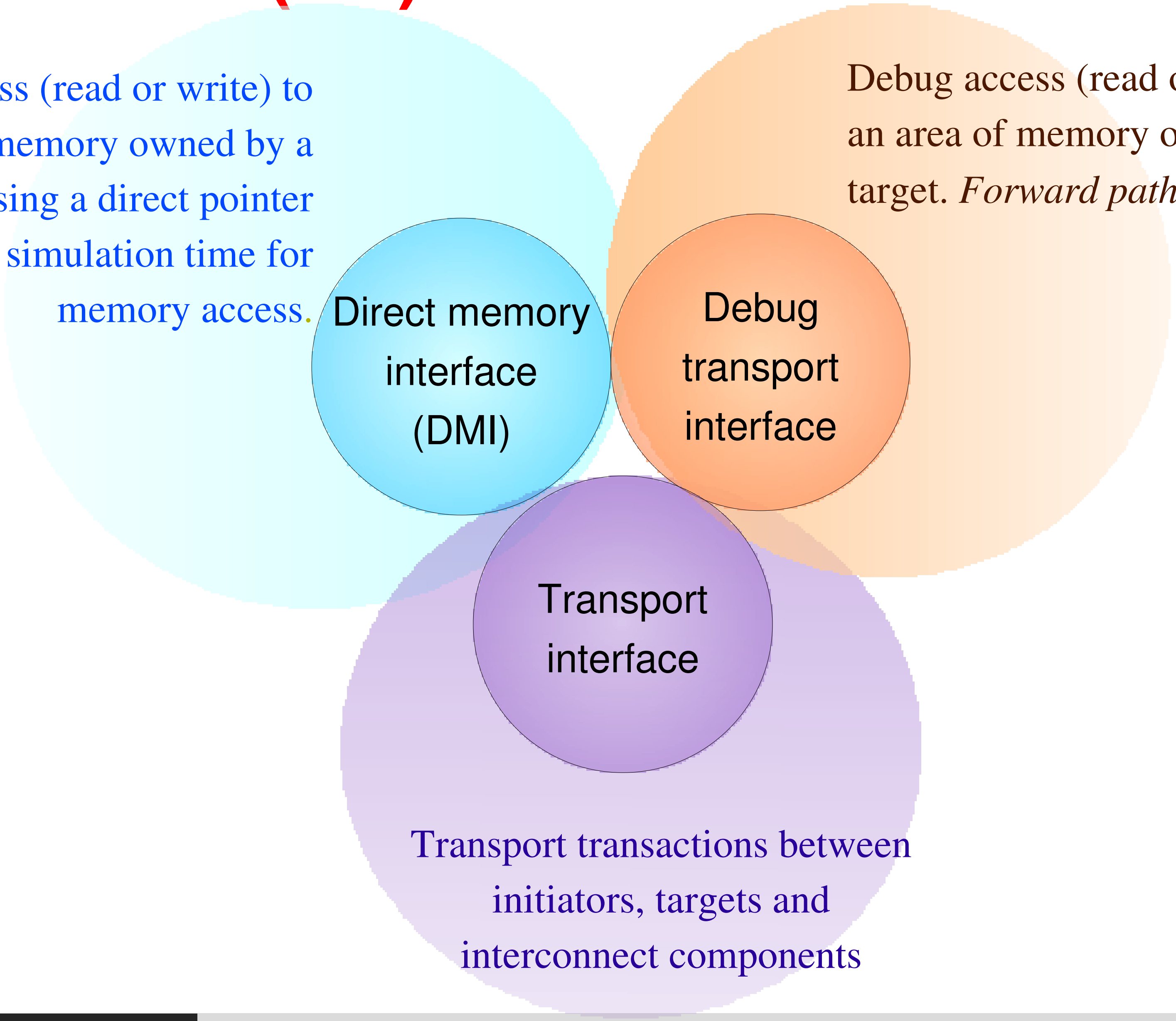
Forward path: transaction object is created by an initiator and passed to other modules

Backward/Return path: transaction object is returned to Initiator by two ways:

1. ***Return path***: Transaction object is returned automatically. Other modules don't send transaction object back to Initiator by calling certain methods

2. ***Backward path***: Other modules send transaction object back to Initiator by calling certain methods

2.3 Interfaces (1/3)

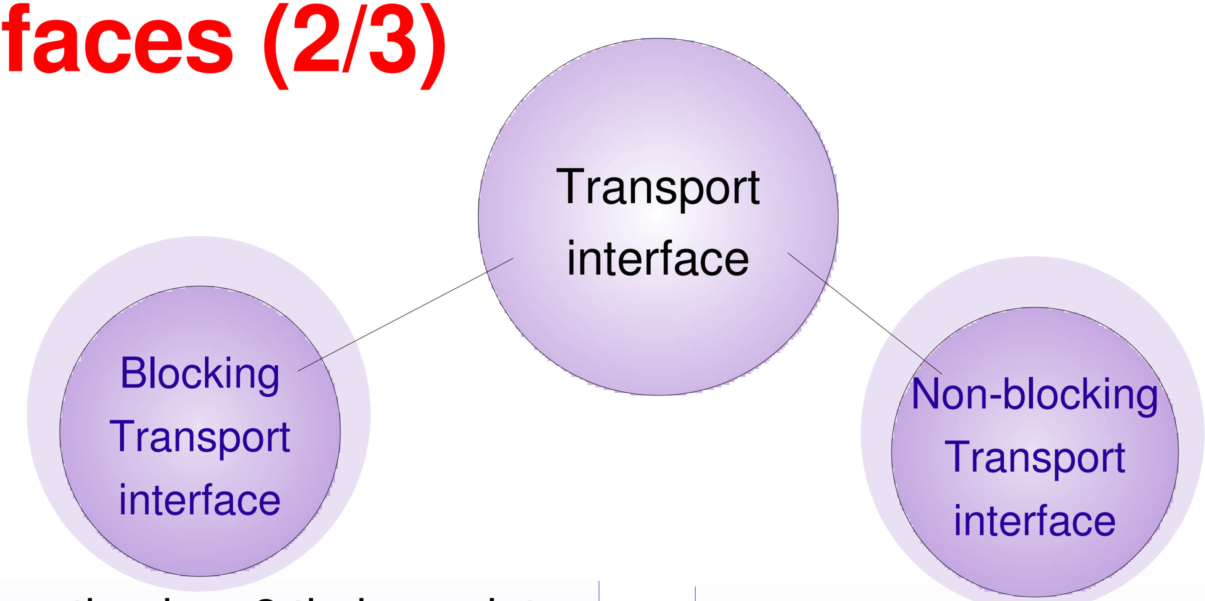


- Direct access (read or write) to an area of memory owned by a target by using a direct pointer
- Speed up simulation time for memory access.

