Bitte decken Sie die schraffierte Fläche mit einem Bild ab.

**Operation Concepts**

Model-Based Development with AURIX PiL Target

www.continental-corporation.com
Why do we need a PiL-Target?

- ISO 26262 **highly recommends** back-to-back testing for ASILs C and D. It notes the importance of testing in a representative target hardware environment, and stresses the need to be aware of differences between the test and hardware environments:

  - *Differences between the test environment and the target environment can arise in the source code or object code, for example, due to different bit widths of data words and address words of the processors.*
Why do we really need a PiL-Target? (1)

› Q4 2018 … „The Magic Bug“
› Strange hydraulic behavior in some maneuvers combined with Datalyser 3 measurement loss
› Reduced analysis possibilities
› Time to solve: 4-6 weeks
› Resources needed:
   2 MBD developers, 1 Basic Functions (BFU) expert,
   HiL system incl. test operator, 1 car incl. test operator
Why do we really need a PiL-Target? (2)

› „It was difficult to find the section in the generated code that caused the controller to reset and to map this error to a compiler bug. “

› „After we proved the correctness of the MATLAB model by manually adding debug variables in generated code we handed over to BFU. “

› „BFU contributed most to solving this issue by stepping through the assembler code. “

› „From my point of view this issue could have been solved much faster. Using a PiL-Target the developer is enabled to run all tests. “

Feedback provided by Software Developer, BFU Developer and MBD Developer
Agenda

› AURIX Introduction
› PiL Hardware
› PiL Project
› PiL Variants
› Jenkins CI
› Questions & Answers
### Introduction - Aurix Overview

<table>
<thead>
<tr>
<th>Customers</th>
<th>Aurix Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMW</td>
<td>TC397x (planned)</td>
</tr>
<tr>
<td>Tesla</td>
<td>TC377x (planned)</td>
</tr>
<tr>
<td>Volkswagen</td>
<td>TC367x (planned)</td>
</tr>
<tr>
<td></td>
<td>TC337x (planned)</td>
</tr>
</tbody>
</table>

PiL – Hardware

› 2\textsuperscript{nd} generation AURIX TC3xx starter kits available
  › Needed to develop Continental Download Tool for Aurix
  › Successor of the starter kits for further development is the Continental Evaluation Board with PCU

› Reuse of hardware that’s not needed any more!
PiL – MathWorks Consulting and Continental

› Start of PiL-Project Q4 2019
   › Windriver Licenses
   › HighTec Toolchain
   › WindRiver introduction
   › Automate Flashing
   › Programming and Test sessions
   › Review

› Start of PiL-Enhancement Project Q4 2020
   › RAM Execution
   › DAS, MCD API
   › AUTOSAR (ongoing)
   › Programming and Test sessions
   › Review
PiL – Variants

› Simple run-time code execution
  › AurixBlinky377.slx

› Verify referenced model code using PIL
  › Aurix_sincos.slx
  › Sincos_SILPIL.slx

› Verify top model code using PIL
  › Aurix377_PIL.slx
Automated Verification using PiL

<table>
<thead>
<tr>
<th>SIMULATION</th>
<th>DEBUG</th>
<th>MODELING</th>
<th>FORMAT</th>
<th>APPS</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated Verification</td>
<td>Simulation Mode</td>
<td>Top model</td>
<td>Settings</td>
<td>Monitor Signals</td>
<td>Stop Time 10</td>
</tr>
<tr>
<td>SIL/PIL Mode</td>
<td>Processor-in-the-Loop (PIL)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Run Verification

Compare Runs

RESULTS
PiL – Aurix377_PIL (2)

› Compare Signals
› Absolute and Relative Tolerances
› Time Tolerance
› Detailed Plots
  › Signal Values
  › Tolerance, Difference
PiL – Example use cases – Execution profiling (1)

Code Execution Profiling Report for
mck_bbw_state_module_PiL_Harness/mkc_bbw_state_module

