Continental
The Future in Motion

Operation Concepts

Model-Based Development with AURIX PiL Target

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.*

Nutzung von Simulink für Projekte nach ISO 26262
Tom Erkkinen, MathWorks
https://de.mathworks.com/company/newsletters/articles/how-to-use-simulink-for-iso-26262-projects.html
ISO 26262-6:2018, Clause 10.4.7

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, 1 HiL system incl. test operator, 1 car incl. test operator

Two independent compiler bugs

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

- Customers
  - BMW
  - Tesla
  - Volkswagen

- Aurix Types
  - TC397x Q1 2020
  - TC377x Q1 2020
  - TC367x (planned) Q4 2020
  - TC337x (planned) Q4 2020

https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/

PiL – Hardware

- 2nd 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

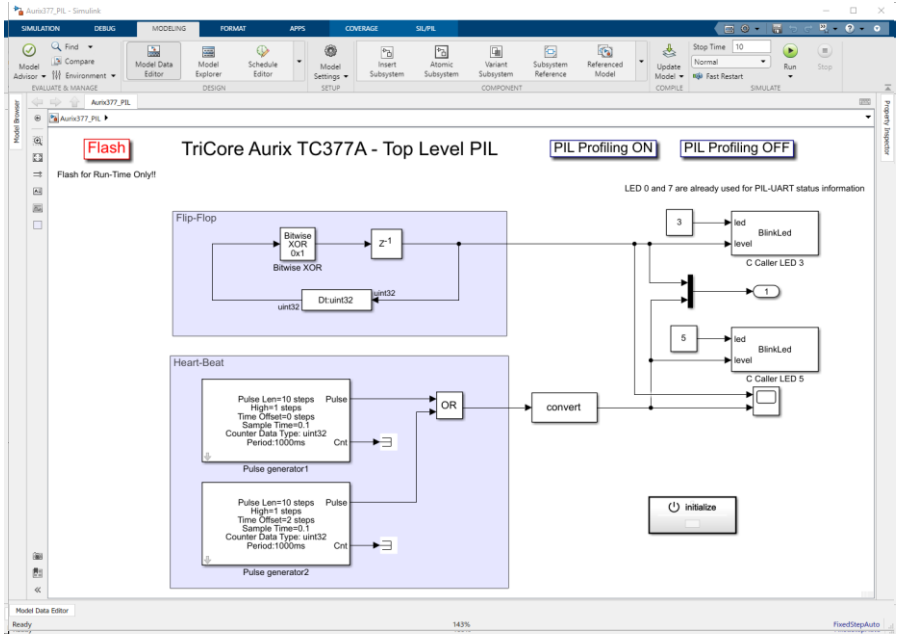
- **Reuse of hardware that's not needed any more!**

