Hardware / Software Codesign

Transaction Level Modeling with SystemC

Florian Eibensteiner

Embedded Systems Design
FH Hagenberg ESD

2019

R 4367

Inhalt

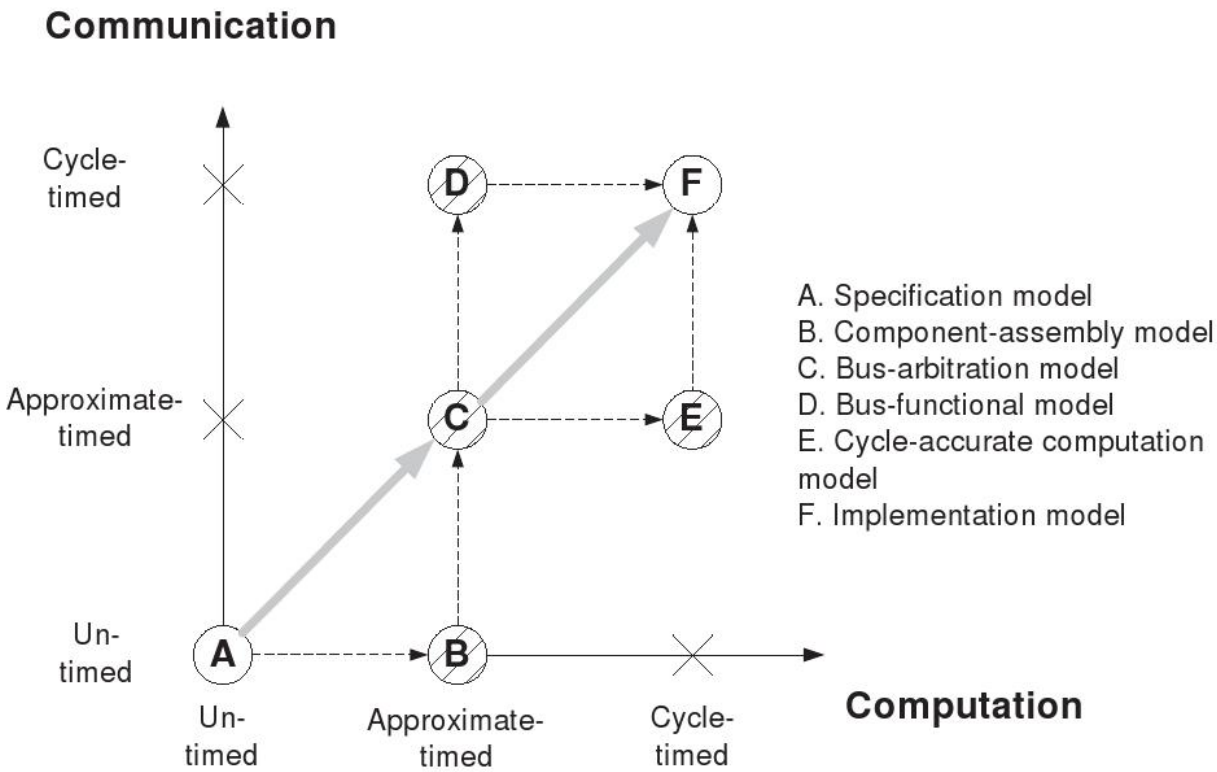
TLM-2.0

- Aktuelle Version: TLM 2.0.1 (bereits in SystemC 2.3.2 enthalten)
- Source Code, Dokumentation und Beispiele auf www.accellera.org erhältlich
- Standardisierung bei IEEE im Laufen
- Einbindung über Header-Files, keine Bibliothek erforderlich
- Erweiterung für SystemC um Kommunikation bzw. Transaktionen zwischen Systemkomponenten zu modellieren

- Literatur (Dokumentation im TLM-Package):
  - OSCI TLM-2.0: The Transaction Level Modeling Standard of the Open SystemC Initiative (OSCI)
  - M. Montoreano: Transaction Level Modeling using OSCI TLM 2.0

