Automotive Powertrain Demonstrator

Sebastian Kehr, Denso Automotive Deutschland GmbH, Eching
Agenda

• Demonstrator Setup
• Development Process
• Workflow
• Results & Outlook
Demonstrator Setup

- **Targets**
  - Automated *migration of legacy software* to a multicore system
  - Demonstration of an efficient tool chain
  - Demonstration of an efficient migration result

- **Use Case**
  - Diesel Engine Management System

**Diagram:**
- **Host PC**
- **Ethernet**
- **Debug/Trace Device**
- **AGBT/JTAG**
- **Platform**
- **AURIX TC399**

**Use Case Details:**
- Motor model
- Sensor data
- Trace/debug
Development Process for Multicore Migration

• Multicore migration from legacy application requires reverse engineering
  – Analysis of data dependencies
  – Timing analysis with trace
  – Documentation ...

• Right side of the V
  – Schedule feasibility
  – Race conditions
  – System timing ...

<table>
<thead>
<tr>
<th>REQ</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>System Architecture</td>
</tr>
<tr>
<td>SWA</td>
<td>Software Architecture</td>
</tr>
<tr>
<td>SWD</td>
<td>Software Design</td>
</tr>
<tr>
<td>SWI</td>
<td>Software Implementation</td>
</tr>
<tr>
<td>UT</td>
<td>Unit Test</td>
</tr>
<tr>
<td>IT</td>
<td>Integration Test</td>
</tr>
<tr>
<td>AT</td>
<td>Acceptance Test</td>
</tr>
</tbody>
</table>

Parallel Source Code
Use-Case-Specific Workflow

Workflow for Multicore Migration in UC5.2

- Host Tracing
  - Silexica (Automotive Flow: Analyze)
  - Elektrobit (Tresos Studio / AutoCore)
- Target Tracing
  - AbsInt (TimingProfiler + TimeWeaver)
  - Symtavision

- AbsInt / Timing Architects (APP4MC)
- Silexica (Automotive Flow: Optimize)
- Timing Architects (TA Simulator, TA Optimizer)
- Denso (Parcus)
- Elektrobit (Tresos Studio / AutoCore)
- Silexica (Automotive Flow: Implement)
- TU Braunschweig
- Uni Kiel (Lodin)
- Accemic (CEDAR)
- Uni Lübeck (TeSSLa)
- Fraunhofer IESE (FERAL framework)
Workflow: Code Analysis & Modelling

- **Host tracing**
  - Runtime environment: AUTOSAR on Windows
  - Static and dynamic dependency analysis with compiler-based analysis technology

- **Target tracing**
  - Same configuration like host tracing
  - Traces from execution on target to derive accurate WCET with hybrid WCET analysis

- **Output: application model**
  1. Data dependencies between runnables
  2. Worst-case latency of end-to-end timing chains
  3. Worst-case execution time (WCET) of runnables and tasks

AMALTHEA model
Workflow: Parallelization

1. Runnable-level parallelization: Each task is split into multiple parallel running tasks
   - Data dependencies and WCET are considered
   - Start and termination of split tasks are synchronised

2. Task-level parallelization: Tasks are distributed to cores
   - Data distribution according to the logical execution time (LET) concept

3. Energy Optimization: Combines approach 1 & 2
   - Considers impact on latency and the processor frequency
   - Parallel schedule quality (PSQ) metric quantifies success of parallelization
Workflow: Integration & Verification

• Generate code for AUTOSAR BSW + RTE
  – Add synchronisation primitives and/or communication buffers

• Verification
  – Comparison of timing between single-core and multicore software
  – Offline
    • Offline evaluation of measurements
    • Concurrency bugs
  – Online
    • Measurement on target and real-time evaluation with specialised FPGA
Results & Outlook

• Workflow / interoperability
  – Tool interoperability based on AMALTHEA model

• Partial automated migration
  – Runnable-level parallelization incl. memory mapping

• Next steps
  – Full automation of runnable-level parallelization
  – Detailing of verification methodology
  – Integration of LET concept and its automation
Thank you for your attention!