LLWEB-00010925
ZSG-F31-12-0029-01

SC-HEAP_E3 : Bus I/F outline

Renesas Electronics Corporation
MCU Software Division

2012/08/27

RENESAS Group CONFIDENTIAL

Purpose of the document

The document describes the outline of bus I/F applied to SC-HEAP_E3.
The document is for the developer of the bus master IP or bus slave IP.
The bus I/F separates the functional part in C++ and communication part in SystemC.
Due to this separation, the functional part in C++ can be easily reused in the other simulator using the different communication.
(e.g. original C++ code in CoMET, WindRiver Simics, etc..)

Table of contests

1. Bus communication image

Bus communication image

- Use TlmInitiatorSocket derived from tlm_initiator_socket and TlmTargetSocket derived from tlm_target_socket.
- Bus master IP is created by being derived from BusMasterBase. Bus slave IP is created by being derived from BusSlaveBase. These base classes instantiate sockets and bus interface parts automatically.
- Functional part should be written in C++. (Due to this separation from SystemC description, the code for the functional can be easily reused.)

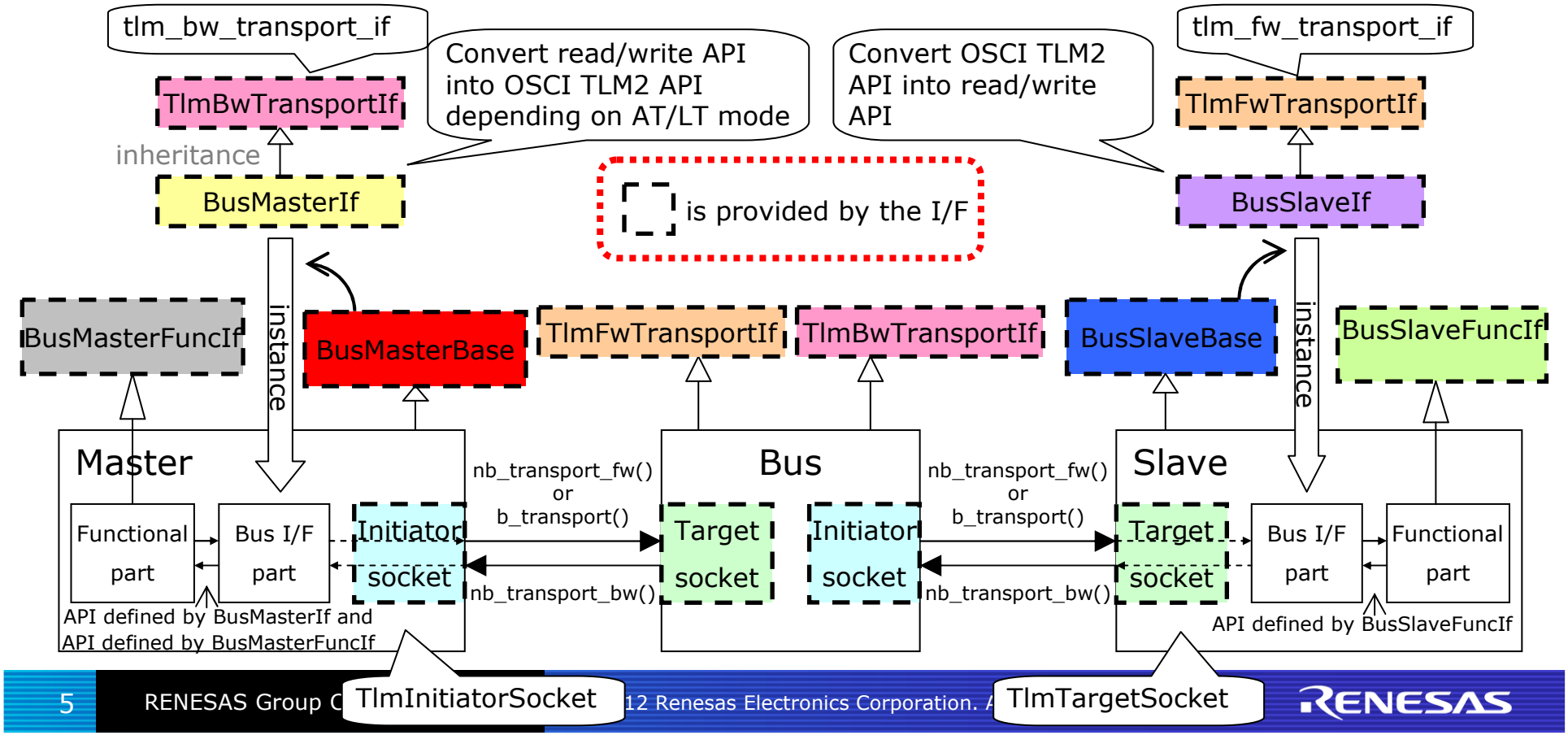


Image (AT:master IP case 1)

Only purple boxes are codes which must be created by user.

Ex. Read at accuracy mode(2phase AT) (blocking in read)

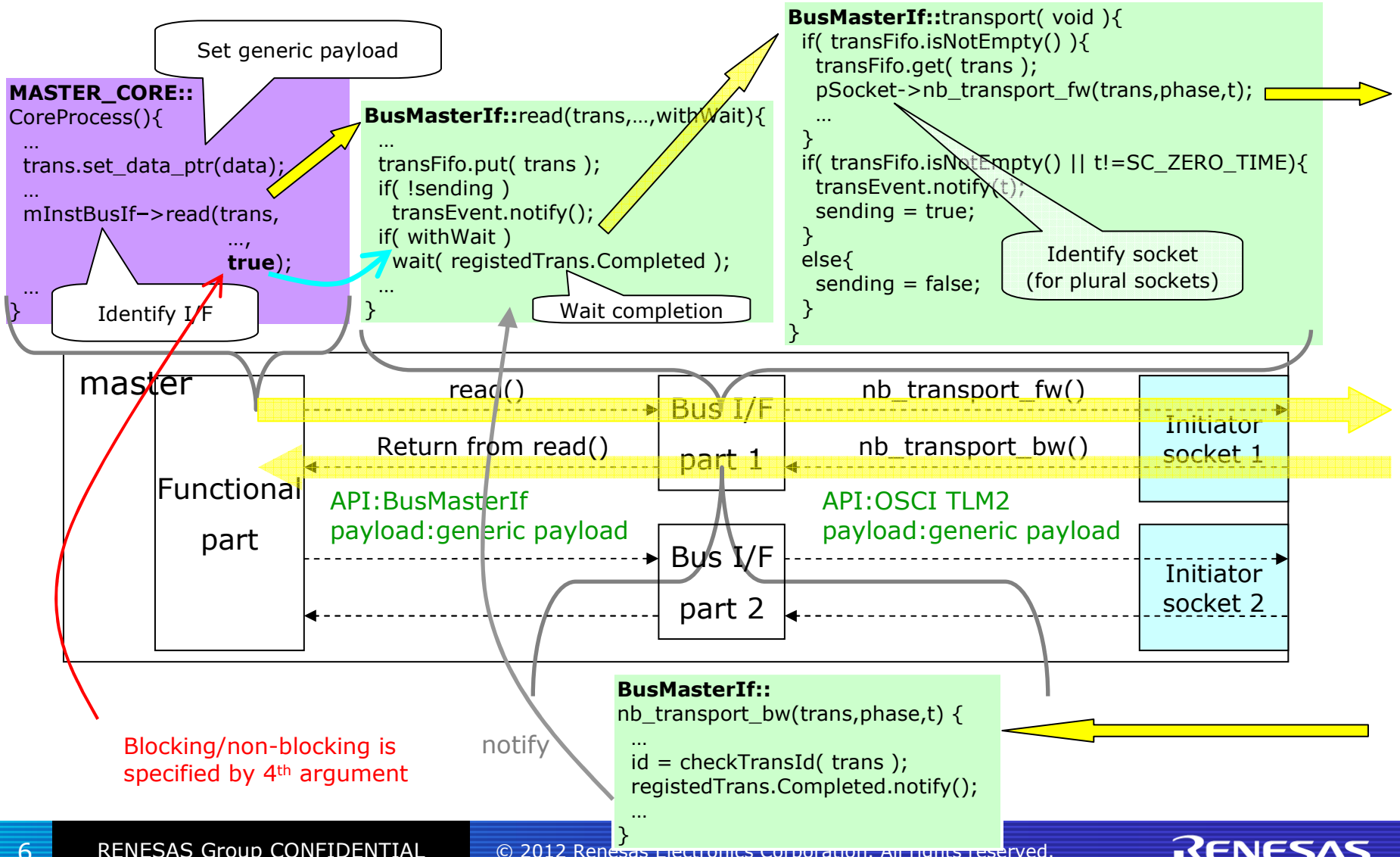


Image (AT:master IP 2)

Only purple boxes are codes which must be created by user.

Ex. Read at accuracy mode(2 phase AT) (non-blocking in read)

```
MASTER_CORE::
CoreProcess(){
  ...
  trans.set_data_ptr(data);
  ...
  mInstBusIf->read(trans,
                   ...,
                   false );
  ...
}
```

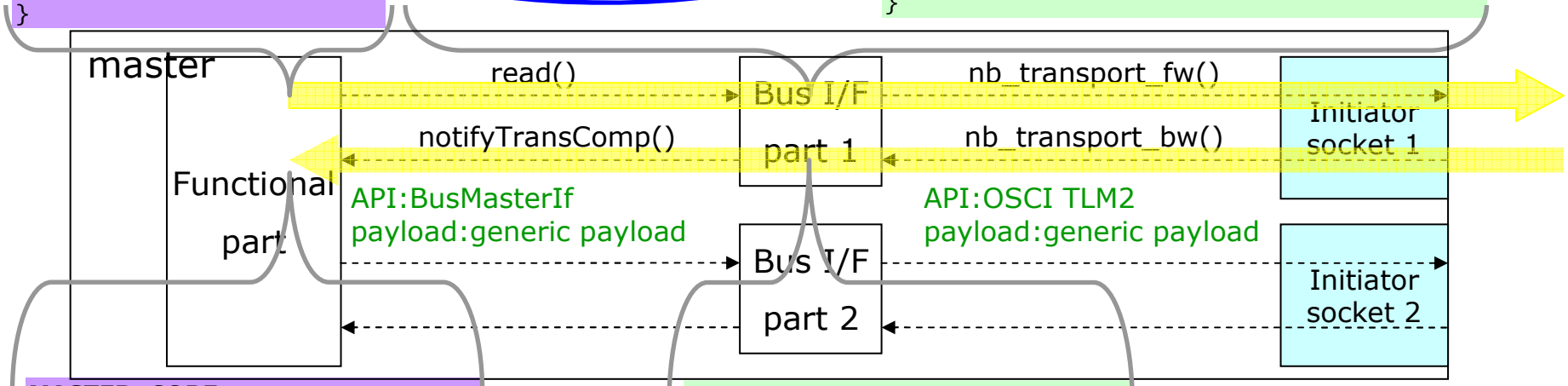
```
BusMasterIf::read(trans,...,withWait){
  ...
  transFifo.put( trans );
  if( !sending )
    transEvent.notify();
  if( withWait )
    wait( registedTrans.Completed );
  ...
}
```

