Methodology for Defining

Bus Specific Extensions to TLM2.0

(Programmer's View & Architecture View)

Umesh Sisodia, usisodia@circuitsutra.com

CEO, CircuitSutra Technologies

CIRCUIT SUTRA TECHNOLOGIES

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION DAY

OCT 31, 2019 | MUNICH | GERMANY

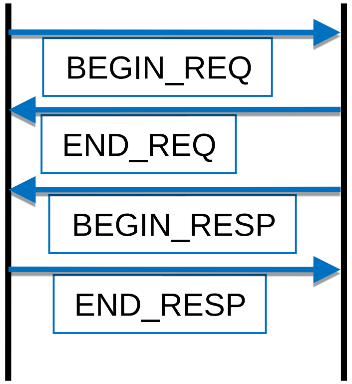
TLM2.0 Overview

TLM2.0 - Accellera standard for the modelling of memory mapped buses

Generic payload

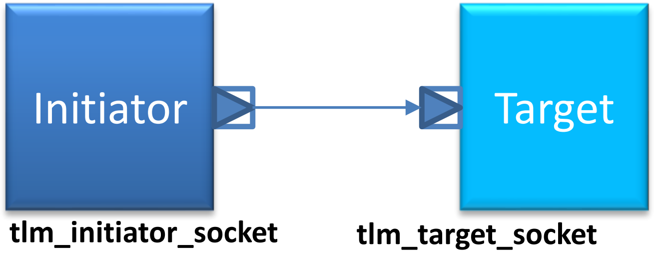
Command
Address
Data
Byte enables
Response status

Phases



Sockets

Templatized with base protocol



Base Protocol

Bus Agnostic Generic protocol

1. Trait Class:

struct tlm_base_protocol_types
{
typedef **tlm_generic_payload** tlm_payload_type;
typedef **tlm_phase tlm_phase**_type;
}

2. Protocol Rules

Core Interfaces

TRANSPORT INTERFACES

void **b_transport**(TRANS& trans, sc_core::sc_time& t)

tlm_sync_enum **nb_transport_fw**(TRANS& trans, PHASE& phase, sc_core::sc_time& t)

tlm_sync_enum **nb_transport_bw**(TRANS& trans, PHASE& phase, sc_core::sc_time& t);

DEBUG INTERFACE

DMI INTERFACES

Interoperability Layer

TLM2.0 – Need for extension

- Supporting SoC Bus specific features
  - ARM - AMBA AXI
  - SiFive TileLink Bus
  - Proprietary Buses
  - Western Digital – OmniExtend :Access off chip memory space
- Supporting lower abstraction levels
  - TLM2.0 supports Loosely Timed (LT) & Approximately Timed (AT) abstraction
  - Cycle Accurate (CA) requires extension of TLM2.0

Bus features not supported by TLM2.0

Cache Coherency
Atomic Operations
Burst Operations
Messages - LogicalData, ArithmeticData, Intent
Addressing options
Conformance level
Protection unit
QoS support
DVM support
Transaction hints
Master ID
..

TLM ABSTRACTION LEVELS

Widely used abstraction levels for Memory Mapped Buses

TL4 (TLM2.0 LT)

- Entire burst is transmitted as a single entity
- Two timing points: Start & End of transaction
- b_transport
  - tlm_generic_payload
- Use Case: Pre-silicon firmware development

TL3 (TLM2.0 AT)

- Entire burst is transmitted as a single entity
- Four timing points
- nb_transport
  - tlm_generic_payload
  - tlm_phase: BEGIN_REQ, END_REQ, BEGIN_RESP, END_RESP
- Use Case: Architecture Exploration

TL1 (TLM2.0 CA)

- A burst is broken into individual beats equivalent to bus_width
- A beat is transmitted as a single entity
- Timing points after every beat
- Extends TLM2.0
- Use Case: Architecture Exploration, Verification