Debug access (read or write) to an area of memory owned by a target. *Forward path* only

Transport transactions between initiators, targets and interconnect components

2.3 Interfaces (2/3)



Transport interface

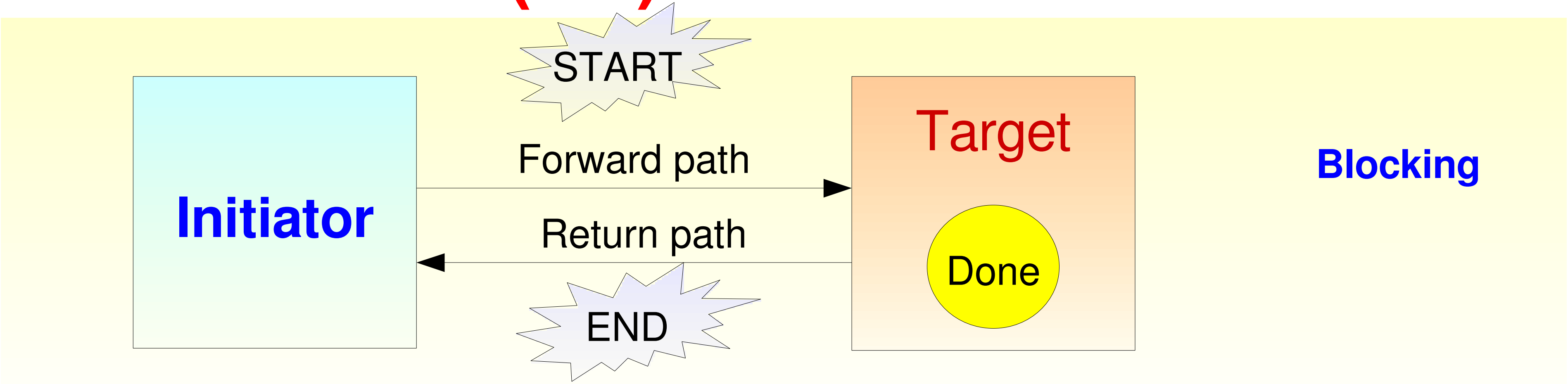
Blocking Transport interface

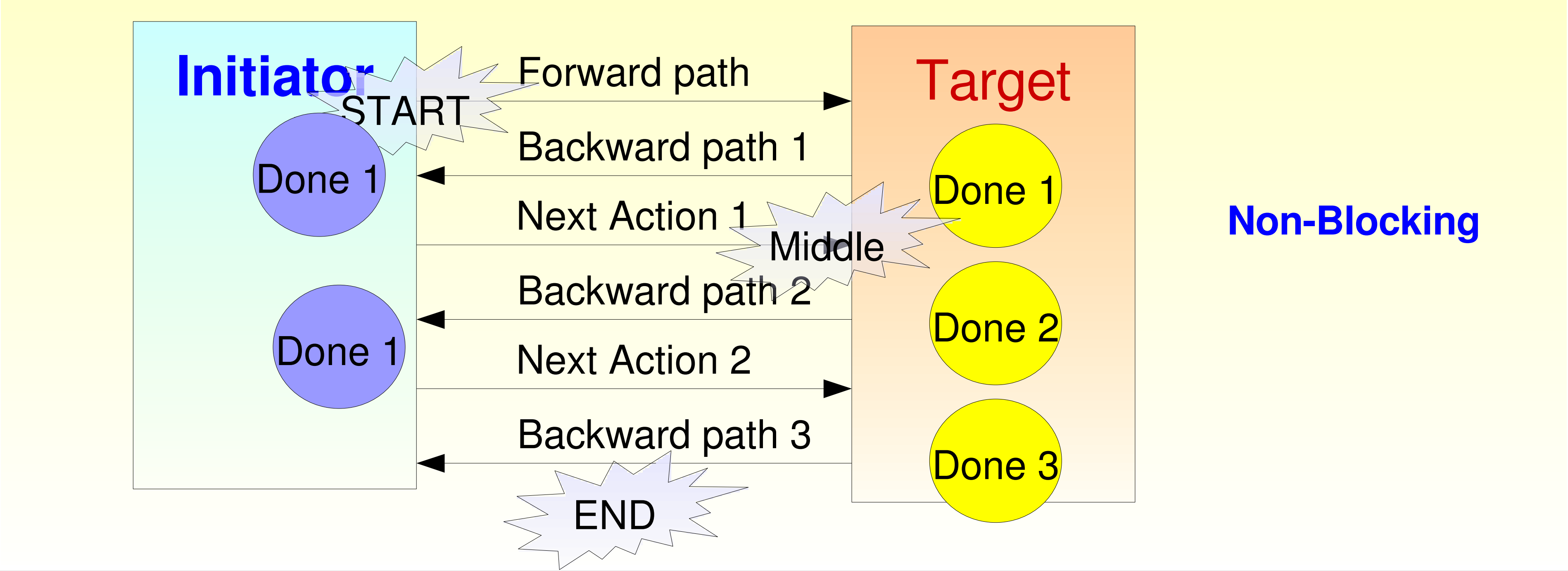
Non-blocking Transport interface

- Each transaction has 2 timing points: START and END of a transaction
- Use Forward path and Return Path only
- “Blocking”: Initiator wishes to complete a transaction with a target by a single call

- Each transaction has multiple timing points
- Use Forward path, Backward/Return Path
- Express the **detailed sequence of interactions** between Initiator and target.
- “Non-Blocking”: transaction is finished through multi calls or single call

2.3 Interfaces (3/3)

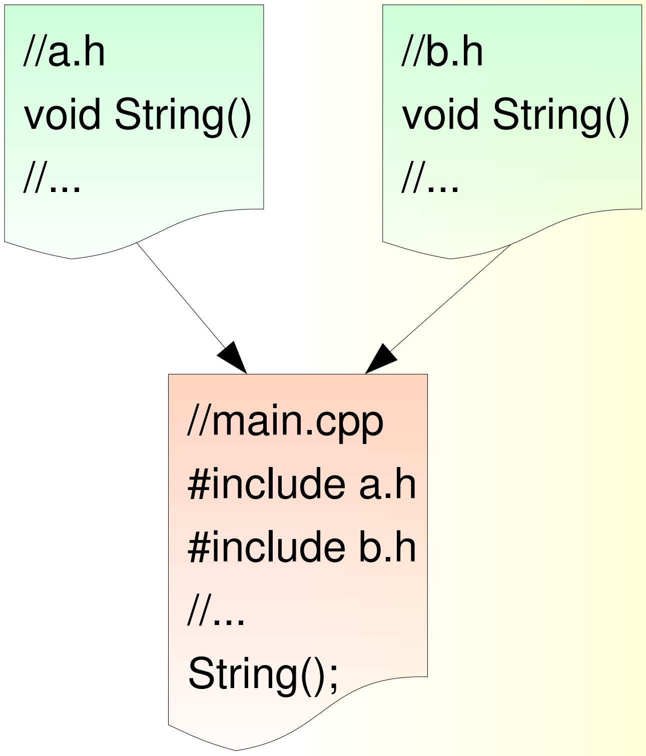




Outline

1. Transaction Level Modeling (TLM)
2. Basic Objects in TLM2.0
3. TLM2.0 library
    - 3.1. Overview
    - 3.2. Some main classes in TLM2.0
    - 3.2. Transport interface classes
    - 3.3. Forward/backward interfaces
    - 3.4. Socket classes
4. Connection
5. Communication
6. TLM2.0 with Forest
7. Topics

