Fehlerlokalisierung in Firmware
Über wertvolle Testfälle, die keine Fehler entdecken

Dr. Bernhard Peischl
Softnet Austria
Institut für Softwaretechnologie
Technische Universität Graz
Agenda

- Quality Assurance
- Model-based Systems
- Detect, Localize, Repair
- Model-based Testing
- Cyber-physical systems
- Test Strategy
- Passing Test Cases
- Idea + Demo
- Passing Test Cases
Twenty years ago …
“Six decades into the computer revolution, four decades since the invention of the microprocessor, and two decades into the rise of the modern Internet, all of the technology required to transform industries through software finally works and can be widely delivered at global scale.”

Software and KIBS

- Simulation services – modelling of complex systems
- Eng. services for embedded (automotive) systems
- IT security
- Augmented reality
- Biotechnology
- Process engineering
- Industrial design
- Radio Frequency IDentification
- Human-computer interface
- Architecture, building services engineering
- Innovation- & knowledge management

Source: Wissensintensive Dienstleistungen, Juni 2010, Land Steiermark
Key-Less Go, Key-Less Entry
Cyber-Physical Systems (CPS)

- A Cyber-physical systems integrates
  - Computing
  - Communication & storage capabilities
  - Monitoring and/or control of entities in the physical world (actuators, sensors, … )

- Quality attributes
  - Dependability
  - Safety
  - Security
  - Efficiency
  - Real Time
Embedded Systems Roadmap

Achatz (Siemens)
Broy (TU München)
Damm (OFFIS)
Dämbkes (EADS)
Grimm (Daimler)
Liggesmeyer (Faunhofer)

Quelle: http://www.safetrans-de.org
Trends (1)

• Model-based Engineering
  – Complexity, interoperability
  – Non-functional constraints
  – Numerous variants (product line)
  – Cross-disciplinary
  – V&V (e.g., verification engineer)
  – Safety in open systems: Security
  – High degree of automation
  – Model-based Testing (MBT)
  – Model-based Systems (MBS)
  – Quantitative results with known confidence level
Trends (2)

**SW/HW:** seamlessly integrated, safe & secure, distributed and embedded

Increased complexity & function, linking together various disciplines, techn. stack

- Abstraction
- Different views (mech. / elec. / softw.)
- Automation
- Visualisation

Source: Fraunhofer IESE & own research.
Detect – Localize - Correct

- Verification and Validation
  - Testing
  - Debugging
  - 50%-80% of overall design cycle time

- Localization and Correction
  - 35% of overall design cycle time

Source: IBM Haifa, EU-Projekt PROSYD, Deliverable 2.1/1
Detection (1): Example SIP

- Session Initiation Protocol (SIP)
  - Session management
  - Establishment, transfer, or termination of sessions
  - Communication sessions between end points
  - Signaling part independent of the media type
  - Text based protocol
  - Uses a request/response model

- Registrar: register method
  - Association of
    - user address and
    - end point
Detection (2): SIP

REGISTER sip:win.traussnig.at SIP/2.0
Via: SIP/2.0/UDP 193.80.71.141:1156;branch=z9hG4bK-ticf2736p42y;rport
From: "bertl" <sip:9001@win.traussnig.at>;tag=715d1kgvae
To: "bertl" <sip:9001@win.traussnig.at>
Call-ID: 0ff9d944b4dc-hkzr8tvkarws@snomSoft-000413FFFFFF
CSeq: 31 REGISTER
Max-Forwards: 70
Contact:
<sip:9001@193.80.71.141:1156;line=ojn9itpa>;flow-id=1;q=1.0;+sip.instance="<urn:uuid:4bcc15eb-5587-463c-8932-36036fa80edf>"
User-Agent: snomSoft/5.3
Allow-Events: dialog
X-Real-IP: 193.80.71.141
Expires: 60
Content-Length: 0

Source: Kapsch CarrierCom AG, Design Document, MissisSIPpi.
Detection (3): SIP

Client 1                      Proxy                          Client 2

INVITE
407 Authentication
ACK

INVITE
100 Trying
180 Ringing

INVITE
100 Trying
180 Ringing

200 OK
ACK

200 OK
ACK

RTP

BYE

200 OK

200 OK

Source: Kapsch CarrierCom AG, Design Document, MissisSIPpi.
Detection (4): SIP

Source: Kapsch CarrierCom AG, Design Document, MissisSIPpi.
Detection (5): SIP

Source: Kapsch CarrierCom AG, Design Document, MissisSIPpi.
Detection (6): Functional Test Strategy

- Test strategy
  - Scenario-based testing
    (Not found, Inv. Req., Unauth., Ok, Delete, …)
  - Coverage-based testing
  - Fault-based testing
  - Random testing