PROGRAMMER'S VIEW EXTENSIONS to TLM2.0

- **Payload extensios for bus specific features**

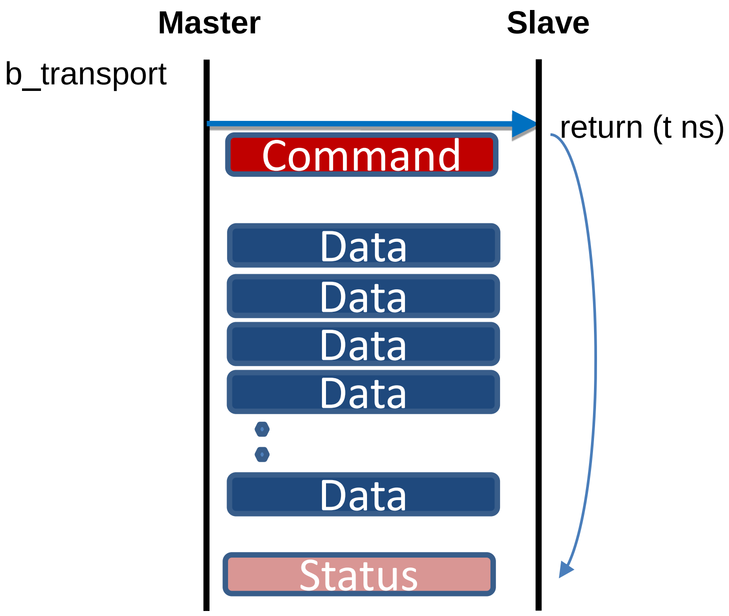
ARCHITECTURE VIEW EXTENSIONS to TLM2.0

- **Payload extensions for bus specific features**
  Superset of Programmer's View Extensions
- **Phase Extensions**
  BEGIN_DATA, END_DATA
  BEGIN_DATA, END_DATA, BEGIN_BURST, END_BURST,
  Any other phase specific to control signals / functionality not covered in payload extensions

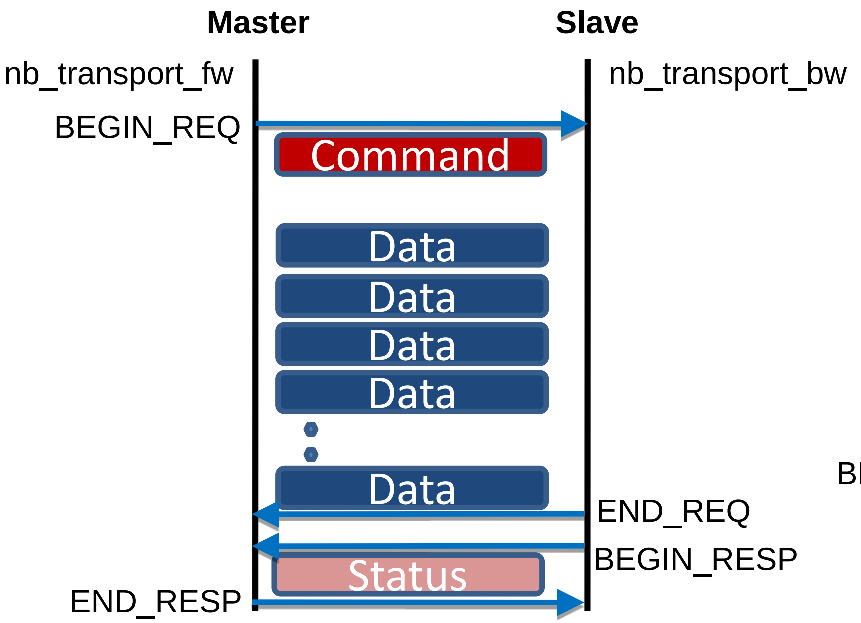
ABSTRACTION LEVELS

Sequence Diagrams: WRITE Transaction

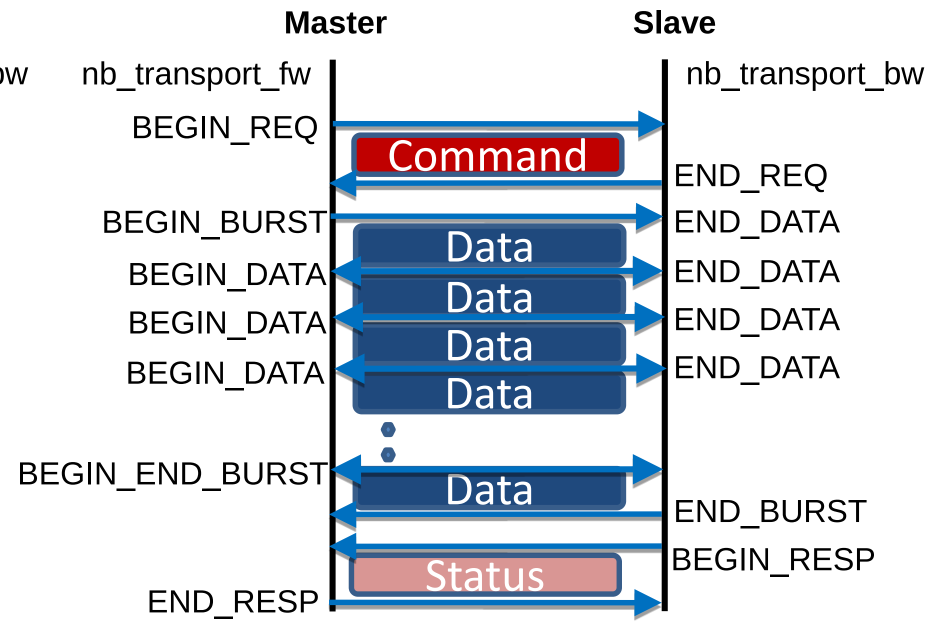
TL4 (TLM2.0 LT)