3.1 Overview (1/2)

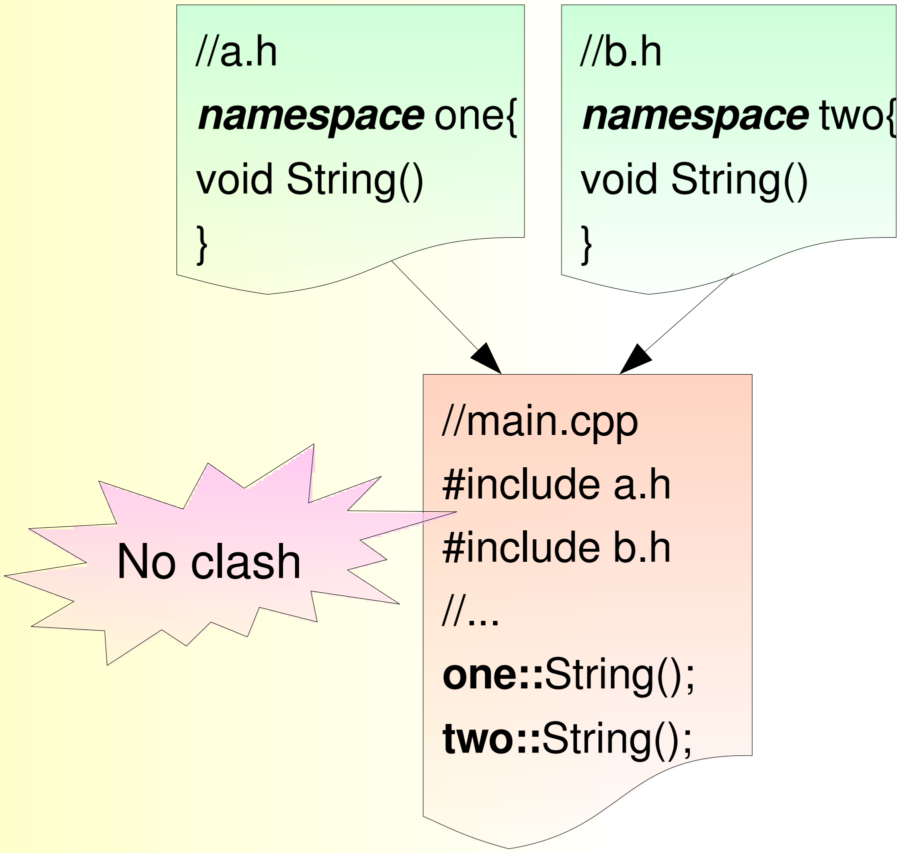


//a.h
void String()
//...

//b.h
void String()
//...

//main.cpp
#include a.h
#include b.h
//...
String();

The String() functions clash
=> It is impossible to use both header files in a single program



//a.h
namespace one{
void String()
}

//b.h
namespace two{
void String()
}

//main.cpp
#include a.h
#include b.h
//...
one:: String();
two:: String();

A ***namespace*** is a 'region' that attaches an additional identifier to any names declared inside it

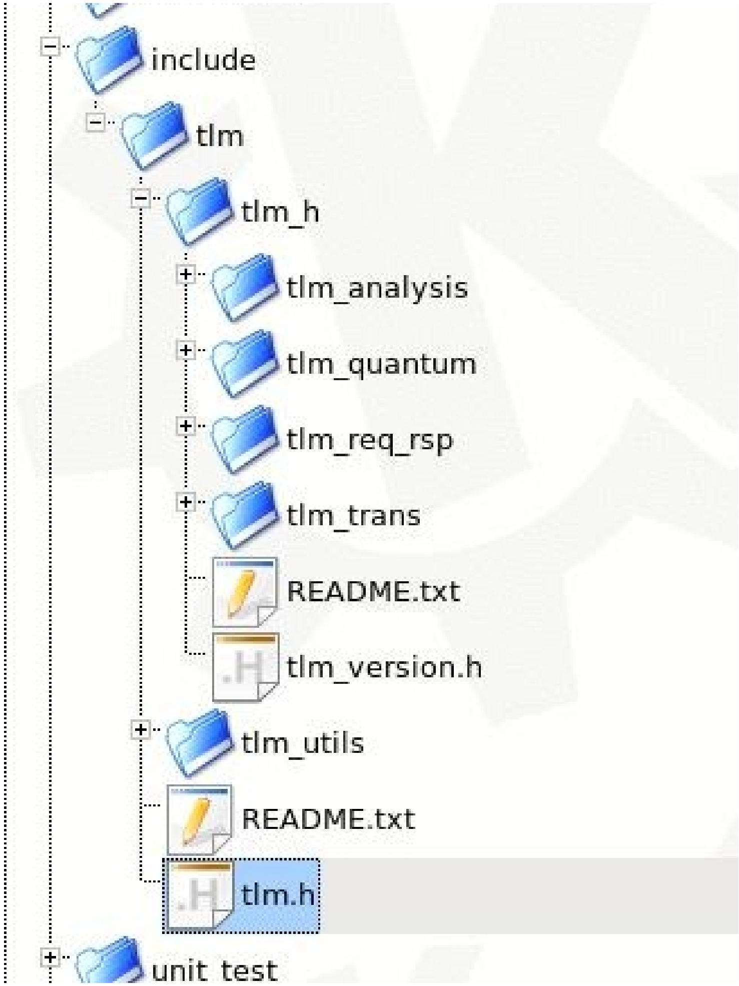
3.1 Overview (2/2)

TLM2.0 library includes many header files (.h). Basically, it consists of two main parts corresponding to two top-level C++ namespaces, **tlm** and **tlm_utils**.

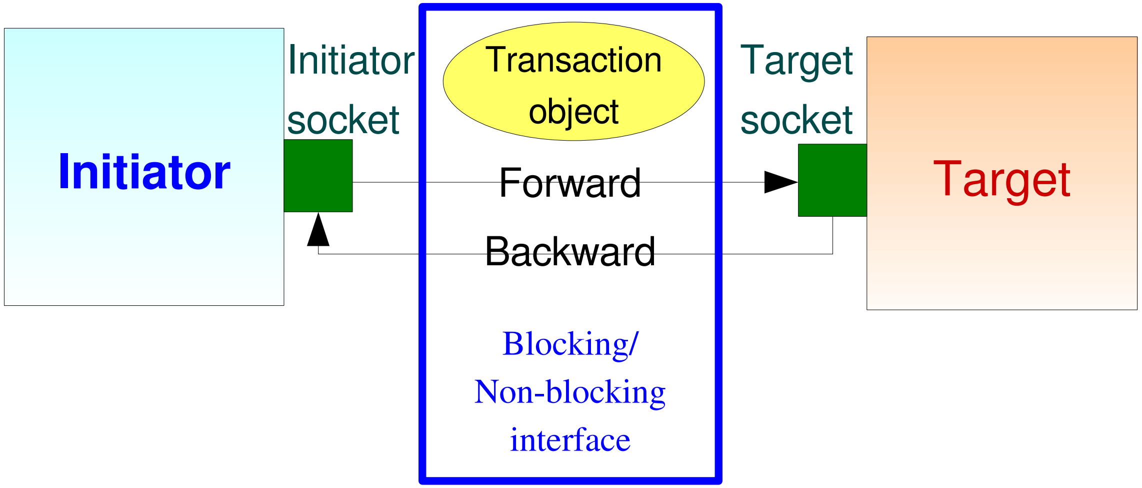
Namespace **tlm**: contains the core classes

Namespace **tlm_utils**: contains the additional classes that inherit from core classes for some certain purposes. It helps users use TLM2.0 more easily

