Makara Journal of Technology

Volume 22 | Issue 3 Article 1

12-3-2018

# Automated Compilation Test System for Embedded System

Mohamad Khairi Ishak
*School of Electrical and Electronic Engineering, University Sains Malaysia, Nibong Tebal, 14300 Penang, Malaysia*, khairiishak@usm.my

Ooi Jun Hwan
*School of Electrical and Electronic Engineering, University Sains Malaysia, Nibong Tebal, 14300 Penang, Malaysia*

Teh Jiashen
*School of Electrical and Electronic Engineering, University Sains Malaysia, Nibong Tebal, 14300 Penang, Malaysia*

Nor Ashidi Mat Isa
*School of Electrical and Electronic Engineering, University Sains Malaysia, Nibong Tebal, 14300 Penang, Malaysia*

Recommended Citation
Ishak, Mohamad Khairi; Hwan, Ooi Jun; Jiashen, Teh; and Mat Isa, Nor Ashidi (2018) "Automated Compilation Test System for Embedded System," *Makara Journal of Technology*: Vol. 22: Iss. 3, Article 1.
DOI: 10.7454/mst.v22i3.3515
Available at: https://scholarhub.ui.ac.id/mjt/vol22/iss3/1

Makara J. Technol. 22/3 (2018), 115-122
doi: 10.7454/mst.v22i3.3515

# Automated Compilation Test System for Embedded System

Mohamad Khairi Ishak*, Ooi Jun Hwan, Teh Jiashen and Nor Ashidi Mat Isa

School of Electrical and Electronic Engineering, University Sains Malaysia, Nibong Tebal, 14300 Penang, Malaysia

*e-mail: khairiishak@usm.my*

## Abstract

Embedded system testing involves testing an integration of software and hardware. It is increasingly difficult to evaluate the functionality of each module within a short time because of the increasing number of tests required. In this paper, a novel stepwise methodology involving the use of an automated compilation test system (ACTS) is proposed, to improve the quality of testing and optimize the testing time using automation. Using the proposed method, the testing coverage can be maximized, while minimizing the manual work and testing time required. This ACTS was used to automate the test code compilation and execution for different hardware modules. The proposed method significantly saved the testing time by approximately 56.42%, compared to the existing method, while ensuring quality testing performance.

## Abstrak

**Sistem Uji Kompilasi Otomatis untuk Sistem Sisipan**. Pengujian sistem sisipan melibatkan menguji suatu penyatuan perangkat lunak dan perangkat keras. Menjadi bertambah sulit untuk mengevaluasi fungsionalitas masing-masing modul dalam waktu singkat karena peningkatan jumlah pengujian yang diperlukan. Di dalam karya tulis ini, diusulkan suatu metodologi bertahap baru penggunaan suatu sistem uji kompilasi otomatis (ACTS), untuk meningkatkan kualitas pengujian dan mengoptimalkan waktu pengujian dengan menggunakan otomatisasi. Dengan menggunakan metode yang diusulkan, jangkauan pengujian dapat dimaksimalkan, sekaligus meminimalkan waktu manual dan waktu pengujian yang diperlukan. ACTS ini digunakan untuk mengotomatisasi kompilasi kode uji dan eksekusi untuk modul-modul perangkat keras yang berbeda. Metode yang diusulkan mengamankan waktu pengujian secara signifikan mendekati 56,42%, dibandingkan dengan metode yang ada, sekaligus memastikan kinerja pengujian kualitas.

*Keywords: embedded, testing system, automated, software, hardware*

## 1. Introduction

An embedded system is an electronically controlled system with an integration of hardware and software. It consists of software application layers that utilize services provided by underlying system service and hardware support layers. A typical embedded system application consists of multiple layers and user tasks, which have different functionalities. Hence, in the design of embedded systems, the interactions among these different layers play very important roles.

There are two important classes of embedded systems, which are safety critical embedded system and technical scientific algorithm-based embedded system [1]. In addition, host-based embedded devices and target-based embedded devices are sub-classifications of embedded systems [2]. On the other hand, testing is an important process that is executed to identify any possible defect present in a system using different debugging methods. Testing needs to be run on an embedded system to determine the functionality of the system and ensure quality within the system development process [3], [4].

