GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

Towards Automation of Timing-Model Derivation

Markus Pister Marc Schlickling

AbsInt Angewandte Informatik GmbH

AbsInt Angewandte Informatik

Verisoft XT

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 is crucial for guaranteeing their correct behavior

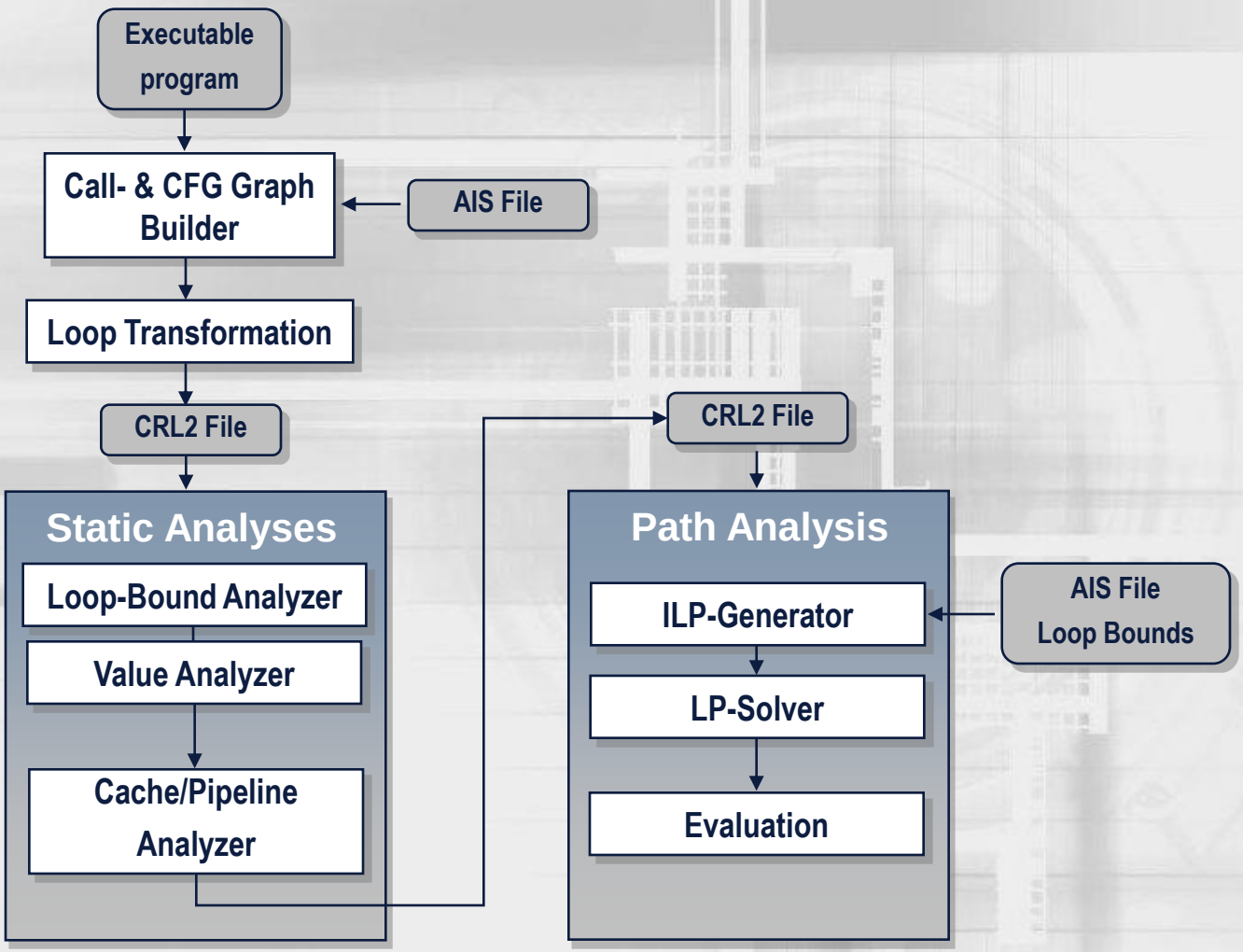
The Timing Problem

- Execution times of tasks vary over time
  - Between different runs
  - Different input

