• Why UML?
• Recapping SystemC
• Transaction Level Abstraction
• UML Wrapper for SystemC
• A Simple Bus Example
• Discussion

Agenda
UML for System Co-design
• Assembly coding
• "Hardware-centric" co-design tools
  - Synopsys's CoCentric tool
  - Cadence's Virtual Component Co-design tool (VCC)
  - "Hardware-centric" co-design tools
• Assembly coding

Classic Practices of SOC Design
Challenges Ahead

- Scalability of SOC design
- Significant amount of variable software and hardware
- Exploding complexity of SOC
Learning from the Software Engineering Experience

Tackling the complexity

• Requirement gathering
• System specification
• Framework/Interface design
• System design, pattern reuse
• Component design, Unit test
• System Integration, Integration test, End-to-end test

Learning from the Software Engineering Experience

UML for System Co-design

National University of Singapore
UML for System Co-design

- UML supports entire design process
- UML supports development processes across development life cycles
- UML supports application domains across application domains
- UML supports implementation languages across implementation languages
- UML provides unifications across historical methods and notations

Why UML?
UML for System Co-design

Static models
• Class diagram
• Object diagram
• Component diagram
• Deployment diagram
• Use cases

Behavioral models
• Statecharts
• Activity diagrams
  (collaboration diagrams)
• Interaction diagrams

Static models
Levels of Abstraction in UML

High-level functionality of the system and its interaction with the outside world: use case diagrams

Interactions between objects or between sub-systems: collaboration or sequence diagrams

Interactions within a class or its associated objects: state chart diagrams

Interactions within an operation (algorithm): activity diagrams

• High-level functionality of the system and its interaction with the outside world: use case diagrams
• Interactions between objects or between sub-systems: collaboration or sequence diagrams
• Interactions within a class or its associated objects: state chart diagrams
• Interactions within an operation (algorithm): activity diagrams
Positioning of SystemC

- Based on C++, Going beyond RTL, Transaction Level Model

**SystemC**

**CORE LANGUAGE**
- Modules; Ports; Processes;
- Events; Interfaces; Channels;
- Data Types
  - 4-valued logic type (01xz)
  - 4-valued logic vectors
  - Bits and Bit Vectors
  - Arbitrary precision integers
  - Fixed point numbers; C++ user defined types

**DATA TYPES**
- Arbitrary precision integers;
- 4-valued logic vectors;
- Bits and Bit Vectors;
- Fixed point numbers;
- C++ user defined types;
- Arbitrary precision integers;
- 4-valued logic vectors;
- Bits and Bit Vectors;
- Fixed point numbers;
- C++ user defined types.

**EVENT-DRIVEN SIMULATION KERNEL**
- Modules; Ports; Processes;
- Events; Interfaces; Channels;
- Data Types

**OTHER LIBRARIES**
- RPN, etc.
- FIFO, SIGNAL, MUTEX, TIMER, etc.
- Elementary Channels: FIFO, SIGNAL, MUTEX, TIMER, etc.

**C++ LANGUAGE STANDARD**
- SystemC
UML for System Co-design

- **METHOD** A C++ method
- **MODULE** A structural entity
- **INTERFACE** An interface
- Provides a set of method declarations

Fundamental of SystemC
UML for System Co-design

• CHANNEL A channel implements one or more interfaces, and serves as a container for communication functionality.

• PORT A port is an object through which a module can access a channel's interface.

• PRIMITIVE CHANNEL A primitive channel is atomic, that is, it does not contain processes.

Fundamental of SystemC
SystemC Processes

• The basic unit of functionality

  - Method process
    - Dynamic sensitivity: next(integer) events
    - When integer, executes its body from the beginning to the end

  - Thread process
    - Dynamic sensitivity: next(integer) events
    - Execution may be suspended by wait(integer) or any of its variants: dynamic sensitivity
    - When integer, executes its body from the beginning to the end
    - States of execution are kept implicitly
    - Resumes from the point of suspension

System Processes
UML for System Co-design

Sensitivity

• Static: attributes of classes representing system processes

• Dynamic:
  - Collaboration diagram for method processes
  - Sequence diagram for thread processes
  - To model interaction among processes, collaboration diagrams may be more useful than sequence diagrams in case the execution of processes is asynchronous

---

UML for System Co-design

[Image of NUS logo]
• Transaction – The exchange of a data or an event between two components of a model

• Transaction requests = calling interface function

Don’t care synchronization details – The exchange of a data or an event between two

Transaction Level Modeling
Transaction-level modeling

• High level of abstraction

- Communication among modules are separated from the implementation of the functional units
- Communication architecture details can be incorporated to allow performance estimation and architecture exploration
- Be constructed very early in the design process

- Transaction-level modeling

UML for System Co-design
• Calls to Module Functions Are Modeled as Transactions

• Avoidance of "sc_signal" channels

• Multiple Concurrent Processes

• Communication via Shared Data Variables

• Not Pin-Accurate

• Synchronization Scheme

TLM in SystemC
Process

Composition relationship between a module and its processes

Attribute dependency

Relationship among processes such as dependency

Precedence
• High performance transaction-level model
Static structure

UML for System Co-design
UML for System Co-design

Static structure (cont'd)
Collaboration of components during blocking access
UML for System Co-design

simple_bus constructor

SC_THREAD(main_action)

handle_request

[if (m_current_request)]

[if (!m_current_request) then get_next_request();]

arbitrate(const simple_bus_request_vec &requests)

return (request);

CheckRequestsForSlaves

burst_read(priority, data, address, length, lock)

wait(request->transfer_done)

requestQueueUpdated

setLock

transferRequest

transfer_doneNotify

[if (stop)]

~simple_bus()

Statechart of Simple Bus

serving blocking read

Statechart of Simple Bus
Calls to Module Functions Are Modeled as Transactions

A Simple Bus Example (cont'd)
Two-phase synchronization scheme

A Simple Bus Example (cont'd)
Considerations on Extensions to UML

- Port binding to a channel or another port
- Module and channel instantiation
- How to model cycle/lock-accurate
- Add timing details

UML for System Co-design
To Add into UML Tools

Others

• For interface in the same package: Top package::interface::interface-name
  belonging to the class representing module

Ports

• Dynamic sensitivity

• Static sensitivity

Events

• SystemC datatypes: logic type (01XZ)

Add support for SystemC predefined interface

• SystemC datatypes: logic type (01XZ)

• SystemC datatypes: logic type (01XZ)
For the convenience of designers:

• Predefined SystemC elements that inherited by user defined classes can be hidden in the model
• Develop a process model for TLM (like RUP) and use UML to describe it

UML for System Co-design
Conclusion

• UML is a feasible approach for system co-design
• UML wrapper for SystemC is practical
• Code generation from UML to SystemC will be doable
• Extension to UML is necessary
Thank YOU