VOL. 12, NO. 9, MAY 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

TESTING DISTRIBUTED EMBEDDED SYSTEMS USING ASSERT MACROS

K. Chaitanya[1], Sastry JKR[2], K. N. Sravani[2], D. Pavani Ramya[2] and K. Rajasekhara Rao[3]
[1]Department of CSE, JNTU Hyderabad, India
[2]Department of E.C.M, K. L. University, Vaddeswaram, India
[3]Department of C.S.E, Usha Rama College of Engineering, Vijayawada, India
E-Mail: kilaru.chaitanya84@gmail.com

ABSTRACT

Distributed Embedded systems are being used these days for monitoring and controlling many applications which are either critical or non-critical. No formal frameworks as such have been presented which can be used for testing the distributed embedded systems. Many approaches have been presented in the literature for testing stand-alone embedded systems which include testing through scaffolding, assert macros, instruction set simulators, logic analysers, in-circuit emulators, monitors etc, It has to be explored to find how best these methods can be used for testing the distributed embedded systems. In this paper, an investigation on use of asserts macros for testing the distributed embedded system is presented. The method has been used to test existing distributed embedded systems that monitors and controls temperatures within a Nuclear reactor system.

Keywords: distributed systems, embedded systems, assert macros.

1. INTRODUCTION

A standalone embedded system must be tested for proper functioning of hardware, software and considering both together. Majority of the code can be tested through several methods that Scaffolding, instruction set simulators, third party tools. The hardware can be tested through logic analysers. Existence of a proper hardware environment can be tested using assert macros. The proper working of the Hardware along with Software can be tested through in-circuit emulators and monitors. Each method aims at testing a particular aspect of proper functioning of an embedded system.

An Assert macro is one good technique to test existence of particular Hardware environment required for executing some software segments. If the required hardware environment is not in existence, the software initiated to be executed shall fail making the entire embedded system non-functional and in-operational. A macro takes a single parameter and evaluates to find whether it is TRUE or FALSE. If the evaluation is TRUE, the assert macro does nothing and the program execution is undertaken in the normal way. If the parameter evaluates to FALSE, assert causes the program to crash, usually printing some useful message along the way perhaps something like "ASSERT FAILE at line 411". Assert macros are used to enable a program to check for finding the existence of the environment required for executing a specific program. Assert Macros can be used to check status of a radio, whether any of the control variables have been set to NULL, existence of a frame or otherwise etc. Programing languages support different kinds of assertions to verify the existence of a proper environment.

The assert macro helps to bring bugs to light sooner rather than later and gives at least some clue about what the problem is as opposed to the nameless, faceless crash one often gets, for example, NULL pointers). This will be very helpful when the embedded system is tested on the host. On the target, however, most embedded systems don't have a convenient display on which assert can print a message. Further, in the application environment, assert calls exit or abort or some other function that stops the application and returns control to the operating system; no corresponding function exists in an embedded system. The assert macro compiles to no code when one compile the code with NDEBUG. This ability to define asserts out of existence is important for two reasons. The first reason being that the assert macro should not crash the system when shipped to the customers. The second reason being that the assert macro should not degrade the performance of the system when it is activated. The assert macro can be deactivated on target using NDEBUG at the time the code is compiled for the target. Assertions are great because they support better testing, make debugging easier by reducing the distance between the execution of a bug and the manifestation of its effects, serve as executable comments about preconditions and post conditions, can act as a gate way drug to formal methods.

Many applications are being developed using several embedded systems which are networked using one of the standard protocol systems which include RS485, I^2C, CAN, USB etc. The way the communication happens between the embedded systems connected on to the same network depends on communication system used for networking. In a distributed system, entire application is divided into sub-system and each sub-system is made to work on a single embedded system. The make the sub-applications to run on an independent system, the entire hardware is also distributed among different embedded systems. A communication system is implemented to ensure that the total processing is done as if a single embedded system is running the entire application.

Testing the distributed embedded system is even complicated as many heterogeneous issues must be taken into account. In addition to testing the individual

functions, the functions which are used for effecting the communication among the embedded systems must also be tested. Standalone embedded systems can be tested using the methods such as scaffolding, Assert Macros, Instruction set simulators, Logic Analysers, in-circuit emulators, monitors etc. The testing methods use a specific mechanism for generating the test case and then proceed to testing the same on live environment.

