Introduction to TLM Mechanics
Presented by David C Black
Topics

• Simulation Use Cases
• Basic Concepts
• **Generic Payload** – Information
• **API** - The Transaction Call
• **Sockets** – Connectivity
• **Loosely Timed** – Fast
• **Approximately Timed** – Accurate
• **Base Protocol** – modeling abstraction
Simulation Use Cases

Architectural Analysis
• Do interconnections meet needs?
• Requires quick modeling and 1st order accuracy

Hardware Verification
• Early development
• Improve schedule and ensure sufficient coverage
• Requires some accuracy

Software Development
• Early development
• Requires speed

Software Performance Analysis
• Analyze code effects on timing & power
• Requires some accuracy
Fundamental Ideas

- TLM 2.0 models Memory-Mapped Bus-Transport
  - Exchange information (payload) between **initiator** and **target**
Four Basic TLM 2.0 Concepts

1. Core interfaces (API)
   my_skt->b_transport(gp, ...)

2. Generic payload
   Command
   Address
   Data
   Byte enables
   Response status

3. Sockets & Connectivity

4. Base protocol
   BEGIN_REQ
   END_REQ
   BEGIN_RESP
   END_RESP

© 2009 XtremeEDA USA Corporation - Version 090201.10
Many Configurations Considered

- MPU
- I/O

- CPU
- Interconnect
- Mem
- I/O

- CPU
- Mem
- Processor Bus
- Peripheral Bus
- Mem
- Periph
- Arbiter

- DMA

© 2009 XtremeEDA USA Corporation - Version 090201.10
Core Interface - The Transaction Call

• Generally one of two simple transport calls
  1. `void b_transport(TRANS& payload, sc_time& t);`
  2. `tlm_sync_enum nb_transport_fw(TRANS& payload, PHASE& ph, sc_time& t);`

• Declared beneath interface classes (enables port/socket)
  – `tlm::tlm_fw_transport_if<>`
  – `tlm::tlm_bw_transport_if<>`

• Implemented as methods (member function) of models
  – TLM models are disguised SystemC channels
Generic Payload — Information

• What is the Payload?
  – Simple structured object that contains:
    • Control (command)
    • Data
    • Various Properties & Extensions
  – Designed for memory-mapped buses
    • Burst size
    • Byte lanes
  – Defined as a class for flexibility

• Payload is passed by reference
  – Speeds up calling by reducing copy overhead
  – Need to be cognizant of lifetime
A Simple Transport Call

```cpp
sc_time tDelay;

const tlm::tlm_generic_payload payload;

const sc_core::uint32 addr(55);

payload.set_address(addr);
payload.set_data_ptr(&data);
payload.set_write(); // command

const sc_core::uint32 tDelay = SC_ZERO_TIME;

// Execute a transaction
b_transport(payload, tDelay);
wait(tDelay);
```
TLM 2.0 Sockets – more than convenient

- Transactions include communications in both directions
  - Initiator to target
  - Target to initiator
- Normally requires port/export and export/port pair (i.e. a lot of binding)
- Replaced with specialized port (simplifies binding)
TLM 2.0 Socket Graphically

Initiator.mdl

init_process

init_skt

init_skt->fw(…)

Backward methods

Target.mdl

targ_skt

targ_skt->bw(…)

Forward methods

targ_process
**Example: Initiator declaration**

```cpp
class Initiator_mdl
: public sc_module, public tlm::tlm_bw_transport_if
{
public:
    tlm::tlm_initiator_socket<> init_skt;

    SC_CTOR(Initiator_mdl);  // Constructor
    ~Initiator_mdl();        // Destructor
    void init_process();     // forward calls made here

    // Backwards flowing API for initiator socket
    // Must implement ALL backward calls (even if trivially)
    tlm::tlm_sync_enum nb_transport_bw( ... );
    void invalidate_direct_mem_ptr( ... );
};
```
Example: Target declaration

class Target_mdl
: public sc_module, public tlm::tlm_fw_transport_if<>
{
public:

  tlm::tlm_target_socket<> targ_skt;

  SC_CTOR(Target_mdl);        // Constructor
  ~Target_mdl();              // Destructor

