Journal of Engineering and Applied Sciences 13 (8): 1947-1962, 2018
ISSN: 1816-949X
© Medwell Journals, 2018

Testing through In-Circuit Emulators, the RS485 based Distributed Embedded System

J. Sasi Bhanu, Y. Venkata Raghavarao and JKR Sastry
St. Martin Engineering College, KLEF University, Vaddeswaram, Hyderabad, India

Abstract: Many distributed embedded systems are being used for implementing different kinds of applications. The trend in embedded systems design in recent years has been towards highly distributed architectures with support for concurrency, data and control flow and scalable distributed computations. Many methods have been proposed in the past for testing a standalone embedded system and not many methods have been proposed for testing distributed embedded systems. Methods such as scaffolding, assert macros, in-circuit emulators, monitors, logic analyzers are used in addition to third party tools for undertaking the testing of standalone systems. However, using of the same for testing a distributed embedded system is complicated. In the case of distributed embedded systems, a test case must be tested considering the related processes which are distributed across severa lembedded systems that are connected within the same network. Some testing has also to be carried for proper messaging/communication taking place between the embedded systems that get connected to the network. Messaging as such is dependent on the type of networking such as I2C, CAN and RS485 is used for establishing the distributed embedded system. It is not possible to undertake testing when any part of the network fails during testing. In this study, a method has been proposed for testing distributed embedded system that gets connected through RS485 based communication system through use of in-circuit-emulators that are capable of isolating the processes related to the test cases that must be tested to ensure that the distributed embedded system has been functioning as per the design.

Key words: In-circuit emulators, distributed embedded systems, RS485 based networking, functioning, processes, embedde

INTRODUCTION

Embedded systems are a different class of systems which throw several challenges, especially, related to testing. The testing process to test the embedded applications involves testing individually hardware, software and both the hardware and software together. The process of testing an embedded application is rather complex. Development and testing of embedded software is especially, difficult because it typically consists of a large number of concurrently executing and interacting tasks. Each task in embedded software is executed at different intervals under different conditions and with different timing requirements. Furthermore, time available to develop and test embedded software is usually quite limited due to relatively short lifetime of the products.

Cost effective testing of embedded software is of critical concern in maintaining competitive edge. Testing an embedded system manually is quite time taking and also will be a costly preposition. Tool based testing of an embedded system has to be considered and put into use to reduce the cost of testing and also complete the testing of the system rather quickly. Testing and debugging embedded systems is difficult and time consuming for simple reason that the embedded systems have neither storage nor adequate user interface. The users are extremely intolerable of buggy embedded systems. Embedded systems deal with external environment by way of sensing the physical parameters and also must provide outputs that control the external environment.

In embedded systems, the issue of testing must consider both hardware and software. The mall-functioning of hardware is detected through software failures. The target embedded system does not support the required hardware and software platform needed for development and testing the software, hardware and both. The software development cannot be done on the target machine. The software is developed on host machine and then installed in the target machine which is then executed.

Comprehensive testing of embedded systems requires the identification of various test methods, the location of carrying testing and the kind of testing that can be conducted using specific method. Comprehensive

Corresponding Author: J. Sasi Bhanu, St. Martin Engineering College, KLEF University, Vaddeswaram, Hyderabad, India

testing includes testing hardware, software and both. The comprehensive testing includes various types of testing to be carried at different locations using different methods.

The entire embedded system application code can be divided primarily into components namely hardware independent code and hardware dependent code. Hardware independent codes are tasks that carry mundane housekeeping and data processing whereas the hardware dependent codes are either interrupt service routines or the drivers that control the operation of the device. It is necessary to identify different types of test cases that tests both hardware independent code and hardware dependent code. The testing techniques and the testing locations where testing must be carried should address the requirements of testing both the hardware independent and hardware dependent code.

Embedded systems software development is done on a host as the target machine will not have sufficient resources to undertake the software development. ES software development process is undertaken on a host and on completing the development and undertaking the testing to certain extent, the code is then copied on to the ROM or flash memory of the target and then the hardware is again tested along with the host. The embedded system is connected to the production system and the testing is carried again. Testing the embedded system after connecting to the production system is absolutely necessary as any amount of testing carried simulating the events initiated by the production system will not clearly depict the working of production system. The real production system may initiate many unforeseen and uncommon events which are considered as part the development of the system.

If an analysis is carried on the entire code of an ES application it will reveal that 80% of code is hardware Independent code and the rest is hardware dependent code. It makes it easy, faster and cost effective to complete testing of the hardware independent code on the host machine itself. The testing of the hardware independent code can be carried at the host by carrying any of techniques that includes Scaffolding, Assert Macros and in-circuit emulators. Using scaffolding, testing of hardware independent code can be carried on the host machine.

In-circuit emulators will be handy to test the response time, throughput and portability issues. An emulator is software that runs on host and simulates the behavior of micro processor and the memory on the target machine. The emulator has the knowledge of locator output, architecture and in-circuit of the target micro processor. Emulators do the simulation by way of using the memory for registers, program counters and address registers and data buffers. The instructions are read from the memory and converted to instructions equivalent to the target machine. Emulators also support a macro language using which testing scenarios are submitted as input to the emulators. Emulator can report response time in terms of the number of target machine instructions executed. The count of instructions executed or number of bus cycles used. The average response time can be computed by multiplying with average in-circuit execution time. Emulators can also execute the startup code and interrupt service routines written in assembly language. Most of the testing related to portability issues can be tested using the emulators as the simulation of instructions of the target machine is done by the emulators. Emulators also help in testing the built-in peripherals such as timer, DMA, UART, etc., as the simulation of such built in emulators is quite possible. Emulators have the prior knowledge of the target processors and related Built-ins.

At the host, several types of testing can be conducted considering the quality of the embedded applications, especially, when bugs are to be investigated whenever the errors are traced while running the application on the host. The test cases are submitted to a third party tool and the third party tool conducts the testing using the image of embedded application and produces the test results back to test process which maintains the test results in the secondary stage using which the audit trail can be conducted.

A Microcontroller which is part of an embedded system is replaced by an in-circuit emulator in the target machine. The emulation software is stored in a separate memory different from application memory and the emulation software maintains execution status of the application in its memory. The execution of the application is controlled by the emulation software. The emulation software also will have communication software component to communicate with the host. Host can initiate series of test cases to in circuit emulator and the test execution results are sent back to the host. The emulator software provides the support to test the software along with the hardware: emulator software supports debugging of the application through commands sent from host. Figure 1 shows the system architecture for undertaking testing considering the target (Embedded system) and the host (PC)

The emulator software provides the support to test the software along with the hardware: emulator software supports debugging of the application through commands sent from the host. In the case when an embedded system fails, the de-bugging of the failure can

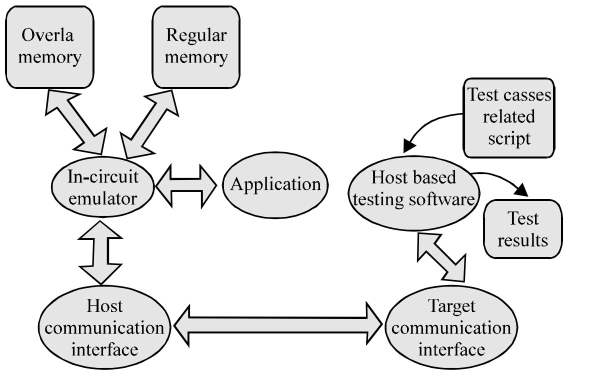
Overla memory
Regular memory
Test casses related script
In-circuit emulator
Application
Host based testing software
Test results
Host communication interface
Target communication interface

Fig. 1: Block diagram connecting the individual embedded systems

be undertaken by using the functions supported by in-circuit emulator which include setting break points, examine the contents of memory and registers, setup through sources code, exercise control on execution of the code through stopping the execution of the code and then resuming the execution as required.

