UML and SystemC
Their Roles in SoC Codesign

Pao-Ann Hsiung
Department of Computer Science & Info. Eng.
National Chung Cheng University
Chiayi, Taiwan, ROC.
http://www.cs.ccu.edu.tw/~pahsiung/

SoC Design Flow and Tools, March 2005
Contents

- Introduction
- UML/SystemC-based SoC Codesign Flows
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
Contents

- **Introduction**
- **UML/SystemC-based SoC Codesign Flows**
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- **Conclusions**
- **References**
Introduction

- **Unified Modeling Language (UML)**
  - An industry standard
  - Driven by OMG “www.omg.org”
  - Widely adopted by Software Engineers
  - Currently still version 1.5
  - Version 2.0
    - Lots of enhancements
      - Especially for system-level design
        - deployment,
        - timing,
        - sequence diagram control flow,
Introduction

- UML was considered for system design by Cadence in 1999
  - However, it was a consensus then that UML was insufficient for system design
  - Nevertheless, now UML 2.0 is much more enhanced for system design
  - Several design companies have already adopted UML in their SoC hw-sw codesign flows
    - STMicroelectronics, Fujitsu, NEC, Alcatel, IBM, etc.
Why UML? Why SystemC?

- There is a rapidly growing trend of using
  - UML as a system-level design language, and
  - SystemC as a system-level implementation language

- Why UML?
  - Platform independent,
  - Implementation independent,
  - Customizable for any application domain,
  - Easily extensible (stereotypes, profiles, tags, etc.)
Why UML? Why SystemC?

Why SystemC?
- Can implement HW and SW and interfaces
- Multiple abstraction levels
- Executable (directly simulatable)
- Reuse of large base of C/C++ reference implementations (JPEG, MPEG, …)
- Synthesizable
  - Under some restrictions
    - Synopsys System Studio
UML versus SystemC?

- **UML vs. SystemC**
  - Users need to only deal with **UML diagrams**
    - SystemC code can be generated (semi-)automatically
  - A **high-level view** of everything
    - Simulation, step-by-step debugging
    - Makes complex system designs more manageable
  - UML is a platform **independent** language
  - System is a platform **specific** language
  - Facilitates transaction-level modeling (TLM)
    - Accelerated functional and performance evaluations
### UML versus SystemC

#### Overlap of Concepts between SystemC and UML 2.0

<table>
<thead>
<tr>
<th>SystemC 2.1</th>
<th>UML 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>class</td>
</tr>
<tr>
<td>process</td>
<td>class</td>
</tr>
<tr>
<td>port</td>
<td>port</td>
</tr>
<tr>
<td>interface</td>
<td>interface</td>
</tr>
<tr>
<td>channel</td>
<td>connector or class</td>
</tr>
<tr>
<td>event</td>
<td>trigger</td>
</tr>
<tr>
<td>sensitivity</td>
<td>trigger</td>
</tr>
</tbody>
</table>
Contents

- Introduction
- **UML/SystemC-based SoC Codesign Flows**
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
Several design flows have been proposed by several different organizations and research results.

Our classification:

- **UML Extension-based Design Flow**
  - NUS, NEC (ACES), Alcatel (OWL), Fujitsu (SLOOP)

- **UML Profile-based Design Flow**
  - STMicro, Fujitsu, U.Catania (Italy)

- **Platform Model-based Design Flow**
  - INRIA (Gaspard2, France), U. Applied Sci Mittweida, Freiberg U. (MOCCA, Germany)
Contents

- Introduction
- UML/SystemC-based SoC Codesign Flows
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
Some UML diagrams are chosen as system input models

- Class, Statechart, Sequence, Collaboration, …

Extensions defined as stereotypes

- Mainly modeling the functional and timing constructs in SystemC

System modeled using the Extended UML

UML models mapped to SystemC code

- Automatic generation of code
UML Ext-based Flow (1)

Models Used:
- Class Diagram
- Statecharts

Tools Used:
- Rhapsody
- Velocity
- Synopsys System Studio

NUS Version (1)