Not wait

Notify to send

```
BusMasterIf::transport( void ){
  if( transFifo.isNotEmpty() ){
    transFifo.get( trans );
    pSocket->nb_transport_fw(trans,phase,t);
    mCore->notifyTransSend(trans,t);
  }
  if( transFifo.isNotEmpty() || t!=SC_ZERO_TIME){
    transEvent.notify(t);
    sending = true;
  }
  else{
    sending = false;
  }
}
```

```
MASTER_CORE::
notifyTransSend(trans, t){
  ...
}
```




```
MASTER_CORE::
notifyTransComp(trans, t){
  ...
}
```

Notify completion by calling notifyTransComp(). BusMasterIf calls notifyTransComp() immediately after nb_transport_bw is called. Functional part uses the read value after t(2nd argument).※

```
BusMasterIf::
nb_transport_bw(trans,phase,t) {
  mCore->notifyTransComp(trans,
t);
  ...
}
```

※timing of using the read data
At single data transfer, the read data can be used after t. At burst transfer, each data can be used each t/data_numter.

Only purple boxes are codes which must be created by user.

Image (AT:slave IP)

Ex. Read at accuracy mode(2 phase AT)

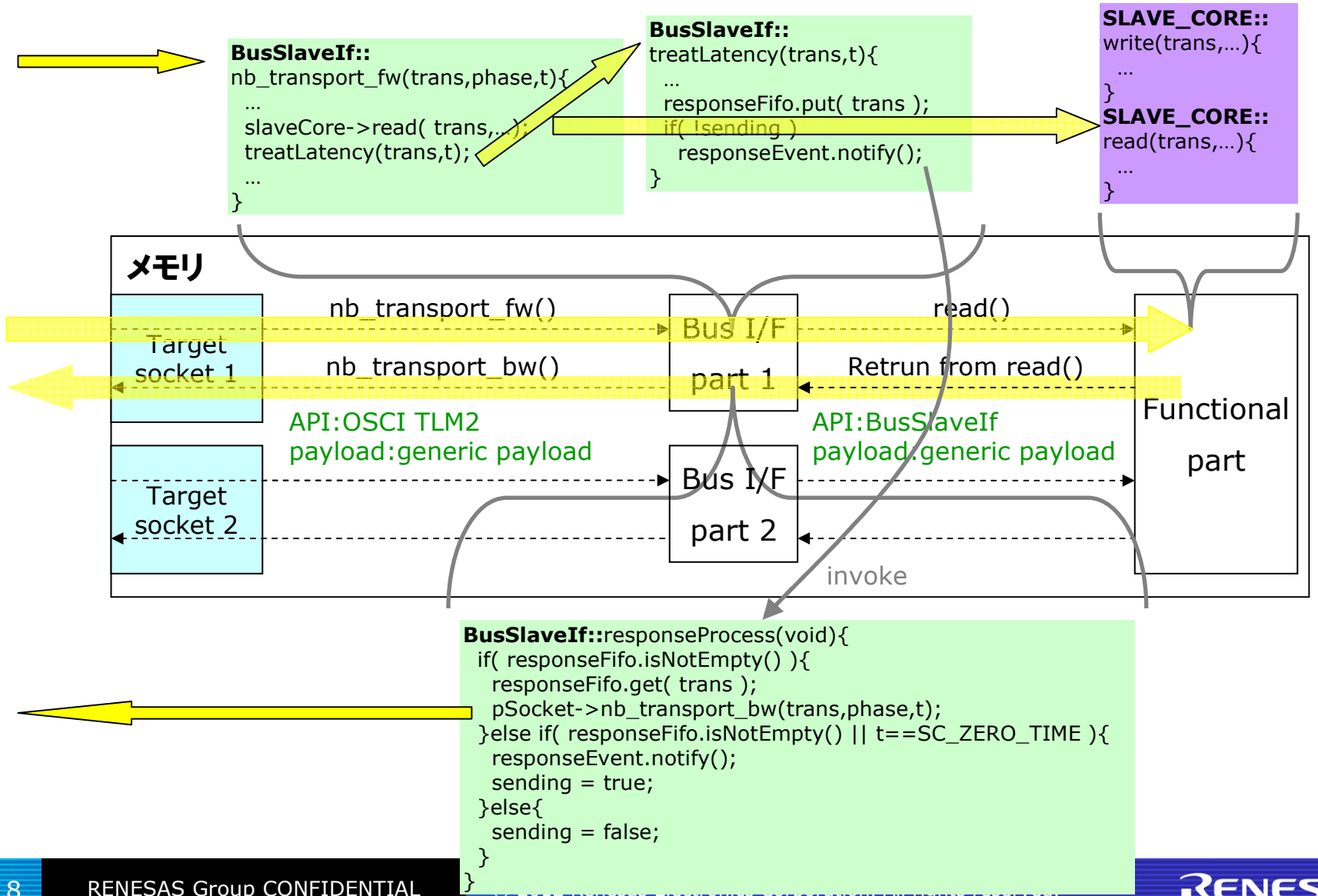


Image (LT:master IP)

Only purple boxes are codes which must be created by user.

Ex. Read at Fast mode(LT)

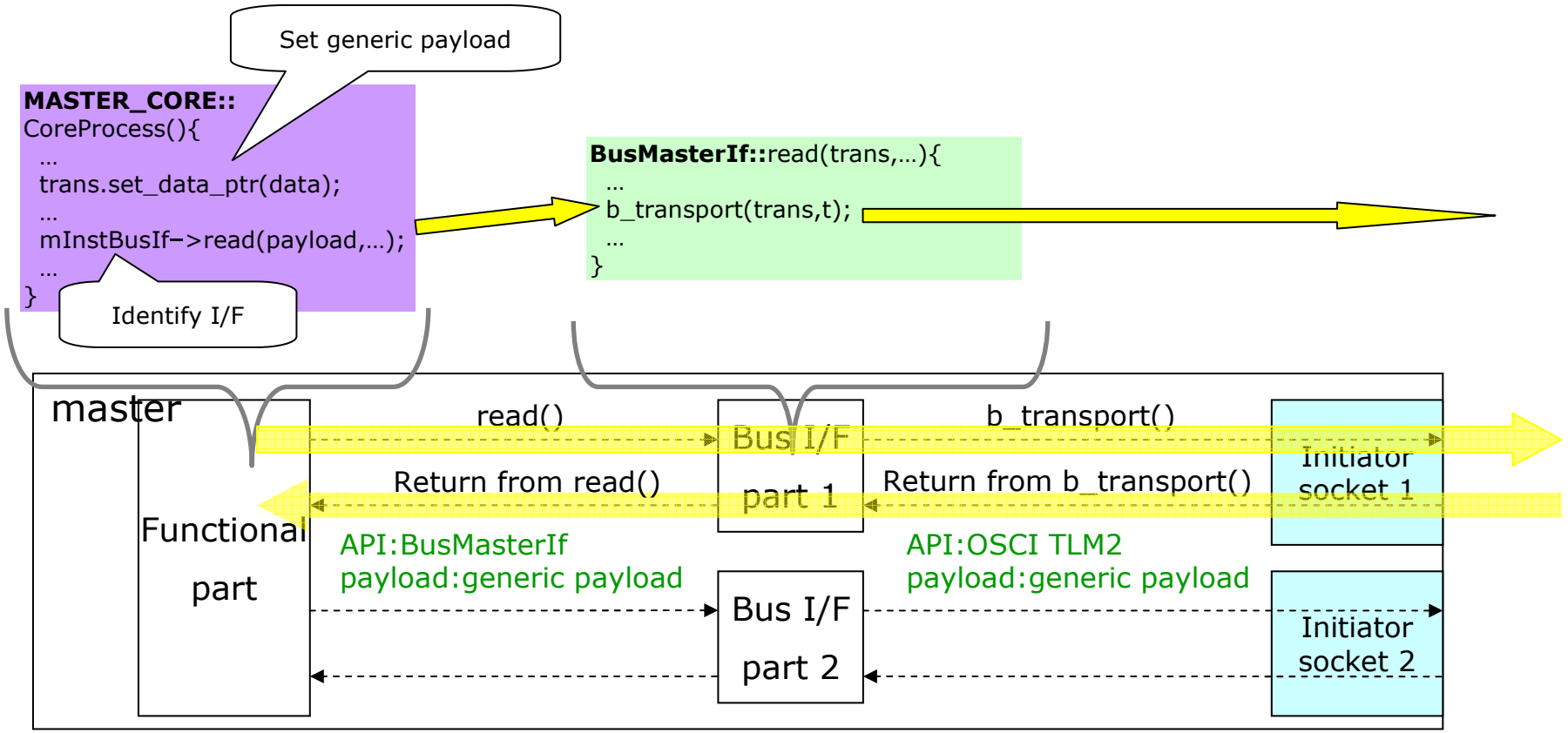


Image (LT:slave IP)

Only purple boxes are codes which must be created by user.