In-circuit emulator also allows seeing the contents of the memory and registers even when the application is crashed. Emulators captures trace of program execution and in the event of any failure, emulators still help in interacting with it through host machine. The entire dump of overlay memory can be viewed and the reason for failure can be investigated. The emulator software can be read from a host and copied into overlay memory.

Different types of testing can be done using the in-circuit emulator which includes testing the inbuilt peripheral devices, memory leakages, response time, functional usage, un-used code, changes to the data at specified locations, high used functions, inter-task communication, etc.

The process flow for undertaking testing at each of the location using in-circuit emulator is shown in Fig. 2. The test cases that should be tested using the in-circuit emulator are generated and stored in a database at the host. The micro controller in the target is replaced by an in-circuit emulator which emulates the execution of the code as if the code is executed by the originally installed micro controller. The in circuit emulator has a separate overlay memory at which trace of the application execution and the emulation software are stored which can be used for debugging even if the rest of the hardware is broken down.

The test cases meant for testing through in-circuit emulator are read one after the other through a separate process and a script file containing the commands and the

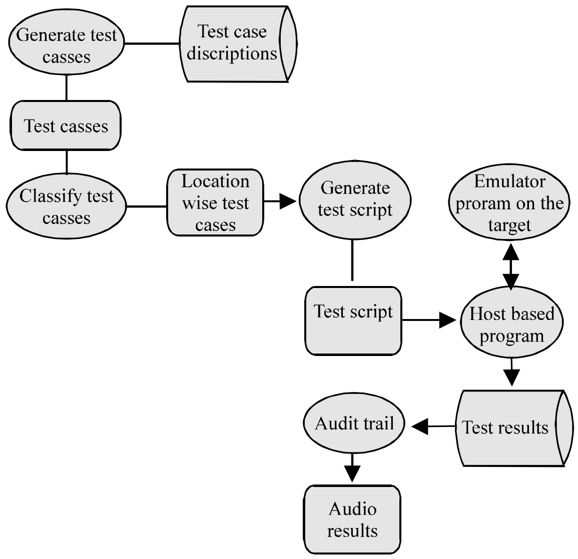

Fig. 2: Networking the embedded systems through RS485 networking

command line arguments which are understandable by the processes resident at the target and the in-circuit emulator is created and stored in the database.

The test process initiates the execution of the commands contained in the script file and communicates the same to a process resident at the host. The process at the target execute the commands received from the host in emulation mode and the results obtained due to execution of the test cases through command execution are sent through a communication interface implemented at both ends of the target and the host. The test results received at the host are stored in a audit trail database file. The target machine can be made to research in emulation or non-emulation mode by setting the mode switch provided on the target board or the mode can also be set by way of asserting the ALE signals of the controller.

The communication between the emulator and the host based program can be achieved through RS232C interface. Emulators are defined with specified address range at the time of relocating the code. All the memory variables, register and the CPU status and control registers are mapped to the internal memory of the emulator. Emulator can communicate with the host even in the case of failure of target.

A distributed system involves several embedded systems which are interconnected trough a networking system such CAN, I^2C, RS 485, USB, etc., using BUS like topology using serial communication system. An embedded application is broken down into several small applications and each of the small application is implemented through an individual embedded system which means both the hardware and software is distributed. The communication between the embedded systems is generally monitored and controlled through central embedded systems or through effecting peer to peer communication.

A distributed embedded system can be considered as a set of functions (or tasks depending on the vocabulary) interacting to realize system functionalities. A common representation uses the notion of communication channels in order to model asynchronous communications. But, if such a model can be useful because of its simplicity, it does not reject the way of communicating when considering actual embedded systems. Communication networks are composed of shared resources (switches, gateways, routers, etc.) accessed by every system participants.

A distributed embedded system involves use of individual microcontroller based systems which are generally heterogeneous in nature. Each microcontroller based system will have built-in interfaces using which communication with other microcontrollers can be achieved. Establishing communication among various microcontrollers is essential to implement a distributed embedded application.

In a distributed embedded application both the hardware and software that comprise entire application is distributed. Communication is necessary among the individual microcontroller based systems for exchanging process information. Networking microcontroller based systems becomes one of the most important criteria in implementing distributed embedded systems. Heterogeneity is the most critical issue that must be considered while designing a network of distributed embedded systems. The heterogeneity is due to interfaces, protocols, implementation of protocols, etc.

Literature review: Several researcher have proposed different approaches to conducting testing of embedded systems. Jacobson *et al*. (1999) have suggested testing of modules of embedded systems by isolating the modules at run time and improving the integration of testing. This method has however failed to support the regression of events. They have suggested testing of embedded systems by simulating the hardware on the host and combining the software with the emulators. This approach however will not be able to deal with all kinds of test scenarios related to hardware. The complete behavior of hardware specially unforeseen behavior cannot be simulated on a host machine.

Nancy Van Schooenderwoert have suggested an approach of carrying unit testing of the embedded systems using agile methods and using multiple strategies. Testing of embedded software is bound up with test of hardware, crossing professional and organizational boundaries. Even with evolving hardware in the picture, agile methods work well provided multiple test strategies are used. This has powerful implications for improving the quality of high reliability systems which commonly have embedded software at their heart.

Regression testing (Tsai *et al*., 2001) has been a popular quality testing technique. Most regression testing's are based on code or software design, Tsai and others have suggested regression testing based on Test scenarios and the testing approach suggested is functional regression testing. Tsai and others have even suggested a WEB based tool to undertake the regression testing.

Tsai *et al*. (2003) have suggested END-TO-END Integration testing of embedded system by specifying test scenarios as thin threads; each thread representing a single function. They have even developed a WEB based tool for carrying end-to-end integration testing. They have suggested a testing approach based on verification patterns, the key concept of this being recognizing the scenarios into patterns and applying the testing approach whenever similar patterns are recognized in any embedded application. But the key to this approach is the ability to identify all test scenarios that occur across all types of embedded applications.

Lee suggested a different approach for conducting integration testing by considering interaction scenarios, since, the integration testing must consider sequence of external input events and internal interactions

In-circuit emulator is an important support tool in development and integration of hardware and software related to microprocessor based systems. Generally, an in-circuit emulator is implemented as a module of a development station. For this reason, it depends completely on the architecture and the bus structure of the host system and it cannot be moved from system to system, since, it cannot be used on its own. Therefore, it represents an expensive investment as it must be acquired with the development system itself (Ergincan and Satic, 1986). Fahir Ergincan and Ali Saa presented stand-alone, in-expensive in-circuit emulator which is suitable for a particular microprocessor.

Armengaud *et al*. (2005) presented embedded electronic communication systems which connect all the embedded systems that are fitted into an automobile system. For successful application of development new test and diagnosis solutions for these distributed systems are required. They have presented a stimulus generation of test systems based on a remote test under the stringent constraints of the automotive industry. They have elaborated a flexible and accurate method that enables a systematic and comprehensive test of data link layer related communication services. Furthermore, they have discussed how the solution can be applied for various different test purposes (which include verification, testing for robustness, interoperability and also to carry maintenance tests and demonstrated its application for testing a data link layer related protocol called "Flex Ray".

In-Circuit Test (ICT) has been used for more than 30 years to test for correct assembly of components on to a Printed Circuit Board (PCB). The premise behind the in-circuit test philosophy was based on gaining net level access to a circuit and driving and sensing signals through the components of that circuit to determine if the components were placed correctly and soldered correctly to the board. Given today's board density and speed requirements, it is becoming more and more challenging to gain access to all of the nets on a given Printed Circuit Assemble (PCA). Even with 100% access, the tester may not be capable of testing all of the nets on a large, high density PCA (due to the sheer number of nets on the board). In addition as more and more logic is integrated into devices and signal integrity begins to dominate interconnect as the primary board level concern; the "relevance" of in circuit test seems to be diminishing. An advanced in-circuit test techniques that help improve the effectiveness of the in-circuit test on large, fast and complex PCAs with limited test point access has been presented by John Malian.

