Towards Model-Driven Development of Hard Real-Time Systems

Integrating ASCET and aiT/StackAnalyzer

Christian Ferdinand, Reinhold Heckmann
AbsInt Angewandte Informatik GmbH

Hans-Jörg Wolff, Christian Renz
ETAS GmbH

Oleg Parshin, Reinhard Wilhelm
Universität des Saarlandes
Outline

- ASCET
- Worst-case execution time analysis and stack usage analysis (aiT and StackAnalyzer)
- A lightweight integration
- Results
- Conclusion + future work
ASCET, aiT, StackAnalyzer

- **ASCET** is designed for **Model based Development Process** with **Automated Code Generation** considering **Automotive Terms and Requirements**

**ASCET Development Environment**

- **aiT** is designed to predict **safe and tight upper bounds of the execution time** (**WCET**) of safety-critical hard real-time tasks
- **StackAnalyzer** is designed to predict **safe and tight upper bounds of the maximum stack usage** of a task

**AbsInt Angewandte Informatik**

- Worst Case Execution Time
- Worst-Case Stack Usage
- Visualization, Documentation
Integration Goals

- Direct feedback on memory usage and execution time
- Optimize performance
- Establish and check memory and runtime quotas
ASCET

ASCET is focusing on ECU software development: modeling, simulation and implementation of function modules for the automated generation of reusable software components.
ASCET Programming methodology

ASCET is based on a model to separate necessary work steps in software engineering. Former working results remain functional while adding more information to the model.
ASCET

- Automated code generation with various optimizations
- Formal “verification”/experimental “verification”
- Supports distributed development
- Import/export of sub-models and (legacy) C-code, IP-protection
- Automotive life cycle support
- Automotive standards like ASAM-MSC2, -MDX, AUTOSAR, SERAP
- Certified IEC 61508-SIL 3
- Since 2000 used for mass production, large installation space (>1700 users), >20 Mio units per year
ASCET, Challenges for Timing Analysis

• Real-time-Architecture and Design
  ▪ Definition of Tasks and Scheduling
  ▪ ASCET comes with a built-in OSEK compliant real-time operating system

• Implementation Specification to adapt the SW-model to the specific Microcontroller
  ▪ Definition of bit resolution, limits, conversion formulas, memory locations, naming conventions...
  - focus on efficient implementation
aiT WCET Analyzer

- Combines global program analysis by abstract interpretation for cache, pipeline, and value analysis with integer linear programming for path analysis in a single intuitive GUI.
The Timing Problem

- The diagram illustrates the relationship between execution time and probability.
- The x-axis represents execution time, while the y-axis represents probability.
- The graph shows the best case execution time and its measurement.
- The unsafe execution time is indicated with red arrows, pointing towards the exact worst case execution time.
- The safe worst case execution time is highlighted with a blue arrow.

---

AbsInt
Angewandte Informatik
Embedded Control Software

- Tends to be large and complex
  - Much functionality
  - Code generator tools
  - 3rd party software
    - RTOS
    - communication libraries
The Timing Problem

\( x = a + b; \)

```
LOAD r2, _a
LOAD r1, _b
ADD r3, r2, r1
```

- **68K (1990)**
- **MPC 5xx (2000)**
- **PPC 755 (2001)**
Input/Output

Application Code

Compiler
Linker

Executable (*.elf / *.out)

Parameters (*.aip)

INTERPROC: vivu4
MEMORY_AREA: 0xFFFFF000 FFFF 1 READ&WRITE CODE&DATA

Specifications (*.ais)

Entry Point

- Worst Case Execution Time
- Visualization, Documentation
Stack Analyzer: Analysis Results

- Stack usage of a single function
  - local
  - global

- Call history from entry Point to function
aiT: Timing Details

**Worst Case Execution Time:** 4836

- **routine: _prime**
- **routine: _even**
- **routine: __MOD**
Challenge 1: Volatiles

```c
if (active) {
    ...
    dst_ptr = ...;
    adr_ptr = ...;
    end_dst_ptr = dst_ptr + noOfTransfers;
    while (dst_ptr < end_dst_ptr) {
        *dst_ptr++ = *adr_ptr++;
    }
}
```

- On some compiler tool chains volatile information is not included in the executable
- aiT works on executables
- ⇒ ASCET „declares“ volatiles in an .ais-file
Challenge 2: Synchronization Code

```c
while (_condition) {
    ...
}
```

- Busy-Wait on volatile variables
- Two solutions:
  - Use an annotation that `condition` is never true (system-wide scheduling analysis)
  - Ask the ASCET-user for an annotation
    - `/* snippet HERE is not analyzed and takes max 4800 cycles */`
Challenge 3: ASCET Interpolation Loops

function(m map) {
    ...
    for((map->ctr1->low = 1); (map->ctr1->low <= 3); (map->ctr1->low++)) {
        for((map->ctr2->low = 1); (map->ctr2->low <= 6); (map->ctr2->low++)) {
            /* code with memory accesses via pointers.... */
        }
    }
    ...

