SystemC and Virtual Prototyping

Dr. Matthias Jung, Fraunhofer Institute IESE
matthias.jung@iese.fraunhofer.de

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

							Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
Transaction Level Modeling (TLM)	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoup-ling & DMI	Phases	Payload Exten-sions	SystemC AMS			
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals								
	Simulation Kernel	Methods & Threads		Channels & Interfaces		Data Types: Logic, Integers, Fixedpoint & Floatingpoint			
		Events, Sensitivity & Notifications		Modules & Hierarchy					
C++									

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Transaction Level Modeling (TLM)	User Libraries									
	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions	SystemC AMS	Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals									
	Simulation Kernel	Methods & Threads		Channels & Interfaces		Data Types: Logic, Integers, Fixedpoint & Floatingpoint			
		Events, Sensitivity & Notifications		Modules & Hierarchy					
C++										

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Recap: TLM Coding Styles and Mechanisms

TLM Use Cases

SW Application Development	SW Performance Analysis	Architecture Analysis	Hardware Verification

TLM 2.0 Coding Style *(Just Guidelines)*

Loosely-Timed (LT)

Single-phase, blocking API

Multi-phase, non-blocking API

Approximately-Timed (AT)

TLM Mechanisms *(Definitive API for enabling Interoperability)*

Blocking transport	DMI	Quantum (Keeper)	Sockets	Generic payload	Extensions	Phases	Non-blocking transport

4

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Recap: TLM Coding Styles and Mechanisms

TLM Use Cases

SW Application

Architecture

SW
Development

HW
Design Space
Exploration

sockets

pay...

transport

Source: Doulos Ldt. www.doulos.com

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Recap: Coding Styles in TLM

Loosely-Timed (LT): SW

- As fast as possible

- Sufficient timing detail to boot OS and run multicore systems and to <u>develop SW</u> or drivers

- Processes can run ahead of simulation time (temporal decupling)

- Each transaction completes in one <u>blocking</u> function call

- Usage of Direct Memory Interface (DMI) e.g. for boot process

Approximately-Timed (AT): HW

- Accurate enough for performance modelling

- Sufficient for architectural HW <u>design space exploration</u>

- Processes run in lockstep with simulation time

- Each transaction has usually 4 timing points i.e. 4 function calls (extensible if required, also less possible); <u>non-blocking</u> behavior

- More detailed than LT and therefore also slower than LT

6

Fraunhofer
IESE
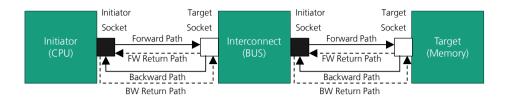
Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Initiators, Targets and Interconnect

- References to the object are passed along the forward and backward paths:
  - LT uses Forward and Return Path
  - AT uses Forward, Backward, FW Return Path, BW Return Path
- AT uses non-blocking transport
- Time is handeled with Payload Event Queues (PEQs)
- Allows modelling of O-O-O Cores and Backpressure
- Base Protocol

7

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

User Libraries									
Transaction Level Modeling (TLM)	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions	**SystemC AMS**	Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals								
	Simulation Kernel	Methods & Threads		Channels & Interfaces		Data Types: Logic, Integers, Fixedpoint & Floatingpoint			
		Events, Sensitivity & Notifications		Modules & Hierarchy					
C++									

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Non-Blocking Transport (AT) Base Protocol

```
status = nb_transport_fw(trans, phase, delay)
```

Initiator   Target

BEGIN_REQ, 0ns

TLM_ACCEPTED, BEGIN_REQ, 10ns

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules [1]: Using BW Path

Initiator   Target

```
BEGIN_REQ, 0ns
```
TLM_ACCEPTED, BEGIN_REQ, 0ns

Request accept delay

```
END_REQ, 0ns
```
TLM_ACCEPTED, END_REQ, 0ns

Latency of Target

```
BEGIN_RESP, 0ns
```
TLM_ACCEPTED, BEGIN_RESP, 0ns

Response accept delay

```
END_RESP, 0ns
```
TLM_COMPLETE, END_RESP, 0ns

P H A S E S

10

© Fraunhofer IESE

Fraunhofer
IESE

An initiator, and only an initiator, may call `b_transport`. The target can return immediately, or can suspend (by calling `wait`) and return later. However, calling `wait` statements in targets should be avoided because a wait statement will result in a context switch of the simulator, which decreases the simulation speed.

The same transaction object could be reused from one call to the next, but the two calls would count as separate transactions.

Your notes:

_____

_____

_____

_____

_____

_____

Alternative View

BEGIN_REQ   END_REQ

BEGIN_RESP   END_RESP

REQ

RESP

Request accept delay

Response accept delay

Latency of Target

11

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules: Using BW Path

HW

- Base Protocol phases
    - BEGIN_REQ → END_REQ → BEGIN_RESP → END_RESP
    - Must occur in increasing simulation time order
    - Phases must change with each call
- nb_transport_fw must not call nb_transport_bw directly and vice versa (PEQ!)
- Generic Payload memory management rules (See TLM Advanced)
- Extensions must be ignorable (See TLM Advanced)
- Target should handle mixed b_transport / nb_transport

12

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules: The Exclusion Rule

- Initiators and targets must honor the *Exclusion Rule*:
  - An Initiator must not send a new request (`BEGIN_REQ`) until it has received the `END_REQ` from the previous transaction
  - An target must not send the next `BEGIN_RESP` until it has received the `END_RESP` from the previous transactions
- The exclusion rules enable flow control like back pressure:
  - E.g. if an input buffer of a target is full the target can defer the sending of `END_REQ` for the transaction that filled the buffer, until the buffer has available space again
  - An RTL ready signal can be modeled by deferring `END_REQ`
  - However, since it is non-blocking, the target can do something else, e.g. sending

13
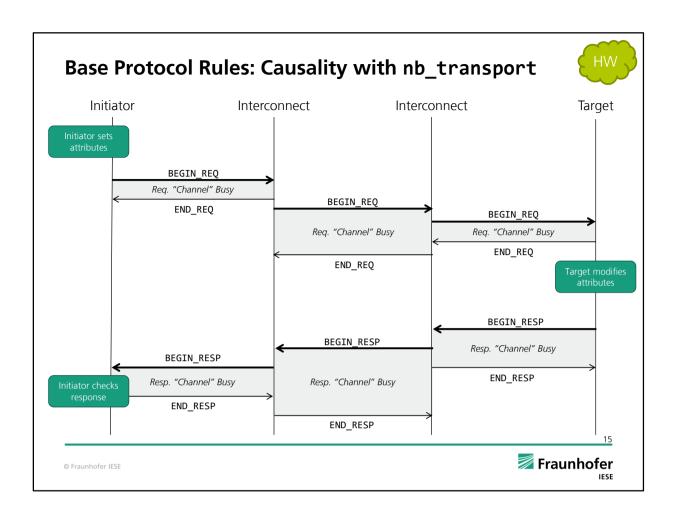
© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Modelling of Backpressure

