SystemC Lab – Introduction

Part 3

Dr. Thomas Wild

Contents

Transaction Level Modeling according to TLM 2.0

- Basics & Background
- 2 Phases Approximately Timed Interaction
- Example

Transaction Level Modeling (TLM)

Motivation

- Increase simulation speed
- Reduce modeling effort
- Allow for easier model adaptability
- ➢ Enable efficient architecture exploration
- ➢ Enable parallel HW and SW development based in virtual prototypes

Measure

- Interaction between SystemC Modules is abstracted
- Function calls instead of signals
  - Based on sc_interface

Standardization

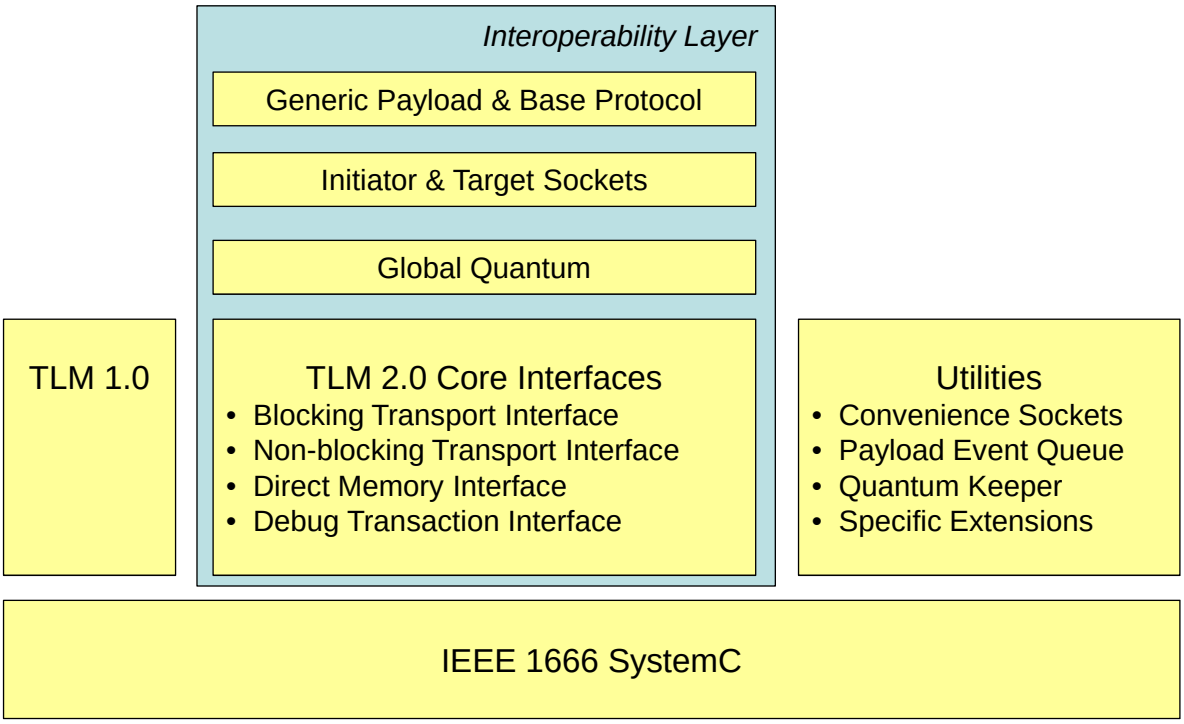
Required to guarantee

- Interoperability of IP (Intellectual Property) modules
- Tools (for architecture exploration, virtual prototyping, …)