Effective validation Quality-of-Service (QoS) properties (e.g., event prioritization, latency and throughput) of a Distributed Real-time Embedded system (DRE) require testing system capabilities in representative execution environments. Unfortunately, evaluating the correctness of such tests is hard, since, it requires validating many states dispersed across many hardware and software components. To address this problem (Hill *et al*., 2011) have presented a method called Test Execution (TE) Score for validating execution correctness of DRE system tests and empirically evaluated TE score in the context of a representative DRE system. Results from this evaluation show that TE Score can determine the percentage correctness in test execution and facilitate trade-off analysis of execution states, thereby, increasing confidence in QoS assurance and improving test quality.

Test case creation activities consume an increasing amount of resources allocated to software development projects. There is a dire need for automating the testing task as much as possible. In this study, we report on the application of academic test case generation tools in an industrial context. Chimisliu and Wotawa (2012) have presented an approach to generate test cases from reactive distributed systems specified as asynchronously communicating UML state charts. They have employed two approaches for the generation process. The first one is fully automated and generates test cases aimed at transition coverage. The second one requires the intervention of the tester in order to annotate states and/or transitions partially describing a test scenario. It is the job of the tool to compute test cases pertaining to the specified test scenario.

Quality is the life of embedded software and the software testing is a basic guarantee for stable and reliable operation of the embedded system. Looking at the difference in target and development environment for embedded software, to increase the consistency of embedded software reliability testing, it becomes much more necessary to construct embedded software testing environment. After analysing the Basic Structure, functionalities and characteristics of distributed embedded software test environment, instead of adopting the conventional developing pattern, a three-layer development pattern is put forward that can fulfil distributed special purpose testing system for specific embedded software testing application (Tian *et al*., 2009)

Liu *et al*. (2010) presented a testing tool for distributed embedded software. They have used Distributed Embedded System Simulating Environment (DESSE) could simulate a highly configurable hardware environment for software testing. The DESSE method 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. Using the tool, some difficult distributed embedded software testing such as "probe effect", non-repeatability, can be undertaken even when certain resources like synchronized global clock is non-existent. This study examines the fundamental problems that one faces when testing a distributed, hard real-time (Schutz, 1994) examined fundamental problems that one face when testing a distributed, hard real-time system. It specifically identifies the influences of the distributed ness and of the real-time requirements of the systems considered. He has shown that the usual problems of testing become more difficult and also more additional problems are introduced when it comes testing of distributed embedded systems due to the addition of more system characteristics. He has identified six fundamental problems: organization, observability, reproducibility, host/target approach, environment simulation and representativetity. These as well as their interrelations are presented in a general framework that is independent of a particular system architecture or application. This framework could serve as a starting point for all activities geared towards particular system architecture or a specific application. They have presented how test problems have been handled when developing a test methodology for the distributed real-time system MARS.

In the recent years, embedded systems have become, so, complex that the development and testing time is becoming extremely time consuming. As embedded systems include more and more functions for new services, embedded systems are presenting challenges with respect to the attributes of security, scalability availability and performance with deterministic behaviour. Saini (2012) has presented various issues that affect testing process and technologies which can be ameliorated by Rational Test Real-Time (RT-RT). All the architectural specification is analysed through adapting object oriented approach while designing the embedded systems.

Embedded control systems such as automotive form large scale distributed ones which consist of numerous CPUs and devise including plants and various hardware components. The complexity of such systems has increased for sophisticated control using numerous types of sensor data. Meanwhile, market pressures lead to reducing development periods for the large scale distributed embedded control systems. A model based design becomes popular (Nakamoto *et al*., 2014) to solve the above problems, especially, automotive industries. Nakamoto *et al*. (2014) have described a wide-area distributed integrated test environment based on the model based design. The key features are communication middleware to enable wide-area distributed test with integrating the programs in widely distributed sites and control program execution with models of multiple abstraction levels in the model based design. Furthermore, they have developed a model agent simulation. Instead of a plant model that might be located and simulated by MATLAB/Simulink in a remote site, a model agent behaves like the model in a local computer. The model agent simulation significantly reduces the test execution time in a prototype evaluation.

Reproducible and deterministic testing of sequential programs can in most cases be achieved by controlling the sequence of inputs to the program (Thane and Hansson, 1999). The behaviour of a distributed real-time system, on the other hand, not only depends on the inputs but also on the order and timing of the concurrent tasks that execute and communicate with each other and the environment. Hence, sequential test techniques are not directly applicable, since, they disregard the significance of order and timing of the tasks. Thane and Hansson (1999) have presented a method for identifying all possible orderings of task starts, pre-emptions and completions for tasks executing in a distributed real-time system. Together with an accompanying testing strategy, this method allows test methods for sequential programs to be applied, since, each identified ordering can be regarded as a sequential program. In the presented analysis and testing strategy, they have considered task sets with recurring release patterns and take into account the effects of clock synchronization and variations in start and execution times of the involved tasks.

Distributed embedded system for experimentation: 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. Coolants have to be injected into the tubes to bring the temperature down when temperatures raises above some pre-defined levels,. 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.

The block diagram showing the connectivity between the embedded systems which forms the distributed embedded system is shown in Fig. 3 and 4, the way the embedded systems have been networked using RS 485 based networking system has been shown.

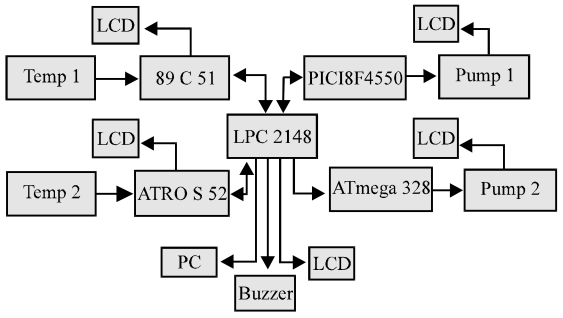

Fig. 3: Block diagram connecting the individual embedded systems

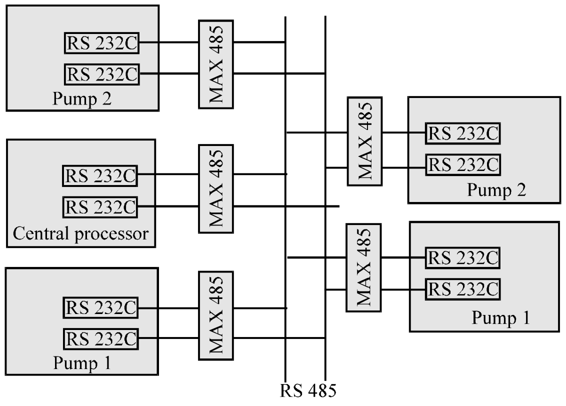

Fig. 4: Environment setup for generation of test cases to be tested at each of the locations

The application meant for monitoring the temperatures within nuclear reactors system is implemented through uses of 5 Micro controller based systems which include LPC2148 (Central Processing Unit), 89C51 (Temp-1 sensing), AT89S52 (Temp2 sensing), PIC18F4550 (Pump-1 Control) and ATmega228 (Pump 2 Control)

Designing, development and implementing the networking of embedded systems becomes one of the most crucial issues when it comes to designing, development and implementing the 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 centralized coordination between the distributed embedded systems.

The individual embedded systems have been networked using 485 protocols. Protocol conversions have been used where ever no native support for the protocol exists. Figure 4 shows the block diagram of a decentralized embedded system with built-in respective interfaces along with individual embedded systems that provides centralized coordination. Interfacing distributed embedded systems through RS485 protocol requires protocol conversion when no native support exists within the respective individual embedded systems.

