Virtual Prototyping from SysML Models

Presentation and Demonstrations

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr

Journée SysML France
Outline

Virtual prototyping in a nutshell
Partitioning with DIPLODOCUS
Methodology
Deployment with AVATAR
Virtual prototyping from design models
Customizing code generation in TTool
Outline

Virtual prototyping in a nutshell

Partitioning with DIPLODOCUS

Deployment with AVATAR
Definition

Generic definition

- From greek: \( \pi \rho \omega \tau \omicron \sigma \upsilon \omicron \nu \)
- A prototype is an early sample, model or release of a product built to test a concept or process or to act as a thing to be replicated or learned from (Wikipedia)

In our context

- Prototyping = experimenting a functional model mapped onto a concrete hardware architecture
- Virtual prototyping = experimenting a functional model mapped onto a hardware model
  - Hardware model can be more or less abstract/concrete
    - Example: CPUs emulated with Instruction Set Simulator, abstract instructions, etc.
Abstraction Level

How to Create a Stable Data Model

(Source: Peek and Poke, July, 2013)
Virtual Prototyping at Two Abstraction Levels

**Partitioning**
- Hardware highly abstracted
- Goal: Analyzing various functionally equivalent implementation alternatives
  - Mapping of functions and communications
- Metrics:
  - CPU load
  - Bus occupation
  - Silicon area

**Deployment**
- Functions to be software-implemented have been designed
- Hardware target is not yet available
  - Or not easy/practical to use
- Goal: Determining precise timing and control issues
  - Scheduling policy
  - Latencies
  - Missed deadlines
TTool: A Multi Profile Platform

**TTool**

- Open-source toolkit mainly developed by Telecom ParisTech / COMELEC
- Multi-profile toolkit
  - DIPLODOCUS, AVATAR, ...
- Support from academic (e.g. INRIA, ISAE) and industrial partners (e.g., Freescale)

**Main ideas**

- Lightweight, easy-to-use toolkit
- Simulation with model animation
- Formal proof at the push of a button
Outline

Virtual prototyping in a nutshell

Partitioning with DIPLODOCUS
Methodology

Deployment with AVATAR
**DIPLODOCUS in a Nutshell**

**DIPLODOCUS = UML Profile**
- System-level Design Space Exploration
- Y-Methodology

**Main features**
- High level of abstraction
  - Exchanged data are unvalued
  - Complexity operator
  - Parametrized hardware nodes
- Formal semantics
- Very fast simulation support
Partitioning with the Y-Methodology

Partitioning → Simulation
Formal verification

Application model → Mapping model

Architecture model → Simulation
Formal verification

Simulation Formal verification

Virtual prototyping in a nutshell
Partitioning with DIPLODOCUS
Deployment with AVATAR
Application Modeling

Simulation
Static analysis

Application modeling

Architecture modeling

Functions are first modeled independently from the architecture

DSE

mapping

Simulation
Static analysis
Architecture Modeling

Simulation
Static analysis

Application modeling

Architecture modeling

Simulation
Static analysis

DSE

Then, architecture is modeled based on generic hardware components: microprocessors, buses, memories, bridges, etc.
Mapping

Simulation
Static analysis

Application modeling
Architecture modeling

DSE

Functions are then associated to architecture components

Static analysis

mapping
Browsing the DIPLODOCUS Methodology

- Application structure
- Application behavior
- Formal verification

- Architecture model
- Mapping model
- Simulation
- Formal verification

- Application modeling
- Architecture modeling
- Mapping
- DSE
- Simulation
- Static analysis
Application Structure (Smart Card system)
Application Behavior

Activity Diagram of the SmartCard component
Formal Verification at Application Level

- No assumption on the underlying architecture
- All possible interleavings between actions are considered
- Formal verification is based on LOTOS/CADP or UPPAAL
  - Press-button approach
Architecture

- Given in terms of parameterized nodes
- CPU, HWA, Bus, Memory, Bridge, etc.
- CPU parameters: scheduling policy, cache miss ratio, miss-branching prediction, pipeline size, etc.
Virtual prototyping in a nutshell
Partitioning with DIPLODOCUS
Deployment with AVATAR

Methodology

Mapping

- Task are mapped on execution nodes (e.g., CPUs, HWAs)
- Channels are mapped on communication and storage nodes
After-Mapping Simulation

- TTool Built-in simulator
- Extremely fast
- Diagram animation
- Step-by-step execution, breakpoints, etc.
After-Mapping Simulation (Cont.)