TLM 2.0

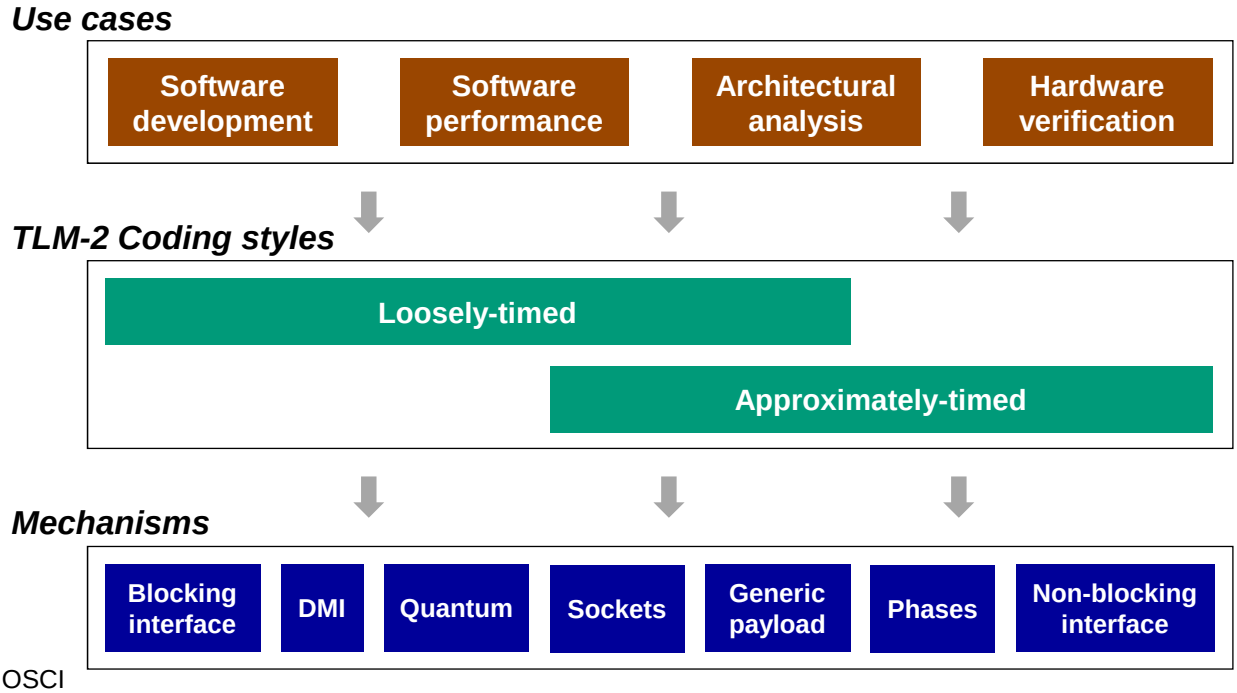
- Standardization by TLM Working group of OSCI (Open SystemC Initiative, www.systemc.org)
- First Version in 2008
- Language Reference Manual, open-source library implementation, modeling examples available from OSCI
- IEEE Standardization under way

TLM 2.0 Classes



Source: OSCI

Use Cases, Coding Styles and Mechanisms

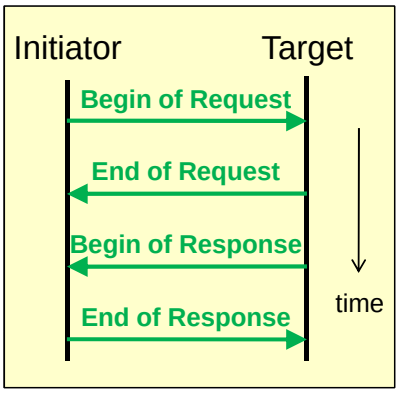


Source: OSCI

TLM 2.0 Modeling styles

General characterization of a transaction between initiator and target in max. 4 phases