RESULTS AND DISCUSSION

Mapping master test cases to various distributed locations: Some of the major testing requirements that must be tested across the entire distributed embedded system (Pilot project) are shown in Table 1. It has been noted in the table, the way the test case is realized considering the method used for undertaking the testing and the locations that are involved in undertaking the testing.

Environment setting for testing distributed embedded systems through in-circuit emulator method: Environment is to be set to start-with for undertaking testing through in-circuit emulator as the test cases are to be presented as test scripts which are to be read and executed by the in-circuit emulator. Figure 4 shows the environment setting required for undertaking testing through in-circuit emulator method. Every embedded system that participates in a distributed embedded network is a standalone system by itself having required interfaces for connecting it into a network. The test cases that can be used for undertaking the testing of a standalone embedded system can be pre-identified and mapped to Master test cases which must be tested considering the entire distributed embedded system. This process allows the splitting of master test cases into test cases that must be tested at a specific location. Each of the split test cases can be associated with a command and the commands are associated with certain arguments. Some of the commands may involve more than one location, especially, when the testing of communication between the distributed elements needs to be undertaken. The test cases are then mapped to the commands through which testing is undertaken. A split process is used to separate the test cases that are to be used for testing at a particular location.

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. The locations, the commands used for testing, the arguments that are related to each of the commands are maintained individually by

Table 1: Master test cases for testing the entire distributed embedded system

Master test case	Testing methods	Testing location
Read temp-1 and write to LCD	Scaffolding	Location-1
Test the RS485 based communication between the 89C51 (System-1) and the central micro controller system-5)	Scaffolding	location-5
Read-temp-1 and send to central micro controller	Scaffolding	Location-1
Read temp-1 and measure throughput	Scaffolding, simulator	Location-1
Test the RS485 based communication between the 89C51 (System-3) and the central micro controller system-5)	Scaffolding	location-5
Read temp-1 and make pump-1 on if temp-1>reference temp-1	Scaffolding	Location-1, 3, 5
Read temp-1 and make pump-1 off if temp-1<reference temp-1	Scaffolding	Location-1, 3, 5
Read temp-1 and make buzzer on if >temp-2	Scaffolding	Location-1, 3, 5
Read temp-1 and make buzzer off if <temp-2	Scaffolding	Location-1, 3, 5
Test response between the reading the temp-1 and starting the pump-1	Logic analyzer, scaffolding, in-circuit emulator	Location-1, 3, 5
Test response between the reading the temp-1 and stopping the pump-1	Logic analyzer, scaffolding, in-circuit emulator	Location-1, 3, 5
Test response between the reading the temp-1 and starting the buzzer	Logic analyzer, scaffolding, in-circuit emulator	Location-1, 5, 5
Test response between the reading the temp-1 and stopping the buzzer	Logic analyzer, scaffolding, in-circuit emulator	Location-1, 5, 5
Test for RS485 based communication between 89C51 between system-2 and the central processor System-5	Scaffolding	Location-2
Read temp2 and write to LCD	Scaffolding	Location-2
Read temp2 and write central embedded systems	Scaffolding	Location-2
Read temp-2 and measure throughput	Scaffolding	Location-2
Test the RS485 based communication between the 89C51 (System-4) and the central micro controller system-5)	Scaffolding	Location-2
Read temp-2 and make pump-2 on if temp-2>Reference temp-2	Scaffolding	Location-2, 5, 4
Read temp-2 and make pump-2 off if temp-2<Reference temp-2	Scaffolding	Location-2, 4, 5
Read temp-2 and make buzzer on if >temp-1	Scaffolding	Location-2
Read temp-2 and make buzzer off if <temp-1	Scaffolding	Location-2, 5, 5
Test response between the reading the temp-1 and starting the pump-1	Logic analyzer, scaffolding, in-circuit emulator	Location-2, 4, 5
Test response between the reading the temp-2 and stopping the pump-2	Logic analyzer, scaffolding, in-circuit emulator	Location-2, 4, 5
Test response between the reading the temp-2 and starting the buzzer	Logic analyzer, scaffolding, in-circuit emulator	Location-2, 5, 5
Test response between the reading the temp-1 and stopping the buzzer	Logic analyzer, scaffolding, in-circuit emulator	Location-2, 5, 5

Table 2: Repository of locations at developed for distributed embedded systems

Location codes	Location description	Location address
L1	Microcontroller based location for sensing and monitoring temperature-1	20
L2	Microcontroller based location for sensing and monitoring temperature-2	30
L3	Microcontroller based location for sensing and monitoring pump-1	40
L4	Microcontroller based location for sensing and monitoring pump-2	50
L5	Microcontroller based location for sensing and monitoring central monitoring system	60

different processes and the list of mapped commands with arguments that should be used at each of the location for carrying any type of testing is generated.

The individual embedded systems that are connected into a network must be provided with an address, so as to recognize the locations uniquely. Table 2 shows the addresses assigned to the individual computing locations.

Test cases are to be presented as commands to in-circuit emulator and the arguments that are associated with the commands must be assigned with range values that can be fed. Table 3 shows the list of arguments that can be mapped to test commands. Arguments are to be mapped to commands. Table 4 shows the mapping of the arguments to the commands.

A repository of test cases that can be used for testing an embedded system comprehensively can be identified based on test history and the same can be maintained as a repository. The method that should be used for undertaking the testing can also be recognized and stored along with the test case. It is also possible to map the application level test cases to the comprehensive

Table 3: Maintain arguments

Name of the argument	Type of the argument	Start values
Location-1	Char(2)	L1
Location-2	Char(2)	L2
Location-3	Char(2)	L3
Location-4	Char(2)	L4
Location-5	Char(2)	L5
Response	Int	10
Throughput	Int	10

Table 4: Mapping arguments to commands

Command	Argument-1	Argument-2
RESTEMP1	10	Location-1
RESTEMP2	10	Location-2
THRUTEMP1	10	Location-1
THRUTEMP2	10	Location-2
RESPUMP1ON	10	Location-3
RESPUMP1OFF	10	Location-3
RSEPUMP2ON	10	Location-4
RSEPUMP2OFF	10	Location-4
RESBUZZERON	10	Location-5
RESBUZZEROFF	10	Location-5

test cases. The mapping of comprehensive test cases with application specific test cases is shown in Table 5.

This simple mapping of application level test cases to comprehensive test case and master test cases to commands provide total environment required for undertaking the testing at individual locations. Each of the test cases identified at application level is split in such a way that test cases meant for testing a distributed system is tested through several test cases and testing