➢ Measurement of the worst-case execution time not possible for complex architectures

➢ Need static analysis (independent of inputs) for safety guarantees

aiT WCET Framework



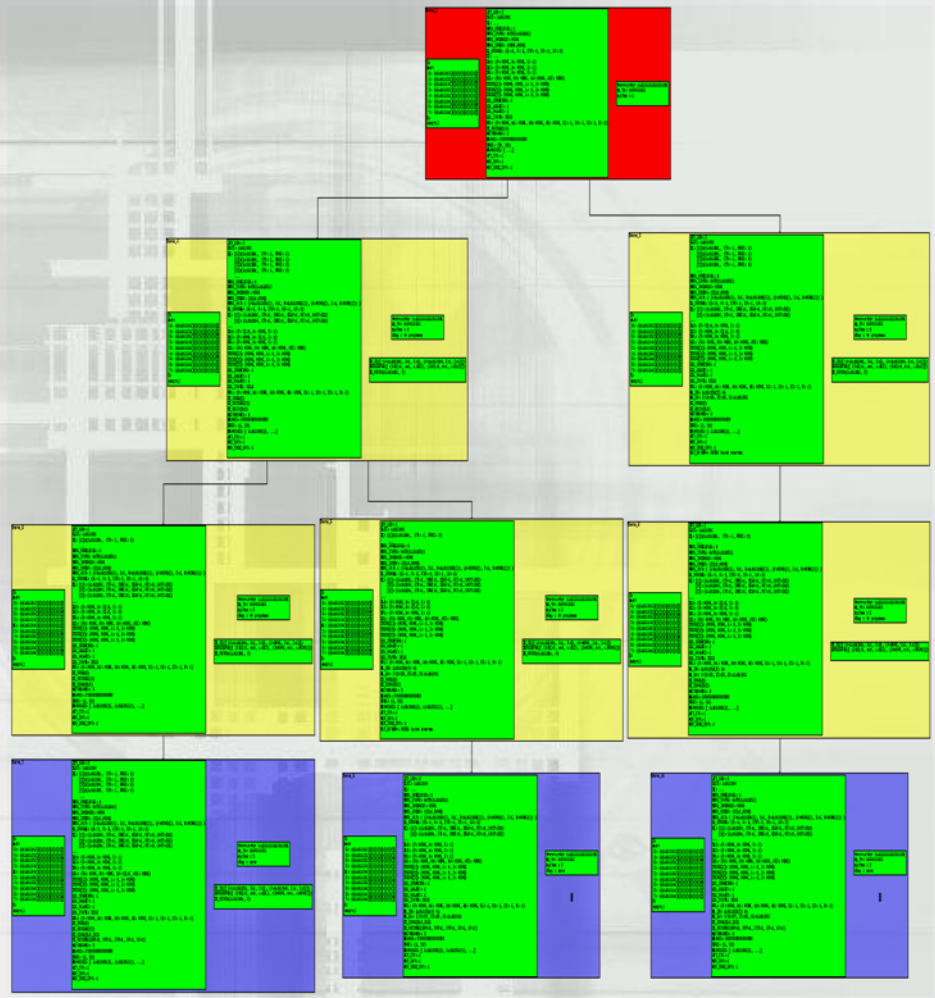
Caches / Pipelines

- 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

Cache / 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)



