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

Dr. Philipp Graf

FZI Forschungszentrum Informatik, Karlsruhe

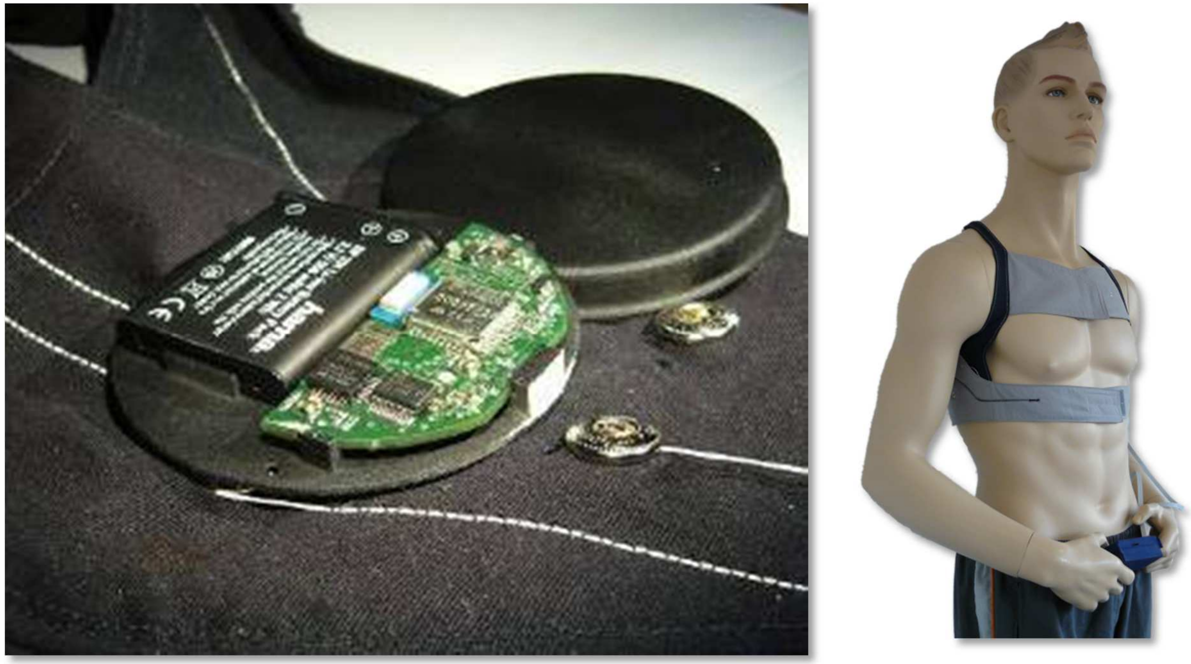
19.10.2011

FZI FORSCHUNGSZENTRUM INFORMATIK

Agenda

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

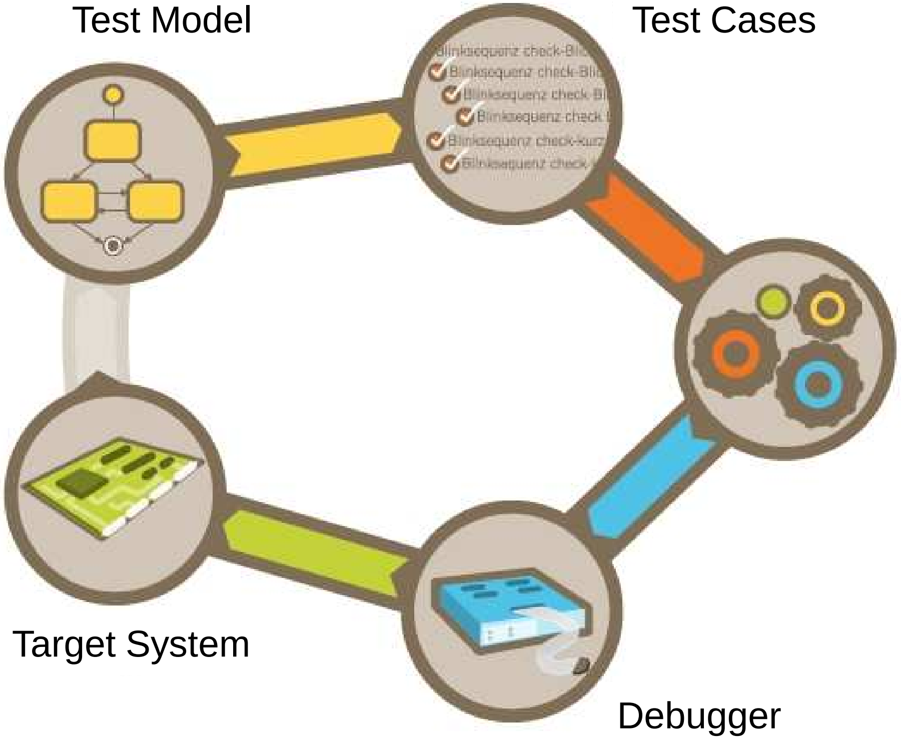
Embedded Software



Textile-integrated ECG and wearable defibrillator

- 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



Test Model

Test Cases

eMOTE
run-time environment

Target System

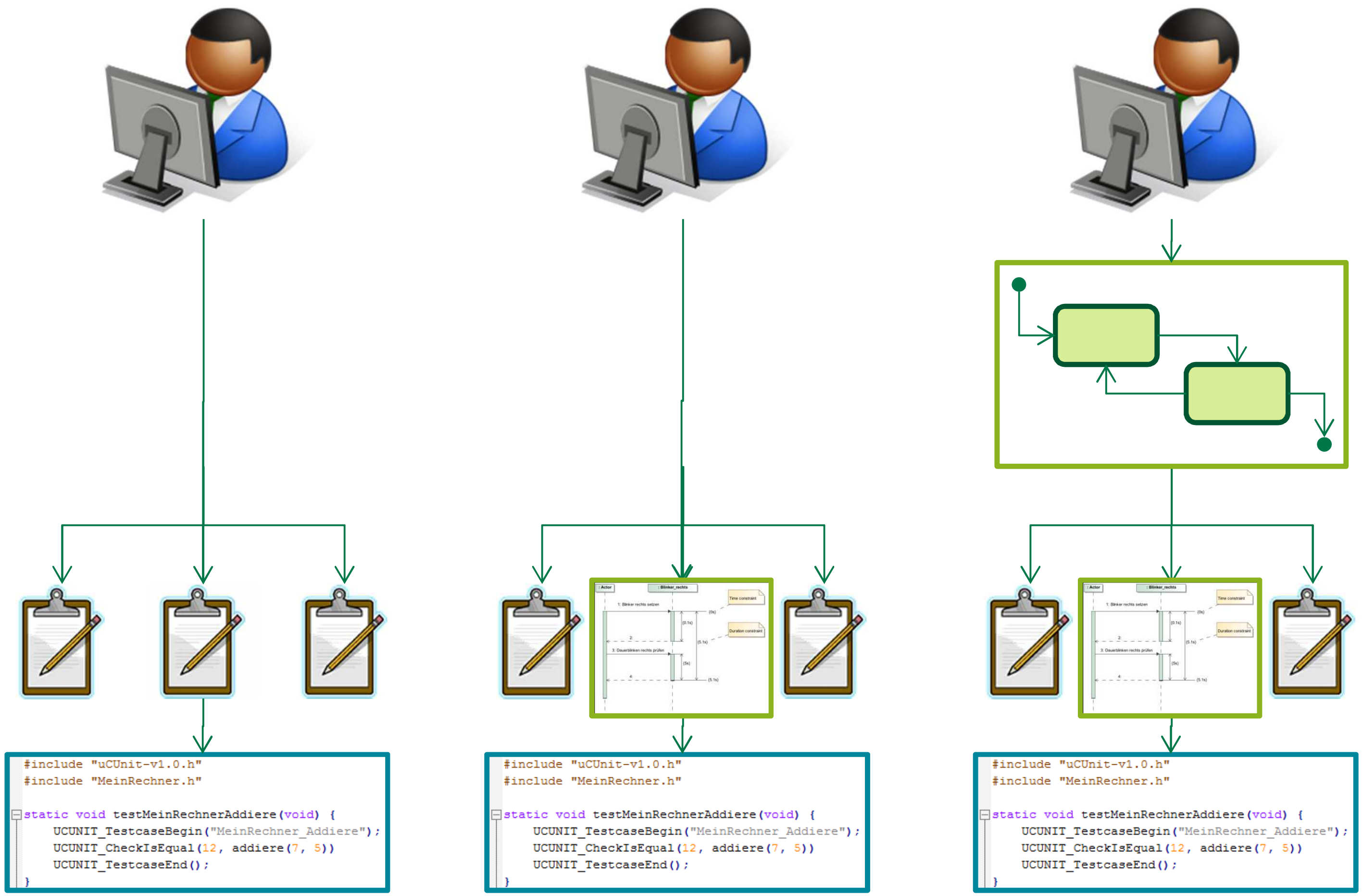
Debugger

- 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
  - Anotate time constraints and verify them during test execution.
- Linking test methodology with code-coverage
  - Information about completenes 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



```
#include "uCUnit-v1.0.h"
#include "MeinRechner.h"

static void testMeinRechnerAddiere(void) {
    UCUNIT_TestcaseBegin("MeinRechner_Addiere");
    UCUNIT_CheckIsEqual(12, addiere(7, 5))
    UCUNIT_TestcaseEnd();
}
```

```
#include "uCUnit-v1.0.h"
#include "MeinRechner.h"

static void testMeinRechnerAddiere(void) {
    UCUNIT_TestcaseBegin("MeinRechner_Addiere");
    UCUNIT_CheckIsEqual(12, addiere(7, 5))
    UCUNIT_TestcaseEnd();
}
```

```
#include "uCUnit-v1.0.h"
#include "MeinRechner.h"

static void testMeinRechnerAddiere(void) {
    UCUNIT_TestcaseBegin("MeinRechner_Addiere");
    UCUNIT_CheckIsEqual(12, addiere(7, 5))
    UCUNIT_TestcaseEnd();
}
```