*All classes mentioned in this presentation are in **tlm** namespace*

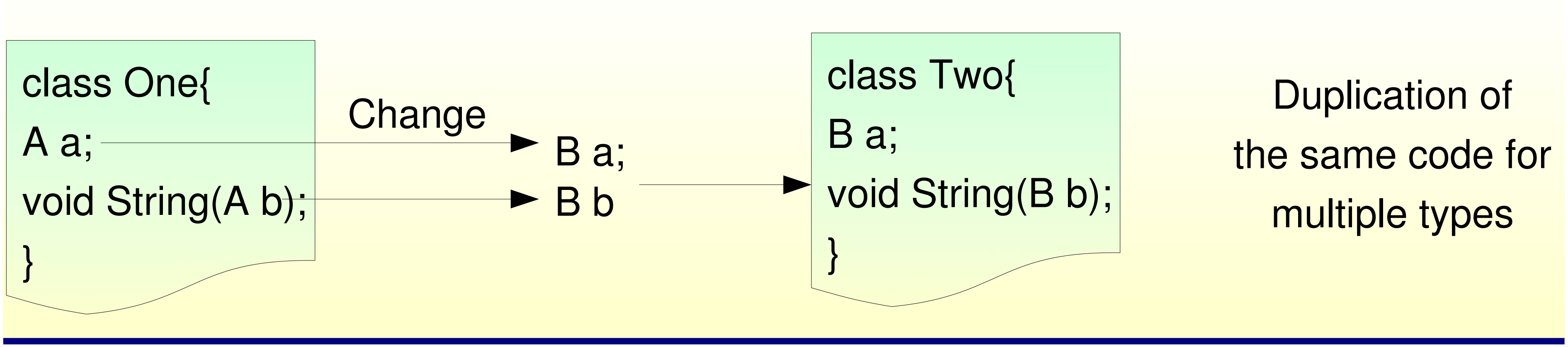


3.2 Some main classes in TLM2.0



- Initiator socket: ***tlm_initiator_socket*** class / Target socket: ***tlm_target_socket*** class
- Transaction object: ***tlm_generic_payload*** class (generic payload object)
- Blocking transport interface: ***tlm_blocking_transport_if*** class
- Non-blocking transport interface: ***tlm_fw_nonblocking_transport_if*** class
  tlm_bw_nonblocking_transport_if class
- Forward transport interface: ***tlm_fw_transport_if*** class
- Backward transport interface: ***tlm_bw_transport_if*** class

3.3 Transport interface classes (1/3)



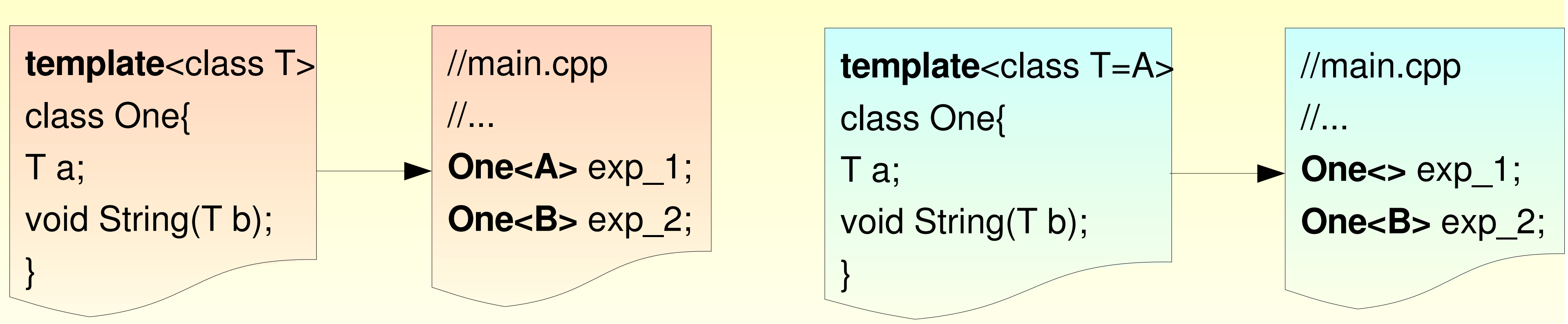
```
class One{
A a;
void String(A b);
}
```

Change

```
B a;
B b
```

```
class Two{
B a;
void String(B b);
}
```

Duplication of
the same code for
multiple types



```
template<class T>
class One{
T a;
void String(T b);
}
```

```
//main.cpp
//...
One<A> exp_1;
One<B> exp_2;
```

```
template<class T=A>
class One{
T a;
void String(T b);
}
```

```
//main.cpp
//...
One<> exp_1;
One<B> exp_2;
```

C++ templates enable you to define a family of functions or classes that can operate on different types of information

3.3 Transport interface classes (2/3)

```
template <typename TRANS = tlm_generic_payload>
class tlm_blocking_transport_if  : public virtual sc_core::sc_interface {
public:
virtual void b_transport (TRANS& trans, sc_core::sc_time& t) = 0;
};
```

Blocking transport interface

Non-Blocking transport interface

```
template <typename TRANS = tlm_generic_payload, typename PHASE = tlm_phase>
class tlm_fw_nonblocking_transport_if  : public virtual sc_core::sc_interface {
public:
virtual tlm_sync_enum nb_transport_fw (TRANS& trans, PHASE& phase, sc_core::sc_time& t) = 0;
};

template <typename TRANS = tlm_generic_payload, typename PHASE = tlm_phase>
 class tlm_bw_nonblocking_transport_if  : public virtual sc_core::sc_interface {
public:
virtual tlm_sync_enum nb_transport_bw (TRANS& trans, PHASE& phase, sc_core::sc_time& t) = 0;
```

3.3 Transport interface classes (3/3)

tlm_phase: this is a class that shows PHASE information between initiator and target. This class contains tlm_phase_enum enum to indicate phases

```
 enum tlm_phase_enum {
        UNINITIALIZED_PHASE=0,
        BEGIN_REQ=1,
        END_REQ,
        BEGIN_RESP,
        END_RESP
};
```

tlm_sync_enum: used for synchronizing between Initiator and Target

```
enum tlm_sync_enum { TLM_ACCEPTED, TLM_UPDATED, TLM_COMPLETED };
```

TLM_ACCEPTED: the call has been accepted

TLM_UPDATED: The transaction object has been updated

TLM_COMPLETED: The transaction object has been updated, and the transaction is complete

3.4 Forward/Backward interface

Revised in v1.2

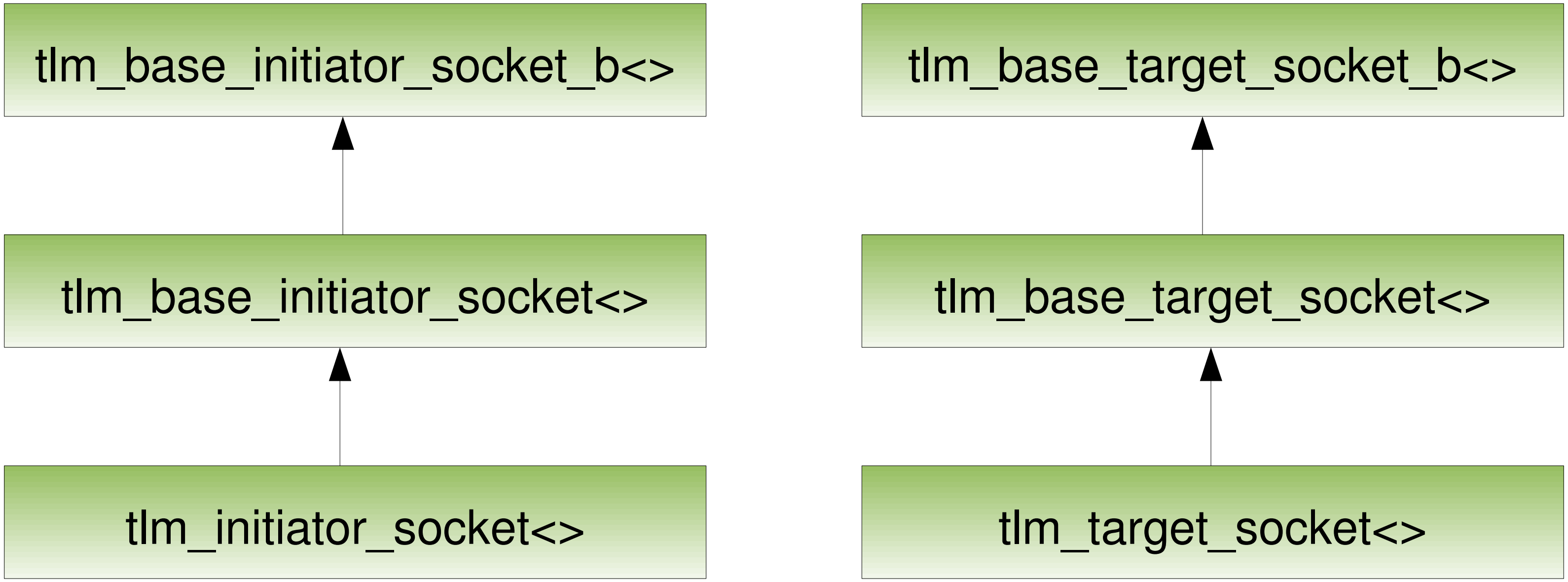
Forward interface

```
class tlm_fw_transport_if:
        public virtual tlm_fw_nonblocking_transport_if<typename TYPES::tlm_payload_type,
                                                        typename TYPES::tlm_phase_type>
      , public virtual tlm_blocking_transport_if<typename TYPES::tlm_payload_type>
      , public virtual tlm_fw_direct_mem_if<typename TYPES::tlm_payload_type>
      , public virtual tlm_transport_dbg_if<typename TYPES::tlm_payload_type>
{};
```

