Automated Test Program Generation for Automotive Devices

2010 International Test Conference
Austin, Texas
November 2 - 4

J. Vollmar, P. Huber
Teradyne, Inc.

A. Drappa
Robert Bosch, GmbH
Purpose

- Show a case study of ATE program generation from a tester-independent test specification
- Scope is analog V/I and time measurement tests for automotive
- Explain requirements and challenges of the interface
- Show potential gain of this approach
Outline

• Problem
  – Test program generation is too expensive

• Concept
  – Combine test specification and test program coding

• Implementation
  – Develop new tester-independent language for automatic conversion

• Results
  – Potential reduction of coding time
  – Acceptable test time impact
Problem Background

Test program development is too labor-intensive
- Next generation ASICs with > 3500 tests
- Difficult to reuse code between devices/developers
- Difficult to validate program against test specs
- Shorter market windows

Past: Test program containing max 1000 tests

<table>
<thead>
<tr>
<th>Test Spec</th>
<th>Test Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000h</td>
<td>1000h</td>
</tr>
</tbody>
</table>

Today: Test program containing 3000 tests

<table>
<thead>
<tr>
<th>Test Spec</th>
<th>Test Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000h</td>
<td>3000h</td>
</tr>
</tbody>
</table>
Solution: Automation

- Describe analog tests in a tester-independent specification
- Generate ATE programs automatically
- Saves program development time
Why Automation Can Help

• Test-related data often already exists
  – Informal test descriptions
  – Design and simulation environments
  – Specification documents (data sheets)

• ATPG using STIL is successful
  – Limited to digital test
  – Why can’t V/I tests be standardized too?

• Automation is the answer elsewhere
  – Design rule checking, floor-planning, mask generation, etc.
Starting Point: Test Spec Analysis

• The Analysis of existing informal test descriptions shows that 80% of the definitions can be covered with only about 20 different test functions
  – Voltage force/measure
  – Differential voltage measure
  – Voltage threshold measure
  – Current force/measure
  – Current threshold measure
  – Time PropDelay, Period, RiseTime (+ 4 more)
  – Digital serial port programming
  – Digital pattern execution

• Why not standardize these functions for automatic conversion?
Concept: Combine Spec and Coding

- Why do we need to define each test twice?
  - Test Specification
  - Test Program

- Example: Test program containing 3000 Tests
  - Traditional Approach:
    - Test Spec Generation: 3000h
    - Test Program Coding: 3000h
  - New Approach (estimate):
    - Test Spec Generation: 4000h
    - Gain: 1400h
    - 600h

Test Spec Generation is expected to increase but test program coding is reduced to 20%
Standard Test Specification

- Formalizes analog test descriptions
- Created by test engineer using a special editor
- Tester-independent (preserves IP)
- Enough detail to generate ATE tests
Key Project Requirements

- Production throughput within 5%
- Auto-generate 80% of tests
- Back-annotation from test program
- Target ATE: Teradyne FLEX IG-XL
Implementation: New Test Language

New tester independent, language-based test definition


```plaintext
// Set up test conditions
Pins(VDD3).Current.Force(-50mA, 1V, 200mA);
Wait(0.4ms);
Pins(VDD1).Current.Force(-250mA, 1V, 1A);
Wait(0.4ms);
Pins(VS2).Current.Force(-50mA, 1V, 200mA);
Wait(1ms);

// Read and report results
pldTmpValueA = Pins(VDD5).Voltage.Meter.Read(10V);
Evaluate(pldTmpValueA);
```
Test Language Challenges

- Optional parameters
- Connect, disconnect, power-up
- Instrument ranges
- Instrument alarms
- V/I compliance voltage
- Tester-specific escape
Test Spec Overall Design

XML data with embedded code blocks

- File structure is an XML schema
  - Common standard for many data types
  - Easy to read and write files with .NET
  - Tools available for viewing/debugging

- Language-based test definitions
  - Best for signal sequencing
  - Allows loops, if/else, settling waits, etc.
  - ‘C’-based languages are widely known
  - Using a subset of standardized C#
Code Generator Tool