Ex. Read fast mode(LT)

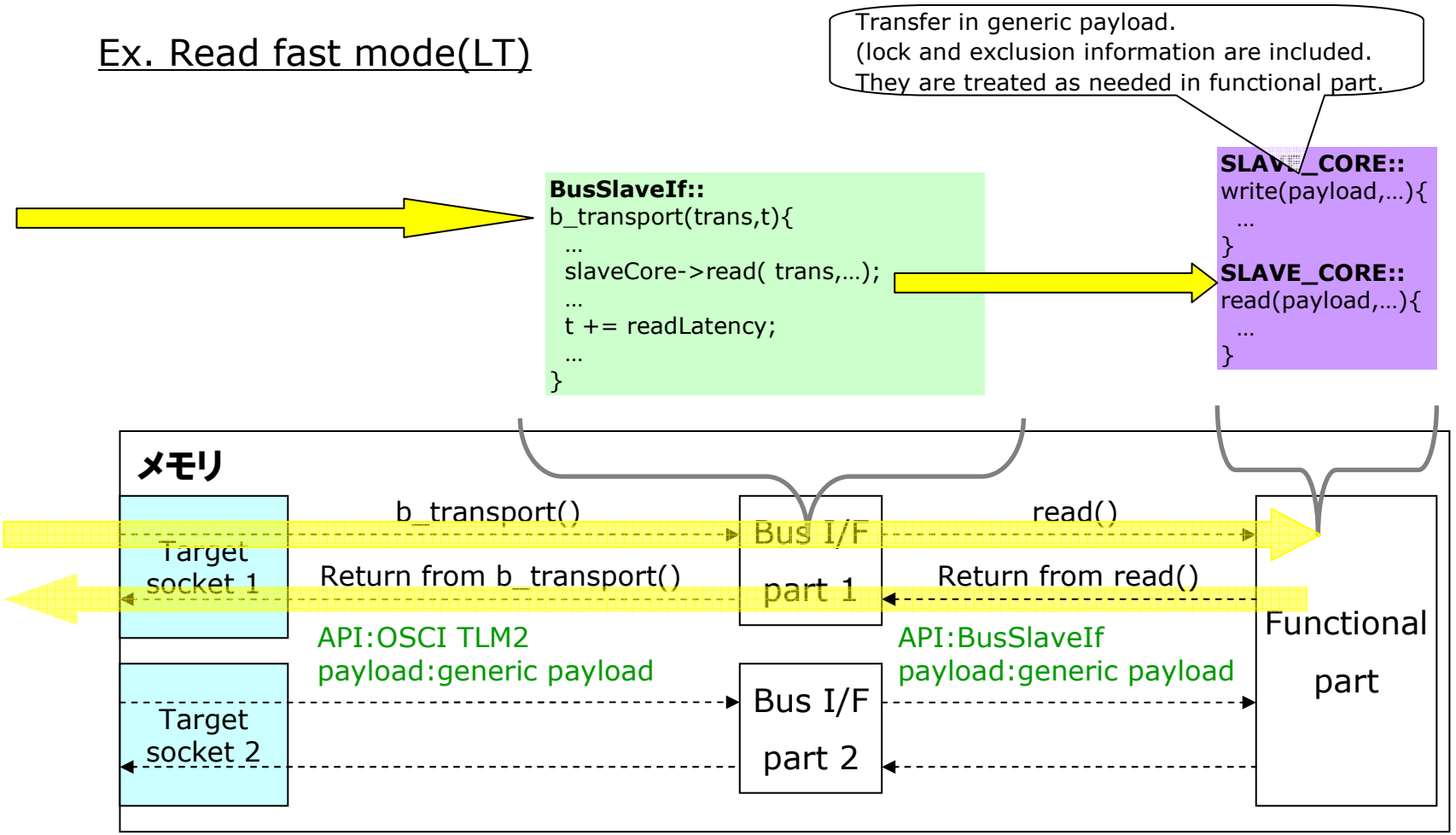


Image (debug access:master IP)

Only purple boxes are codes which must be created by user.

Ex. Read at debug access

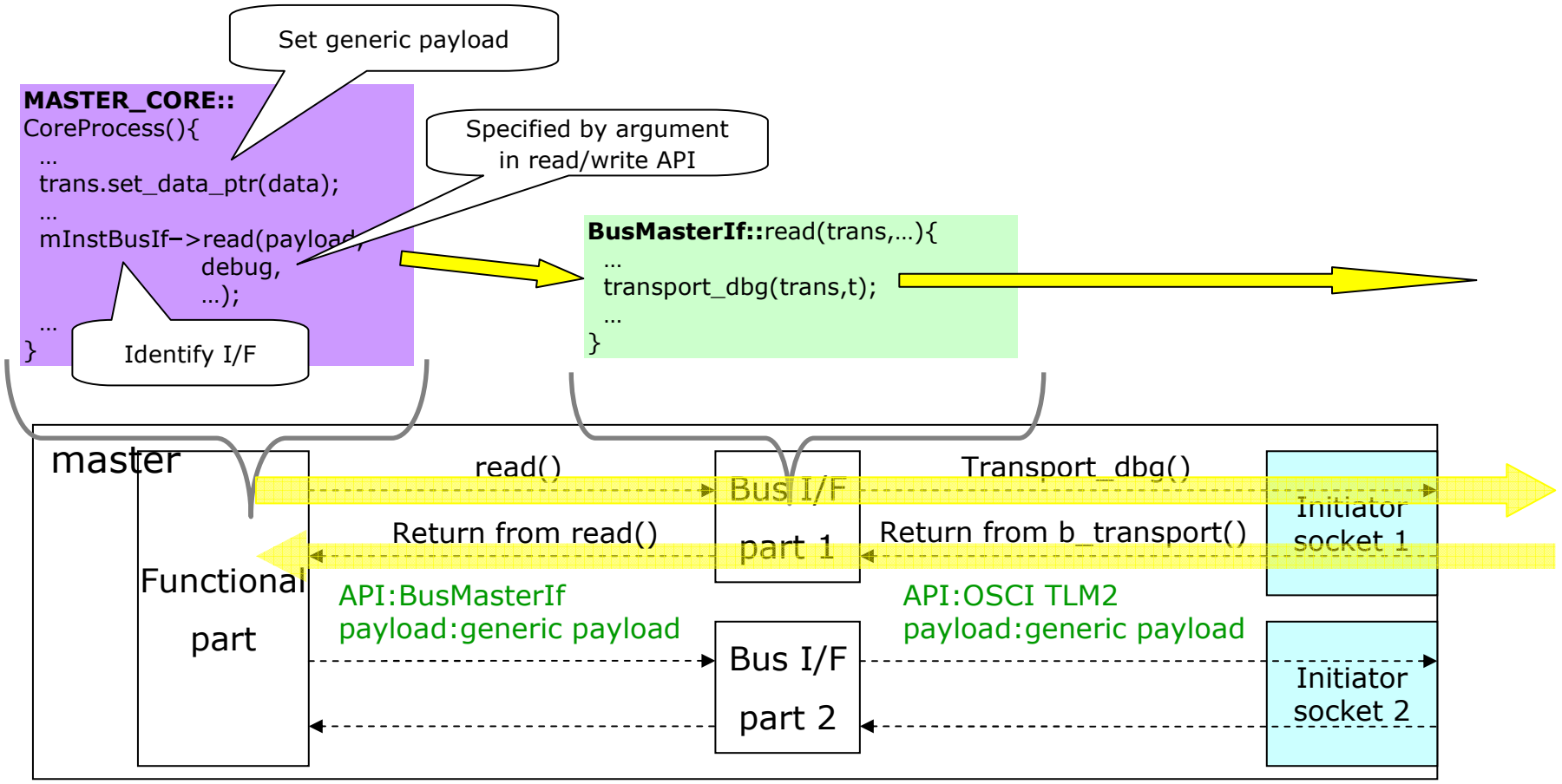


Image (debug access:slave IP)

Only purple boxes are codes which must be created by user.

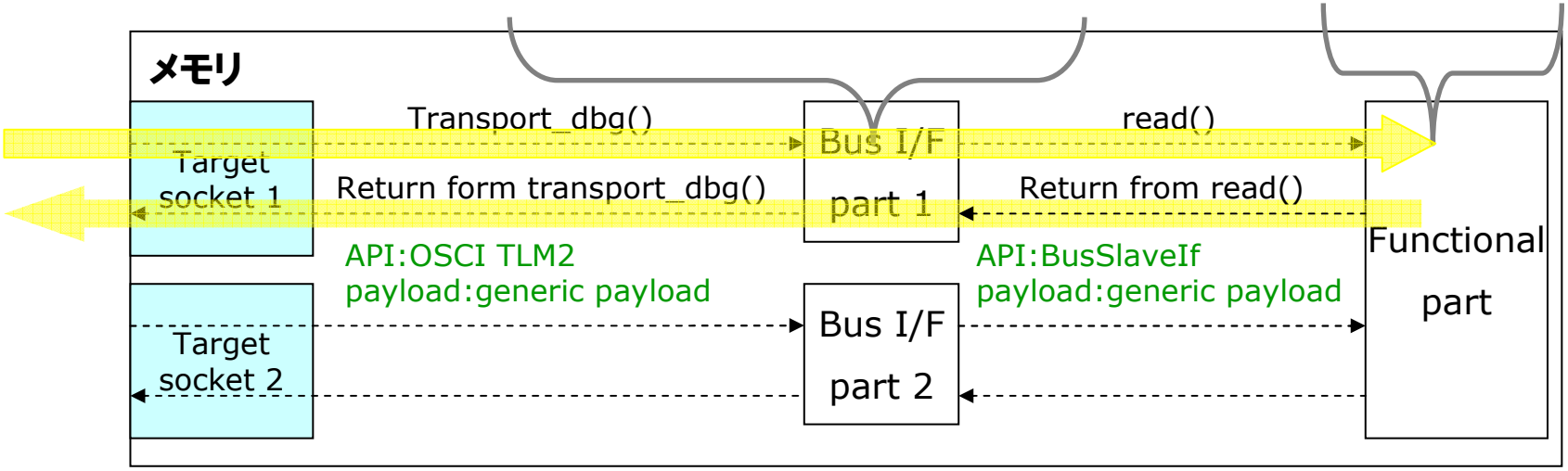
Ex. Read at debug access

Transfer in generic payload.
(lock and exclusion information are included.
They are treated as needed in functional part.

debug access

```
BusSlaveIf::
transport_dbg(trans,t){
  …
  slaveCore->read( trans,…,debug);
  …
}
```

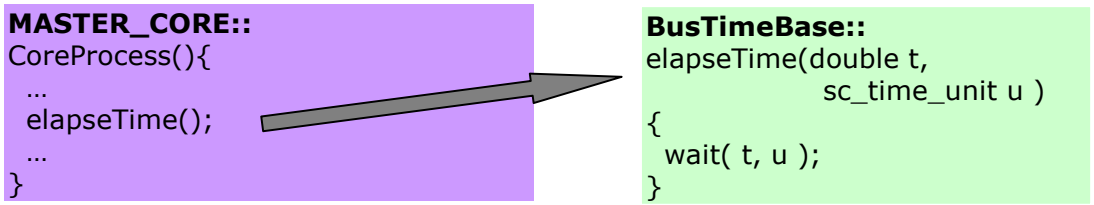
```
SLAVE_CORE::
write(payload,…,debug){
  …
}
SLAVE_CORE::
read(payload,…,debug){
  …
}
```



Time spending

Only purple boxes are codes which must be created by user.

The IP which is needed to spend time (synchronization), should be used API of BusTimeBase class.