Table 5: Mapping standard test cases to master test cases and testing methods

Standard test case serial-1	Comprehensive test case	Testing methods	Master test case
1	Read temp-1 and write to LCD	Scaffolding	1
2	Test the RS485 based communication between the 89C51 (System-1) and the central micro controller system-5)	Scaffolding	2
3	Read-temp-1 and send to central micro controller	Scaffolding	3
4	Read temp-1	Scaffolding	4
5	Measurer throughput	Instruction set simulator, in-circuit emulator	4
6	Test the RS485 based communication between the 89C51 (System-3) and the central micro controller system-5)	Scaffolding	5
7	Read temp-1	Scaffolding	6
8	Test temp-1>reference temp-1	Scaffolding	6
9	Test for pump-1 on	Scaffolding	6
10	Read temp-1	Scaffolding	7
11	Test temp-1<reference temp-1	Scaffolding	7
12	Test for pump-1 off	Scaffolding	7
13	Read temp-1	Scaffolding	8
14	Test temp-1>temp-2	Scaffolding	8
15	Test for buzzer on	Scaffolding	8
16	Read temp-1	Scaffolding	9
17	Test temp-1 !>temp-2	Scaffolding	9
18	Test for buzzer off	Scaffolding	9
19	Test for response time of temp-1	Instruction set simulator, in-circuit emulator	10
20	Test for temp-1>reference temp-1	Scaffolding	10
21	Test for response time of the pump-1 on	Instruction set simulator, in-circuit emulator	10
22	Test for response time of temp-1	Instruction set simulator, in-circuit emulator	11
23	Test for temp-1>reference temp-1	Instruction set simulator, in-circuit emulator	11
24	Test for response time of the pump-1 off	Instruction set simulator, in-circuit emulator	11
25	Test for response time of temp-1	Instruction set simulator, in-circuit emulator	12
26	Test for temp-1>reference temp-1	Scaffolding	12
27	Test for response time of the buzzer on	Instruction set simulator, in-circuit emulator	12
28	Test for response time of temp-1	Instruction set simulator, in-circuit emulator	13
29	Test for temp-1>reference temp-1	Instruction set simulator, in-circuit emulator	13
30	Test for response time of the buzzer off	Instruction set simulator, in-circuit emulator	13
31	Test the RS485 based communication between the 89C51 (System-2 and the central micro controller system-5)	Scaffolding	14
32	Read temp-2 and write to LCD	Scaffolding	15
33	Read-temp-2 and send to central micro controller	Scaffolding	16
34	Read temp-2	Scaffolding	17
35	Measurer throughput	Instruction set simulator, in-circuit emulator	17
36	Test the RS485 based communication between the 89C51 (System-4) and the central micro controller system-5)	Scaffolding	18
37	Read temp-2	Scaffolding	19
38	Test temp-2>reference temp-2	Scaffolding	19
39	Test for pump-2 on	Scaffolding	19
40	Read temp-2	Scaffolding	20
41	Test temp-2<reference temp-2	Scaffolding	20
42	Test for pump-2 off	Scaffolding	20
43	Read temp-2	Scaffolding	21
44	Test temp-2>temp-1	Scaffolding	21
45	Test for buzzer on	Scaffolding	21
46	Read temp-2	Scaffolding	22
47	Test temp-2>temp-1	Scaffolding	22
48	Test for buzzer on	Scaffolding	22
49	Test for response time of temp-2	Instruction set simulator, in circuit emulator	23
50	Test for temp-2>reference temp-2	Scaffolding	23
51	Test for response time of the pump-2 on	Instruction set simulator, in-circuit emulator	23
52	Test for response time of temp-2	Instruction set simulator, in-circuit emulator	24
53	Test for temp-2>reference temp-2	Scaffolding	24
54	Test for response time related to pump-2 off	Instruction set simulator, in-circuit emulator	24
55	Test for response time when temp-2 is sensed	Instruction set simulator, in-circuit emulator	25
56	Test for temp-2>reference temp-2	Scaffolding	25
57	Test for response time of the buzzer on	Instruction set simulator, in-circuit emulator	25
58	Test for response time of temp-2	Instruction set simulator, in-circuit emulator	26
59	Test for temp-2>reference temp-2	Scaffolding	26
60	Test for response time of the buzzer off	Instruction set simulator, in-circuit emulator	26

undertaken solely at different locations. The test cases that are split to enable undertaking the testing at individual locations are shown in Table 6. It has been considered that the when emulator based testing is

Table 6: Splitting master test cases into test cases that can be executed at different locations

Master test case	Test case	Split master case	Testing methods	Location of testing
1	Read temp-1 and write to LCD	1	Scaffolding	Location-1
2	Test the CAN based communication between the 89C51 (System-1) and the central micro controller system-5)	2	Scaffolding	Location-5
3	Read-temp-1 and send to central micro controller	3	Scaffolding	Location-1
4	Read temp-1	4A	Scaffolding	Location-1
	Measurer throughput	4B	Emulator	Location-1
5	Test the CAN based communication between the 89C51 (System-3) and the central micro controller system-5)	5	Scaffolding	Location-5
6	Read temp-1	6A	Scaffolding	Location-1
	Test temp-1>Reference temp-1	6B	Scaffolding	Location-5
	Test for pump-1 on	6C	Scaffolding	Location-3
7	Read temp-1	7A	Scaffolding	Location-1
	Test temp-1<Reference temp-1	7B	Scaffolding	Location-5
	Test for pump-1 off	7C	Scaffolding	Location-3
8	Read temp-1	8A	Scaffolding	Location-1
	Test temp-1>temp-2	8B	Scaffolding	Location-5
	Test for buzzer on	8C	Scaffolding	Location-5
9	Read temp-1	9A	Scaffolding	Location-1
	Test temp-1>temp-2	9B	Scaffolding	Location-5
	Test for buzzer off	9C	Scaffolding	Location-5
10	Test for response time of temp-1	10A	Logic analyzer	Location-1
	Test for temp-1>reference temp-1	10B	Scaffolding	Location-5
	Test for response time of the pump-1 on	10C	Emulator	Location-3
11	Test for response time of temp-1	11A	Logic analyzer	Location-1
	Test for temp-1>reference temp-1	11B	Scaffolding	Location-5
	Test for response time of the pump-1 off	11C	Emulator	Location-3
12	Test for response time of temp-1	12A	Logic analyzer	Location-1
	Test for temp-1>reference temp-1	12B	Scaffolding	Location-5
	Test for response time of the buzzer on	12C	Emulator	Location-5
13	Test for response time of temp-1	13A	Logic analyzer	Location-1
	Test for temp-1>reference temp-1	13B	Scaffolding	Location-5
	Test for response time of the buzzer off	13C	Emulator	Location-5
14	Test the RS485 based communication between the 89C51 (System-2 and the central micro controller system-5)	14	Scaffolding	Location-2
15	Read temp-2 and write to LCD	15	Scaffolding	Location-2
16	Read-temp-2 and send to central micro controller	16	Scaffolding	Location-2
17	Read temp-2	17A	Scaffolding	Location-2
	Measurer throughput	17B	Emulator	Location-2
18	Test the RS485 based communication between the 89C51 (System-4) and the central micro controller system-5)	18	Scaffolding	Location-2
19	Read temp-2	19A	Scaffolding	Location-2
	Test temp-2>reference temp-2	19B	Scaffolding	Location-5
	Test for pump-2 on	19C	Scaffolding	Location-4
20	Read temp-2	20A	Scaffolding	Location-2
	Test temp-2<reference temp-2	20B	Scaffolding	Location-5
	Test for pump-2 off	20C	Scaffolding	Location-4
21	Read temp-2	21A	Scaffolding	Location-2
	Test temp-2>temp-1	21B	Scaffolding	Location-5
	Test for buzzer on	21C	Scaffolding	Location-5
22	Read temp-2	22A	Scaffolding	Location-2
	Test temp-2>temp-1	22B	Scaffolding	Location-5
23	Test for response time of temp-2	23A	Logic analyzer	Location-2
	Test for temp-2>reference temp-2	23B	Scaffolding	Location-5
	Test for response time of the pump-2 on	23C	Emulator	Location-4
24	Test for response time of temp-2	24A	Logic analyzer	Location-2
	Test for temp-2>reference temp-2	24B	Scaffolding	Location-5
	Test for response time related to pump-2 off	24C	Emulator	Location-4
25	Test for response time when temp-2 is sensed	25A	Logic analyzer	Location-2
	Test for temp-2>reference temp-2	25B	Scaffolding	Location-5
	Test for response time of the buzzer on	25C	Emulator	Location-5
26	Test for response time of temp-2	26A	Logic analyzer	Location-2
	Test for temp-2>reference temp-2	26B	Scaffolding	Location-5
	Test for response time of the buzzer off	26C	Emulator	Location-5

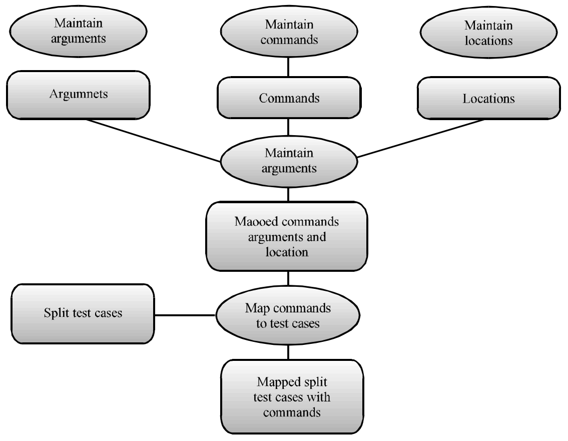

Fig. 5: Process flow for undertaking testing at an individual location

considered, simulator based testing is avoided as every test that can be done using a simulator can also be undertaken using in-circuit emulator.

