TLM-2.0 in Action: An Example-based Approach to Transaction-level Modeling and the New World of Model Interoperability

John Aynsley, Doulos
TLM Introduction

CONTENTS

• What is TLM and SystemC?

• Creating a virtual platform model

• The OSCI TLM-2.0 standard

• Achieving speed and interoperability
Transaction Level Modeling

Pin Accurate

Simulate every event!

100-10,000 X faster simulation!

Copyright © 2007-2009 by Doulos. All rights reserved.
Reasons for using TLM

- **Software development**
- **Firmware / software**
- **Test bench**
  - **Fast**
  - **TLM**
    - Ready before RTL
  - **RTL**

- **Hardware verification**
  - "TLM = golden model"

Accelerates product release schedule
**Multiple Languages**

- **Firmware / software**
- **TLM**
- **RTL**
- **Test bench**

**Mixed language environments**

- SystemC used as golden reference
- VHDL, Verilog, SystemVerilog
- Home grown C++ / SystemC
- VHDL, Verilog for design
Reasons for using SystemC

Industry standard IEEE 1666™

Robust open source proof-of-concept C++ simulator
- Only requires a C++ compiler
- Flexibility re platforms and licensing
- Tool vendors add value

Easy integration

Common language across disciplines
- Builds bridges between system, s/w and h/w
TLM Introduction

CONTENTS

• What is TLM and SystemC?
• Creating a virtual platform model
• The OSCI TLM-2.0 standard
• Achieving speed and interoperability
Use Cases

- **FUNCTIONAL VIEW**
  - Algorithm developer
    - Untimed

- **ARCHITECTURE VIEW**
  - Tuning the platform
    - Approximately-timed

- **VERIFICATION VIEW**
  - Functional verification
    - Untimed through Cycle Accurate

- **RTL**
  - Implementation

- **PROGRAMMERS VIEW**
  - Software developer
    - Loosely-timed
Typical Use Case: Virtual Platform

Multiple software stacks

Software

CPU  ROM  RAM  DMA

Interrupt  Timer  I/O  Bridge

Multiple buses and bridges

Bridge

DSP  ROM  RAM

Interrupt  Timer  A/D

Digital and analog hardware IP blocks

I/O  Memory interface  RAM  DMA  Custom peripheral  D/A
Virtual Platform Characteristics 1

- Register accurate, functionally complete
- No clock, no pins, no implementation detail
- Loose or approximate timing only
- Fast enough to boot software O/S in seconds
- Available months before RTL
- Accurate enough to stay in use post-RTL
## Virtual Platform Characteristics 2

<table>
<thead>
<tr>
<th>Instruction Set Simulator or software stubs</th>
<th>Transaction-Level Model</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Available early</strong></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Fast enough to run applications</strong></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Little or no hardware detail</strong></td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td><strong>No timing information</strong></td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction-Level Model</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Available early</strong></td>
<td>✗</td>
</tr>
<tr>
<td><strong>Fast enough to run applications</strong></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Register-accurate</strong></td>
<td>✗</td>
</tr>
<tr>
<td><strong>Some timing information</strong></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Register-accurate and pin-accurate</strong></td>
<td>✓</td>
</tr>
<tr>
<td><strong>Cycle-accurate timing</strong></td>
<td>✓</td>
</tr>
</tbody>
</table>

- ✓: Available
- ✗: Not Available

Copyright © 2007-2009 by Doulos. All rights reserved.
Transaction-Level Modeling

Simple functional models, e.g. C programs

Concurrent simulation environment
SystemC / TLM is the Glue!

- Transaction-level modeling is communication-centric

Diagram:
- Native SystemC
  - Bus model
  - SystemC Wrapper
    - ISS
    - Object code
  - SystemC Wrapper
    - C/C++
  - SystemC Wrapper
    - VHDL
    - Verilog

TLM Introduction

CONTENTS

• What is TLM and SystemC?
• Creating a virtual platform model
• The OSCI TLM-2.0 standard
• Achieving speed and interoperability
OSCI TLM-2.0 Standard

• Transaction-level memory-mapped bus modeling
• Based on SystemC
• Released in June 2008
• OSCI LRM in 2009

Speed!

Interoperability!

comparable to ISS

between TLM models of IP blocks
Coding Styles and Mechanisms

**Use cases**

- Software development
- Software performance
- Architectural analysis
- Hardware verification

**TLM-2 Coding styles** (*just guidelines*)

- Loosely-timed
- Approximately-timed

**Mechanisms** (*definitive API for TLM-2.0 enabling interoperability*)

- Blocking transport
- DMI
- Quantum
- Sockets
- Generic payload
- Phases
- Non-blocking transport
Coding Styles

• **Loosely-timed** = as fast as possible
  - Only sufficient timing detail to boot O/S and run multi-core systems
  - Processes can run ahead of simulation time (temporal decoupling)
  - Each transaction completes in one function call
  - Uses direct memory interface (DMI)

