Semi-Automatic Derivation of Timing Models for WCET Analysis

Marc Schlickling\textsuperscript{12} Markus Pister\textsuperscript{12}

\textsuperscript{1}Saarland University, Saarbrücken, Germany
\textsuperscript{2}AbsInt Angewandte Informatik GmbH, Saarbrücken, Germany

AVACS R2 Subproject Meeting
Outline

1. Introduction
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework
Outline

1. Introduction
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework

2. Timing Model Derivation
   - Overview
   - Model Preprocessing
   - Processor State Abstractions
Outline

1 Introduction
- Integration into AVACS project structure
- Motivation
- aiT WCET Framework

2 Timing Model Derivation
- Overview
- Model Preprocessing
- Processor State Abstractions

3 Automation/Integration of the Derivation Process
- Overview
- Static Analyzes
- Transformation Tools
- Pipeline Analyzer Code Generation
## Outline

1. **Introduction**
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework

2. **Timing Model Derivation**
   - Overview
   - Model Preprocessing
   - Processor State Abstractions

3. **Automation/Integration of the Derivation Process**
   - Overview
   - Static Analyzes
   - Transformation Tools
   - Pipeline Analyzer Code Generation

4. **Conclusion**
1 Introduction
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework

2 Timing Model Derivation
   - Overview
   - Model Preprocessing
   - Processor State Abstractions

3 Automation/Integration of the Derivation Process
   - Overview
   - Static Analyzes
   - Transformation Tools
   - Pipeline Analyzer Code Generation

4 Conclusion
Integration into AVACS project structure

- **Work package 4: Node performance guarantees (Timing Analysis)**
  - Task 1: Derivation and Analysis of Abstract Processor Models
  - Task 2: Architectural Abstraction
Motivation

- Growing support of human life by complex embedded systems
- Safety-critical systems often have to fulfill strict timing constraints

Timing validation of the systems behavior if crucial for guaranteeing their correct behavior
aiT WCET Framework

Legend:
- Data
- Analysis

1. Input Executable
2. Decoding Phase
3. Control-flow Graph (CFG)
4. Value Analysis
5. Annotated CFG
6. Pipeline Analysis
7. Timing Information
8. Path Analysis
9. WCET Bound

AVACS Workshop 09/2010  Marc Schlickling and Markus Pister
Semi-Automatic Derivation of Timing Models for WCET Analysis 6/20
Caches / Pipelines

Features of modern processors:
- Out-of-order execution
- Speculation (dynamic, static)
- Caches (replacement policy, write policy)
- Branch prediction
- ...

Instruction latencies vary and depend on:
- Processor pipeline state (execution context)
- Environmental state (cache contents, hardware configuration, ...)
- Input program (size, number of memory accesses, ...)

Cache/Pipeline analysis: computes basic block execution times
Caches/Pipeline Analysis within aiT

- Based on timing model of the target architecture
- Abstract simulation of task execution
  - Non-deterministic due to lack of information (input and/or processor state)
- Timing model currently hand-crafted by human experts based on processor manuals
- Modern processors automatically synthesized from formal hardware specifications (including the instruction timing)
Outline

1 Introduction
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework

2 Timing Model Derivation
   - Overview
   - Model Preprocessing
   - Processor State Abstractions

3 Automation/Integration of the Derivation Process
   - Overview
   - Static Analyzes
   - Transformation Tools
   - Pipeline Analyzer Code Generation

4 Conclusion
Overview

- Processor specification too large to be used in aiT framework: Infineon PCP2 (40.000 loc), Gaisler Research Leon 2 (80.000 loc), Infineon TriCore 1.3 (250.000 loc)
- Specification needs to be compressed
Model Preprocessing

- **Goal:** Reduce specification size
- Eliminating parts not relevant for the timing behavior of the system

**Methods**

- **Environmental Assumption Refinement**
  - Gap between highly configurable processors and very specific usage within embedded systems
  - Some processor features are not used for a particular embedded system
  - Specification of unused features can be ignored/removed

- **Timing-Dead Code Elimination**
  - Functional behavior vs. timing behavior
  - Internal functionality of execution units not needed
  - Only the timing behavior is needed ("We do not need to know how an adder is adding but how long the addition takes.")