Remember – System Modeling Graph

- Auftrennung der Funktionalität in Kommunikation und Berechnung



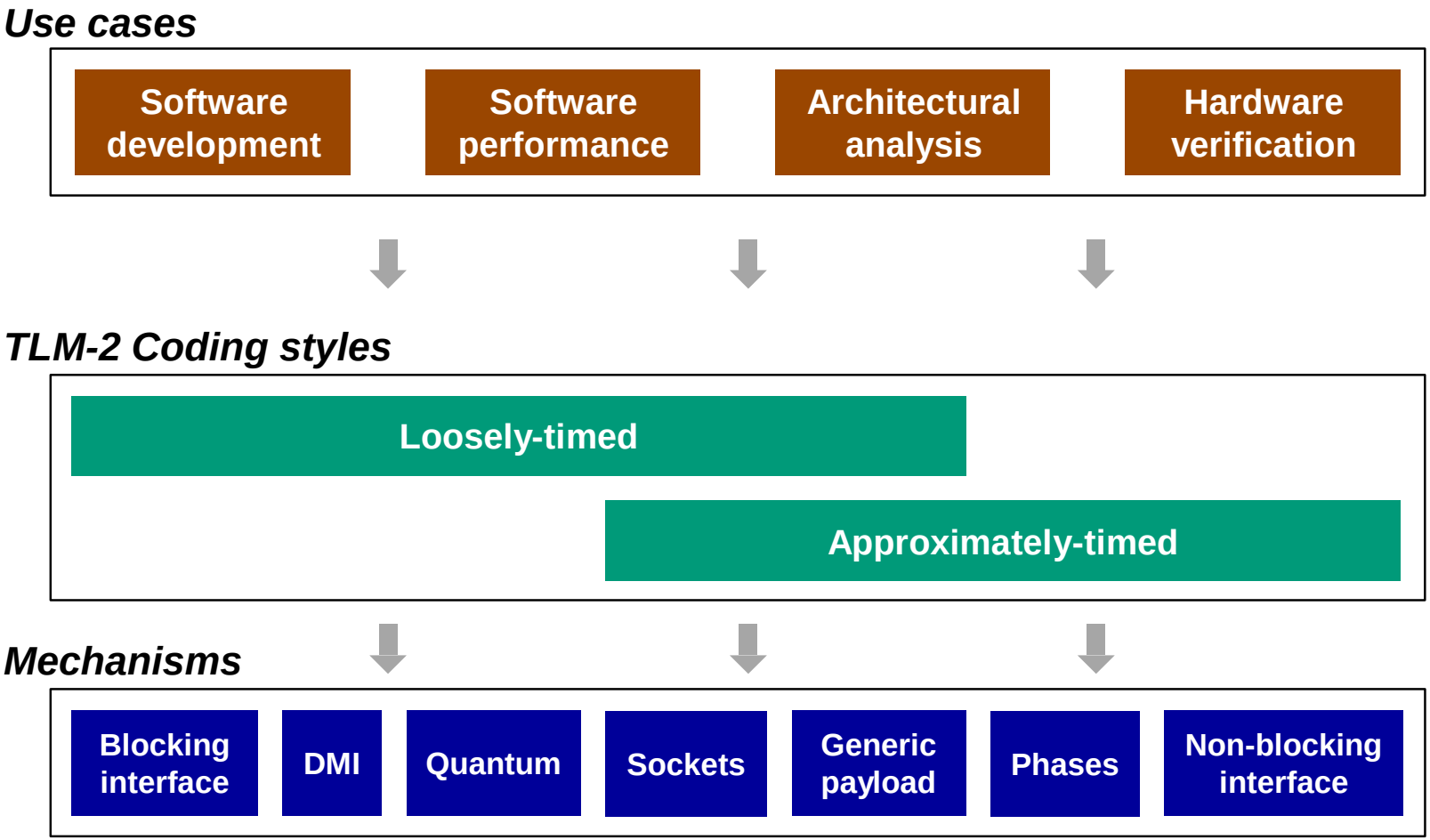
Warum soll TLM verwendet werden?

- Systemmodell mit den wichtigsten Hardwarekomponenten und Schnittstellen
- Exploration der Architektur und Performance-Analyse
- Firmwareentwicklung – Ausführen der Software auf einem virtuellen Prototyp
- Verifikation der Hardware durch ein „Golden Model“
- Geschwindigkeit:
  - Systementwurf schneller als bei RTL
  - Simulation wesentlich schneller als bei RTL-Modellen

Was bringt TLM-2.0.1?

- Transaction-Level Memory Mapped Bus Modeling (kompatible API)
- Register- und bitgenaue Schnittstellen
- Geschwindigkeit: Ausführen von Software (auch OS)
- Loosely-timed und Approximately-Timed Modeling
- Generische Mechanismen für die Modellierung der Payload und anderer Erweiterungen
- Blocking und non-blocking Transfers