BEGIN_REQ
END_REQ
BEGIN_RESP
END_RESP

- The exclusion rules enable flow control like back pressure:

  - E.g. if an input buffer of a target is full the target can defer the sending of END_REQ for the transaction that filled the buffer, until the buffer has available space again

  - An RTL ready signal can be modeled by deferring END_REQ

  - However, since it is non-blocking, the target can do something else, e.g. sending   Implication of Last Point the Request Accept Delay will be longer

- Also Response exclusion rule must be honored!

14

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules: Causality with `nb_transport`

Initiator	Interconnect	Interconnect	Target

Initiator sets attributes

BEGIN_REQ

Req. "Channel" Busy

END_REQ

BEGIN_REQ

Req. "Channel" Busy

END_REQ

BEGIN_REQ

Req. "Channel" Busy

END_REQ

Target modifies attributes

BEGIN_RESP

BEGIN_RESP

Resp. "Channel" Busy

BEGIN_RESP

Resp. "Channel" Busy

END_RESP

Initiator checks response

Resp. "Channel" Busy

END_RESP

END_RESP

15

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules: Causality with `nb_transport`

- `BEGIN_REQ` and `BEGIN_RESP` are <u>propagated</u> from end-to-end
- `END_REQ` and `END_RESP` are not propagated → they are <u>local</u> to each hop
- The `END_REQ` and `END_RESP` phases indicate that the requests and response "channels" are no longer busy and may be used for the next request or response respectively
- Initiator should not check the transaction payload until it has received the response   Implication of Last Point it is possible that the Target still needs to modify the data

16

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules: Pipelining of Transactions

HW

Initiator   Target

BEGIN_REQ

- Exclusion Rule

END_REQ

- Order within transaction must be always the same

BEGIN_REQ

END_REQ

- Transactions can be out-of-order

BEGIN_RESP

END_RESP

BEGIN_RESP

END_RESP

© Fraunhofer IESE

Fraunhofer
IESE
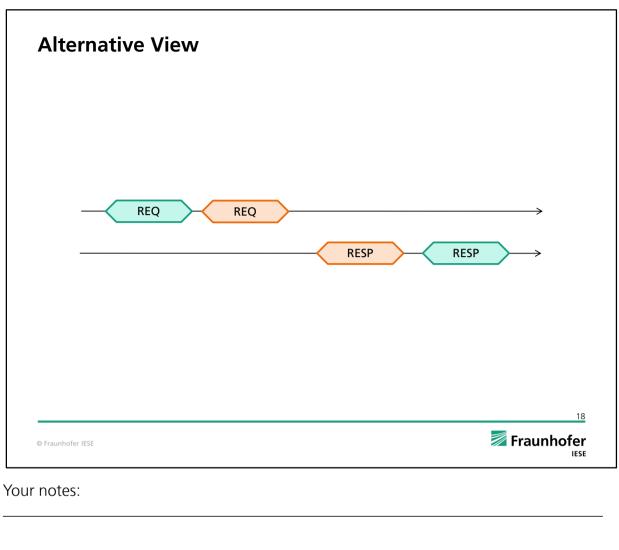
For **nb_transport_(fw|bw),** the timing annotation on successive calls for a given transaction must be strictly non-decreasing. For separate transactions, the mutual order is unconstrained, i.e. out-of-order transaction ordering is allowed.

Your notes:
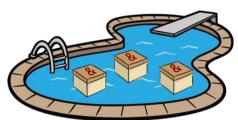
_____

_____

_____

_____

_____

_____

_____

_____

Alternative View

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

							SystemC AMS		
	User Libraries								
Transaction Level Modeling (TLM)	Sockets & <u>Generic Payload</u>	Blocking & Non-Blocking	Temporal Decoup-ling & DMI	Phases	Payload Exten-sions		Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals								
	Simulation Kernel	Methods & Threads		Channels & Interfaces			Data Types: Logic, Integers, Fixedpoint & Floatingpoint		
		Events, Sensitivity & Notifications		Modules & Hierarchy					
	C++								

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Generic Payload Pools (a.k.a Memory Management)

- Allocating of generic payload objects consumes wall-clock time
- Idea: do not allocate for each transaction a new generic payload → **Reuse**
    - A *Memory Manager* handles the generic payload objects in a pool
    - Before sending a transaction a payload object is `allocate`d from the pool
    - Modules that send (or receive) this payload object must increase a reference count (`acquire`), which signalizes the memory manager that this object is still in use.
    - If a module is finished with the payload object, the reference count is decreased (`release`)
    - If the reference count is 0 the payload is freed and goes back into the pool.
    - If all transactions in the pool are in use a new one is generated

20

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Memory Manager in TLM

- The memory manager is not part of the SystemC TLM Standard
- The TLM Standard provides only an interface class:

```
class tlm_mm_interface
{
    public:
    virtual void free(tlm::tlm_generic_payload*) = 0;
    virtual ~tlm_mm_interface(){}
}
```
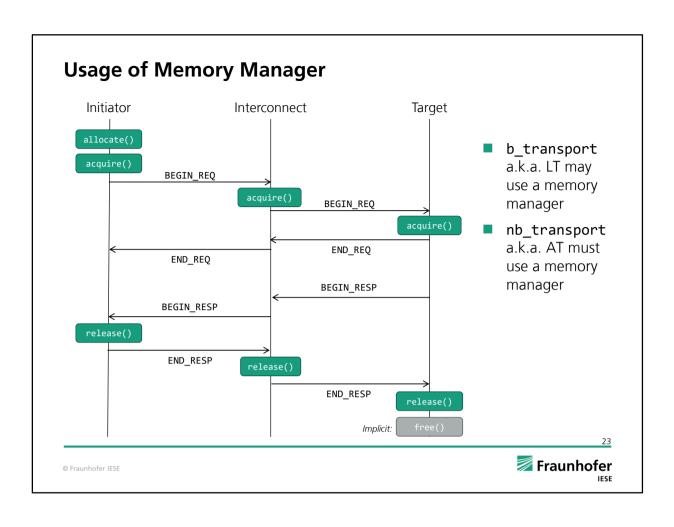
- The implementation of the memory manager is up to the end user
- Virtual Platform tools like Synopsys Platform Architect have clever implementations