- Begin / end of request
- Begin / end of response



Loosely timed

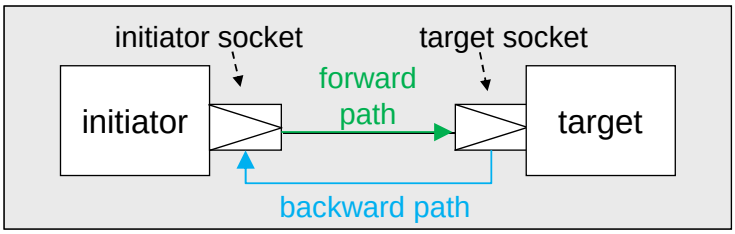
- Focus on simulation performance
- Transaction carried out in one or two phases
  - Blocking call from initiator to target
- Initiator and Target of transaction may be temporally decoupled
  - Synchronization after a certain time quantum

Approximately timed

- Focus more on timing accuracy
- 2 or 4 phases with non-blocking function calls
- Forward path: Initiator calls target for begin of request (and end of response)
- Backward path: Target calls initiator for (end of request and) begin of response

Sockets

Connect initiator with target



simple_initiator_socket and *simple_target socket*

- Easy to be used socket
- Part of the TLM 2.0 utilities
- Provide standardized functions to be called …
  - in forward direction (from initiator to target)
    - *b_transport*
    - *nb_transport_fw*
  - in backward direction (from target to initiator)
    - *nb_transport_bw*

In this lab we only consider
- non-blocking interaction
- approximately timed
- with 2 phases

Simple sockets use dynamic processes. Therefore,

- `#define SC_INCLUDE_DYNAMIC_PROCESSES` before `#include "systemc.h"`

Functions for approximately timed interaction (cont.)

nb_transport_fw(...)

- **Called from** a process or member function within **initiator** via the *simple_initiator_socket*
- **Declared** and **implemented** in **target**
- **Registered** in the constructor of the target with the simple_target_socket

nb_transport_bw(...)

- **Called from** a process or member function within **target** via the *simple_target_socket*
- **Declared** and **implemented** in **initiator**
- **Registered** in the constructor of the initiator with the simple_initiator_socket

Functions for approximately timed interaction

```
tlm_sync_enum  nb_transport_fw( TRANS& , PHASE& , sc_time& );
tlm_sync_enum  nb_transport_bw( TRANS& , PHASE& , sc_time& );
```

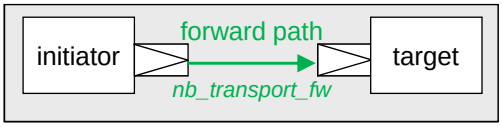
Transaction object, *tlm_generic_payload* instance

tlm_phase:
- BEGIN_REQ
- END_REQ
- BEGIN_RESP
- END_RESP

- TLM_ACCEPTED
  Function arguments unmodified (ignored) on return
  (relevant for 4-phase interaction)
- TLM_UPDATED
  Function arguments modified on return
  Target has moved to the next state of the base protocol
- TLM_COMPLETED
  Function arguments modified on return
  Target has moved straight to the final phase of the base protocol

Transitions for each base protocol transaction

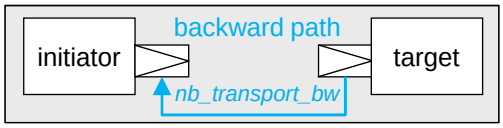
Phase 1: Request



Initiator calls *nb_transport_fw*
with *phase* BEGIN_REQ

Target returns TLM_UPDATED
with *phase* END_REQ

Phase 2: Response



Target calls *nb_transport_bw*
with *phase* BEGIN_RESP

Initiator returns TLM_COMPLETED
with *phase* END_RESP

(4 phase case uses 4 function calls for begin and end of reqest/response.)