UML Ext-based Flow (1)

- Class Diagram
UML Ext-based Flow (1)

- Statechart
UML Ext-based Flow (1)

- SystemC Block Diagram
UML Ext-based Flow (1)

- Generated SystemC complies with the Synopsys SystemC compiler restrictions, hence is synthesizable
- The input UML is translated into SystemC code and then the SystemC code is synthesized
Models Used:
- Class Diagram
- Statecharts
- Structure Diagram

Tools Used:
- Rose RT
- RT2SystemC
- Synopsys System Studio

UML Ext-based Flow (3)

Overall Design Flow

UML

Design Problem Formulation
(Use Case Diagrams)

Functional Specification
(Class, State Charts, Activity, Sequence Diagrams)

UML Database Exploration

Design summary

Component Mapping

Platform Selection

Communication Refinement

Web-based Interface

Software synthesis

Discrete Event models

RTL netlist

CYBER Behavioral Synthesizer

RTOS .c tasks

SystemC Behavioral HW/SW Co-simulation

Partitioning

Hardware Perf/Power Estimation

Debug opt

CysimGen

Target C Compiler

Interface synthesis

Classmate Architectural Co-Simulation

ACES Codesign Environment

External Tools
UML Ext-based Flow (3)

Functional Specification
UML Ext-based Flow (3)

- Models used
  - Object Model Diagram
  - State Diagram
  - Sequence Diagram

- Tools used
  - front-end tools (self-developed)
    - Rhapsody
    - Translator, Code Generator
  - Back-end tools
    - CYBER, CysimGen, Classmate, C compiler
UML Ext-based Flow (3)

- Object Model Diagram

MainCntr
- Reset
- Progress
- Finish

AveControl
- row_count : int
- col_count : int
- triggerInReady(): void
- triggerFinished(): void
- readPixelNeighbours(): void
- storeAverageToOutputBuff(): void

FileIO
- inputFile : ifstream
- outputFile : ofstream
- initialize(): void
- readFileToMemory(): void
- writeMemoryToFile(): void

aveAdder
- computeAverage(): void
- triggerOutReady(): void

Class Instance → Functional Component
Relation Instance → Event

HW-SW Partitioning
State Diagram for Module AveControl

- **Init**: row_count = 0;
- **CheckRow**: col_count = 0;
- **CheckCol**:
  - [row_count >= height] triggerFinished();
  - [row_count < height]
  - /col_count++;
  - [col_count >= width] row_count++;
  - [col_count < width]
- **StoreAvgInBuff**: storeAverageIntoOutBuffer();
- **SendProcessingData**: event_gen((void*)itsAveAdder, "inready", "aveAdder");
UML Ext-based Flow (3)

Sequence Diagram

:MainCntr | :AveControl | :aveAdder | :FileIO

reset()

progress()

Repeated on every 3x3 block in the image

inready()

outready()

finish()
#include <AveControl.h>

// External memories
extern sc_int InputBuffer[height*width];
extern sc_int OutputBuffer[height*width];

SC_MODULE(AveControl) {
  sc_in clk clk;
  sc_in<bool> var;

  // Input terminals
  sc_in<sc_int<8>> o11; // input data
  sc_in<bool> outready; // input event
  sc_in<bool> Reset;    // " "
  sc_in<bool> Progress; // " "

  // Output terminals
  sc_out<bool> Finish; // output event
  sc_out<bool> lready; // " "
  sc_out<sc_int<8>> t00; // output data

  ... Omitted ...
}

SC_CTOR(AveControl) {
  SC_CTHREAD(main, clk.pos());
  watching(zst.delayed() == 0);
}

void main(void) {
  Init:
    aces_wait(Progress);
    row_count=0;
    goto CheckRow;

  CheckRow:
    col_count=0;
    if ( row_count < height ) {
      goto CheckCol;
    }
    else { // Send Finish
      event_gen((void*)itsMainCnt, "finish", "MainCnt");
      event_gen((void*)itsFileIO, "finish", "FileIO");
      goto Init;
    }