<table>
<thead>
<tr>
<th>Test Purpose</th>
<th>Registrar A</th>
<th>Registrar B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pass</td>
<td>fail</td>
</tr>
<tr>
<td>not found</td>
<td>880</td>
<td>0</td>
</tr>
<tr>
<td>inv. req.</td>
<td>1008</td>
<td>320</td>
</tr>
<tr>
<td>unauth.</td>
<td>130</td>
<td>302</td>
</tr>
<tr>
<td>ok</td>
<td>1104</td>
<td>384</td>
</tr>
<tr>
<td>delete</td>
<td>1148</td>
<td>132</td>
</tr>
<tr>
<td>Total</td>
<td>4270</td>
<td>1138</td>
</tr>
</tbody>
</table>

Detection (7): MBT Industrialization

- UML Activity Diagram
- UML Statechart
- Workflow model
- Source code annotations
- Proprietary models (parameterize,…)
- Process algera (e.g. LOTOS)
  - ……

Engineering Model

Analysis Model

- Extended Finite State Machines (EFSM)
- Symbolic Transition Systems (STS)
- Labelled Transition Systems (LTS)
- Constrains Models (CSP)
- Abstract State Machines (ASM)
  - ……
  - ……
Localization (1): Failing Test Case

Localization (2): Failing Test Case

Localization (3): Computing Diagnoses

• Conflict
  – Contradictive Sub-Model
  – Conflict A = \{N1, N3, N5\}
  – Conflict B = \{N2, N4, N5\}

• Diagnoses
  – \{N5\}
  – \{N1, N2\}
  – \{N1, N4\}
  – \{N2, N3\}
  – \{N3, N4\}

• Diagnoses = Hitting Sets over Conflicts!
Localization (4): Debugger

- Modelling
- Correcting
- Diagnosis
- Observing
- Simulating

Table 1: Truth table for the example

<table>
<thead>
<tr>
<th>(~A)</th>
<th>(i_1)</th>
<th>(i_2)</th>
<th>(o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
VHDL Example: 4-bit Adder

1. \( g_1 = a_1 \&\& b_1; \)
2. \( p_1 = !((a_1 \&\& b_1) || (!a_1 \&\& !b_1)); \)
3. \( g_2 = a_2 \&\& b_2; \)
4. \( p_2 = !((a_2 \&\& b_2) || (!a_2 \&\& !b_2)); \)
5. \( g_3 = a_3 \&\& b_3; \)
6. \( p_3 = !((a_3 \&\& b_3) || (!a_3 \&\& !b_3)); \)
7. \( g_4 = a_4 \&\& b_4; \)
8. \( p_4 = !((a_4 \&\& b_4) || (!a_4 \&\& !b_4)); \)
9. \( z_1 = !((c_1 \&\& p_1) || (!c_1 \&\& !p_1)); \)
10. \( s_{11} = p_1 \&\& c_1; \)
11. \( s_{12} = g_1 || s_{11}; \)
12. \( z_2 = !((s_{12} \&\& p_2) || (!s_{12} \&\& !p_2)); \)
13. \( s_{21} = c_1 \&\& p_1 \&\& p_2; \)
14. \( s_{22} = g_1 \&\& p_2; \)
15. \( s_{23} = (s_{21} || s_{22}) || g_2; \)
16. \( z_3 = !((s_{23} \&\& p_3) || (!s_{23} \&\& !p_3)); \)
17. \( s_{31} = c_1 \&\& p_1 \&\& p_2 \&\& p_3; \)
18. \( s_{32} = g_1 \&\& p_2 \&\& p_3; \)
19. \( s_{33} = g_2 \&\& p_3; \)
20. \( s_{34} = ((s_{31} || s_{32}) || s_{33}) || g_3; \)
21. \( z_4 = !((s_{34} \&\& p_4) || (!s_{34} \&\& !p_4)); \)
22. \( s_{41} = c_1 \&\& p_1 \&\& p_2 \&\& p_3 \&\& p_4; \)
23. \( s_{42} = g_1 \&\& p_2 \&\& p_3 \&\& p_4; \)
24. \( s_{43} = g_2 \&\& p_3 \&\& p_4; \)
25. \( s_{44} = g_3 \&\& p_4; \)
26. \( c_n = (((g_4 || s_{41}) || s_{42}) || s_{43}) || s_{44}; \)

\[ a_1 = \text{true} \]
\[ a_2 = \text{false} \]
\[ a_3 = \text{true} \]
\[ a_4 = \text{true} \]
\[ b_1 = \text{false} \]
\[ b_2 = \text{true} \]
\[ b_3 = \text{false} \]
\[ b_4 = \text{false} \]
\[ c_1 = \text{false} \]
\[ z_1 = \text{false} \]
\[ z_2 = \text{true} \]
\[ z_3 = \text{true} \]
\[ z_4 = \text{false} \]
\[ c_n = \text{false} \]
Demo Software Debugger

Dr. B. Peischl

Localization: Passing Test Case

No conflicts $\rightarrow$ No diagnoses
Software Debugging: Example (1)
Software Debugging: Example (3)
Diagnosis Problem: Single Testcase

Diagnosis problem
\((SD, COMP, OBS)\)
for a single test case.