test engineer/
developer

test model

description/
model test case

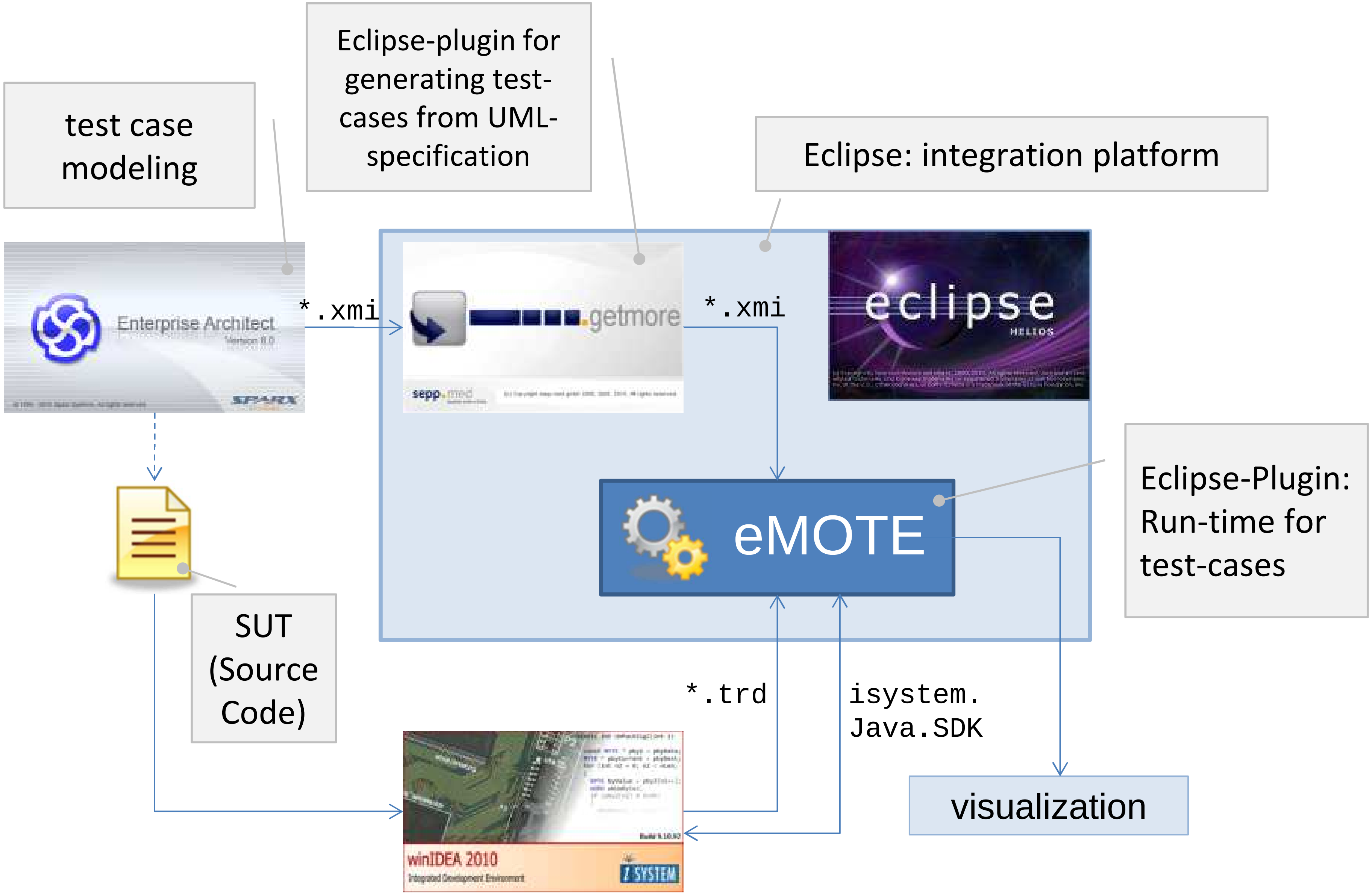
executable
test case

manual programming
of test case

modeling of
test cases

test model and
test case generation

eMOTE Architecture



test case modeling

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

Eclipse: integration platform

*.xmi

*.xmi

eMOTE

Eclipse-Plugin: Run-time for test-cases

SUT (Source Code)