  CheckCol:
    Omitted
}
OWL (OFDM Wireless LAN) Project
- Wireless LAN chipset
  - Orthogonal Frequency Division Multiplexing (OFDM) modulation technique

Models used
- Vision Document, Class, Sequence diagrams

Tools used
- Matlab, Rational Unified Process, …
## UML Ext-based Flow (4)

### OWL Project Iterations

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>It0</th>
<th>It1</th>
<th>It2</th>
<th>It3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Feasibility, Requirements and Algorithms</td>
<td>Architecture and High Level Modelling</td>
<td>Detailed Design and Cosimulation</td>
<td>Hardware Prototype (FPGA)</td>
<td>Silicon</td>
</tr>
<tr>
<td>Systems</td>
<td>Capture reqts; Agree Vision Doc; Create UC model; Develop key algo’s</td>
<td>Specify Architecture; Create HL SystemC / UML model; Demonstrate architecture can meet project requirements</td>
<td>Cosimulation of SystemC, VHDL and SW</td>
<td>Cosimulation of SW on target and HW on FPGA; Conduct system V&amp;V</td>
<td>Silicon Verification; Product Qualification or Approvals</td>
</tr>
<tr>
<td>Hardware</td>
<td>Involvement with SystemC specification; Early trials of SystemC to VHDL mapping</td>
<td>Detailed Design of VHDL</td>
<td>Port design to FPGA</td>
<td>Back-end design and silicon fab</td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>Specify SW Arch; Create Host-Based SW framework and basic functionality</td>
<td>Build further SW functionality on It0 framework; Port to target</td>
<td>Build further functionality on It0/It1 framework</td>
<td>Build further SW functionality</td>
<td></td>
</tr>
</tbody>
</table>
Using Matlab as test data I/O for SystemC models
SLOOP (System-Level design with OO Process)

SLOOP employs four models to develop SoC incrementally before software and hardware implementation.
UML Ext-based Flow (5)

- **Conceptual Model**
  - To grasp functional and nonfunctional requirements

- **Functional Model**
  - Computation and communication workloads

- **Architectural Model**
  - Processing and communication resources

- **Performance Model**
  - Maps: process $\rightarrow$ processing resources, communication $\rightarrow$ communication resources

4 Models used in SLOOP
Realizing each model

- **Modeling phase**: specifies results of analysis and design using UML
- **Implementation phase**: implements the UML model into C++/SystemC as an executable model
UML Ext-based Flow (5)

- Tools used:
  - ObjecTime ROOM
  - Rose RT

- UML Extensions
  - Module
  - Port
  - Interface
  - Channel
Application of the SLOOP design process to an image decoding system
Conceptual Model

- **Use case diagrams**
  - Boundary of target system
  - Associate the external stimulus **actors**

- **Class diagrams**
  - Describe the **data structure**

- **Sequence diagrams**
  - Represent **scenarios** of use case
Functional Model
Executable Functional Model

- The functional model is also implemented with SystemC as an executable model
- The computation workload and communication workload
  - Obtained by execution of the functional model
  - Help designer to decide the initial parameters of the architectural model
## Computation Workload and Communication Workload

### Table 1. Computation Workload

<table>
<thead>
<tr>
<th>Module</th>
<th>Operation</th>
<th>Number of invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamcapturer</td>
<td>Processing stream data</td>
<td>8,100</td>
</tr>
<tr>
<td>decoder</td>
<td>Processing stream data</td>
<td>8,100</td>
</tr>
<tr>
<td></td>
<td>Processing VLD data</td>
<td>119,596</td>
</tr>
<tr>
<td></td>
<td>Processing IQ data</td>
<td>119,596</td>
</tr>
</tbody>
</table>