VHDL – Hardware Description Language

Sample 3-bit Counter in VHDL

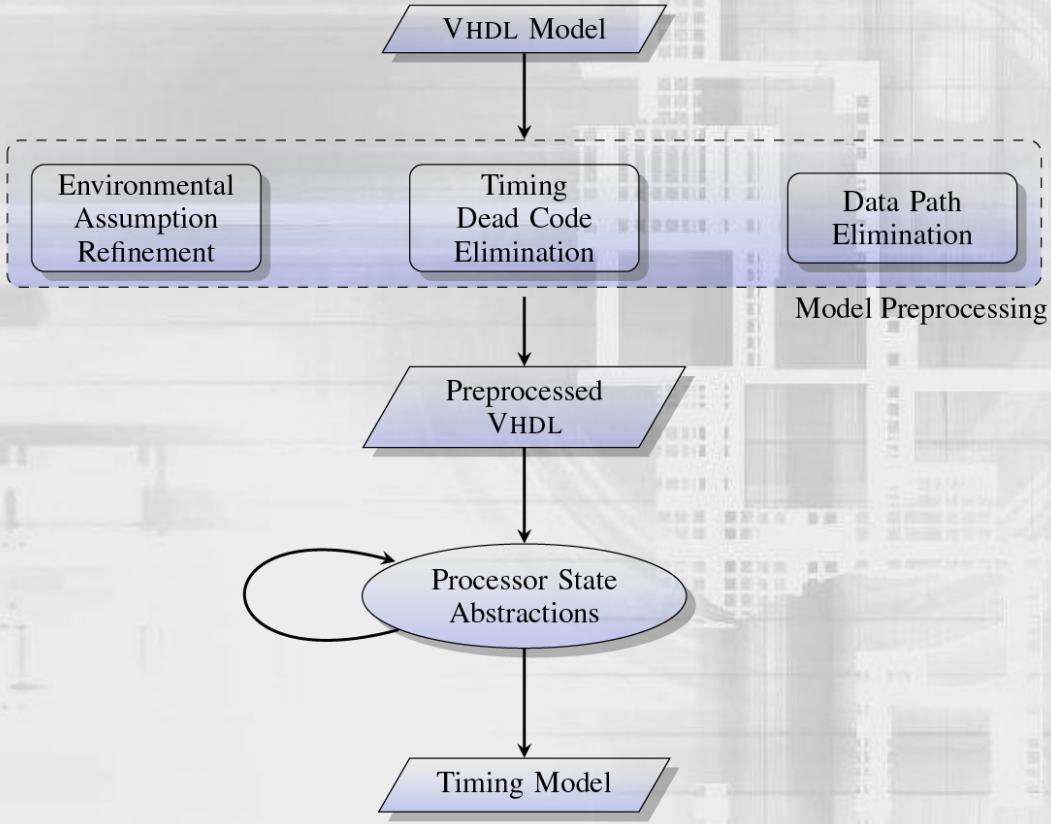
```
entity counter is
    port (clk:in std_logic; rst:in std_logic;
          val:out std_logic_vector (2 downto 0));
end;

architecture rtl of counter is
    signal cnt:std_logic_vector (2 downto 0);
begin
   P1: process (clk,rst) is
       if (rst='1') then
          cnt <= "000";
       elsif (rising_edge (clk)) then
          cnt <= cnt + '1';
       end if;
   end;
   P2: process (cnt) is
       val <= cnt;
   end;
end;
```

- Hierarchically organized
- Defined in the IEEE standard 1076
- Two-level semantics
  - Process execution
  - Synchronization + Restart + Time
- Process execution
  - Sequential, imperative semantics
  - Assignments to variables immediately take effect
  - Assignments to signals are delayed
  - Executes, until suspended
- Second level
  - After all processes have suspended
  - Check if restart of processes is necessary
    - Yes: restart these processes (delta cycle)
    - No: wait for timeout (theta cycle)
  - Repeat

Deriving the Timing Model

- Processor specification too large to be used in aiT framework
  PCP2 (~40.000 loc), Leon2 (~80.000 loc), 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)

Model preprocessing

- Methods (cont.)
  - 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.)