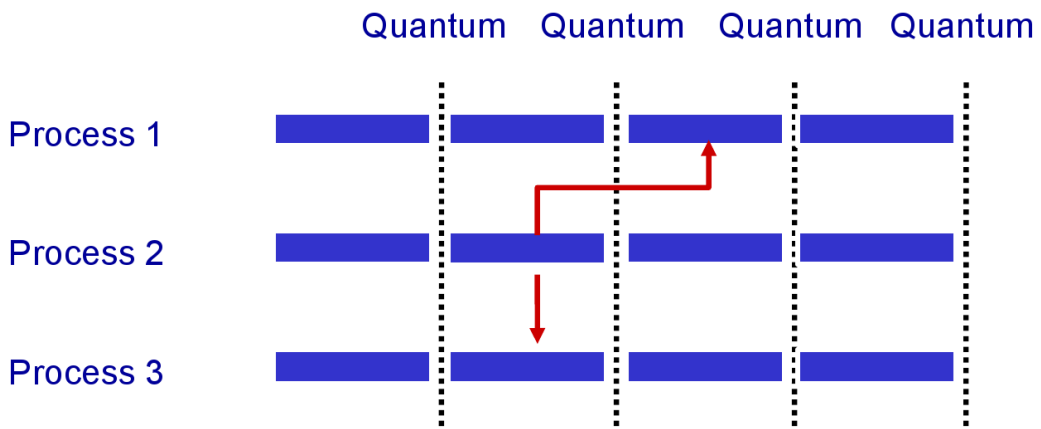
Use Cases, Coding Styles, and Mechanisms



Inhalt

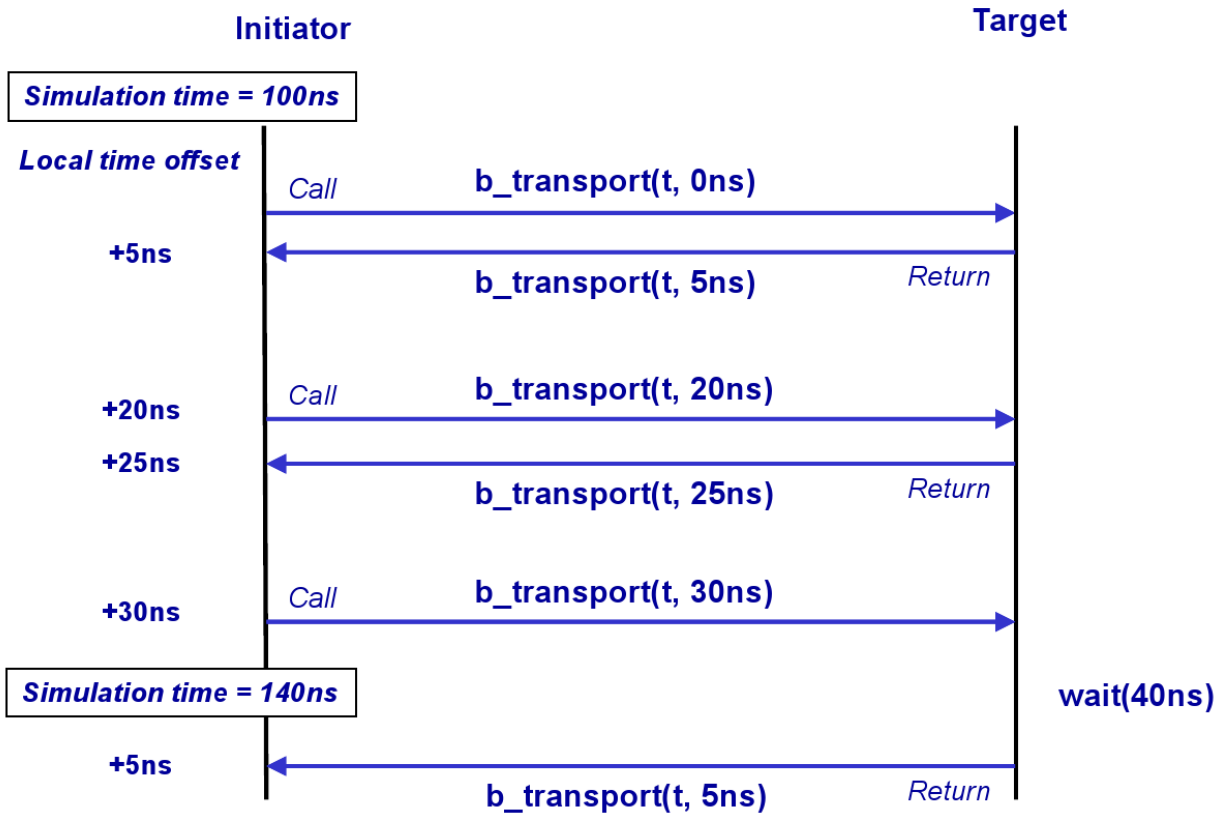
Loosely-Timed

- Loser Zusammenhang zwischen Timing und Daten (Programmer's View → PV-Modell)
- Prozesse können in der Simulationszeit vorlaufen (temporal decoupling)
- Minimierung der Context-Switches → sehr schnelle Simulationen
- Synchronisation um deterministische Kommunikation zu ermöglichen → Zeit vergeht in vielfachen eines Quatums