  // Forwards flowing API for target socket
  // Must implement ALL forward calls (even if trivially)
  void b_transport(...);
  tlm::tlm_sync_enum nb_transport_fw(...);
  bool get_direct_mem_ptr(...);
  unsigned int transport_dbg(...);
};
Example: Top-level connectivity

```cpp
class Top_mdl
 : public sc_module
{
public:
  Initiator *init_ptr;
  Target    *targ_ptr;

  SC_CTOR(Top_mdl) { // Constructor (initialization)
    // Elaboration (create & connect hardware)
    init_ptr = new Initiator_mdl("init");
    targ_ptr = new Target_mdl("targ");
    init_ptr->init_skt.bind( targ_ptr->targ_skt );

  }

  ~Top_mdl(); // Destructor
};
```
Model Coding Styles

• TLM 2.0 introduces two terms
  – **Loosely Timed (LT) modeling** – "fast"
  – **Approximately Timed (AT) modeling** – "accurate"

• Terminology that roughly defines
  – Simulation speed
  – Timing accuracy
  – Model applicability

• Older terminology
  – UnTimed (UT), Programmer's View (PV), Programmer's View with Timing (PVT), Cycle Accurate
  – Too specific and often missed the point
Loosely Timed (LT) - fast

• Use Case
  – Early Software Development
  – Hardware Verification of portions of functionality
  – Speed up portions of other use cases prior to analysis

• Characteristics
  – Just enough detail to boot O/S and run multi-core systems
  – Processes can run ahead of simulation time (temporal decoupling)
  – Transactions have two timing points: *begin* and *end*
  – May be easier to code due to less detail

```c
template<
class PAYLD = tlm_generic_payload>
class tlm_blocking_transport_if : sc_interface {
  virtual void b_transport( PAYLD& , sc_time& )=0;
};
```
Loosely Timed Mechanisms

• Impediments to speed
  – Context switching
    • wait();
    • next_trigger(); return;
    • Doesn't matter whether SC_THREAD or SC_METHOD
  – Complex bus protocols and lots of processes
• Reduce context switching to improve simulation speed
  – Temporal decoupling
    • When time doesn't matter as much
    • Keep track of "local" time
  – Direct Memory Interface
    • Bypass bus for some accesses to gain performance
Initiator is blocked until return from b_transport

Not optimal
Blocking Transport/LT Modeling (2 of 2)

Fewer context switches = faster simulation
Approximately Timed – Accurate

• Use cases
  – Architectural Analysis, Software Performance Analysis
  – Hardware Verification

• Characteristics
  – Sufficient for architectural and performance exploration
  – Processes run in lock-step with simulation time
  – Transactions have four timing points (extensible)

```cpp
#include <tlm.h>

template<typename PAYLD = tlm_generic_payload,
          typename PHASE = tlm_phase>
class tlm_fw_nonblocking_transport_if
  : public sc_interface {
  virtual
    tlm_sync_enum
    nb_transport_fw(PAYLD&, PHASE&, sc_time&) = 0;
};
```
Approximately Timed Mechanisms

• Concerns – Providing sufficient accuracy
  – Negotiation time
  – Command (read/write) execution time
  – Response time
  – Simulation time fidelity
  – Bus interactions

• Techniques
  – Base Protocol with four timing points
  – Non-blocking calls (slight coding complexity)
  – Payload Event Queue to model transport delay
Non-Blocking/AT Modeling continued

- Also has backward initiated component

```cpp
template<typename PAYLD = tlm_generic_payload, 
          typename PHASE = tlm_phase>
class tlm_bw_nonblocking_transport_if : sc_interface {
    virtual
        tlm_sync_enum nb_transport_bw(PAYLD&, PHASE&, sc_time&) = 0;
};
```

Initiator

Forward path

Target

Backward path
Four Timing Points = Four Phases

- **BEGIN_REQ** (Begin Request)
  - Initiator acquires bus
  - Connections becomes "busy" and blocks further requests
  - Payload becomes "busy"

- **END_REQ**
  - Target "accepts" request and completes the handshake
  - Bus freed to start additional requests

- **BEGIN_RESP**
  - Target acquires bus to provide a response
  - Bus becomes "busy"

- **END_RESP**
  - Initiator acknowledges response to complete it
  - Bus freed
  - Payload reference freed up
Non-Blocking Return Type

- **TLM_ACCEPTED**
  - Transaction, phase and timing arguments unmodified (ignored)
  - Target may respond later (depending on protocol)

- **TLM_UPDATED**
  - Transaction, phase and timing arguments updated (used)
  - Target has advanced the protocol state machine to the next state