*.trd

isystem.
Java.SDK

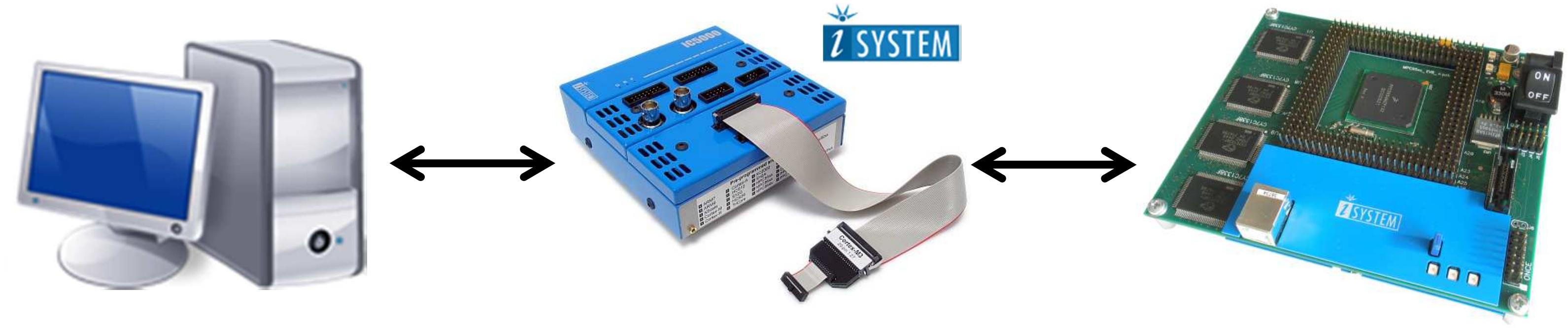
visualization

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

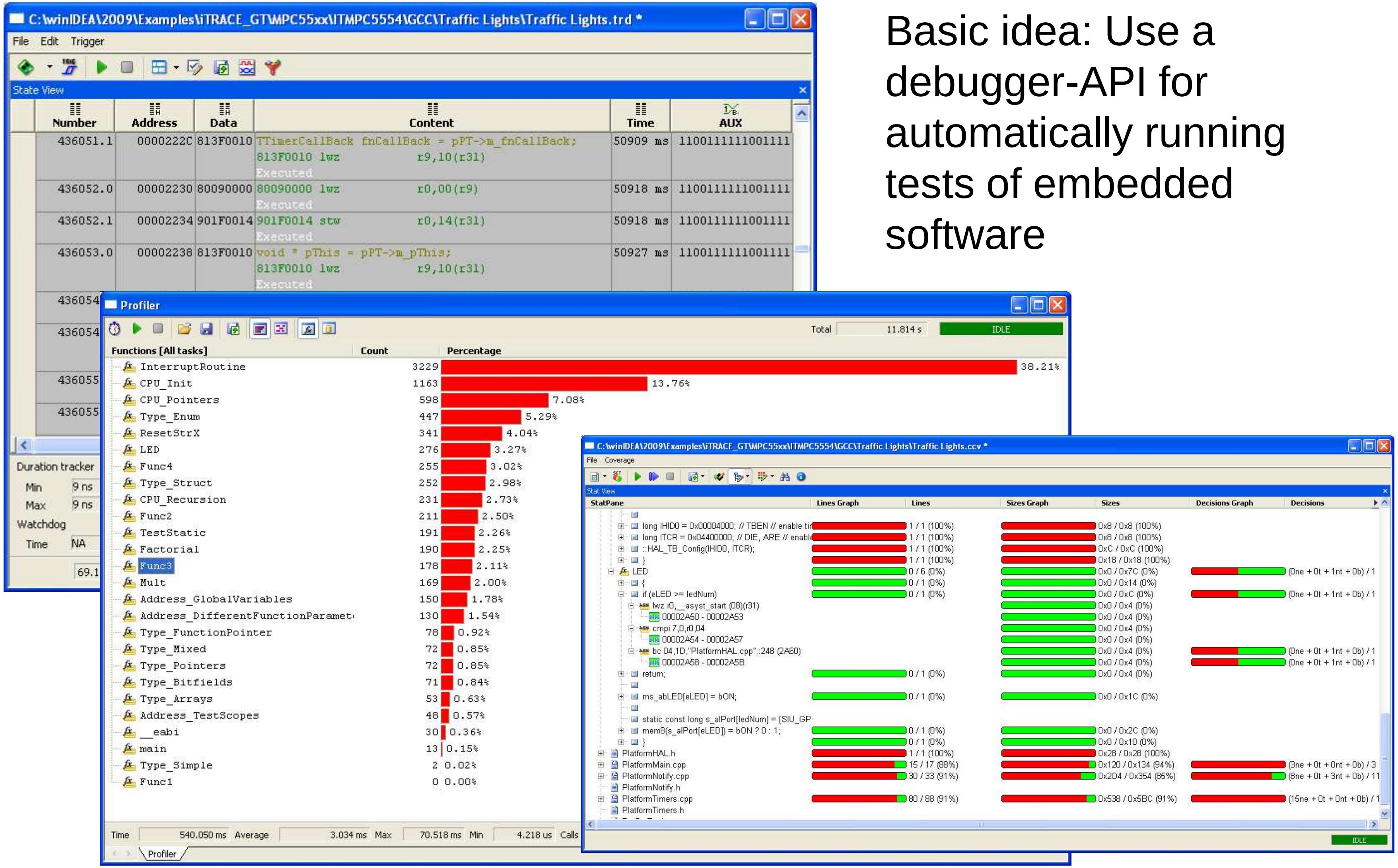
- Aspects
  - Real-time
  - Communication with periphery / external chips
  - Often hard to bring system/software into initial state
  - Often strongly coupled software-components (performance!)

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



C:\winIDEA\2009\Examples\iTRACE_GT\MPC55xx\ITMPC5554\GCC\Traffic Lights\Traffic Lights.trd *

Number	Address	Data	Content	Time	AUX
436051.1	0000222C	813F0010	TTimerCallBack fnCallBack = pPT->m_fnCallBack; 813F0010 lwz r9,10(r31) Executed	50909 ms	1100111111001111
436052.0	00002230	80090000	80090000 lwz r0,00(r9) Executed	50918 ms	1100111111001111
436052.1	00002234	901F0014	901F0014 stw r0,14(r31) Executed	50918 ms	1100111111001111
436053.0	00002238	813F0010	void * pThis = pPT->m_pThis; 813F0010 lwz r9,10(r31) Executed	50927 ms	1100111111001111
436054					
436054					
436055					
436055					

Duration tracker

Min 9 ns

Max 9 ns

Watchdog

Time NA

69.1

Profiler

Total 11.814 s IDLE

Functions [All tasks]	Count	Percentage
InterruptRoutine	3229	38.21%
CPU_Init	1163	13.76%
CPU_Pointers	598	7.08%
Type_Enum	447	5.29%
ResetStrX	341	4.04%
LED	276	3.27%
Func4	255	3.02%
Type_Struct	252	2.98%
CPU_Recursion	231	2.73%
Func2	211	2.50%
TestStatic	191	2.26%
Factorial	190	2.25%
Func3	178	2.11%
Mult	169	2.00%
Address_GlobalVariables	150	1.78%
Address_DifferentFunctionParamet	130	1.54%
Type_FunctionPointer	78	0.92%
Type_Mixed	72	0.85%
Type_Pointers	72	0.85%
Type_Bitfields	71	0.84%
Type_Arrays	53	0.63%
Address_TestScopes	48	0.57%
__eabi	30	0.36%
main	13	0.15%
Type_Simple	2	0.02%
Func1	0	0.00%