### Table 2. Communication Workload

<table>
<thead>
<tr>
<th>Channel</th>
<th>Token</th>
<th>Number of tokens</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream2decoder_ch</td>
<td>EncodedBlock</td>
<td>8,100</td>
</tr>
<tr>
<td>decoder2huffmandecoder_ch</td>
<td>BitVector</td>
<td>13,585</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Architectural Model

- Modeled using deployment diagram of UML
Performance Model

- **mpu**: Processor
  - scheduling: KindOfSch = PriorityBased
  - Deploys
    - compositor
    - decoder

- **HW1**: Hardware
  - Deploys
    - streamcapture

- **HW2**: Hardware
  - Deploys
    - huffinanddecoder

- **bus**: Bus
  - width: Integer = 32
  - transLatency: Integer = 10

- **ram**: RAM
  - bitwidth: Integer = 32
  - readLatency: Integer = 10
  - writeLatency: Integer = 10
  - Deploys
    - decoder2q_ch
    - idct2deschuffle_ch
    - stream2decoder_ch
    - idct

- **HW3**: Hardware
  - Deploys
    - idct

- **HW4**: Hardware
  - Deploys
    - iq

- **HW5**: Hardware
  - Deploys
    - bimapwriter
Performance Model (Cont.)

- Map the **functional model** onto the **architectural model**
- Implementation of the performance model with **SystemC**
  - Obtain the performance **evaluation results**
  - Satisfied the performance requirement or not
  - **Bottleneck analysis** can help us to improve the selected architecture
Performance Evaluation Metric

- Performance metric
  $\Rightarrow$ Throughput (frames per second)

- Requirement of performance
  $\Rightarrow$ 30 frames per second
Experiment 1

- ‘decoder’ & ‘compositor’ => processor
- Other modules => hardware components
- The communication channel => realized with RAM and bus
- Simulation results:
  - Throughput = 12 frames per second
  - Bus utilization = 57%

*Throughput well below the requirement!*
Utilizations of Modules in Experiment 1
Utilization of Channels in Experiment 1
Experiment 2

- Remove the module ‘decoder’ from processor => Map it onto a new hardware component
- Evaluated this modified architecture in order to see how the performance had changed
- Simulation results:
  - Throughput = 24 frames per second
  - Bus utilization = 73%

*The bus was the bottleneck!*
Experiment 3

- Connected ‘decoder’, ‘huffmandecoder’, and ‘iq’ **directly** without utilizing bus
- Expand the bus width to **64 bits**
- Simulation results:
  - Throughput = **30 frames per second**
  - Bus utilization is **50%**

*Decided to select this architecture for implementation!*
Contents

- Introduction
- UML/SystemC-based SoC Codesign Flows
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
**UML Profile-based Design Flow**

- **Bridging the gap between UML and SystemC**

  **Approach 1:** reverse engineer SystemC libraries to create a library of classes that can be used when modeling – does not take advantage of UML graphical capabilities

  **Approach 2:** define a UML profile to capture central SystemC constructs – module, process, port–interface, channel, event
UML Profile-based Design Flow

Defining a SystemC Profile in UML

- UML lacks a set of predefined data types—reuse the ones from SystemC (and C++)—define your own datatypes

Provide stereotypes for the most commonly used constructs—«module», «process», «port»

- «profile» SystemC
- «metaclass» Class
- «stereotype» Process
  - pk: ProcessKind
- «primitive» int
- «primitive» sc_bit
- «enum» ProcessKind
  - SC_METHOD
  - SC_THREAD
  - SC_CTHREAD

{A class stereotyped as process must be compositely owned by a class stereotyped as module}
int sc_main(int, char*[]) {
    // declare channel
    sc_fifo<int> fifo(10);
    // instantiate blocks
    writer w("writer");
    reader r("reader");
    // connect to channel
    w.out (fifo);
    r.in (fifo);
    sc_start(-1); // run forever
    return 0;
}
UML Profile-based Design Flow

