The model of software execution time remote testing

Ratybor Chopey, Bohdan Knysh,
Dmytro Fedasyuk

Software Department, Lviv Polytechnic National University, UKRAINE, Lviv, S. Bandery street 12, E-mail: chopey.ratybor@gmail.com, knysh.bohdan@gmail.com, fedasyuk@gmail.com

Abstract – The work deals with the problems of testing embedded systems in the case when the latter are geographically distributed, which is gradually getting more common. The analysis of the application domain has revealed the lack of information concerning remote testing of the execution time of an embedded system. The authors have investigated into the possibility of evaluating the duration of time-critical functions of a distant embedded system. We've introduced a model and an algorithm for measuring the firmware execution time remotely that sustained approbation with a number of experiments.

Keywords – execution time testing, worst-case execution time, embedded system, Keil uVision, UVSOCK.

I. Introduction

Software engineering in general and quality assurance in particular are constantly evolving, with the latter have been providing sophisticated methods, processes and tools for testing software no matter how complex it might be. Meticulous quality assurance of modern software assumes unit testing, integration testing, and other reputed approaches. Most of these methods are not easy to apply to embedded systems because of the specific nature of such systems, their hardware restrictions and the fact that the firmware execution depends on the performance of all the peripheral devices included into any embedded systems which makes the behavior of the latter less determined due to the fact that the latency of peripheral devices might be arbitrary to some extent. Besides, many functions of embedded systems are not supposed to be tested by software engineers, because typically an embedded system being developed is a part of some larger system that is resided in a distant place and thus is not available during the phases of developing and testing. Hence, in order to check whether the embedded system being developed works properly, they use various simulators.

The fact that the whole embedded system cannot be fully accessed caused evolution of static methods for testing the software execution time. These methods do not assume the actual execution of the software being under evaluation either in hardware or in simulator. The input data for these methods may be presented by the source code and, in addition, by the hardware architecture model. The difficulties of using static methods for analyzing software execution time are as follows: the results might be and typically are too pessimistic, besides, creation and analysis of hardware models are time-consuming. As a result, the system gets excessively backed-up.

These problems are addressed by a number of software execution time testing methods [1-4], the efficiency of which has been proved when testing real embedded systems. However, such methods are not intended for remote testing.

In order to solve the problem of testing a remote embedded system the authors of [5] have proposed the architecture of a tool that gives the possibility to test a system via TCP. The proposed tool allows its users to access all the resources of an embedded system, test the latter automatically and archive the testing results. The developed tool is effective for integration testing but not applicable for software execution time testing. Since embedded systems are typically hard real-time systems, careful evaluation of software execution time is of primary importance.

All the above-mentioned leads to the idea of investigation into the possibility of remote testing software execution time directly in an embedded system being subjected to quality assurance.

II. The model of remote testing process

The model of remote testing process assumes that a client-server architecture, shown in Fig. 1, should be used.

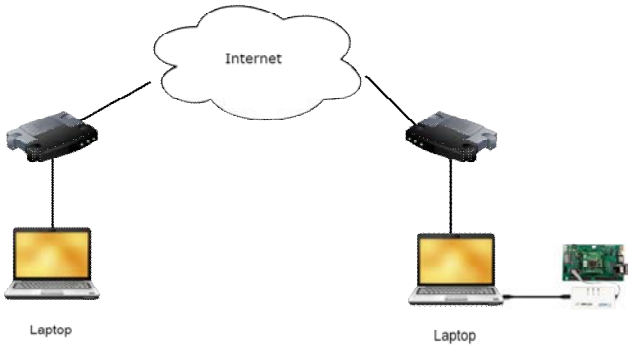

Fig.1 The model of software execution time remote testing process

A server might be a personal computer which is connected with an embedded system being tested via a programmer. A TCP server created by Keil uVision, allows the QA engineers to use standard integrated development environment's features including variables watch and control, breakpoints, controlling the code execution in a debug mode, etc. A client is a personal computer, which executes the testing algorithm, sends test data and instructions to the server, and measures the software execution time. Keil uVision's tools are accessed via API uVision Socket Interface [6].