TLM Generic Payload

- Class for transaction objects to be passed via sockets
- Methods:

```
tlm_command get_command() const;
void set_command( const tlm_command );

sc_dt::uint64 get_address() const;
void set_address( const sc_dt::uint64 );

unsigned char* get_data_ptr() const;
void set_data_ptr( unsigned char* );

unsigned int get_data_length() const;
void set_data_length( const unsigned int );

tlm_response_status get_response_status() const;
void set_response_status( const tlm_response_status );
```

TLM_READ_COMMAND
TLM_WRITE_COMMAND
…

- tlm_command
- address
- *data
- data_length
- tlm_response_status
- *extension
- …

TLM_OK_RESPONSE
TLM_INCOMPLETE_RESPONSE
TLM_GENERIC_ERROR_RESPONSE
…

Payload Event Queue

- Provided in TLM Utilities
  - `peq_with_get<tlm_generic_payload> peq_1;`
- Used for maintaining a queue of notifications of events with associated transaction object (payload instance)
  - At time t: write payload instance into a to peq with annotated delay D (method ***notify(transaction, D)***)
  - At time t+D: peq will trigger an event in process that waits for event (method ***get_event()***):
    ➔ get payload (method ***get_next_transaction()***)
    ➔ process transaction
  - Decouple execution of *nb_transport_fw* or *nb_transport_bw* from processing of transaction in the target and initiator, respectively.

Payload Event Queue - Example

module_1.h

```
#include "systemc.h"
#include "tlm.h"
...
#include "tlm_utils/peq_with_get.h"

SC_MODULE (module_1) {
  ...
private:
  peq_with_get<tlm_generic_payload> peq;

private:
  // member function or process
  void write_into_peq();
  // process
  void process_peq_transactions();

public:
  // constructor
  SC_CTOR(module_1): peq("my_peq")
  {
    SC_THREAD(process_peq_transactions);
  }
};
```

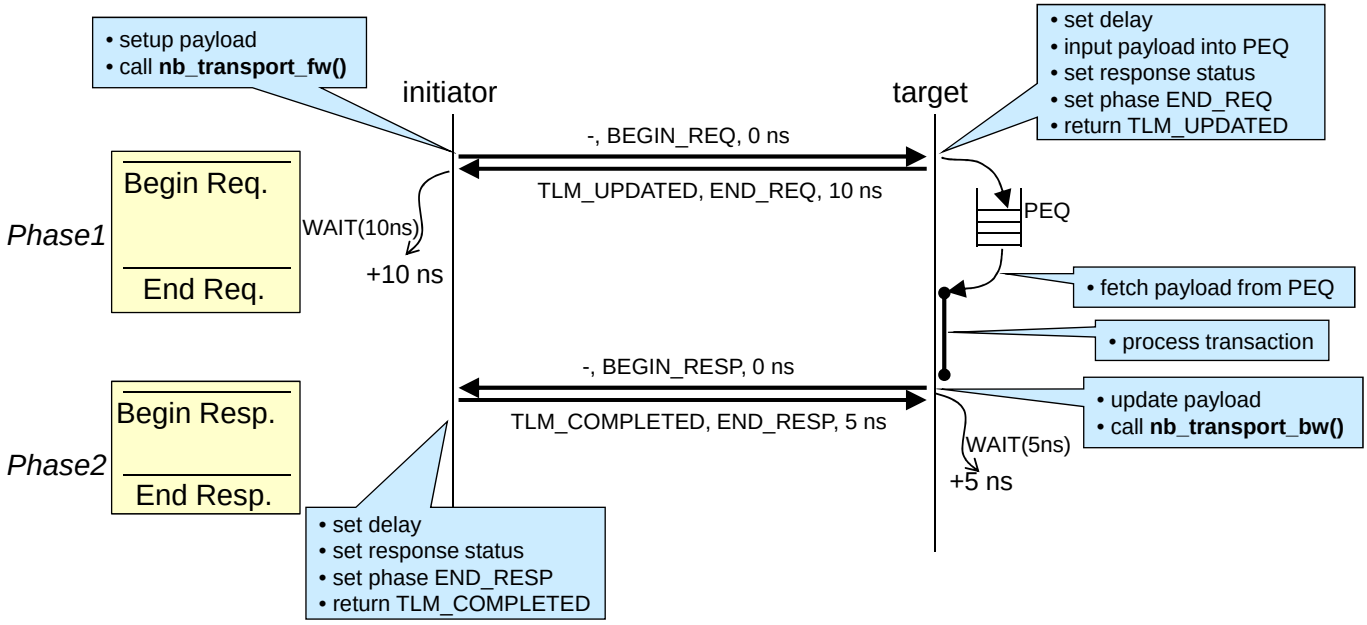
module_1.cpp

```
#include "module_1.h"

void module_1::write_into_peq(){
  tlm_generic_payload trans;
  sc_time delay;
  ...
  // modify trans and delay
  ...
  peq.notify(trans, delay);
  ...
}
void module_1:: process_peq_transactions(){
  tlm_generic_payload payload;
  ...
  wait( peq.get_event() );

  payload = peq.get_next_transaction();

  ... // further processing of payload

}
```

