

Applying an Automatic Inspection Environment for Automotive Platforms

Akihisa Morikawa, Tomohito Mizuno, Ayumu Kataoka, Meika Sugimoto,
Kinuya Inoue, Mai Takeuchi and Hiroyuki Hattori
Witz Corporation

Copyright © 2008 SAE International

ABSTRACT

We have developed automotive platforms consisting of several components. Since testing these components takes so much time and will need even more in the future, we have developed an environment to inspect codes automatically and repeatedly. This paper shows the makeup and features of the inspection environment and improvement results. Moreover, we applied it to the performance evaluation of several operating systems (OS).

INTRODUCTION

We developed automotive platforms consisting of several components (the OSEK OS, the protected OS, the time triggered OS, the OSEK COM communication module, etc). We provide them for free to the public as open-source software.

If open-source software is shared widely, multiplex developments by software vendors will decrease and the development of high layer applications that users require will increase. Consequently, this industry can be developed. However, some people do not trust the quality of open-source software, so we did not gain as many users as we expected.

To improve the quality of open-source software, it is necessary to enrich component test variations. However, the increase in the number of tests is a problem. Since we have developed free software, we need to shorten inspection time and make efforts in quality improvement.

For example, as to the TOPPERS/OSEK kernel (Ref.1) that we developed as OSEK/VDX OS (Ref.2), we used the test suite that corresponds to the MODISTARC specifications which OSEK/VDX releases (Ref.3, Ref.4). We have more than 100 test sequences. It takes about 5 minutes to carry out only one test sequence on the M3A-ZA36 (CPU: M32102S6FP, Renesas), so we needed more than 9 hours to carry out all test sequences. We show the breakdown in Table.1. The majority are manual operations.

Test execution job	Execution time of one sequence (s)	Execution time of all tests (h)
Load the program	120	3.7
Execute	30	0.925
Save the execution log	120	3.7
Verify the result	30	0.925

Table 1: Items with the manual execution time of the component test of the TOPPERS/OSEK kernel.

Since we carried out a recurrence test, whenever we fixed the bugs that we discovered by test execution, the time needed for the test doubled.

Moreover, because the object code generated with the target processor and the compiler is different, it is necessary to execute the test separately for the built-in software. It corresponds to more than ten kinds of variations for the TOPPERS/OSEK kernel, and it is scheduled to increase in the future.

We summarized the time needed in Table 2. In the case of the component test of the TOPPERS/OSEK kernel, about 2 man-months are necessary today. Furthermore, twice the man-hours are necessary when we doubled the number of tests to improve test quality.

TOPPERS/OS EK kernel test suite	1 execution times (h)	Average execution frequency	Target variation	Total time (h)
MODISTARC specification	9.25	2-3	10	185-277.5
MODISTARC specification + test quality improvement (estimate)	18.5	2-3	10	370-555

Table 2: Time needed for component test of the TOPPERS/OSEK kernel.

Furthermore, we tested more than ten kinds of components, and the above-mentioned content applies to about all components equally. Moreover, more components are scheduled to be developed for the AUTOSAR platform, which is expected to be the next generation automotive platform in the future. In the AUTOSAR specification, 63 components are introduced. (Ref.5)

From the above, during the component development on which we will work in the future, "The test time is shortened while maintaining the component test quality" is an important topic that should be solved.

EXAMINATION OF METHODS

We aimed to shorten the test time by keeping the component test quality, and examined the following methods.

- Development of a non-dependent target platform
- Development of an automatic inspection environment

Development of a non-dependent target platform

This method develops a platform where we do not pay attention to the difference between the target processors and the compilers, and we developed components to correspond to this platform. In doing so, we can only guarantee the behavior of components to be tested on this platform, even if we do not perform the test on the real target. As a result, we can prevent the increase of test man-hours.

This method is effective for the component that does not contain the target-dependent part. However, most of the components we developed depend on the target, such as the OS and the communication drivers, so we cannot achieve success with this method.

Development of an automatic inspection environment

This method developed the environment to inspect components automatically and repeatedly. Presently we are loading the test program into the target manually. If we execute it automatically, it is assumed that people are not needed. Therefore we can inspect it repeatedly without considering man-hours.

The automotive components like OS and communication middleware that we test have various characteristics. In addition, they correspond to various targets and development tools. Therefore, we need an inspection environment supporting a variety of test objects. This is the technically difficult point.

We selected 'Development of automatic inspection environment', whose feasibility is high.

THE TARGET VALUE

We set a target value to develop an automatic inspection environment and utilize it. We carried out component tests 3 times or more for only one component. Because we want to see an effect in two years, we aim to execute tests 60 times or less. (60 times = 3 test execution * 10 components * 2 years)

In the graph of Figure 1, we compared the estimated man-hours of previous test executions and a test execution that uses an automatic inspection environment. We estimated that the development man-hours for the automatic inspection environment is 480 hours. If the test execution time can be reduced to 1/10 or less, we can get the correct effect by executing tests 60 times or more. (The turning point is the red dotted line.)