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Example Memory Manager

```cpp
class MemoryManager : public tlm::tlm_mm_interface {
    private:
    unsigned int numberOfAllocations;
    unsigned int numberOfFrees;
    std::vector<tlm::tlm_generic_payload*> freePayloads;

    public:
    MemoryManager(): numberOfAllocations(0), numberOfFrees(0) {}

    ~MemoryManager(){
        for(tlm::tlm_generic_payload* payload: freePayloads) {
            delete payload;
            numberOfFrees++;
        }
    }
    tlm::tlm_generic_payload* MemoryManager::allocate(){
        if(freePayloads.empty()) {
            numberOfAllocations++;
            return new tlm::tlm_generic_payload(this);
        } else {
            tlm::tlm_generic_payload* result = freePayloads.back();
            freePayloads.pop_back();
            return result;
        }
    }

    void free(tlm::tlm_generic_payload* payload) {
        payload->reset(); //clears all fields and extensions
        freePayloads.push_back(payload);
    }
};
```

Transaction Pool implemented as `std::vector`

Cleanup transaction pool

Allocate new payload

Reuse Payload and remove from pool

Add transaction back to the pool

22

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Usage of Memory Manager

Initiator   Interconnect   Target

`allocate()`

`acquire()`

BEGIN_REQ →

`acquire()`

BEGIN_REQ →

`acquire()`

← END_REQ

← END_REQ

← BEGIN_RESP

← BEGIN_RESP

`release()`

END_RESP →

`release()`

END_RESP →

`release()`

Implicit: `free()`

- **b_transport** a.k.a. LT may use a memory manager

- **nb_transport** a.k.a. AT must use a memory manager

Fraunhofer IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Transaction Level Modeling (TLM)	User Libraries					SystemC AMS			
	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions		Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals								
	Simulation Kernel	Methods & Threads		Channels & Interfaces		Data Types: Logic, Integers, Fixedpoint & Floatingpoint			
		Events, Sensitivity & Notifications		Modules & Hierarchy					
	C++								

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

The Payload Event Queue

```
C  →  B  →  A  →  ( PEQ )  →  A  →  C  →  B
```

@110ns   @100ns   @100ns   @140ns   @130ns   @120ns
delay=20ns   delay=20ns   delay=40ns

- AT models usually synchronize incoming calls with the simulation time
- Timing annotation is similar to b_transport:
    - *Hey target, please pretend that you received this message 10ns in the future*
- An AT component is usually not calling `wait()` statements for synchronizing with time, it rather posts the incoming transaction into a *Payload Event Queue* (PEQ). The PEQ internally synchronizes with the simulation time and calls a callback function in order to process the transaction when the time is reached.
- A component receives a transaction that should be processed in the future, so it puts the transaction into a PEQ in order to process it when time is reached

25

Fraunhofer
IESE
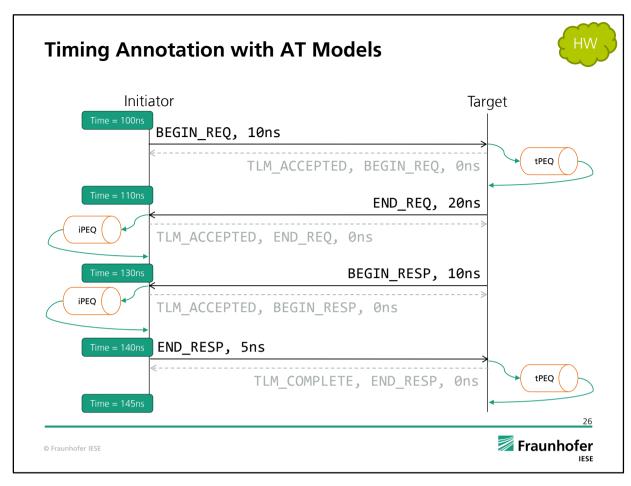
Your notes:

at the BEGIN_REQ phase, we put all incoming events into the PEQ, and always return immediately - the PEQ will ensure we get a wake up of the Initiator after a certain time as annotated during the call - we simulate transport time using this.

The queue also sorts events according to time - in the above example, events arrived from Initiator in the order A B C, but they will be returned from target in the temporal order B C A because of the annotated time sent by the initiator - callbacks are used for this - they are used to acknowledge the timing annotations that are sent along with the request.

AT models use the PEQs to synchronize incoming calls with the simulation time (instead of calling wait() statements) - unlike LT where we add up delays on a local time variable and do temporal decoupling

_____

_____

_____

_____

_____

Timing Annotation with AT Models

Initiator   Target

Time = 100ns	

BEGIN_REQ, 10ns →

tPEQ

← TLM_ACCEPTED, BEGIN_REQ, 0ns

Time = 110ns	

END_REQ, 20ns

iPEQ

TLM_ACCEPTED, END_REQ, 0ns →

Time = 130ns	

BEGIN_RESP, 10ns

iPEQ

TLM_ACCEPTED, BEGIN_RESP, 0ns →

Time = 140ns	

END_RESP, 5ns →

tPEQ

← TLM_COMPLETE, END_RESP, 0ns

Time = 145ns	

26

Fraunhofer
IESE
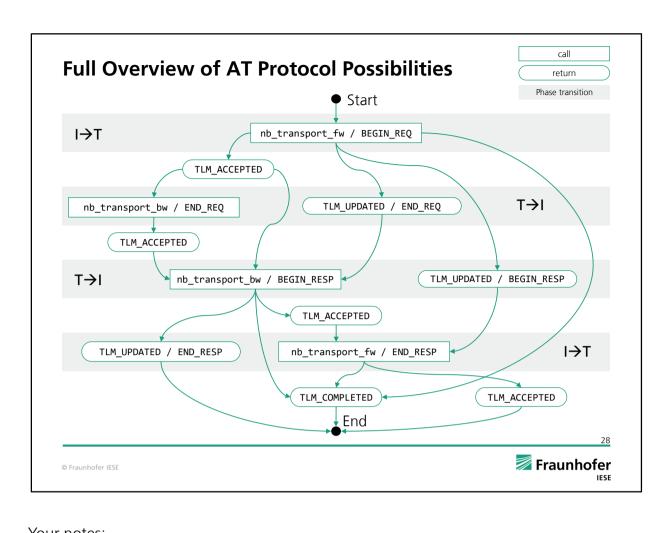
In AT models it is usual to synchronize incoming `nb_transport` calls with the simulation time. In the message sequence chart above, an initiator sends `BEGIN_REQ` with a timing annotation of 10 ns. The target returns `TLM_ACCEPTED` and is obliged to behave as if it had received the transaction at `sc_time_stamp() + 10ns`. An AT component is usually not calling wait statements for synchronizing with time, it rather posts the incoming transaction into a *Payload Event Queue* (PEQ). The payload event queue internally synchronizes with the simulation time (`wait`) and calls a callback function in order to process the transaction when the time is reached. In other words, a component receives a transaction that should be processed in the future, so it puts the transaction into a queue such that it is actually processed in the future.