Process flow for undertaking testing through in-circuit emulator: Entire embedded application code could be divided into a set of functions and the interlinking of the functions can be recognized as function code sequences. Execution of a test case involves execution of functions in a sequence. The process flow for undertaking testing using in-circuit emulator is shown in Fig. 6.

Test scripts are generated for undertaking testing at distributed locations for undertaking comprehensive testing of distributed embedded system. The test scripts are generated using the mapping of test cases with the commands which are mapped with different types of arguments. The generated test cases are written to an OS file called test script file.

The following algorithm is used for generating test scripts. Algorithm 1 shows the process involved in undertaking testing at a particular location.

Algorithm 1:

Step-1
Reading test cases from the split test cases that have been identified to be tested at a location using in-circuit Emulator

Step-2
Do the following process for each of test case to be executed at a location.

```
a.  Select a test case
b.  Select Script commands related to the test case
c.  Select the arguments related to the command
      i.  For each of the argument, select argument type and value range that can be used for selecting a value for the argument
      ii. Select value for the argument by using a process of randomization
d.  Generate the script
e.  Read the test script into the Emulator
```

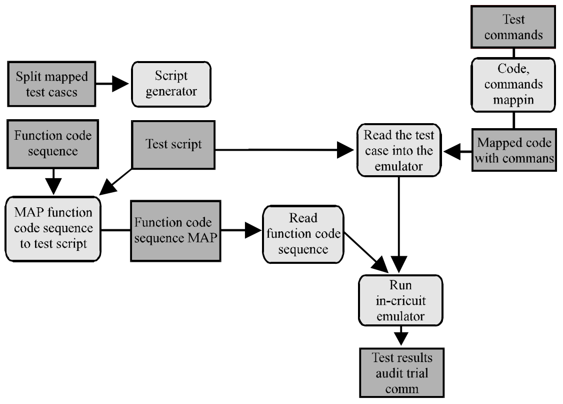
Test commands
Split mapped test cascs
Script generator
Code, commands mappin
Function code sequence
Test script
Read the test case into the emulator
Mapped code with commans
MAP function code sequence to test script
Function code sequence MAP
Read function code sequence
Run in-cricuit emulator
Test results audit trial comm

Fig. 6: Merging the test results obtained through use of instruction set simulator at each of the location

Step-3
The emulator on the host will read the script string and then transmit the same to the target

Step-4
The emulator program on the target shall receive the script string from the host and find the sequence of functions that must be executed

Step-5
Call the functions in the sequence required and at the end of completing execution of all the functions store the test results

Step-6
Transmit the test results to the host

Step-7
Receive the test results and write the results to audit trail

Algorithm for undertaking the testing at a computing location using an emulator: The test scripts generated using the algorithm shown in Fig. 7 are shown in Table 7. The test scripts are read into emulator by transmitting the same by the host side testing related program.

Testing at individual locations through in-circuit emulator: Testing at individual locations can be carried using the testing cases identified for undertaking testing at that location. The following description shows the way the testing is carried at location-1. Testing at other locations is undertaken in similar way.

The test cases that must be used for testing at a location-1 which are in relation to the master test cases are selected. For the test cases selected test scripts are generated as shown in Table 8.

Preparing testing environment for location-1: The ES application is a set of functions and the functions must be called in a sequence at any of the distributed locations for undertaking testing of a particular test case. For each of the testing to be carried, the sequence in which the functions must be called is mapped and stored in a repository as shown in Table 9.

Table 7: Generated test scripts

Split master test case	Test case	Location of testing	Test script
4B	Measurer throughput temp-1	Location-1	THRUTEMP1, THRUPUT = 10, Location = L1
10A	Test for response time of temp-1	Location-1	RESPTEMP1, RESPONSE = 10, Location = L1
10C	Test for response time of the pump-1 on	Location-3	RESPPUMP1, RESPONSE = 10, Location = L1
11A	Test for response time of temp-2	Location-1	RESPTEMP2, RESPONSE = 10, Location-2 = L2
11C	Test for response time of the pump-2 off	Location-3	RESPPUMP2, RESPONSE = 10, Location-2 = L2
12A	Test for response time of temp-1	Location-1	RESPTEMP1, RESPONSE = 10, Location-1 = L1
12C	Test for response time of the buzzer on	Location-5	RESPBUZZER, RESPONSE = 10, Location-1 = L5
13A	Test for response time of temp-1	Location-1	RESPTEMP1, RESPONSE = 10, Location-1 = L1
13C	Test for response time of the buzzer off	Location-5	RESPBUZZER, RESPONSE = 10, Location = L5
17B	Measurer throughput temp-2	Location-2	THRUTEMP2, THRUPUT = 10, Location = L2
23A	Test for response time of temp-2	Location-2	RESPTEMP2, RESPONSE = 10, Location = L2
23C	Test for response time of the pump-2 on	Location-4	RESPPUMP2, RESPONSE = 10, Location = L4
24A	Test for response time of temp-2	Location-2	RESPTEMP2, RESPONSE = 10, Location = L1
24C	Test for response time related to pump-2 off	Location-4	RESPPUMP2, RESPONSE = 10, Location = L4
25A	Test for response time of temp-2	Location-2	RESPTEMP2, RESPONSE = 10, Location = L2
25C	Test for response time of the buzzer on	Location-5	RESPBUZZER, RESPONSE = 10, Location = L5
26A	Test for response time of temp-2	Location-2	RESPTEMP2, RESPONSE = 10, Location-1 = L2
26C	Test for response time of the buzzer off	Location-5	RESPBUZZER, RESPONSE = 10, Location-1 = L5

Table 8: Test cases to be tested at location-1 to be tested through in-circuit emulator

Split master test case	Test case	Test script
4B	Measurer throughput temp-1	THRUTEMP1, THRUPUT = 10, Location = L1
10A	Test for response time of temp-1	RESPTEMP1, RESPONSE = 10, Location = L1
11A	Test for response time of temp-2	RESPTEMP2, RESPONSE = 10, Location-2 = L2
12A	Test for response time of temp-1	RESPTEMP1, RESPONSE = 10, Location-1 = L1
13A	Test for response time of temp-1	RESPTEMP1, RESPONSE = 10, Location-1 = L1

Table 9: List of functions that process the throughput and response time of temp1

Function code	Name of the function	Function description
F01	ReadTemp-1 ()	Reads the temp1 from sensor 1
F02	CountAndAcuumulate()	Counts the number of times temp-1 is read and accumulates the time taken to read the temp-1
F03	CompResponse()	Computes the response time

Table 10: mapping the scripts with generated functions

Test serial	Split master test case	Script	Function code sequence
1	4B	THRUTEMP1, RESPONSE = 10, Location = L1	ReadTemp1(), CountAndAcuumulate()
2	10A	RESPTEMP1, RESPONSE = 10, Location = L1	ReadTemp1(), CountAndAcuumulate(), CompResponse()
3	11A	RESPTEMP1, RESPONSE = 10, Location = L1	ReadTemp1(),CountAndAcuumulate(), CompResponse()
4	12A	RESPTEMP1, RESPONSE = 10, Location = L1	ReadTemp1(), CountAndAcuumulate(), CompResponse()
5	13A	RESPTEMP1, RESPONSE = 10, Location = L1	ReadTemp1(), CountAndAcuumulate(), CompResponse()

Table 11: Test results obtained through in-circuit emulator executed at location-1 processor

Split test case	Test case	Script command	Input variable-1	Input variable 1 value	Input variable-2	Input variable-2 value	Input variable-3	Input variable-3 value	Test output variable	Test output	Test fail/ pass
4B	Measurer throughput temp-1	THRUTEMP1	THRUPUT	10	-	-	-	-	TOUTTEMP1	10	Pass
10A	Test for response time of temp-1	RESPTEMP1	RESPONSE	10	-	-	-	-	TOUTTEMP1	10	Pass
11A	Test for response time of temp-2	RESPTEMP1	RESPONSE	10	-	-	-	-	TOUTTEMP1	10	Pass
12A	Test for	RESPTEMP1	RESPONSE	10	-	-	-	-	TOUTTEMP1	10	Pass
13A	Test for	RESPTEMP1	RESPONSE	10	-	-	-	-	TOUTTEMP1	10	Pass

The functions to be called and the sequence in which the functions must be called for facilitating the testing when test script is to be executed using in-circuit emulator have been mapped to each of the test case and the mapping is fed to the emulator for running the code as required for testing a test case. The mapping of the test script to function code is shown in Table 10 and 11.

