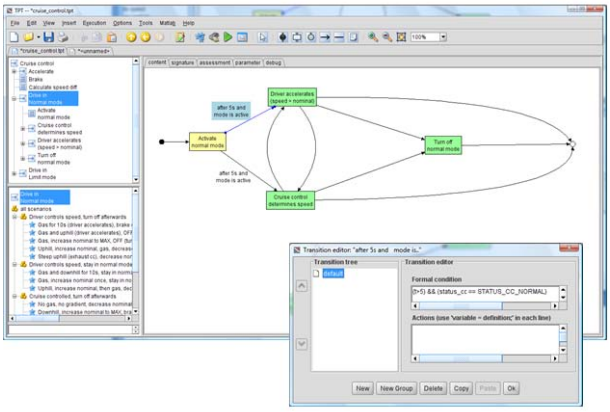
Time Partition Testing

Systematic automated testing of embedded systems

PikeTec GmbH, http://www.piketec.com

There are only a few tools for testing embedded systems in the automotive domain. Their function usually lies in the test-management or automation of tests by means of test scripts. Time Partition Testing (TPT) noticeably exceeds this. It combines a systematic and very graphic modelling technique for test cases with a fully automatic test execution in different environments and automatic test evaluation in a unique way. In doing so, TPT even supports testing of control systems.

Time Partition Testing (TPT) was developed to graphically and compactly model tests of embedded systems – especially those with continuous behaviour –, to automate those tests and to offer a systematic that supports the tester in selecting an ideal amount of test cases. The intuitive graphic modelling technique for test cases, the mechanisms for the structuring of complex test problems, the simple integration in any test-bench and a number of additional interesting features makes TPT a unique test solution. All this has contributed to TPT becoming the central testing tool in a large number of development projects at car manufacturers and suppliers and has become an integral part of the process chain.



Embedded systems

TPT is specialized on testing embedded systems whose inputs and outputs can be represented as signals. Most control systems belong to this system class. The implementation language of the embedded system (e.g. 'C' code, C++, MATLAB/Simulink/Stateflow, Statemate or a combination of multiple languages) is irrelevant for TPT. TPT does not require a particular implementation language. Because of this flexibility, TPT can be deployed in various different application areas.

Portability

Because TPT can operate irrespective of the platform, embedded systems can be tested at different stages of integration. Test cases modelled in TPT can be used in model in the loop (MiL) at early design stages, software in the loop (SiL) when the software is integrated, and hardware in the loop (HiL) when the software is in the target hardware environment as well as in real system environment without having to alter the test cases. Efficient testing at very low cost becomes possible through this form of reuse. Additionally, the test results can directly be compared between levels of integration.

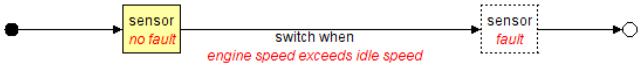
Continuous signals

Embedded software-based control systems interact with their real environment in real-time, are very complex and usually have a continuous but time discrete behaviour. Classic test approaches are not suitable for such systems since these are mostly specialized on the modelling of individual discrete values or value sequences and not on the description of continuous signals.

With its test modelling language, TPT provides mechanisms for the simple and easily understandable modelling of signals which accommodates the continuous character of the systems.

Reactive testing

With TPT, each test case can specifically react to the system's behaviour during the testing process in real time – for instance in order to react on the system exactly when a certain system-state occurs or a sensor signal exceeds a certain threshold. If, for example, a sensor failure for an engine controller is to be simulated when the engine idling speed is exceeded, it has to be possible to react to the event "engine idling speed exceeded" in the description of the test case.

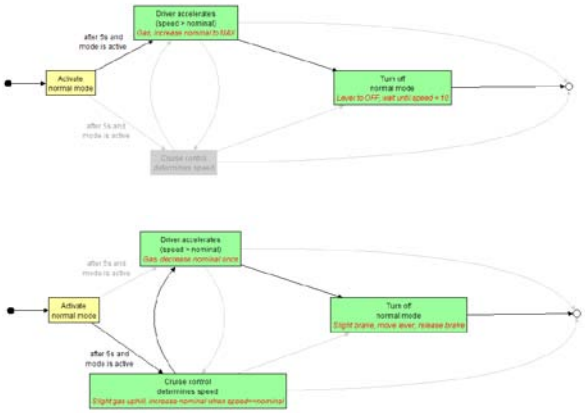


Graphic test cases

The exact process of individual test cases is modelled graphically with the aid of special state machines in TPT. Natural-language texts as an element of the graphics support the simple and demonstrative readability even for non-programmers. Substantial techniques such as parallel and hierarchical state machines, conditional branching, signal description as well as measured signals allow an intuitive and graphic modelling even of complex test cases.

