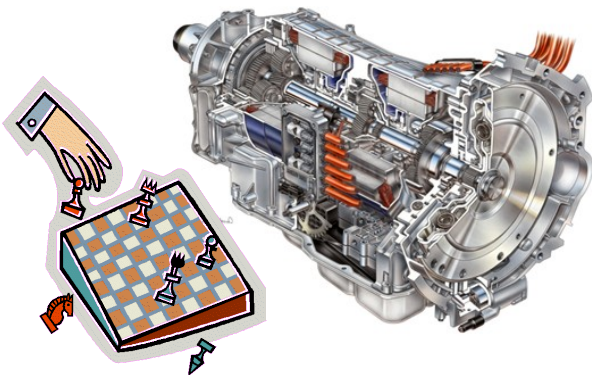




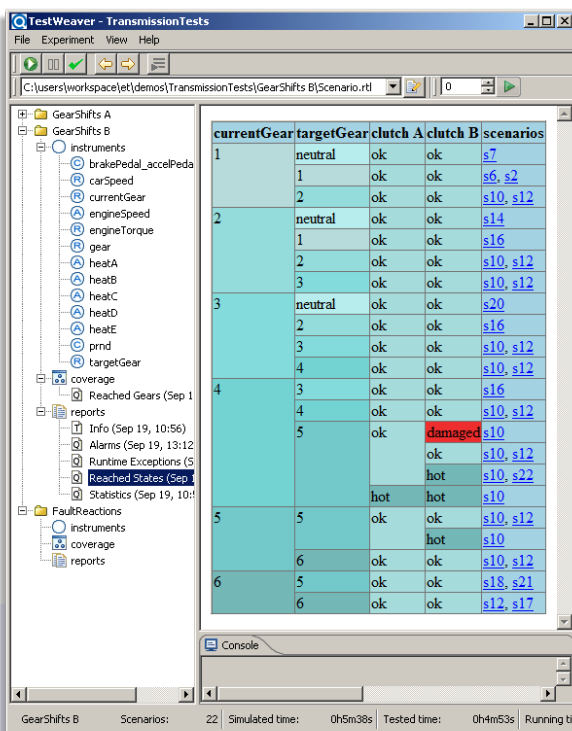
TestWeaver 2.3

Higher test coverage with less work



TestWeaver is a tool for automated test and validation of embedded systems using MiL, SiL, HiL simulation.

For testing a system, TestWeaver does not require predefined test scripts. TestWeaver generates, runs and evaluates thousands of tests autonomously – using a unique technology for intelligent test generation and evaluation of reactive systems.



Features

- automatic & intelligent test generation. Also integrated, classical test automation methods based on:
 - Interactive scenario recording and replay
 - Script-based test automation, e.g. with Python
- reactive test execution using MiL, SiL or HiL
- automatic evaluation and report generation
- positive and negative tests supported
- fault injection tests
- worst-case analysis
- reuse of tests for MiL, SiL and HiL possible
- 100% reproducible problems on MiL and SiL
- system-state coverage reports
- source-code coverage, e. g. with CTC++
- source-code debugging for C/C++ with MS Visual Studio

Advanced support for automotive applications via Silver:

- ASAP2/A2L, CAN, MDF, XCP, FMI, ISO 26262
- flashing of ECU vehicle configuration parameters
- range monitoring for all ECU signals
- monitoring stack and time required by control functions

Problems that can be found with TestWeaver:

- division by zero, access violation, infinite loops, non-determinism, range violations
- oscillations of discrete and continuous signals
- algorithmic errors, such as: state estimation significantly wrong for longer times
- system-level problems: poor shift quality, clutch overheating, inadequate fault reaction

Usually coding errors can be found using quite simple models, while quality assessment typically requires closed-loop simulation with a calibrated plant model.

Supported modeling and development environments:

- C/C++, for instance Microsoft Visual Studio
- MATLAB/Simulink, Real Time Workshop, Dymola
- via Silver: AMESim, SimulationX, FMU, Python

TestWeaver runs under Windows on standard PCs.