TL3 (TLM2.0 AT)



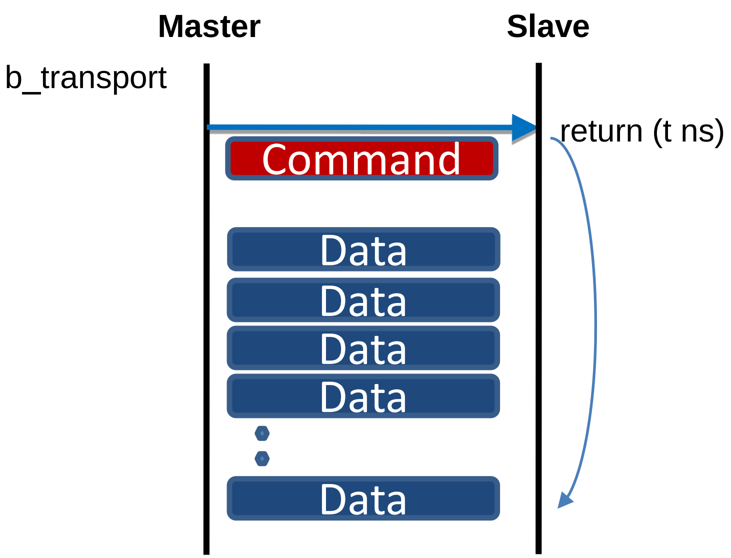
TL3 (TLM2.0 CA)



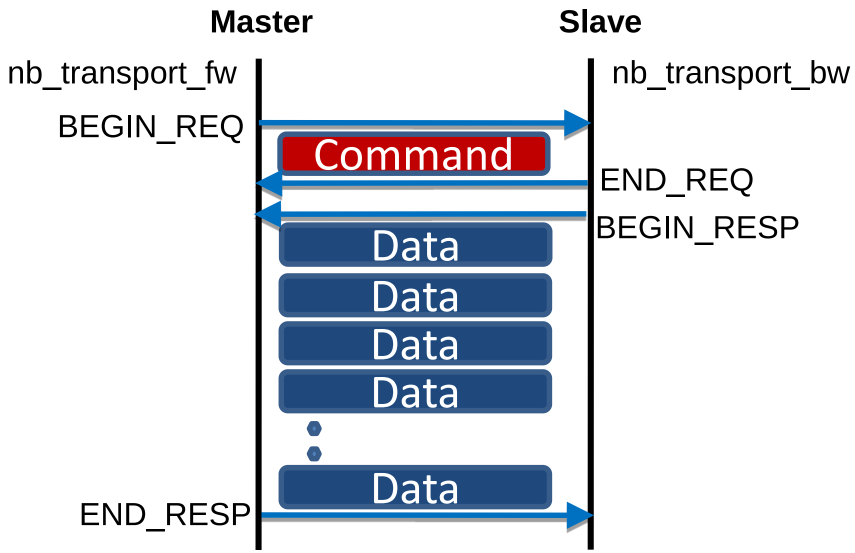
ABSTRACTION LEVELS

Sequence Diagrams: READ Transaction

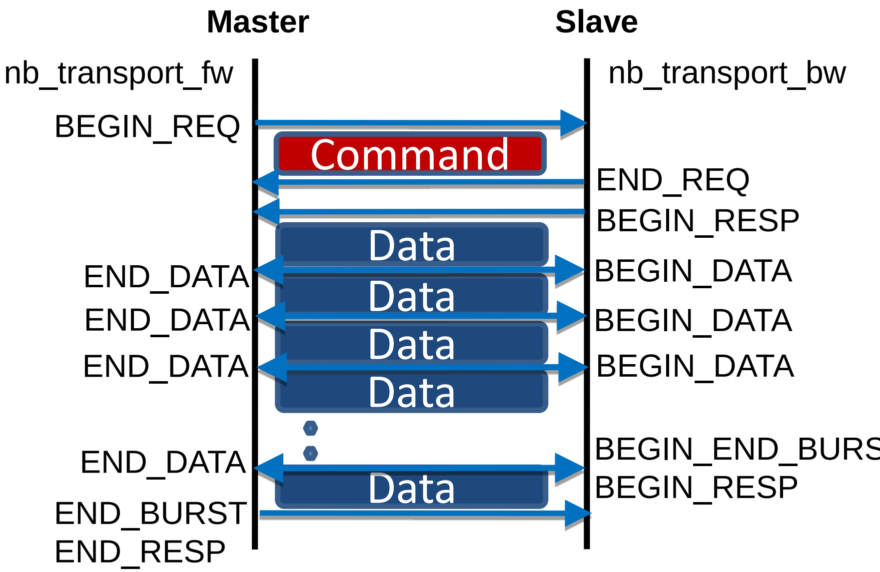
TL4 (TLM2.0 LT)