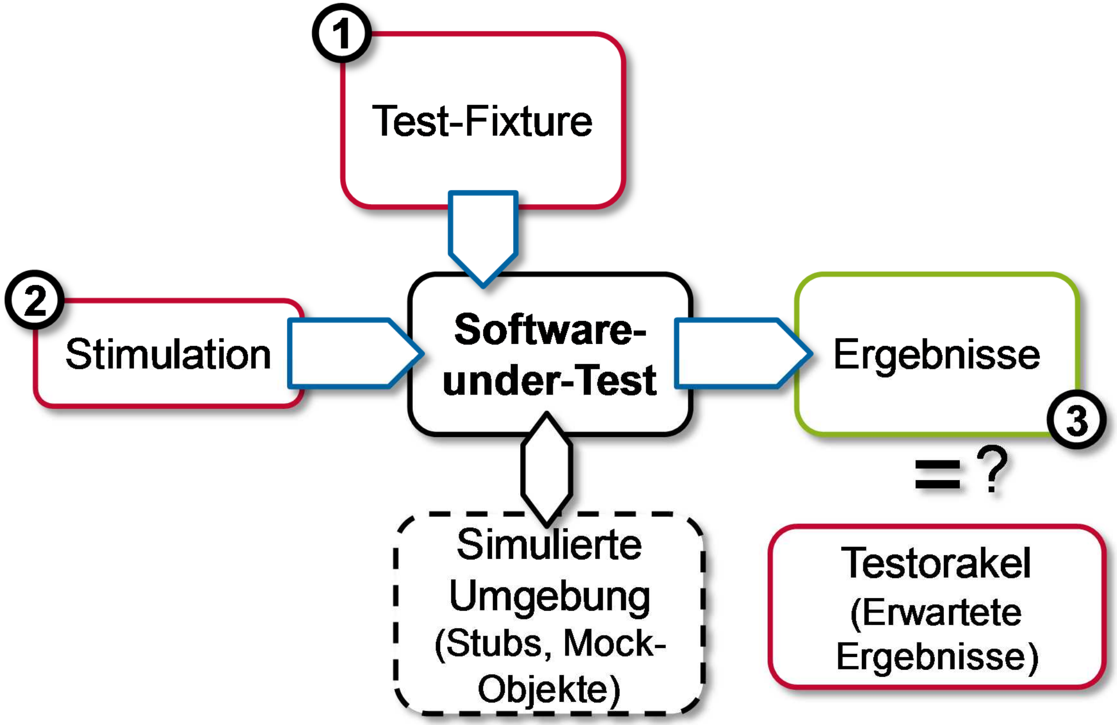
Time 540.050 ms Average 3.034 ms Max 70.518 ms Min 4.218 us Calls

C:\winIDEA\2009\Examples\iTRACE_GT\MPC55xx\ITMPC5554\GCC\Traffic Lights\Traffic Lights.ccv *

StatPane	Lines Graph	Lines	Sizes Graph	Sizes	Decisions Graph	Decisions
long IHID0 = 0x00004000; // TBEN // enable ti	🟥	1 / 1 (100%)	🟥	0x8 / 0x8 (100%)		
long ITCR = 0x04400000; // DIE, ARE // enable	🟥	1 / 1 (100%)	🟥	0x8 / 0x8 (100%)		
::HAL_TB_Config(IHID0, ITCR);	🟥	1 / 1 (100%)	🟥	0xC / 0xC (100%)		
}	🟥	1 / 1 (100%)	🟥	0x18 / 0x18 (100%)		
LED	🟩	0 / 6 (0%)	🟩	0x0 / 0x7C (0%)	🟥🟩	(0ne + 0t + 1nt + 0b) / 1
{	🟩	0 / 1 (0%)	🟩	0x0 / 0x14 (0%)		
if (eLED >= ledNum)	🟩	0 / 1 (0%)	🟩	0x0 / 0xC (0%)	🟥🟩	(0ne + 0t + 1nt + 0b) / 1
lwz r0,__asyst_start (08)(r31)			🟩	0x0 / 0x4 (0%)		
00002A50 - 00002A53			🟩	0x0 / 0x4 (0%)		
cmpi 7,0,r0,04			🟩	0x0 / 0x4 (0%)		
00002A54 - 00002A57			🟩	0x0 / 0x4 (0%)		
bc 04,1D,"PlatformHAL.cpp"::248 (2A60)			🟩	0x0 / 0x4 (0%)	🟥🟩	(0ne + 0t + 1nt + 0b) / 1
00002A58 - 00002A5B			🟩	0x0 / 0x4 (0%)	🟥🟩	(0ne + 0t + 1nt + 0b) / 1
return;	🟩	0 / 1 (0%)	🟩	0x0 / 0x4 (0%)		
ms_abLED[eLED] = bON;	🟩	0 / 1 (0%)	🟩	0x0 / 0x1C (0%)		
static const long s_alPort[ledNum] = {SIU_GP						
mem8(s_alPort[eLED]) = bON ? 0 : 1;	🟩	0 / 1 (0%)	🟩	0x0 / 0x2C (0%)		
}	🟩	0 / 1 (0%)	🟩	0x0 / 0x10 (0%)		
PlatformHAL.h	🟥	1 / 1 (100%)	🟥	0x28 / 0x28 (100%)		
PlatformMain.cpp	🟥🟩	15 / 17 (88%)	🟥🟩	0x120 / 0x134 (94%)	🟥	(3ne + 0t + 0nt + 0b) / 3
PlatformNotify.cpp	🟥🟩	30 / 33 (91%)	🟥🟩	0x2D4 / 0x354 (85%)	🟥🟩	(8ne + 0t + 3nt + 0b) / 11
PlatformNotify.h						
PlatformTimers.cpp	🟥🟩	80 / 88 (91%)	🟥🟩	0x538 / 0x5BC (91%)	🟥	(15ne + 0t + 0nt + 0b) / 1
PlatformTimers.h						