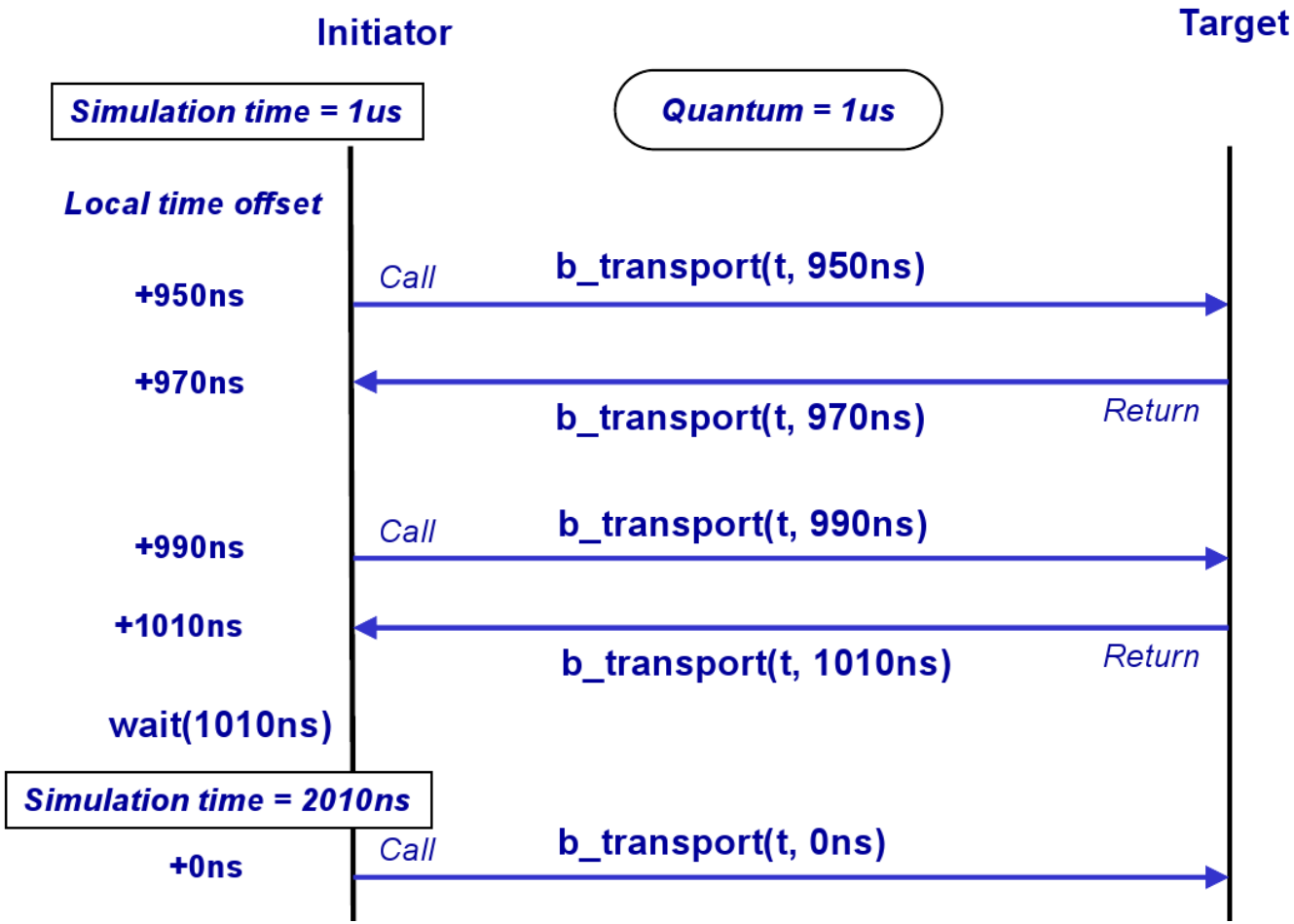


Temporal Decoupling

- Prozesse können in der Zeit vorlaufen
- Synchronisation aller Prozesse nach verstreichen eines Zeitquantums

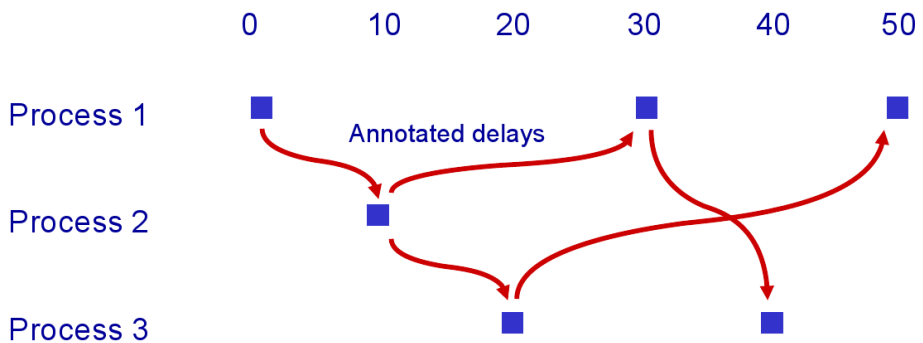


Time Quantum



Approximately-Timed

- Cycle-Count-Accurate oder Cycle-Approximate (Programmer's View with Time $\rightarrow$ PVT-Modell)
- Notwendig für Exploration der Architektur und Analyse der Performance
- Prozesse laufen synchron zur Simulationszeit