TL3 (TLM2.0 AT)



TL3 (TLM2.0 CA)



Opportunity for standardization

Potential Requirement

- Consistence method for Bus specific Extensions
  - Programmer's View extensions
  - Architecture View extensions
- Extending TLM2.0 for Cycle Accurate abstraction
- Mix & match models at different abstraction levels in the same simulation
- Change the abstraction level at run time
- Mix & match models having different bus protocols, different bus width
- Ease the model development
- Ease the development of adaptors

TLM2.0 Standard released in 2008

Widely being used in the industry

Right time to review the best practices. Enhance the standard to cover all aspects of memory mapped bus modelling

Requirement for the standardization of a cycle accurate coding style remains an open issue, possibly to be addressed by a future Accellera Systems Initiative standard.
SystemC Language Reference Manual

TLM2.0 Extension Mechanism

Base Protocol
Bus Agnostic Generic protocol

1. Trait Class:

```
struct tlm_base_protocol_types
{
typedef tlm_generic_payload tlm_payload_type;
typedef tlm_phase tlm_phase_type;
}
```

2. Protocol Rules

Ways to extend TLM2.0 Base Protocol

A. Use the generic payload directly, with ignorable extensions
B. Define a new protocol traits class containing a typedef for tlm_generic_payload.
C. Define a new protocol traits class and a new transaction type

The option (B) & (C) enforce the compile time bindability checks of sockets. Hence the master socket will bind with only that slave socket which has the exactly same protocol extension

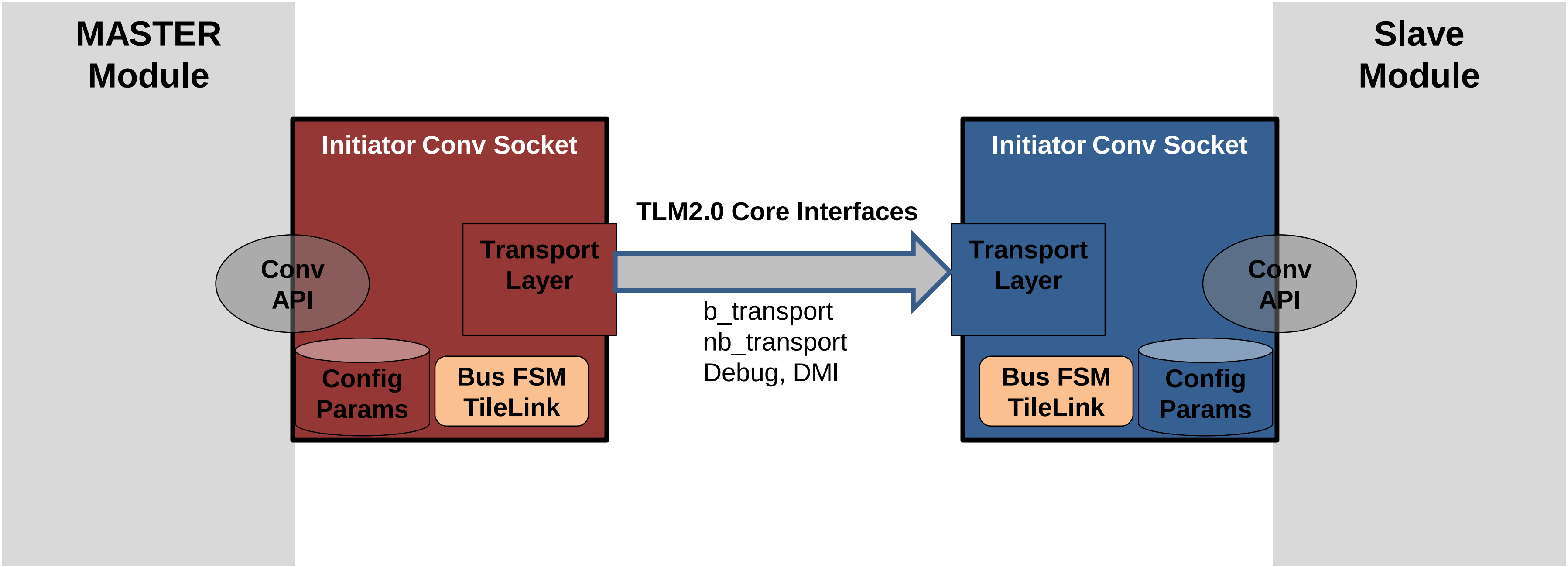
Our methodology
Use Option A, it provides greater flexibility and code re-usability