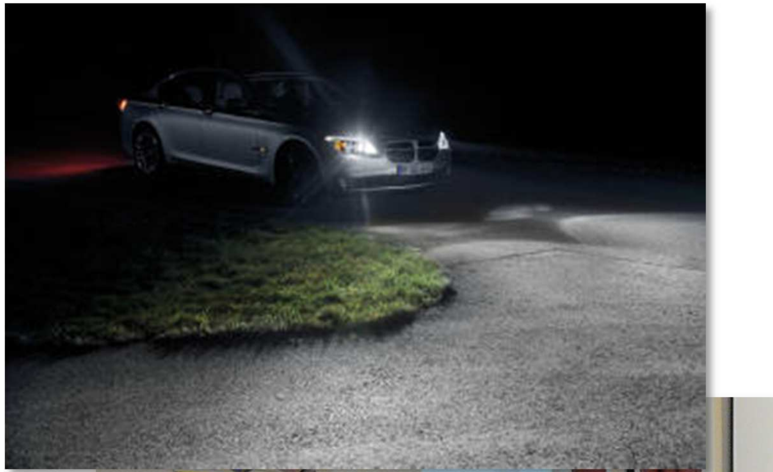
Basic idea: Use a debugger-API for automatically running tests of embedded software

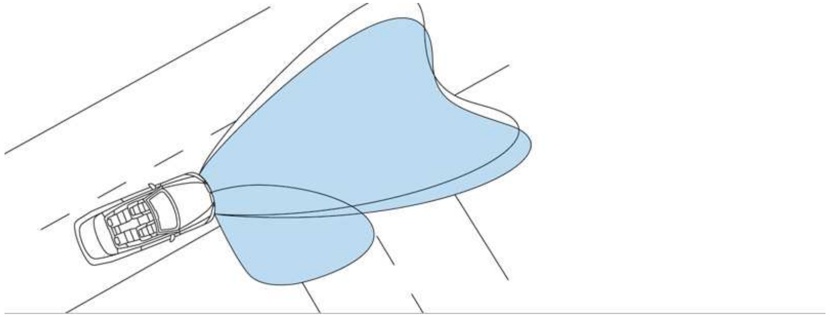
Apprach for Grey-Box Testing

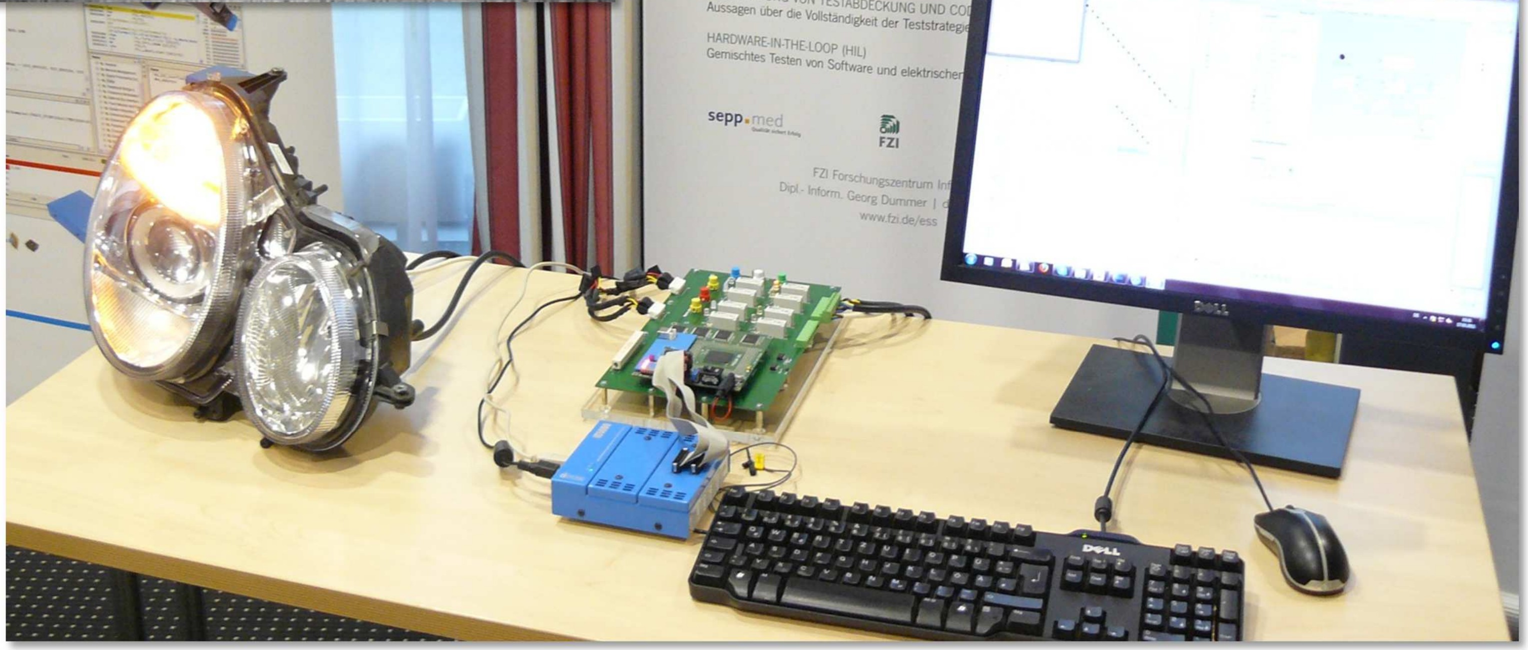

1
Test-Fixture
2
Stimulation
Software-under-Test
Ergebnisse
3
=?
Simulierte Umgebung (Stubs, Mock-Objekte)
Testorakel (Erwartete Ergebnisse)