Errors may occur during the communication between hardware and software as embedded systems consist of both hardware and software. Moreover, defects that are detected in the later stage of system development may affect the product quality as well as lead to higher cost of resolution. Therefore, it is important to develop a timesaving testing technique to detect any possible interaction faults that may occur.

In this research, a testing methodology consisting of black box and white box testing is presented for testing embedded systems. This method can be used to automate testing and improves the quality of testing by reducing the testing time and maximizing the overall coverage. Furthermore, it can help to prioritize the test case based on the test coverage and defect area to prepare for the regression test.

## 2. Materials and Methods

### *A. Embedded System Testing*

**Embedded System Design.** An embedded system is typically designed with an assortment of software and hardware to perform specific tasks in computational environments. It is usually constructed with the least powerful computers that can meet the functional and performance requirements. Embedded systems generally use microprocessors that combine multiple functions of a computer on a single device. Figure 1 shows the typical structure of an embedded system in terms of four layers, where the first three layers consist of software, and the fourth consists of one layer of hardware.

Two operating systems, namely Linux and Windows Embedded, are popularly used to enable the implementation of embedded systems [5]. Most, but not all, embedded systems are real-time systems in which the correctness of an operation not only depends on its accuracy of functionality but also on the timing of the produced result. Embedded platforms are designed based on complex integrated systems, which involve different multitasking environments. In real-time embedded systems, each component can perform several different tasks at the same time, and this dramatically increases the interactions across components [6]. During the integration of different layers, if the interactions between components are not fully tested, unexpected results may occur and lead to more critical problems. Additionally, the temporal behavior is another important functional behavior in the real-time systems [7], [8]. For real-time embedded system, all requests involved must be handled and completed within the allocated period after the event has been triggered. Some general characteristics of embedded systems [9], [10], which have been observed from common experience, are discussed in the following sections:

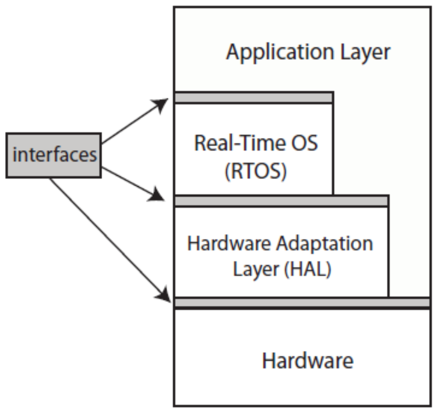

**Figure 1. Structure of an Embedded System [4]**

***Platform dependent***. Most embedded software systems are not tested in a single runtime environment. Because of the hardware-dependent characteristics of an embedded software, different test results may be obtained when the test is run on host or target environment [11]-[14]. Therefore, the embedded system must be tested in both host and target environments to enhance the test coverage; however, this increases the test development cost, and test distribution to both host and target environments is time-consuming.

***Accuracy of response.*** To determine the accuracy of an embedded system in meeting the pre-requirements, the system performance can be used as an indicator. Moreover, the response time taken for the request to complete a single transaction can be another suitable indicator to obtain the accuracy of embedded system [15].

***Limited resources (memory usage).*** Memory is an important element to consider during the design of an embedded system because of the limited memory allocated for different tasks.

**Software Testing.** Software testing is a process of developing and executing a test plan or written program with given inputs to ensure the program runs as designed and without errors. It is one of the important stages in system development, as it represents the final stage of validation of design and specification. An ideal software testing can identify any possible error with minimum execution time and maximum test coverage.

Software testing plays an important role in reducing any possible operating cost caused by a defect injected in the requirement specification and prevents such defect from persisting to the final stage of a product life cycle. Inadequate validation of the software system may lead to higher fixing cost. The condition can be worse if the defect is found in the field and requires recalling product from customers [16]-[18]. A successful and effective testing process reduces a defect lifecycle as short as possible; thus, most testing processes are involved in the early stage of development process.