\(n \ldots \text{no. cycles}\)

\(SD_i \ldots \text{SD for cycle } i\)

\(C_i \ldots \text{Components for cycle } i\)

\(OBS_i \ldots \text{Observations for cycle } i\)

\[SD = \bigcup_{i=1..n} SD_i\]

\[COMP = \bigcup_{i=1..n} C_i\]

\[OBS = \bigcup_{i=1..n} OBS_i\]
Diagnosis Problem: Test Suite (1)

Diagnosis problem 

\((SD^*, COMP^*, OBS^*)\)

for a test suite \(TC_1, TC_2, TC_3, \ldots, TC_k\).

\(k \ldots \) no. test cases

\(SD_j \ldots\) SD considering \(TC_j\)

\(C_i^j \ldots\) instance of component \(C\) at cycle \(i\) in test case \(j\)

\(OBS_i^j \ldots\) Observations for cycle \(i\) of test case \(TC_j\)
Diagnosis Problem: Test Suite (2)

\[ SD^* = \bigcup_{j=1..k} SD_j \cup \{ \neg AB(C_0^j) \rightarrow \neg AB(C_1^j) \land \neg AB(C_2^j) \land \neg AB(C_3^j) \land \ldots \neg AB(C_n^j) \} \]

\[ COMP^* = \bigcup_{j=1..k} \bigcup_{i=0..n} C_i^j \]

\[ OBS^* = \bigcup_{j=1..k} \bigcup_{i=1..n} OBS_i^j \]
### Exploiting Passing Testcases: Example

<table>
<thead>
<tr>
<th>assumption</th>
<th>in1</th>
<th>in2</th>
<th>out</th>
<th>inter</th>
<th>verdict</th>
</tr>
</thead>
<tbody>
<tr>
<td>( AB(\text{not}), \neg AB(\text{xor}) )</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>fail</td>
</tr>
<tr>
<td>( AB(\text{xor}) ) \neg AB(\text{xor})</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>pass</td>
</tr>
</tbody>
</table>

System description with Ackermann constraints

- The same combination of input values applied to a deterministic component $C$ produces the same output for every instance of $C$.

- System description with Ackermann constraints

\[
SD_A = SD^* \cup CON_A,
\]

\[
CON_A = \neg AB(C_i) \land \forall_{l=1}^{m} i_{ci}^l = i_{cj}^l \rightarrow \forall_{p=1}^{n} o_{ci}^p = o_{cj}^p
\]

$i_{ci}^l$…. input no. $l$ of component $C_i$

$o_{ci}^p$… output no. $p$ of component $C_i$
Filtering Algorithm (Post Processing)

1. For all $D \in \Delta$ do

2. For all test cases $TC_i \in TC$ do
   a. Let $i_{Di}$ denote the input values and let $o_{Dj}$ denote the output values of component $D$ by assuming $AB(D) \land \{\neg AB(C) \mid C \in COMP \setminus D\}$
   b. If there exits $i, j$, $i \neq j$, such that $i_{Di} = i_{Dj} \land o_{Di} \neq o_{Dj}$ then remove $D$ from $\Delta$

3. return $\Delta$
ISCAS’89 (1): Empirical Results

ISCAS’89 (2): Empirical Results

ISCAS’89 (3): Double-Fault Diagnoses

Use-Inspired Research

Source: Donald Stokes, Pasteur’s Quadrant
Conclusion

• Cyber-physical systems
  – V&V gains importance
  – MBT on the brink of industrialization
  – MBS application in the field of software / HDLs

• V&V
  – 50%-80% of overall design cycle
  – 35% go into localizing the root cause of misbehavior

• Automated Debugger (an MBS application)
  – Idea, Tool
  – Failing and passing test cases
  – Passing test cases contribute to localizing the root cause
Questions?

SW/HW: seamlessly integrated, safe & secure, distributed and embedded

Increased complexity & function, linking together various disciplines, techn. stack

- Abstraction
- Different views (mech./elec./softw.)
- Automation
- Visualisation

Pure Basic Research (Bohr)  
Use-Inspired Basic Research (Pasteur)

Quest for Fundamental Understanding

Design Cycle without Localization  
Specify Design Detect Locate Report

Design Cycle with Localization  
Specify Design Detect Locate Cared

Consideration of Use

Questions?

UML Activity Diagram
- UML Statechart
- Workflow model
- Source code annotations
- Proprietary models (parameterize, …)
- Process algebra (e.g., LOTOS)

Extended Finite State Machines (EFSM)
- Symbolic Transition Systems (STS)
- Labelled Transition Systems (LTS)
- Constrained Models (CSP)
- Abstract State Machines (ASM)

No conflicts → No diagnoses

Dr. B. Peischl  
Contact

Dr. Bernhard Peischl

Softnet Austria
Institut für Softwaretechnologie
Technische Universität Graz
bernhard.peisch@soft-net.at
www.soft-net.at