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

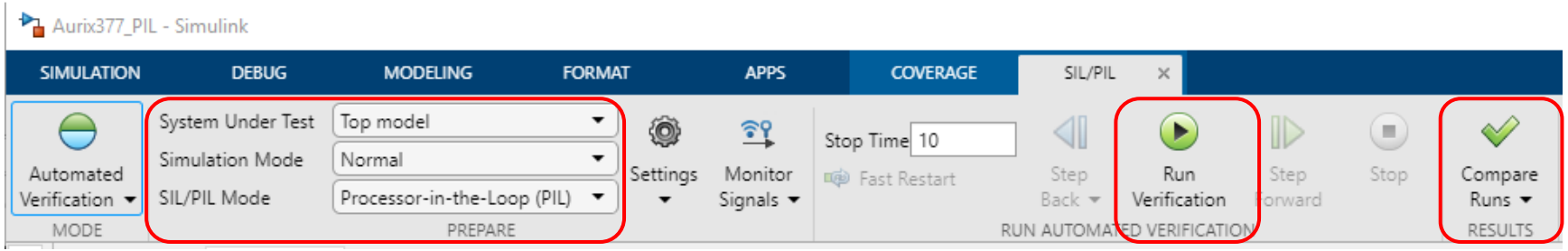
PiL – Variants

- 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



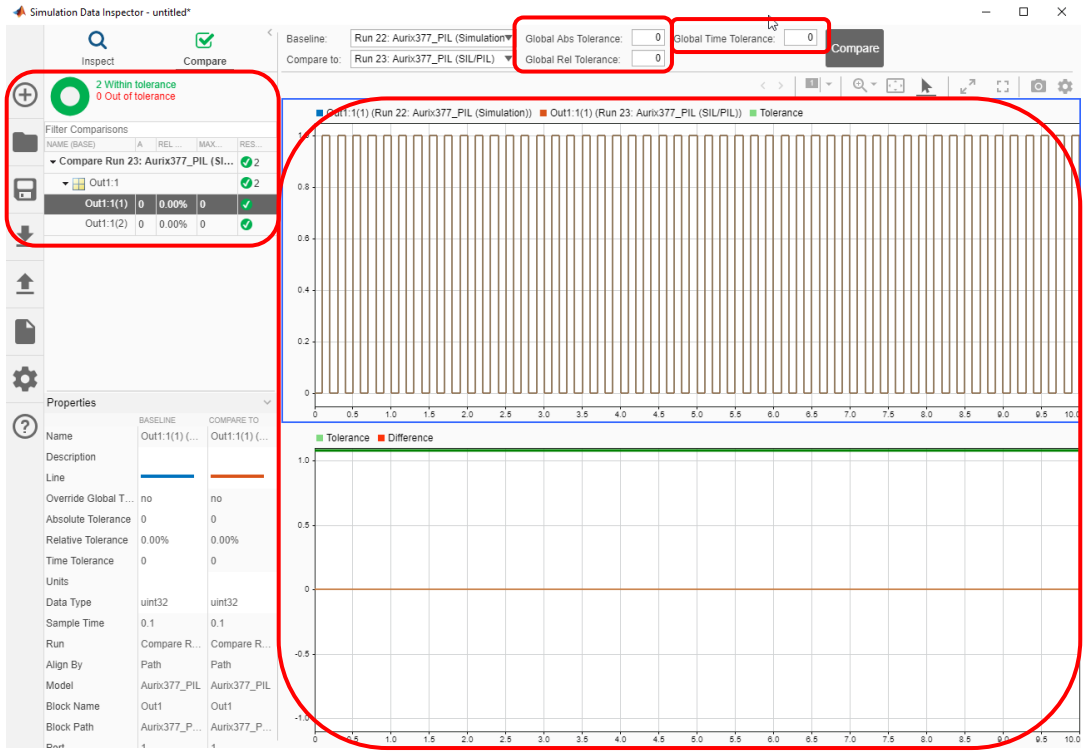
PiL – Aurix377_PIL (1)

› Automated Verification using PiL



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

Code Execution Profiling Report for mkc_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

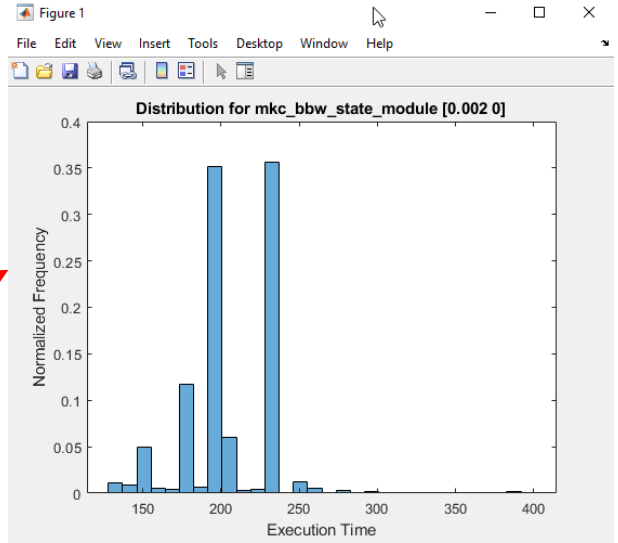
Total time	888439
Unit of time	Ticks
Command	report(executionProfile, 'Units', 'Ticks');
Timer frequency (ticks per second)	unavailable
Profiling data created	21-Sep-2020 10:31:13

2. Profiled Sections of Code

Section	Maximum Execution Time in ticks	Average Execution Time in ticks	Maximum Self Time in ticks	Average Self Time in ticks	Calls
mkc_bbw_state_module_initialize	33	33	33	33	1
[+] mkc_bbw_state_module_Init	142	142	81	81	1
[-] mkc_bbw_state_module [0.002 0]	399	207	243	147	4282
GetD_TmcTrvSig	58	54	58	54	2714
MkcBbwStateMachine	28	28	28	28	1
MkcBbwStateMachine	34	34	34	34	7
MkcBbwStateMachine	28	28	28	28	6
MkcBbwStateMachine	28	28	28	28	4
MkcBbwStateMachine	52	35	52	35	13
MkcBbwStateMachine	28	28	28	28	11
MkcBbwStateMachine	28	28	28	28	18