Undertaking the testing through in-circuit emulator at location-1: The emulator program on the host reads a test script from the script file and sends the same to the target. The emulator program on the target parses the command and its related arguments. Using the command the list of functions and the sequence of the functions to be executed are fetched. Pointers to the functions are added into a queue. Another process contained within the emulator reads the queue and executes the functions. The functions are called by passing arguments which are associated with the commands. The execution of function leads to completing the testing and generating test results. The test results obtained after testing using the above process are sent to the host where the results are written to audit trail are shown in Table 12 and 13.

Testing at location-2

Identifying the test cases for location-2: The test cases that must be used for testing at a location-2 through split test cases are selected. Test scripts are generated representing the test cases which are shown in Table 14 and 15.

Preparing testing environment for location-2: The ES application is a set of functions and the functions must be called in a sequence at any of the distributed locations for undertaking testing of a particular test case. For each of the testing to be carried, the sequence in which the functions must be called is mapped and stored in a repository as shown in Table 16.

The functions to be called and the sequence in which the functions must be called for facilitating the testing when test script is to be executed using the in-circuit Emulator have been mapped to each of the test case and the mapping is fed to the in-circuit emulator for running the code as required for testing a test case. The mapping of the test script to function code is shown in Table 17.

Undertaking the testing through instruction set simulator at location-2: The emulator code resident on the host will makes available the script to the emulator resident on the target. The emulator on the target reads the script, parses the command and its related arguments. Using the commands, the emulator fetches the list of functions and the sequence in which they must be executed. Pointers to the functions are added into a queue. Another process contained within the emulator reads the queue and executes the functions. The functions are called by passing arguments which are associated with the commands. The execution of function leads to completing the testing and generating test results. The test results obtained after testing using the above process which are written to audit trail are shown in Table 18.

Testing at location-3

Identifying the test cases for location-3: The test cases that must be used for testing at a location-3 and the related script code generated is shown in Table 19 for each of the test case that must be tested at location-3.

Table 12: Test cases for testing at location-2 through in circuit emulator

Split master test case	Test case	Script code
17B	Measurer throughput	THRUTEMP2, THRUPUT = 10, Location = L2

Table 13: List of functions that are used to compute throughput and response of temp 2

Function code	Name of the function	Function description
F02	CountAndAcuumulate()	Counts the number of times temp-2 is read and accumulates the time taken to read the temp-2

Table 14: Mapping the scripts with generated functions

Test serial	Test case	Script	Function calling sequence
17B	Measurer throughput of temp 2	THRUTEMP2, THRUPUT = 10, Location = L2	CountAndAcuumulate()

Table 15: Test results obtained through in-circuit executed at location-2 processor

Split test case	Test case	Script command	Input variable-1	Input variable 1 value	Input variable-2	Input variable-2 value	Input variable-3	Input variable-3 value	Test output variable	Test output	Test fail/ pass
17B	Test for throughput of temp-2	THRUTEMP1	RESPONSE	10	-	-	-	-	TOUTTEMP2	230	Pass

Table 16: Test cases and script code for testing at location-3 through in-circuit emulation

Master test case	Test case	Script code
10 C	Test for response time of the pump-1 on	RESPPUMP1ON, RESPONSE = 10, Location = L3
11 C	Test for response time of the pump-1 off	RESPPUMP1OFF, RESPONSE = 10, Location = L3

Table 17: List of functions that are used for computing response from pump-1

Function code	Name of the function	Function description
F03	CompResponse()	Computes the response time of pump-1

Table 18: Mapping the scripts with generated functions for testing at location-3

Test serial	Test case	Script	Function calling sequence
1	10 C	RESPPUMP1ON, RESPONSE = 10, Location = L1	CompResponse()
2	11 C	RESPPUMP1OFF, RESPONSE = 10, Location = L1	CompResponse()

Preparing testing environment for location-3: The ES application is a set of functions and the functions must be called in a sequence at any of the distributed locations for undertaking testing of a particular test case. For each of the testing to be carried, the sequence in which the functions must be called is mapped and stored in a repository as shown in Table 21.

The functions to be called and the sequence in which the functions must be called for facilitating the testing when a test script is to be executed using the in-circuit emulator have been mapped. The MAP is used by in-circuit emulator for running the code as required for testing a test case. The mapping of the test script to function code is shown in Table 22.

Merging test results and producing an audit trail: The test results obtained from each of the testing undertaken through in-circuit emulation are merged based on the test cases 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 instruction set simulation method at each of the distributed locations are shown in Fig. 7.

The test result that could be obtained related to the entire distributed system level is achieved through merging the test results obtained at each of the test location. The test results for the entire distributed embedded system are shown in Table 23-28.

Table 19: Test results obtained through instruction set simulation executed at location-3

Split test case	Test case	Script command	Input variable-1	Input variable 1 value	Input variable-2	Input variable-2 value	Input variable-3	Input variable-3 value	Test output variable	Test output	Test fail/ pass
10C	Test response for Pump-1 on	RESPPUMP1	RESPONSE	10	-	-	-	-	TOUTPUMP1	230	Pass
11C	Test response for Pump-1 off	RESPPUMP1	RESPONSE	20	-	-	-	-	TOUTPUMP2	460	Pass

Table 20: Test cases and script code for testing at location-4 through instruction set simulation

Master test case	Test case	Script code
19C	Test for response time of the Pump-2 on	RESPPUMP1ON, RESPONSE = 10, Location = L3
20C	Test for response time of the Pump-2 off	RESPPUMP1OFF, RESPONSE = 10, Location = L3

Table 21: List of functions that are used for computing response from pump-2

Function code	Name of the function	Function description
F03	CompResponse()	Testing for response of pump-2

Table 22: Mapping test scripts to application functions for testing at location-4

Test serial	Test case	Script	Function calling sequence
1	19 C	RESPPUMP2ON, RESPONSE = 10, Location = L1	CompResponse()
2	20 C	RESPPUMP2OFF, RESPONSE = 10, Location = L1	CompResponse()

Table 23: Test results obtained through instruction set simulation executed at location-4

Split test case	Test case	Script command	Input variable-1	Input variable 1 value	Input variable-2	Input variable-2 value	Input variable-3	Input variable-3 value	Test output variable	Test output	Test fail/ pass
19 C	Test response for pump-2 on	RESPPUMP2	RESPONSE	10	-	-	-	-	TOUTPUMP2	230	Pass
20 C	Test response for pump-2 off	RESPPUMP2	RESPONSE	20	-	-	-	-	TOUTPUMP2	460	Pass

Table 24: Test cases and script code for testing at location-5 through instruction set simulation

Master test case	Test case	Script code
12C	Test for response time of the buzzer on	RESPBUZZERON, RESPONSE = 10, Location = L5
13C	Test for response time of the buzzer off	RESPBUZZEROFF, RESPONSE = 10, Location = L5