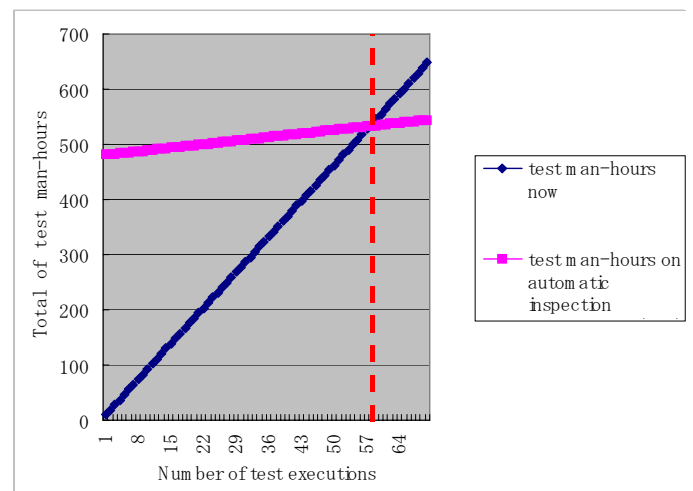


Figure 1: Frequency of component test execution and man-hours (estimate)

Depending on various factors, we show that we reduced the test execution time from the previous value to 1/10 or less as the target value.

AUTOMATIC INSPECTION ENVIRONMENT

Functional requirements

We list below the functional requirements necessary for an automatic verification environment.

- The test program is loaded into the target automatically.
- Automatic execution of test program on target
- Automatic check of test result
- Sequential execution of tests
- The previous test programs should be able to be reused.
- The component of various characteristics can be tested.
- It can work on various targets.
- The test program generated by various development tools can work.

System configuration

In order to fill the above-mentioned functional requirements, we made the system configuration shown in Figure 2.

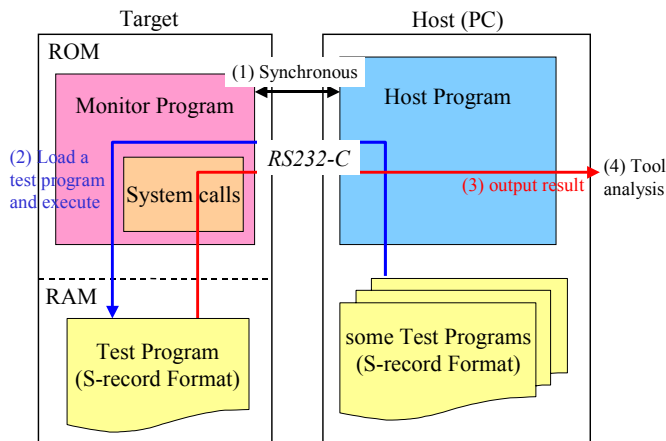


Figure 2: Composition chart of automatic inspection environment

This system consists of a target and the host. The host program forwards a test program to the target. The target program receives the test program from the host and puts it on RAM in the target. These processes only are repeated. Thus, it is the general system specifications that did not perform processing specialized in something.

As a result, our system has satisfied the four requirements below. To satisfy the requirement “the component of various characteristics can be tested”, it is necessary to make contrivances to create the test programs. (For instance, when testing the communication function between nodes of the communication middleware, it is necessary to implement the processing which sends the echo back from the other node on the test program.)

- The previous test program can be incorrectly assigned.
- It can work on various targets.
- The test program generated by various development tools can work
- The component of various characteristics can be tested.

Moreover it is possible to satisfy “The component of various characteristics can be tested” to create the test program, because it is a general specification. (For instance, when testing the communication function between nodes of the communication middleware, it is necessary to implement the processing which sends the echo back from the other node on the test program.)

Necessary functions for the host program are as follows.

- Synchronous establishment with monitor program
- Sequential execution of test program
- RS-232C I/O function
- Standard output function of RS-232C input information

Necessary functions for the monitor program are as follows.

- Synchronous establishment with host program
- The test program is loaded into RAM.
- Execution control of test program
- Interruption management
- RS-232C I/O function
- Provide system calls

Operation sequence

The operation sequence between the target and the host is shown in Figure 3.

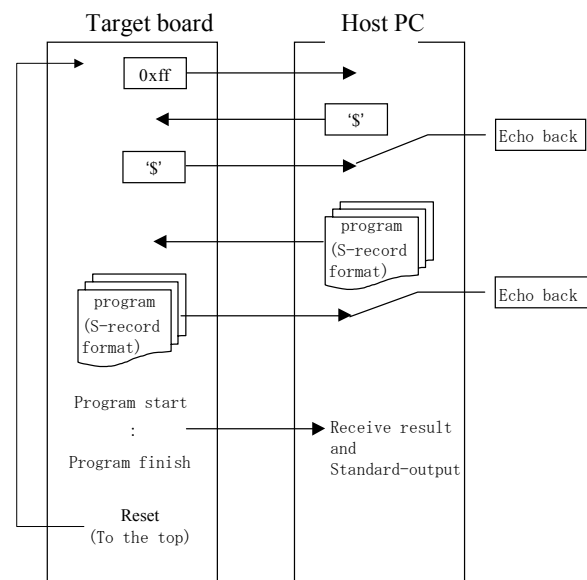


Figure 3 Operation sequence between the target and the host

- Synchronous establishment with host program

When the target is reset, the monitor program transmits 0xff via RS-232C. The host program receives it and transmits '\$' via RS-232C. When the host program receives the echo back '\$' from the target, it is considered a successful synchronous establishment.