Virtual prototyping in a nutshell
Partitioning with DIPLODOCUS
Deployment with AVATAR

Methodology

After-Mapping Simulation (Cont.)
After-Mapping Formal Verification

- TTool built-in simulator can compute all possible execution paths
- Graph analysis and visualization
After-Mapping Coverage-Enhanced Simulation

Possibility to select a given part of the model to be explored

- Minimum percentage of operators coverage
- Minimum percentage of branch coverage

Implementation: TTool built-in model-checking techniques
A Few Case Studies …

- MPEG coders and decoders (Texas Instruments)
- LTE SoC (Freescale)
- Partitioning in vehicle embedded systems (EVITA project)
- Partitioning and code generation for Software-Defined Radio systems (SACRA project)
Deployment: Overview

Virtual prototyping from SysML Models
AVATAR Design

- Block Definition and Internal Block diagrams are merged
- Synchronous and asynchronous communications
- Interactive simulation
- Formal verification of safety and security properties
Principle of Code Generation

- Only AVATAR design diagrams are taken into account
- Generated code relies on POSIX threads
  - One thread per block
- Synchronous communications between blocks is implemented in the AVATAR runtime with POSIX mutex
  - Asynchronous communications relies on linked lists managed in the AVATAR runtime
  - Time is handled based on POSIX `clock_gettime()` with `CLOCK_REALTIME` option
- ...
Method

Virtual prototyping from SysML Models
Steps

1. Model refinement
2. Selection of an OS, setting of options of this OS (scheduling algorithm, . . . )
3. Selection of a hardware platform, and selection of a task allocation scheme
5. Manual code improvement - Code might also be manually added at model level
6. Code compilation and linkage with OS
7. Simulation platform boots the OS and executes the code
8. Execution analysis: directly in TTool (sequence diagram) or with debuggers (e.g., gdb)
Support: SoCLib and MutekH

Hardware platform simulator: SoCLib (www.soclib.fr)
- Virtual prototyping of complex Systems-on-Chip
- Supports several models of processors, buses, memories
  - Example of CPUs: MIPS, ARM, SPARC, Nios2, PowerPC
- Two sets of simulation models:
  - TLM = Transaction Level Modeling
  - CABA = Cycle Accurate Bit Accurate

Embedded Operating System: MutekH (www.mutekh.fr)
- Natively handles heterogeneous multiprocessor platforms
- POSIX threads support
- Note: any Operating System supporting POSIX threading and that can be compiled for SoCLib could be used
Virtual prototyping in a nutshell
Partitioning with DIPLODOCUS
Deployment with AVATAR

Virtual prototyping from design models
Customizing code generation in TTool
(Virtual) Prototyping: Code Generation

Select options and then, click on 'start' to launch code generation / compilation / execution.
Virtual Prototyping: SocLib Simulation

SoCLib simulation based on a SystemC engine

Cycle Accurate System Simulator
ASIM/LIP6/UPMC
E-mail support: Richard.Buchmann@asim.lip6.fr
Contributors: Richard Buchmann, Sami Taktak, Paul-Jerome Kingbo, Frederic P?trot, Nicolas Pouillon

Last change: Dec 6 2011

Initializing memories with 5a
caba-vgmn-mutekh_kernel_tutorial SoCLib simulator for MutekH
Initializing memories with 5a
Initializing memories with 5a
Virtual Prototyping: Console

Console of MutekH
(Virtual) Prototyping: Trace

UML sequence diagram updated when simulating with SoCLib

TTTool displays execution traces in a sequence diagram.
Customizing Generated Code with Your Own Code: Application and Block Code

- Global code of the application
  - Inclusion of header files, global variables, ...
- Code global to one given block

Global code

Code specific to the block under edition
Customizing Generated Code with Your Own Code: State Entry Code

- Code executed whenever a state is reached
Use of Customized Generated Code

Console debug
- Using e.g. `printf()` function

Connection to a graphical interface
- Piloting the code with a graphical interface
- Visualizing what’s happening in the executed code
- Connection to graphical interface via, e.g., `sockets`
Use of Customized Generated Code (Cont)

Graphical interface for the microwave oven

- Socket connection to a graphical interface programmed in Java
Virtual prototyping in a nutshell
Partitioning with DIPLODOCUS
Deployment with AVATAR

Virtual prototyping from design models
Customizing code generation in TTool

To Go Further ...

TTool, DIPLODOCUS, AVATAR
ttool.telecom-paristech.fr