Your notes:

_____

_____

_____

_____

_____

_____

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Full Overview of AT Protocol Possibilities

call
return
Phase transition

● Start

I→T

nb_transport_fw / BEGIN_REQ

TLM_ACCEPTED

nb_transport_bw / END_REQ

TLM_UPDATED / END_REQ

T→I

TLM_ACCEPTED

T→I

nb_transport_bw / BEGIN_RESP

TLM_UPDATED / BEGIN_RESP

TLM_ACCEPTED

TLM_UPDATED / END_RESP

nb_transport_fw / END_RESP

I→T

TLM_COMPLETED

TLM_ACCEPTED

End

© Fraunhofer IESE

≡ Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Full Overview of AT Protocol Possibilities

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules [1]: 4 Phase Handshake

Initiator   Target

```
BEGIN_REQ, 0ns  ───────────────────────────────────>

<─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─
              TLM_ACCEPTED, BEGIN_REQ, 0ns

                              END_REQ, 0ns
<───────────────────────────────────────────────────

─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─>
TLM_ACCEPTED, END_REQ, 0ns

                          BEGIN_RESP, 0ns
<───────────────────────────────────────────────────

─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─>
TLM_ACCEPTED, BEGIN_RESP, 0ns

END_RESP, 0ns ──────────────────────────────────────>

<─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─
              TLM_COMPLETE, END_RESP, 0ns
```

30

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

4 Phase Handshake: Initiator

```cpp
class Initiator: public sc_module, public tlm::tlm_bw_transport_if<> {
    public:
    tlm::tlm_initiator_socket<> socket;
    MemoryManager mm;
    int data[16];
    tlm::tlm_generic_payload* requestInProgress;
    sc_event endRequest;
    tlm_utils::peq_with_cb_and_phase<Initiator> peq;

    SC_CTOR(Initiator): socket("socket"),
                        requestInProgress(0),
                        peq(this, &Initiator::peqCallback)
    {
        socket.bind(*this);
        SC_THREAD(process);
    }

    protected:
    void process() {
        tlm::tlm_generic_payload* trans;
        tlm::tlm_phase phase;
        sc_time delay;

        for (int i = 0; i < 100; i++) {
            trans = mm.allocate();
            trans->acquire();

            trans->set_command(...);
            ...
            trans->set_response_status(tlm::TLM_INCOMPLETE_RESPONSE);

            if (requestInProgress) {
                wait(endRequest);
            }
            requestInProgress = trans;
            phase = tlm::BEGIN_REQ;
            delay = ...;

            tlm::tlm_sync_enum status;
            status = socket->nb_transport_fw( *trans, phase, delay );
            ...
            wait(randomDelay());
        }
    }
}
```

Generate a sequence of random transactions

Grab a new transaction from the memory manager pool

BEGIN_REQ/END_REQ exclusion rule

```cpp
    virtual tlm::tlm_sync_enum nb_transport_bw(tlm::tlm_generic_payload& trans,
                                               tlm::tlm_phase& phase,
                                               sc_time& delay)
    {
        peq.notify(trans, phase, delay);
        return tlm::TLM_ACCEPTED;
    }


void peqCallback(tlm::tlm_generic_payload& trans,
                 const tlm::tlm_phase& phase)
    {
        if (phase == tlm::END_REQ || ...)
        {
            requestInProgress = 0;
            endRequest.notify();
        }
        else  if (phase == tlm::BEGIN_RESP)
        {
            checkTransaction(trans);

            tlm::tlm_phase fw_phase = tlm::END_RESP;
            sc_time delay = sc_time(...);

            socket->nb_transport_fw( trans, fw_phase, delay );

            trans.release();
        }
        else if (phase == tlm::BEGIN_REQ || phase == tlm::END_RESP)
        {
            SC_REPORT_FATAL(name(), "Illegal transaction phase received");
        }

    }
};
```

Queue the transaction into the peq until the annotated time has elapsed

Payload event queue callback

also used to honor exclusion rule

Wake-up suspended main process

Do something with transaction

Allow MM to free the transaction object

31

Fraunhofer
IESE

4 Phase Handshake: Target

HW

```cpp
class Target: public sc_module, public tlm::tlm_fw_transport_if<> {
    public:
    tlm::tlm_target_socket<> socket;
    tlm::tlm_generic_payload* transactionInProgress;
    sc_event targetDone;
    bool responseInProgress;
    tlm::tlm_generic_payload* nextResponsePending;
    tlm::tlm_generic_payload* endRequestPending;
    tlm_utils::peq_with_cb_and_phase<Target> peq;

    SC_CTOR(Target) : socket("socket"),
        transactionInProgress(0),
        responseInProgress(false),
        nextResponsePending(0),
        endRequestPending(0),
        peq(this, &Target::peqCallback)
    {
        socket.bind(*this);

        SC_METHOD(executeTransactionProcess);
        sensitive << targetDone; dont_initialize();
    }
    ...
    tlm::tlm_sync_enum nb_transport_fw(tlm::tlm_generic_payload& trans,
                                       tlm::tlm_phase& phase,
                                       sc_time& delay)
    {
        peq.notify( trans, phase, delay);
        return tlm::TLM_ACCEPTED;
    }

    void peqCallback(tlm::tlm_generic_payload& trans,
                     const tlm::tlm_phase& phase)
    {
        sc_time delay;

        if(phase == tlm::BEGIN_REQ) {
        trans.acquire();

            if (!transactionInProgress) {
                sendEndRequest(trans);
            }
            else {
                endRequestPending = &trans;
            }
        }
```