Run time bindability check. To check the compatibility of connecting sockets

BUS_WIDTH configuration at run time

Convenience Sockets

Encapsulate BUS protocol & complex TLM rules



Transport Layer

- TLM2.0 Core interfaces

Convenience Layer

- Convenience APIs
  - Bus protocol independent
  - Abstraction Independent
  - Easy to use APIs
  - Model developers do not worry about the complex bus protocol and TLM rules
- Run time bindability
- Configurable bus width
- Configurable Abstraction Level
- Memory Manager
- Automatic DMI usage

Bus protocol FSM

- Bus specific TLM2.0 Extensions:
  - Programmer View
  - Architecture View
- Supports Various Abstraction Levels
  - TL4: TLM2.0 LT
  - TL3: TLM2.0 AT
  - TL1: CA

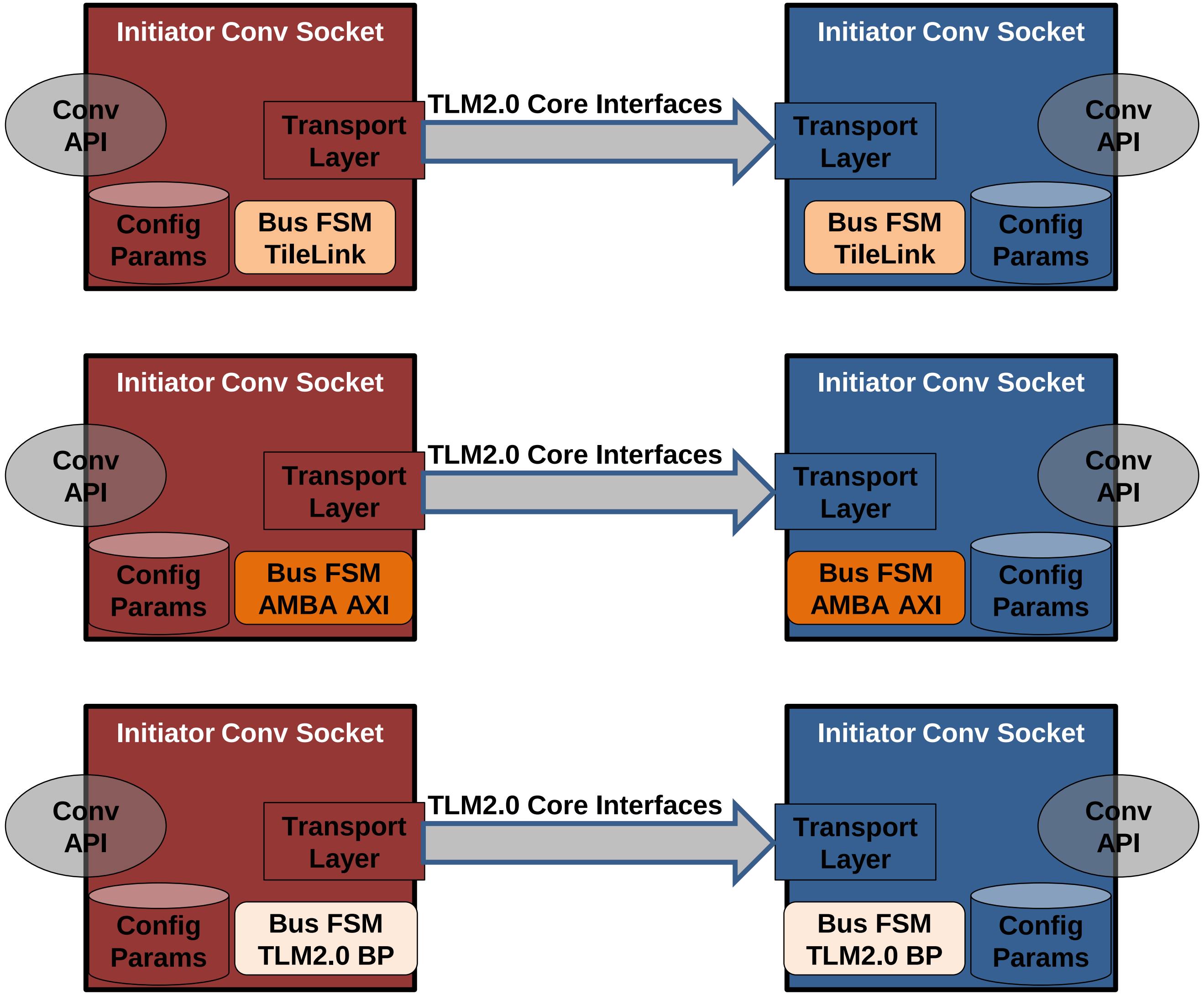
Elaboration Time Config Params

- buswidth, read_buswidth, write_buswidth
- Clock period
- Max Outstanding Transactions: READ / WRITE

Simulation Time Config Params

- Abstraction Levels (TL4, TL3, TL4)
- Tracing ON / OFF

Convenience Sockets



By changing the Bus FSM implementation we can quickly get the convenient socket for any SoC bus

TileLink Bus

TileLink_conv_initiator_socket<BUSWIDTH> i_socket

TileLink_conv_target_socket<BUSWIDTH> t_socket

AMBA Bus

AMBA_conv_initiator_socket<BUSWIDTH> i_socket

AMBA_conv_target_socket<BUSWIDTH> t_socket

TLM2.0 Base Protocol

TLM20_conv_initiator_socket<BUSWIDTH> i_socket

TLM20_conv_target_socket<BUSWIDTH> t_socket

Convenience Sockets

Configurable Bus width

- TLM2.0 Sockets: BUSWIDTH template parameter to enforce compile time bindability:

    tlm_initiator_socket <BUSWIDTH>

    tlm_target_socket <BUSWIDTH>

- Convenience socket
  - Supports BUSWIDTH template parameter. Makes it compatible with TLM2.0
  - Also supports to run time configuration of BUSWIDTH using config parameter. To activate this, set the template parameter value to 0

    TileLink_conv_initiator_socket<0>

    TileLink_conv_target_socket<0>

In-built DMI Handling