Blocking Transport

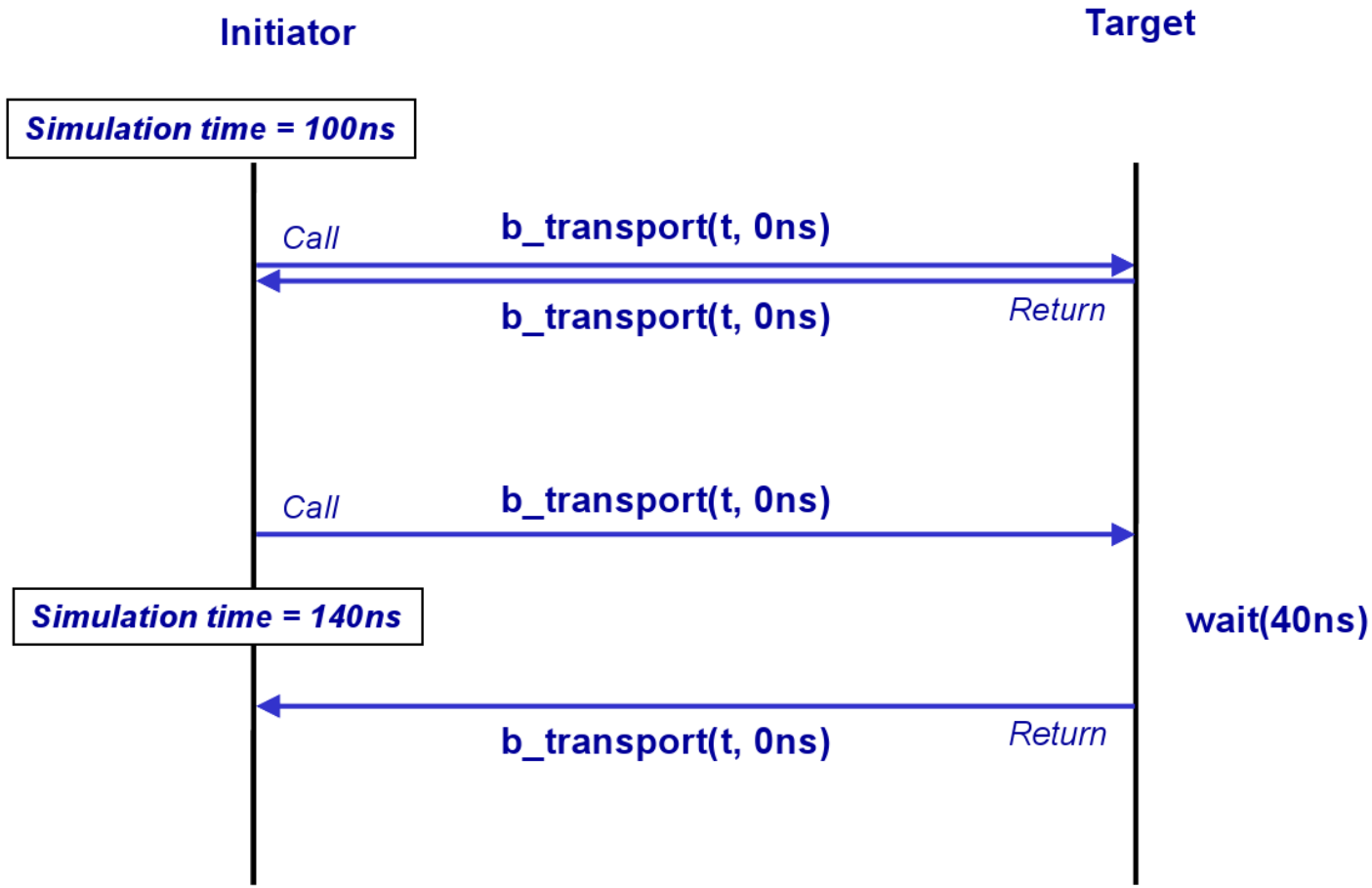
- Initiator ruft Funktion des Targets auf (läuft im Kontext des Callers)
- Initiator wird solange blockiert bis Transaktion abgeschlossen ist