> Increment the transaction reference count

> Put back-pressure on initiator by deferring END_REQ

```cpp
        else if (phase == tlm::END_RESP) {
            ...
            transactionInProgress = 0;

            responseInProgress = false;

            if (nextResponsePending) {
                sendResponse(*nextResponsePending);
                nextResponsePending = 0;
            }

            if (endRequestPending) {
                sendEndRequest(*endRequestPending);
                endRequestPending = 0;
            }
        }
        else // tlm::END_REQ or tlm::BEGIN_RESP
        {
            SC_REPORT_FATAL(name(), "Illegal transaction phase received");
        }
    }

    void sendEndRequest(tlm::tlm_generic_payload& trans)
    {
        tlm::tlm_phase bw_phase;
        sc_time delay;

        bw_phase = tlm::END_REQ;
        delay = ...; // Accept delay

        tlm::tlm_sync_enum status;
        status = socket->nb_transport_bw( trans, bw_phase, delay );

        delay = delay + randomDelay();
        targetDone.notify( delay );

        assert(transactionInProgress == 0);
        transactionInProgress = &trans;
    }
    ...
```

> Flag must only be cleared when END_RESP is sent

> Target itself is now clear to issue the next BEGIN_RESP

> unblock the initiator by issuing END_REQ

> Queue internal event to mark beginning of response

32

Fraunhofer
IESE

4 Phase Handshake: Target

```
void executeTransactionProcess()
{
    executeTransaction(*transactionInProgress);

    if (responseInProgress)
    {
        ...
        nextResponsePending = transactionInProgress;
    }
    else
    {
        sendResponse(*transactionInProgress);
    }
}


void sendResponse(tlm::tlm_generic_payload& trans)
{
    tlm::tlm_sync_enum status;
    tlm::tlm_phase bw_phase;
    sc_time delay;

    responseInProgress = true;
    bw_phase = tlm::BEGIN_RESP;
    delay = SC_ZERO_TIME;
    status = socket->nb_transport_bw( trans, bw_phase, delay );

    ...

    trans.release();
}
...
};
```

Method process that runs on targetDone event

Target must honor BEGIN_RESP/END_RESP exclusion rule

Function for sending the response

Tell the memory manager to free transaction

33

© Fraunhofer IESE

Fraunhofer
IESE

Base Protocol Rules [2]: Using the Return Paths

Initiator   Target

Time = 100ns

BEGIN_REQ, 0ns

TLM_UPDATED, END_REQ, 10ns

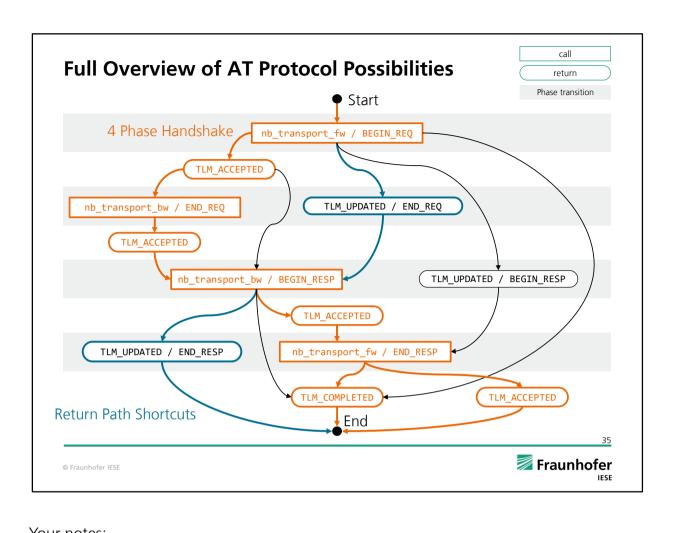Time = 110ns

Time = 150ns

BEGIN_RESP, 0ns

TLM_UPDATED, END_RESP, 5ns

Time = 155ns

- The FW return path can be used as an alternative to the BW path
- The BW return path can be used as an alternative to the FW path
    - State must be set to `TLM_UPDATED`
- Timing must be increased

34

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Full Overview of AT Protocol Possibilities

call
return
Phase transition

● Start

4 Phase Handshake

nb_transport_fw / BEGIN_REQ

TLM_ACCEPTED

nb_transport_bw / END_REQ

TLM_UPDATED / END_REQ

TLM_ACCEPTED

nb_transport_bw / BEGIN_RESP

TLM_UPDATED / BEGIN_RESP

TLM_ACCEPTED

TLM_UPDATED / END_RESP

nb_transport_fw / END_RESP

Return Path Shortcuts

TLM_COMPLETED

TLM_ACCEPTED

End ●

35

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules [4]: Early Completion

Initiator   Target

Time = 100ns

BEGIN_REQ, 0ns

TLM_COMPLETED, d.c., 10ns

Time = 110ns

- The initiator sends `BEGIN_REQ` and the target immediately returns `TLM_COMPLETED` (useful for modelling simple I/O, where no ACK is required)
- The targets pre-empts any further communication by signaling a so called *Early completion*
- Initiator should immediacy check the response status of the generic payload object
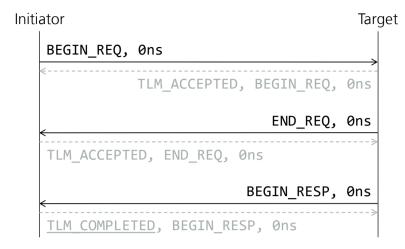- With `TLM_COMPLETED` the caller should always ignore the phase argument

36

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules [4]: Early Completion

Initiator   Target

```
BEGIN_REQ, 0ns
                    TLM_ACCEPTED, BEGIN_REQ, 0ns
                                      END_REQ, 0ns
TLM_ACCEPTED, END_REQ, 0ns
                                   BEGIN_RESP, 0ns
TLM_COMPLETED, BEGIN_RESP, 0ns
```
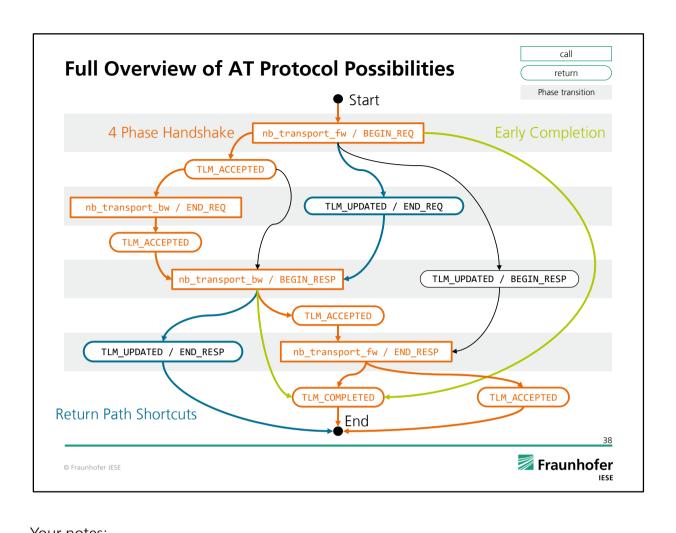
- ■ The initiator can ealy complete the transaction by returning `TLM_COMPLETED`
- ■ The initiator pre-empts therefore any further communication

