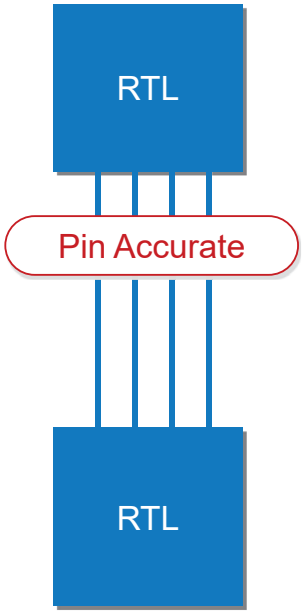
The Definitive Guide to SystemC: TLM-2.0 and the IEEE 1666-2011 Standard

David C Black, Doulos

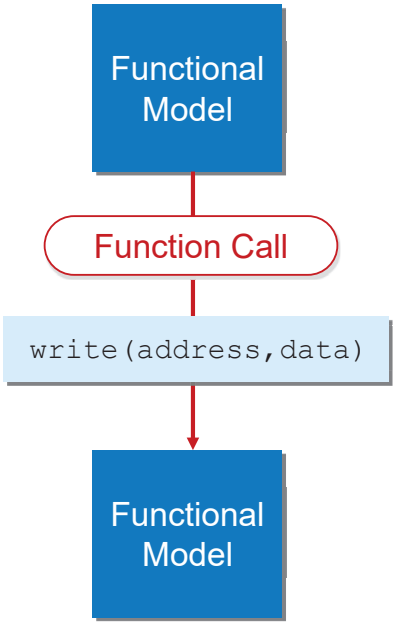
Track 3: The Definitive Guide to SystemC
TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

Transaction Level Modeling



Simulate every event!



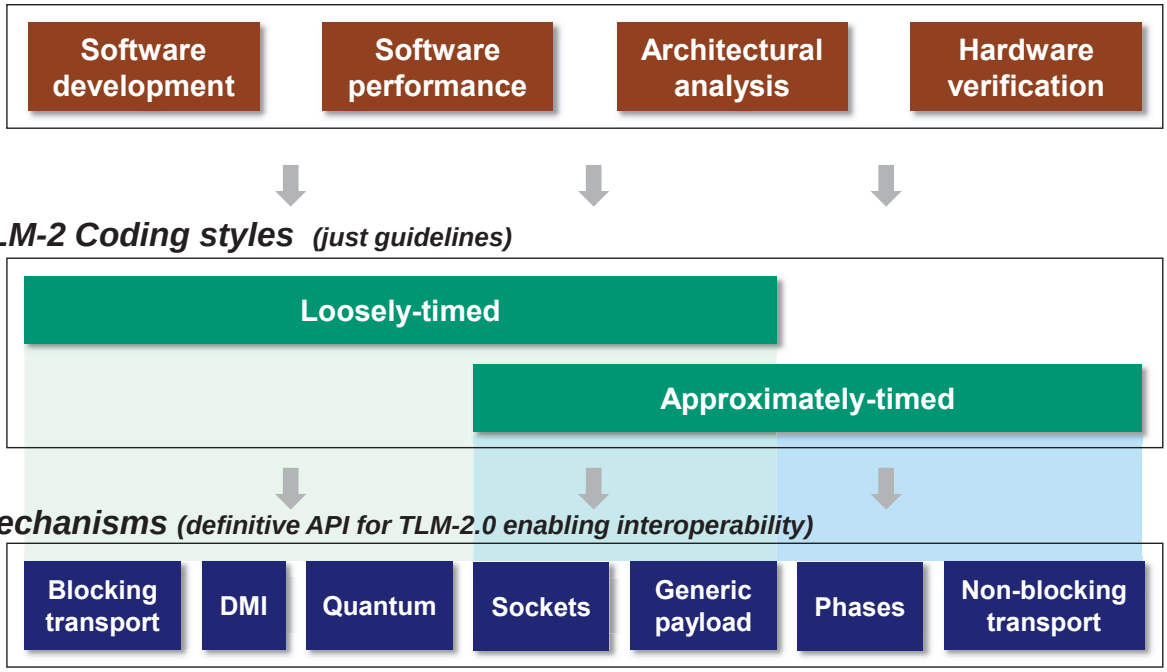
100-10,000 X faster simulation!