- Initiator Conv socket automatically uses DMI at TL4 abstraction level If the connected slave model support DMI,

Run time bindability

- Handshake between Initiator and Target socket during elaboration phase
- TLM_IGNORE_COMMAND sent with the set of parameters to check
- Following config parameters are checked for compatibility
  - Buswidth, read_buswidth, write_buswidth
  - clock

Changing the abstraction level at run time

- Initiator conv socket have a config parameter to set abstraction level at run time
- Initiator communicates change in abstraction to target using TLM_IGNORE_COMMAND
- Slave adjust itself to new abstraction level

Convenience Sockets – Master Convenience APIs

Forward APIs: Called by the model, Implemented in Convenient Socket

Get Transaction Pointer

tlm_generic_payload* get_trans(uint32_t data_size, uint32_t be_size)
tlm_generic_payload * get_trans_b(tlm::tlm_command cmd, uint32_t data_size, uint32_t be_size = 0)
tlm_generic_payload * get_trans_nb(tlm::tlm_command cmd, uint32_t data_size, uint32_t be_size=0,

READ & WRITE:

- **Blocking APIs**

void request_read_b(tlm::tlm_generic_payload &trans, sc_time& delay)
void request_write_b(tlm::tlm_generic_payload &trans, sc_time& delay)

- **Blocking APIs with Callback visitor object**

void request_read_b(tlm::tlm_generic_payload &trans, master_visitor_r &rcallback_obj, sc_time& delay)
void request_write_b(tlm::tlm_generic_payload &trans, master_visitor_w &wcallback_obj, sc_time& delay)

- **Non Blocking APIs with Callback visitor object**

void request_read_nb(tlm::tlm_generic_payload &trans, master_visitor_r &rcallback_obj, sc_time& delay)
void request_write_nb(tlm::tlm_generic_payload &trans, master_visitor_w &wcallback_obj, sc_time& delay)

DEBUG, DMI (for non end point master),
Other Misc functionality (Transaction Priority change etc..)

ABSTRACTION INDEPENDENT

The actual transaction may be broken into several phases and requires multiple forward and backward transport calls between master and slave socket.

