Transaction Level Modeling: An Overview

Daniel Gajski
Lukai Cai

Center for Embedded Computer Systems
University of California, Irvine
www.cecs.uci.edu/~{gajski, lcai}
Acknowledgement

We would like to thank transaction level team at CECS that has contributed many ideas through numerous lunch discussions:

Samar Abdi
Rainer Doemer
Andreas Gerstlauer
Junyu Peng
Dongwan Shin
Haobo Yu
Overview

- Motivation for TLM
- TLM definition
- TLMs at different abstraction levels
- TLMs for different design domains
- SL methodology = model algebra
- Conclusion
Motivation

• SoC problems
  • Increasing complexity of systems-on-chip
  • Shorter times-to-market

• SoC solutions
  • Higher level of abstraction – transaction level modeling (TLM)
  • IP reuse
  • System standards

• TLM questions
  • What is TLM ?
  • How to use TLM ?

• This paper
  • TLM taxonomy
  • TLM usage
TLM Definition

• TLM = < {objects}, {compositions} >

• Objects
  • Computation objects + communication objects

• Composition
  • Computation objects read/write abstract (above pin-accurate) data types through communication objects

• Advantages
  • Object independence
    – Each object can be modeled independently
  • Abstraction independence
    – Different objects can be modeled at different abstraction levels
Abstraction Models

- Time granularity for communication/computation objects can be classified into 3 basic categories.
- Models B, C, D and E could be classified as TLMs.

A. "Specification model"
   "Untimed functional models"

B. "Component-assembly model"
   "Architecture model"
   "Timed functional model"

C. "Bus-arbitration model"
   "Transaction model"

D. "Bus-functional model"
   "Communicatin model"
   "Behavior level model"

E. "Cycle-accurate computation model"

F. "Implementation model"
   "Register transfer model"
A: “Specification Model”

Objects
- Computation
- Behaviors
- Communication
- Variables

Composition
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait

B1
\[ v1 = a^2; \]

B2
\[ v2 = v1 + b^2; \]

B3
\[ v3 = v1 - b^2; \]

B4
\[ v4 = v2 + v3; \]
\[ c = \text{sequ}(v4); \]
B: “Component-Assembly Model”

Objects
- Computation
  - Proc
  - IPs
  - Memories
- Communication
  - Variable channels

Composition
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait

Computation:
- Initial object
- Computation objects
- Communication objects
- Composition

Communication Flow:
- Cycle-timed
- Approximate-timed
- Un-timed

Objects:
- B1: v1 = a*a;
- B2: v2 = v1 + b*b;
- B3: v3 = v1 - b*b;
- B4: v4 = v2 + v3; c = sequ(v4);

Composition:
- PE1
- PE2
- PE3

Graphical Representation:
- Node A
- Node B
- Node C
- Node D
- Node E
- Node F
- Node G

Links:
- Computation
- Communication

Time Models:
- Un-timed
- Approximate-timed
- Cycle-timed

Synchronization:
- Notify/Wait
C: “Bus-Arbitration Model”

Objects
- Computation
  - Proc
  - IPs (Arbiters)
  - Memories
- Communication
  - Abstract bus channels

Composition
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait

Computation

Communication

Cycle-timed
Approximate-timed
Un-timed

Un-timed
Approximate-timed
Cycle-timed

Objects

PE1
B1
\[ v1 = a^3; \]

PE2
B2
\[ v2 = v1 + b^2; \]

PE4 (Arbiter)
B3
\[ v3 = v1 - b^2; \]

PE3
B4
\[ v4 = v2 + v3; c = \text{sequ}(v4); \]

v3 = v1 - b*b;
B3

v4 = v2 + v3;
c = \text{sequ}(v4);
B4

1. Master interface
2. Slave interface
3. Arbiter interface

1. PE1
2. PE2
3. PE3
4. PE4 (Arbiter)
D: “Bus-Functional Model”

**Objects**
- **Computation**
  - Proc
  - IPs (Arbiters)
  - Memories
- **Communication**
  - Protocol bus channels

**Communication**
- Cycle-timed
- Approximate-timed
- Untimed

**Computation**
- Cycle-timed
- Approximate-timed
- Untimed

**Composition**
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait
E: “Cycle-Accurate Computation Model”

Objects
- Computation
  - Proc
  - IPs (Arbiters)
  - Memories
  - Wrappers
- Communication
  - Abstract bus channels

Composition
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait

Communication

Computation

Objects - Computation
- Proc
- IPs (Arbiters)
- Memories
- Wrappers
- Communication
- Abstract bus channels

Composition
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait
**F: “Implementation Model”**