Transaction type

```
template < typename  TRANS  =  tlm_generic_payload >

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

Transaction object *Timing annotation*

Blocking Transport



Blocking Transport – Beispiel

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

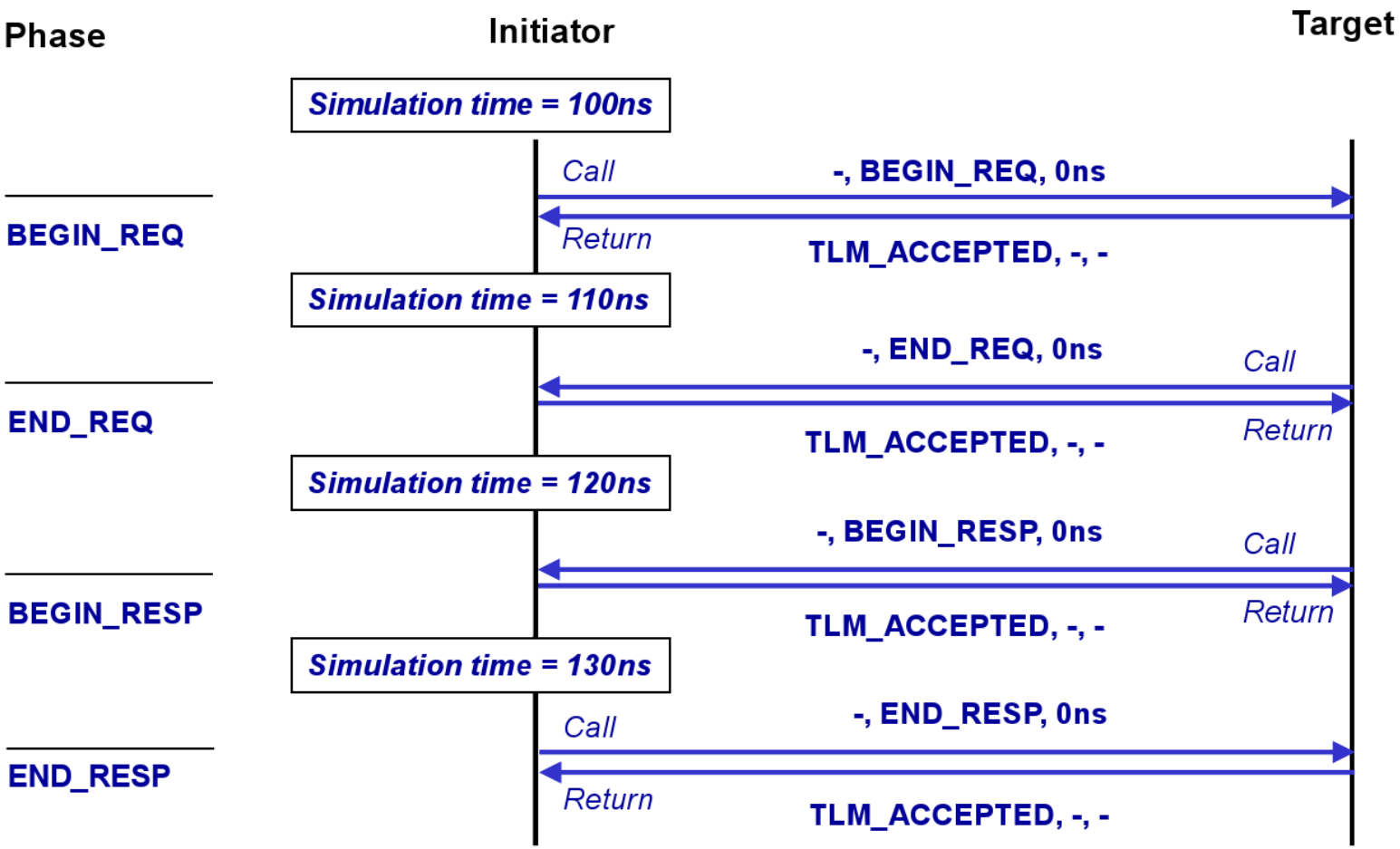
- Empfänger führt Transaktion unmittelbar aus, Out-Of-Order – Loosely-Timed
- Empfänger führt Transaktion aus und wartet – Approximately-Timed
- Wird eher für Loosely-Timed Modelle verwendet

Non-Blocking Transport

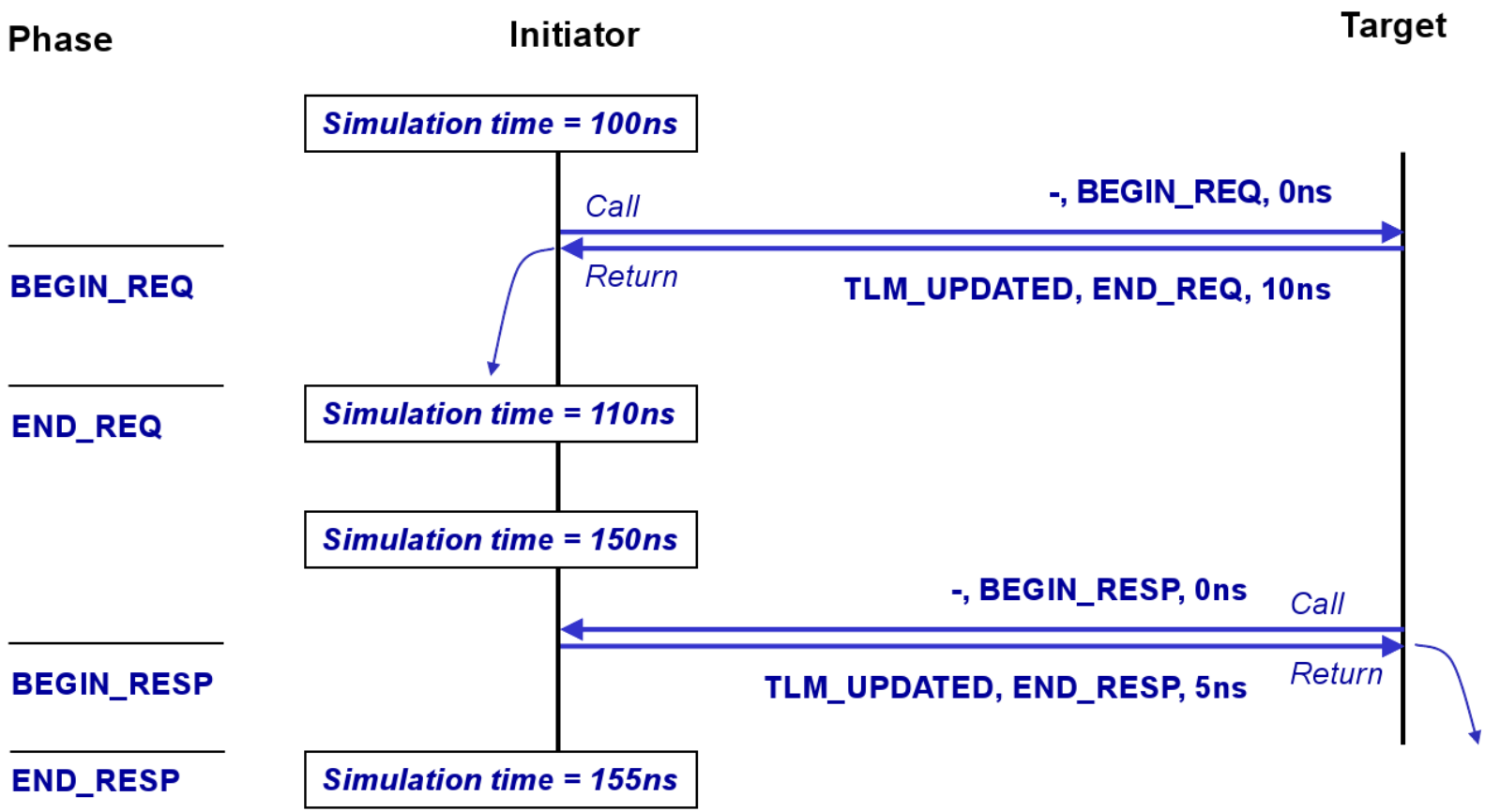
- Transaktion wird auf mehrere Phasen aufgeteilt.
- Einzelne Phasen benötigen Zeit – Timing kann besser angepasst werden
- Initiator wird nicht mehr blockiert – Kommunikation läuft über *Forward*- und *Backward*- bzw. *Return*-Pfad

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


template < typename TRANS = tlm_generic_payload,
           typename PHASE = tlm_phase>

class tlm_fw_nonblocking_transport_if : public virtual sc_core::sc_interface {
public:
  virtual tlm_sync_enum nb_transport(      TRANS& trans,
                                           PHASE& phase,
                                           sc_core::sc_time& t ) = 0;
};
```