37

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Full Overview of AT Protocol Possibilities

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules [3]: Skip `END_REQ`

HW

Initiator   Target

```
BEGIN_REQ, 0ns                                      ───────►

          ◄──────  TLM_ACCEPTED, BEGIN_REQ, d.c.              END_REQ
                                 BEGIN_RESP, 0ns              missed!
          ◄──────

TLM_ACCEPTED, BEGIN_RESP, d.c.  ─ ─ ─ ─ ─ ─ ─ ─►

END_RESP, 0ns                                       ───────►

          ◄──────  TLM_COMPLETED, END_RESP, 0ns
```

- `END_REQ` is pre-empted by the target sending `BEGIN_RESP` in the next BW call

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Base Protocol Rules (7): Skip `END_REQ` (Shortcut)

Initiator   Target

BEGIN_REQ, 0ns ───►

◄ -
                TLM_UPDATED, BEGIN_RESP, 0 ns   END_REQ
                                                        missed!

END_RESP, 0ns ──►

◄ -
                TLM_COMPLETED, END_RESP, 0ns
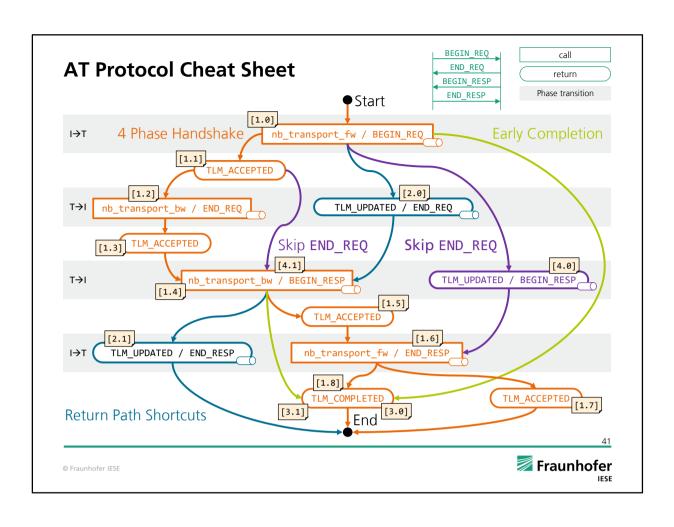
■ **`END_REQ`** is pre-empted by the target sending directly **`BEGIN_RESP`** via
  `TLM_UPDATED` over the return path

40

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

User Libraries									
Transaction Level Modeling (TLM)	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions	**SystemC AMS**	Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals								
	Simulation Kernel	Methods & Threads		Channels & Interfaces			Data Types: Logic, Integers, Fixedpoint & Floatingpoint		
		Events, Sensitivity & Notifications		Modules & Hierarchy					
C++									

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

SystemC and Virtual Prototyping

Dr. Matthias Jung, Fraunhofer Institute IESE
matthias.jung@iese.fraunhofer.de

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

TLM Advanced

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

SYSTEMC™

	User Libraries								

Transaction Level Modeling (TLM)

Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions	SystemC AMS	Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
						Linear DAE solver		Scheduler
						Synchronization layer		

SystemC

	Predefined Primitive Channels: Mutexes, FIFOs & Signals		
Simulation Kernel	Methods & Threads	Channels & Interfaces	Data Types: Logic, Integers, Fixedpoint & Floatingpoint
	Events, Sensitivity & Notifications	Modules & Hierarchy	

C++

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Simple Sockets

- "Simple" because they are easy to use – less to code …
- Do not bind sockets to objects, instead register methods with each socket
- Dummy method implementations are provided:
    - `get_direct_mem_ptr`
    - `transport_dbg`
    - `invalidate_direct_mem_ptr`
    - `nb_transport_bw`
- Target need only register either **b_transport** or **nb_transport_fw**
- Automatic conversion between blocking and non-blocking

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Simple Sockets: Initiator Example

```cpp
#include "tlm_utils/simple_initiator_socket.h"
...

SC_MODULE(Initiator) {
    public:
    tlm_utils::simple_initiator_socket<Initiator> iSocket;
    ...
    public:
    SC_CTOR(Initiator): iSocket("iSocket"), requestInProgress(0), peq(this, &Initiator::peqCallback)
    {
        iSocket.register_nb_transport_bw(this, &Initiator::nb_transport_bw);

        SC_THREAD(process);
        ...
    }

    void process() {
        ...
    }

    tlm::tlm_sync_enum nb_transport_bw(tlm::tlm_generic_payload& …, tlm::tlm_phase& …, sc_time& …) {
        ...
    }
    ...
};
```

> Instantiate simple socket

> Register function

> Implementation of Function

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/blob/master/tlm_simple_sockets/

49

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Simple Sockets: Target Example

```cpp
#include "tlm_utils/simple_target_socket.h"
...
SC_MODULE(Target)
{
    tlm_utils::simple_target_socket<Target> tSocket;
    ...
    SC_CTOR(Target) : tSocket("tSocket"), ...
    {
        tSocket.register_b_transport(this, &Target::b_transport);
        tSocket.register_nb_transport_fw(this, &Target::nb_transport_fw);
    }

    void b_transport(tlm::tlm_generic_payload& trans, sc_time& delay) {
        ...
    }

    tlm_sync_enum nb_transport_fw(tlm::tlm_generic_payload& …, tlm::tlm_phase& …, sc_time& …) {
        ...
    }
...
};
```

Instantiate simple target socket

Register functions

Implementation of Functions

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Tagged Simple Sockets: Interconnect Example

`simple_target_socket_tagged`

```
Initiator        Interconnect        Target
   A                  A                 A
               1             1
Initiator                            Target
   B                                   B
               2             2
```

`simple_initiator_socket_tagged`

- Register the same callback method with several sockets
- Distinguish origin of incoming transactions using socket id
- Interconnect is usually template class for number of target and initiator sockets