OK Help

- Code Execution Profiling, measure runtime in Ticks (3.3ns)

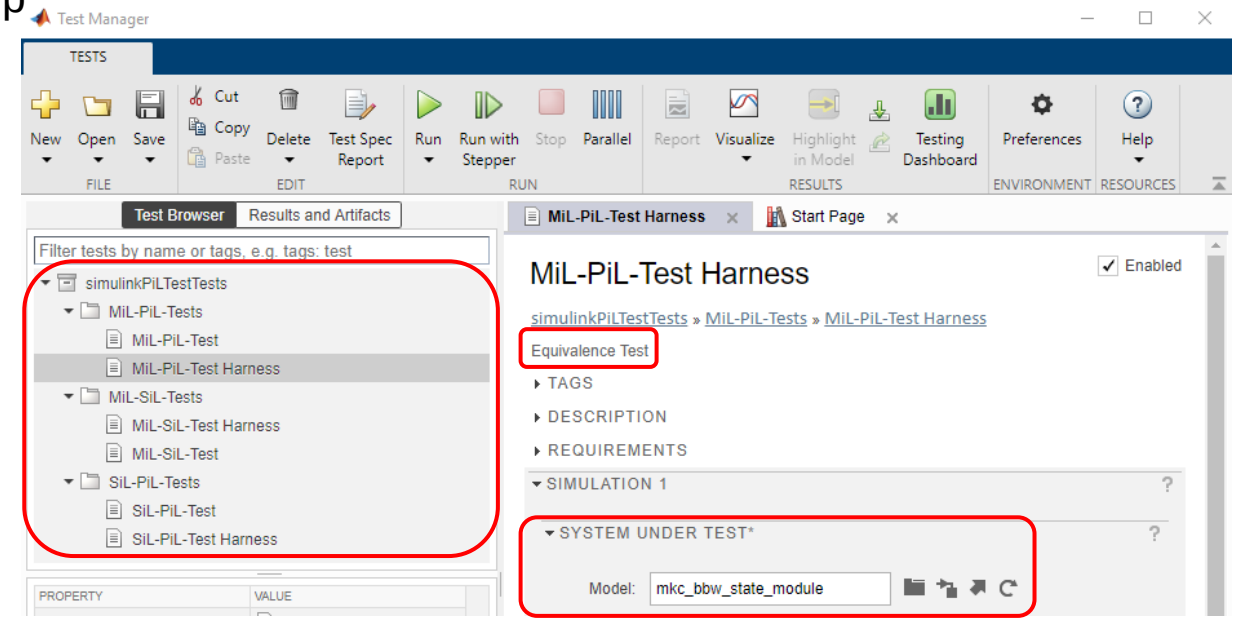


PiL – Example use cases - Execution profiling (2)

- Display execution times for complete simulations
- Display execution times and signal data in one plot