- A SystemC Metamodel
UML Profile-based Design Flow

- 4 Parts in a SystemC Profile
  - Structure and Communication
    - UML stereotypes for SystemC Core Layer structures
      - modules, ports, interfaces, channels
  - Data Types
    - UML classes for SystemC data types
  - Behavior and Synchronization
    - UML stereotypes of SystemC Core Layer behavior
  - Predefined Channels, Interfaces, Ports
    - Library model or profile
UML Profile-based Design Flow

- SystemC Profile Architecture

WiSME’2004, DATE’2005
### STRUCTURE AND COMMUNICATION

<table>
<thead>
<tr>
<th>MODULE</th>
<th><img src="image1" alt="Diagram" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PORT</strong></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>INTERFACE</strong></td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>PRIMITIVE CHANNEL</strong></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td><strong>HIERARCHICAL CHANNEL</strong></td>
<td><img src="image5" alt="Diagram" /></td>
</tr>
</tbody>
</table>

#### THREAD PROCESS
- Within the operation compartment of a module’s class or of a channel’s class with the keyword «sc_thread».

#### EVENT
- Within the operation compartment of a module’s class or of a channel’s class with the keyword «sc_event».

### BEHAVIOR AND SYNCHRONIZATION

| THREAD PROCESS | As a UML method state machine. |
| EVENT | As a label for a signal trigger on a state machine transition. |

| WAIT STATEMENT | ![Diagram](image6) |
# SystemC CORE: Structure, Communication, and Behavior

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>Base Class</th>
<th>Tag</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sc_module</strong></td>
<td>Class (from StructuredClasses)</td>
<td>None</td>
<td>Modules containing processes are <em>active</em>.</td>
</tr>
<tr>
<td><strong>sc_port</strong></td>
<td>Port (from Ports)</td>
<td>max_if : int = 1 maximum number of channels that may be attached to the port. The default value is 1 (== simple port). 1. A port owns a single required interface, the so called <em>type</em> of the port. 2. A simple port is connected to one channel or to one parent port only, and it can still have zero or more child ports. 3. A port can be connected to one or more channels implementing the same port's <em>interface type</em> (a <em>multi-port</em>). 4. A port providing events that trigger cases, within the module that exposes the port, must be set <em>behavior</em> port.</td>
<td></td>
</tr>
<tr>
<td><strong>sc_interface</strong></td>
<td>Interface (from Interfaces)</td>
<td>None</td>
<td>Cannot have data fields.</td>
</tr>
<tr>
<td><strong>sc_prim_channel</strong></td>
<td>Class (from Communications)</td>
<td>None</td>
<td>1. Provides interfaces required by the ports to which the channel is attached. 2. Is <em>passive</em>.</td>
</tr>
<tr>
<td><strong>sc_channel</strong></td>
<td>Class (from StructuredClasses)</td>
<td>None</td>
<td>Provides interfaces required by the ports to which the channel is attached.</td>
</tr>
<tr>
<td><strong>sc_connector</strong></td>
<td>Connector (from InternalStructures)</td>
<td>None</td>
<td>Always connects ports to channel instances.</td>
</tr>
<tr>
<td><strong>sc_relay_connector</strong></td>
<td>Connector (from InternalStructures)</td>
<td>None</td>
<td>Always connects a parent module's port to one or more lower level module's ports in a hierarchical structure.</td>
</tr>
<tr>
<td><strong>sc_method</strong></td>
<td>Operation, Behavior (from Communications, BasicBehaviors)</td>
<td>1. sensitive: String specifies the process's static sensitivity list of events 1. dont_initialize: Boolean = false states whether the process is not initialized when execution starts. Default is false. 1. Must be applied both to the implementing behavior and to the behavior's specification (i.e. the operation that the behavior implements). Both the behavior and its specification have no input parameters and none return parameter. 2. If applied to an operation, the stereotype sc_module or sc_channel must have been applied to the featu-ring class. 3. Cannot call other processes behaviors directly. 4. Applied to a behavior, it cannot be invoked while it is still executing from a previous invocation (the isReentrant attribute value is always false). 5. A method process cannot be explicitly suspended through the wait mechanism of the thread processes.</td>
<td></td>
</tr>
<tr>
<td><strong>sc_thread</strong></td>
<td>Operation, Behavior (from Communications, BasicBehaviors)</td>
<td>1. sensitive: String specifies the process's static sensitivity list of events. 1. dont_initialize: Boolean = false states whether the process is not initialized when execution starts. Default is false. 1. Must be applied both to the implementing behavior and to the behavior's specification (i.e. the operation that the behavior implements). Both the behavior and its specification have no input parameters and none return parameter. 2. If applied to an operation, the stereotype sc_module or sc_channel must have been applied to the featu-ring class. 3. Cannot call other processes behaviors directly. 4. Applied to a behavior, it cannot be invoked while it is still executing from a previous invocation (the isReentrant attribute value is always false).</td>
<td></td>
</tr>
<tr>
<td><strong>sc_event</strong></td>
<td>Signal (from Communications)</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
Conventional SoC Design Flow