- Use run-control and read-out of variables/memory/registers
  1. Bring software into defined state
  2. Trigger test excution
  3. Read/Reconstruct results and global variables for evaluationg tests

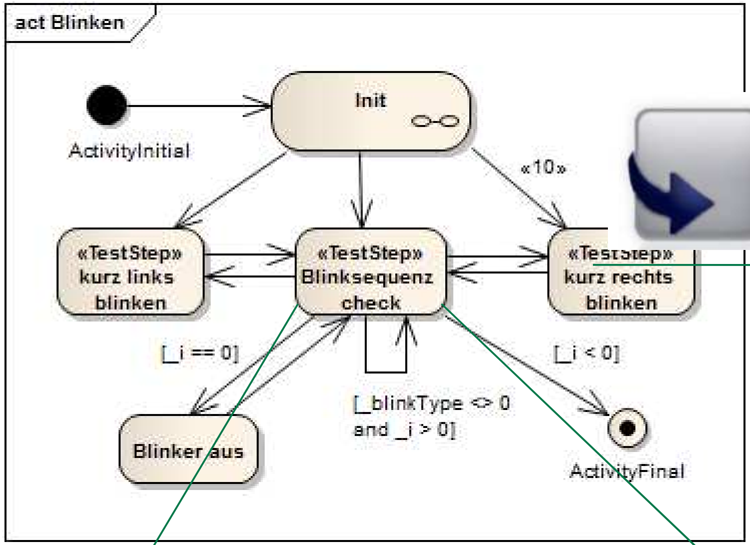
Case Study: Headlamp / Cornering Lights



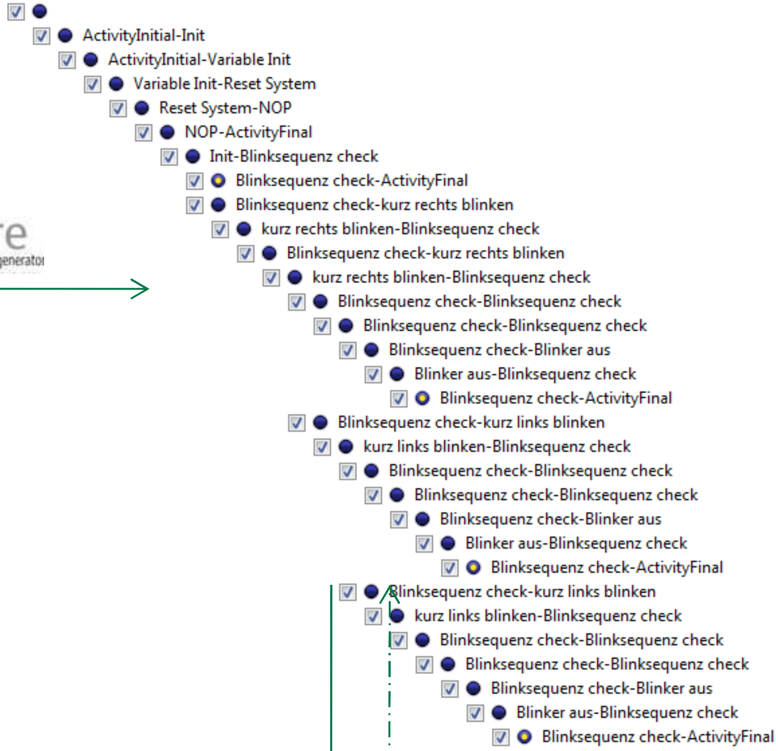



TESTEN VON ECHTZEITBEDINGUNGEN
Berücksichtigung zeitlicher Randbedingungen beim
VERKNÜPFUNG VON TESTABDECKUNG UND COD
Aussagen über die Vollständigkeit der Teststrategie
HARDWARE-IN-THE-LOOP (HIL)
Gemischtes Testen von Software und elektrischen
sepp.med
FZI
FZI Forschungszentrum Inf
Dipl.- Inform. Georg Dummer | d
www.fzi.de/ess
DELL

Case Study: Workflow in eMOTE

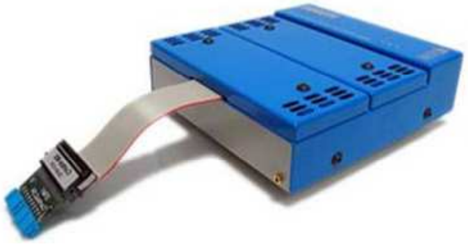

act Blinken
Init
ActivityInitial
«10»
«TestStep» kurz links blinken
«TestStep» Blinksequenz check
«TestStep» kurz rechts blinken
[_i == 0]
[_i < 0]
[_blinkType <> 0 and _i > 0]
Blinker aus
ActivityFinal