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

Use Cases, Coding Styles and Mechanisms

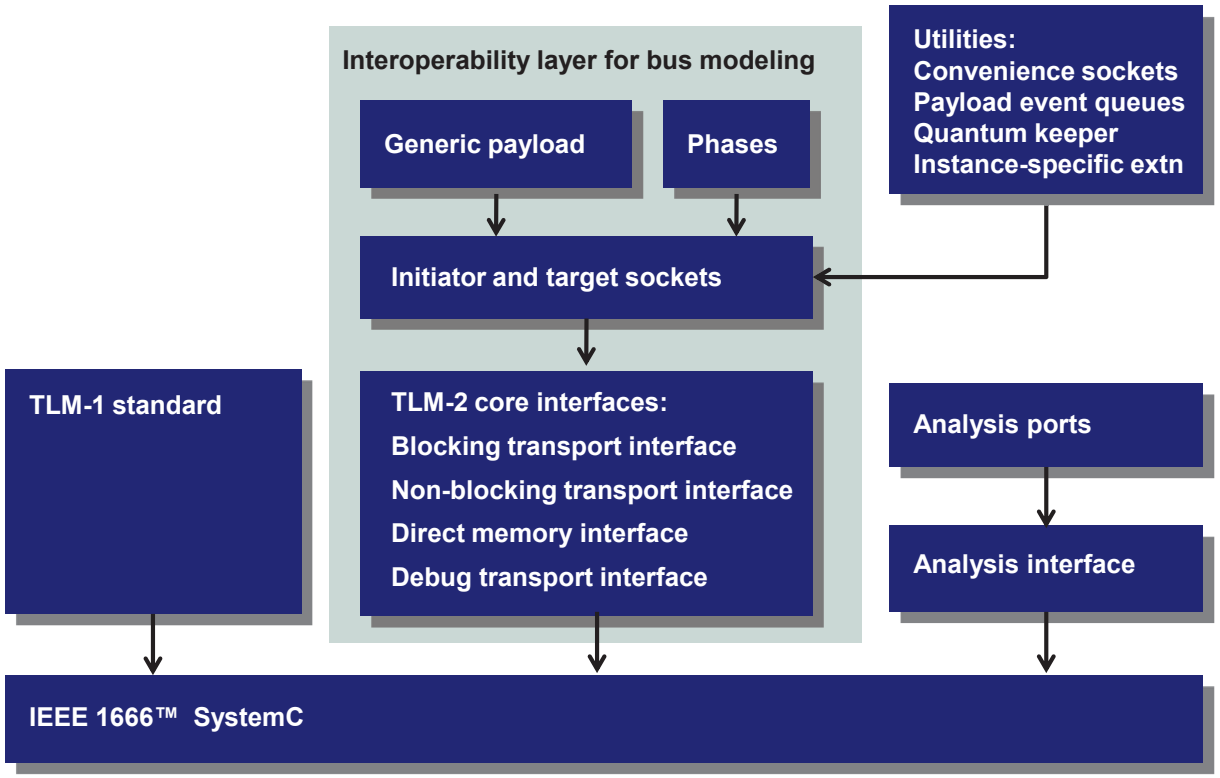
Use cases



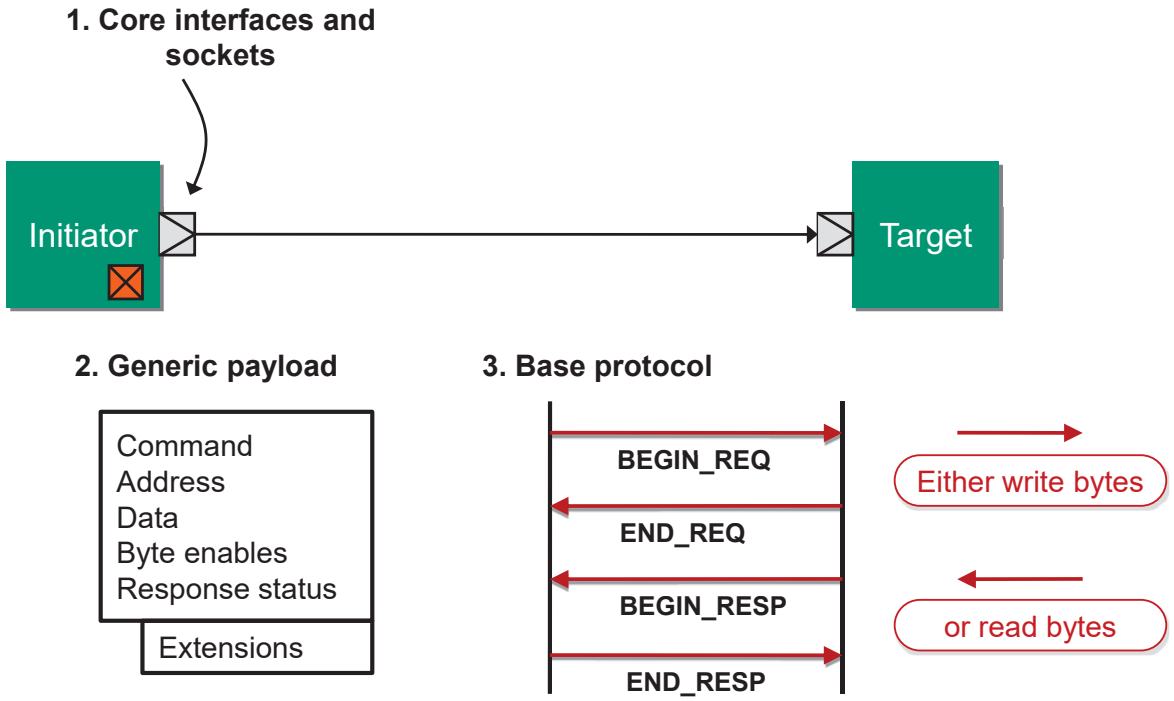
Coding Styles

- Loosely-timed = as fast as possible
  - Register-accurate
  - Only sufficient timing detail to boot O/S and run multi-core systems
  - b_transport – each transaction completes in one function call
  - Temporal decoupling
  - Direct memory interface (DMI)
- Approximately-timed = just accurate enough for performance modeling
  - *aka* cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - nb_transport – each transaction has 4 timing points (extensible)
- Guidelines only – not definitive