51

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Tagged Simple Sockets: Interconnect Example

```cpp
template<unsigned int I, unsigned int T>
SC_MODULE(BUS)
{
    public:
    tlm_utils::simple_target_socket_tagged<BUS> tSocket[T];
    tlm_utils::simple_initiator_socket_tagged<BUS> iSocket[I];

    SC_CTOR(BUS)
    {
        for(unsigned int i = 0; i < T; i++)
        {
            tSocket[i].register_b_transport(this,
                &BUS::b_transport, i);
            tSocket[i].register_nb_transport_fw(this,
                &BUS::nb_transport_fw, i);
        }

        for(unsigned int i = 0; i < I; i++)
        {
            iSocket[i].register_nb_transport_bw(this,
                &BUS::nb_transport_bw, i);
        }
    }

    private:
    std::map<tlm::tlm_generic_payload*, int> bwRoutingTable;
    std::map<tlm::tlm_generic_payload*, int> fwRoutingTable;

    virtual void b_transport( int id,
                        tlm::tlm_generic_payload& trans,
                        sc_time& delay )
    {
        sc_assert(id < T);
        int outPort = routeFW(id, trans, false);
        iSocket[outPort]->b_transport(trans, delay);
    }
```

```cpp
    virtual tlm::tlm_sync_enum nb_transport_fw( int id,
                tlm::tlm_generic_payload& trans,
                tlm::tlm_phase& phase,
                sc_time& delay )
    {
        sc_assert(id < T);
        int outPort = 0;

        if(phase == tlm::BEGIN_REQ) {
            trans.acquire();
            outPort = routeFW(id, trans, true);
        }
        else if(phase == tlm::END_RESP) {
            outPort = fwRoutingTable[&trans];
            trans.release();
        }
        else {
            SC_REPORT_FATAL(name(),"ERROR!");
        }

        return iSocket[outPort]->nb_transport_fw(
                                    trans, phase, delay);
    }

    virtual tlm::tlm_sync_enum nb_transport_bw( int id,
                tlm::tlm_generic_payload& trans,
                tlm::tlm_phase& phase,
                sc_time& delay )
    {
        int inPort = bwRoutingTable[&trans];

        return tSocket[inPort]->nb_transport_bw(
                                    trans, phase, delay);
    }
};
```

52

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Simple Sockets: Target Example
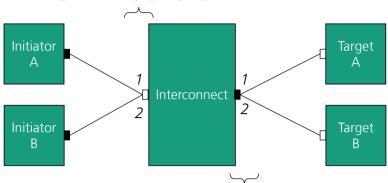
```
int routeFW(int inPort, tlm::tlm_generic_payload &trans, bool store)
{
    int outPort = 0;

    if(trans.get_address() < 512)
    {
        outPort = 0;
    }
    else if(trans.get_address() >= 512 && trans.get_address() < 1024)
    {
        trans.set_address(trans.get_address() - 512);
        outPort = 1;
    }
    else {
        trans.set_response_status( tlm::TLM_ADDRESS_ERROR_RESPONSE );
    }

    if(store) {
        bwRoutingTable[&trans] = inPort;
        fwRoutingTable[&trans] = outPort;
    }

    return outPort;
}
```

Implementation of the memory map

Correct address, i.e. subtract offset such that target gets always addresses starting at 0

Store from where transaction comes and where it goes

53

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Simple Sockets: Target Example

```cpp
int sc_main (...)
{

    Initiator * cpu1   = new Initiator("C1");
    Initiator * cpu2   = new Initiator("C2");

    Target * memory1   = new Target("M1");
    Target * memory2   = new Target("M2");

    Interconnect<2,2> * bus = new BUS<2,2>("B1");

    cpu1->iSocket.bind(bus->tSocket[0]);
    cpu2->iSocket.bind(bus->tSocket[1]);

    bus->iSocket[0].bind(memory1->tSocket);
    bus->iSocket[1].bind(memory2->tSocket);

    sc_start();

    return 0;
}
```

Number of components must be known at compile time!

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/blob/master/tlm_simple_sockets/

54

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Multipasstrough Sockets

Multi_passthrough_target_socket

multi_passtrough_initiator_socket

- Similar to Tagged sockets: also an id is used for identification
- The id is determined by the order of the binding to the multipasstrough socket
- Dynamic binding, number of comp. does not have to be known at compile time

55

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Multipasstrough Sockets: Interconnect Example

```cpp
SC_MODULE(BUS)
{
    public:
    tlm_utils::multi_passthrough_target_socket<BUS> tSocket;
    tlm_utils::multi_passthrough_initiator_socket<BUS> iSocket;

    SC_CTOR(Interconnect) : tSocket("tSocket"), iSocket("iSocket")
    {
        tSocket.register_b_transport(this, &BUS::b_transport);
        tSocket.register_nb_transport_fw(this, &BUS::nb_transport_fw);
        iSocket.register_nb_transport_bw(this, &BUS::nb_transport_bw);
    }

    private:
    std::map<tlm::tlm_generic_payload*, int> bwRoutingTable;
    std::map<tlm::tlm_generic_payload*, int> fwRoutingTable;

    void b_transport( int id, tlm::tlm_generic_payload& trans, sc_time& delay ) {
        ...
    }

    tlm::tlm_sync_enum nb_transport_fw( int id,  ... ) {
        ...
    }

    virtual tlm::tlm_sync_enum nb_transport_bw( int id, ...) {
        ...
    }
};
```

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Multipasstrough Sockets: Interconnect Example

```cpp
int sc_main (...)
{
    Initiator * cpu1  = new Initiator("C1");
    Initiator * cpu2  = new Initiator("C2");

    Target * memory1  = new Target("M1");
    Target * memory2  = new Target("M2");

    Interconnect * bus = new BUS("B1");

    cpu1->iSocket.bind(bus->tSocket);
    cpu2->iSocket.bind(bus->tSocket);

    bus->iSocket.bind(memory1->tSocket);
    bus->iSocket.bind(memory2->tSocket);

    sc_start();

    return 0;
}
```

> The order of the binding determines the ids for the function calls!

> Try code on github:
> https://github.com/TUK-SCVP/SCVP.artifacts/blob/master/tlm_multipasstrough_sockets

57

© Fraunhofer IESE

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

	User Libraries									
Transaction Level Modeling (TLM)	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions	**SystemC AMS**	Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		
SystemC	Predefined Primitive Channels: Mutexes, FIFOs & Signals								
	Simulation Kernel	Methods & Threads		Channels & Interfaces		Data Types: Logic, Integers, Fixedpoint & Floatingpoint			
		Events, Sensitivity & Notifications		Modules & Hierarchy					
	C++								

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Modelling of Backpressure

BEGIN_REQ
END_REQ
BEGIN_RESP
END_RESP

CPU — Memory Controller

- The exclusion rules enable flow control like back pressure:

  - E.g. if an input buffer of a target is full the target can defer the sending of END_REQ for the transaction that filled the buffer, until the buffer has available space again

  - An RTL ready signal can be modeled by deferring END_REQ

  - However, since it is non-blocking, the target can do something else, e.g. sending

- Also Response exclusion rule must be honored!