2. class

Class (1)

[For bus master IP]

class	outline
BusMasterBase	Base class of bus master IP. Bus width and socket number is specified with template arguments.
BusMasterIf	The I/F class is derived from tlm_bw_transport_if. The class is instanced and bound to socket automatically, by calling member function of BusMasterBase. The transaction is transferred from bound socket when read/write API of BusMasterIf called from functional part of master IP. (read/write is converted to API of OSCI TLM2.0)
BusMasterFuncIf	Functional part of bus master IP should be derived from the class. API of transaction completion is defined as pure virtual function.
TlmInitiatorSocket	The socket class is derived from tlm_initiator_socket. The class is instanced and bound to I/F automatically, by calling member function of BusMasterBase.

[For bus slave IP]

class	outline
BusSlaveBase	Base class of bus slave IP. Bus width and socket number is specified with template arguments.
BusSlaveIf	The I/F class is derived from tlm_fw_transport_if. The class is instanced and bound to socket automatically, by calling member function of BusSlaveBase. The class call read/write API of BusSlaveFuncIf when transaction is received from bound socket.
BusSlaveFuncIf	Functional part of bus slave IP should be derived from the class. Read/write API is defined as pure virtual function. And memory size setting API which is called from end_of elaboration() is also defined.(setfunc())
TlmTargetSocket	The socket class is derived from tlm_initiator_socket. The class is instanced and bound to I/F automatically, by calling member function of BusMasterBase.

Class (2)

[For time spending]

class	outline
BusTimeBase	The IP which is needed to spend time (synchronize) in functional part, should be derived from the base class. The class is prepared to isolate Functional part from SystemC.

Class for bus master IP

BusMasterBase.h

```
template<unsigned int BUSWIDTH, unsigned int S_NUM>
class BusMasterBase
{
  BusMasterBase( void );                                  constructor
  ~BusMasterBase( void );                                 destructor

  void setMasterResetPort( sc_in<bool> *resetPort, … );   Set reset port
  void setMasterFreqPort( sc_in<uint64> *freqPort, … );   Set frequency port
  void setInitiatorSocket( unsigned char *socketName, … ); Instantiate sockets

  TlmInitiatorSocket<BUSWIDTH> *iSocket[S_NUM];           Socket pointer array
  BusMasterIf<BUSWIDTH> *mBusMasterIf[S_NUM];             Bus I/F pointer array
}

template<unsigned int S_NUM>
class BusMasterBase<32, S_NUM>
{
  BusMasterBase( void );
  ~BusMasterBase( void );

  void setMasterResetPort32( sc_in<bool> *resetPort, … );
  void setMasterFreqPort32( sc_in<uint64> *freqPort, … );
  void setInitiatorSocket32( unsigned char *socketName, … );

  TlmInitiatorSocket<32> *iSocket32[S_NUM];
  BusMasterIf<32> *mBusMasterIf32[S_NUM];
}                                                         Specialized for 32bits*

template<unsigned int S_NUM>
class BusMasterBase<64, S_NUM>
{
  BusMasterBase( void );
  ~BusMasterBase( void );

  void setMasterResetPort64( sc_in<bool> *resetPort, … );
  void setMasterFreqPort64( sc_in<uint64> *freqPort, … );
  void setInitiatorSocket64( unsigned char *socketName, … );

  TlmInitiatorSocket<64> *iSocket64[S_NUM];
  BusMasterIf<64> *mBusMasterIf64[S_NUM];
}                                                         Specialized for 64bits*
```

* About specialized template.
  This is solution for creation of several socket which is various bus width.

Ex. Case of creation of master IP which has 32bits-socket x 2 and 64bits-socket x 3:

```
class MASTER()
  :BusMasterBase<32,2>,
   BusMasterBase<64,3>
{
  …
}
```

Class for bus master IP

Continuation of BusMasterBase.h

```
template<unsigned int S_NUM>
class BusMasterBase<128, S_NUM>
{
  BusMasterBase( void );
  ~BusMasterBase( void );

  void setMasterResetPort128( sc_in<bool> *resetPort, … );
  void setMasterFreqPort128( sc_in<uint64> *freqPort, … );
  void setInitiatorSocket128( unsigned char *socketName, … );

  TlmInitiatorSocket<128> *iSocket128[S_NUM];
  BusMasterIf<128> *mBusMasterIf128[S_NUM];
}
```

Specialized for 128bits*

Class for bus master IP

BusMasterIf.h

```
template<unsigned int BUSWIDTH>
class BusMasterIf :
  public sc_module,
  public virtual TlmBwTransportIf
{
  BusMasterIf( sc_module_name name,
               sc_in<bool>, &resetPort,
               sc_in<uint64> &freqPort,
               TlmInitiatorSocket<BUSWIDTH> &iSocket );
  ~BusMasterIf( void );

  bool read( TlmTransactionType &trans,
             bool debug = false,
             BusTime_t t = 0,
             bool withWait = false );
  bool write( TlmTransactionType &trans,
              bool debug = false,
              BusTime_t t = 0,
              bool withWait = false );

  TlmSyncEnum nb_transport_bw( TlmTransactionType &trans,
                               TlmPhase &phase,
                               BusTime_t &t );
  void invalidate_direct_mem_ptr( uint64 startRange, uint64 endRange );
  void setFuncModulePtr( BusMasterFuncIf *pFuncModule );
  void setTransNmbLmt( unsigned int nmb );
  void setBusProtocol( BusProtocol_t protocol );
}
```

constructor

destructor

read

write

typedef of sc_time

DMI is not supported

OSCI TLM2.0 API

Set functional part pointer

Set transaction limit number

Set bus protocol

* About transaction completion notice API
  At using argument withWait of the read/write function in True, read/write function returns after transaction completion. At using withWait in False, read/write function returns immediately after a transaction transmission, and Bus I/F part notifies Functional part of transaction completion by calling completion notice API notifyTransComp defined in Functional part.

Class for bus master IP

BusMasterFuncIf.h

```
class BusMasterFuncIf
{
  BusMasterFuncIf( void );
  ~BusMasterFuncIf( void );

  virtual void notifyTransSend( TlmTransactionType &trans, BusTime_t &t );
  virtual void notifyTransComp( TlmTransactionType &trans, BusTime_t &t );
}
```

Constructor

Destructor

Notify transaction transmission

Notify transaction completion

When using the APIs called from Bus I/F part, the are overwritten in Functional part.

Class for bus slave IP

BusSlaveBase.h

```
template<unsigned int BUSWIDTH, unsigned int S_NUM>
class BusSlaveBase
{
  BusSlaveBase( void );                                  // Constructor
  ~BusSlaveBase( void );                                 // Destructor

  void setSlaveResetPort( sc_in<bool> *resetPort, … );   // Set reset port pointer
  void setSlaveFreqPort( sc_in<uint64> *freqPort, … );   // Set frequency port pointer
  void setTargetSocket( unsigned char *socketName, … );  // Instantiate sockets

  TlmInitiatorSocket<BUSWIDTH> *tSocket[S_NUM];          // Socket pointer array
  BusSlaveIf<BUSWIDTH> *mBusSlaveIf[S_NUM];              // Bus I/F pointer array
}

template<unsigned int S_NUM>
class BusSlaveBase<32, S_NUM>                            // Specialized for 32bits*
{
  BusSlaveBase( void );
  ~BusSlaveBase( void );

  void setSlaveResetPort32( sc_in<bool> *resetPort, … );
  void setSlaveFreqPort32( sc_in<uint64> *freqPort, … );
  void setTargetSocket32( unsigned char *socketName, … );

  TlmInitiatorSocket<32> *tSocket32[S_NUM];
  BusSlaveIf<32> *mBusSlaveIf32[S_NUM];
}

template<unsigned int S_NUM>
class BusSlaveBase<64, S_NUM>                            // Specialized for 64bits*
{
  BusSlaveBase( void );
  ~BusSlaveBase( void );

  void setSlaveResetPort64( sc_in<bool> *resetPort, … );
  void setSlaveFreqPort64( sc_in<uint64> *freqPort, … );
  void setTargetSocket64( unsigned char *socketName, … );

  TlmTargetSocket<64> *tSocket64[S_NUM];
  BusSlaveIf<64> *mBusSlaveIf64[S_NUM];
}
```

* About specialized template.
  This is solution for creation of several socket which is various bus width.

Ex. Case of creation of master IP which has 32bits-socket x 2 and 64bits-socket x 3:

```
class SLAVE()
  :BusSlaveBase<32,2>,
   BusSlaveBase<64,3>
{
  …
}
```

Class for bus slave IP

BusSlaveBase.h (continuation)

```
template<unsigned int S_NUM>
class BusSlaveBase<128, S_NUM>
{
  BusSlaveBase( void );
  ~BusSlaveBase( void );

  void setSlaveResetPort128( sc_in<bool> *resetPort, … );
  void setSlaveFreqPort128( sc_in<uint64> *freqPort, … );
  void setTargetSocket128( unsigned char *socketName, … );

  TlmTargetSocket<128> *tSocket128[S_NUM];
  BusSlaveIf<128> *mBusSlaveIf128[S_NUM];
}
```

Specialized for 128bits*

Class for bus slave IP

BusSlaveIf.h

```
template<unsigned int BUSWIDTH>
class BusSlaveIf :
  public sc_module,
  public virtual TlmFwTransportIf
{
  BusSlaveIf( sc_module_name name,
              sc_in<bool>, &resetPort,
              sc_in<uint64> &freqPort,
              TlmTargetSocket<BUSWIDTH> &tSocket );
  ~BusSlaveIf( void );

  void setReadLatency( unsigned int readLatency );
  void setWriteLatency( unsigned int writeLatency);
  void setReadInitialLatency( unsigned int readLatency );
  void setWriteInitialLatency( unsigned int writeLatency);
  void setReadFirstDataLatency( unsigned int readLatency );
  void setWriteFirstDataLatency( unsigned int writeLatency);
  void setReadNextDataLatency( unsigned int readLatency );
  void setWriteNextDataLatency( unsigned int writeLatency);
  void setFuncModulePtr( BusSlaveFuncIf *funcModule );
  void setTransNmbLmt( unsigned int nmb );
  void setBusProtocol( BusProtocol_t protocol );

  TlmSyncEnum nb_transport_fw( TlmTransactionType &trans,
                               TlmPhase &phase,
                               sc_core::sc_time &t );
  void b_transport( TlmTransactionType &trans, sc_core::sc_time &t );
  int transport_dbg( TlmTransactionType &r );
  bool get_direct_mem_ptr( TlmTransactionType &trans, tlm::tlm_dmi& dmiData );

}
```

Constructor

Destructor

For APB

For AXI

Set latencies

Set functional part pointer

Set transaction limit number

Set bus protocol

OSCI TLM2.0 API

Dummy(DMI is not supported)

Class for bus slave IP

BusSlaveFuncIf.h

```
class BusSlaveFuncIf
{
  BusSlaveFuncIf( void );
  ~BusSlaveFuncIf( void );

  virtual void read( unsigned int offsetAddress,
                     TlmTransactionType &trans,
                     BUSTime_t *t,
                     bool debug = false ) = 0;

  virtual void write( unsigned int offsetAddress,
                      TlmTransactionType &trans,
                      BusTime_t *t,
                      bool debug = false ) = 0;

  virtual void setfunc( ADDRESS_TYPE size );
}
```

Constructor

Destructor

*2

*1 APIs called from the Bus I/F part

The memory size set function (It's called from bus)

*1:read and write API must be overwritten in Functional part class. setfunc API should be overwritten only at setting the memory size from bus.