Software testing should be carefully planned to develop an effective test strategy and avoid any major financial loss that can arise from poor planning [19]-[21]. During the software testing planning, the test strategy should be focused on balancing both testing costs and test coverage based on the criterion of maximum allowable defects [22], [23]. In general, software testing can be divided into two different categories, which are black box testing and white box testing [24], [25]-[27]. Both testing methodologies have their merits and demerits, depending on the testing requirements and scope. Furthermore, some important principles in software testing can affect the outcome and efficiency of the overall testing process, and they include: 1) Test case must be developed based on the requirements of the system; 2) A test case must contain a clear definition of the expected output or results; 3) Invalid and unexpected as well as valid and expected input conditions must be written in a test case; 4) A test case should be able to examine a program to check if it behaves as expected; 5) Created test cases should be repeatable for use in regression testing; 6) Every test result should be inspected and analyzed thoroughly. A test case must always be designed with the aim of finding a defect.

***B. Design of Embedded System Testing Technique***

An automated system was designed to improve the quality of testing by way of reducing the testing time, compared to manual testing, while optimizing (maintaining or improving) the overall test coverage. The design structure of the proposed test methodology comprises the following stages: 1) Analysis of system design; 2) Requirement specification; 3) Test code compilation and execution using a combination of different testing methods; 4) Test case prioritization; 5) Test sequence review.

A state diagram that presents the flow of the design structure is shown in Figure 2. By planning through the state diagram, every objective of the stages involved in the design architectures can be effectively achieved.

In the first stage, system design analysis, the overall design architecture of the system is analyzed. The terminologies of the communication between hardware and software are investigated to help in mastering the software development involved in the embedded system testing.

In the requirement specification phase, different environments of software code development platform are described and discussed in detail. In addition, the differences between different testing methods are identified to help in drafting test plans that can improve test coverage.

In the phase of test code compilation using a combination of different testing methods, different testing methodologies from both black box and white box testing are used to develop the most effective testing method. An automated compilation test system (ACTS) is proposed to automate the overall compilation process, which saves a lot of time, compared to manual testing.

In the test cases prioritization phase, test results are analyzed using tables and graphs. Afterward, the performances of both test compilation and execution are evaluated based on the total consumed time to further prioritize the test case for subsequent regression tests [28].

In the final stage of the proposed methodology, that is, test sequence review, test sequences are reviewed to determine a more effective testing flow for regression test. The goal of the review is to maximize the test coverage with minimum execution time.

Additionally, a PXIe embedded controller and several PXIe modular products from Keysight Technologies are used in the overall embedded system testing [29]. M9037A is a 3U, four-slot PXIe embedded controller from Keysight Technologies. One of the hardware configuration setups that was used to implement the proposed methodology in testing an embedded system is shown in Figure 3. In this setup, Keysight M9037A embedded controller is installed in the slot 1 of Keysight M9018A PXIe chassis together with M9381A vector signal generator and M9391A vector signal analyzer in slot 2 to slot 9.

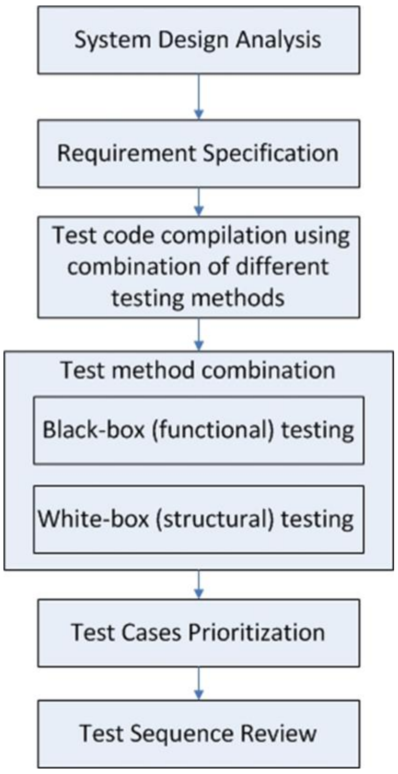
System Design Analysis
Requirement Specification
Test code compilation using combination of different testing methods
Test method combination
Black-box (functional) testing
White-box (structural) testing
Test Cases Prioritization
Test Sequence Review