The TLM 2.0 Classes

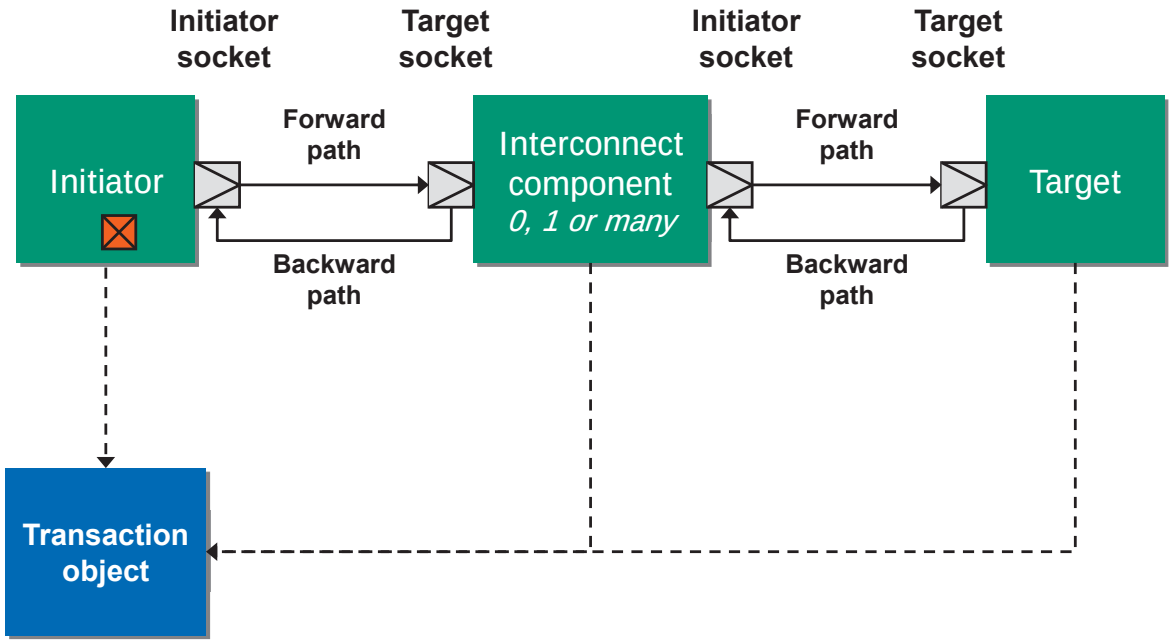


Interoperability Layer



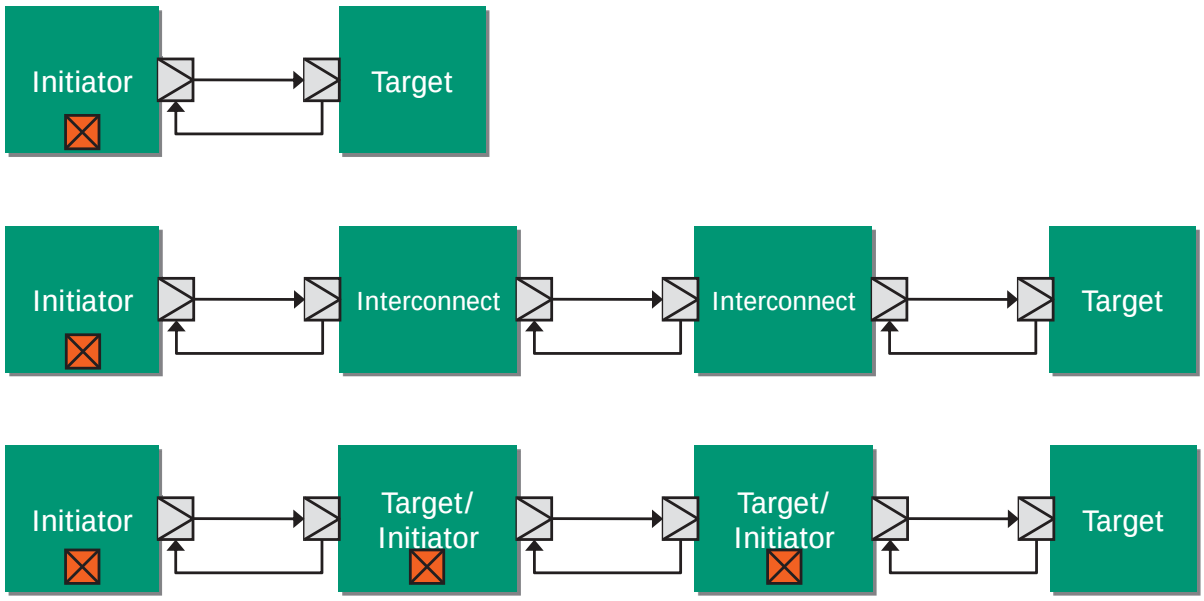
- Maximal interoperability for memory-mapped bus models