Backward interface

```
class tlm_bw_transport_if:
        public virtual tlm_bw_nonblocking_transport_if<typename TYPES::tlm_payload_type,
                                                        typename TYPES::tlm_phase_type>
      , public virtual tlm_bw_direct_mem_if
{};
```

3.4 Socket classes (1/2)


tlm_base_initiator_socket_b<>
tlm_base_initiator_socket<>
tlm_initiator_socket<>
tlm_base_target_socket_b<>
tlm_base_target_socket<>
tlm_target_socket<>

We use **tlm_initiator_socket** and **tlm_target socket**, but the main functions of socket are defined in **tlm_base_initiator_socket** and **tlm_base_target_socket**

3.4 Socket classes (2/2)

Revised in v1.2

The main methods: bind and operation() are to connect socket to another socket and backward/forward interface

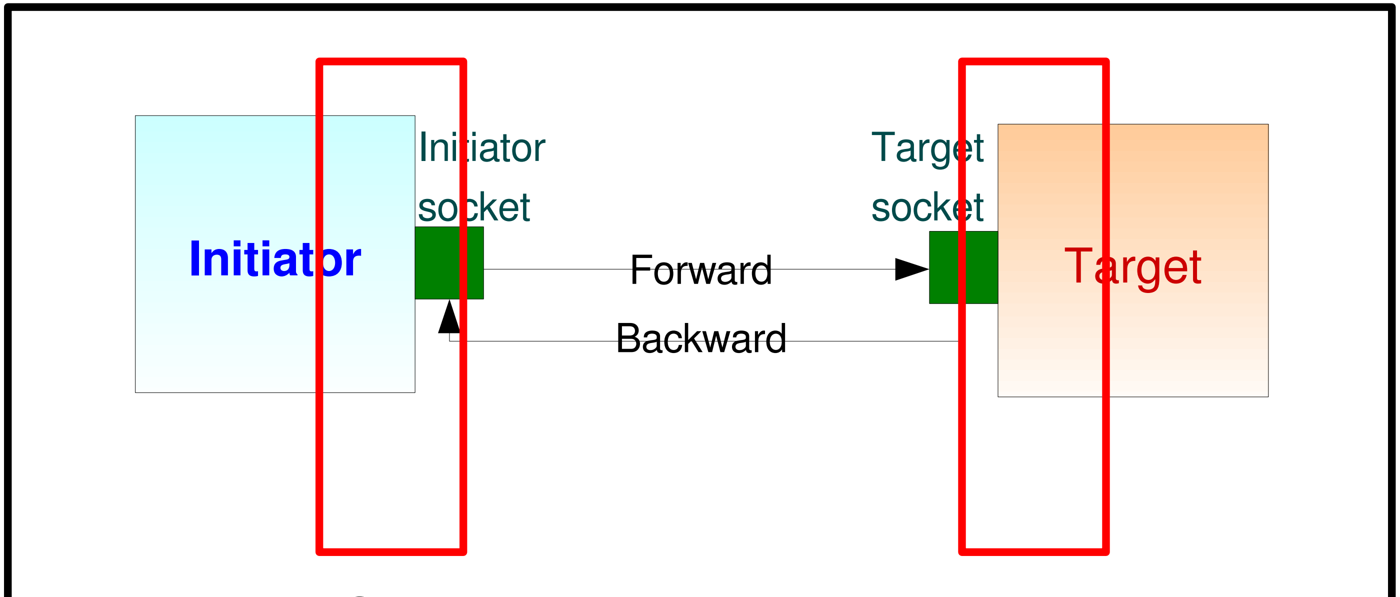
Initiator socket class	Target socket class
void **bind** (tlm_base_target_socket_b & s);	void **bind** (tlm_base_initiator_socket_b & s);
void **operator()** (tlm_base_target_socket_b & s);	void **operator()** (tlm_base_initiator_socket_b & s);
void **bind** (tlm_base_initiator_socket_b& s);	void **bind** (tlm_base_target_socket_b& s);
void **operator()** (tlm_base_initiator_socket_b& s);	void **operator()** (tlm_base_target_socket_b& s);
void **bind** (tlm_bw_transport_if& ifs);	void **bind** (tlm_fw_transport_if& ifs);
void **operator()** (tlm_bw_transport_if& s);	void **operator()** (tlm_fw_transport_if& s);

Outline

1. Transaction Level Modeling (TLM)
2. Basic Objects in TLM2.0
3. TLM2.0 library
4. Connection
    4.1. Module and socket
    4.2. Socket and another socket
5. Communication
6. TLM2.0 with Forest
7. Topics

4. Connection

Revised in v1.2



Step 1: connect Module with socket

4.1 Module and socket (1/3)

Step 1: Choose Path type

Based on **Destination point**.

Step 2: Declare module with chosen path

Inherit from ***tlm::tlm_bw_transport_if<>*** for Backward/Return path

Inherit from ***tlm::tlm_fw_transport_if<>*** for Forward path

Step 3: Declare corresponding socket

tlm::tlm_initiator_socket<> for initiator socket

tlm::tlm_target_socket<> for target socket

Step 4: Connect module to socket

Using operator () in constructor of module

4.1 Module and socket (2/3)

Revised in v1.2

Example:

HPB Master (Initiator) <-> initiator socket

This module is Initiator -> Destination of Backward path

```
class Chpbc: public sc_module, public tlm::tlm_bw_transport_if<>{
        //Declare Initiator socket
        tlm::tlm_initiator_socket<> ini_socket;
        SC_CTOR(Chpbc){
                ini_socket(*this);      //Connect initiator socket to module
        }
}
```

4.1 Module and socket (3/3)

Example:

