Semi-Automatic Derivation of Timing Models for WCET Analysis

Markus Pister , Marc Schlickling

AbsInt
Angewandte Informatik

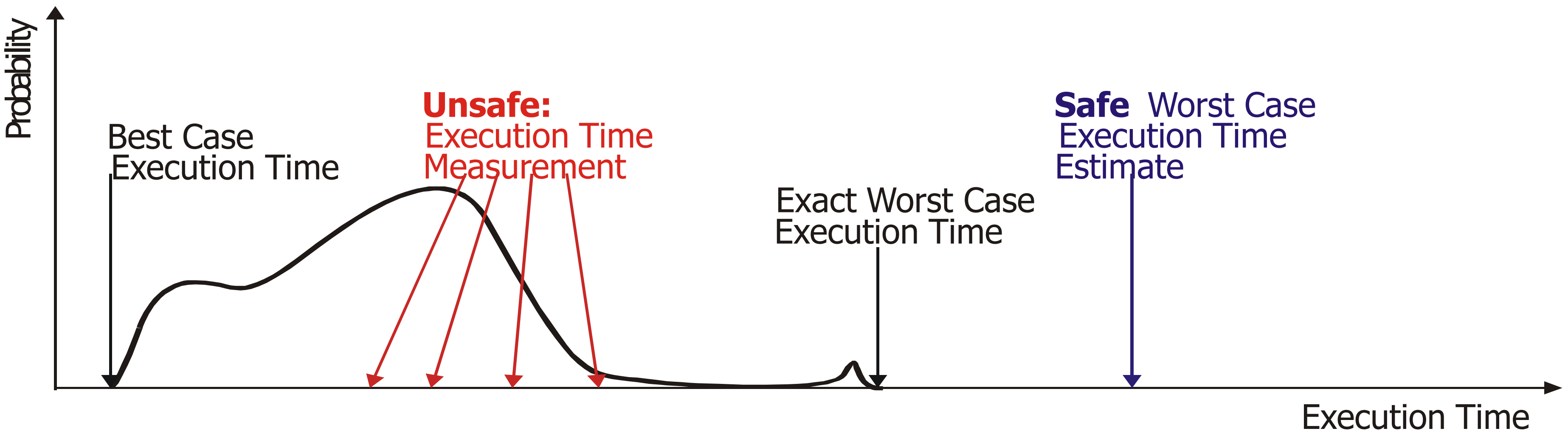
Compiler
Design
Lab

Motivaton

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

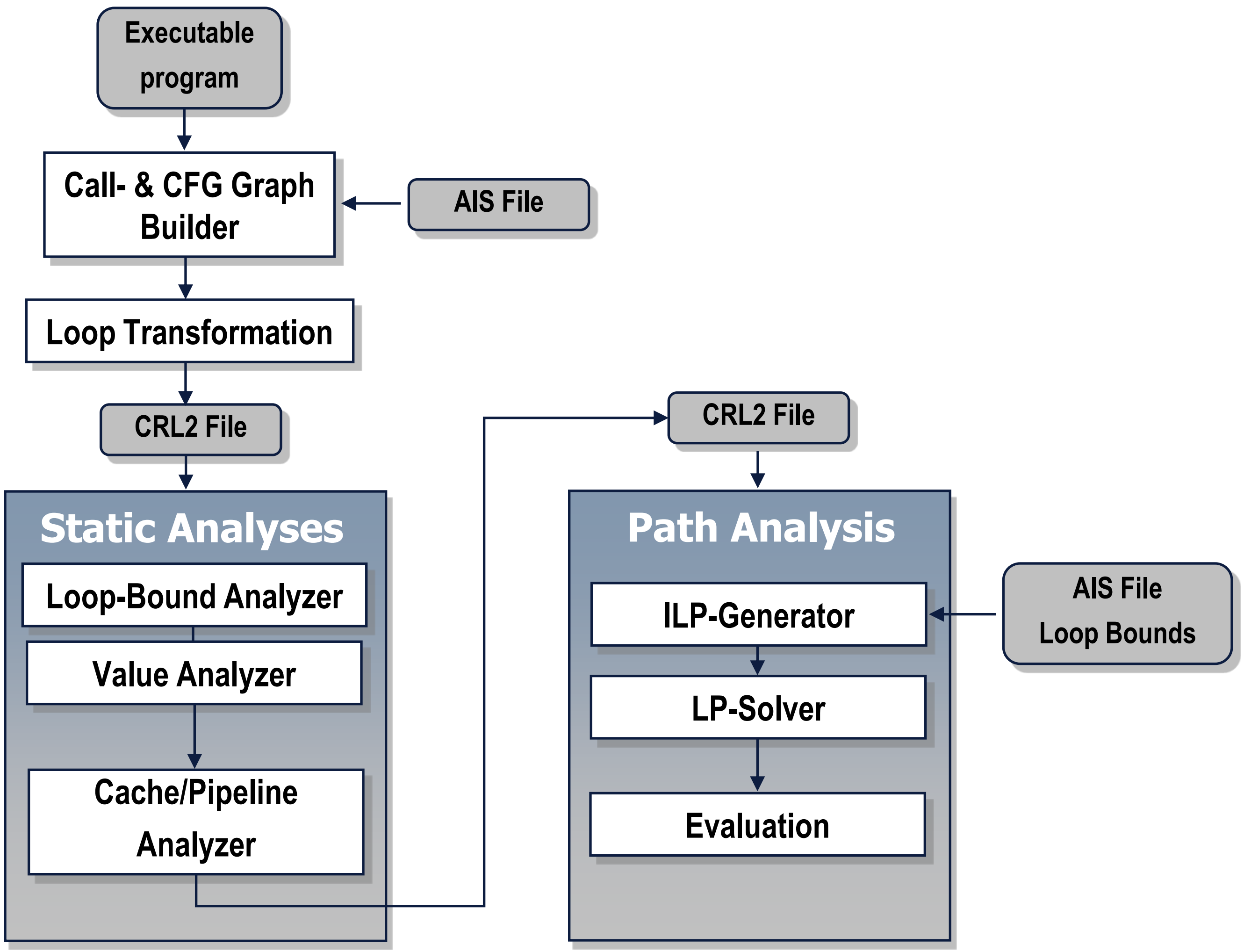
➢ Timing validation of the systems behavior is crucial for guaranteeing their correct behavior

The Timing Problem



- Execution times of tasks vary over time
  - Different input
  - Different system state
- Measurement of the worst-case execution time not possible for complex architectures
- Need static analysis (independent of inputs) for safety guarantees