The code execution profiling report provides metrics based on data collected from a SIL or PiL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PiL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

<table>
<thead>
<tr>
<th>Section</th>
<th>Maximum Execution Time in Ticks</th>
<th>Average Execution Time in Ticks</th>
<th>Maximum Self Time in Ticks</th>
<th>Average Self Time in Ticks</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>mck_bbw_state_module_initial</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>mck_bbw_state_module_init</td>
<td>142</td>
<td>142</td>
<td>81</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>mck_bbw_state_module (0.002.0)</td>
<td>390</td>
<td>207</td>
<td>243</td>
<td>147</td>
<td>428</td>
</tr>
<tr>
<td>GetD_TmcTrxSgl</td>
<td>58</td>
<td>54</td>
<td>58</td>
<td>54</td>
<td>271</td>
</tr>
<tr>
<td>MknBvwStateMachine</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>MknBvwStateMachine</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>MknBvwStateMachine</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>MknBvwStateMachine</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>MknBvwStateMachine</td>
<td>52</td>
<td>35</td>
<td>52</td>
<td>35</td>
<td>1</td>
</tr>
<tr>
<td>MknBvwStateMachine</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>28</td>
<td>1</td>
</tr>
</tbody>
</table>

2. Profiled Sections of Code

Code Execution Profiling, measure runtime in Ticks (3.3ns)
PiL – Example use cases - Execution profiling (2)

› Display execution times for complete simulations

› Display execution times and signal data in one plot
PiL – Example use cases – Back-to-back testing (1)

› Use Test Manager to set up Equivalence Tests for …
  › MiL-SiL
  › MiL-PiL
  › SiL-PiL

… using the same model, harnesses and simulation inputs
PiL – Example use cases – Back-to-back testing (2)

› Explore Equivalence Criteria Result
  › MiL vs. PiL
  › Tolerance and Difference
## PiL – Example use cases – Measureing Code Coverage

- Generate Coverage Information
- Highlight coverage in models
- View annotated C-code

### Summary

<table>
<thead>
<tr>
<th>File Contents/Complexity</th>
<th>Decision</th>
<th>Condition</th>
<th>MCDC</th>
<th>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. mkc_bbw_state_module.c</td>
<td>42 91%</td>
<td>100%</td>
<td>100%</td>
<td>95%</td>
</tr>
<tr>
<td>2. mkc_bbw_state_module_Init</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
<tr>
<td>3. mkc_bbw_state_module</td>
<td>40 91%</td>
<td>100%</td>
<td>100%</td>
<td>95%</td>
</tr>
<tr>
<td>4. mkc_bbw_state_module_initialize</td>
<td>1</td>
<td>--</td>
<td>--</td>
<td>100%</td>
</tr>
</tbody>
</table>

Vehicle Dynamics

4. Mai 2021

AMS VED RD P C © Continental AG
PiL – Example use cases – Jenkins CI (1)

- Commit model change
- Create Pull Request
- PiL-Test execution using Jenkins CI
PiL – Example use cases – Jenkins CI (2)

› Same handling as MiL and SiL Tests

› Uses own Jenkins Node with Evaluation Board attached

› One node can handle multiple Evaluation Boards with different µC (TC377x, TC397x, …)
PiL-Target - Live Demo
PiL-Target - Conclusion

› Using a PiL-Target you can
  › validate the target compiler and linker
  › get an idea about the performance of your algorithm by the use of profiling
  › generate coverage data for your algorithm while running on the target processor
  › find hardware dependent issues early and fast

› If we had a PiL-Target back in 2018 when the Magic Bug happened, we would have been able to identify the bug long before the generated code was running on the ECU.
PiL-Target – Future Topics

› Use MCD (JTAG) remotely to access hardware connected to other PCs.
  › Use PiL-Target in “Work from Home” arrangements.

› Support AUTOSAR models

› Use AURIX PiL Target and the new C-Code Integration available with R2021a within Software Factory and run handwritten code on PiL-Target
Questions?
Thank you for your attention!