**Figure 2. State Diagram of the Proposed Methodology**

**Figure 3. Hardware Configuration Setup**

**Test Code Compilation: Black Box Testing Method.** In the proposed test methodology, all-pairs testing, which is also known as pairwise testing, is used. This testing approach takes all possible discrete combinations of the parameter available in the test code using a combinatorial method. The total configuration combinations that are available in this embedded system testing approach are shown in Table 1.

For each programming language, there are three possible solution platforms, two solution configurations, and two configuration modes; hence, the total number of test cases to cover all possible combinations is 12 configurations (3 × 2 × 2 = 12) for every project file. After listing all possible discrete configurations, another testing method from white box testing, which is known as decision coverage testing, is used to further reduce the number of configurations needed to be tested.

**Test Code Compilation: White Box Testing Method.** The decision coverage testing approach is used to examine the internal structure of a project file to determine the valid solution platform before the test code compilation. This can ensure that every possible configuration from each decision point is executed at least once, thereby ensuring that all test configurations are covered thoroughly [30]. The flowchart of this decision coverage method that is applied in the test code compilation process is shown in Figure 4.

At the beginning of the process, the internal structure of the project file is read and examined. In the analysis process, the validity of every solution platform is examined, and for each valid platform, the compilation process is executed to ensure that no error occurs during the test code compilation. The result of each compilation is then processed and analyzed to export a proper and more readable test result.

According to the flowchart in Figure 4, first, the validity of "x86" solution platform is examined. If it is valid, the compilation process will be executed before proceeding to the next solution platform, which is "x64" platform checking. On the contrary, if the solution platform is not valid, the next solution platform will be checked. In summary, the whole solution platform checking process will be repeated until all four solution platforms have been examined. For each valid solution platform, the compilation process will be performed before checking the next solution platform.

**Automated Compilation Test System.** Generally, by implementing the proposed ACTS, an embedded system testing with different hardware modules can be tested based on the automated testing flowchart as shown in Figure 5. Here, most of the manually performed actions in the existing testing methods are automated to save the testing time and improve the quality of result.

**Table 1. Configuration System**

| No. | Solution Platform | Solution Configuration | Configuration Mode |
|---|---|---|---|
| 1 | (Win32/x86) | Debug | Simulation |
| 2 | x64 | Release | Hardware |
| 3 | Any CPU | | |
| **Total** | 3 | 2 | 2 |