III. Testing algorithm

The developed algorithm assumes that the software execution time is to be measured using two breakpoints, one at the beginning, another at the end of the code fragment being tested. The time elapsed between two breakpoints should be measured. The remote testing algorithm is shown in Fig. 2

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

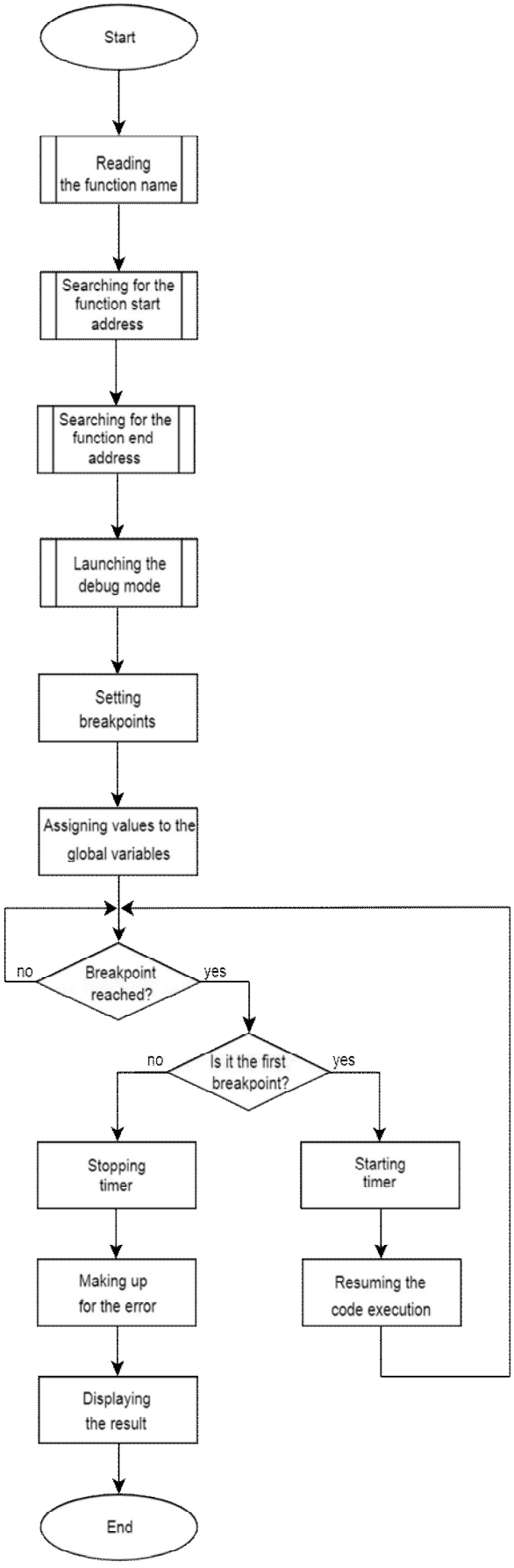

Fig.2 The block diagram of the algorithm of software execution time remote testing

Step 1. Reading the name of the function the execution time of which is to be tested.

Step 2. Seeking for the address where the assembler-like code of the function is placed in the embedded system's program memory. At this step we perform syntax analysis of the map-file auto-generated by the IDE during compilation along with other auxiliary files. Any map-file contains the names of all the functions included into the assembly, and the names of all the global variables along with the addresses indicating where these variables reside in RAM. In the case if no traces of the function have been detected in the map-file, the algorithm sends a corresponding message to the user interface.

Step 3. Seeking for the end of the function. This step requires syntax analysis of the listing file, another file generated during compilation. The file contains a C-language code and its assembler-like "translation".

Step 4. Setting up Keil uVision in the debug mode and starting the firmware execution in a real embedded system.

Step 5. Setting up breakpoints at the function's beginning and ending addresses. When entering into a breakpoint the integrated development environment Keil uVision which serves in this case as a TCP sever generates an event. The event causes the corresponding callback-function to be invoked on the client side. The callback function is the most significant part of the proposed algorithm since it's the very function responsible for measuring the execution time of the function being tested.

Step 6. Assigning such values to the corresponding global variables that would cause control flow to the function to be tested.

Step 7. When entering into the callback-function indicating that some preset breakpoint has just been reached, we start the timer in order to measure the code execution time.

Step 8. Sending an instruction prescribing that the program should resume its execution.

Step 9. When the callback-function is triggered again (because the breakpoint corresponding to the end of function has been reached), we stop the timer, define the error of program execution time measurement, make up for it and show the result. The nature of the error and the ways of compensation for it are described in the next section.

IV. Measurement inaccuracy

The error of program execution time measurement is a sum of methodological error and the random error. The methodological error is caused by the measurement algorithm and can be defined by (1).

$$t_A = t_{Start} + t_{CHO} + t_{Stop} \quad (1)$$

where t_{Start} is the execution time of the function that resumes firmware in the embedded system;

t_{CHO} is the duration of the procedure that checks the number of the breakpoint which caused invocation of the callback-function.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

t_{Stop} the execution time of the function stopping the timer used for measurements.

