eMOTE: A Real-time Approach to Model-based Testing of Embedded Software

Dr. Philipp Graf
FZI Forschungszentrum Informatik, Karlsruhe

19.10.2011
Agenda

- Testing of Embedded Software: The Project eMOTE
- From Unit-Testing to Model-Based Testing
- Hardware-support for Grey- und White-Box Testing
Embedded Software

- Innovation in Embedded Systems increasingly driven by Embedded Software
  - Often safety relevant, updates expensive
  - Software-QA vital, testing the most important tool
The Project eMOTE

- Generation of test cases from a test model
- Test execution directly on the target microcontroller
- No influence on run-time behavior or target code
- Testing advanced properties like timing, code coverage
What are the main project goals?

- **Non-invasive testing**
  - Test-cases should not influence temporal run-time behavior of program execution.

- **Testing of time constraints**
  - Annotate time constraints and verify them during test execution.

- **Linking test methodology with code-coverage**
  - Information about completeness of the test strategy based on measuring code-coverage.

- **Combining external data sources with software tests**
  - Including Hardware-in-the-Loop approaches allows mixed test of software and electronic signals.
Testing: Manually and Model-Based

- **Manual programming of test case**
- **Modeling of test cases**
- **Test model and test case generation**

*test engineer/developer*

*test model*

*description/model test case*

*executable test case*
eMOTe Architecture

test case modeling

Eclipse-plugin for generating test-cases from UML-specification

Eclipse: integration platform

SUT (Source Code)

Eclipse-Plugin: Run-time for test-cases

visualization

WinIDEA 2010 Integrated Development Environment

*.xmi

*.xmi

*.trd

isystem. Java.SDK
Test-Execution in Embedded Systems: Technical Specifics

- Tests can be executed on…
  - development computer
  - Plattform and periphery have to be emulated
  - target platform (mikrocontroller, SoC,…)
  - Integration and test-execution more complex
  - Testing of boundary cases can be impossible

- Aspects
  - Real-time
  - Communication with periphery / external chips
  - Often hard to bring system/software into initial state
  - Often strongly coupled software-components (performance!)
Hardware-Based Debugging-Tools

- **Debugger**
  - Tool for finding, diagnosing and eliminating defects in software
  - Allows run-control and inspection of program state

- **Hardware-based tools for Embedded Systems**
  - Access via processor interfaces or emulation

- **Trace**
  - Recording program flow and changes to data (variables, registers,…) without influencing timing
  - Analyse recording later
Using Hardware-Debuggers as Testing Tools

Basic idea: Use a debugger-API for automatically running tests of embedded software.
Approach for Grey-Box Testing

1. Bring software into defined state
2. Trigger test execution
3. Read/Reconstruct results and global variables for evaluating tests

- Use run-control and read-out of variables/memory/registers
Case Study: Headlamp / Cornering Lights
Case Study: Workflow in eMOTE

- func: [doBlink, [0,0,0,1], rv]
  expect:
  - blinkType == ${_blinkType}
  - blinkRestart == 0

isystem.connect

passed/failed
Next Step: White-Box Test

- Detailed recording of events (control flow, data flow) when executing a test case (tracing)
  - Include recorded events in test evaluation

- Leads to interesting scenarios:
  - Recursive calling-sequences of (sub-)functions
    - Considering order, point in time and parameters
  - Following variables and input-/output-signals over time
    - Recording and testing „signal sequences“
  - Properties and performance of the embedded operating systems
White-Box Test
Test Execution (Calling-Sequences)

1. generate
2. configure
3. test execution
4. record
5. reconstruct
6. compare

- test model
- system - stimulation/expected flow
- trace-data
- reconstructed program flow

19.10.2011

© FZI Forschungszentrum Informatik
Thank you!