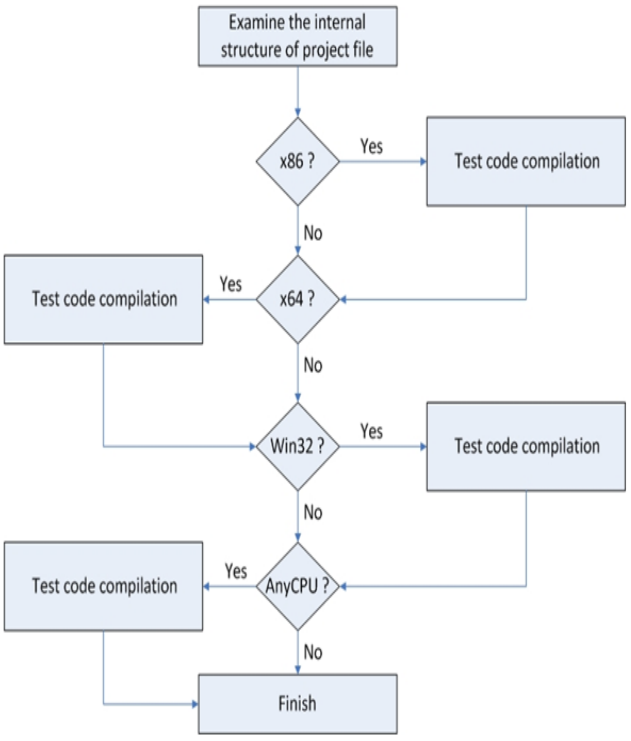

**Figure 4. Decision Coverage Flowchart**

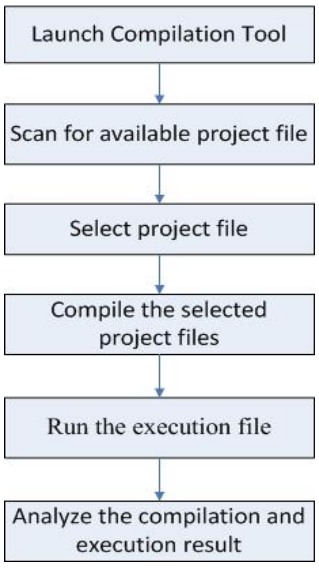
Launch Compilation Tool
Scan for available project file
Select project file
Compile the selected project files
Run the execution file
Analyze the compilation and execution result

**Figure 5. ACTS Flowchart**

Based on this automated testing flowchart, instead of launching the test code project file in MS Visual Studio one by one, the designed compilation tool is first launched. Afterward, a location path is specified, and all available test code project files are scanned and listed under a list box table. From the list box table, all project files are selected, and test code compilation is automatically performed. After the test code compilation is completed, all created execution files are automatically run without any user interaction. Finally, the compilation and execution test results that have been saved in text file format are analyzed together. In the proposed test methodology, the overall testing process is not repeated; hence, the testing time is less.

Furthermore, this software-based test methodology eliminates any restriction or dependency on instruments, and hence, its implementation is more flexible to the user.

## 3. Results and Discussion

The ACTS, which was developed in this research, has been fully utilized to reduce the manual testing time that is required in the overall embedded system testing process.

In an embedded system, different hardware modules were used to validate the integration of the hardware and software in the system. Every hardware module was developed with its own software driver to enable communication between the embedded controller and the individual hardware modules.

In every software driver of the hardware module, several test codes, which are also known as example codes, had been developed by the respective software driver developer. These test codes were developed in different languages such as C, C++, C#, and Visual Basic (VB) to provide different testing platforms. Moreover, every test code was developed with distinct functionalities to test the different integrations between the hardware and software.

**Test Code compilation.** From the compilation result, different analyses can be performed to further understand the status of the compilation test. Examples of some analyses that were performed on the M9018A compilation result are summarized in Table 2. From Table 2, a graph analysis can be further implemented to obtain a better overview of the compilation test result to help in regression test planning. Based on the respective solution platforms in Table 2, where each platform has two solution configurations ("Debug" and "Release"), project files for C and C++ will have four outputs, while project files for C# and VB will have six outputs. In other words, the total numbers of "pass" and "fail" must match the total outputs for each project file.

Table 2 shows a total of eight project files, which consist of two project files for each available language (C, C++, C#, and VB). Project files of C and C++ were compiled with two "pass" and two "fail," while all project files of C# and VB were successfully compiled without error. Through the combination of different solution configurations and solution platforms, there were four configurations for each of the example codes in C and C++, while for C# and VB, there were six configurations for each of the example codes.

Figure 6 demonstrates that for the same hardware module, only a slight difference existed between the project file or test code compilation times in both the simulation and hardware modes, as both modes were compiled using the same test code. Since the test code compilation is hardware-independent, both simulation mode and hardware mode did not significantly affect the compilation time.