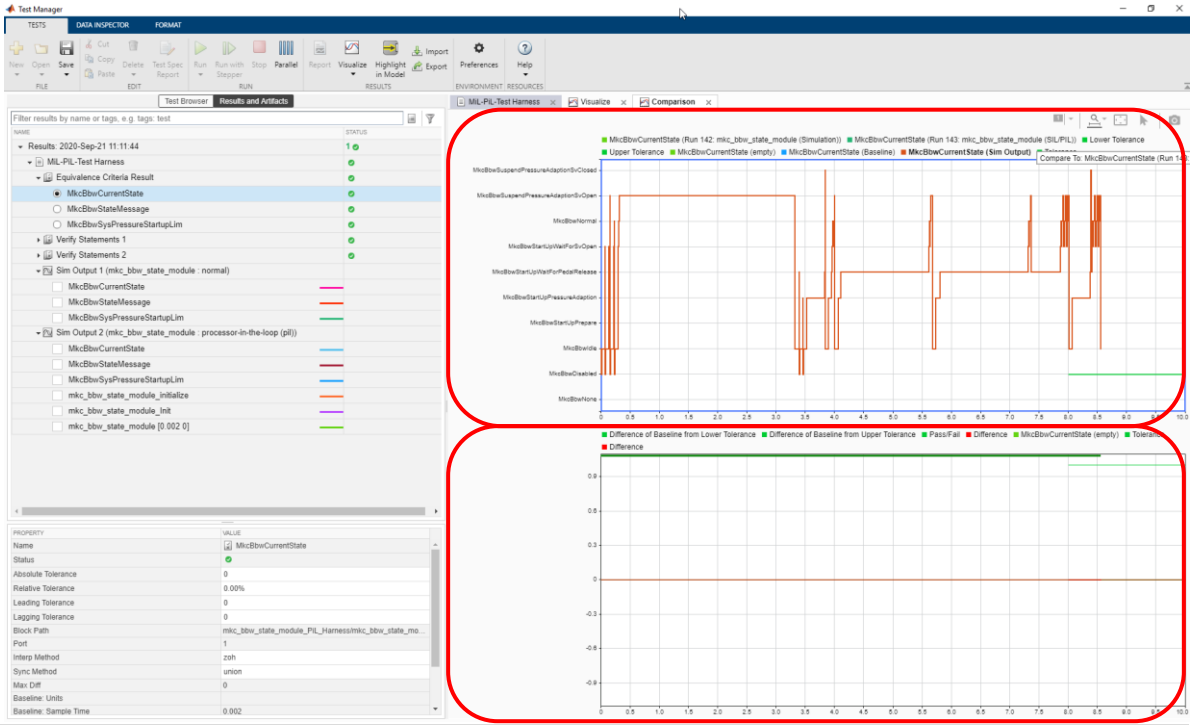
PiL – Example use cases – Back-to-back testing (1)

- Use Test Manager to set up Equivalence Tests for …
  - MiL-SiL
  - MiL-PiL
  - SiL-PiL
- … using the same model, harnesses and simulation inputs



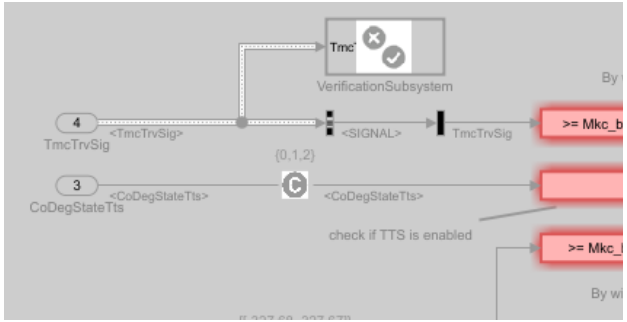
PiL – Example use cases – Back-to-back testing (2)