*2:BusTime_t is typedef of sc_core::sc_time

Time spending class

BusTimeBase.h

```
class BusTimeBase
{
  BusTimeBase( void );
  ~BusTimeBase( void );

  void elapseTime( double t, BusTimeUnit_t u );
  void elapseTime( BusTime_t t );
  void elapseTime( double t );

  bool readResetPort( void );
  void setResetForBusTimeBase( sc_in<bool> *resetPort );

}
```

Constructor

Destructor

Synchronize simulation time

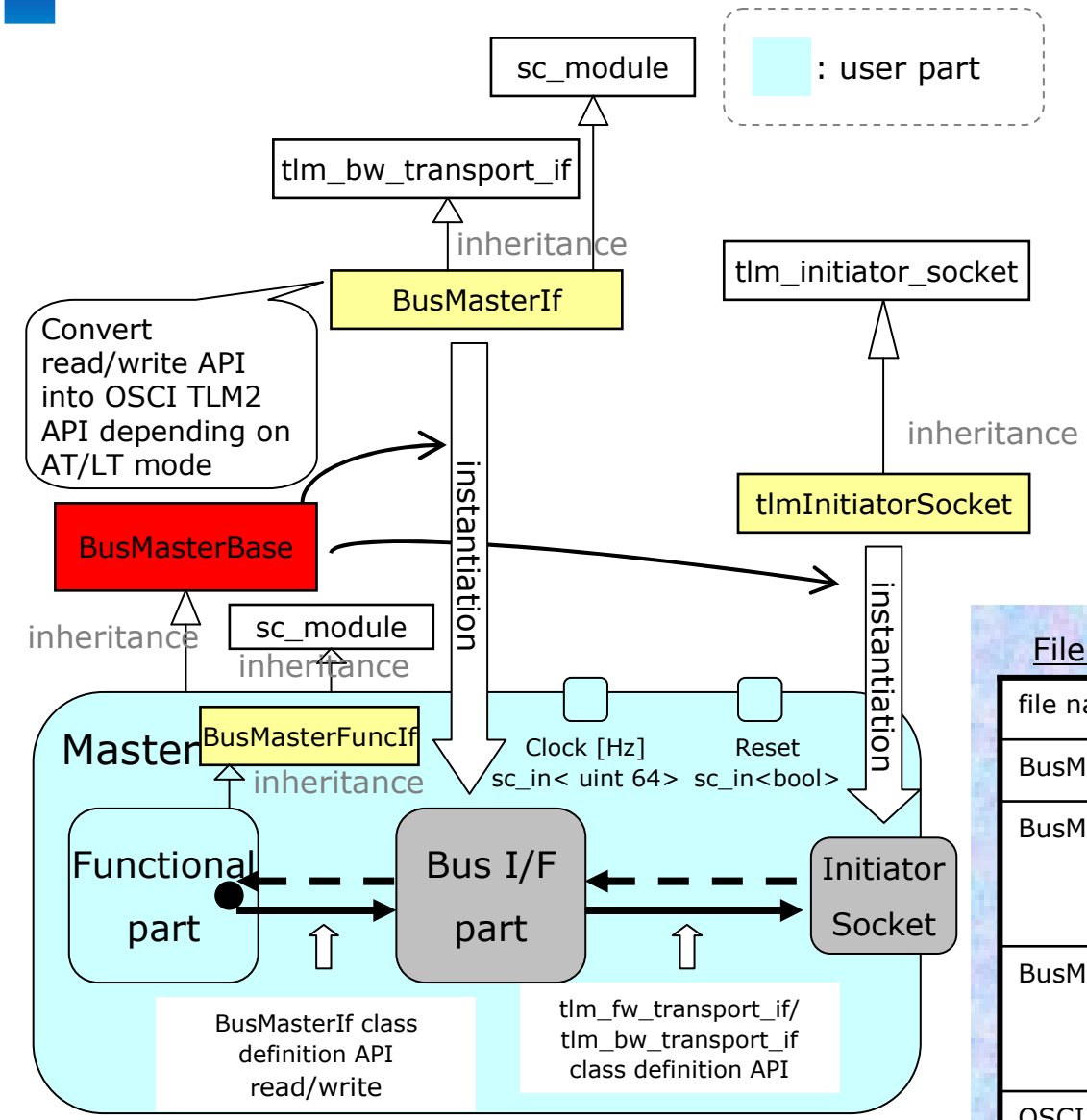
Read reset port value

Set reset port pointer

※: BusTime_t is typedef of sc_core::sc_time_t.
BusTimeUnit_t is typedef of sc_core::sc_time_unit.

3. How to create master/slave IP

How to create Bus master IP



procedure

1. Bus master IP is derived from BusMasterBase. (A bus width of socket and the number are specified by a template argument of BusMasterBase class.)
2. The Functional part class derived from BusMasterFuncIf class is instantiated.
3. Port and SystemC description are written in Bus master IP layer and the description should be separated from the Functional part.

File list related to Bus master IP creation

file name	explanation
BusMasterBase.h	Base class of bus master IP is defined.
BusMasterIf.h	Bus I/F class is defined. When BusMasterBase is inherited, this class is instantiated automatically in the bus master IP.
BusMasterFuncIf.h	Base class of Functional part of bus master IP is defined. API which is called by Bus I/F class is defined as pure virtual.
OSCI2.h	TlmInitiatorSocket class which is derived from tlm_initiator_socket and extended payload classes are defined.

Example of bus master IP
(64 bit width socket x 1, 32-bit width sockets x 2)

MASTER.h

```
#include "BusMasterBase.h"
#include "CoreFunction.h"

class MASTER :
  public sc_module, public BusMasterBase<64,1>, public BusMasterBase<32,2>
{
 public:
  SC_HAS_PROCESS( MASTER );
  MASTER( sc_module_name __name = "MASTER" );
  ~MASTER( void );

  void process( void );
  …
 public:
  sc_in<bool>    resetPort;
  sc_in<uint64> freqPortt;
 private:
  CoreFunction *mpCore;
};
```

Include BusMasterBase.h

Specify bus width and number of the socket.

This means that 2 sockets of the 32bit width are made.

Derived from sc_module

Derived from BusMasterBase

Necessary port is defined

Pointer of Functional part

Example of bus master IP
(64-bit width socket x 1, 32-bit width sockets x 2)

MASTER.cpp

```
#include "MASTER.h"

MASTER::MASTER( sc_module_name __name ) :
  sc_module( __name ),
  BusMasterBase<64,1>(),
  BusMasterBase<32,2>(),
  resetPort( "resetPort" ),
  freqPort( "freqPort" ),
  mpCore( (CoreFunction *)0 )
{
  // BusMasterBase setting for 64bit bus socket
  setMasterResetPort64( &resetPort );
  setMasterFreqPort64( &freqPort );
  setInitiatorSocket64( "isx" );
  // BusMasterBase setting for 32bit bus socket
  setMasterResetPort32( &resetPort, &resetPort );
  setMasterFreqPort32( &freqPort, &freqPort );
  setInitiatorSocket32( "isg", "isl" );
  // instantiation for Core function
  mpCore = new CoreFunction( mBusMasterIf64[0],
                             mBusMasterIf32[0],
                             mBusMasterIf32[1] );
  mBusMasterIf64[0]->setFuncModulePtr( mpCore );
  mBusMasterIf32[0]->setFuncModulePtr( mpCore );
  mBusMasterIf32[1]->setFuncModulePtr( mpCore );
  mBusMasterIf64[0]->setBusProtocol( BUS_AXI );
  mBusMasterIf32[0]->setBusProtocol( BUS_APB );
  mBusMasterIf32[1]->setBusProtocol( BUS_APB );
  mBusMasterIf64[0]->setTransNmbLmt( 10 );
  mBusMasterIf32[0]->setTransNmbLmt( 1 );
  mBusMasterIf32[1]->setTransNmbLmt( 1);

  SC_THREAD( process );
  …
}
```

```
MASTER::~MASTER( void ){}

void MASTER::process( void )
{
  while(1){
    …
    mpCore->transportProcess();
    …
  }
}
```

Reset port and frequency port pointers are set

Create sockets

Instantiate functional part

Set Functional part pointer to Bus I/F part

Set Bus I/F part pointer to Bus I/F part

Set bus protocol to Bus I/F part

Set number of transaction limit

The description in SystemC is written in outer fence of IP

Member function of BusMasterIf class

Member of BusMasterBase class

Example of bus master IP
(64 bit width socket x 1, 32-bit width sockets x 2)

CoreFunction.h

```
#include "BusMasterIf.h"
#include "BusMasterFuncIf.h"

class CoreFunction : BusMasterFuncIf{
 public:
  CoreFunction( BusMasterIf<64>*, BusMasterIf<32>*, BusMasterIf<32>* );
  ~CoreFunction(void);
  void transportProcess( void );
  void notifyTransSend( TlmTransactionType &trans, BusTime_t &t );
  void notifyTransComp( TlmTransactionType &trans, BusTime_t &t );
 private:
  BusMasterIf<64> *mBusIf1;
  BusMasterIf<32> *mBusIf2, *mBusIf3;
  unsigned char mData1[1024], mData2[1024], mData3[1024];
  TlmBasicPayload mPayload1, mPayload2, mPayload3;
  TlmG3mExtension mG3mExt1, mG3mExt2, mG3mExt3;
  …
};
```

Include BusMasterIf.h

Define the Functional part class

Derived from BusMasterFuncIf

Set Bus I/F part pointer

Overwrite pure virtual function

Create payload object and data storage.
At least these objects are kept between start and end of transaction.

Only TlmBasicPayload is used at debug access.