**Test Code Execution.** After the project file or test code compilation, all execution files that were created during the compilation process were searched and executed using

**Table 2. M9018A Configuration Table**

| Project File | Platform | Configuration | P | F |
|---|---|---|---|---|
| AgM9018_CppIviC _Monitor | 2 | 2 | 2 | 2 |
| AgM9018_CppIviC _Trigger | 2 | 2 | 2 | 2 |
| CPP_M9018Monitor | 2 | 2 | 2 | 2 |
| CPP_M9018Trigger | 2 | 2 | 2 | 2 |
| CS_M9018Monitor | 3 | 2 | 6 | 0 |
| CS_M9018Trigger | 3 | 2 | 6 | 0 |
| VB_M9018Monitor | 3 | 2 | 6 | 0 |
| VB_M9018Trigger | 3 | 2 | 6 | 0 |

**Remarks:** P – Pass; F – Fail

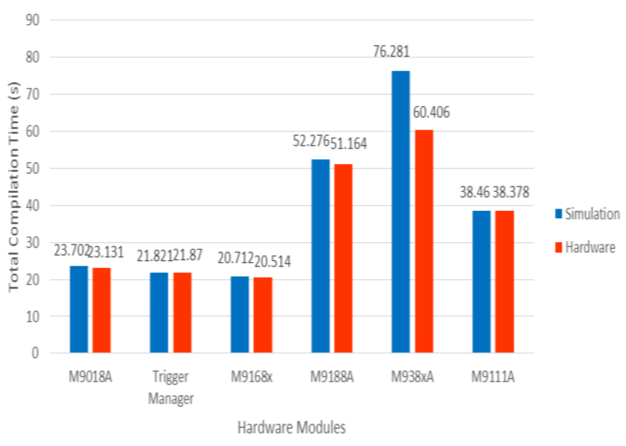

**Figure 6. Compilation Time Analysis for Different Modules**

ACTS. All result contents which were generated from the execution files were saved and exported to a result file in text format. In addition, the total execution time that was required to complete the overall execution process was recorded in the result file as an indicator element to be used in regression test planning.

Besides the test execution result, the execution time needed for test code execution of different hardware modules using the ACTS were recorded to show the comparison between different hardware modules. Afterward, graph analysis was performed to display a clearer overview of the result (Figure 7).

Figure 7 shows that the execution times for all execution files created during the compilation process were longer when ran in hardware mode, compared to simulation mode. Different hardware modules had different execution times, depending on the total project files available for each module and the complexity of the test code developed. In this case, the M938xA module was not included in the comparative analysis because of the high contrast between its execution time in the hardware mode compared to other modules.

**Comparison between different methods.** Every hardware module had different project files that were developed with different customized test codes. Moreover, different test codes may have different complexities, which can affect the total compilation and execution times. Through the experiment and implementation of ACTS on different hardware modules, the obtained average times consumed are compared in Table 3.

This comparison table shows the total testing time, including compilation and execution times, for the project files. Since every project file had different compilation and execution times because of different design structures, an average time was taken to make a general comparison between the existing manual testing method and the proposed automated testing method. For 30 configurations, the average time consumed for both compilation and execution using ACTS was calculated through the result analysis.

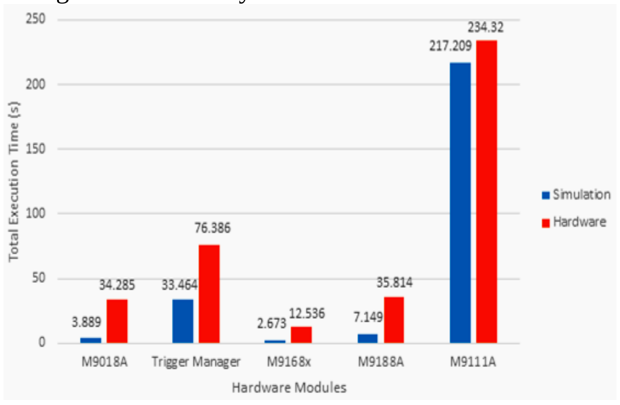

**Figure 7. Execution Time Analysis for Different Modules**

From Table 3, the total time saved by implementing the proposed methodology in a system of 30 configurations was around 1388 seconds, which is around 56.42% of the time consumed using the existing method. In other words, time consumption was greatly improved. Moreover, all manual or manpower-operated testing, including test code compilation and execution, in the existing method are automated in the proposed test system. In the existing method, because of the large-number of configurations that are involved for every single hardware module, a lot of time and manpower are consumed for solution platform and solution configuration switching. Hence, the implementation of the proposed methodology not only saves time but also saves manpower resources.