In order to define the methodological error we evaluate the computational complexity of each algorithm's working stage that affects the total duration:

$$O_A = O_{Start}(2) + O_{CHO}(1) + O_{Stop}(2) \quad (2)$$

where $O_{Start}(2)$ is the computational complexity of the function that resumes the firmware in the embedded system.

$O_{CHO}(1)$ is the computational complexity of the procedure that checks the number of the breakpoint which caused the invocation of the callback-function.

$O_{Stop}(2)$ is the computational complexity of the function that stops the timer.

In accordance with the performed calculations of the computational complexity we determine the methodological measurement error using the formula $t_A = O_A(5)/N$, where *N* is the amount of instructions per second (for a personal computer).

The random error is caused by the delay of transmitting data via the Internet:

$$t_N = t_{StartN} + t_{BP} \quad (3)$$

where t_{StartN} is the duration of sending the instruction, prescribing that the remote embedded system should resume executing its firmware, via the Internet;

t_{CHO} is the duration of sending the instruction, telling the remote embedded system that it should invoke the callback function, via the Internet;

The duration of transmitting any instruction via the Internet is a random value that is comprised of the delay of signal transmitting, the delay of processing it in network nodes, and the delay in the receiving buffers [7, 8].

In order to find out the delay of sending packages between personal computers via the Internet, we use Ping utility.

The software execution time is calculated as:

$$t_{EX} = t_M - t_A - t_N \quad (4)$$

where t_M is the measured software execution time.

VII. Experiments

In order to investigate into the proposed remote testing algorithm experimentally we've implemented the latter in the form of a separate software unit written in C#.

Experiments were conducted using two personal computers Lenovo W520 including processors with the clock frequency of 2.2HHz and RAM 8 GByte and a real-time embedded system running under control of STM32F407 microcontroller with the clock frequency 8 MHz.

The function chosen for testing was the implementation of the bubble sort algorithm (Fig. 3).

```
void bubblesort(int *a, int n)
{
 for(j = 0; j < n – 1; j++)
 {
  for(i = 0; i < n – 1; i++)
  {
    if (arr[i] > arr[i + 1])
    {
      tmp = arr[i];
      arr[i] = arr[i + 1];
      arr[i + 1] = tmp;
    }
   }
 }
}
```

Fig.3 The function implementing the bubble sort algorithm

The choice of an algorithm, implementation of which was to be tested, can be attributed to the following facts: the algorithm is well-known and fully researched, it has relatively low computational complexity (N^2) and the time of its execution depends solely on the amount of instructions per second for the specific microcontroller.

We've tested the bubble sort algorithm on an array of 100, 200, 500, 1000, 2000, 5000 and 10000 items.

The results of testing the execution time of the function that implements the chosen sort algorithm performed on different amounts of input data are summarized in Table 1.

TABLE 1

THE TESTING RESULTS

№ of items	Calculated execution time, s	Measured execution time, s	Average Ping, s
100	0,00125	0,00349	0,035
200	0,005	0,0063	0,041
500	0,03125	0,034275	0,52
1000	0,125	0,127	0,045
2000	0,500	0,504	0,045
5000	3,125	3,144	0,055
10000	12,500	12,514	0,054

The dependence of the relative measurement inaccuracy on the execution time of the function implementing the sort algorithm is depicted in Fig. 4.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua

