Improving Timing Accuracy for TLM-LT Models

Advanced Temporal Decoupling (ATD)

European SystemC Users’ Group Workshop
2013/03/22 Grenoble

Simon Hufnagel, Hendrik Post, Nico Bannow
Robert Bosch GmbH
Corporate Sector
Research and Advance Engineering

Christoph Grimm
Technical University Kaiserslautern
Content

- Context
- Transaction Level Modeling
- Advanced Temporal Decoupling (ATD)
- Case Study “Night Vision“
- Conclusion
Seamless Modeling

Advanced Temporal Decoupling

Context

Compilation

Results

Implementation

Analysis

Modeling

Optimization

Refinement, Evaluation, Test

Simulation

Environment

C-Code Translation

Algorithm

MATLAB SIMULINK

Idea

ATD Modeling Method

Final System
Context

Central Question

„How to efficiently create accurate and performing virtual prototypes?”

Transparent TLM*) Advanced Temporal Decoupling (ATD)

*) see 26th ESCUG Workshop Dresden 2012:
“Using IP-XACT to ease system development with SystemC TLM The Transparent TLM (TTLM) Approach”
Simon Hufnagel, Bosch, DE,
Christoph Grimm, Vienna University of Technology, AT
Context

Abstraction Level Overview

**Functional Level**
- no hardware aspects (architecture, timing)
- communication based on shared variables and function calls

**Transaction Level**
- basic hardware structure
- transaction based communication
- timing accuracy from untimed to cycle accurate

**Register Transfer Level**
- synthesizable hardware specification
- signal based communication
- clock cycle accurate timing
Content

- Context
- Transaction Level Modeling
- Advanced Temporal Decoupling (ATD)
- Case Study “Night Vision”
- Conclusion
Transaction Level Modeling

Transaction Level Modeling Variants

Data Granularity

- bus word
- bus packet
- application packet

Timing Accuracy

- untimed
- approximately timed
- cycle accurate

TLM-PV: programmer's view
TLM-PVT: programmer's view with timing
TLM-CA: cycle-accurate

figure according to: “Transaction Level Modeling – An Abstraction Beyond RTL”, L. Maillet-Contoz and F. Ghenassia, Springer, 2005
Transaction Level Modeling

Transaction Level Modeling Techniques*)

→ timing granularity varies with the number of function calls per transaction

loosely timed

single function call

approximately timed

multiple function calls

Temporal Decoupling*)

- global simulation time is extended by local simulation time offset
- communication time consumption is aggregated

Interfered bus accesses by multiple masters

- 3 word **low priority** transfer from Master 1 to Slave 1 starting @ 100ns
- 3 word **high priority** transfer from Master 2 to Slave 2 starting @ 200ns

**Expected timing behavior:**

- **Low priority** transfer from Master 1 to Slave 1 starts at 100ns and continues for 50ns.
- **High priority** transfer from Master 2 to Slave 2 starts at 200ns and ends at 400ns.

**Diagram:**

- **Thread 1**
  - Low priority transfer starts at 100ns and ends at 150ns.
  - Continuation of low priority transfer from 150ns to 300ns.

- **Thread 2**
  - High priority transfer starts at 200ns and ends at 400ns.
  - Interruption of low priority transfer from 300ns to 400ns.

**Notes:**

- **Shared Resource Access (SRA)**
successive single word transfers
- Masters send one single word TLM transaction per clock cycle
- Bus arbitrates masters within each clock cycle

Exhaustive timing accuracy
- Low simulation performance due to tiny simulation step width
One burst transfer per master

- Masters initiate **one** application packet transfer at designated start times
- Typically, transactions are executed immediately within master’s context

High simulation speed due to lowered simulation time granularity

Reduced simulation accuracy due to disregarded resource contention
Further simulation speed improvement due to reduced context switch count

In order execution is not assured automatically & timing errors are accumulated
Related Work

Handling Speed vs. Accuracy Trade-Off

- Global Quantum / Quantum Keeper¹)
  - used to narrow local offset compared to global simulation time
  - large quantum $\rightarrow$ high simulation speed, but low timing accuracy
  - disregards functional dependencies $\rightarrow$ synchronization is fully obliged to the designer
  - can not guarantee absence of timing errors even for low quantum values

- Trace driven simulation acceleration
  - trace extraction from high level system representation²)
  - trace extraction from lower level cycle accurate simulation³)

Related Work

Improving Timing Accuracy

Statistical approaches\(^1\),\(^2\):
- resource contention effects are approximated using statistical methods or finer grained previous simulations
- cycle accuracy can not be achieved

