How to achieve verification closure of configurable code by combining static analysis and dynamic testing

Antonello Celano, STMicroelectronics,
Alexandre Langenieux, MathWorks
Context

• Develop generic MCAL Module drivers (e.g. Mcu,CAN) for PowerPC and ARM for Automotive customers

• Those drivers must be configurable for each customer

• Respect of Automotive safety and security standards
Challenge

• Extreme configurable code

• Fulfill standard code verification requirements
  • MISRA-C:2012
  • CERT-C
  • Code Coverage (Decision, Condition, MC/DC)

• Increase confidence in software verification

<table>
<thead>
<tr>
<th>Project</th>
<th>Number of variant parameters</th>
<th>Number of boolean preprocessor macro (#define)</th>
<th>Number of software variants considering only boolean parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU Driver</td>
<td>357</td>
<td>58</td>
<td>$2^{58} = 288 \times 10^{19}$</td>
</tr>
<tr>
<td>CAN Module</td>
<td>97</td>
<td>50</td>
<td>$2^{50} = 10^{15}$</td>
</tr>
</tbody>
</table>
MCAL and CAN Drivers Variant Definition

• Configurable code
  • Metacode
  • Source Code
  • Configuration parameters

• Variant
  • Source code
  • Generated code
    • Metacode
    • "Valued" cfg parameters
Variant Subset Selection for Verification Closure

Variant Selection:
- iterative process
- developer expertise
- aggregated code coverage
Verification Closure Development Process

• On each selected variant
  • Testing
    • Functional testing
    • Robustness testing
  • Static analysis
    • Coding rules checking
    • Code metrics

• Consolidate results for each category
Results

• Errors found earlier in software, before reaching the customer

• Quality of all possible variants is controlled

<table>
<thead>
<tr>
<th>Project</th>
<th>Number of software variants considering only boolean parameters</th>
<th>Subset of variants used with the described methodology</th>
<th>Coverage (Statement, branch, MCDC) thresholds</th>
<th>Achieved Coverage Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU Driver</td>
<td>$2^{38} = 288\times10^{15}$</td>
<td>177</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>CAN Module</td>
<td>$2^{50} = 10^{15}$</td>
<td>179</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Future Enhancements

• Automatic extraction of the smallest software variants

• Improve code metrics consolidation across variants

• Extend code verification to formal code verification
Take Away

• Significant improvement of
  • productivity of ST development team
  • the quality of the configurable software

• Reusable framework beyond firmware development

• Possible to extend this method to other software verification activities
Questions