59

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Modelling of Backpressure

HW

```cpp
DECLARE_EXTENDED_PHASE(INTERNAL);

SC_MODULE(Target) {
    bool responseInProgress;
    tlm::tlm_generic_payload* endRequestPending;
    tlm_utils::peq_with_cb_and_phase<Target> peq;
    unsigned int numberOfTransactions;
    unsigned int bufferSize;
    std::queue<tlm::tlm_generic_payload*> responseQueue;
    ...

    void peqCallback(...) {
        sc_time delay;

        if(phase == tlm::BEGIN_REQ) {
            trans.acquire();
            if (numberOfTransactions < bufferSize) {
                sendEndRequest(trans);
            } else {
                endRequestPending = &trans;
            }
        } else if (phase == tlm::END_RESP) {
            ...
            numberOfTransactions--;
            ...
            if (responseQueue.size() > 0) {
                gp * next = responseQueue.front();
                responseQueue.pop();
                sendResponse(*next);
            }
            if (endRequestPending) {
                sendEndRequest(*endRequestPending);
                endRequestPending = 0;
            }
        }
        else if(phase == INTERNAL) {
            executeTransaction(trans);
            if (responseInProgress) {
                responseQueue.push(&trans);
            } else {
                sendResponse(trans);
            }
        }
    }
}
```

> Internal protocol phase, instead of using a sensitive process!

> Check if input buffer is full

> Reduce transaction counter

> Send next response

> Unblock Initiator

> Honor response exclusion rule

```cpp
    void sendEndRequest(tlm::tlm_generic_payload& trans) {
        tlm::tlm_phase bw_phase;
        sc_time delay;
        bw_phase = tlm::END_REQ;
        delay = randomDelay();

        tlm::tlm_sync_enum status;
        status = socket->nb_transport_bw(trans, bw_phase, delay);
        delay = delay + randomDelay();
        peq.notify(trans, INTERNAL, delay);

        numberOfTransactions++;
        printBuffer(bufferSize, numberOfTransactions);
    }

    void sendResponse(tlm::tlm_generic_payload& trans) {
        tlm::tlm_sync_enum status;
        tlm::tlm_phase bw_phase;
        sc_time delay;

        responseInProgress = true;
        bw_phase = tlm::BEGIN_RESP;
        delay = SC_ZERO_TIME;
        status = socket->nb_transport_bw( trans, bw_phase, delay );

        if (status == tlm::TLM_UPDATED) {
            peq.notify(trans, bw_phase, delay);
        }
        else if (status == tlm::TLM_COMPLETED) {
            numberOfTransactions--;
            responseInProgress = false;
        }
        trans.release();
    }
};
```

> Try code on github:
> https://github.com/TUK-SCVP/SCVP.artifacts/blob/master/tlm_at_backpressure

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

60

The SystemC language architecture diagram:

Transaction Level Modeling (TLM)						SystemC AMS			
	Sockets & Generic Payload	Blocking & Non-Blocking	Temporal Decoupling & DMI	Phases	Payload Extensions		Electrical Linear Networks (ELN)	Linear Signal Flow (LSF)	Timed Data Flow (TDF)
							Linear DAE solver		Scheduler
							Synchronization layer		

User Libraries (top spanning row)

Predefined Primitive Channels: Mutexes, FIFOs & Signals

SystemC			
Simulation Kernel	Methods & Threads	Channels & Interfaces	Data Types: Logic, Integers, Fixedpoint & Floatingpoint
	Events, Sensitivity & Notifications	Modules & Hierarchy	

C++

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

61

Self defined Protocol Phases

```
BEGIN_REQ  ───────────────────────────────►
END_REQ    ◄───────────────────────────────

OWN_PHASE1 ◄───────────────────────────────

OWN_PHASE2 ───────────────────────────────►

BEGIN_RESP ◄───────────────────────────────
END_RESP   ───────────────────────────────►
```

- Base protocol can be enhanced by phase extensions
- Usage of the macro: **DECLARE_EXTENDED_PHASE**(INTERNAL);
- More synchronization points → higher accuracy → slower simulation speed

62

Interleaving of transaction is possible   **Fraunhofer**
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Payload Extensions

- Payload extensions are a flexible and powerful method to carry additional non-standard attributes (e.g. priority, …)
- Extensions are attached to the normal generic payload
- Extensions can be:
  - Ignorable (ensures compatibility with base protocol)
  - Mandatory (a new protocol is needed!)
  - Private (only used by a single module e.g. Interconnect for storing routing information instead using maps)
  - Sticky (remain when transaction is returned to a pool)
  - Auto (freed when transaction is returned to a pool)

Generic payload object

Command
Address
Data
Byte Enables
Response Status

Extension A
Extension B
Extension C

64

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Payload Extensions Example

```cpp
class routingExtension : public
tlm::tlm_extension<routingExtension> {
    private:
    unsigned int inputPortNumber;
    unsigned int outputPortNumber;

    public:
    routingExtension(unsigned int i, unsigned int o) :
        inputPortNumber(i), outputPortNumber(o)
    {}

    tlm_extension_base* clone() const {
        return new routingExtension(
            inputPortNumber, outputPortNumber);
    }

    void copy_from(const tlm_extension_base& ext) {
        const routingExtension& cpyFrom =
                static_cast<const routingExtension&>(ext);
        inputPortNumber = cpyFrom.getInputPortNumber();
        outputPortNumber = cpyFrom.getOutputPortNumber();
    }

    unsigned int getInputPortNumber() const {
        return inputPortNumber;
    }

    unsigned int getOutputPortNumber() const {
        return outputPortNumber;
    }
};
```

```cpp
SC_MODULE(Interconnect)
{

    ...

    routingExtension* ext;

    ext = new routingExtension(
        inPort, outPort);

    trans.set_auto_extension(ext);

    ...

    routingExtension *ext = NULL;

    trans.get_extension(ext);

    outPort = ext->getOutputPortNumber();

    ...

};
```

Try code on github:
https://github.com/TUK-SCVP/SCVP.artifacts/blob/master/tlm_payload_extensions/

65

Fraunhofer
IESE

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

Your notes:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____