Furthermore, the test compilation result which was saved in csv format further reduced the extra work of reinserting the result manually. Through the graph analysis, the total number of tests that were “pass” or “fail” can be easily identified. For example, in Table 2, the compilation test passes in C# and VB but fails in C and C++. With this type of graph analysis, testing can be prioritized to focus on the failing area during regression test planning [31].

Although the testing sequence can be prioritized based on this graph, it still depends on the testing team to decide whether to continue testing on project files from C# and VB to ensure the resolution implemented do not affect or cause any breakdown on the existing test status. The continuous testing of all project files can help to maintain the stability of the software by ensuring no new error is introduced into the system after the changes are implemented.

**Table 3. Testing Method Comparison Table**

| **Testing Phase** | **Testing Method** | **Time (Seconds)** | | **Time Saved (s)** |
|---|---|---|---|---|
| | | **EM** | **ACTS** | |
| Test code compilation and execution | Code compilation/ configuration | 300 | 22 | 278 |
| | Code execution/ file exploring | 600 | 30 | 570 |
| | Result analysis | 900 | 900 | 0 |
| Test case prioritizatin | Graph analysis | 60 | 60 | 0 |
| Test sequence review | Regression test | 600 | 60 | 540 |
| **Total time (seconds)** | | 2460 | 1072 | 1388 |
| **Total improvement (%)** | | 1072 ÷ 2460 × 100 = 56.42% | | |

**Remarks:** EM – Existing method

Other than this, an execution file that is compiled in the simulation mode will output a predefined value that is hardcoded in the development codes. While for an execution file compiled in the hardware mode, interaction between the hardware and software will be first established before starting data retrieval from the hardware module or making data changes on the hardware module. In general, execution time for execution file in hardware mode is longer than in simulation mode, as in the former, there are interactions between the hardware and software involved in the overall process. Furthermore, the possibility of acquiring an error is higher in hardware mode compared to simulation mode, which makes the testing in hardware mode more important than that in simulation mode.

From the results and conducted analyses, the implementation of the proposed methodology improved the testing process by way of reducing the required testing time and manpower resources while ensuring quality testing performance from both software and hardware perspectives.

## 4. Conclusion

In this study, an ACTS comprising both black box and white box testing methods is proposed to enhance testing effectiveness. In the system, test case prioritization and test sequence review are performed after the test execution to help in planning an effective regression testing. Moreover, several functional modules were developed in batch command script and C# programming language to help in completing the whole research. For proof of concept, the proposed method was implemented on different hardware modules in an embedded system, where the general characteristics of this method were exhibited. The implementation of the proposed method resulted in significant time saving, as most of the actions in the existing manual testing were automated; hence, a testing time reduction of around 56.42%, compared to the existing method, was realized, while maintaining the same coverage. Through different analyses using table and graphs, it was demonstrated that this proposed methodology can be effectively implemented for embedded system testing.

## References

[1] S.P. Karmore, A.R. Mahajan, International Conference on Emerging Trends in Engineering and Technology, Nagpur, 2013, pp. 46-47.

[2] M. Wahler, E. Ferranti, R. Steiger, R. Jain, K. Nagy, International Conference on Software Testing, Verification and Validation, Montreal, QC, 2012, pp. 457-466.

[3] R. Harwahyu, A.S. Manaf, B.S. Ananto, B.A. Wicaksana, R. F. Sari, J. Comput. Sci. 9/6 (2013) 810.

[4] S.P. Karmore, A.R. Mahajan, International Conference on, Colombo, 2013, pp. 567-572.

[5] J. Guan, J. Offutt, Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on, Graz, 2015, pp. 1-10.

[6] Wei-Tek Tsai, L. Yu, F. Zhu, R. Paul, IEEE Software. 22/4 (2005) 75.