peq.get_event() may also be used in static sensitivity declaration

TLM 2.0 Example

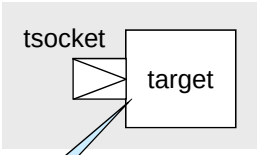
Approximately timed modeling via non-blocking interface (2 phases)



TLM Example – Target

```
#include "systemc.h"
#include "tlm.h"
#include "tlm_utils/simple_target_socket.h"

using namespace sc_core;
using namespace tlm_utils;

SC_MODULE(target){
    simple_target_socket<target> tsocket;

    tlm_sync_enum nb_transport_fw(tlm_generic_payload &payload,
                  tlm_phase &phase, sc_time &delay_time);
    peq_with_get<tlm_generic_payload> target_peq;
    void target_proc();

    SC_CTOR(target):
        tsocket("tsocket"), target_peg("t_peq")
    {

      SC_THREAD(target_proc);
      tsocket.register_nb_transport_fw(this,
                  &target::nb_transport_fw);
      ...
    }
};
```

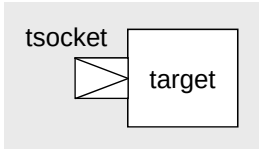
target.h



Target socket

- *nb_transport_fw* declaration
- PEQ for decoupling nb_transport_fw from target_proc()
- call constructor of target socket and PEQ
- register *nb_transport _fw* with target socket

TLM Example – Target (cont.)



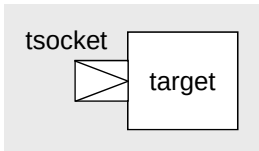
target.cpp

```
#include "target.h"
...
tlm_sync_enum target::nb_transport_fw (tlm_generic_payload
          &payload, tlm_phase &phase, sc_time &delay_time)
{
   tlm_command cmd = payload.get_command();
   unsigned char* ptr = payload.get_data_ptr();
   unsigned int len = payload.get_data_length();

   if(cmd == TLM_WRITE_COMMAND) {
      ...
      payload.set_response_status(TLM_OK_RESPONSE);
      ...
   } else if(cmd == TLM_READ_COMMAND){
      ...
      payload.set_response_status(TLM_GENERIC_ERROR_RESPONSE);
      ...
   }
   delay_time = sc_time(..., SC_NS);

   target_peq.notify(payload, delay_time);

   phase  = END_REQ;
   return TLM_UPDATED;
}
```

- *nb_transport_fw* implementation
- evaluate transaction
- set response status
- set delay according to duration of interaction
- enter transaction into PEQ
- conclude phase 1, i.e. request

Technische Universität München

TLM Example – Target (cont.)



target.cpp

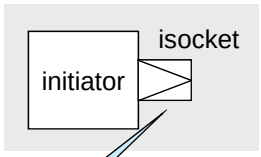
```
...
void target::target_proc()
{
   tlm_generic_payload *trans;
   sc_time delay;
   tlm_phase ph;
   tlm_response resp;

   while(true){
     wait(target_peq.get_event());

     trans = target_peq.get_next_transaction();
     if(cmd == TLM_WRITE_COMMAND) {
       ...
       trans->set_response_status(...);
     }

     delay = SC_ZERO_TIME;
     ph  = BEGIN_RESP
     resp = tsocket->nb_transport_bw(*trans, ph, delay);
     if(resp != TLM_COMPLETED){
        ... }
     wait(delay);
   }
}
```