Many complications exists when testing a distributed embedded system has to be carried using Assert Macros. The complications include the following:

a) A separate computer (HOST) is required for testing at each of the individual embedded system considering the hardware that is used to implement the embedded system that is one of the processing systems within the distributed embedded system. Each of the embedded system is different and generally built using a different microcontroller based system which is heterogeneous considering specifically the processor architecture.
b) The working of all the individual systems and the working of all the embedded systems together had to be tested considering all the heterogeneous issues and the communication issues.

There is a need to consider a topology to interconnect the individual embedded system and a communication method must also be selected like CAN to facilitate communication between the embedded systems

Test cases are required for undertaking testing using assert macros. Environment testing is generally undertaken using the assert macros. The test cases typically includes

a) Checking for existence of environment required for proper execution of the ES applications
b) Checking for availability of signals
c) Checking for proper asserting of the actuators
d) Checking for proper values of the parameter to be in a range

The testing for proper environment as such is distributed. Therefore while testing at individual location can be undertaken, the combined result of undertaking testing considering all the computing locations has to be arrived at. The process flowrequired for undertaking the testing of distributed embedded systems using assert macros is shown in Figure-1.

Figure-1. Process flow for undertaking testing using Assert Macros.

It can be seen that Macros are generated based on the kind of testing that must be conducted at each of the location. Test cases that must be used for testing each of the distributed embedded are generated by following a separate process all together. The generated test cases are included into the source code by way of pointing mechanism that is manually in built into the code. Macros are inserted into the source code and then compiled to get an executable which can be run on the HOST machine. Test results are produced as an outcome of executed assert macros with the source ES application. Three distinct test processes are required for undertaking testing using assert macros which include Generation of assert macros based on the inputted test cases, Including the Assert Macros into source code and merging the test results and produce test outcomes which represent working of the ES system as whole.

2. PILOT PROJECT DESCRIPTION

Monitoring the temperatures within nuclear reactor tubes is one of the most important issues when it comes to uranium enrichment. Sensors are mounted on to the nuclear reactor tubes which are distantly situated. Many temperatures at various points within each of the Nuclear reactor tube must be sensed and it is also necessary to maintain proper gradients across various points at which the temperatures are measured. When temperatures rise above some pre-defined levels, coolants have to be injected into the tubes to bring the temperature down. Pumps are used for injecting the coolants into the tubes. The temperature sensing and implementing the actuating mechanisms that control the process of pumping is achieved through various embedded systems. The operators must be alerted when the temperature gradients go beyond uncontrollable levels through asserting a buzzer and lighting a pattern of LEDs as the case may be.

A historical database of temperatures sensed, pumping levels implemented, temperature gradients, status of triggering buzzer etc., are written on to a PC into a database for providing the historical evidences. Each part of sensing and actuating requires a kind of response time and therefore needs to be sensed, monitored and controlled individually through a separate embedded system. There is a need for coordinating the functions between the individual embedded systems for achieving the sensing and actuating in real time. This leads to the need for interconnecting the individual embedded systems that help in establishing the communication between the embedded systems which are individually responsible for either sensing, actuating or monitoring the process taking place within the Nuclear reactor system.

Designing, development and implementing the networking of embedded systems becomes one of the most crucial issues when it comes to distributed embedded systems. One of the major issues that must be addressed is heterogeneity that exists among different types of Microcontroller based systems which are used for developing and implementing different parts of a distributed embedded system. These requirements leads to implementation of distributed embedded systems, each designated to monitor and control either the sensing or actuating mechanisms with the need for the centralised coordination between the distributed embedded systems. The individual embedded systems have being networked using 485 protocols. Protocol conversions have been used wherever no native support for the protocol exists. Figure-2 shows various decentralised embedded systems with built-in interfaces along with individual embedded systems that provide centralised coordination. Interfacing distributed embedded systems through RS485 protocol requires protocol conversion when no native support exists within the respective individual embedded systems.