- **TLM_COMPLETED**
  - Transaction, phase and timing arguments updated (used)
  - Target has advanced the protocol state machine to the final phase

```c
enum tlm_sync_enum
{
    TLM_ACCEPTED, TLM_UPDATED, TLM_COMPLETED
};
```
Base Protocol and AT

- TLM 2.0 defines "Base Protocol"
  - High-level concept modeling time progression
  - Maps to many bus protocols in a rough manner
  - A single TLM transaction may cross one or more busses
    - Possibly each with a different hardware protocol

- Following slides illustrate some of the TLM possibilities
  - 4 distinct phases using the backward path
  - 2 distinct phases using the return path
  - 1 combined phase with early return completion
  - Use of timing annotation possible as well
AT Using the Backward Path (4 phases/calls)

Initiator

Call

nb_transport_fw(gp, BEGIN_REQ, 0ns)

TLM_ACCEPTED, -, -, 0ns

Return

TLM_ACCEPTED, -, -, 0ns

Return

Call

nb_transport_bw(gp, END_RESP, 0ns)

Return

TLM_COMPLETED, -, -, 0ns

Call

nb_transport_bw(gp, BEGIN_RESP, 0ns)

Return

TLM_ACCEPTED, gp, -, 0ns

Call

nb_transport_fw(gp, END_RESP, 0ns)

Return

TLM_COMPLETED, -, -, 0ns

Bus busy

Bus busy


\[ t_{\text{sim}} = 200\text{ns} \]

\[ t_{\text{sim}} = 220\text{ns} \]

\[ t_{\text{sim}} = 262\text{ns} \]

\[ t_{\text{sim}} = 270\text{ns} \]
AT Using the Return Path
(2 phases/calls)

Initiator

Call

nb_transport_fw(gp, BEGIN_REQ, 0ns)

TLM_UPDATED, -, END_REQ, 20ns

Return

Target

Bus busy

Bus busy

Initiator

Call

wait(20ns)

Return

TLM_UPDATED, -, END_REQ, 20ns

Initiator

Call

wait(20ns)

Return

TLM_UPDATED, -, END_REQ, 20ns

Initiator

Call

wait(8ns)

Return

TLM_COMPLETED, -, END_RESP, 8ns

Initiator

Call

wait(20ns)

Return

TLM_COMPLETED, -, END_RESP, 8ns

Initiator

Call

wait(8ns)

Return

TLM_COMPLETED, -, END_RESP, 8ns
AT with Early Return Completions (1 call)

Call \texttt{nb_transport_fw}(gp, \text{BEGIN\_REQ}, 0\text{ns})

Return \texttt{TLM\_COMPLETED}, -, \texttt{END\_RESP}, 70\text{ns}

$t_{\text{sim}} = 200\text{ns}$

$t_{\text{local}} = 270\text{ns}$

$\text{wait}(70\text{ns})$

$t_{\text{sim}} = 270\text{ns}$

$t_{\text{local}} = 0\text{ns}$
TLM Mechanics Summary

- **Targets** and **Initiators**

- **API Transports** Information
  - Two types: blocking & non-blocking

- **Sockets** provide interconnect

- Base Protocol Supports Two Major Coding Styles
  - **Loosely Timed** (LT) for Speed (Coding & Run-Time)
  - **Approximately Timed** (AT) for Accuracy and Analysis
The end...
While we wait for the next speaker...

- "Furious activity is no substitute for understanding." -- H. H. Williams

- "Ron's first law: All extreme positions are wrong." -- Ron Garret

- "C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole leg." -- Bjarne Stroustrup

- "Any philosophy that can be put in a nutshell belongs there." -- Sydney J. Harris
While we wait for the next speaker...

• "Some people, when confronted with a [programming] problem, think, 'I know, I'll use regular expressions.' Now they have two problems." -- Jamie Zawinski

• "Arguing that Java is better than C++ is like arguing that grasshoppers taste better than tree bark." -- Thant Tessman

• "If you think C++ is not overly complicated, just what is a protected abstract virtual base pure virtual private destructor and when was the last time you needed one?" -- Tom Cargill
"Within C++, there is a much smaller and cleaner language struggling to get out." -- Bjarne Stroustrup

"It has been discovered that C++ provides a remarkable facility for concealing the trivial details of a program -- such as where its bugs are." -- David Keppel

"When asked which (programming language) I prefer, I usually answer that I was asked the wrong question. The right question should be, 'Which one do I hate the least?' And the answer to that question is, 'The one I'm not currently working with.'" -- Janick Bergeron