aiT WCET Framework



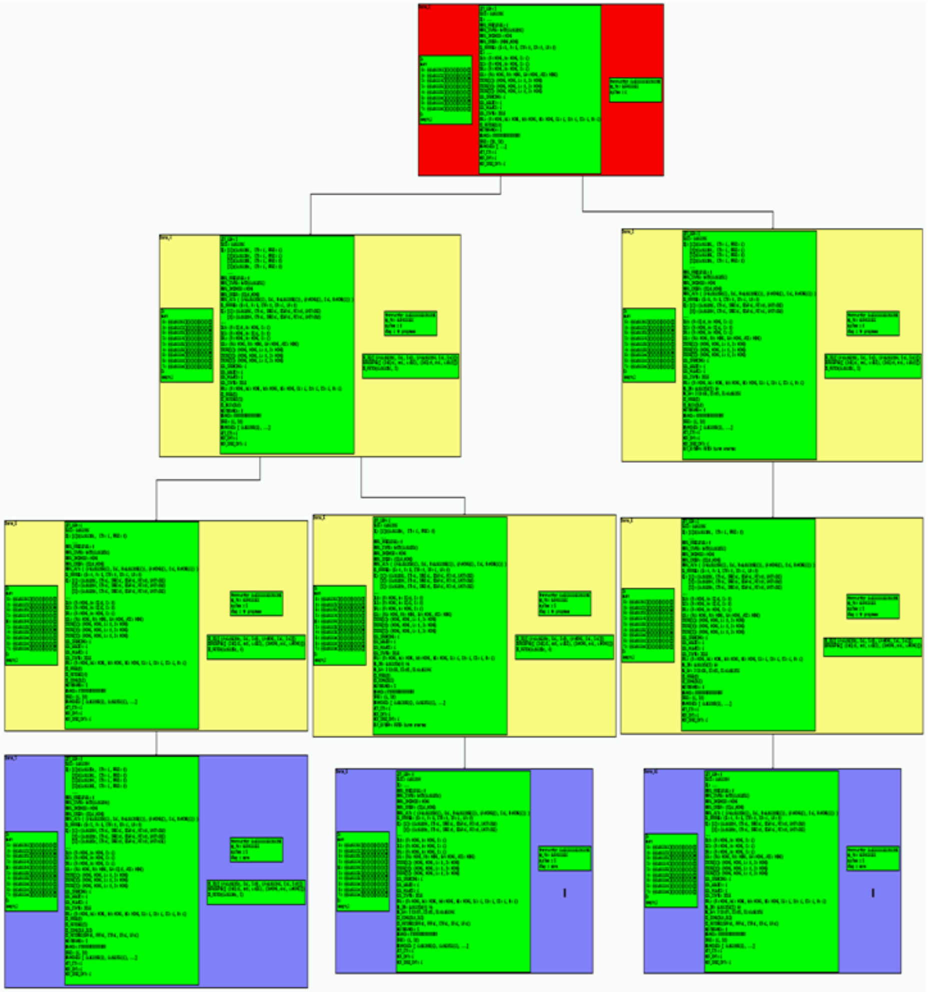
Caches / Pipelines

- Modern processors support features like
  - Out-of-order execution
  - Speculation (dynamic, static)
  - Caches (replacement policy, write policy)
  - Branch prediction
- Instruction latencies vary and depend on:
  - Processor pipeline state (i.e. execution context)
  - Environmental state (cache contents, hardware configuration, ...)
  - Input program (size, number of memory accesses, ...)