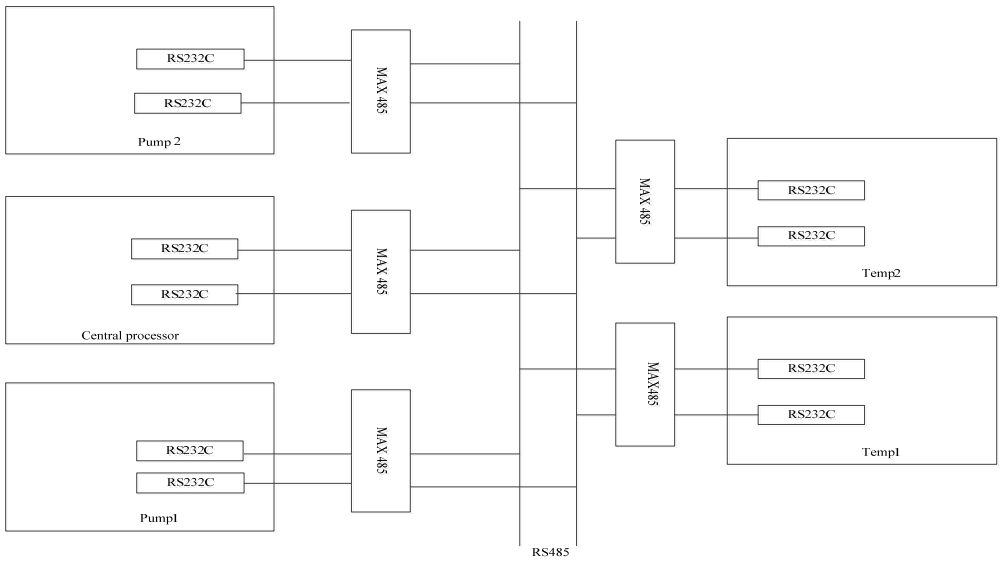

Figure-2. Top level view of a distributed embedded system.

Some of the Major Testing requirements that must be tested across the entire distributed embedded system (Pilot project) are shown in the Table-1. It has been noted in the table, the way a test case is realized considering the method used for undertaking the testing and the locations that are involved in undertaking the testing. The test cases that must be used for undertaking testing using assert Macros are shown in Table-2.

Table-1. Requirements for testing pilot distributed embedded system.

Serial number	Test case	Testing method	Testing location
1	Read Temp-1 and write to LCD	Scaffolding, Assert Macro	Location-1
2	Test the RS485 based communication between the 89C51(system-1) and the central Micro Controller System-5)	Scaffolding	Location-5
3	Read Temp-1 and send to Central Micro Controller	Scaffolding	Location-1
4	Read Temp-1 and measure throughput	Scaffolding, Simulator	Location-1
5	Test the RS485 based communication between the 83C51(system-3) and the Central Micro Controller System-5)	Scaffolding	Location-5
6	Read Temp-1 and make Pump-1 on if Temp-1>Reference Temp-1	Scaffolding	Location-1,3,5
		Assert Macros	Location -5
7	Read Temp-1 and make Pump-1 off if Temp-1<Reference Temp-1	Scaffolding	Location-1,3,5
		Assert Macros	Location-1
8	Read Temp-1 and make buzzer on if> Temp-2	Scaffolding, Assert Macros	Location-1,3,5
		Assert Macros	Location-5
9	Read Temp-1 and make buzzer off if> Temp-2	scaffolding	Location-1,3,5
		Assert Macro	Location-5
10	Test response between the Reading the Temp-1 and starting the pump-1	Logic Analyser, Scaffolding	Location-1,3,5
11	Test response between the Reading the Temp-1 and stopping the pump-1	Logic Analyser, Scaffolding	Location-1,3,5
12	Test response between the Reading the Temp-1 and starting the buzzer	Logic Analyser, Scaffolding	Location-1,5,5
13	Test response between the Reading The Temp-1 and stopping the buzzer	Logic Analyser, Scaffolding	Location-1,5,5
14	Read Temp-2 and measure Throughput	Scaffolding, Assert Macro	Location-2
15	Test the RS485 based communication Between the 89C51(system-4) and the Central Micro Controller (system-5)	Scaffolding	Location-2
16	Read Temp-2 and make pump-2 on if Temp-2 > Reference Temp-2	Scaffolding	Location-2,5,4
		Assert Macros	Location-4

Serial number	Test case	Testing method	Testing location
17	Read Temp-2 and make pump-2 off if Temp-2 < Reference Temp-2	Scaffolding	Location-2,4,5
		Assert Macros	Location-4
18	Read Temp-2 and make buzzer on if > Temp-1	Scaffolding	Location-2
		Assert Macros	Location-5
19	Read Temp-2 and make buzzer off if < Temp-1	Scaffolding	Location-2,5,5
		Assert Macros	Location-5
20	Test response between the Reading the Temp-2 and starting the Pump-1	Logic Analyser, Scaffolding	Location-2,4,5
21	Test response between the Reading the Temp-2 and stopping the pump-2	Logic Analyser, Scaffolding	Location-2,4,5
22	Test response between the Reading the Temp-2 and starting the Buzzer	Logic Analyser, Scaffolding	Location-2,5,5
23	Test response between the Reading the Temp-1 and stopping the buzzer	Logic Analyser, Scaffolding	Location-2,5,5

Table-2. Test cases for testing through assert macros.