All this handling is done within the convenience socket, the IP model is relieved from this complex handling

Convenience Sockets – Master Convenience APIs

Backward APIs: Called by the Convenience socket, Implemented in the model

- Model registers these APIs with the convenience layer through callback visitor objects while submitting the READ / WRITE transactions
- Called by the convenience layer at different timing points of during the lifecycle of a transaction

Callback Visitor for Read Transaction

request_status read_phase_data(boost::function0<void > resume_read, tlm::tlm_generic_payload &trans, uint32_t offset, uint32_t size, uint32_t& wait_clock_cycle, request_level r_level)

- Called after every READ_DATA phase of READ transaction
- **request_level:** end_of_phase, end_of_burst, end_of_transaction
- **Return Value:** OK, WAIT_STATE, WAIT_STATE_START
  - WAIT_STATE - Insert wait state for wait_clock_cycle
  - WAIT_STATE_START - Insert indefinite wait state, will be resumed by calling resume_read

Callback Visitor for WRITE Transaction

write_phase_data(tlm::tlm_generic_payload &trans, uint32_t offset, uint32_t& size)

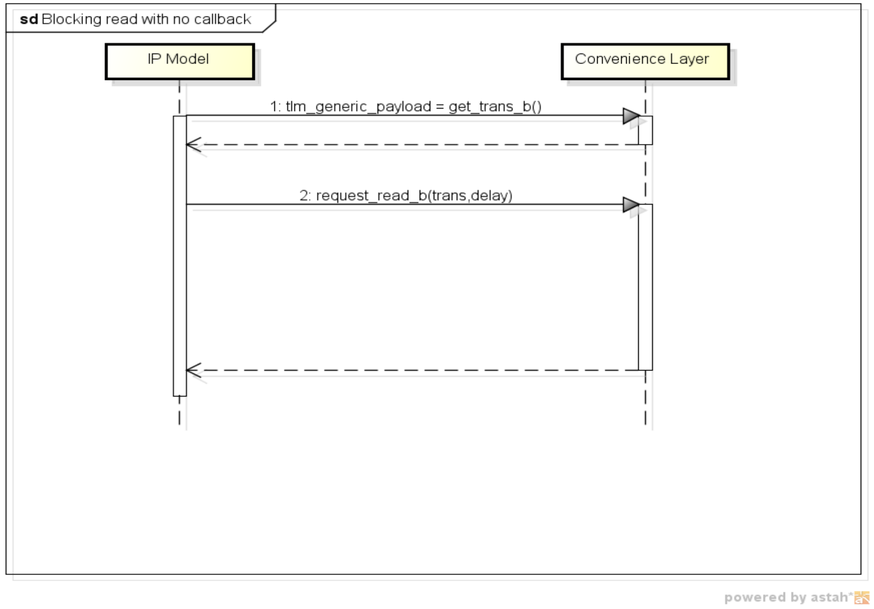
- Called after every WRITE_DATA phase of WRITE transaction

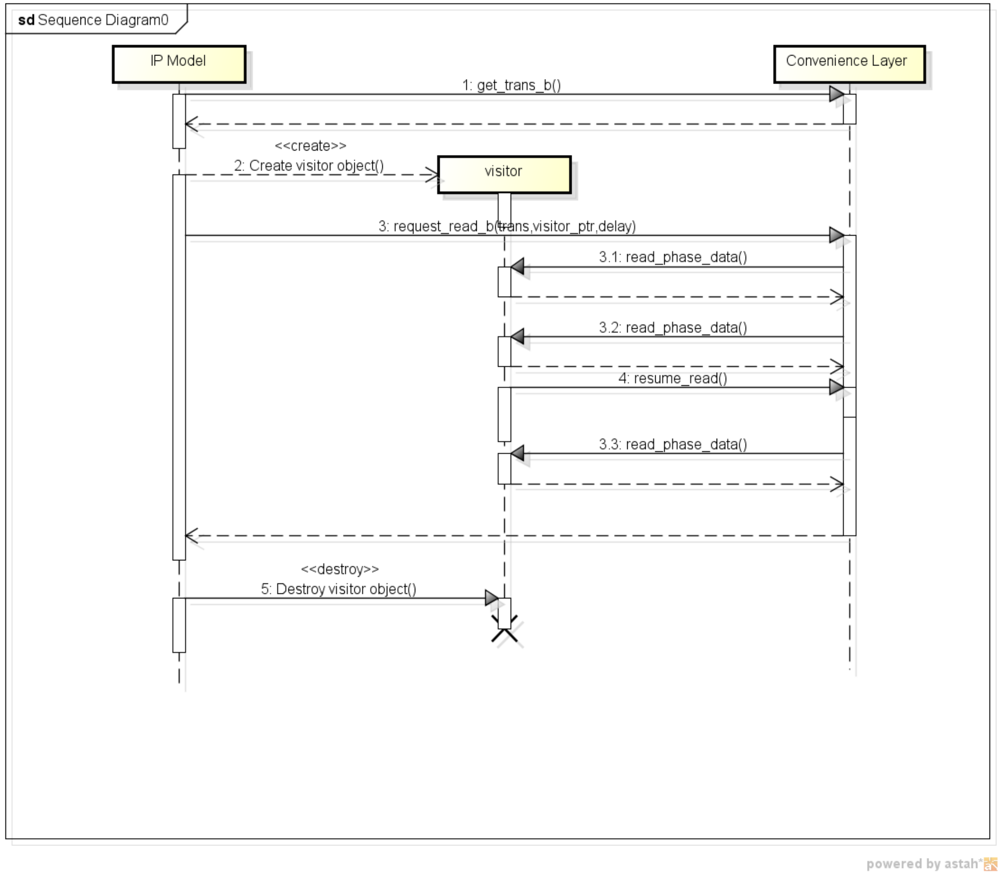
request_status write_status (boost::function0<void > resume_write_status, tlm::tlm_generic_payload &trans, uint32_t& wait_clock_cycle)