Initiators, Targets and Interconnect



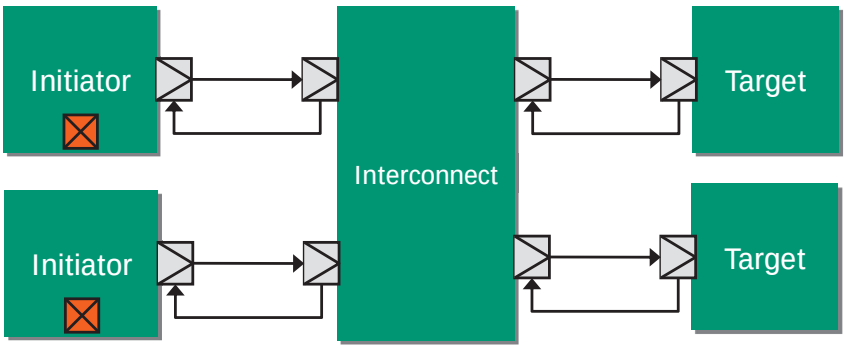
- Single transaction object for request and response
- References to the object are passed along the forward and backward paths

TLM-2 Connectivity



- Roles are dynamic; a component can choose whether to act as interconnect or target
- Transaction memory management needed

Convergent Paths

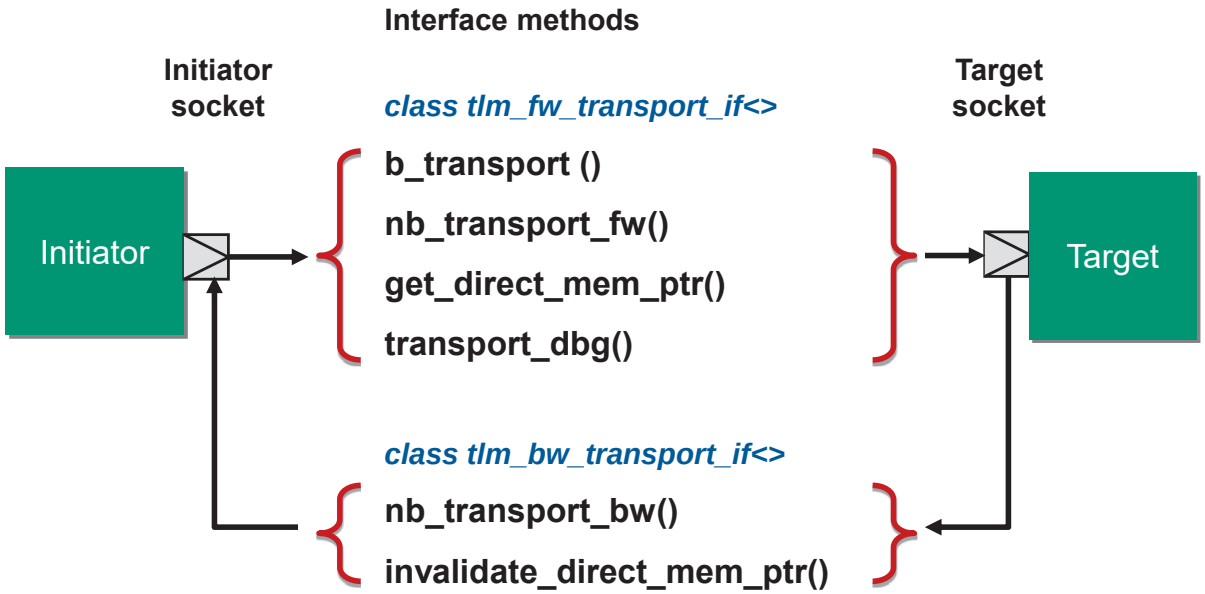


- Paths not predefined; routing may depend on transaction attributes (e.g. address)
- Whether arbitration is needed depends on the coding style

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

Initiator and Target Sockets



- Sockets provide fw and bw paths, and group interfaces

Initiator Socket

```
#include "tlm.h"
struct  Initiator: sc_module,  tlm::tlm_bw_transport_if<>
{
   tlm::tlm_initiator_socket<>  init_socket;

   SC_CTOR(Initiator) : init_socket("init_socket") {
      SC_THREAD(thread);
      init_socket.bind( *this );
   }

   void thread() { ...
      init_socket->b_transport( trans, delay );
      init_socket->nb_transport_fw( trans, phase, delay );
      init_socket->get_direct_mem_ptr( trans, dmi_data );
      init_socket->transport_dbg( trans );
   }

   virtual  tlm::tlm_sync_enum  nb_transport_bw( ... ) { ... }
   virtual void  invalidate_direct_mem_ptr( ... ) { ... }
};
```