➢ Cache/Pipeline analysis: Computes basic block execution times

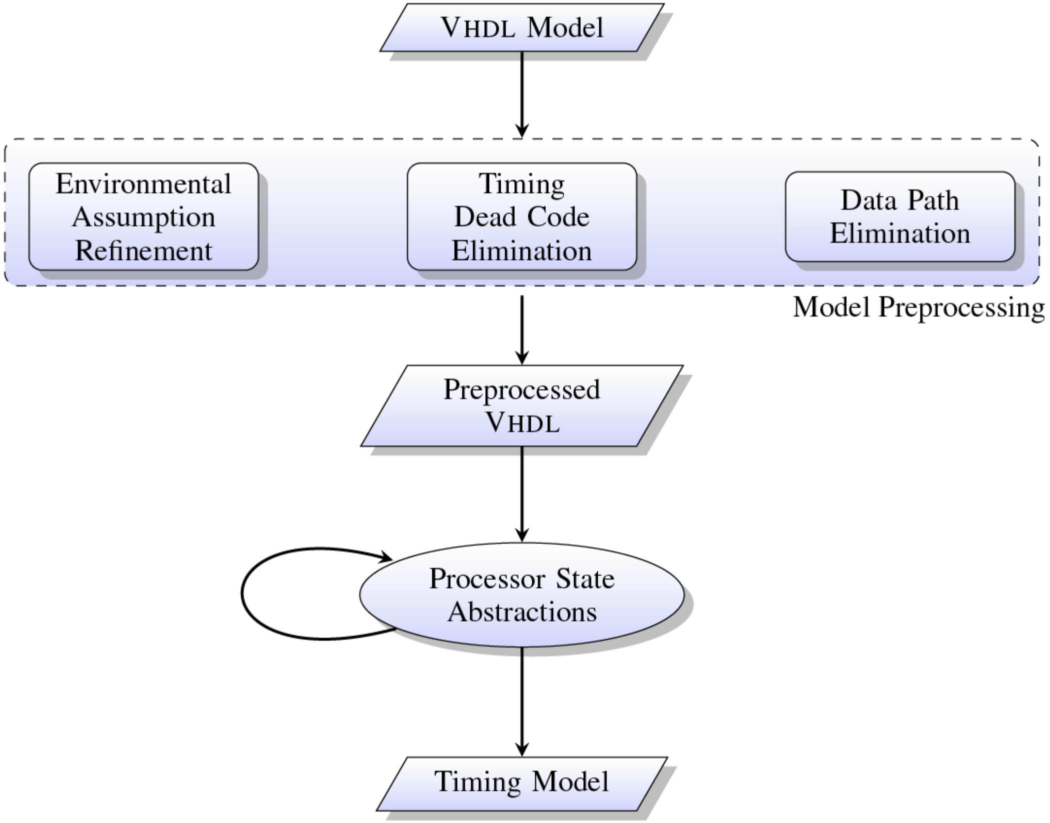
Caches/Pipeline Analysis within aiT

- Based on timing model of the target system
- 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 out of formal hardware specifications (including the instruction timing)



Deriving the Timing Model

- Processor specification too large to be used in aiT framework
  Infineon PCP2 (~40.000 loc), Leon2 (~80.000 loc), Infineon TriCore 1.3 (~250.000 loc)
- ➢ Specification needs to be compressed



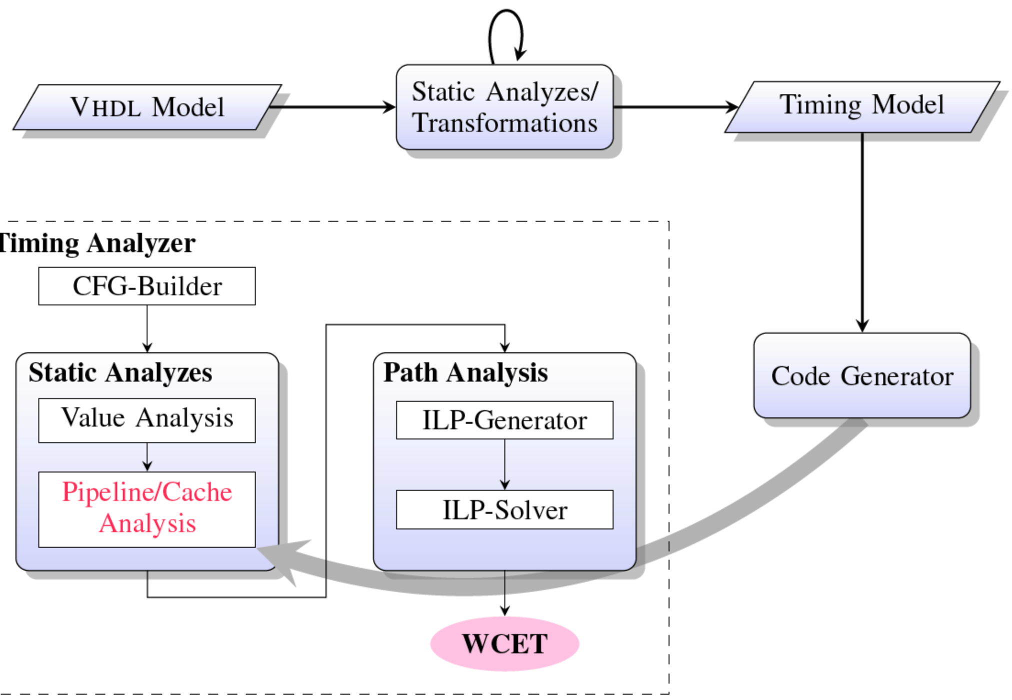
Model Preprocessing

- 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
  - 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 to large for an efficient timing analysis
- ➢ Further model size reduction necessary
- Approximating parts of the processor state
  - ➢ Processor state abstractions
- Possible Abstractions
  - Process Substitution
    - Replace VHDL processes by custom simulation routines (not necessarily in VHDL)
    - Example: Cache Abstraction
  - Domain Abstraction
    - Type changes
    - Example: Address $\rightarrow$ Address range
  - Memory Abstraction
    - Elimination of large memory arrays
    - Control-flow interface
    - Adopt VHDL design to use value analysis results for memory/register accesses

Automation of the Derivation Process



- Utilize static program analyses and transformation tools to automate
  - Model preprocessing
  - Processor state abstractions
- Based on static analysis framework for VHDL
- Generate cache/pipeline analysis out of the derived timing model

Static Analyses

- Environmental Assumption Refinement
  - Obtaining initial signal values during system reset
    - E.g. contents of hardware configuration registers
  - Identification of unused parts of the model
  - ➢ Forward slice “reset is activated”
  - ➢ Constant propagation on the result
- Timing-Dead Code Detection
  - Only parts of the model affect timing behavior
  - Combined static analyzes
    - Backward slices from each pipeline exit
    - Everything not contained in the union over these slices is dead.