Systematic test cases

TPT was developed specifically for testing of continuous and reactive behaviour of embedded systems. Because of its systematic approach in test case generation, TPT even keeps track of very complex systems whose thorough testing requires a large amount of test cases thus making it possible to find failures in the system under test with an ideal amount of test cases.



The underlying idea of TPT's systematic is the separation of similarities and differences among the test cases: most test cases are very similar in their structural process and can "only" be differentiated in a few, but crucial details. TPT makes use of this fact by jointly modelling and using joint structures. On the one hand, redundancies are thus avoided. On the other hand, it is made very clear what the test cases actually differ in – i.e. which specific aspect they respectively test. The comparability of test cases and thus the overview is improved in this approach and the attention of the tester is focused on the essential – the differentiating features of the test cases.

The hierarchical structure of the test cases makes it possible to break complex test problems down into sub-problems thus also improving the clarity and – as a result – the quality of the test.

These modelling techniques support the tester in finding the actually relevant cases, avoiding redundancies and keeping track of even large numbers of test cases.

What makes a good test?

All testers want automated tests – preferably right through from the test execution to the evaluation of the observed system behaviour to the documentation of the test results. This desire is perfectly understandable as automation is a necessary requirement for easy repeatable tests, clearly reproducible tests and complete regression tests after system adjustments or version changes. In this respect, it also stands to reason that the test infrastructure that is responsible for automation is given considerable importance in practice. It is, after all, the level of automation and thus the performance of the test infrastructure that determines the efficiency of the test.

The question of test efficiency is very important in many practical projects as it is bound up with costs and development time for the project. However, a second criterion that is just as important for an optimal test is the quality of the test.

Testing is never perfect, as it is well-known that no test in the world can guarantee to find all errors in a system. A test is only as good as its test cases: the aim is to check as many potential failures as possible with as few systematically selected test cases as possible. The more probable or critical an error is, the more relevant it is for the test to check.

In practice, the difficulty lies in deciding which of the almost endless number of potential faulty places are probable and critical, and thus test-relevant, and which can be ignored (without knowing which actual errors are present in the system).

Most of the existing test solutions and test languages unfortunately do not help this decision, but require the tester to select the test cases before he/she begins to implement the tests. This means that the selection is often only based on the tester's intuition. Time Partition Testing (TPT), however, has been developed with the aim of not only rendering automatable tests possible but also offering a systematic that helps the tester to select the most optimal number of test cases possible. This way TPT addresses both the efficiency of the test and the test quality.

Test evaluation

In addition to the modelling of the test cases, the expected system behaviour for individual test cases should also be automatically tested in order to assure efficient test processes. TPT offers the possibility to compute the properties for the expected behaviour online (during test execution) and offline (after test execution). While online evaluation uses the same modelling techniques as test modelling, offline evaluation offers decidedly more far-reaching possibilities for more complex evaluations, including operations such as comparisons with external reference data, limit-value monitoring, signal filters, analyses of state sequences and time conditions.
The offline evaluation is, technically speaking, based on the Python script language, which has been extended by specific syntactic language elements and a specialized evaluation library in order to give optimal support to the test evaluation. The use of a script language ensures a high degree of flexibility in the test evaluation: access to reference data, communication with other tools and development of one's own domain-specific libraries for test evaluation is supported. Besides of the script based test result evaluation user interfaces provide simple access to the test assessments and help non-programmers to avoid scripting.