WiSME’2004, DATE’2005
UML Profile-based Design Flow
A simple communication modeling example
  - From SystemC 2.0 package
Producer module
Consumer module
FIFO channel
  - write_if (blocking write)
    - write(char)
    - reset()
  - read_if (blocking read)
    - read(char&)
    - num_available()
class write_if : virtual public sc_interface {
    public:
        virtual void write(char) = 0;
        virtual void reset() = 0; }

    class read_if : virtual public sc_interface {
        public:
            virtual void read(char &) = 0;
            virtual int num_available() = 0; }

read_write_if.h
#include "read_write_if.h"

class producer: public sc_module {

public:
    sc_port<write_if> out;
    void main();
    SC_HAS_PROCESS(producer);
    producer(sc_module_name mn): sc_module(mn) {
        SC_THREAD(main);
    }
};
UML Profile-based Design Flow

```cpp
#include "producer.h"
void producer::main() {
    const char *c = "Visit www.systemc.org\n";
    out->reset();
    while (*c)
        out->write(*c++);
}
```

`producer.cpp`
// file fifo.h
#include "read_write_if.h"
class fifo : public sc_channel, public write_if, public read_if {
public:
    fifo(sc_module_name mn): sc_channel(mn) {
        num_elements = first = 0; }
    void write(char);
    void reset();
    void read(char&);
    int num_available();
private:
    enum e { max = 10 }; char data[max];
    int num_elements, first;
    sc_event write_event, read_event;
};
// file fifo.cpp
#include "fifo.h"
void fifo::write(char c) {
    if(num_elements == max)
        wait(read_event);
    data[(first+num_elements)%max] = c;
    ++ num_elements;
    write_event.notify(); }
void fifo::read(char& c){
    if(num_elements == 0)
        wait(write_event);
    c = data[first];
    -- num_elements;
    first = (first + 1) % max;
    read_event.notify(); }
void fifo::reset() { num_elements = first = 0; }
int fifo::num_available() { return num_elements;}

fifo.cpp
class top: public sc_module
{
  public:
    fifo * fifo_inst; // a fifo instance
    producer * producer_inst; // a producer instance
    consumer * consumer_inst; // a consumer instance
  top(sc_module_name mn): sc_module(mn) {
    fifo_inst = new fifo("Fifo1");
    producer_inst = new producer("Producer1");
    consumer_inst = new consumer("Consumer1");
    producer_inst->out(*fifo_inst);
    consumer_inst->in(*fifo_inst);
  }
};
Contents

- Introduction
- UML/SystemC-based SoC Codesign Flows
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
Platform Model-based Design Flow
Platform Model-based Design Flow

Y-Model & TLM & RTL

INRIA DART
Application PIM

- **UML 2 Component** based
  - Component models some computation
  - Ports model input and output capabilities

- **Three kinds of components**
  - **Compound** component
    - Task parallelism as a dependency graph
  - **Data parallel** component
    - Potential concurrent computations
  - **Elementary** component
    - Basic computation unit
    - May be implemented by an IP