- 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



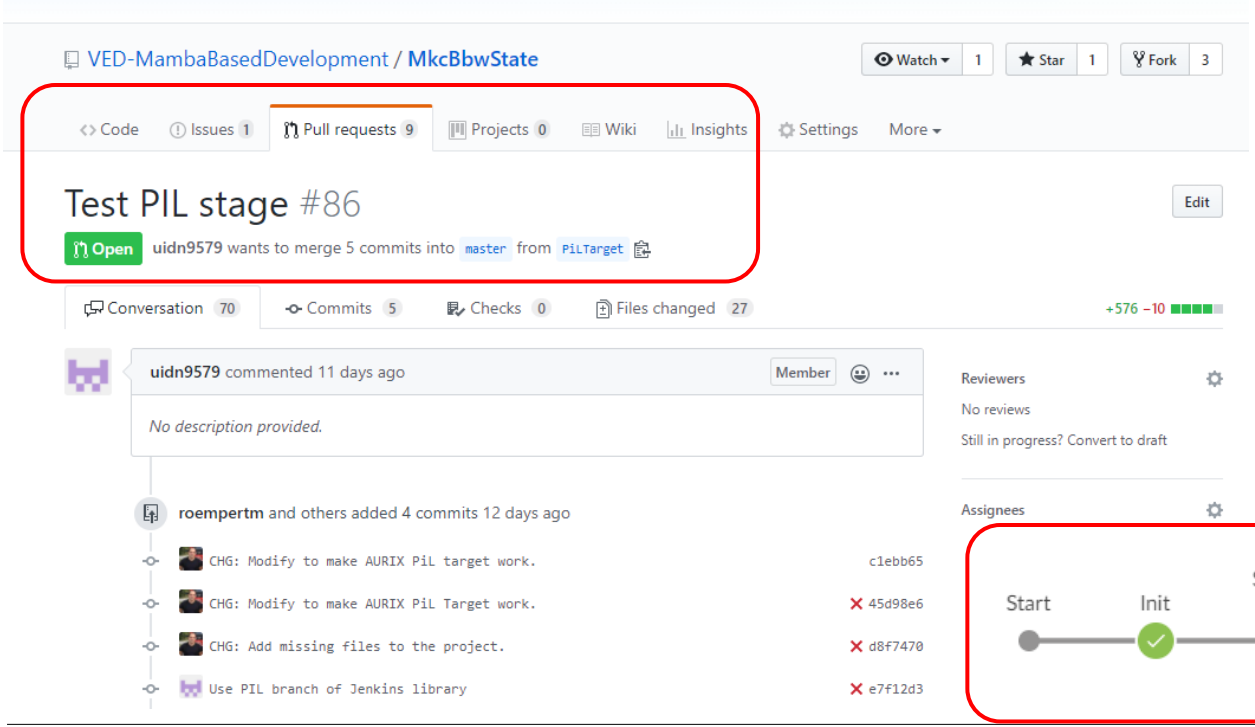
Summary

File Contents/Complexity — **Test 1**

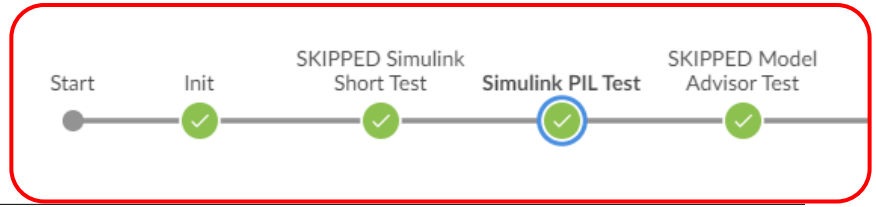
	Decision	Condition	MCDC	Statement
1 . mkc_bbw_state_module.c	42 91%	100%	100%	95%
2 . . . mkc_bbw_state_module_Init	1 --	--	--	100%
3 . . . mkc_bbw_state_module	40 91%	100%	100%	95%
4 . . . mkc_bbw_state_module_initialize	1 --	--	--	100%

```
217  /* End of Switch: '<S1>/Switch' */
218
219  /* Chart: '<S3>/MkcBbwStateMachine' incorporates:
220   *  Constant: '<S1>/Constant6'
221   *  Inport: '<Root>/CoDegStateBbw'
222   *  Inport: '<Root>/MkcSpcPressDecInhibit'
223   *  Inport: '<Root>/MkcValctrlValveState'
224   *  Inport: '<Root>/SysPrsSigCalibFiltFast'
225   *  Inport: '<Root>/SystemPostRun'
226   *  Inport: '<Root>/TmcPrsSigUnLim'
227   *  Selector: '<S1>/Selector7'
228   *
229   * Block requirements for '<S3>/MkcBbwStateMachine':
230   *  1. (13423408-SW_REQ_13423409-v3x-9.1): Brake By Wire State machine-
231   */
232  if (mkc_bbw_state_module_DW.is_active_c3_mkc_bbw_state_module == 0U) {
233    mkc_bbw_state_module_DW.is_active_c3_mkc_bbw_state_module = 1U;
234
235    /*  system start */
236    mkc_bbw_state_module_DW.is_c3_mkc_bbw_state_module =
237      mkc_bbw_state_module_IN_MkcBbwDisabled;
238    SetD_MkcBbwCurrentState(MkcBbwDisabled);
239
240    /*  BrakeByWire deactivated by CoDeg */
241  } else if (mkc_bbw_state_module_DW.is_c3_mkc_bbw_state_module == 1) {
242    /*  brake by wire activated */
243    if (Get_CoDegStateBbw() != ((uint32_T)Wap_disabled)) {
244      mkc_bbw_state_module_DW.is_c3_mkc_bbw_state_module =
245        mkc_bbw_state_module_IN_MkcBbwEnabled;
246      mkc_bbw_state_module_DW.is_MkcBbwEnabled =
247        mkc_bbw_state_module_IN_MkcBbwIdle;
248      SetD_MkcBbwCurrentState(MkcBbwIdle);
249
250      /*  In idle the system stays functionally in hydraulic fallback lev
251      /*  reset counter */
252      mkc_bbw_state_module_DW.timeout_counter = 0U;
253    }
254  } else {
```

PiL – Example use cases – Jenkins CI (1)