Combined interface required by socket

Protocol type defaults to base protocol

Initiator socket bound to initiator itself

Calls on forward path

Methods for backward path

tlm_initiator_socket must be bound to object that implements entire bw interface

Target Socket


```
struct  Target: sc_module,  tlm::tlm_fw_transport_if<>
{
   tlm::tlm_target_socket<>  targ_socket;

   SC_CTOR(Target) : targ_socket("targ_socket") {
      targ_socket.bind( *this );
   }
   virual void b_transport( ... ) { ... }
   virtual tlm::tlm_sync_enum  nb_transport_fw( ... ) { ... }
   virtual bool get_direct_mem_ptr( ... ) { ... }
   virtual unsigned int transport_dbg( ... ) { ... }
};
```

Combined interface required by socket

Protocol type default to base protocol

Target socket bound to target itself

Methods for forward path

tlm_target_socket must be bound to object that implements entire fw interface

Socket Binding


```
SC_MODULE(Top)  {
   Initiator  *init;
   Target     *targ;

   SC_CTOR(Top)  {
      init  = new Initiator("init");
      targ = new Target("targ");
      init->init_socket.bind( targ->targ_socket );
   }
   ...
```

Bind initiator socket to target socket

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

The Generic Payload

- Has typical attributes of a memory-mapped bus

Attribute	Type	Modifiable?
Command	tlm_command	No
Address	uint64	Interconnect only
Data pointer	unsigned char*	No (array – yes)
Data length	unsigned int	No
Byte enable pointer	unsigned char*	No
Byte enable length	unsigned int	No
Streaming width	unsigned int	No
DMI hint	bool	Yes
Response status	tlm_response_status	Target only
Extensions	(tlm_extension_base*)[ ]	Yes

Response Status

enum tlm_response_status	Meaning
TLM_OK_RESPONSE	Successful
TLM_INCOMPLETE_RESPONSE	Transaction not delivered to target (default)
TLM_ADDRESS_ERROR_RESPONSE	Unable to act on address
TLM_COMMAND_ERROR_RESPONSE	Unable to execute command
TLM_BURST_ERROR_RESPONSE	Unable to act on data length/ streaming width
TLM_BYTE_ENABLE_ERROR_RESPONSE	Unable to act on byte enable
TLM_GENERIC_ERROR_RESPONSE	Any other error

- Set to TLM_INCOMPLETE_RESPONSE by the initiator
- May be modified by the target
- Checked by the initiator when transaction is complete

Generic Payload - Initiator


```
void thread_process() {  // The initiator
    tlm::tlm_generic_payload  trans;
    sc_time  delay = SC_ZERO_TIME;

    trans.set_command( tlm::TLM_WRITE_COMMAND );
    trans.set_data_length( 4 );
    trans.set_streaming_width( 4 );
    trans.set_byte_enable_ptr( 0 );

    for ( int i = 0; i < RUN_LENGTH; i += 4 ) {
        int  word = i;
        trans.set_address( i );
        trans.set_data_ptr( (unsigned char*)( &word ) );
        trans.set_dmi_allowed( false );
        trans.set_response_status( tlm::TLM_INCOMPLETE_RESPONSE );

        init_socket->b_transport( trans, delay );

        if ( trans.is_response_error() )
            SC_REPORT_ERROR("TLM-2", trans.get_response_string().c_str());
        ...
}
```