Bosch Program Generator - CodeGen Demo.btp - [Spec Editor]

<table>
<thead>
<tr>
<th>Device Name</th>
<th>Spec Version</th>
<th>Valid After</th>
<th>Spec Number</th>
<th>Author/Team</th>
<th>ASIC Data Sheet</th>
<th>Code Definitions</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM9xx</td>
<td>0.33</td>
<td>1/1/2008</td>
<td></td>
<td></td>
<td></td>
<td>public ValueList g_pldTestValue; public ValueList g_pldTestValue1; public ValueList g_pldTestValue2;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Condition</th>
<th>Description</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>EWSHT</td>
<td>VM</td>
<td>140</td>
</tr>
<tr>
<td>EWSRT</td>
<td>VM</td>
<td>25</td>
</tr>
<tr>
<td>EWSCT</td>
<td>VM</td>
<td>-40</td>
</tr>
<tr>
<td>FTHT</td>
<td>EM</td>
<td>140</td>
</tr>
<tr>
<td>FTRT</td>
<td>EM</td>
<td>25</td>
</tr>
<tr>
<td>FTCT</td>
<td>EM</td>
<td>-40</td>
</tr>
<tr>
<td>EWS2HT</td>
<td>VM2</td>
<td>140</td>
</tr>
<tr>
<td>EWS2RT</td>
<td>VM2</td>
<td>25</td>
</tr>
<tr>
<td>EWS2CT</td>
<td>VM2</td>
<td>-40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM940</td>
<td></td>
</tr>
<tr>
<td>DM921</td>
<td></td>
</tr>
<tr>
<td>DM925</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generating the ATE Program

Program Template

Standard Test Specification

Tester Specific Program

Tester Configuration
Program Template

- Specific to the ATE platform
- Included in all generated test programs
- Forms the common parts of the test program
  - Operator interface
  - Handler/prober communication
  - Data collection setup
  - Utility functions
Resource Assignment

- Test requirements are given by the test spec
- Tester features are given by the configuration
- The Code Generator Tool can align requirements and features to develop a resource assignment.

Currently this is still a manual process because this feature is under development.
Results and Status

- Implemented code generator for FLEX
- Coded ~40 test modules using new language
- Verified test execution on ATE
- Execution time averages within 3% of hand-crafted benchmark programs
- Planning to integrate into test process
## Test Time Comparison

<table>
<thead>
<tr>
<th>Test Type</th>
<th>Benchmark (ms)</th>
<th>Generated (ms)</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuity</td>
<td>72.7</td>
<td>77.9</td>
<td>7.2%</td>
</tr>
<tr>
<td>Leakage Current</td>
<td>44.4</td>
<td>44.8</td>
<td>0.9%</td>
</tr>
<tr>
<td>Linear Regulators</td>
<td>345.5</td>
<td>351.7</td>
<td>1.8%</td>
</tr>
<tr>
<td>Supply Current</td>
<td>44.0</td>
<td>45.0</td>
<td>2.3%</td>
</tr>
<tr>
<td>Mixed VI Tests</td>
<td>127.1</td>
<td>134.4</td>
<td>5.7%</td>
</tr>
<tr>
<td>Functional tests</td>
<td>27.0</td>
<td>27.0</td>
<td>0.0%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>660.7</strong></td>
<td><strong>680.8</strong></td>
<td><strong>3.0%</strong></td>
</tr>
</tbody>
</table>
Lessons Learned: Advantages

• Test spec language is faster to learn than ATE

• Test methods are easier to reuse with instrument-independent code

• Test spec language is easier to read for people other than the author

• Separation of test spec from tester config makes it easy to change testers
Lessons Learned: Disadvantages

• More effort and discipline is required to write the test specification

• Must be able to tolerate up to 3% runtime hit without hand-optimization

• Hand-optimization makes back-annotation more difficult

• Test engineers already experienced in ATE need additional training in the new language
Conclusions

• Technique reduces test program development time

• Test specifications are more accurate and complete

• ATE code generator can be implemented

• Standards for defining analog tests are worth pursuing