Master test case serial	Test case	Location for testing
1	Test for proper sensing of Temp-1 signal at the output of the Temp-1 sensor	Location-1
6	Test for proper sensing of temp-1 and Pump-1 to be on	Location-5
7	Test for proper sensing of Temp-1 and Pump-1 to be off	Location-1
8	Test for sensing Temp-1, Temp-2 and Buzzer of condition	Location-5
9	Test for sensing Temp-1, Temp-2 and Buzzer on condition	Location-5
14	Test for proper sensing of Temp-2 signal at the output of the Temp-2 sensor	Location-2
16	Test for Temp-2 and proper asserting of Pump-2 signal to be on	Location-4
17	Test for Temp-2 and proper non assertion of Pump-2 signal	Location-4
18	Test for Temp-2 and buzzer on	Location-5
19	Test for Temp-2 and buzzer off	Location-5

3. RELATED WORK

Changes to an embedded system are inevitable. Changes to embedded system meant for monitoring and controlling a mission critical or safety critical system must be carried online. The programs/ tasks which have been changed while embedded system is up and running must be tested thoroughly before the same are brought to the running state. Sasi Bhanu *et al*., [1] have presented a method for undertaking online testing of a standalone embedded system through test processes that are added to the production system. The method proposed by them do not use any of the standard testing methods used for testing the standalone embedded system. The have presented a testing mechanism that gets evolved dynamically. The testing processes as such get evolved dynamically.

This paper addresses the need to integrate formal assertions into the modelling, implementation, and testing of state chart based designs. An iterative process which is meant for the development and verification of state chart prototype models augmented with state chart assertions using the State Rover tool has been presented by Doron Drusinky *et al*., [2]. The novel aspects of the proposed process include writing formal specifications using state chart assertions, JUnit-based simulation and validation of state chart assertions, JUnit-based simulation and testing of state chart prototype model segmented with state chart assertions, automatic, Junit based, white-box testing of state chart prototypes augmented with state chart assertions, and spiral adjustment of model and specification using the test results. They have demonstrated the process with a prototype of a safety-critical computer assisted resuscitation algorithm (CARA) software for a casualty intravenous fluid infusionpump.

Software testing is a very expensive and time consuming process. It can account for up to 50% of the total cost of the software development. Distributed systems make software testing a daunting task. Hany *et*

al., [3] have described a novel multi-agent framework for testing 3-tier distributed systems. They have described framework architecture as well as the communication mechanism among agents in the architecture. Web-based application is examined as a case study to validate the proposed framework. The framework is considered as a step forward to automate testing for distributed systems in order to enhance their reliability within an acceptable range of cost and time.

Hui Liu, Maozhong J *et al*., [4] have presented a testing tool for distributed embedded software. They have presented a Distributed Embedded System Simulating Environment (DESSE) that could simulate a highly configurable hardware environment for software testing. The DESSE could simulate real-time network using regular Full-Duplex Fast Ethernet. To support software testing, the DESSE has the ability to monitor the system and take all kinds of tests with scripts. The tool, to some extent could solve some of the difficulties in the distributed embedded software testing: the"probe effect", non-repeatability, and the lack of a synchronized global clock, the complexity of the hardware configuration and the lack of hardware resources.

Dynamically testing software that has been augmented with assertions increases the defect Observability of the test cases provided that the assertions happen during testing. Jeffrey Voas *et al*., [5] have presented an approach to localise the assertion based on finding regions of the code that appear to be un testable and then making them more testable. This is accomplished through a combination of static and dynamic analyses. They have also explored the phenomenon where assertions which are designed to boost the fault observability provided by test scheme cannot lower the fault observability provided by a different testing scheme and in fact, may actually increase it. This demonstrates a unique and cost effective benefit of assertions that have not been exploited earlier and lays forth, a new avenue for finding higher return-on-investment testing techniques

Executable assertions can be inserted into a program to find software faults. Unfortunately, the process of designing and embedding these assertions can be expensive and time consuming. Hwei Yint James RI *et al*., [6] have developed the C-Patrol tool to reduce the overhead of using assertions in C programs. C-Patrol allows a developer to reference a set of previously defined assertions, written in virtual C, bind assertion parameters, and direct the placement of the assertions by a pre-processor

Embedded assertions have long been recognized as a potentially powerful tool for automatic runtime detection of software faults during debugging, testing and maintenance, yet despite the richness of the notations and the maturity of the techniques and tools that have been developed for programming with assertions, assertions area development tool that has seen little widespread use in practice. The main reasons seem to be due to previous assertion processing tools did not integrate easily with existing programming environments, and it is not well understood what kinds of assertions are most effective at detecting software faults. David S. Rosenblum *et al*., [7] have an assertion processing tool that was built to address the concerns of ease-of-use and effectiveness. The tool is called APP, an Annotation Pre-processor for C programs developed in UNIX-based development environments. APP has been used to develop a number of software systems over the past three years. They have presented a classification of the assertions that were most effective at detecting faults. While the assertions that are described guard against many common kinds of faults and errors, the very commonness of such faults demonstrates the need for an explicit, high-level, automatically checkable specification of required behaviour.