Postponed timing correction in case of resource conflicts:

\[ \text{Quantum Giver}^{3), \ TLM+ \ Resource Model}^{4), \ Result Oriented Modeling}^{5) \]

- resource contention is detected and simulation time of the corresponding thread is updated accordingly
- transactions are executed with non-preemptable semantic (either at the designated start time or end time of the transaction)

Related Work

Transaction Level Modeling Trade-Offs

Data Granularity

- bus word
- bus packet
- application packet

Timing Accuracy

- untimed
- approximately timed
- cycle accurate

Simulation Slowdown

TLM

TLM-CA

RTL

TLM-PVT

Fixed Data Granularity and cooperative multitasking limits Timing Accuracy

figure according to: “Transaction Level Modeling – An Abstraction Beyond RTL”, L. Maillet-Contoz and F. Ghenassia, Springer, 2005
Advanced Temporal Decoupling

Solution: Advanced Temporal Decoupling

- Allows **variable data granularity** to achieve cycle accuracy
- Based on **Temporal Decoupling** for high simulation speed

use look-a-head to determine potential future preemptions

- Resources contain **Temporal Decoupled Semaphores** (TDSem) managing resource accesses

![Diagram](image)

Temporal Decoupled Semaphores (TDSem)
Content

- Context
- Transaction Level Modeling
- Advanced Temporal Decoupling (ATD)
- Case Study “Night Vision“
- Conclusion
Advanced Temporal Decoupling – Concept

ATD simulation course: two alternating phases

- Shared Resource Access (SRA) creation phase
  - execute runnable simulation threads in temporal decoupled manner
  - threads register SRAs at corresponding TDSems **without processing** them
  - suspend threads at predefined **synchronization points**

![Diagram](image_url)
## Advanced Temporal Decoupling – Concept

### Synchronization Points

Temporal decoupling has to stop if data dependencies would be violated

<table>
<thead>
<tr>
<th>Synchronization Point</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read access from external resource</td>
<td></td>
</tr>
<tr>
<td>Read access from local volatile variable</td>
<td></td>
</tr>
<tr>
<td>Access to simulation time</td>
<td>In case there are pending resource accesses</td>
</tr>
<tr>
<td>Notification of and wait for user events</td>
<td>In case there are pending resource accesses</td>
</tr>
</tbody>
</table>

remark: Synchronization Points are integrated into TTLM library components

- instructions not necessarily comprising data dependency
  - write access to external resource considering *error type conversion*)
  - write access to local volatile variable
  - occurrence of interrupts

Advanced Temporal Decoupling – Concept

Shared Resource Access execution phase

1. check if there are pending SRAs within “sc_time_to_pending_activity()”
2. select next SRA
3. calculate time budget
Advanced Temporal Decoupling – Concept

Shared Resource Access execution phase (cont’d)

4. (partially) execute next SRA considering time budget
5. delay related and/or conflicting SRAs and synchronization points
Advanced Temporal Decoupling – Concept

Shared Resource Access execution phase (cont’d)

6. repeat steps 1 to 5 until there are no pending SRAs within “sc_time_to_pending_activity()”

7. advance global simulation time and continue with next SRA creation phase

incorporation of user defined scheduler  delay of conflicting low priority SRA
Advanced Temporal Decoupling

Simple TLM example performance evaluation

- Speedup $\geq 2.5x$ (larger transaction size $\rightarrow$ larger speedup)
- Simulation accuracy equals cycle accurate model

![Transactions / s](chart.png)

<table>
<thead>
<tr>
<th></th>
<th>Transactions / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLM-CA</td>
<td>332k</td>
</tr>
<tr>
<td>ATD sync after each application packet</td>
<td>835k</td>
</tr>
<tr>
<td>ATD sync after 10 application packets</td>
<td>940k</td>
</tr>
</tbody>
</table>

Thread 1: 2 words
Thread 2: 3 words
Advanced Temporal Decoupling

Features

- synchronous and asynchronous resource accesses

- preemptable and non-preemptable resource accesses

- Preemptable low priority resource access is interrupted by high priority access

- Non-preemptable low priority resource access defers execution of high priority access
Advanced Temporal Decoupling

Features (cont’d)

- supported resource properties:
  - interacting resources
  - multiport resources
  - dynamic access priorities

- user specified scheduler for simultaneous resource accesses

- generation of profiling information
  - resource utilization ratio
  - minimum interaction matrix
  - ...
Advanced Temporal Decoupling – Implementation