Table 25: List of functions that are used for computing response from buzzer

Function code	Name of the function	Function description
F03	CompResponse()	Testing for response of pump-2

Table 26: Mapping test scripts to Application functions for testing at location-4

Test serial	Test case	Script	Function calling sequence
1	12C	RESPBUZZERON, RESPONSE = 10, Location = L5	CompResponse()
2	13C	RESPBUZZEROFF, RESPONSE = 10, Location = L5	CompResponse()

Table 27: Test results obtained through in-circuit emulator executed at location-5

Split test case	Test case	Script command	Input variable-1	Input variable 1 value	Input variable-2	Input variable-2 value	Input variable-3	Input variable-3 value	Test output variable	Test output	Test fail/ pass
12C	Test response for buzzer on	RESPBUZZER	RESPONSE	10	-	-	-	-	TOUTBUZZER	230	Pass
13C	Test response for buzzer off	RESPBUZZER	RESPONSE	20	-	-	-	-	TOUTBUZZER	460	Pass

Table 28: Distributed system level experimental results

Split test case	Test case	Script command	Input-1	Input variable-1 value	Input-2	Input-2 value	Output variables	Test output	Test fail/pass
4	Read Temp-1 and measure throughput	THRUTEMP1	TEMP-1	180	THRU	10	THRUTEMP1	230	Pass
10	Test response between the reading the temp-1 and starting the pump-1	RESPPUMP1	TEMP-1	55	RESP	10	RESPTEMP1	230	Pass
11	Test response between the reading the temp-1 and stopping the pump-1	RESPPUMP2	TEMP-1	78	RESP	20	RESPTEMP1	460	Pass
12	Test response between the reading the temp-1 and starting the buzzer	RESPBUZZER	TEMP-1	80	RESP	30	RESPTEMP1	690	Pass
13	Test response between the reading the temp-1 and stopping the buzzer	RESPBUZZER	TEMP-1	50	RESP	40	RESPBUZZER	920	Pass
17	Read temp-2 and measure throughput	THRUPUMP2	TEMP-2	60	RESP	10	THRUTEMP2	230	Pass
23	Test response between the reading the temp-2 and starting the pump-2	RESPPUMP2	TEMP-2	45	RESP	10	RESPPUMP2	230	Pass
24	Test response between the reading the temp-2 and stopping the pump-2	RESPUMP2	TEMP-2	56	RESP	20	RESPPUMP2	460	Pass
25	Test response between the reading the Temp-2 and starting the buzzer	RESPBUZZER	TEMP-2	43	RESP	30	RESPBUZZER	690	Pass
26	Test response between the reading the Temp-2 and stopping the buzzer	RESPBUZZER	TEMP-2	77	RESP	40	RESPBUZZER	920	Pass

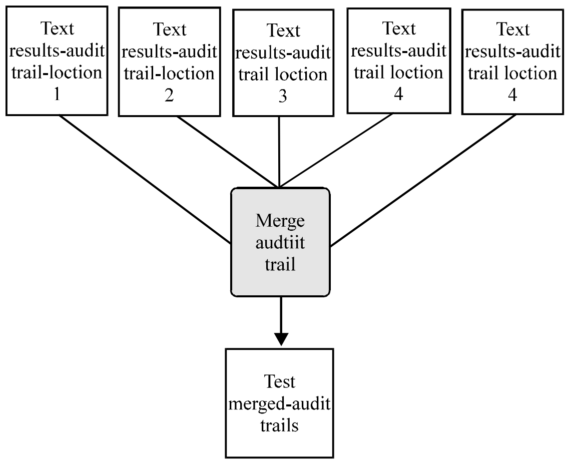
Text results-audit trail-loction 1
Text results-audit trail-loction 2
Text results-audit trail loction 3
Text results-audit trail loction 4
Text results-audit trail loction 4
Merge audtiit trail
Test merged-audit trails

Fig. 7: Merging the test results obtained through use of Instruction set simulator at each of the location

CONCLUSION

The standard methods such as instruction set simulators, scaffolding, assert macros, in-circuit emulators, etc. which are in existence as on today can be used for testing standalone embedded systems. However, the same methods can be extended and used for testing the distributed embedded systems. Testing through in-circuit emulation provides a basis of testing considering both hardware and software together. The test cases that are to be used for testing entire embedded systems can be broken into test cases that can be tested at different locations and merge the results obtained out of testing at individual locations to get the overall picture of testing entire distributed embedded system. The testing through instruction in-circuit emulators can be achieved through breaking and merging process which is presented in this study.

REFERENCES

Armengaud, E., A. Steininger and M. Horauer, 2005. Efficient stimulus generation for testing embedded distributed systems the FlexRay example. Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA'05) Vol. 1, September 19-22, 2005, IEEE, Catania, Italy, pp: 761-770.

Chimisliu, V. and F. Wotawa, 2012. Model based test case generation for distributed embedded systems. Proceedings of the 2012 IEEE International Conference on Industrial Technology (ICIT), March 19-21, 2012, IEEE, Graz, Austria, ISBN: 978-1-4673-0340-8, pp: 656-661.

Ergincan, F. and A. Saatci, 1986. A stand-alone in-circuit emulator. Microprocessing Microprogramming, 17: 159-167.

Hill, J., P. Varshneya and D. Schmidt, 2011. Evaluating distributed real-time and embedded system test correctness using system execution traces. Open Comput. Sci., 1: 167-184.

Jacobson, I., G. Booch and J. Rumbaugh, 1999. The Unified Software Development Process. Addison-Wesley, Boston, Massachusetts, USA., ISBN:9780321822000, Pages: 512.

Lee, N.H. and S.D. Cha, 2003. Generating test sequences from a set of MSCs. Comput. Netw., 42: 405-417.

Liu, H., M. Jin and C. Liu, 2010. Construction of the simulating environment for testing distributed embedded software. Proceedings of the 5th International Conference on Computer Science and Education (ICCSE'10), August 24-27, 2010, IEEE, Hefei, China, ISBN:978-1-4244-6002-1, pp: 97-101.

Nakamoto, Y., T. Ito, K. Yabuuchi and T. Osaki, 2014. Wide-area distributed integrated test environments for distributed embedded control system. Proceedings of the 2014 IEEE 17th International Symposium on Object-Component-Service-Oriented Real-Time Distributed Computing (ISORC), June 10-12, 2014, IEEE, Reno, Nevada, ISBN: 978-1-4799-4430-9, pp: 174-179.

Saini, D.K., 2012. Software testing for embedded systems. Intl. J. Comput. Appl., 43: 1-6.

Schutz, W., 1994. Fundamental issues in testing distributed real-time systems. Real Time Syst.,7: 129-157.

Thane, H. and H. Hansson, 1999. Towards systematic testing of distributed real-time systems. Proceedings of the 20th IEEE Symposium on Real-Time Systems, December 1-3, 1999, IEEE, Phoenix, Arizona USA., pp: 360-369.

Tian, P., J. Wang, H. Leng and K. Qiang, 2009. Construction of distributed embedded software testing environment. Proceedings of the International Conference on Intelligent Human-Machine Systems and Cybernetic (IHMSC'09) Vol. 1, August 26-27, 2009, IEEE, Hangzhou, Zhejiang, China, ISBN:978-0-7695-3752-8, pp: 470-473.

Tsai, W.T., L. Yu, A. Saimi and R. Paul, 2003. Scenario-based object-oriented test frameworks for testing distributed systems. Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03), May 30, 2003, IEEE, San Juan, Puerto Rico, USA., pp: 288-294.

Tsai, W.T., X. Bai, R. Paul and L. Yu, 2001. Scenario-based functional regression testing. Proceedings of the 25th Annual International Conference on Computer Software and Applications (COMPSAC'01), October 8-12, 2001, IEEE, Chicago, Illinois, USA., pp: 496-501.