- Commit model change
- Create Pull Request
- PiL-Test execution using Jenkins CI

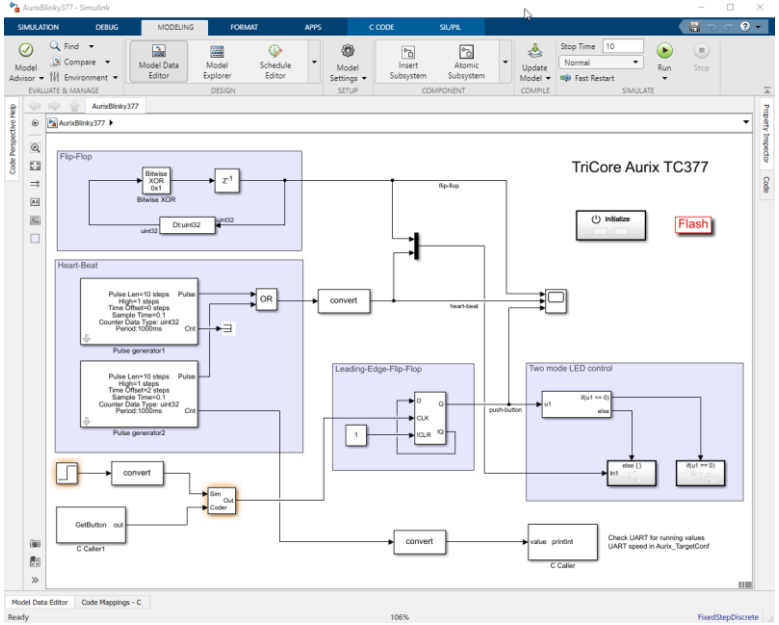


PiL – Example use cases – Jenkins CI (2)

- 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, …)


```
111    on Jenkins: compPath
112    D:\casdev\ci_ws\MkcBbwState\buildPipelineConfig.json used for configuration on Jenkins
113    Check for required license
114     Running simulinkPiLTestTests > PiL-Tests
115      Setting up simulinkPiLTestTests > PiL-Tests
116        Evaluating TestClassSetup: setupOnce
117      Done setting up simulinkPiLTestTests > PiL-Tests in 0.0009776 seconds
118       Running simulinkPiLTestTests > PiL-Tests/MiL-PiL-Test
119        Evaluating TestMethodSetup: setup
120        Evaluating Test: MiL-PiL-Test
121        Evaluating TestMethodTeardown: teardown
122       Done simulinkPiLTestTests > PiL-Tests/MiL-PiL-Test in 169.3993 seconds
123       Running simulinkPiLTestTests > PiL-Tests/MiL-PiL-Test Harness
124        Evaluating TestMethodSetup: setup
125        Evaluating Test: MiL-PiL-Test Harness
126
```