• **Approximately-timed** = just accurate enough for performance modeling
  - *aka* cycle-approximate or cycle-count-accurate
  - Sufficient for architectural exploration
  - Processes run in lock-step with simulation time
  - Each transaction has 4 timing points (extensible)
Interoperability Layer

1. Core interfaces and sockets

Initiator

Target

2. Generic payload

Command
Address
Data
Byte enables
Response status

Extensions

3. Base protocol

BEGIN_REQ
END_REQ
BEGIN_RESP
END_RESP

Maximal interoperability for memory-mapped bus models
struct Initiator: sc_module
{
    tlm_utils::simple_initiator_socket<Initiator> socket;

    SC_CTOR(Initiator) : socket("socket")
    {
        SC_THREAD(thread_process);
    }
...
};

struct Target: sc_module
{
    tlm_utils::simple_target_socket<Target> socket;

    SC_CTOR(Target) : socket("socket")
    {
        socket.register_b_transport(this, &Target::b_transport);
    }

    virtual void b_transport(tlm::tlm_generic_payload& trans, sc_time& delay);
};
Example – Socket Binding

```c
struct Top: sc_module
{
    Initiator *initiator;
    Target   *target;

    SC_CTOR(Top)
    {
        initiator = new Initiator ("initiator");
        target   = new Target ("target");

        initiator->socket.bind( target->socket );
    }
};
```
Example - Initiator

void thread_process() {
    tlm::tlm_generic_payload* trans;
    sc_time delay;

    ...

    trans = m_mm.allocate();
    trans->acquire();

    trans->set_command( tlm::TLM_WRITE_COMMAND );
    trans->set_data_length( 4 );
    trans->set_streaming_width( 4 );
    trans->set_byte_enable_ptr( 0 );
    trans->set_address( addr );
    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->get_response_status() <= 0 )
        SC_REPORT_ERROR("TLM-2", trans->get_response_string().c_str());

    trans->release();
}
virtual void b_transport(
    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 ) {
        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 );
}
Initiators, Targets and Interconnect

Initiator socket → Forward path → Target socket

Forward path

Backward path

Initiator socket → Forward path → Target socket

Backward path

Interconnect component

0, 1 or many

Target

Transaction object

• Single transaction object

Copyright © 2007-2009 by Open SystemC Initiative and Doulos. All rights reserved.
Core Interfaces

- Sockets group interfaces, bind both paths with one call, and are strongly typed.
TLM Introduction

CONTENTS

• What is TLM and SystemC?
• Creating a virtual platform model
• The OSCI TLM-2.0 standard
• Achieving speed and interoperability
Interoperability versus Internals

**Initiator**
- Core interfaces
- Sockets
- Generic payload
- Base protocol

**Target**
- Interoperability layer

**Coding Style**
- Loosely- or Approximately-timed

**Utilities**
- Convenience sockets
- Quantum keeper (LT)
- Payload event queues (AT)
- Instance-specific extensions (GP)
Loosely-timed

Initiators

Process scheduling and context switching

Ticks

Cycle-accurate simulation

Quantum

Temporally decoupled simulation

Quantum

• Each initiator runs ahead to quantum boundary before context switching
• Limited synchronization between initiators
• Used with DMI to bypass interconnect

Blocking transport
Quantum
DMI

Copyright © 2007-2009 by Doulos. All rights reserved.
Transactions are annotated with delays

Each process is synchronized using the SystemC scheduler

A transaction has multiple phases
b/nb Conversion

**Models should be performance-matched**
The Generic Payload

• Typical attributes of memory-mapped busses
  • reads, writes, byte enables, single word transfers, burst transfers, streaming

• Off-the-shelf general purpose payload
  • used with the Base Protocol for abstract bus modeling
  • *ignoreable* extensions allow full interoperability

• Used to model specific bus protocols
  • mandatory extensions
  • can only bind sockets with same protocol type (compile-time check)
  • use the same generic payload machinery
  • low implementation cost when bridging protocols
Kinds of Extension

• Generic payload extensions can be
  • Ignorable – compliant to the base protocol
  • Mandatory – necessitates a new protocol type
  • Private – only used by a single module (hence ignorable)
  • Instance-specific – usually private (hard to access elsewhere)
  • Sticky – remain when transaction is returned to a pool
  • Auto – freed when transaction is returned to a pool

• Base protocol phases BEGIN_REQ, END_REQ, BEGIN_RESP, END_RESP

• Extended phases
Kinds of Interoperability

- Base protocol, generic payload + ignorable extensions
- Functional incompatibilities still possible

- New protocol, generic payload + extensions
- Cannot bind sockets of differing protocols
- Generic payload and base protocol still exploited for consistency of coding style
- Generic payload extension mechanism exploited for ease-of-adaption
Levels of Use

1. Buy models with the TLM-2.0 sticker
2. Write LT components
   - Beware: nb_transport & endianness
3. Write AT components
4. Support LT/AT switching