Static Analyses (2)

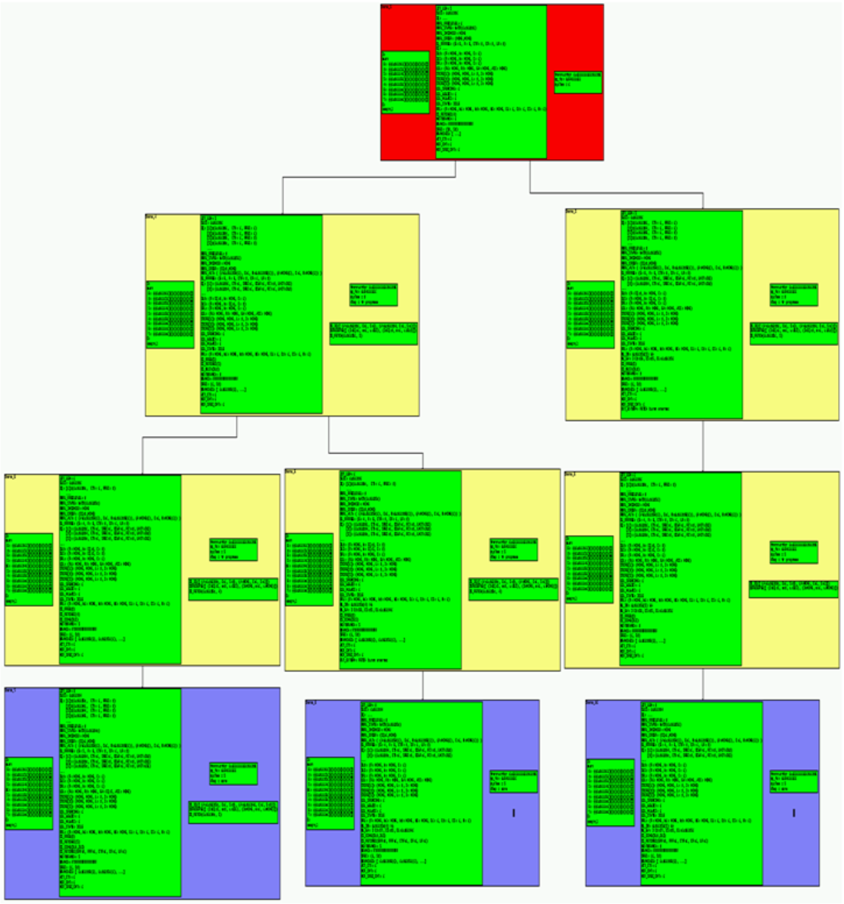
- Domain Abstraction
  - Abstractions approximate parts of the processor state by abstract values
  - Domains of signals/variables have to be changed
    - Example: Address ranges instead of exact addresses
  - Functors for changed types need to be adjusted
  - ➢ Identify all locations for needed functor adjustments for a given domain change

Transformation Tools

- DomainAbstracter
  - Automate type transformations on the model
  - Based on
    - Type transformation specification (e.g. Address → Address range)
    - Implementation for operators on the new domain
- DeadCodeEliminator
- 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

Code Generation

- Automatic generation of the cache/pipeline analysis out of the transformed and abstracted VHDL model
- Generated analysis perfectly fits into the aiT tool chain
- Generated code
  - 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



Conclusion

- Safety-critical systems with hard real-time constraints need a timing validation
- aiT is an industrial usable tool for the determination of safe and precise upper bounds on Worst-Case Execution Time of tasks
- 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
- VHDL specifications contain the timing behavior of the system
- The timing model can be semi-automatically derived out of such VHDL descriptions

➢ Removed human involvement up to a certain degree

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