Example of bus master IP
(64 bit width socket x 1, 32-bit width sockets x 2)

CoreFuntion.cpp

```
#include "CoreFunction.h"

CoreFunction::CoreFunction(BusMasterIf<64> *busIf1,
                           BusMasterIf<32> *busIf2,
                           BusMasterIf<32> *busIf3 )
 : mBusIf1( busIf1 ), mBusIf2( busIf2 ), mBusIf3( busIf3 )
{
 for( int i; i<(int)1024; i++ ) mData[i] = 0;
 mPayload1.set_extension( &mG3mExt1 );
 mPayload2.set_extension( &mG3mExt2 );
 mPayload3.set_extension( &mG3mExt3 );
 …
}

CoreFunction::~CoreFunction( void ){}

void CoreFunction::transportProcess( void )
{
 mPayload1.set_address(0x10000001); // set payload
 …
 mBusIf1->write( mPayload1,… );        //write from socket1
 …
 mPayload2.set_address(0xFFFF0100); // set payload
 …
 mBufIf2->read( mPayload2,… );         //read from socket2
 …
}

void CoreFunction::notifyTransComp(TlmTransactionType &trans, BusTime_ &t )
{
 …
}

void CoreFunction::notifyTransComp(TlmTransactionType &trans, BusTime_t &t)
{
 …
}
```

Extension payload pointer set to generic payload.

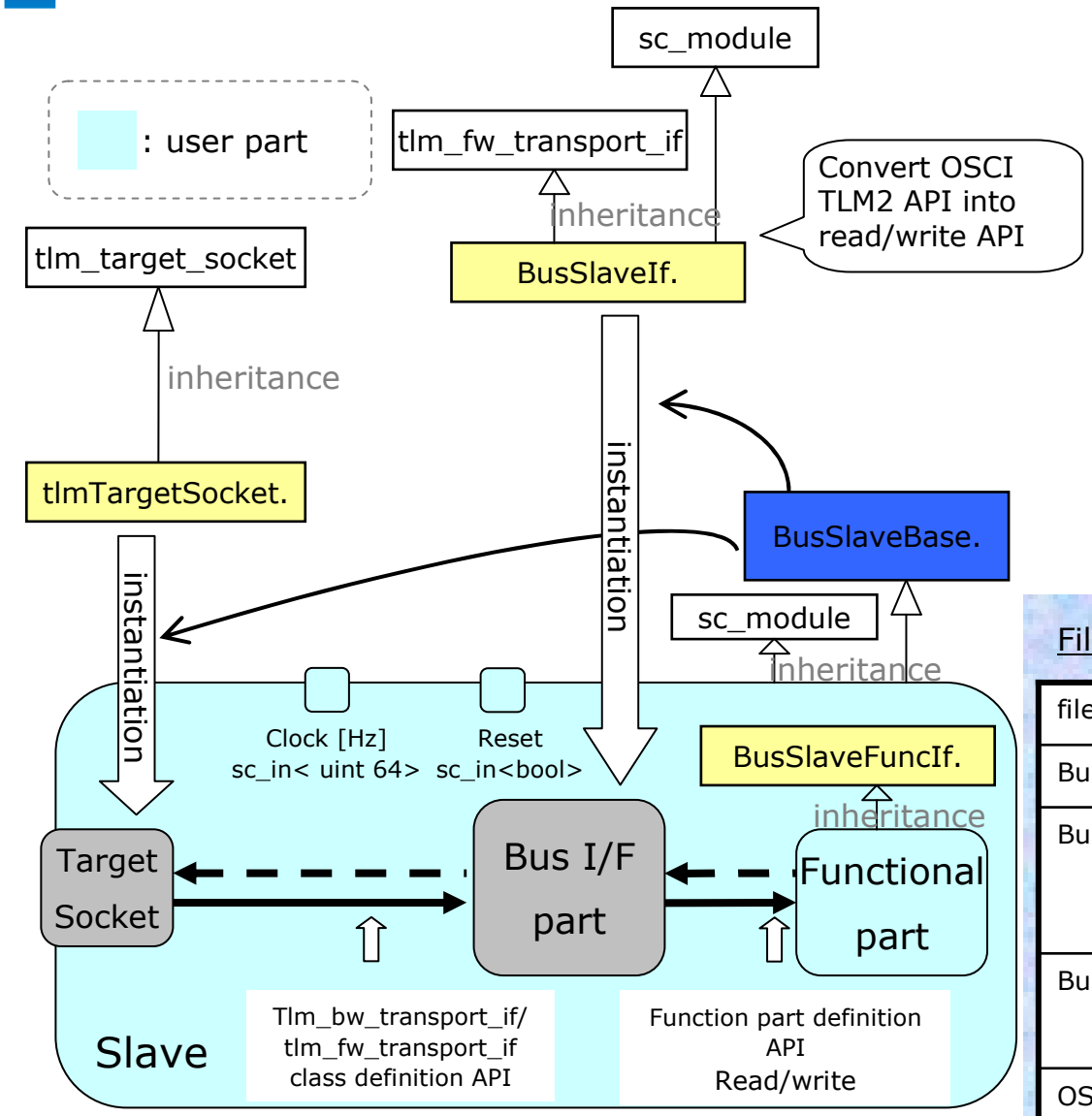
Identify I/F at transmission

Overwrite transaction transmission notice API

Overwrite transaction completion notice API

Data is get from trans.

How to create Bus slave IP



Procedure

1. Bus slave IP is derived from BusSlaveBase.(A bus width of socket and the number are designated by a template argument in a BusSlaveBase class.)
2. The Functional part class derived from BusSlaveFuncIf class is instantiated.
3. Port and SystemC description are written in Bus slave IP layer and the description should be separated from the Functional part.

File list related to Bus slave IP

file name	explanation
BusSlaveBase.h	Base class of bus slave IP is defined.
BusSlaveIf.h	Bus I/F class is defined. When BusSlaveBase is inherited, this class is instantiated automatically in the bus slave IP.
BusSlaveFuncIf.h	Base class of Functional part of bus slave IP is defined. API called by Bus I/F part is defined as pure virtual.
OSCI2.h	TlmTargetSocket class which is derived from tlm_target_socket and extended payload classes are defined.

Example of Bus slave IP (32-bit width socket x 1)

Include BusSlaveBase.h

Specify bus width and the number of socket.

The description in SystemC is written in outer fence of IP

SLAVE.h

```
#include "BusSlaveBase.h"
#include "MemoryFunction.h"

class SLAVE
 : public sc_module, public BusSlaveBase<32,1>
{
 public:
  MemoryFunction *mFunc;

  SLAVE( sc_module_name name,
         unsigned int       rLatency,
         unsigned int       wLatency );

  ~SLAVE(void);

  sc_in<bool>    resetPort;

  sc_in<uint64> freqPortt;

};
```

Derived from sc_module

Derived from BusMasterBase

Functional part pointer

Necessary port is defined

SLAVE.cpp

```
#include "SLAVE.h"

SLAVE::SLAVE( sc_module_name name,
              unsigned int       rILatency,
              unsigned int       wILatency,
              unsigned int       rFDLatency,
              unsigned int       wFDLatency,
              unsigned int       rNDLatency,
              unsigned int       wNDLatency ):
   sc_module( name ),
   BusSlaveBase<32,1>()
{
   setSlaveResetPort32( &resetPort );
   setSlaveFreqPort32( &freqPort );
   setTargetSocket32( "ts" );
   mFunc = new MemoryFunction();
   mBusSlaveIf32[0]->setFuncPtr( mFunc );
   mBusSlaveIf32[0]->setReadInitLatency( rILatency );
   mBusSlaveIf32[0]->setWriteInitLatency( wILatency );
   mBusSlaveIf32[0]->setReadFirstDataLatency( rFDLatency );
   mBusSlaveIf32[0]->setWriteFirstDataLatency( wFDLatency );
   mBusSlaveIf32[0]->setReadNextDataLatency( rNDLatency );
   mBusSlaveIf32[0]->setWriteNextDataLatency( wNDLatency );
   mBusSlaveIf32[0]->setBusProtocol( BUS_APB );
   mBusSlaveIf32[0]->setTransNmbLmt( 1 );
}

SLAVE::~SLAVE(void){}
```

Instantiate functional part

Set latency to Bus I/F

Member of BusSlaveBase class

Member of BusSlaveIf class

Example of Bus slave IP (32-bit width socket x 1)

Include BusSlaveFuncIf.h

MemoryFunction.h

```
#include "BusSlaveFuncIf.h"

class MemoryFunction
    : public BusSlaveFuncIf
{
 public:
  MemoryFunction(void);
  ~MemoryFunction(void);
  void read( TlmBasicPayload &payload,… );
  void write( TlmBasicPayload &payload,… );
  void setfunc( ADDRESS_TYPE size );
 private:
  unsigned char *mMem;
};
```