PiL-Target - Live Demo

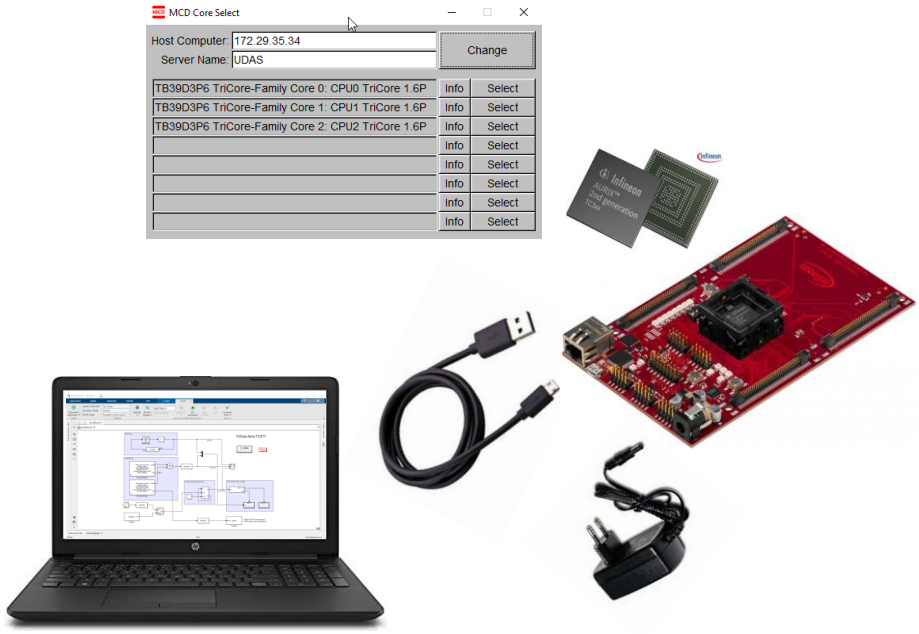


PiL-Target - Conclusion

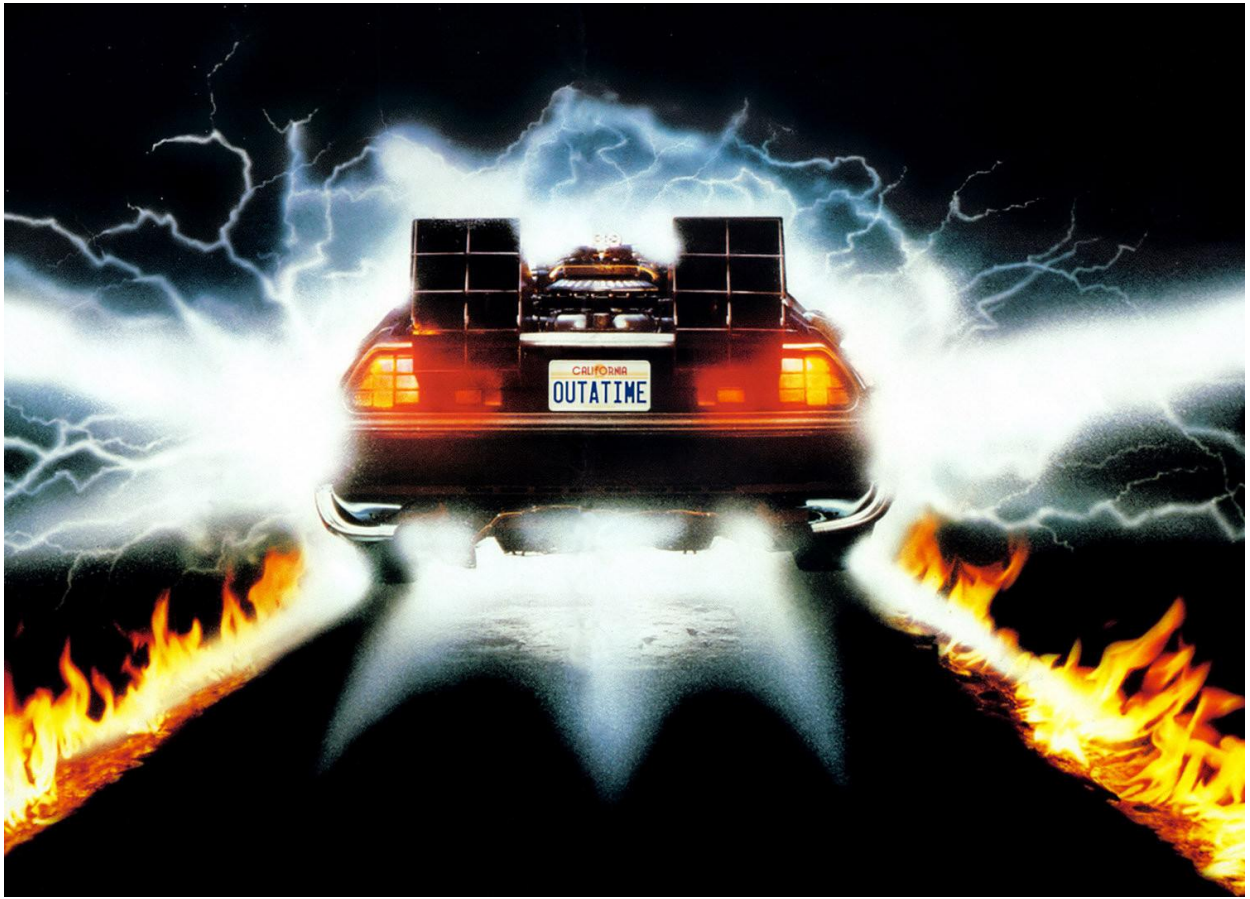
- Using a PiL-Target you can
  - validate the target compiler and linker
  - get an idea about the performance of your algorithm by the use of profiling
  - generate coverage data for your algorithm while running on the target processor
  - find hardware dependent issues early and fast
- If we had a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