Non-Blocking Transport – Backward Path



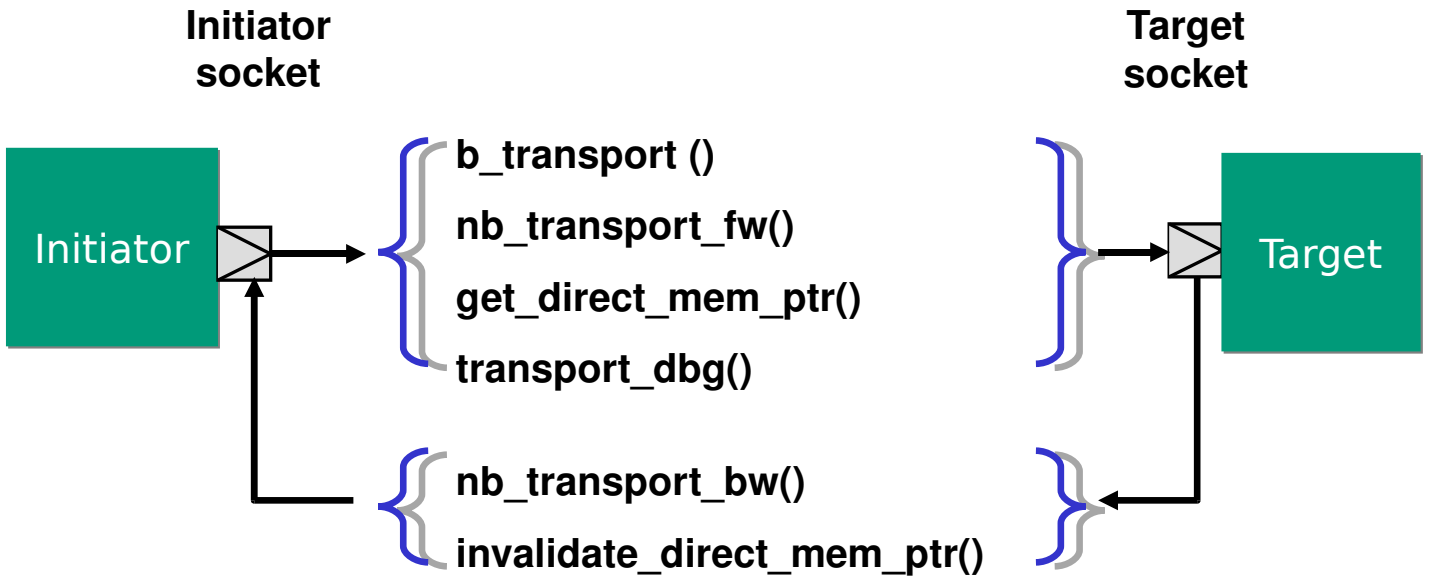
Non-Blocking Transport – Return Path



Inhalt

Initiator and Target

- Initiator: Modul welches eine neue Transaktion beginnt
- Target: Modul welches auf die Transaktion antwortet
- Transaktion: Datenstruktur welche zwischen Initiator und Target ausgetauscht wird
- Ein Modul kann beides sein (Initiator und Target)
- Austausch von Transaktionen läuft über Sockets



Standard Socket

- Blocking und Non-blocking Transport können gemsicht werden
- Hierarchische Verbindungen sind möglich
- Initator: verbinden mit Schnittstelle die Backward-Interface implementiert
- Target: verbinden mit Schnittstelle die Forward-Interface implementiert

```
template <  unsigned int  BUSWIDTH = 32,
                typename  TYPES    = tlm_base_protocol_types,
                     int  N        = 1,
 sc_core::sc_port_policy  POL      = sc_core::SC_ONE_OR_MORE_BOUND>

class tlm_initiator_socket
...
class tlm_target_socket
```

Socket – Beispiel Initiator

```
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

Socket – Beispiel Target

```
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( ... ) { ... }
};

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 );
  }
};
```

Combined interface required by socket

Protocol type default to base protocol

Target socket bound to target itself

Methods for forward path

Bind initiator socket to target socket

Socket – Beispiel Simple Socket

```
struct  Interconnect : sc_module
{
 tlm_utils::simple_target_socket<Interconnect>    targ_socket;
 tlm_utils::simple_initiator_socket<Interconnect> init_socket;

 SC_CTOR(Interconnect) : targ_socket("targ_socket"), init_socket("init_socket")
 {
  targ_socket.register_nb_transport_fw(    this, &Interconnect::nb_transport_fw);
  targ_socket.register_b_transport(        this, &Interconnect::b_transport);
  targ_socket.register_get_direct_mem_ptr(this, &Interconnect::get_direct_mem_ptr);
  targ_socket.register_transport_dbg(      this, &Interconnect::transport_dbg);
  init_socket.register_nb_transport_bw(    this, &Interconnect::nb_transport_bw);
  init_socket.register_invalidate_direct_mem_ptr(
                                           this,  &Interconnect::invalidate_direct_mem_ptr);
 }
 virtual void b_transport( ... );
 virtual tlm::tlm_sync_enum  nb_transport_fw( ... );
 virtual bool get_direct_mem_ptr( ... );
 virtual unsigned int transport_dbg( ... );
 virtual tlm::tlm_sync_enum nb_transport_bw( ... );
 virtual void invalidate_direct_mem_ptr( ...);
};
```

Generic Payload

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