Two kinds of interface contract violations can occur in component-based software: A client component can fail to satisfy a requirement of a component it is using, or a component implementation can fail to fulfil its obligations to the client. The traditional approach to detecting and reporting such violations is to embed assertion checks into component source code, with compile-time control over whether they are enabled. This works well for the original component developers, but it fails to meet the needs of component clients who do not have access to source code for such components. A wrapper-based approach, in which contract checking is not hard-coded into the underlying component but is "layered" on top of it, offers several relative advantages Stephen H. Edwards *et al*. [8]. It is practical and effective for C++ classes. Checking code can be distributed in binary form along with the underlying component, it can be installed or removed without requiring recompilation of either the underlying component or the client code, it can be selectively enabled or disabled by the component client on a per-component basis, and it does not require the client to have access to any special tools (which might have been used by the component developer) to support wrapper installation and control. Experimental evidence indicates that wrappers in C++ impose modest additional overhead compared to in lining assertion checks.

Dynamically testing software that has been augmented with assertions increases the defect observability of the test cases provided that the assertions have been executed during testing. J. VOAS*et al*., [9] presented an approach to assertion localization that is based on finding regions of the code that appear to be untestable and then making them more testable which is accomplished through a combination of static and dynamic analyses. They have also explored the phenomenon where assertions which are designed to boost the fault observability provided by a test scheme cannot lower the fault observability and order by a different testing scheme and in fact may actually increase it. This demonstrates a unique and cost effective benefit of assertions not before exploited and lays forth a new avenue for finding higher Return on investment testing techniques.

Thorough testing of distributed systems, particularly peer-to-peer systems can prove difficult due to the problems inherent in deploying, controlling and monitoring many nodes simultaneously. This problem will only increase as the scale of distributed systems continues

to grow. Daniel Hughes [10] has presented a frame work that implements a test bed environment using a semi-centralized peer to-peer network as a substrate for sharing resources made available from standard PCs. This framework automates the process of test-case deployment using a combination of Reflection and Aspect Oriented Programming. This allows 'point-and-click 'publishing of software onto the test-bed. The framework also provides a common monitoring, control and logging interface for all nodes running on the network. Together, these features greatly reduce deployment-time for real-world test scenarios. Automated insertion and removal of test code also ensures that the testing process does not compromise the correctness of the final system.

4. INVESTIGATIONS AND FINDINGS

4.1 Environment setting for testing distributed embedded system thorough assert macros

Figure-3 shows the environment setting required for undertaking testing through Assert macro method.