➢ Model size reduction by statically available information (without introducing nondeterminism so far)

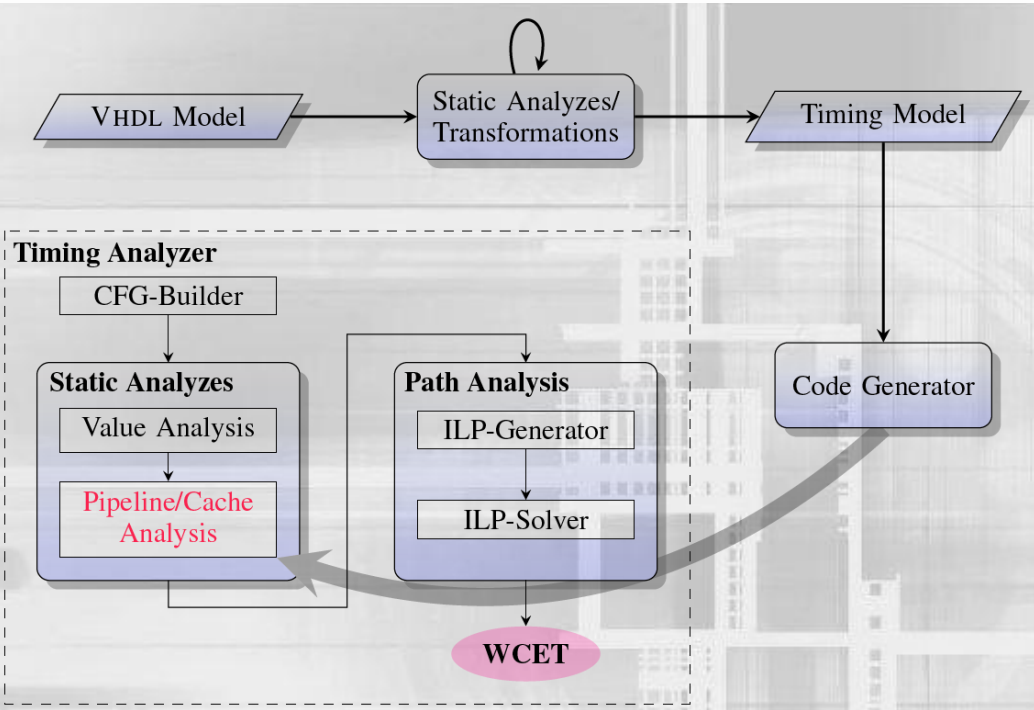
Processor State Abstractions

- 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
    - Example: Cache Abstraction
  - Domain Abstraction
    - Type changes
    - Example: Address → 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 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 values during system reset
  - Constant propagation to identify unused parts of the model
  - ➢ Forward slice “reset is activated”
  - ➢ Constant propagation on the result
  - ➢ Identify dead code

Example:

Assumption:

```
Signal s in {0,1}

if (s = `0` || s = `L`) then
  r <= NAME_ERROR;
elsif (s = `-`) then
  r <= MODE_ERROR;
elsif (s = `1`) then
  r <= OPEN_OK;
else
  r <= STATUS_ERROR;
endif;
```

→ **r in {NAME_ERROR, OPEN_OK}**

Static Analyses (2)

- Timing-Dead Code
  - Only parts of the model that affect timing behavior
  - ➢ Identify all locations where instructions leave the processor pipeline (can be more than one location)
  - ➢ Compute backward slices for each location
  - ➢ Remove all parts of the model not contained in the union over these slices
- 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 due to use of value analysis results
  - Functors for changed types need to be adjusted
  - ➢ Identify all locations for needed functor adjustments for a given domain change

Transformation Tools

- Domain-abstract
  - Automate type transformations on the model
  - Input
    - The model
    - Type transformation specification (e.g. Address → Address range)
    - Implementation for functors on the new domain
  - Incorporates the changes into the model
- Process-replace
  - Automate the replacement of VHDL processes
  - Input
    - The model
    - The name of the process to be removed
    - 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