- wait until PEQ notifies next transaction
- fetch transaction from PEQ
- process
- set response status
- Start phase 2, i.e. response
- verify correct end of transaction: is phase 2 correctly finished
- wait for time annotated by initiator

TLM Example – Initiator

initiator.h

```
#include "systemc.h"
#include "tlm.h"
#include "tlm_utils/simple_initiator_socket.h"

using namespace sc_core;
using namespace tlm_utils;

SC_MODULE(initiator){
  simple_initiator_socket<initiator> isocket;

  tlm_sync_enum nb_transport_bw(tlm_generic_payload &payload,
                  tlm_phase &phase, sc_time &delay_time);
  void initiator_proc();
  sc_event transaction_done;

  SC_TOR(initiator):
      isocket("isocket")
  {
    SC_THREAD(initiator_proc);
    tsocket.register_nb_transport_bw(this,
                &target::nb_transport_bw);
  }
};
```

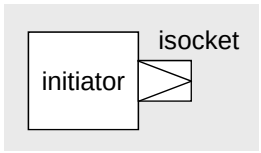


Initiator socket

- *nb_transport_bw* declaration

- call constructor of isocket

TLM Example – Initiator (cont.)



initiator.cpp

```
#include "initiator.h"

void initiator::initiator_proc(){

  tlm_generic_payload trans;
  sc_time delay;
  tlm_phase phase;
  tlm_sync_enum result;
 // further declarations for data, data_length, address,

  while(true){
     ... // setup transaction payload
     ... // prepare data, data_length, address
     delay = SC_ZERO_TIME;   // sc_time(0, SC_NS);
     phase = BEGIN_REQ;
     result = isocket->nb_transport_fw(trans, phase, delay);

     if(result != TLM_UPDATED){
        ... };

     ... // go on with processing
     wait(transaction_done);
     ... // do further processing
  }
}
```

```
trans.set_command(TLM_READ_COMMAND);
trans.set_data_length(data_length);
trans.set_data_ptr(data);
trans.set_address(address)
```

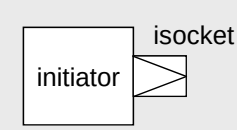
Start phase 1 of transaction

- verify correct end of phase 1 of transaction

Wait till phase 2 of transaction is finished (event is notified by *nb_transport_bw*)

Payload instance *trans* must not be modified or reused within initiator before transaction is completely finished !!!

TLM Example – Initiator (cont.)



initiator.cpp

```
...
tlm_sync_enum initiator::nb_transport_bw(tlm_generic_payload
          &payload, tlm_phase &phase, sc_time &delay_time);
{

   if(phase != BEGIN_RESP)
      cout<<sc_time_stamp()<<" "<<name()<<" error in phase transition"
          <<endl;


   delay = sc_time(20, SC_NS);

   transaction_done.notify(delay);


   ph  = END_RESP;
   return TLM_COMPLETED;
}
```

- Check correct phase
- Calculate delay for response
- Notify completion of transaction. *initiator_proc()* is waiting for this event.
- Finish phase 2 of transaction.

TLM Example – Top Level of the Model

main.cpp

```
#include "systemc.h"
#include "initiator.h"
#include "target.h"

int sc_main() {
  ..
  initiator mod1("initiator_instance");
  target mod2("target_instance");
  ...

  mod1.isocket.bind(mod2.tsocket);

  ...

  sc_start(5000, SC_NS);

  return(0);
}
```

- header files of all modules
- instantiate modules
- interconnect sockets
- start simulation