Derived from BusSlaveFuncIf

read/write must be overwritten. setfunc should be overwritten if setfunc is used

MemoryFunction.cpp

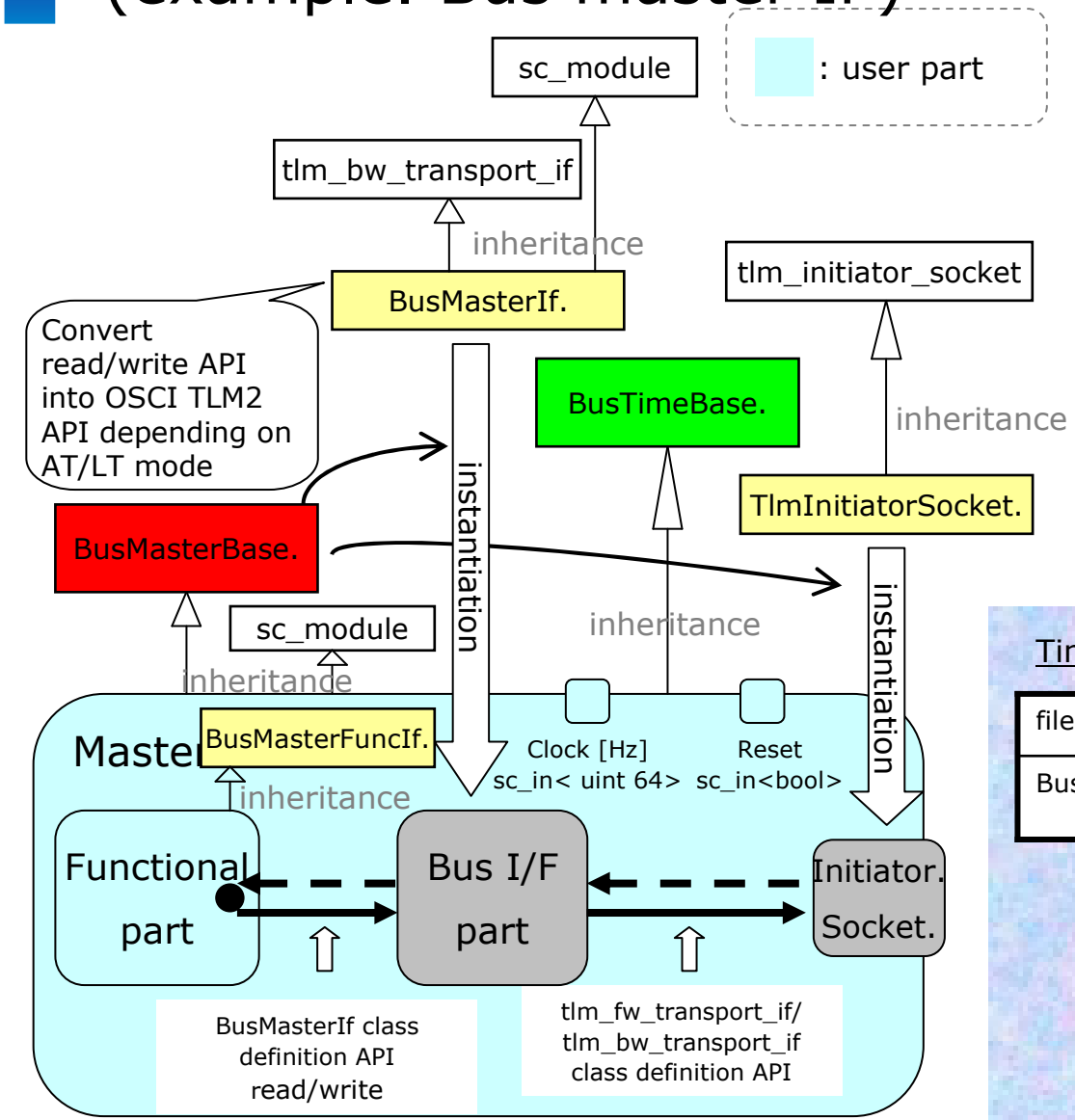
```
SLAVE::MemoryFunction::MemoryFunction(void){}
SLAVE::MemoryFunction::~MemoryFunction(void){}

void SLAVE::MemoryFunction::
read( TlmBasicPayload &payload, … )
{
  memcpy( payload.Data, &mMem[payload.Address], payload.Length );
}

void SLAVE::MemoryFunction::
write( TlmBasicPayload &payload, … )
{
  memcpy( &mMem[payload.Address], payload.Data, payload.Length );
}

void SLAVE::MemoryFunction::
setfunc( ADDRESS_TYPE size )
{
  mMem = new unsigned char [size];
  for( int i=0; i<size; i++ )  mMem[i] = 0;
}
```

Read API

Write API

Memory size set API needed if memory size is set by bus.

How to create Time spending class (example. Bus master IP)



Procedure

1. IP is created by being derived from BusTimeBase in addition to a bus master IP making procedure.

Time management related file list

file name	explanation
BusTimeBase.h	API to manage time in Functional part in the IP is defined.

Example of Time spending class

- Regardless of master or slave, the module which spends time in the Functional part, should be derived from the base class. .

MASTER.h

```
#include "BusMasterBase.h"
#include "BusTimeBase.h"
#include "CoreFunction.h"
class MASTER
 : public sc_module,
   public BusMasterBase<32,1>,
   BusTimeBase
{

  CoreFunction *mFunc;

  SC_HAS_PROCESS( MASTER );
  MASTER( … );
  ~MASTER(void);

  void process( void );
  …
};
```

Derived from BusTimeBase

MASTER.cpp

```
#include "master.h"

MASTER::MASTER( sc_module_name name, … ):
    sc_module( name ),
    BusSlaveBase<32,1>(),
    BusTimeBase()
{
  setResetForBusTimeBase( &resetPort );
  …
  mFunc = new CoreFunction( …, this );
  …
}

void MASTER::process( void )
{
  while{
    mFunc->coreProcess( … );
  }
}
```

Set reset port pointer

CoreFunction.h

```
#include "BusMasterIf.h"

class CoreFunction
{
 public:
  CoreFunction(…);
  ~CoreFunction(void);
  void coreProcess( … );
  …
 private:
  mpParent;
  …
};
```

CoreFunction.cpp

```
#include "CoreFunction.h"

CoreFunction::CoreFunction( BusTimeBase *parent )
:mpParent(parent)
{}

void CoreFunction::coreProcess( … )
{
  …
  parent->elapseTime( 10, SC_NS );
  …
}
```

Time spending

API of BusTimeBase class

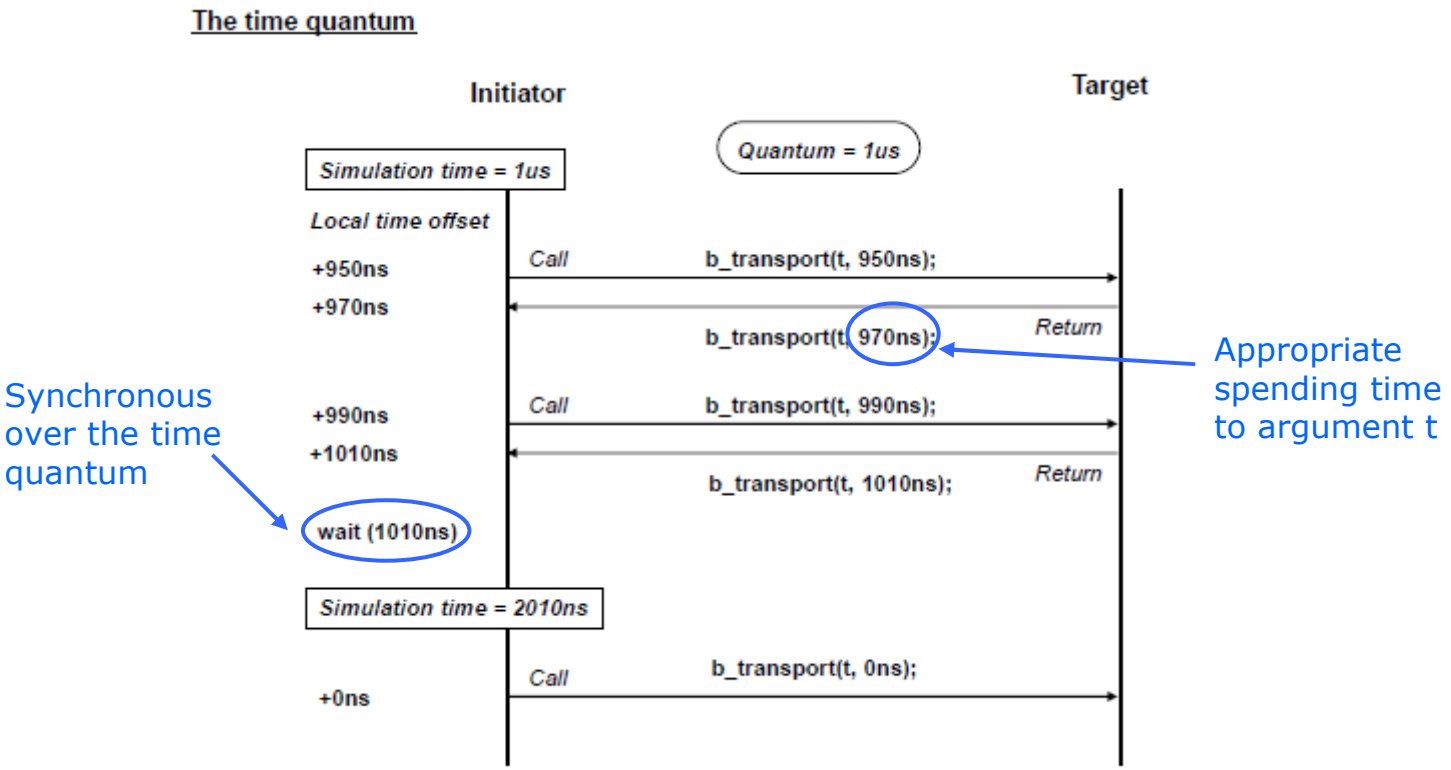
The notice at the IP creation

- Transaction object (payload) should be created by Functional part of bus master in all communication mode (AT,LT and debug access), and it's necessary to keep the object until transaction is completed. Bus slave must not create another transaction object by itself in backward path (nb_transaction_bw) and must use the transaction object received by a forward pass (nb_transaction_fw) in AT.
- Should consider that there is a possibility that a debug transaction processing performs in the normal transaction processing.

4. OSCI TLM2.0 coding style

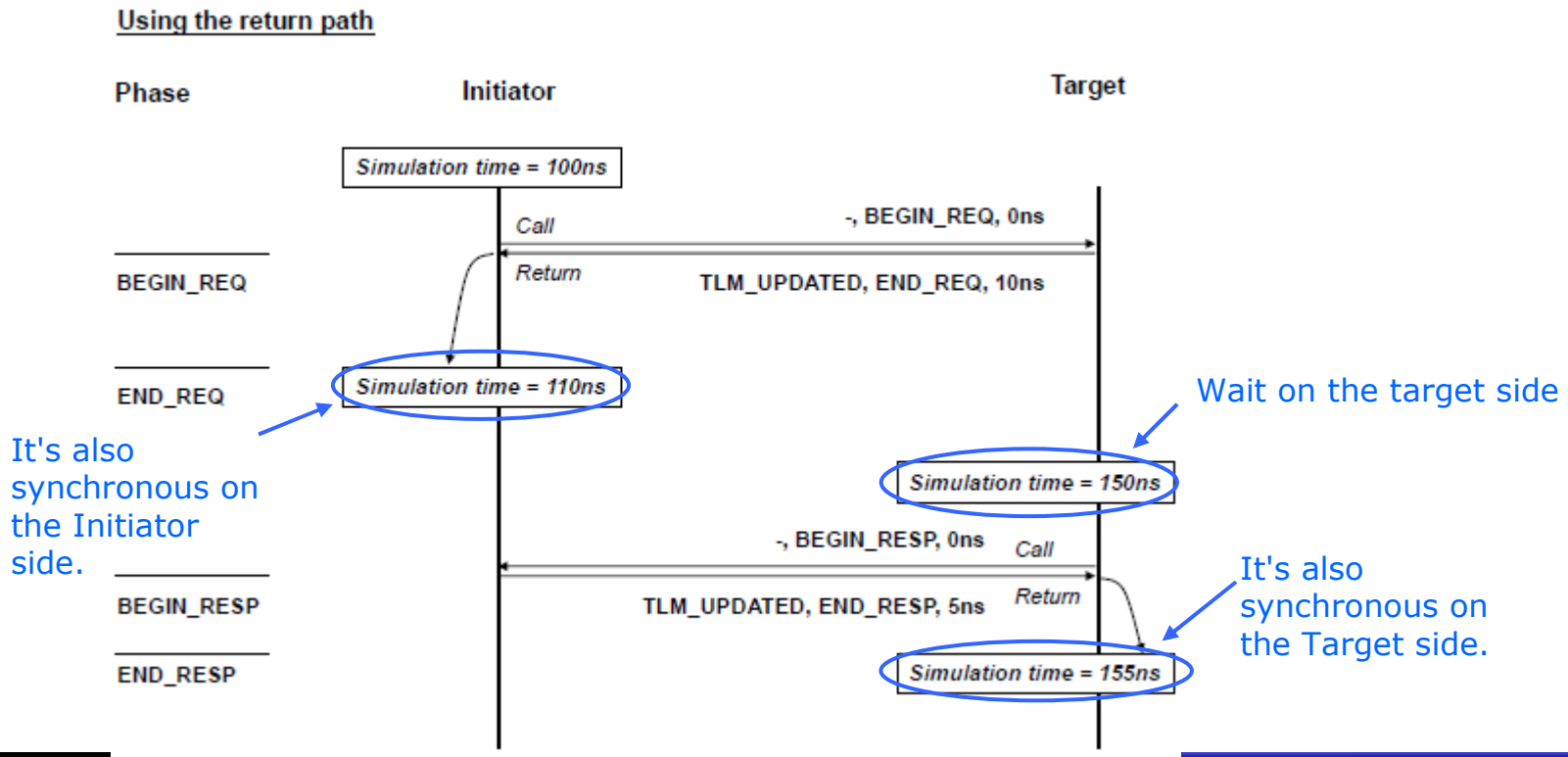
Coding style: Fast mode (LT)