.getmore
Der Testfallgenerator



```
☑ ●
  ☑ ● ActivityInitial-Init
    ☑ ● ActivityInitial-Variable Init
      ☑ ● Variable Init-Reset System
        ☑ ● Reset System-NOP
          ☑ ● NOP-ActivityFinal
            ☑ ● Init-Blinksequenz check
              ☑ ○ Blinksequenz check-ActivityFinal
              ☑ ● Blinksequenz check-kurz rechts blinken
                ☑ ● kurz rechts blinken-Blinksequenz check
                  ☑ ● Blinksequenz check-kurz rechts blinken
                    ☑ ● kurz rechts blinken-Blinksequenz check
                      ☑ ● Blinksequenz check-Blinksequenz check
                        ☑ ● Blinksequenz check-Blinksequenz check
                          ☑ ● Blinksequenz check-Blinker aus
                            ☑ ● Blinker aus-Blinksequenz check
                              ☑ ○ Blinksequenz check-ActivityFinal
                      ☑ ● Blinksequenz check-kurz links blinken
                        ☑ ● kurz links blinken-Blinksequenz check
                          ☑ ● Blinksequenz check-Blinksequenz check
                            ☑ ● Blinksequenz check-Blinksequenz check
                              ☑ ● Blinksequenz check-Blinker aus
                                ☑ ● Blinker aus-Blinksequenz check
                                  ☑ ○ Blinksequenz check-ActivityFinal
                          ☑ ● Blinksequenz check-kurz links blinken
                            ☑ ● kurz links blinken-Blinksequenz check
                              ☑ ● Blinksequenz check-Blinksequenz check
                                ☑ ● Blinksequenz check-Blinksequenz check
                                  ☑ ● Blinksequenz check-Blinker aus
                                    ☑ ● Blinker aus-Blinksequenz check
                                      ☑ ○ Blinksequenz check-ActivityFinal
```

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



isystem.connect

passed/failed

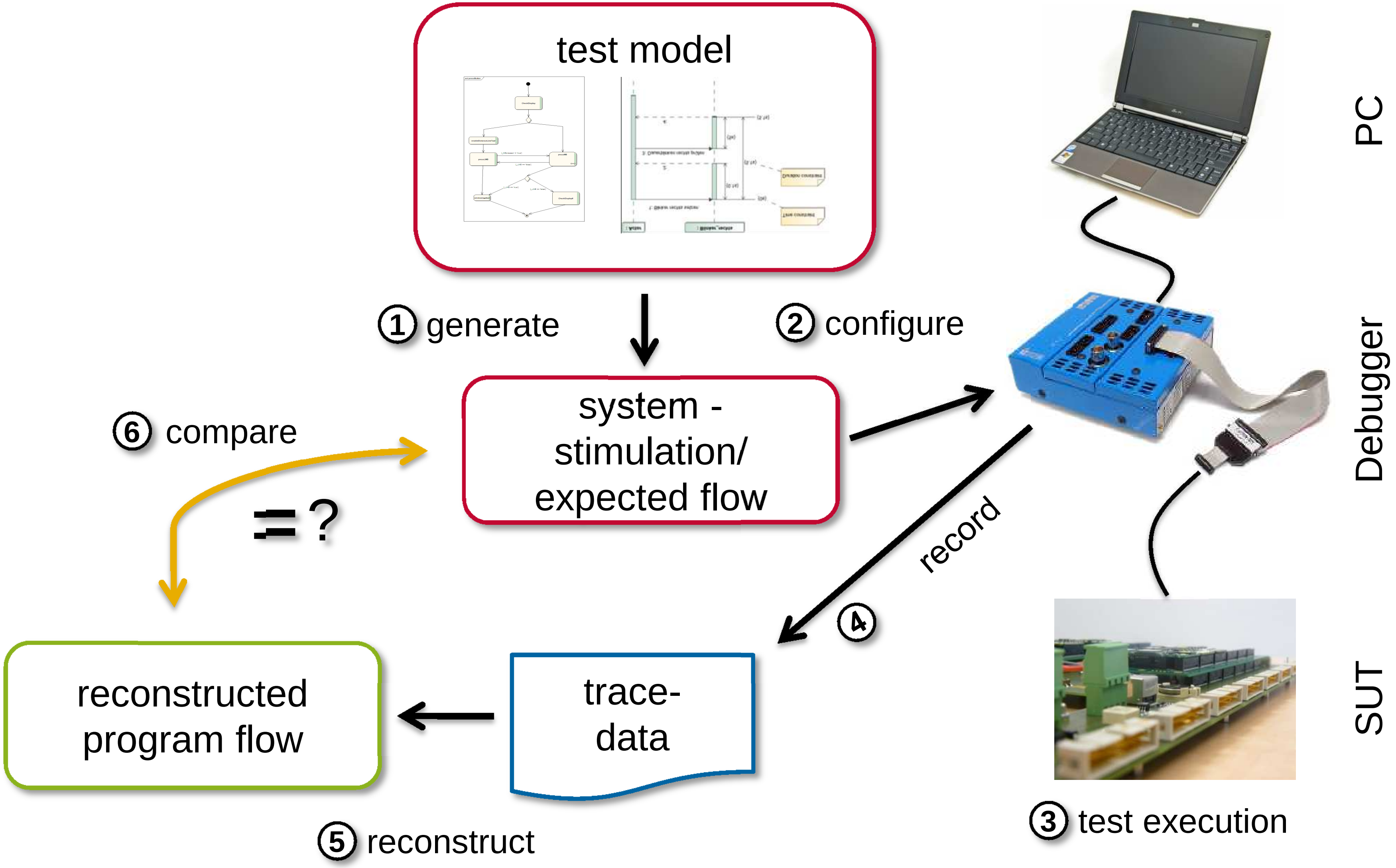
eMOTE

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)



test model

① generate

② configure

system - stimulation/ expected flow

⑥ compare

=?

record

④

reconstructed program flow

trace-data

⑤ reconstruct

③ test execution

PC

Debugger

SUT

Thank you!

eMOTE

Dr. Philipp Graf

Department Manager
Embedded Systems & Sensors Engineering (ESS)

FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe
Tel.: +49-721-9654-180
Fax: +49-721-9654-181
http://www.fzi.de