**Objects**
- Computation
  - Proc
- IPs (Arbiters)
- Memories
- Communication
  - Buses (wires)

**Composition**
- Hierarchy
- Order
  - Sequential
  - Parallel
  - Piped
  - States
- Transitions
  - TI, TOC
- Synchronization
  - Notify/Wait

---

**Objects**

- Computation
  - Proc
- IPs (Arbiters)
- Memories
- Communication
  - Buses (wires)
## Characteristics of Different Abstraction Models

<table>
<thead>
<tr>
<th>Models</th>
<th>Communication time</th>
<th>Computation time</th>
<th>Communication scheme</th>
<th>PE interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification model</td>
<td>no</td>
<td>no</td>
<td>variable</td>
<td>(no PE)</td>
</tr>
<tr>
<td>Component-assembly model</td>
<td>no</td>
<td>approximate</td>
<td>variable channel</td>
<td>abstract</td>
</tr>
<tr>
<td>Bus-arbitration model</td>
<td>approximate</td>
<td>approximate</td>
<td>abstract bus channel</td>
<td>abstract</td>
</tr>
<tr>
<td>Bus-functional model</td>
<td>time/cycle accurate</td>
<td>approximate</td>
<td>protocol bus channel</td>
<td>abstract</td>
</tr>
<tr>
<td>Cycle-accurate computation model</td>
<td>approximate</td>
<td>cycle-accurate</td>
<td>abstract bus channel</td>
<td>pin-accurate</td>
</tr>
<tr>
<td>Implementation model</td>
<td>cycle-accurate</td>
<td>cycle-accurate</td>
<td>bus (wire)</td>
<td>pin-accurate</td>
</tr>
</tbody>
</table>
Model Algebra

- Algebra = < {objects}, {operations} >  [ex: a * (b + c)]

- Model = < {objects}, {compositions} >  [ex: ]

- Transformation $t(model)$ is a change in objects or compositions.

- Model refinement is an ordered set of transformations, < $t_m$, ..., $t_2$, $t_1$>, such that $model\ B = t_m( \ldots ( t_2( t_1( model\ A ) ) ) \ldots )$

- Model algebra = < {models}, {refinements} >

- Methodology is a sequence of models and corresponding refinements
Model Definition

- Model = < {objects}, {composition rules} >
- Objects
  - Behaviors (representing tasks / computation / functions)
  - Channels (representing communication between behaviors)
- Composition rules
  - Sequential, parallel, pipelined, FSM
  - Behavior composition creates hierarchy
  - Behavior composition creates execution order
    - Relationship between behaviors in the context of the formalism
- Relations amongst behaviors and channels
  - Data transfer between channels
  - Interface between behaviors and channels
Model Transformations (Rearrange and Replace)

- **Rearrange object composition**
  - To distribute computation over components
- **Replace objects**
  - Import library components
- **Add / Remove synchronization**
  - To correctly transform a sequential composition to parallel and vice-versa
- **Decompose abstract data structures**
  - To implement data transaction over a bus
- **Other transformations**
  - 

\[ a \times (b + c) = a \times b + a \times c \]

Distributivity of multiplication over addition

analogous to…….

Distribution of behaviors (tasks) over components
Model Refinement

- **Definition**
  - Ordered set of transformations $< t_m, \ldots, t_2, t_1 >$ is a refinement
    - \( \text{model } B = t_m( \ldots ( t_2( t_1( \text{model } A ) ) ) \ldots ) \)

- **Derives a more detailed model from an abstract one**
  - Specific sequence for each model refinement
  - Not all sequences are relevant

- **Equivalence verification**
  - Each transformation maintains functional equivalence
  - The refinement is thus correct by construction

- **Refinement based system level methodology**
  - Methodology is a sequence of models and refinements
Verification

- Transformations preserve equivalence
  - Same partial order of tasks
  - Same input/output data for each task
  - Same partial order of data transactions
  - Same functionality in replacements
- All refined models will be “equivalent” to input model
  - Still need to verify first model using traditional techniques
  - Still need to verify equivalence of replacements
Synthesis

- Set of models
- Sets of design tasks
  - Profile
  - Explore
  - Select components / connections
  - Map behaviors / channels
  - Schedule behaviors/channels
- Each design decision => model transformation
- Detailing is a sequence of design decisions
- Refinement is a sequence of transformations
- Synthesis = detailing + refinement
- Challenge: define the sequence of design decisions and transformations
SCE Experiment is Very Positive

Source: http://www.cecs.uci.edu/~cad/sce.html
Conclusion

• Computation and communication objects of TLM are connected through abstract data types

• TLM enables modeling each component independently at different abstraction levels

• The major challenge is to define necessary and sufficient set of models for a design flow

• The next major challenge is to define model algebra and corresponding methodology for each application such that algorithms and tools for modeling, verification, exploration, synthesis and test can be easily developed

• **Opportunities are bigger than anything seen before**