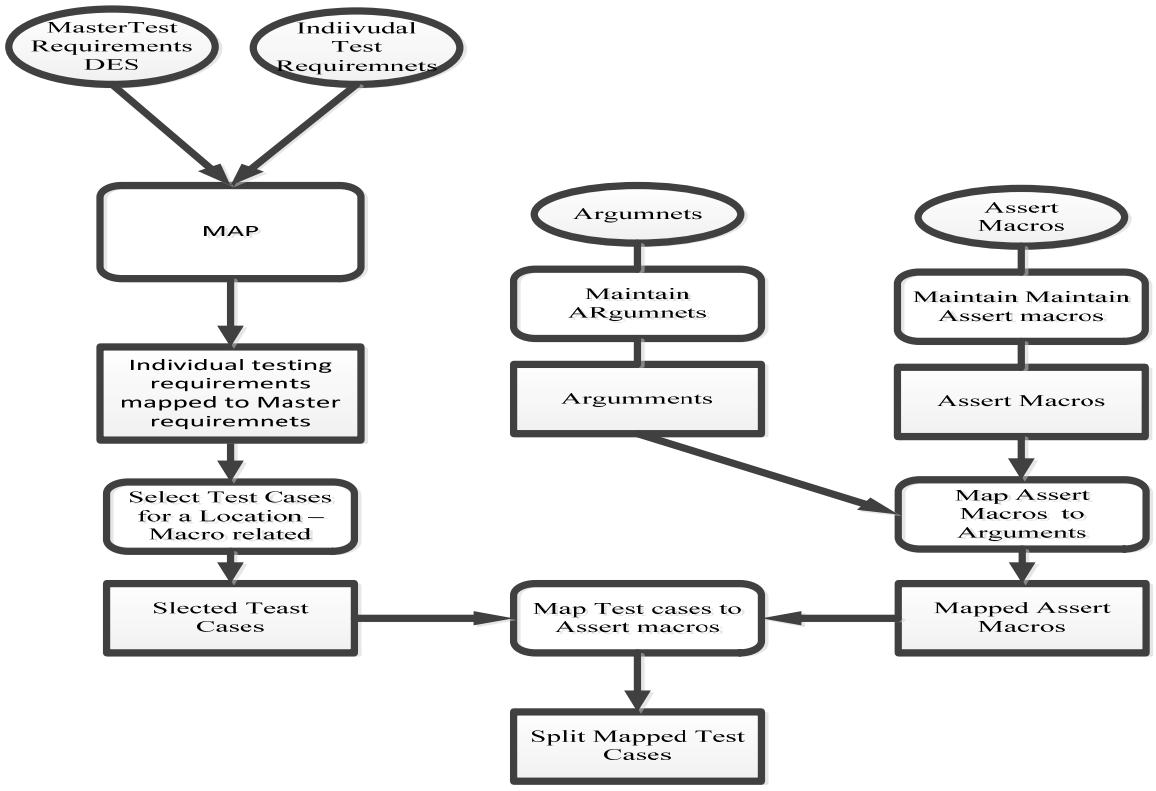

Figure-3. Environment setup for generation of test cases for a specific location.

Every embedded system that participates in a distributed embedded network is a standalone system by itself having required interfaces for connecting into a network. It is possible to find the locations where testing can be undertaken to realise the testing requirements. Master test cases can be pre-identified along with the methods that should be used for undertaking testing. The testing requirements are mapped to the master test cases which reveals the kind of testing, locations where testing should be conducted and the methods to be used for undertaking testing.

A separate process is used for splitting the test cases which must be tested through Assert macro and at different locations. An assert macro is attached with a test case. The parameters that must be tested are mapped to an assert Macro and the parameters associated with the assert macro are evaluated to TRUE or FALSE. As such Assert Macro is designed to handle only one parameter. However in this presentation assert macros with multiple parameters are evaluated to find combined truth values considering all the parameters put together.

Each of the test case is associated with an assert Macro and a set of parameter to the assert macro. The basic idea behind splitting the master test cases into many individual test cases is to undertake testing at different locations, and merge the test results obtained at each location to generate the test results for entire distributed embedded systems. Table-3 shows some of the entries in the repository that are related to setting the environment required for undertaking testing of distributed embedded systems through assert macros. Test Macros associated with a parameter list are associated with the test cases through test case serial numbers.

4.2 Process flow for undertaking testing through asserts macros

Figure-4 shows the process involved in undertaking testing at a particular Location. Test macros are generated using the details stored in Table-2. A separate process merges into the source code by way of recognising the variables that are processed by the instructions contained in the code. The updated source code is then compiled and executed to obtain the test results.

Table-3. Environment testing for undertaking testing of a distributed embedded system.

Assert macro	Test case handled by assert macro	Test case number	Parameter list
MACRO-1	Test for proper sensing of Temp-1 signal at the output of the Temp-1 sensor	1	Temp-1
MACRO-2	Test for proper sensing of temp-1 and Pump-1 to be on	6	Temp-1, Pump-1-Status
MACRO-3	Test for proper sensing of Temp-1 and Pump-1 to be off	7	Temp-1, Pump-1-Status
MACRO-4	Test for sensing Temp-1, Temp-2 and Buzzer of condition	8	Temp-1, Temp-2, Buzzer-Status
MACRO-5	Test for sensing Temp-1, Temp-2 and Buzzer on condition	9	Temp-1, Temp-2, Buzzer-Status
MACRO-6	Test for proper sensing of Temp-2 signal at the output of the Temp-2 sensor	14	Temp-2
MACRO-7	Test for Temp-2 and proper asserting of Pump-2 signal to be on	16	Temp-2, Pump-2-Status
MACRO-8	Test for Temp-2 and proper non assertion of Pump-2 signal	17	Temp-2, Pump-2-Status
MACRO-9	Test for Temp-2 and buzzer on	18	Temp-2, Buzzer-Status
MACRO-10	Test for Temp-2 and buzzer off	19	Temp-2, Buzzer-Status