- **Data-Path Elimination**
  - Modeling data paths increase the resource consumption
  - Latency of instructions often not affected by content of registers/memory cells
  - Can be factored out of the cache/pipeline analysis part (cf. value analysis)
Processor State Abstractions

- On complex architectures (TriCore, MPC755) preprocessed model still too large for an efficient timing analysis
- Further model size reduction necessary
- Approximating part of the processor state $\rightarrow$ processor state abstractions
- Possible abstractions
  - Process Substitution
    - Replace VHDL processes by custom simulation routines (not necessarily VHDL)
    - Example: Cache Abstraction
  - Domain Abstraction
    - Type changes
    - Example: Address $\rightarrow$ Range of Addresses
  - Memory Abstraction
    - Elimination of large memory arrays
    - Control-flow interface
    - Adopt VHDL design to use value analysis results for memory/register accesses
Timing Model Derivation

VHDL Model

Environmental Assumption Refinement
Timing Dead Code Elimination
Data Path Elimination

Model Preprocessing

Preprocessed VHDL

Processor State Abstractions

Timing Model
Outline

1 Introduction
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework

2 Timing Model Derivation
   - Overview
   - Model Preprocessing
   - Processor State Abstractions

3 Automation/Integration of the Derivation Process
   - Overview
   - Static Analyzes
   - Transformation Tools
   - Pipeline Analyzer Code Generation

4 Conclusion
Utilize static program analyzes and transformation tools to partially automate
- Model preprocessing
- Processor state abstractions

Based on static analysis framework for VHDL

Generate cache/pipeline analysis out of the derived timing model
Static Analyzes

- **Reset Analysis**
  - Obtaining initial signal values during system reset
  - Detects activation sequences
  - Example: contents of hardware configuration registers
  - Realized by a combining constant propagation with forward slicing
  - Identification of clock domains

- **Assumption Refinement Analysis**
  - Identification of unused parts of the model
  - Example: Exclusion of interrupt handling
  - Realized by a value analysis based on intervals combined with a live variables analysis
    - Switch to polyhedral domain possible
  - Identification of timing dead code

- **Slicing**
  - Interactive understanding/exploration of the VHDL model
  - Identification of timing dead code
  - Backward slice from each pipeline exit
  - Everything not contained in the union over these slices is dead
Transformation Tools

- **DomainAbstractor**
  - Automate domain transformations on the model based on domain transformation mapping
  - Detection of needed operators on new domain
  - Built-in support for common domain transformations
  - Features different transformation scopes
  - Features transformation proposals

- **TimingDeadCodeEliminator**
  - Removal of code snippets marked as timing dead
  - Features Timing Dead Code propagation due to removals

- **ProcessReplacer**
  - Automate the replacement of VHDL processes
  - Based on implementation of an update function that simulates the timing behavior of the replaced process
  - Replaces the given process by the custom implementation
Pipeline Analyzer Code Generation

- Automatic generation of cache/pipeline analysis from (abstracted) VHDL model
- Code perfectly fits into the aiT tool chain
- Update function that computes the transition of one processor clock cycle for a given abstract processor state
- Update function can compute multiple possible successor states due to the introduced non-determinism in the timing model
- Single execution trace vs. execution tree
Outline

1. Introduction
   - Integration into AVACS project structure
   - Motivation
   - aiT WCET Framework

2. Timing Model Derivation
   - Overview
   - Model Preprocessing
   - Processor State Abstractions

3. Automation/Integration of the Derivation Process
   - Overview
   - Static Analyzes
   - Transformation Tools
   - Pipeline Analyzer Code Generation

4. Conclusion
Conclusion

- Computation based on timing models that currently are hand-crafted
  - Time consuming process
  - Error prone due to human involvement and uncertainties in the processor documentation

- Semi-automatic derivation of timing models from VHDL possible

- Integration of the derived timing models into aiT tool chain

- Derived timing models cover the whole system (extending over the processor clock domain)

- Removed human involvement up to a certain degree

- Bridging the gap between formal hardware specifications and Worst-Case Execution Time Analysis

- Precise and correct timing models

- Speeds up the creation time from a unit of months to weeks