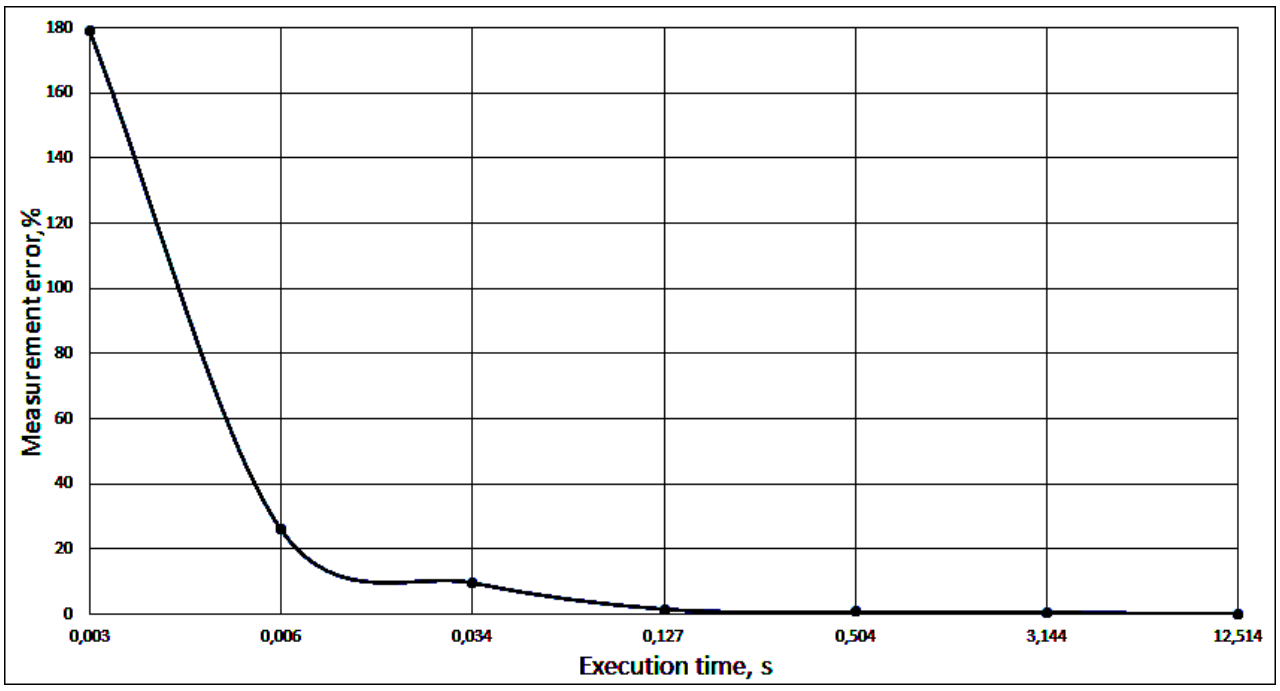

Fig. 4 The dependence of the relative measurement error on the function's execution time

Conclusion

Upon the proposed model and algorithm of software execution time remote testing we've investigated into the possibility to test embedded systems remotely in practice.

Having performed a number of experiments we researched the influence of delays in sending data via the Internet on the obtained results. It has been detected that measurement of delays in sending five packages and averaging the results cannot provide a sufficient accuracy for compensating for the measurement error. This fact can be attributed to the delays' being of arbitrary nature and their dependence of the network route selected for packages being sent and of the readiness of network nodes to process packages.

The performed investigation into the relative measurement error has proved that the error increases when the execution time of the function being tested reduces. This means that the developed algorithm is reasonable to use when testing time-consuming threads and functions.

In order to reduce the relative measurement error it's reasonable to use auxiliary algorithms that force the choice of the route for sending packages to the remote embedded system via the Internet, for network technologies allow us to select a fixed route. This would narrow down the amount of factors contributing into the random part of the measurement inaccuracy and make the averaged delay of sending a package more relevant.

References

[1] M. Wahler, E. Ferranti, R. Steiger, R. Jain and K. Nagy, "CAST: Automating Software Tests for Embedded Systems", 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012.

[2] R. Kirner, "The WCET Analysis Tool CalcWcet167", Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, pp. 158-172, 2012.

[3] D. Fedasyuk, R. Chopey and B. Knysh, "Architecture of a tool for automated testing the worst-case execution time of real-time embedded systems' firmware", 2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 2017.

[4] J. Engblom, A. Ermedahl and F. Stappert, "Structured Testing of Worst-Case Execution Time Analysis Methods", in 21st Real-Time System Symposium, Orlando, 2000, pp. 154–163.

[5] J. Perpiñán, "Remote Testing of Embedded Software", Software Quality, pp. 259-271, 2001.

[6] "Application Note 198: Using the uVision Socket Interface", Keil.com, 2017. [Online]. Available: http://www.keil.com/appnotes/docs/apnt_198.asp. [Accessed: 29- Sep- 2017].

[7] M. Klumash, O. Lavriv, B. Buhyl and R. Bak, "Model' zabezpechennya parametriv yakosti obsluhovuvannya systemy rozpodilu mul'tyservisnoho trafiku", Visnyk Natsional'noho universytetu "L'vivs'ka politekhnika", vol. 705, pp. 138-144, 2011.

[8] K. Trubchaninova and K. Polyakova, "Doslidzhennya propusknoyi zdatnosti merezhi dostupu v zalezhnosti vid typu abonenta", Informatsiyno-keruyuchi systemy na zaliznychnomu transporti, vol. 5, pp. 23-28, 2013.

Lviv Polytechnic National University Institutional Repository http://ena.lp.edu.ua