1. SystemC-TLM

Transaction Level Modeling
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
An example
Simulation Use Cases

**Architectural Analysis**
- Do interconnections meet needs?
- Requires quick modeling and 1\textsuperscript{st} order accuracy

**Software Development**
- Early development
- Requires speed

**Hardware Development**
- Early development
- Improve schedule and ensure sufficient coverage
- Requires some accuracy

**Hardware Verification**
- Early reference
- (Golden) Model
## Virtual Platform Characteristics

<table>
<thead>
<tr>
<th>Instruction Set Simulator or software stubs</th>
<th>Transaction-Level Model</th>
<th>RTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available early</td>
<td>✓ Available early</td>
<td>✓</td>
</tr>
<tr>
<td>Fast enough to run applications</td>
<td>✓ Fast enough to run applications</td>
<td>✓ Too slow to run applications</td>
</tr>
<tr>
<td>Little or no hardware detail</td>
<td>✓ Register-accurate</td>
<td>✓ Register-accurate and pin-accurate</td>
</tr>
<tr>
<td>No timing information</td>
<td>✓ Some timing information</td>
<td>✓ Cycle-accurate timing</td>
</tr>
</tbody>
</table>

- ✓: Yes
- ✗: No
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
   my_skt->b_transport(gp, ...)

2. Sockets & Connectivity

2. Generic payload
   Command
   Address
   Data
   Byte enables
   Response status
   Extensions

4. Base protocol
   BEGIN_REQ
   END_REQ
   BEGIN_RESP
   END_RESP
Many Configurations Considered

Diagram showing various configurations involving MPU, CPU, Interconnect, Mem, I/O, Processor bus, DMA, Peripheral bus, Mem, Periph, and Arbiter.
Core Interface – The Transaction Call

- Generally one of two simple transport calls
  1. `void b_transport(TRANS& payload, sc_time& t);`
  2. `void 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 calling by reducing copy overhead
  • Need to be cognizant of lifetime
Simple Transport Call

```cpp
sc_time tDelay;
//sc_core::time_step(addr(55));
tlm::tlm_generic_payload payload;
sc_core::uint32 addr(55);

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

// Execute a transaction
b_transport(payload, tDelay);
wait(tDelay);
```
TLM 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->fw()
- Backward methods

Target_mdl

- Foreward methods
- init_skt->bw()
- target_process

Backward methods

Forward methods
Example: Initiator declaration

class Initiator_mdl
: public sc_module, public tlm_bw_transport_if<<
{
public:

tlm::tlm_initiator_socket<> init_socket;
SC_CTOR(Initiator_mdl);  // Constructor
~Initiator_mdl();        // Destructor
void init_process();     // forward call 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_fw_transport_if
{
public:
    tlm::tlm_target_socket<> targ_skt;
    SC_CTOR(Target_mdl); // Constructor
    ~Target_mdl(); // Destructor

    // Forwards flowing API for initiator socket
    // Must implement all forward calls (even if trivially)
    void b_transport ( ... );
    tlm::tlm_sync_enum nb_transport_fw( ... );
    void get_direct_mem_ptr ( ... );
    void transport_dbg ( ... );
};

Example: Top-level connectivity

class Top_mdl : public sc_module
{
public:
    Initiator_mdl *init_ptr;
    Target_mdl  *targ_ptr;
    SC_CTOR(Top_mdl) {
        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), Programmers View (PV),
    Programmers View with Timing (PVT), Cycle Accurate
  • Too specific and often miss 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 multicore-systems
  - Processes can run ahead of simulation time (temporal decoupling)
  - Transactions have two timing points: `begin` and `end`
  - Maybe easier to code due to less detail

```cpp
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 purposes to gain performance
Blocking Transport/ LT Modeling (1/2)

Initiator

\[ t_{sim} = 100\text{ns} \]

Call

\[ b\_transport(gp, 0\text{ns}) \]

Initiator is blocked until return from \( b\_transport \)

Target

\[ \text{wait}(40\text{ns}) \]

Not Optimal
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

```cpp
template<typename PAYLD  = tlm_generic_payload, 
         typename PHASE = tlm_phase >
class tlm_nonblocking_transport_if : sc_interface {
   virtual void 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

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

![Diagram showing Initiator and Target with Forward and Backward paths]
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) on return
  - Target may respond later (depending on protocol)

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

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

```cpp
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
Notation for Message Sequence Charts

Status = nb_transport (trans, phase, delay);

Simulation time = 5us

Local time
+10ns
+20ns

= sc_time_stamp()

For temporal decoupling, local time is added to simulation time (explained on slides)

Call

Return

TLM_COMPLETED, BEGIN_RESP, 10ns

Arguments passed to function

Values returned from function
Using the Backward Path

**Phase**

BEGIN_REQ

**Initiator**

Simulation time = 100ns

Call -, BEGIN_REQ, 0ns

Return

Target

TLM_ACCEPTED, -, -

**Phase**

BEGIN_REQ

Simulation time = 110ns

-, END_REQ, 0ns

Call

Simulation time = 120ns

TLM_ACCEPTED, -, -

Return

BEGIN_RESP

Simulation time = 130ns

-, BEGIN_RESP, 0ns

Call

-, END_RESP, 0ns

Return

Transaction accepted now, caller asked to wait

SystemC-TLM
Using the Return Path

Callee annotates delay to next transition, caller waits
Early Completion

BEGIN_REQ

Simulation time = 100ns

Call

TLM_COMPLETED, -, 10ns

Return

Simulation time = 110ns

END_RESP

Callee annotates delay to next transition, caller waits
Timing Annotation

Phase

Initiator

Simulation time = 100ns

Call

-, BEGIN_REQ, 10ns

TLM_ACCEPTED, -, -

Return

Simulation time = 110ns

Payload

Event Queue

Simulation time = 125ns

-, END_REQ, 10ns

Call

Return

TLM_ACCEPTED, -, -

Simulation time = 135ns

Target

Payload Event Queue

BEGIN_REQ

END_REQ
Summary: Key Features of TLM-2

- Transport interfaces with timing annotation and phases
- DMI and debug interfaces
- Loosely-timed coding style and temporal decoupling for simulation speed
- Approximately-timed coding style for timing accuracy
- Sockets for convenience and strong connection checking
- Generic payload for memory-mapped bus modelling
- Base protocol for interoperability between TL- models
- Extensions for flexibility of modelling