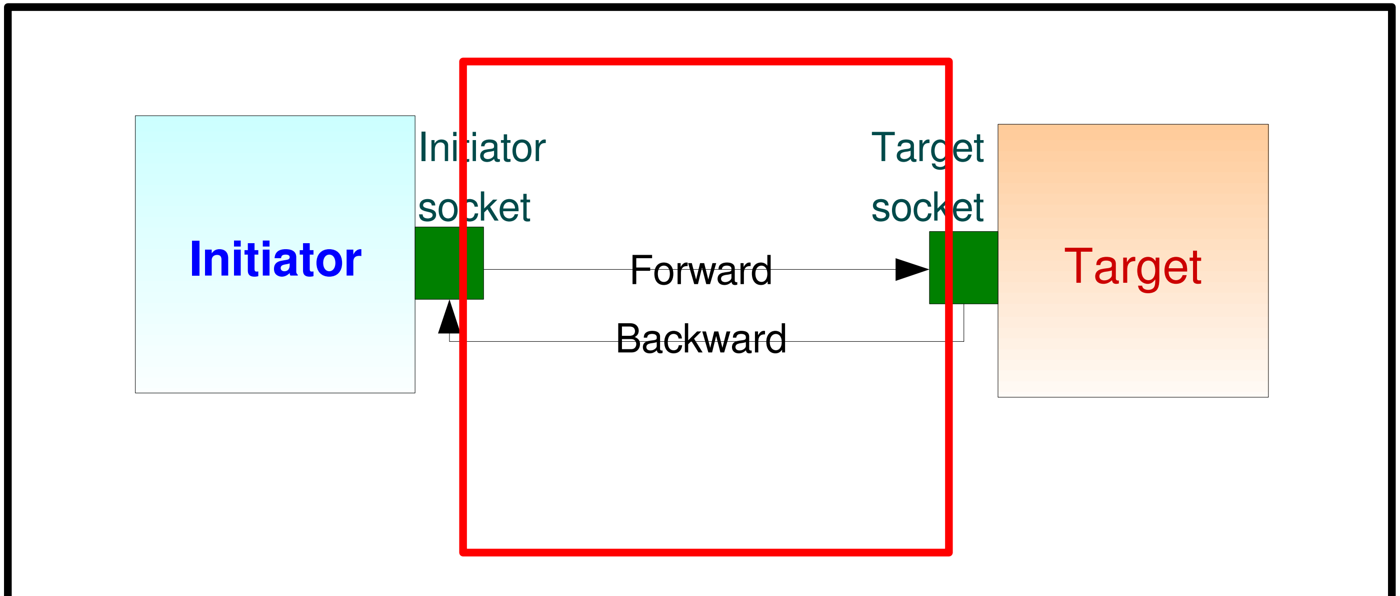
TMU (target) <-> target socket

This module is target -> Destination of Forward path

```
class Ctmu: public sc_module, public tlm::tlm_fw_transport_if<>{
    //Declare Target socket
    tlm::tlm_target_socket<> tgt_socket;
    SC_CTOR(Chpbc){
        tgt_socket(*this);    //Connect initiator socket to module
    }
}
```

4. Connection

Revised in v1.2



Step 2: connect socket with socket

4.2 Socket and another socket (1/2)

Single socket:

Connect initiator socket to target socket

Connect initiator socket to another initiator socket

Connect target socket to another target socket

Multi socket:

Connect multi initiator socket to many target socket

Connect multi initiator socket to many initiator socket

Connect multi target socket to many target socket

4.2 Socket and another socket (2/2)

Revised in v1.2

Example:

Initiator socket of Master HPB <-> Target socket of TMU

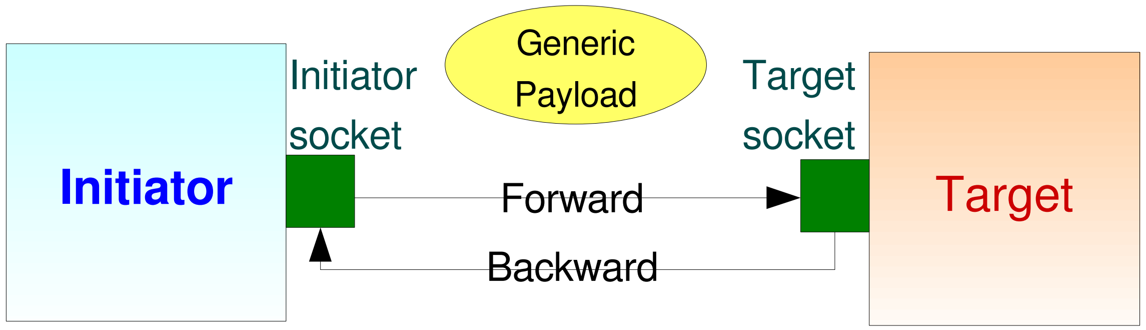
```
class Chpbc: public sc_module, public tlm::tlm_bw_transport_if<>{
        tlm::tlm_initiator_socket<> ini_socket;
        Ctmu *tmu;
        SC_CTOR(Chpbc){
                ini_socket(*this);       //Connect initiator socket to module
                tmu = new Ctmu(“tmu”);
                ini_socket(tmu->tgt_socket);  //Connect initiator socket to target socket
        }
}
```

Outline

1. Transaction Level Modeling (TLM)
2. Basic Objects in TLM2.0
3. TLM2.0 library
4. Connection
5. Communication
    5.1. Generic Payload
    5.2. Preparation
    5.3. Block communication
    5.4. Non-Blocking communication (Not Yet)
6. TLM2.0 with Forest
7. Topics

5. Communication

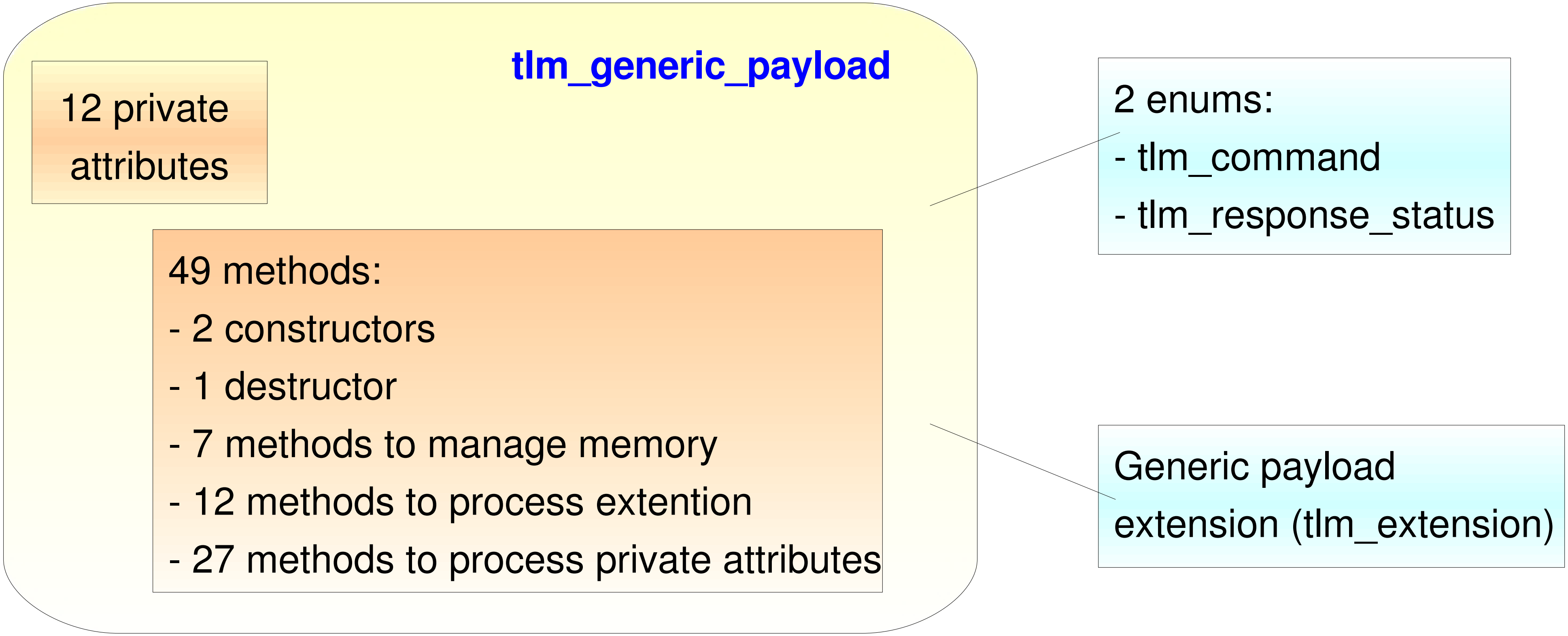
Revised in v1.2



Step 2: communication through Generic payload

5.1 Generic payload (1/6)

Generic payload (tlm_generic_payload class) is a very important object in TLM2.0. It includes all information of transaction transferred among modules.