Would usually pool transactions

8 attributes you must set

Generic Payload - Target


```
virtual  void  b_transport(  // The target
    tlm::tlm_generic_payload& trans,  sc_core::sc_time& t )
{
    tlm::tlm_command    cmd   = trans.get_command();
    sc_dt::uint64       adr   = trans.get_address();
    unsigned char*      ptr   = trans.get_data_ptr();
    unsigned int        len   = trans.get_data_length();
    unsigned char*      byt   = trans.get_byte_enable_ptr();
    unsigned int        wid   = trans.get_streaming_width();

    if ( byt != 0 || len > 4 || wid < len || adr+len > memsize ) {
        trans.set_response_status( tlm::TLM_GENERIC_ERROR_RESPONSE );
        return;
    }

    if ( cmd == tlm::TLM_WRITE_COMMAND )
        memcpy( &m_storage[adr], ptr, len );
    else if ( cmd == tlm::TLM_READ_COMMAND )
        memcpy( ptr, &m_storage[adr], len );

    trans.set_response_status( tlm::TLM_OK_RESPONSE );
}
```

6 attributes you must check

Target supports 1-word transfers

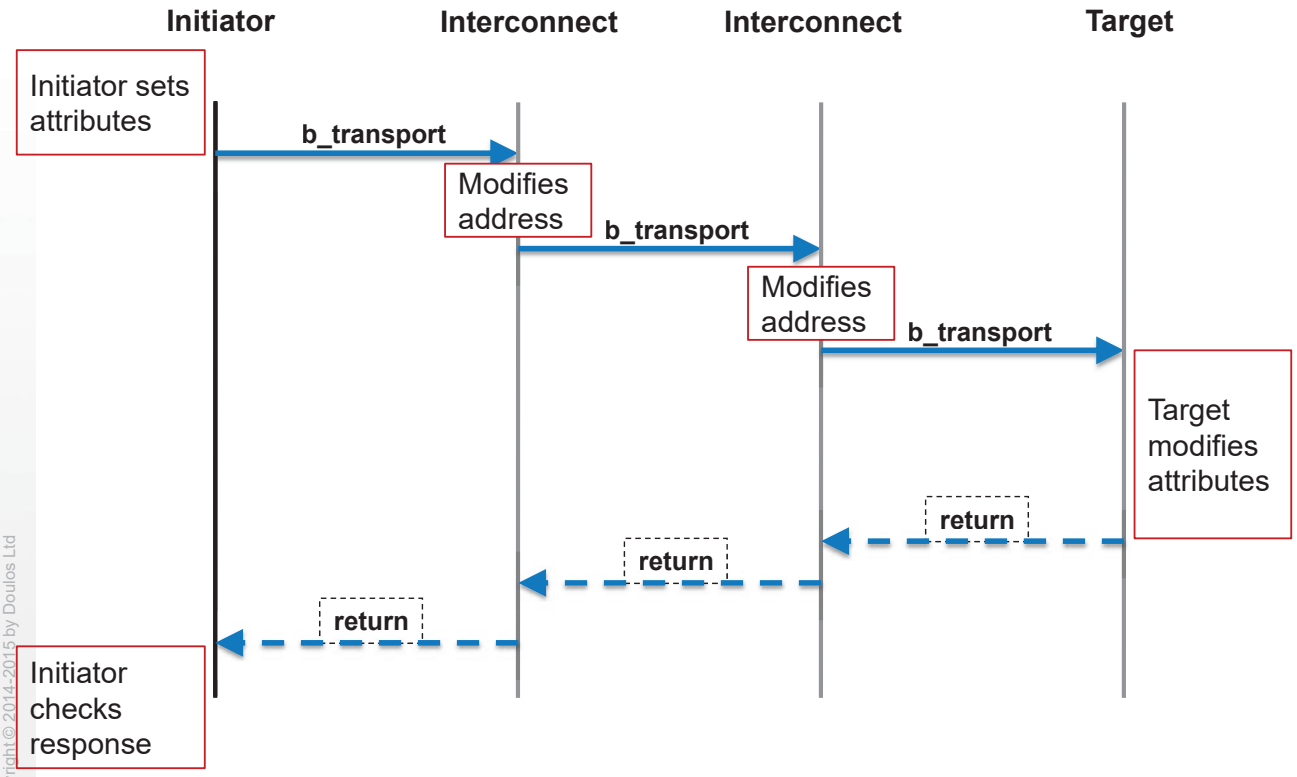
Execute command

Successful completion

TLM-2.0 and IEEE 1666-2011

- What is IEEE-1666-2011
- Transaction Level Modeling
- The architecture of TLM-2.0
- Initiator, interconnect, target & Sockets
- The generic payload
- Loosely-timed coding style
- Extensions & Interoperability
- Process Control
- sc_vector