SRA Management

⇒ Facts

- each SRA has exactly one source (thread / superior SRA)
- each thread / SRA might have an arbitrary amount of pending SRAs
- SRA registration might happen out-of-order (temporal decoupling)
- each SRA is associated to exactly one resource
- actual SRA execution might differ from model semantics
- related SRAs influence each other

⇒ Implementation:

- SRA Tree for inter-SRA-dependencies
- SRA Matrix for resource conflict detection
Advanced Temporal Decoupling – Implementation

SRA Tree for inter-SRA dependencies

- One SRA Tree per thread
- Arbitrary amount of pending SRAs per thread / parent SRA

![Diagram showing SRA Tree structure with levels and relationships between threads and SRAs.]

- Thread = root
- Simulated time

completed SRAs
async SRA
sync SRA
pending SRAs
active SRA

# SRA Matrix for resource contention detection

The SRA matrix for resource contention detection is a dynamic matrix that allows for the identification of resource contention among threads. The matrix is structured as follows:

<table>
<thead>
<tr>
<th>TDSem</th>
<th>Th0</th>
<th>Th1</th>
<th>Th2</th>
<th>...</th>
<th>Th_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDSem_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDSem_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDSem_j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the matrix, each row represents a thread (Th), and each column represents a semaphore (TDSem). The SRA (Synchronization and Resource Allocation) values are marked at the intersections of threads and semaphores, indicating when a thread is waiting for a semaphore to become available.

Example:

- SRA_{TDSem_j} indicates the SRA value for thread th_i with semaphore TDSem_j.
- SRA_{TDSem_i} indicates the SRA value for thread th_i with semaphore TDSem_i.

This matrix helps in identifying contention points and managing resources efficiently in a concurrent system.
ATD provides dynamic Data Granularity

Advanced Temporal Decoupling

figure according to: “Transaction Level Modeling – An Abstraction Beyond RTL”, L. Maillet-Contoz and F. Ghenassia, Springer, 2005
Advanced Temporal Decoupling

ATD Usage

- Initiators remain unchanged (exception: synchronization points)
- Integrate TDSem into resources ... having multiple target ports ... allowing asynchronous accesses
- TDSem not needed for cascaded single port resources allowing synchronous accesses only
- Pass transaction to TDSem within TLM \((n)\) b_transport call
- Implement SRAExecution_if which gets called to process the transaction
Content

 ➔ Context
 ➔ Transaction Level Modeling
 ➔ Advanced Temporal Decoupling (ATD)
 ➔ Case Study “Night Vision“
 ➔ Conclusion
Case Study “Night Vision“

Night Vision

- automotive driver assistance application
- improves obstacle visibility at night
- based on advanced video processing algorithms
Case Study “Night Vision“

Model properties of existing virtual prototype

➤ complete representation of Night Vision video computation system (HW architecture, video algorithms, …)
➤ cycle accurate implementation
➤ runs original Night Vision video algorithms
➤ based on SystemC with “Module-Adapter“ extension*)

*) see “Modeling and Optimization of a Multiprocessor Control Unit using SystemC”, PhD thesis N. Bannow, 2009
Case Study “Night Vision“

Model structure and dataflow

1. frame transfer from camera
2. DMA based transfer to uC buffer
3. frame processing by NV algorithms
4. DMA based transfer back to FPGA buffer
5. output on display

Source: „Modeling and Optimization of a Multiprocessor Control Unit using SystemC“, PhD thesis N. Bannow, 2009
Case Study “Night Vision”

Night Vision evaluation results

- simulation runtime improvement: ~5x
- delta cycle count reduction: ~20x
- simulation accuracy equals cycle accurate model

- further improvements possible, if abstracted software model is used
Content

⇒ Context
⇒ Transaction Level Modeling
⇒ Advanced Temporal Decoupling (ATD)
⇒ Case Study “Night Vision”
⇒ Conclusion
Summary

Advanced Temporal Decoupling ...

- ... enables cycle accuracy in context of temporal decoupling
  - automatic execution order preserving management of resource contention effects
  - time budget calculation for dynamic data granularity
  - simple TLM-LT like user API

- ... improves simulation performance of TLM-CA models
  - speed up ~5x in industrial simulation model
  - TLM-CA simulation accuracy is retained

- ... can be used in conjunction with TTLM methodology
  - model structure: TTLM library + TTLM generator
  - simulation performance + accuracy: Advanced Temporal Decoupling
  - synchronization points integrated into TTLM library elements
Thanks for your attention!

Questions ??