- Called to communicate the response of the write status sent by slave to master

Sequence Diagram: Master

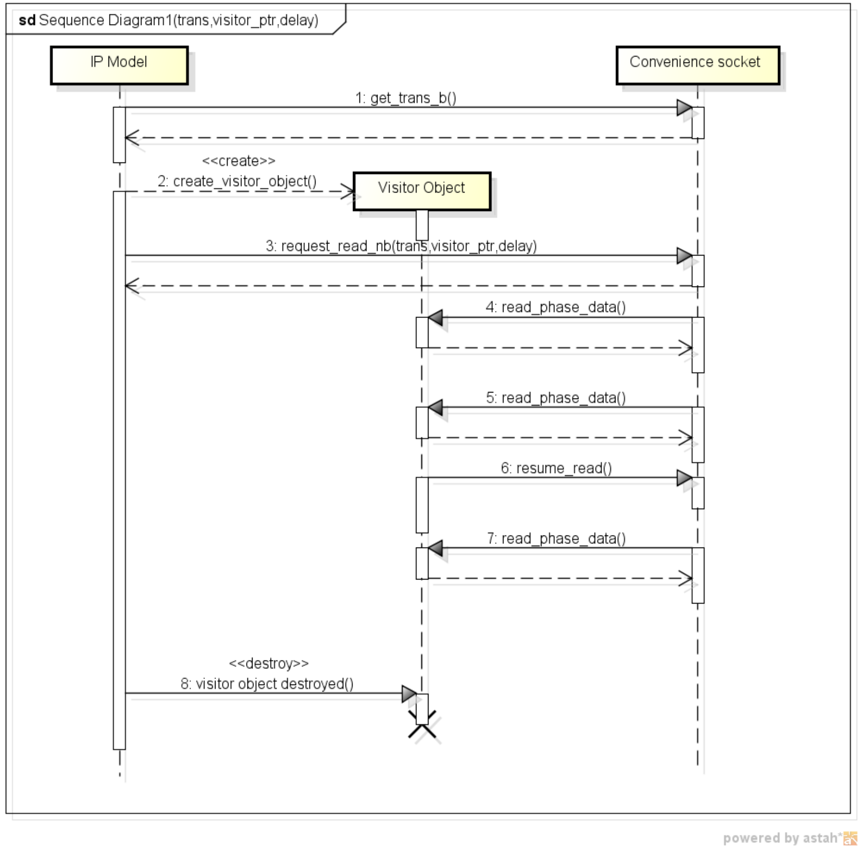
Blocking READ: with & without callback





Sequence Diagram: Master

Non Blocking READ Command



SYSTEMS INITIATIVE

SYSTEMC EVOLUTION DAY
OCT 31, 2019 | MUNICH | GERMANY

Convenience Sockets – Slave Convenience APIs

Work in Progress ..

- Slave to register the READ / WRITE handler with the conv socket. These will be called whenever the READ / WRITE transaction is received from master.
- Slave to register callback functions per transaction object. These will be called by conv socket at different timing points in the transaction
- Slave to have the option to provide / consume data in various units: One beat at a time, One burst at a time, Entire transaction in one go
- Slave should be able to insert wait state in the middle of a transaction
- Standard interface for DMI / Debug similar to TLM2.0 simple target socket

Thank you for your time

info@circuitsutra.com