Benefit

- fast development: early problem detection
- high test coverage: thousands of high-quality tests
- less work: more automation, less scripting

How to use TestWeaver

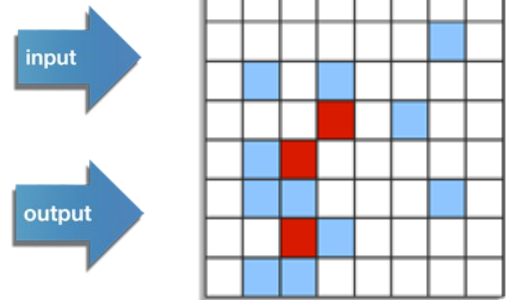
1

The starting point is an executable model of the system under test. This model consists typically of (i) a simulation model of the controlled physical components, e. g. a vehicle model, and (ii) the software controlling the system. In model-based development projects, such models are usually available anyway. You can use a wide range of tools to build the model, see frontpage. TestWeaver needs no access to the model source. You can test compiled models without knowing their source code.

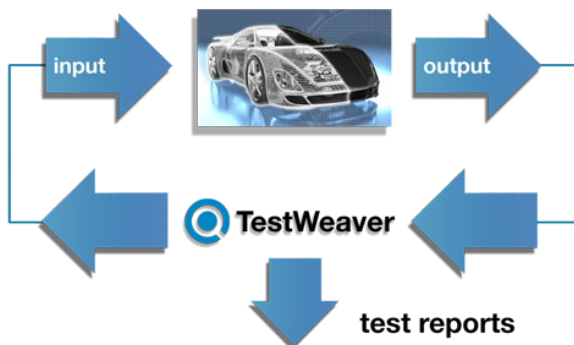


2

Select key inputs and outputs of the system. Typical inputs are acceleration and brake pedal, steering angle, road properties, and variables to control fault injection, e. g. sensor faults. Typical outputs are state variables of the control software, and key variables of the vehicle model, e. g. current and target gears, engine torque, etc. Classify the output values, to enable TestWeaver to distinguish desired from unwanted behavior. Likewise, classify input values, to enable TestWeaver to distinguish nominal inputs from inputs used to activate a component fault, e. g. of a sensor. For real valued inputs and outputs, classification means to partition the real axis in intervals, and assign the above properties to each interval. This turns the infinite space spanned by all inputs and outputs into a grid with a finite number of discrete states.



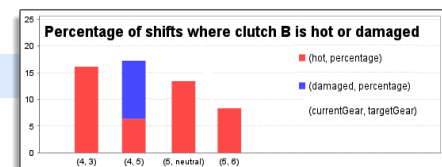
3



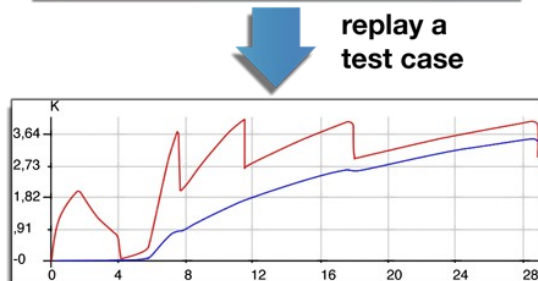
Connect simulation signals with TestWeaver. Connector libraries are provided, e. g. for C, Simulink, Dymola, Silver and Python.

Run TestWeaver: TestWeaver will autonomously drive the system model using the inputs and observe system response through the outputs. TestWeaver's goal is to drive the system into undesired states and to maximize coverage of the system states in the grid defined above, i. e. to reach every reachable discrete state at least once.

alarms	currentGear	targetGear	scenarios
Array index overflow	4	neutral	s13, s9
Access violation	5	neutral	s21, s37
Division by 0	3	neutral	s53, s75
clutchB overheating	4	5	s4, s8
targetGear oscillations	2	3	s62



4



Analyze the problems found by TestWeaver. The problems found and the coverage reached during an experiment is reported using tables and histograms. Every problem found can be replayed in the simulation or development environment for detailed debugging and analysis. Interesting tests can be collected in test databases, e. g. for regression tests, or can be exported into other tools, e. g. for HiL tests.

Our services

- Consulting for test and validation of systems
- Support for developing simulation models
- TestWeaver integration in development projects