tlm_generic_payload

12 private attributes

49 methods:
- 2 constructors
- 1 destructor
- 7 methods to manage memory
- 12 methods to process extention
- 27 methods to process private attributes

2 enums:
- tlm_command
- tlm_response_status

Generic payload extension (tlm_extension)

5.1 Generic payload (2/6)

Some main Attributes

No.	Data type	Name	Explanation
1	sc_dt::unit64	m_address	- Address value. The **start address** on the system memory map of the contiguous block of data begin read or written - Need not be word-aligned

2	tlm_command	m_command	Command value:
			enum **tlm_command** {
			TLM_READ_COMMAND, TLM_WRITE_COMMAND, TLM_IGNORE_COMMAND};

3	unsigned char*	m_data	Data pointer
4	unsigned int	m_length	- The number of bytes to be copied to or from the data array, inclusive of any bytes disabled by the byte enable attribute

5	tlm_response_status	m_response_status	- Indicate whether an error has occurred during the transaction
			enum **tlm_response_status** {
			TLM_OK_RESPONSE = 1,
			TLM_INCOMPLETE_RESPONSE = 0,
			TLM_GENERIC_ERROR_RESPONSE = -1,
			TLM_ADDRESS_ERROR_RESPONSE = -2,
			TLM_COMMAND_ERROR_RESPONSE = -3,
			TLM_BURST_ERROR_RESPONSE = -4,
			TLM_BYTE_ENABLE_ERROR_RESPONSE = -5
			};
			(-4): the target is unable to execute the transaction with the given streaming width in case that initiator supports it
			(-1): Any other error

5.1 Generic payload (3/6)

Some main Attributes

No.	Data type	Name	Explanation
7	unsigned char*	m_byte_enable	Byte enable pointer. Macro TLM_BYTE_DISABLED (0): this byte is disabled Macro TLM_BYTE_ENABLED (0xFF): this byte is enabled
8	unsigned int	m_byte_enable_length	Byte enable length
9	tlm_array<tlm_extension_base*>	m_extensions	Extension array

5.1 Generic payload (4/6)

Some main methods

No.	Prototype	Explanation
1	bool **is_read**()	Return true if command attribute is TLM_READ_COMMAND
2	void **set_read**()	Set the command attribute to TLM_READ_COMMAND
3	bool **is_write**()	Return true if command attribute is TLM_WRITE_COMMAND
4	void **set_write**()	Set the command attribute to TLM_WRITE_COMMAND
5	tlm_command **get_command**()	Get the current command value
6	void **set_command** (const tlm_command command)	Set the command value to command attribute
7	sc_dt::unit64 **get_address**()	Return the current value of the address attribute
8	void **set_address**(const sc_dt::unit64 address)	Set the address attribute to the value passed as an argument The value of the address attribute need not be word-aligned
9	unsigned char* **get_data_ptr**()	Return the current value of the data pointer attribute
10	void **set_data_ptr** (unsigned char* data)	Set the data pointer attribute to the value passed as an argument. For the read command, the contents of the data array will be overwritten by the target

5.1 Generic payload (5/6)

Some main methods

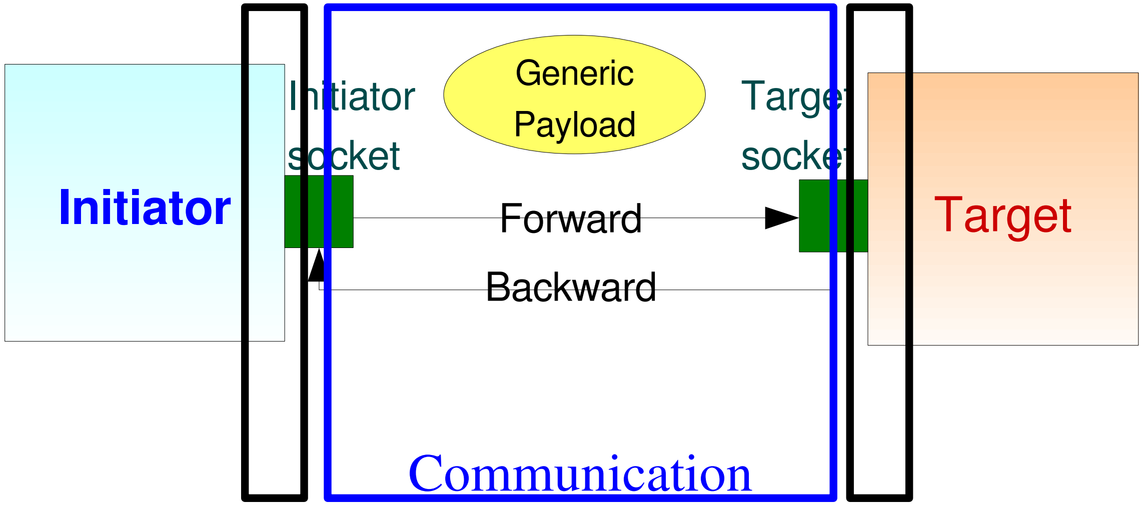
No.	Prototype	Explanation
11	unsigned int **get_data_length**()	Return the current value of the data length attribute. The data length attribute as ***the number of bytes*** to be copied to or from the data array
12	void **set_data_length** (const unsigned int length)	Set the data length attribute to the value passed as an argument
13	bool **is_response_ok**()	Return **true** if an only if the current value of the response status attribute is TLM_OK_RESPONSE
14	bool **is_response_error**()	Return **true** if an only if the current value of the response status attribute is NOT TLM_OK_RESPONSE
15	tlm_response_status **get_response_status**()	Return the current value of the response status attribute
16	void **set_response_status** (const tlm_response_status)	Set the response status attribute
17	std::string **get_response_string**()	Return the current value of the response status attribute as a text string
18	unsigned char* **get_byte_enable_ptr**()	Return the current value of the byte enable pointer attribute
19	void **set_byte_enable_ptr** (unsigned char* byte_enable)	Set the pointer to the byte enable array to the value passed as an argument

5.1 Generic payload (6/6)

Some main methods

No.	Prototype	Explanation
20	unsigned int **get_byte_enable_length**()	Return the current value of the byte enable length attribute
21	void **set_byte_enable_length** (const unsigned int byte_enable_length)	Set the byte enable length attribute

5.2 Preparation (1/6)



Initiator and target communicate together by **calling functions**. All function must be defined in Initiator module and target module before communication.

5.2 Preparation (2/6)

Initiator module: inherit from ***tlm::tlm_bw_transport_if<>***

Moreover, ***tlm::tlm_bw_transport_if<>*** inherit publicly and virtually from:

tlm::tlm_bw_non_blocking_transport_if<>

tlm::tlm_sync_enum **nb_transport_bw** (tlm::tlm_generic_payload & trans, tlm::tlm_phase& phase, sc_time& delay)

tlm::tlm_bw_direct_mem_if

void **invalidate_direct_mem_ptr**(sc_dt::uint64 start_range, sc_dt::uint64 end_range)

=> Initiator module MUST define both of two methods:

nb_transport_bw

invalidate_direct_mem_ptr

5.2 Preparation (3/6)

```
class Chpbc: public sc_module, public tlm::tlm_bw_transport_if<>{
        //Declare Initiator socket
        tlm::tlm_initiator_socket<> hpb_master;
        SC_CTOR(Chpbc){
                ini_socket(*this);      //Connect initiator socket to module
        }
        tlm::tlm_sync_enum nb_transport_bw (tlm::tlm_generic_payload & trans,  tlm::tlm_phase&
                                                    phase, sc_time& delay)
        {
               //Do something
        }
         void invalidate_direct_mem_ptr(sc_dt::uint64 start_range, sc_dt::uint64 end_range)
         {
               //Do something
         }
}
```

5.2 Preparation (4/6)

Target module: **inherit** from ***tlm::tlm_fw_transport_if<>***.