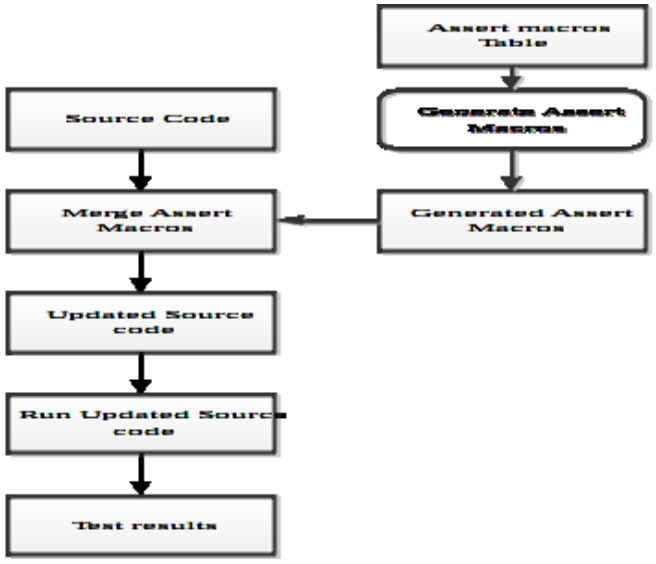

Figure-4. Process flow for undertaking testing using assert macros.

4.3 Undertaking testing at individual locations through assert macros

Testing using assert macros is undertaken at each of the Location using the process flow shown in the Figure 3. Only those test cases that are to be tested at an allocation are selected and the required macros are generated and the same are merged into the source code based on the parameters that are associated with the macros. The source code is visited line by line and on tracing the variables matching the parameters related to assert macros, the assert macros are inserted into the code. The updated code is compiled and executed to produce the test results. The test cases related to different locations and the test results produced due to testing at different locations is shown in Table-4.

Table-4. Test results obtained through assert macros.

Master Test case serial	Test case	Location for Testing	Assert macro	Parameter-1	Parameter-2	Parameter - 3	State of testing	Action
1	Test for proper sensing of Temp-1 signal at the output of the Temp-1 sensor	Location-1	Macro-1	Temp-1			TRUE	Continue execution
6	Test for proper sensing of temp-1 and Pump-1 to be on	Location-5	Macro-2	Temp-1	PUMP-1		False	Suspend program with a LCD display
7	Test for proper sensing of Temp-1 and Pump-1 to be off	Location-1	Macro-3	Temp-1	PUMP-1		TRUE	Continue execution
8	Test for sensing Temp-1, Temp-2 and Buzzer of condition	Location-5	Macro-4	Temp-1	Buzzer		TRUE	Continue execution
9	Test for sensing Temp-1, Temp-2 and Buzzer on condition	Location-5	Macro-5	Temp-1	Temp-2	Buzzer	TRUE	Continue execution
14	Test for proper sensing of Temp-2 signal at the output of the Temp-2 sensor	Location-2	Macro-6	Temp-2			TRUE	Continue execution
16	Test for Temp-2 and proper asserting of Pump-2 signal to be on	Location-4	Macro-7	Temp-2	Pump-2		TRUE	Continue execution
17	Test for Temp-2 and proper non assertion of Pump-2 signal	Location-4	Macro-8	Temp-2	Pump-2		TRUE	Continue execution
18	Test for Temp-2 and buzzer on	Location-5	Macro-9	Temp-2	Buzzer		TRUE	Continue execution
19	Test for Temp-2 and buzzer off	Location-5	Macro-10	Temp-2	Buzzer		TRUE	Continue execution

4.4 Merging test results and producing an audit trail

The test results obtained through assert macros are merged based on the test case serial which are originally identified as a set of testing requirements. The process of merging the test results which are obtained by conducting testing using assert macro methods at distributed locations are shown in the Figure-5.

5. CONCLUSIONS

The standard methods such as scaffolding, Assert Macros etc., that are in existence as on today using which the testing of standalone embedded systems is being carried must be modified and new methods must be invented for undertaking the testing of distributed embedded systems. The investigation presented in this paper related to breaking single test cases that must be tested across entire distributed embedded system into many test cases that each of the test cases generated be tested at a specific individual embedded system. The test results obtained at each location is integrated through mapping and merging so as to get the overall status of testing the entire distributed embedded system. The process of undertaking testing through Assert Macros is presented in this paper. Assert Macros help testing the existence of the required environment before moving with the execution with further real-time processing. The absence of the environment may lead to suspending the program or to handling of an exceptional condition. The overall Test Results after undertaking merging are shown in the Table-5.

