System Design with ForSyDe II

ForSyDe Design Flows

Seyed Hosein Attarzadeh Niaki
and the ForSyDe team
Outline

• Transformational Design Refinement
  ◦ Design Transformations
  ◦ Implementation Mapping
• Refinement by Replacement
  ◦ Co-simulation with Foreign Models
• Designing Multi-Processor Real-Time SoCs
• Demo
TRANSFORMATIONAL
DESIGN REFINEMENT
ForSyDe allows to describe heterogeneous models at different abstraction levels.
ForSyDe Design Flow

- Specification Model
- Transformational Design Refinement
- Implementation Model
- Transformation Library
- Architecture Model
- Implementation Mapping
  - Functional Domain
  - Implementation Domain
    - Hardware Implementation
    - Communication Implementation
    - Software Implementation
Semantic-Preserving Design Transformations

- $\text{mapSY} (+2) \circ \text{mapSY} (+5)$ can be transformed into $\text{mapSY} (+7)$ Possible in the synchronous model, since computation takes no time (not even a delta delay)!
Non-Semantic-Preserving Design Transformations

- Also non-semantic preserving design transformations are needed in order to arrive at an efficient implementation
  - requires verification, since semantics are changed
Implementation Mapping: Hardware Synthesis in ForSyDe-Haskell

Wednesday, October 02, 2013   System design Languages (IL2452)
Dealing with Legacy Code and IP Blocks

SEMI-FORMAL REFINEMENT BY FOREIGN MODEL INTEGRATION
Problem Statement

Verifiable during the design flow

Implementation
Problem Statement

• Heterogeneous
• Simulatable

High-level Model

Implementation

Verifiable
during the
design flow
Problem Statement

High-level Model

- Heterogeneous
- Simulatable

Implementation

Verifiable during the design flow

IP Blocks & Legacy Code

Different languages & Tools

Wednesday, October 02, 2013  System design Languages (IL2452)
Refinement-by-Replacement Idea

- A set of refinement operations are defined
- Applied iteratively to lower the abstraction level of the model
  - Dependencies between operations should be respected
  - Rollbacks enable design space exploration

Putting All Together – Refinement By Replacement
Putting All Together – Refinement By Replacement
Dealing with Legacy Code and IP Blocks

CO-SIMULATION WITH FOREIGN MODELS
Heterogeneous Co-simulation

- Simulation bus
- Heterogeneous framework

Sim I  Sim II  Sim III
Modeling Framework

Sim I  Modeling Framework  Sim III
Sim II
Heterogeneous Co-simulation

- Simulation bus
  - Homogeneous modeling framework
    - One standard interface
    - No EDA/IP vendor support needed
  - Simulation bus

- Heterogeneous framework
  - Better handling of heterogeneity
  - Natural interfaces for each model

Modeling Framework

Wednesday, October 02, 2013  System design Languages (IL2452)
Wrappers, In Short

Wrapper processes provide a clean way for integration of legacy cores in a formal framework

- Co-simulation and co-execution (SIL, HIL, etc.)

Each model communicates/synchronizes using its natural MoC

- Easier to integrate external models

No specific EDA/IP vendor support needed

Model wrapper examples for the SY MoC

- GDB wrapper
- HDL wrapper
- Simulink wrapper

Wednesday, October 02, 2013   System design Languages (IL2452)
Wrappers and Refinement-by-Replacement

Wrappers

A General Approach
Wrapper Processes
Model Wrappers

Refinement Operations

Partial Refinement
Semi-Formal

The Design Flow

Application of Ref. Ops
Rollbacks and DSE

Wednesday, October 02, 2013  System design Languages (IL2452)
Platform-Based Design Using ForSyDe

DESIGNING MULTI-PROCESSOR REAL-TIME SOCS

Simplified Design Flow
Simplified Design Flow

Heterogeneous System Model
- Formal Base (MoCs)
- Executable
- Analyzable
Simplified Design Flow

Heterogeneous System Model
- Formal Base (MoCs)
- Executable
- Analyzable

Flexible Target Architecture
- Multiprocessor
- Predictable Performance

Wednesday, October 02, 2013  System design Languages (IL2452)
**Simplified Design Flow**

**Heterogeneous System Model**
- Formal Base (MoCs)
- Executable
- Analyzable

**Design Constraints**
- Real-Time
- Power Efficiency
- ...

**Flexible Target Architecture**
- Multiprocessor
- Predictable Performance

---

**Wednesday, October 02, 2013**

System design Languages (IL2452)
Simplified Design Flow

Heterogeneous System Model
- Formal Base (MoCs)
- Executable
- Analyzable

Design Constraints
- Real-Time
- Power Efficiency
- ...

Flexible Target Architecture
- Multiprocessor
- Predictable Performance

Design Space Exploration (based on formal model and predictable architecture)

Objective: Find a mapping that satisfies design constraints
Simplified Design Flow

Heterogeneous System Model
- Formal Base (MoCs)
- Executable
- Analyzable

Design Constraints
- Real-Time
- Power Efficiency
- ...

Flexible Target Architecture
- Multiprocessor
- Predictable Performance

Design Space Exploration
(based on formal model and predictable architecture)

Objective:
Find a mapping that satisfies design constraints

Wednesday, October 02, 2013  System design Languages (IL2452)
Simplified Design Flow

Heterogeneous System Model
- Formal Base (MoCs)
- Executable
- Analyzable

Design Constraints
- Real-Time
- Power Efficiency
- ...