- **Hierarchical description**
Application Metamodel
Application PIM Example
Hardware Architecture PIM

- UML 2 Component based
- Classified along 2 criteria: structural and functional
- Hierarchy to allow several granularity views
- Classification refinement for common components: CPU, memory, ...
- Repetitive components
HW Architecture PIM (Classification Refinement)
HW Architecture PIM: Example

A bi-processor shared memory architecture
HW Architecture PIM: Repetitive Component Example
Association PIM

- Expresses mapping and scheduling
  - Of an application
  - On a hardware architecture
  - Includes the application and hardware architecture models
- Adds an association view
  - Links between application components and active hardware components
  - Links between application ports and dependences and passive hardware components
- Takes advantage of
  - Repetition expression
  - Hierarchy
Model to Model Transformation

- Transformer from the INRIA DART group
  - [http://www.lifl.fr/west/mdaTransf](http://www.lifl.fr/west/mdaTransf)
  - Open source and customizable
  - Others can use it, improve it, provide other rule representation, …
  - Takes into account the remarks on OMG-QVT proposals
Model to Model Transformation

ISP = Intensive Signal Processing (a project by INRIA DART)
The INRIA DART group used the following development tools:

- **Models**
  - UML 2 tool (*Tau G2*)
- **Visual tool for development**
  - *Eclipse* plugin (home made):
    - from models to code generation
- **Model to model transformation**
  - (Code generation)
  - *MDA Transf*
Platform Model-based Design Flow

Gaspard2:
An IDE for SoC visual co-modeling

(INRIA DART)
Platform Model-based Design Flow (RTR)

- Model of Reconfigurable HW Architecture for RTR systems used by MOCCA team

RTR = Run-Time Reconfigurable
MOCCA = Model Compiler for reConfigurable Architectures

MOCCA: FDL’2003, uSoC’2004, MoDES’2004
Platform Model-based Design Flow (RTR)

MOCCA: FDL’2003, uSoC’2004, MoDES’2004
Platform Model-based Design Flow (RTR)

- Relationships among the models
MOCCA

- UML 2.0 Model Compiler
- Automated transition from platform independent application design to platform specific implementation
- Automated Design Partitioning
  - Support for partially or completely manually specified partitioning
  - Estimation of Execution and Implementation characteristics
    - Type and Message Analysis
    - Scheduling, Allocation, Binding
    - Branch Prediction
- Automated Synthesis of Platform Dependent Implementation
- RTR-Manager and Broker
Contents

- Introduction
- UML/SystemC-based SoC Codesign Flows
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
Conclusions

- Before UML 2.0, there were several extensions of UML for SystemC modeling and MDA design flow

- Now, we have
  - “profiles”
  - “platform models”
  - “model compilers,” “model transformers,” …

- UML models are “SYNTHESIZED” into HW!!!
Contents

- Introduction
- UML/SystemC-based SoC Codesign Flows
  - UML Extension-based Design Flows (x 5)
  - UML Profile-based Design Flows (x 1)
  - Platform Model-based Design Flows (x 2)
- Conclusions
- References
References (Ext-based, NUS)


References (Ext-based, ACES, OWL, SLOOP)

- M. Lajolo, NEC Laboratories, USA, M. Prevostini, University of Lugano, Switzerland, “UML in an Electronic System Level Design Methodology,” *International Workshop on UML for SoC Design* (uSoC), DAC’2004 workshop, USA.


References (Platform-based, INRIA, MOCCA)


Links

- OMG [http://www.omg.org](http://www.omg.org) (for UML)
- SystemC [http://www.systemc.org](http://www.systemc.org)
- A UML SoC Workshop sponsored by DAC 2004 (presentations available)
- INRIA DART
  - [http://www.inria.fr/recherche/equipes/dart.en.html](http://www.inria.fr/recherche/equipes/dart.en.html)
- MDA Transf
  - [http://www.lifl.fr/~dumoulin/mdaTransf/](http://www.lifl.fr/~dumoulin/mdaTransf/)