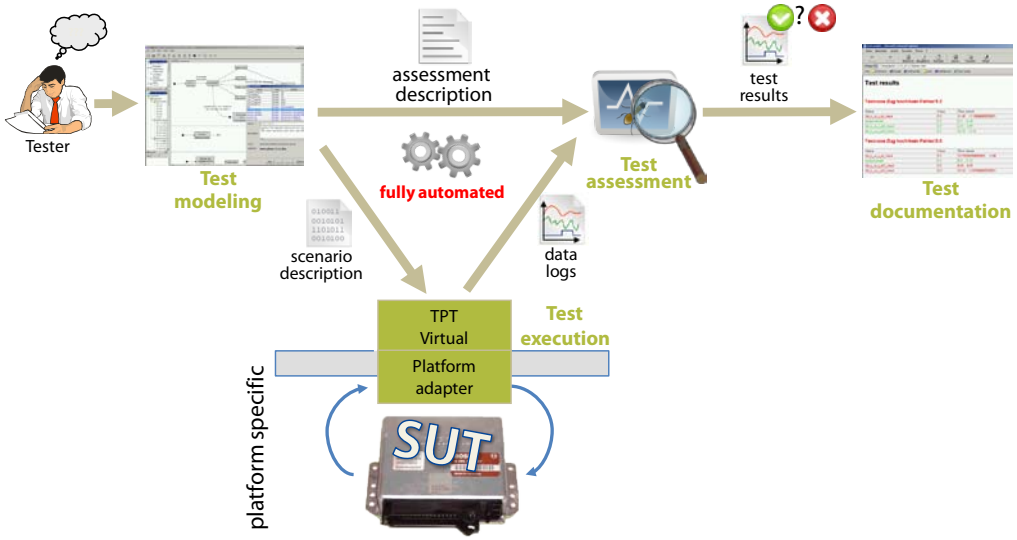
Tool integration

TPT can be very flexibly applied to almost all test and development environments for embedded systems. Such environments can be variously complex, ranging from a simple 'C' development environment to a complex HiL integration test bench.
If TPT is integrated with a test environment, TPT concentrates solely on the systematic modelling and automation of test procedures. More specific tasks, such as the activation of certain I/O interface cards for HiL test benches or code instrumentalisation for coverage analyses, continue to be performed by the tool environments that are specialized for these tasks. The interface between TPT and the environment has been kept clear and simple for a seamless integration.
This means that TPT does not require the development or test environments to be changed, but can be used as a supplement in the established environments.

Automation

TPT completely automates the test execution; test evaluation and test report generation.
Tests with TPT can be automatically performed in almost all test environments. The heart of the test execution is the so-called TPT Virtual Machine (TPT-VM), which optimally processes the test cases translated into special byte-code and thus controls the test. The integration of TPT into an existing test environment just requires the TPT-VM to be embedded into the test environment and signal data to be exchanged between the test environment and the TPT-VM at run-time – which is normally possible with little effort.

Appropriate integrations for example exist for tests of MATLAB/Simulink and TargetLink models, CAN, LIN, MiL, SiL and HiL-tests with PROVEtech:TA from MB-technology GmbH and MESSINA. Please contact PikeTec for integration in special environments.



Test Modelling: The tester models the test cases with the help of TPT's graphic description techniques. The procedure/sequence and the expected behaviour are precisely defined for each test case.
The test cases are automatically translated into byte-code. The definitions of the expected values are compiled into a Python script.
Test Execution: TPT's Virtual Machine controls the test procedure and exchanges signal data with the system to be tested at run-time. All signals to the system interfaces are recorded. Via MCD-3 automatic measurement of controller-internal signals via e.g. INCA are available.
Test Assessment: The expected properties are automatically assessed on the basis of the recorded signals.
Test Documentation: The resulting results of the test case are shown in the test report.

Real-time ability

TPT's Virtual Machine is able to process tests in real time with defined response behaviour. The response times of TPT test cases are normally given within micro seconds – depending on the complexity and test hardware.
The TPT-VM is implemented in ANSII-C and requires a memory of just a few kilobytes and can completely do without a dynamic memory allocation, allowing it to be applied in minimalist and environments with few resources too.

Test documentation

TPT presents the result of the test evaluation to the tester in a HTML, report, in which not only the pure information "success", "failed" or "unknown" can be depicted as the test result for each test case, but also details such as characteristic parameters or signals that have been observed in the test execution or computed in the test evaluation.
The content of the test documentation as well as the structure of the document can be freely configured with the help of a template.

Requirements tracing

Industry norms such as ISO 61508 or DO-178B require traceability of requirements and tests. TPT offers an interface to requirements tools like Telelogic DOORS in order to support these activities.