Causality with b_transport



Timing Annotation and b_transport

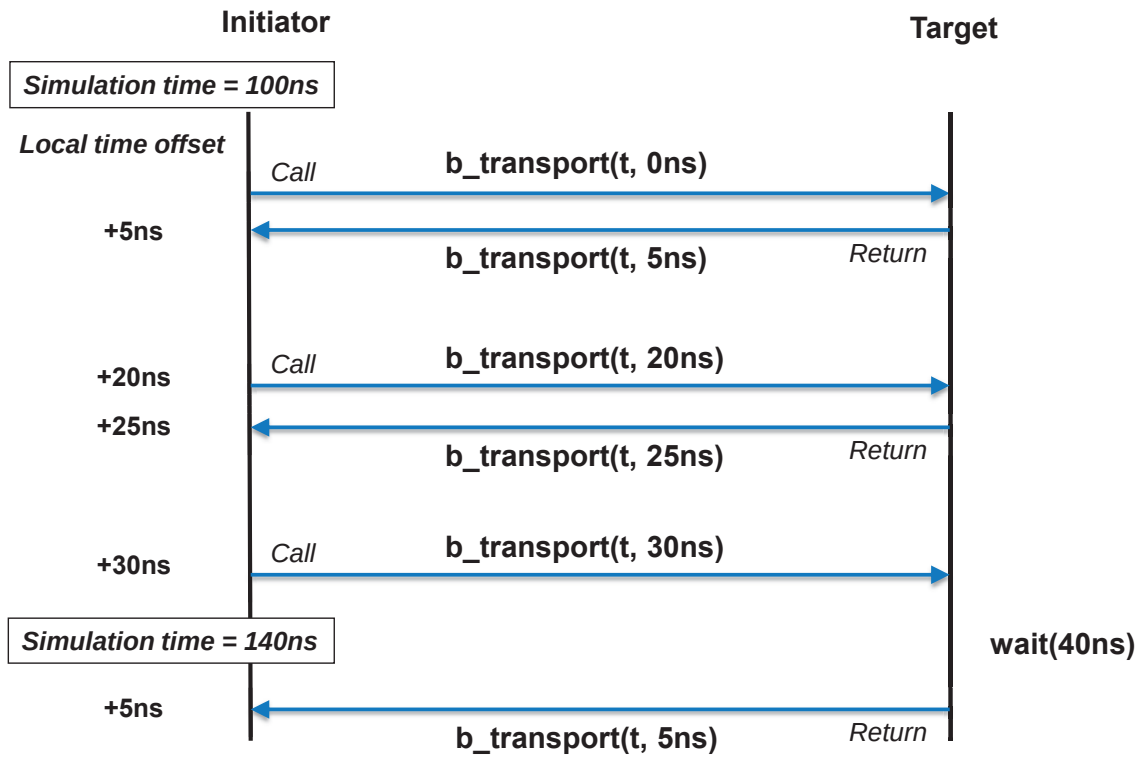
```
virtual  void  b_transport ( TRANS& trans , sc_core::sc_time& delay )
{
  Behave as if method were called at sc_time_stamp() + delay
  ...
  delay = delay + latency;
}
```

```
socket->b_transport( transaction, delay );

Behave as if method returned at sc_time_stamp() + delay
```

- Recipient may
  - Execute transactions immediately, out-of-order – Loosely-timed
  - Schedule transactions to execute at proper time – Approx-timed
  - Pass on the transaction with the timing annotation

Temporal Decoupling



Direct Memory Interface