- LT with a temporal decoupling is used.
- Interconnect and slave IP appropriate the spending time to 2nd argument t of b_transport and bus master synchronizes simulation time in any timing.



Coding style : accuracy mode (2 phase AT)

- AT using the return pass
- Basically Time is spent by the IP which actually spend the time.



5. Definition of bus communication timing and parameterization of latency

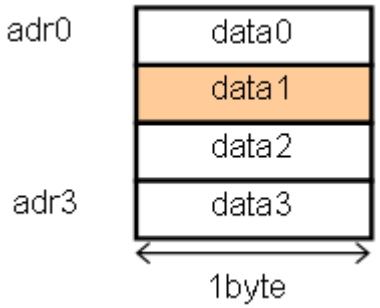
Refer to “SC-HEAP_E3: bus IF TLM timing specification(ZSG-F31-12-0030-01)”

6. Data packing

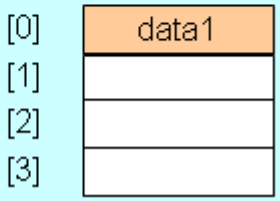
Data packing

Data is packed from the identified start address only for the size.

Ex. The case of 1 byte access to the address which is 1 offset address in 4 bytes alignment.



Data packed in generic payload
size : 1byte
address : adr1
data : array of unsigned char (passed by reference)
data[0]= **data1**、 [1]= don't care、 [2]=don't care、 [3]=don't care
image :



※ The byte enable pointer attribute of tlm_generic_payload isn't used.

7. Payload

Payload

Use generic payload + extension payload by tlm_extension class of OSCI TLM-2.0 is used.

Followings are used according to the bus.

[Payload list]

Class name	Function	Note
TlmBasicPayload.	Basic payload which includes command, address and data.	Typedef of tlm_generic_payload
TlmG3mExtension	G3MSS common side band signals	The extension payload which derived from tlm_extension class
TlmAxiExtension.	Extension payload for AXI	The extension payload which derived from tlm_extension class
TlmVpiExtension.	Extension payload for VPI	The extension payload which derived from tlm_extension class
TlmAhbExtension.	Extension payload for AHB	The extension payload which derived from tlm_extension class
TlmApbExtension.	Extension payload for APB	The extension payload which derived from tlm_extension class

Payload handling

3 payloads are used by AXI bus.
- TlmBasicPayload.
- TlmG3mExtension
- TlmAxiExtension.

Ex. AXI bus

Case of payload set in bus master

```
TlmBasicPayload mBasicPayload;                          //create basic payload
mBasicPayload.set_read();                               //set payload members
…
TlmG3mExtension mG3mExtension;                          //create G3M extension payload
mG3mExtension.setDmaAccess(false);                      //set payload members
…
TlmAxiExtension mAxiExtension;                          //create AXI extension payload
mAxiExtension.setBurstType(INCR);                       //set payload members
…
mBasicPayload.setExtension( mG3mExtension );
mBasicPayload.setExtension( mAxiExtension );
read( mBasicPayload );                                  //transmission
```

Case of payload get in bus slave

```
void read( TlmBasicPayload &payload ){
  …
  bool isRead = payload.is_read();                      //get the value
  …
  TlmG3mExtension *G3mExtension;
  payload.get_extension( G3mExtension );
  bool dmaAccess = G3mExtension->getDmaAccess();        //get the value
  …
  TlmAxiExtension *AxiExtension;
  payload.get_extension( AxiExtension );
  BURST_TYPE burstType = AxiExtension->getBurstType(); //get the value
  …
}
```

Payload member(1)

[Basic payload (TlmBasicPayload)]

member name	fuction name	bit number	type	bus	signal on the hardware	API
m_address	address	64	uint64	all buses	Payload	void set_address (unsigned char*) uint64 get_address (void)
m_command	command	-	BUS_COMMAND (typedef of tlm_command)	all buses	Payload	void set_write (void) void set_read (void) bool is_write (void) bool is_read (void) (It's set as read in case of all except for read/write.)
m_data	data pointer	64	unsigned char*	all buses	Payload	void set_data_ptr (unsigned char*) unsigned char* get_data_ptr (void)
m_length	data length	-	unsigned int	all buses	Payload	void set_data_length (unsigned int) unsigned int get_data_length (void)

Payload member(2)

[G3M subsystem common side band signal (TlmG3mExtension)]

member name	function name	bit number	type	bus use	signal on the hardware	API
mDmaAccess	DMA access	1	bool	all bus	G3MSS common side band	void setDmaAccess(bool) bool isDmaAccess(void)
mTcId	TCID (thread identifier)	6	unsigned char	all bus	G3MSS common side band	void setTcId(unsigned char) unsigned char getTcId(void)
mVcId	VCID (virtual machine identifier)	3	unsigned char	all bus	G3MSS common side band	void setVcId(unsigned char) unsigned char getVcId(void)
mPeId	PEID (PE identifier)	3	unsigned char	all bus	G3MSS common side band	void setPeId(unsigned char) unsigned char getPeId(void)
mSpId	SPID (system protection identifier)	2	unsigned char	all bus	G3MSS common side band	void setSpId(unsigned char) unsigned char getSpId(void)
mUserMode	UM (0:SV and the 1: user)	1	bool	all bus	G3MSS common side band	void setUserMode(bool) bool isUserMode(void)
mVirtualMode	VM (0: native and 1: virtual)	1	bool	all bus	G3MSS common side band	void setVirtualMode(bool) bool isVirtualMode(void)

Payload member(3)

[Extended payload for AXI (TlmAxiExtension)]

Member name	function name	bit number	Type	bus use	signal on the hardware	API	note
mBurstType	burst type	2	enum AxiBurst_t	AXI	signal for AXI I/F	void setBurstType(AxiBurst_t) AxiBurst_t getBurstType(void)	
mLock	lock type	1	bool	AXI	signal for AXI I/F	void setLock(bool) bool isLock(void)	
mCachable	cachable/ uncachable	1	bool	AXI	signal for AXI I/F	void setCachable(bool) bool isCachable(void)	
mBufferable	bufferable/ unbufferable	1	bool	AXI	signal for AXI I/F	void setBufferable(bool) bool isBufferable(void)	
mSecure	protection type	3	unsigned char	AXI	signal for AXI I/F	void setSecure(unsigned char) unsigned char getSecure(void)	
mTransId	transaction ID	depend on master number	unsigned int	AXI	signal for AXI I/F	void setId(unsigned int) uint32 getId(void)	
mBitOpType	Side band signal (bit operation type)	2	enum AxiBitop_t	AXI	AXI bus side band	void setBitOpType(AxiBitop_t) AxiBitop_t getBitOpType(void)	
mBitOpPos	Side band signal (bit position at bit operation)	3	unsigned char	AXI	AXI bus side band	void setBitOpPos(unsigned char) unsigned char getBitOpPos(void)	

Payload member(4)

[Extended payload for VPI (TlmVpiExtension)]

member name	function name	bit number	type	bus use	signal on the hardware	API	note
mMasterId	master ID	3	unsigned int	VPI	Signal for VPI I/F	void setMasterId(unsigned int) unsigned int getMasterId(void)	
mPacketId	packet ID	6	unsigned int	VPI	Signal for VPI I/F	void setPacketId(unsigned int) unsigned int getPacketId(void)	
mSlaveId	slave ID	4	unsigned int	VPI	Signal for VPI I/F	void setSlaveId(unsigned int) unsigned int getSlaveId(void)	
mRequestType	request type	4	enum VpiRequest_t	VPI	Signal for VPI I/F	void setRequestType(VpiRequest_t) VpiRequest_t getRequestType(void)	

Payload member(5)

[Extended payload for AHB (TlmAhbExtension)]

member name	function name	bit number	type	bus use	signal on the hardware	API	Note
mBurstType	burst type	2	enum AhbBurst_t	AHB	Signal for AHB I/F	void setBurstType(AhbBurst_t) AhbBurst_t getBurstType (void)	
mLock	lock type	1	bool	AHB	Signal for AHB I/F	void setLock(bool) bool isLock(void)	
mCachable	cachable/ uncachable	1	bool	AHB	Signal for AHB I/F	void setCachable(bool) bool isCachable(void)	
mbufferable	bufferable/ unbufferable	1	bool	AHB	Signal for AHB I/F	void setBufferable(bool) bool isBufferable(void)	
mPrivilege	privilege/user	1	bool	AHB	Signal for AHB I/F	void setPrivilege(bool) bool isPrivilege (void)	
mDataOrOp	Data/OP code	1	enum AhbDataOrOP_t	AHB	Signal for AHB I/F	void setDataOrOp(AhbDataOrOp_t) AhbDataOrOp_t getDataOrOp(void)	

Payload member(6)

[Extended payload for APB (TlmApbExtension)]

member name	function name	bit number	type	bus use	signal on the hardware	API	note
mLock	lock signal	1	bool	APB	APB side band	void setLock(bool) bool isLock(void)	
mExclusion	Signal for exclusion	1	Bool	APB	APB side band	void setExclusion(bool) bool isExclusion(void)	

LLWEB-00010925 ZSG-F31-12-0029-01

Renesas Electronics Corporation