Moreover, ***tlm::tlm_fw_transport_if<>*** inherit publicly and virtually from:

tlm::tlm_fw_non_blocking_transport_if<>

tlm::tlm_sync_enum **nb_transport_fw** (tlm::tlm_generic_payload & trans, tlm::tlm_phase& phase, sc_time& delay)

tlm::tlm_fw_direct_mem_if

bool **get_direct_mem_ptr**(tlm::tlm_generic_payload & trans, tlm::tlm_dmi& dmi_data)

tlm::tlm_blocking_transport_if<>

void **b_transport**(tlm::tlm_generic_payload& trans, sc_core::sc_time& t)

tlm::tlm_transport_dbg_if<>

unsigned int **transport_dbg**(tlm::tlm_generic_payload& trans)

=> Target module MUST define four above functions

5.2 Preparation (5/6)

```
class Ctmu: public sc_module, public tlm::tlm_fw_transport_if<>{
        //Declare Target socket
        tlm::tlm_target_socket<> tgt_socket;
        SC_CTOR(Chpbc){
                tgt_socket(*this);      //Connect initiator socket to module
        }
        tlm::tlm_sync_enum nb_transport_fw (tlm::tlm_generic_payload & trans,  tlm::tlm_phase&
                                                        phase, sc_time& delay){}
        bool get_direct_mem_ptr(tlm::tlm_generic_payload & trans, tlm::tlm_dmi&  dmi_data){}
        void b_transport(tlm::tlm_generic_payload& trans, sc_core::sc_time& t){}
        unsigned int transport_dbg(tlm::tlm_generic_payload& trans)
}
```

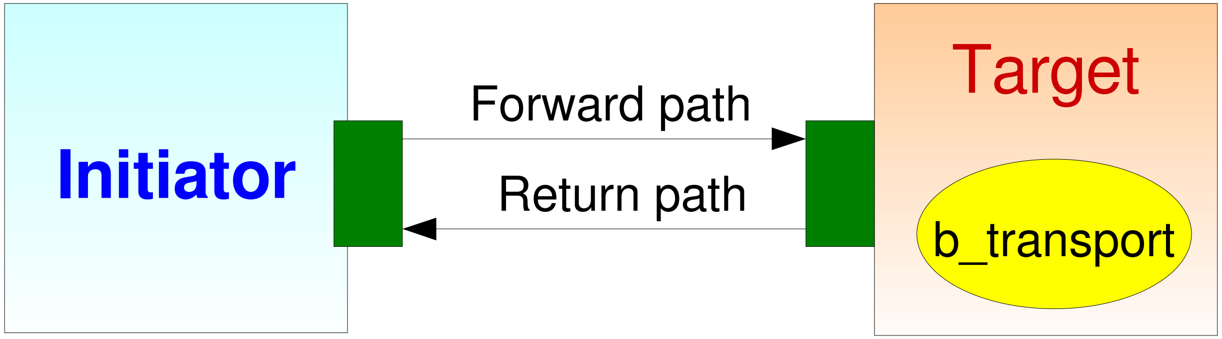
5.2 Preparation (6/6)

Summary

Function	Where to be defined	Explanation
nb_transport_bw	Initiator	- Called by Target Module - Main function of Non-Blocking transaction
nb_transport_fw	Target	- Called by Initiator Module - Main function of Non-Blocking transaction
b_transport	Target	- Called by Initiator - Main function of Blocking transaction
invalidate_direct_mem_ptr	Target	- Called by Initiator - Used in Direct memory interface
get_direct_mem_ptr	Initiator	- Called by Target - Used in Direct memory interface
transport_dbg	Target	- Called by Target - Used in Debug interface

TLM2.0 provide us these pure virtual functions, attributes, methods. ***The contents of each function is depended on us***

5.3 Block communication (1/3)



Blocking interface: Initiator module -> Target module
(Initiator socket -> Target socket)

Step 1: Create transaction object (tlm::tlm_generic_payload object)

Step 2: Set values or information for transaction object by using methods of tlm::tlm_generic_payload

Step 3: From Initiator socket call b_transport function

5.3 Block communication (2/3)

Revised in v1.2

Example:

```
void HPB_Thread(){
//Create transaction object
    tlm::tlm_generic_payload hpb_transport;

//Set information for Generic payload
    hpb_transport.set_address(addr);
    hpb_transport.set_command(tlm::TLM_READ_COMMAND);
    hpb_transport.set_data_length(dsize);
    hpb_transport.set_response_status(tlm::TLM_INCOMPLETE_RESPONSE);

// Call b_tranport
    sc_time delay_time = SC_ZERO_TIME;
    ini_socket->b_transport(hpb_transport, delay_time);
}
```

5.3 Block communication (3/3)

void b_transport(tlm::tlm_generic_payload & trans, sc_core::sc_time& t)

- Defined in target module

- We can use event, wait of SystemC in b_transport

- The call to b_transport will mark the first timing point. The return from b_transport will mark the final timing point

- ***sc_time t***: delay time. How it affects our modules is depended on how we implement it.

- Get information/values of generic payload by methods of tlm_generic_payload

Outline

1. Transaction Level Modeling (TLM)
2. Basic Objects in TLM2.0
3. TLM2.0 library
4. Connection
5. Communication
6. **TLM2.0 with Forest**
7. Topics

6. TLM2.0 with Forest

Modify Makefile:

Add path of TLM2.0 library:

TLM = /shsv/sld/Common/Lib/04_TLM/TLM2.0-2008-06-09

(*This path is according to where you store TLM2.0 library*)

Add define option:

DEFS = -DSHX2 ..... -**DSC_INCLUDE_DYNAMIC_PROCESSES** $(FOREST_TYPE) $(FOREST_SNC_TYPE) $(FOREST_SHWY_WIDTH)

Modify Makefile.defs:

INCDIRS = $(INCDIR) -I. -I$(SYSTEMC)/include **-I$(TLM)/include/tlm** $(addprefix -I, $(SEARCH_DIR))

6. TLM2.0 with Forest

TLM2.0 has two namespace:

Namespace **tlm**: contains main classes

=> #include <tlm.h>

Namespace **tlm_utils**: contains utility classes

=> Include file according to which class to use

For example, #include "tlm_utils/simple_initiator_socket.h"

(Use tlm_simple_initiator_socket class)

Always declare namespace

Example:

tlm::tlm_initiator_socket<>

tlm::tlm_fw_transport_if<>

Outline

1. Transaction Level Modeling (TLM)
2. Basic Objects in TLM2.0
3. TLM2.0 library
4. Connection
5. Communication
6. TLM2.0 with Forest
7. Topics

Topics

1. Research transaction object (tlm::tlm_generic_payload class), especially extension class (tlm::tlm_extension)

2. Research non-blocking transport interface, applied for complicated module/bus/bridge

3. Research Direct Memory interface: a method to increase simulation speed of memory access between Initiator and target

4. Connect multi-sockets – Use initiator socket or target socket complicatedly

5. Timing in TLM2.0: apply temporal decoupling, quantum and quantum keeper in loosely-timed coding style

RENESAS

Thank you for your attention

Renesas Design Viet Nam Co., Ltd.