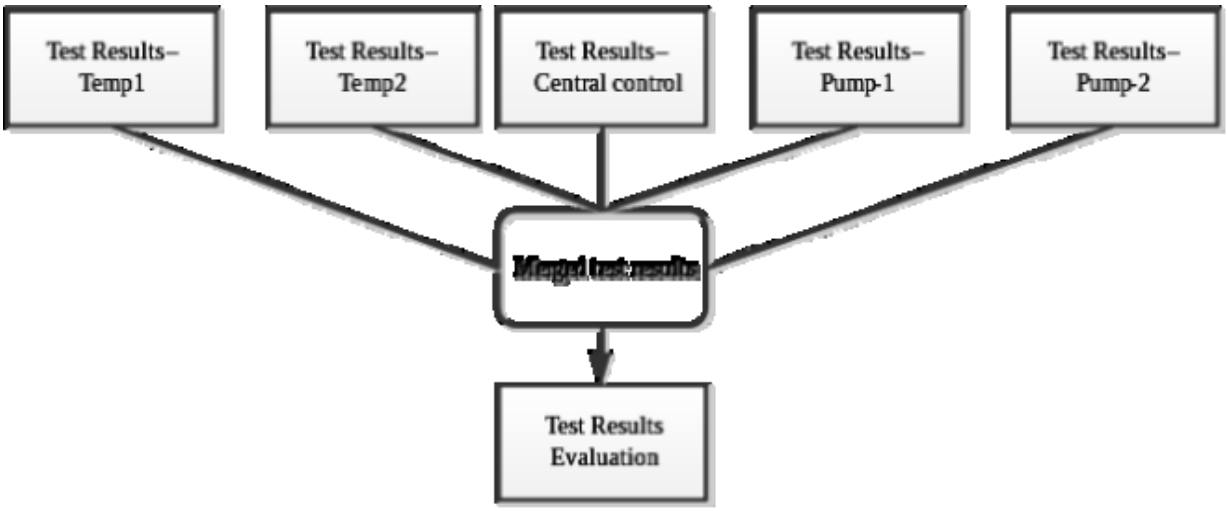

Figure-5. Merging the test results obtained from scaffolding at each of the location.

Table-5. Test results obtained through assert macros.

Merged test cases	Test case	State of testing	Action
1	Test for proper sensing of Temp-1 signal at the output of the Temp-1 sensor	TRUE	Continue execution
6, 7	Test for proper sensing of temp-1 and Pump-1 to be on and off	FALSE	Suspend program with a LCD display
8, 9	Test for sensing Temp-1, Temp-2 and Buzzer on and of condition	TRUE	Continue execution
14	Test for proper sensing of Temp-2 signal at the output of the Temp-2 sensor	TRUE	Continue execution
16, 17	Test for Temp-2 and proper asserting of Pump-2 signal to be on and off	TRUE	Continue execution
18, 19	Test for Temp-2 and buzzer on and off	TRUE	Continue execution

REFERENCES

[1] J. Sasi Bhanu, A.VinayA Babu, p. Trimurthy. 2015. Implementing dynamically Evolved online testing of Embedded Systems. Indian Journal of Science and Technology.

[2] Doron Drusinky, Man-Tak Shing and Kadir alpaslan Demir. 2006. Creation and validation of embedded assertion state charts. IEEE international workshop on rapid system prototyping.

[3] Hany F.EL Yamany, MIRIAM A.M. CAPRETZ and LuizF. Capretz. 2006. A Multi - Agent Frame work for testing distributed systems. International conference on computer software and applications.

[4] Hui Liu, Maozhong J in and Chao Liu. 2010. Construction of simulating environment for testing distributed embedded software. 5th International Conference on Computer Science and Education.

[5] Jeffrey Voas and Lora Kasab. 1999. Using Assertions to make untestable Software more testable, Software quality professional. 1(4): 31-34.

[6] Hwei Yint James RI. Bieman. 1994. Improving software testability with assertion insertion. International test conference.

[7] David S. Rosenblum. 1992. Towards a Method of Programming with Assertions. International conference on Software Engineering.

[8] Stephen H. Edwards, Murali Sitaraman. 2004. Contract-Checking Wrappers for C++ Classes. IEEE Transactions on Software Engineering. 30(11).

[9] J. VOAS. 1997. Building Software Recovery Assertions from a Fault Injection-based Propagation Analysis. In Twenty First Annual Conferences on Computer Software and Applications Conference. COMPSAC '97.

[10] 2004. Daniel Hughes A Framework for Testing Distributed System in 1992. 4th International conference on Peer-to-Peer Computing.