Design Level Debugging of Timing Behavior in Embedded Systems: Using a Model-Based Approach

Padma Iyenghar, Clemens Westerkamp, Juergen Wuebbelmann & Elke Pulvermueller

Project - funded by

Bundesministerium für Wirtschaft und Technologie

Federal Ministry of Economics & Technology (BMWi), Germany

Industrial Partner

Willert Software Tools GmbH
Outline

1. Introduction
2. Model-Based Debugging Approach - Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Outline

1. Introduction
2. Model-Based Debugging Approach - Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Introduction

- Embedded Systems – Ubiquitous
  - Memory size, speed and real-time constraints

- Traditional approach
  - Debugging methods & tools

- Model Driven Architecture (e.g. UML diagrams in design model)
  - Model Driven Development (MDD)
  - Design-level debugging (visualize target behavior) → UML timing diagram

Design-Level Debugging of Timing Behavior in Embedded Systems: Using a Model-Based Approach
• Embedded System
  – Duration constraint and timing information: significant
  – Timing Diagrams: de-facto standard
Introduction

- **Existing approach**: Dynamic source code instrumentation
  - E.g. Rhapsody (MDD tool): suitable for large systems
  - Possibility to influence real-time behavior
  - Debugging using UML timing diagrams: Not available

- **Required**: Robust model-based debugging with MDD
  - Minimal overhead → Possibility to leave debug code in production code
  - Visualize target behavior in real-time even for small platforms

- **Proposed**: Design-level debugging approach
  - Minimally intrusive
  - UML timing diagram
Outline

1. Introduction
2. Model-Based Debugging Approach - Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Model-Based Debugging Approach - Concept (1)

- Model Driven Development (MDD)
  - Requirements specification
  - Design Model
- Automatic Code Generation
- Target Debugger GUI (Model-Based)
  - [Run-time interaction and behavior analysis]
- Target side
  - Real-Time Embedded Target (RTOS)
  - Target Monitor
- Host side
  - Debug Interface
  - Based on back-annotated data from target

- Design level
- Code level

Deploy interface
Outline

1. Introduction
2. Model-Based Debugging Approach – Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Model-Based Debugging Approach – Prototype

Model Driven Development (MDD)
- Requirements
  - Design Model (MDD tool)
  - Automatic Code Generation (MDD tool)

Model-based Debugging
- Model-based Target Debugger GUI
  - [UML Sequence & Timing diagrams] (Qt)

Debug Interface
- XML file
- Linker map file
- Source files
- AWK script

Deployment interface

Real-Time Embedded System
- (Real-Time eXecution Framework [RXF])

Target Monitor
- (C Language)

Custom-defined frame format for notifications
- e.g – Event consumed notification: |15(len)09(id)
  | (94060010-event) (C0060010-source) (11400000-dest)
  | (84030000-curr.time) (02-state) (01011010-evParam)|
Outline

1. Introduction
2. Model-Based Debugging Approach - Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Illustrative Example – MIDI System Analyzer

MIDI Keyboard

MIDI Interface

RS-232

Deploy interface

JTAG

RS-232

Host Computer

RS-232

Debug interface

Evaluation Board (Cortex M3)
MIDI System Case Study – Results

Target Debugger – MIDI system behavior

UML Sequence Diagram

UML Timing Diagram

Design-Level Debugging of Timing Behavior in Embedded Systems: Using a Model-Based Approach
Outline

1. Introduction
2. Model-Based Debugging Approach – Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Performance Overhead

(1) Memory Footprint in Embedded System

Memory overhead for existing vs proposed approach

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
</tr>
</tbody>
</table>

Legend

- Existing Approach (MBT Tool)
- Proposed approach

Application Scenario (increasing size and complexity)

(2) Time Spent in Target Monitor Routine

- Time taken to send trace data to host computer
  - Example: Longest frame size → Event consumed notification (23 bytes)
  - Time spent in monitor: 50μs (measured using logic analyzer)
Outline

1. Introduction
2. Model-Based Debugging Approach - Concept
3. Model-Based Debugging – Prototype
4. Illustrative Example
5. Performance metrics
6. Summary and Conclusion
Summary and Conclusion

- Model-based, design level debugging approach
  - Target Monitor with static (constant) overhead
    - Memory size (approx. 1 Kbyte) accommodative for small targets
    - Time (inside monitor) known before hand - can be included in system design
    - Bundle with production code (end user‘s decision)
    - Opportunity to debug small targets at design level (UML)

- Future Work: Support for additional target platforms

- Further Application(s): Deploying/executing Model-Based Testing (MBT) in small (resource-constrained) embedded targets.
Thank You!

Questions

Contact

Padma Iyenghar\textsuperscript{1,2}, Clemens Westerkamp\textsuperscript{1}, Juergen Wuebbelmann\textsuperscript{1} & Elke Pulvermueller\textsuperscript{2}

1- University of Applied Sciences, Osnabrueck, Germany
2- University of Osnabrueck, Germany

\{piyengha, elke.pulvermueller\}@uos.de
\{c.westerkamp, j.wuebbelmann\}@hs-osnabrueck.de

Willert Software Tools GmbH
http://www.willert.de