[7] S. Kukolj, V. Marinkovic, M. Popovic, S. Bognar, Engineering of Computer Based Systems (ECBS-EERC), 2013 3rd Eastern European Regional Conference on the, Budapest, 2013, pp. 153-156.

[8] M.A. Wehrmeister, L.M. Ceron, J.L.d. Silva, Computing System Engineering (SBESC), Brazilian Symposium on, Natal, 2012, pp. 119-124.

[9] H. Wu, Systems and Informatics (ICSAI), 2012 International Conference on, Yantai, 2012, pp. 2524-2527.

[10] H.M. Qian, C. Zheng, International Conference on, Wuhan, 2009, pp. 1-5.

[11] M.A. Wehrmeister, An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems. M¨unster: Verlag Monsenstein und Vannerdat, 2009.

[12] M.A. Wehrmeister, E.P. Freitas, C.E. Pereira, F.R. International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07), Santorini Island, 2007, pp. 428-432.

[13] D.K. Saini, H. Saini, 3rd International Conference on Quality, Reliability and INFOCOM Technology (Trends and Future Directions), Indian National Sciences and Academics, New Delhi (India) Conference proceeding, 2006.

[14] D.K. Saini, N. Gupta, J. Inf. Technol. 3/3 (2007) 17.

[15] J. Liu, E.A. Lee, IEEE Control Syst, 23/1 (2003) 75.

[16] S.M.A. Shah, D. Sundmark, B. Lindström, S.F. Andler, in IEEE Access, 4 (2016) 1859-1871.

[17] B. Qu, Z. Chen, Y. Lu, Future Computer and Communication (ICFCC), 2010 2nd International Conference on, Wuhan, 2010, pp. V2-370-V2-373.

[18] B.G. Van Treuren, J.M. Miranda, IEEE Des.Test. Comput. 20/2 (2003) 25.

[19] U. Connie, Smith, G. Lloyd Williams, Performance Solutions A Practical Guide to Creating Responsive, Scalable Software. Boston: Pearson Education, vol. 20, no. 5, 2003.

[20] H. Jin, L.Y. Chen, L.M. Zeng, B.L. Li, International Conference on, Sichuan, 2008, pp. 243-247.

[21] L. Shuping, P. Ling, Computer Science and Information Technology, 2008. ICCSIT '08. International Conference on, Singapore, 2008, pp. 463-466.

[22] N. Chouhan, M. Dutta, M. Singh, Computational Intelligence and Communication Networks (CICN), 2014 International Conference on, Bhopal, 2014, pp. 1106-1112.

[23] M. Sharma, B.S. Chandra, in Software Engineering Advances (ICSEA), 2010 Fifth International Conference on, 2010, pp. 459-464.

[24] T. Murnane, K. Reed, in Software Engineering Conference, Proceedings. 2001 Australian, 2001, pp. 12-20.

[25] S. Liu, Y. Chen, J. Syst. Software. 81/2 (2008) 248.

[26] J.H. Hayes, A.J. Offutt, in Software Reliability Engineering, Proceedings. 10th International Symposium on, 1999, pp. 199–209.

[27] J.H. Andrews, S. Haldar, Y. Lei, C. Felix, H. Li. Tool Support for Randomized Unit Testing. Proceedings of the First International Workshop on Random Testing, July 2006 (RT'06). pp. 36-45.

[28] D. Stotts, M. Lindsey, A. Antley. An Informal Formal Method for Systematic JUnit Test Case Generation, Technical Report TR02-012 April 2002. pp 2-12.

[29] J. Hartmann, C. Imoberdoff, M. Meisinger. UML-Based Integration Testing, International Symposium on Software Testing and Analysis, ACM Press, 2000, pp. 60-70.

[30] H. Yuan, T. Xie Substra: Proceedings of the 2006 International Workshop on Automation of Software Test (Shanghai, China, May 23-23, 2006). AST '06. ACM Press, New York, NY, 2006, pp. 64-70.

[31] Q. Gu, B. Tang, D. Chen, International Symposium on Parallel and Distributed Processing with Applications, Taipei, 2010, pp. 419-426.