Flexible Target Architecture
- Multiprocessor
- Predictable Performance

Design Space Exploration (based on formal model and predictable architecture)
Objective: Find a mapping that satisfies design constraints

Implementation
- Customized HW
- Efficient SW
Overview

Software Design Flow

- Designer models executable system model
- Abstract analyzable models are extracted
- MoC theory is used for design space exploration and synthesis => efficient mapping
- Code is generated for each individual processor
Step 1: Designer Creates Executable System Model

- Designer creates an executable system model using the ForSyDe SystemC libraries
  - System functionality is expressed in C/C++
  - Software parts are modeled using the (C)SDF or the synchronous MoC libraries
  - The environment can be modeled using discrete-event or continuous time MoC libraries

- **Result:** Heterogeneous SystemC Model that is based on formal MoC semantics
Step 2: Designer Verifies Functionality

- Designer verifies system functionality
  - Present approach uses simulation, where the environment can be modeled as part of the test-bench
  - Since the model is based on formal semantics, future approaches can use formal verification (after step 3)
Step 3: Formal Analyzable Abstract Models are Extracted

- During the elaboration phase formal analyzable models are automatically extracted
  - XML-files that represent an abstract view of the concurrent process network
    - Process network graph with properties like production or consumption rates
    - Size of data tokens is extracted
  - C/C++ code of leaf processes is extracted as separate files

- Result: Full information of executable SystemC model is available in an analyzable format
Step 4: Design Space Exploration

- **Mapping problem**: Which process shall be mapped to which processor?
  - Design constraints: guaranteed throughput, memory constraints, power constraints, …
  - Abstract system model
  - Model of target platform architecture
  - Worst case execution time of C/C++ functions needs to provided (by estimates or static analysis)

- Mapping problem can be formulated as optimization problem
Target Architecture

- The target architecture needs to be predictable in order to give performance guarantees, which are used by the mapping algorithm
  - Worst case execution time of functions
  - Worst case communication time
- Message-passing architectures (like network-on-chip) fit perfectly to ForSyDe, since there is no shared data, but also communication time needs to be bounded and predictable
Step 5: Synthesis to Target Platform

**Process Network Synthesis**

1. Implementation of static schedule
2. Generation of internal FIFO buffers
3. Code generation for each process, including function calls for message passing communication
Step 5: Synthesis to Target Platform

Communication Synthesis

- Abstract message passing functions are implemented on the target platform

Assumption: Target platform provides libraries with primitives for message passing
Step 5: Synthesis to Target Platform

Compilation

- C-code for each processor is compiled to target processor
- Efficient C-compilers for target processors can be used
From ForSyDe Models to MPSoCs

Designer Creates Executable System Model
- Using ForSyDe-SystemC
- SY/SDF MoC for functionality
- DE/CT for environment

Designer Verifies Functionality
- Simulation
- Formal verification

Formal Analyzable Abstract Models are Extracted
- XML + C/C++

Design Space Exploration
- The mapping problem
- Based on W.C.E.T and constraints

Synthesis to Target Platform
- Process network synthesis
- Communication synthesis
- Compilation
From ForSyDe Models to MPSoCs

Process network synthesis | Communication synthesis

The mapping problem

Designer Creates Executable System Model

Virtual processor

CompOSe

MB

DMA

Lmem

Cmem

Virtual NoC

HW arbiter

Æthereal NoC

mem front end

mem back end

Virtual memory

HW arbiter

XML + C/C++

Designer Verifies Functionality

Simulation

Formal verification

Formal Analyzable Abstract Models

The mapping problem based on W.C.E.T and constraints

Designer Creates Executable System Model

SystemC - SY/SDF MoC for functionality
dE/CT for environment

PE 1

Nios II/e

Performance counter

DMA controller

TDM arbitrator

Shared memories

PE 2

Nios II/e

Performance counter

DMA controller

TDM arbitrator
A Mixed-Criticality Design Flow
Current Status of ForSyDe

- ForSyDe is implemented as domain specific language in Haskell and SystemC.
- Several libraries for different models of computation exist and can be simulated as integrated model.
- ForSyDe processes are formally defined.
- ForSyDe supports modeling at different levels of abstraction.
- There exists a back-end for hardware design and synthesis (Haskell-ForSyDe is synthesized into VHDL).
- High-level and synthesizable models can be co-simulated giving access to powerful test benches.
- Foreign models can be co-simulated or co-executed (Hardware-in-the-loop) using ForSyDe-wrappers.
- ForSyDe has been used in software design flows of different (predictable) platforms

https://forsyde.ict.kth.se
Other Notable Applications

- **Adaptivity & runtime reconfiguration**
  - Parameter, mode, function and interface adaptivity

- **Modeling cyber-physical systems**
  - Interacting physical & computational processes
Putting stuff in action

THE TRANSCEIVER SYSTEM DEMO
Example: A Transceiver System
Transceiver System – Simulation

1. Synchronous Input Signal
   \[ \text{in} = \{0, 1, 2, 3, 4, 5\} \]

4. Gaussian Noise

3. Transceiver Output

5. Noisy Transceiver Input

7. Synchronous Output Signal
   \[ \text{out} = \{0, 1, 10, 3, 4, 5\} \]
Transceiver System – Synthesis in Altera Quartus II
How to Contribute to/Use ForSyDe

Use the libraries (Haskell/SystemC) to model your applications

Develop more synthesis backends

Connect ForSyDe models to verification tools

Develop more MoCs for ForSyDe

Define more transformations for design refinement

Develop/enrich your own design methodology with ideas used in ForSyDe