⇒ ASCET produces annotations:

loop "_abc" + 1 loop max 3;

loop "_abc" + 2 loop max 6;
Real-time tasks under Rubus OS on C16x taken from Volvo CE application
Integration with ETAS/ASCET

- aiT/StackAnalyzer are started from the ASCET-main-menu. ASCET generates all annotation files and the analyses are performed in the background.

- The .apf-files start the aiT tool for a detailed inspection of the analysis results.
Practical Experiments, Execution Time

<table>
<thead>
<tr>
<th>Procedure name</th>
<th>Measured</th>
<th>aiT</th>
<th>Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ANALOGIN16_IMPL_AdInterrupt</td>
<td>291</td>
<td>297</td>
<td>2.0%</td>
</tr>
<tr>
<td>_ANALOGIN16_IMPL_AnalogIn16</td>
<td>6</td>
<td>12</td>
<td>100.0%</td>
</tr>
<tr>
<td>_CONVERTER_IMPL_convert</td>
<td>26</td>
<td>32</td>
<td>23.0%</td>
</tr>
<tr>
<td>_DISTAB12_IMPL_measurement_a</td>
<td>263</td>
<td>289</td>
<td>9.9%</td>
</tr>
<tr>
<td>_DISTAB12_IMPL_measurement_b</td>
<td>263</td>
<td>289</td>
<td>9.9%</td>
</tr>
<tr>
<td>_DISTAB12_IMPL_measurement_c</td>
<td>263</td>
<td>289</td>
<td>9.9%</td>
</tr>
<tr>
<td>_PIDT1_MODULE_IMPL_normal</td>
<td>2980</td>
<td>3144</td>
<td>5.5%</td>
</tr>
<tr>
<td>_PIDT1_MODULE_IMPL_out</td>
<td>133</td>
<td>139</td>
<td>4.5%</td>
</tr>
<tr>
<td>_PWMOUT7_2_IMPL_PwmOut7_2</td>
<td>109</td>
<td>116</td>
<td>6.4%</td>
</tr>
<tr>
<td>_PWMOUT7_7_IMPL_PwmOut7_7</td>
<td>116</td>
<td>123</td>
<td>6.0%</td>
</tr>
</tbody>
</table>

Table 1. Comparison of maximal measured run-times and WCETs predicted by aiT (in cycles)

- Engine throttle control module specified in ASCET, Tasking compiler v7.5., STM ST10F269 microcontroller board. Run-times extracted from bus traces (ISYSTEMS ILA 128 logic analyzer)
- The worst-case path information provided by aiT was used to manually construct a corresponding input.
Practical Experiments: Stack Usage

<table>
<thead>
<tr>
<th>Procedure name</th>
<th>Simulated</th>
<th>StackAnalyzer</th>
<th>Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ANALOGIN16.IMPL_AdInterrupt</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_ANALOGIN16.IMPL_AnalogIn16</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_CONVERTER.IMPL_convert</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_DISTAB12.IMPL_measurement_a</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_DISTAB12.IMPL_measurement_b</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_DISTAB12.IMPL_measurement_c</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_PIDT1 MODULE.IMPL_normal</td>
<td>16/32</td>
<td>16/32</td>
<td>0.0%</td>
</tr>
<tr>
<td>_PIDT1 MODULE.IMPL_out</td>
<td>4/0</td>
<td>4/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_PWMOUT7.2.IMPL_PwmOut7.2</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
<tr>
<td>_PWMOUT7.7.IMPL_PwmOut7.7</td>
<td>0/0</td>
<td>0/0</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Table 2. Comparison of maximal simulated system/user stack usage and usage as predicted by StackAnalyzer (in bytes)

- **ST10/C16x** uses two stacks.
- Most generated functions neither use local variables nor call subroutines, i.e. the stack usage is zero.
Conclusion

- WCET and Stack Usage analysis for ASCET
  - Fully automatic
    - allows you to inspect the timing behavior of (the timing critical parts of) the generated code
    - ASCET automatically generates all annotations
  - The analysis results
    - are determined without the need to change the code (only additional annotations are produced by ASCET)
    - hold for all inputs and all executions
  - Precise
    - demonstrated by experiments
Additional Results

- Abhik Dey’s master thesis examined a complex lambda probe model
  - Comparison of the impact of different modeling techniques as well as compiler settings
  - Reduced the WCET by 87%
  - Not limited to hard real-time contexts

- Increases safety and saves development time.
Future

- Tighter integration
  - aiT and StackAnalyzer results accessible in the graphical model representation
  - ASCET has very precise information on the values → this can be used for even better precision

- Analysis of complete ERCOS\textsuperscript{EK} task

- System-level schedule analysis

- Compare aiT WCET results with measurements
  - Are the timings measured in the prototype environment typical average execution times? ...